Weblate Documentation

Release 2.16

Michal Čihař

Aug 13, 2017

Contents

1 About Weblate **3**
- 1.1 Project goals . . . 3
- 1.2 Project name . . . 3
- 1.3 Project website . . . 3
- 1.4 Authors . . . 3

2 Translators guide **5**
- 2.1 Weblate basics . . . 5
- 2.2 Registration and user profile . . . 5
- 2.3 Translating using Weblate . . . 11
- 2.4 Downloading and uploading translations . . . 17
- 2.5 Checks and fixups . . . 19

3 Application developer guide **25**
- 3.1 Starting with internationalization . . . 25
- 3.2 Managing translations . . . 29
- 3.3 Reviewing source strings . . . 29
- 3.4 Promoting the translation . . . 33
- 3.5 Translation progress reporting . . . 34

4 Administrators guide **37**
- 4.1 Quick setup guide . . . 37
- 4.2 Installation instructions . . . 38
- 4.3 Weblate deployments . . . 59
- 4.4 Upgrading Weblate . . . 68
- 4.5 Authentication . . . 77
- 4.6 Access control . . . 81
- 4.7 Translation projects . . . 86
- 4.8 Language definitions . . . 97
- 4.9 Continuous translation . . . 98
- 4.10 Translation process . . . 105
- 4.11 Checks and fixups . . . 112
- 4.12 Machine translation . . . 117
- 4.13 Configuration . . . 120
- 4.14 Sample configuration . . . 133
- 4.15 Management commands . . . 145
- 4.16 Whiteboard messages . . . 154

4.17 Advertisement . . . 155
4.18 Component Lists . . . 156
4.19 Optional Weblate modules . . . 156

5 Frequently Asked Questions **159**
5.1 Configuration . . . 159
5.2 Usage . . . 162
5.3 Troubleshooting . . . 164
5.4 Features . . . 165

6 Supported formats **167**
6.1 Automatic detection . . . 167
6.2 GNU Gettext . . . 167
6.3 XLIFF . . . 169
6.4 Java properties . . . 169
6.5 Joomla translations . . . 169
6.6 Qt Linguist .ts . . . 170
6.7 Android string resources . . . 170
6.8 Apple OS X strings . . . 170
6.9 PHP strings . . . 171
6.10 JSON and nested structure JSON files . . . 171
6.11 WebExtension JSON . . . 172
6.12 .Net Resource files . . . 172
6.13 CSV files . . . 172
6.14 YAML files . . . 173
6.15 Others . . . 173

7 Version control integration **175**
7.1 Accessing repositories . . . 175
7.2 Git . . . 177
7.3 GitHub . . . 178
7.4 Mercurial . . . 179
7.5 Subversion . . . 179

8 Weblate's Web API **181**
8.1 REST API . . . 181
8.2 Notification hooks . . . 201
8.3 Exports . . . 202
8.4 RSS feeds . . . 203

9 Weblate Client **205**
9.1 Installation . . . 205
9.2 Synopsis . . . 205
9.3 Description . . . 205
9.4 Files . . . 207
9.5 Examples . . . 208

10 Weblate's Python API **209**
10.1 Instalation . . . 209
10.2 `wlc` . . . 209
10.3 `wlc.config` . . . 210
10.4 `wlc.main` . . . 210

11 Changes **211**
11.1 weblate 2.16 . . . 211

11.2 weblate 2.15 . . . 211
11.3 weblate 2.14.1 . . . 212
11.4 weblate 2.14 . . . 212
11.5 weblate 2.13.1 . . . 213
11.6 weblate 2.13 . . . 213
11.7 weblate 2.12 . . . 213
11.8 weblate 2.11 . . . 214
11.9 weblate 2.10.1 . . . 214
11.10 weblate 2.10 . . . 214
11.11 weblate 2.9 . . . 215
11.12 weblate 2.8 . . . 215
11.13 weblate 2.7 . . . 216
11.14 weblate 2.6 . . . 216
11.15 weblate 2.5 . . . 216
11.16 weblate 2.4 . . . 218
11.17 weblate 2.3 . . . 219
11.18 weblate 2.2 . . . 219
11.19 weblate 2.1 . . . 220
11.20 weblate 2.0 . . . 220
11.21 weblate 1.9 . . . 221
11.22 weblate 1.8 . . . 221
11.23 weblate 1.7 . . . 221
11.24 weblate 1.6 . . . 222
11.25 weblate 1.5 . . . 222
11.26 weblate 1.4 . . . 223
11.27 weblate 1.3 . . . 224
11.28 weblate 1.2 . . . 224
11.29 weblate 1.1 . . . 225
11.30 weblate 1.0 . . . 225
11.31 weblate 0.9 . . . 225
11.32 weblate 0.8 . . . 226
11.33 weblate 0.7 . . . 226
11.34 weblate 0.6 . . . 227
11.35 weblate 0.5 . . . 227
11.36 weblate 0.4 . . . 227
11.37 weblate 0.3 . . . 227
11.38 weblate 0.2 . . . 228
11.39 weblate 0.1 . . . 228

12 Contributing **229**
12.1 Code and development . . . 229
12.2 Coding standard . . . 229
12.3 Developer's Certificate of Origin . . . 229
12.4 Testsuite . . . 230
12.5 Reporting issues . . . 230
12.6 Security issues . . . 230
12.7 Starting with our codebase . . . 231
12.8 Earning money by coding . . . 231
12.9 Translating . . . 231
12.10 Funding Weblate development . . . 231

13 Internals **233**
13.1 Modules . . . 233

14 License 235

15 Indices and tables 237

HTTP Routing Table 239

Python Module Index 241

Contents:

CHAPTER 1

About Weblate

Project goals

Web based translation with tight git integration supporting wide range of file formats and makes it easy for translators to contribute.

The translations should be kept within same repository as source code and translation process should closely follow development.

There is no plan in heavy conflict resolution as these should be primarily handled on git side.

Project name

The project is named as mixture of words web and translate.

Project website

You can find project website at <https://weblate.org/>, there is also demonstration server at <https://demo.weblate.org/>. This documentation can be browsed on <https://docs.weblate.org/>.

Authors

This tool was written by Michal Čihař <michal@cihar.com>.

CHAPTER 2

Translators guide

Weblate basics

Projects structure

Each project can contain various components. The reason for this structure is that all components in a project are expected to have a lot in common. Whenever translation is made in single component, it is automatically propagated to others within same project (this is especially useful when translating more version of same project, but can be disabled, see *Component configuration*).

Registration and user profile

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 following two simple steps:

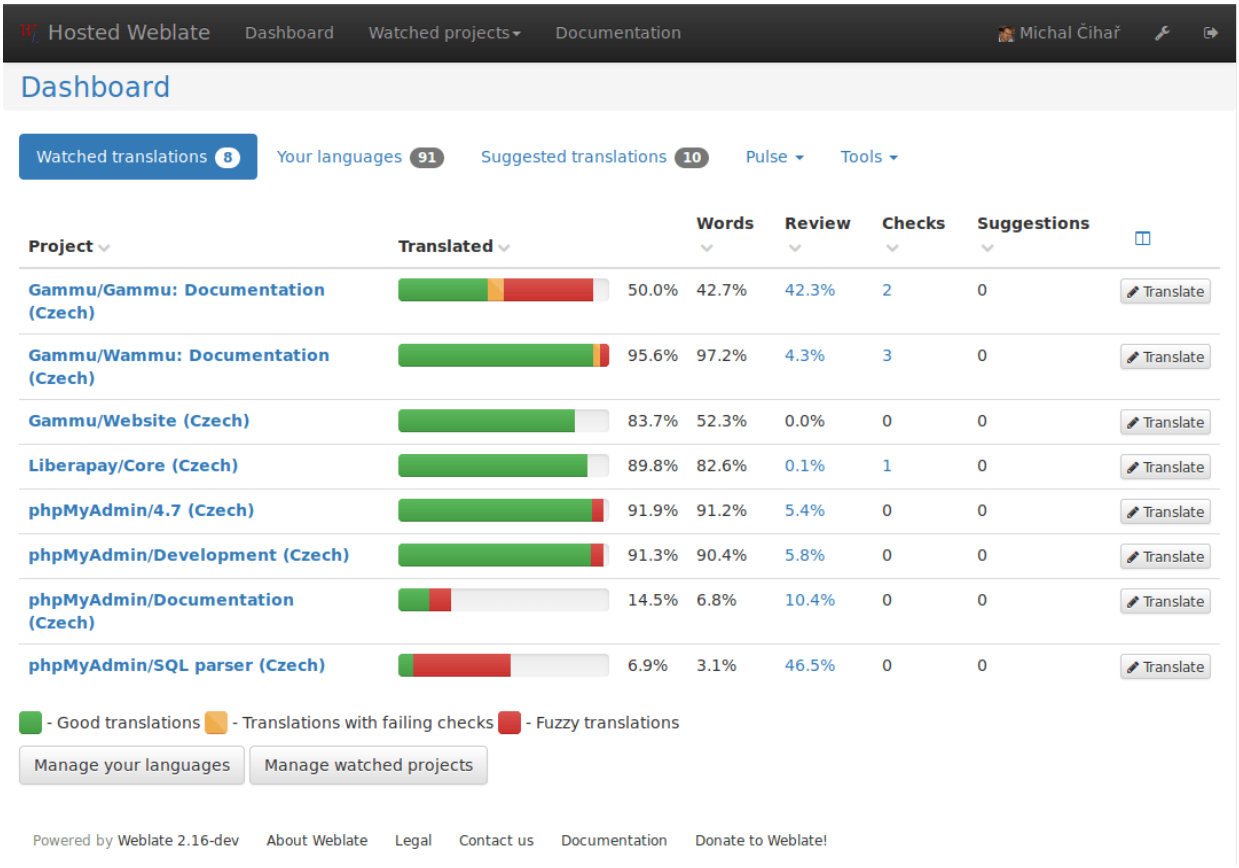
1. Fill out the registration form with your credentials
2. Activate registration by following in email you receive
3. Possibly adjust your profile to choose which languages you know

Dashboard

When you log in to Weblate, you will see an overview of projects and components and 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 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.

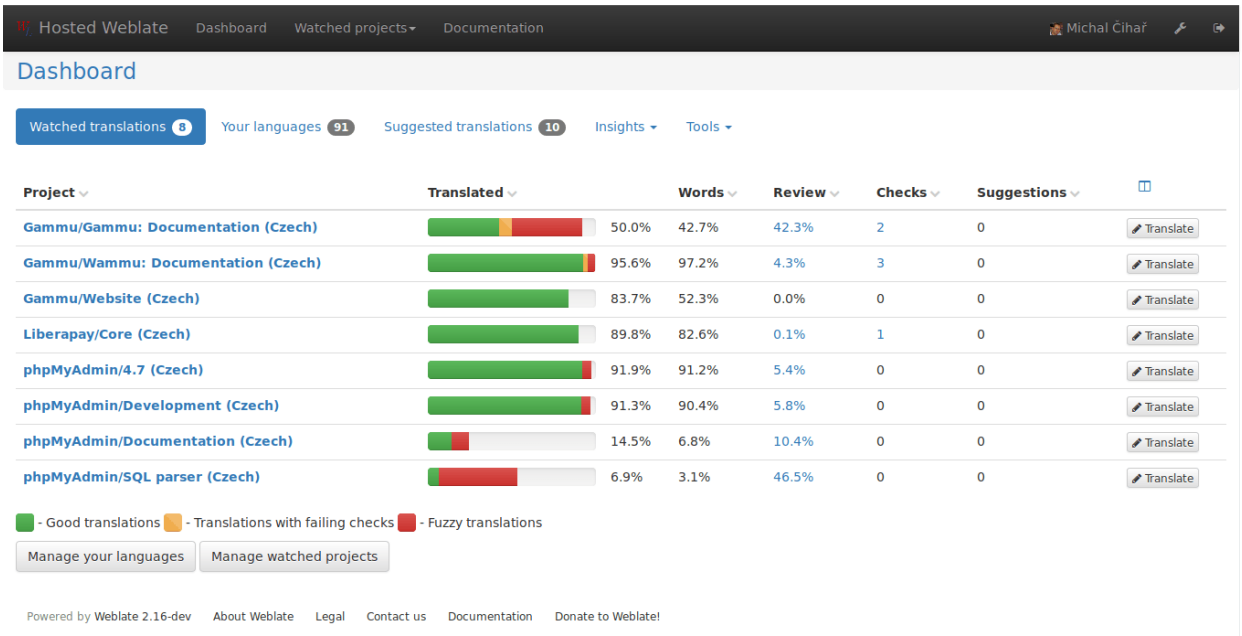
User profile

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.

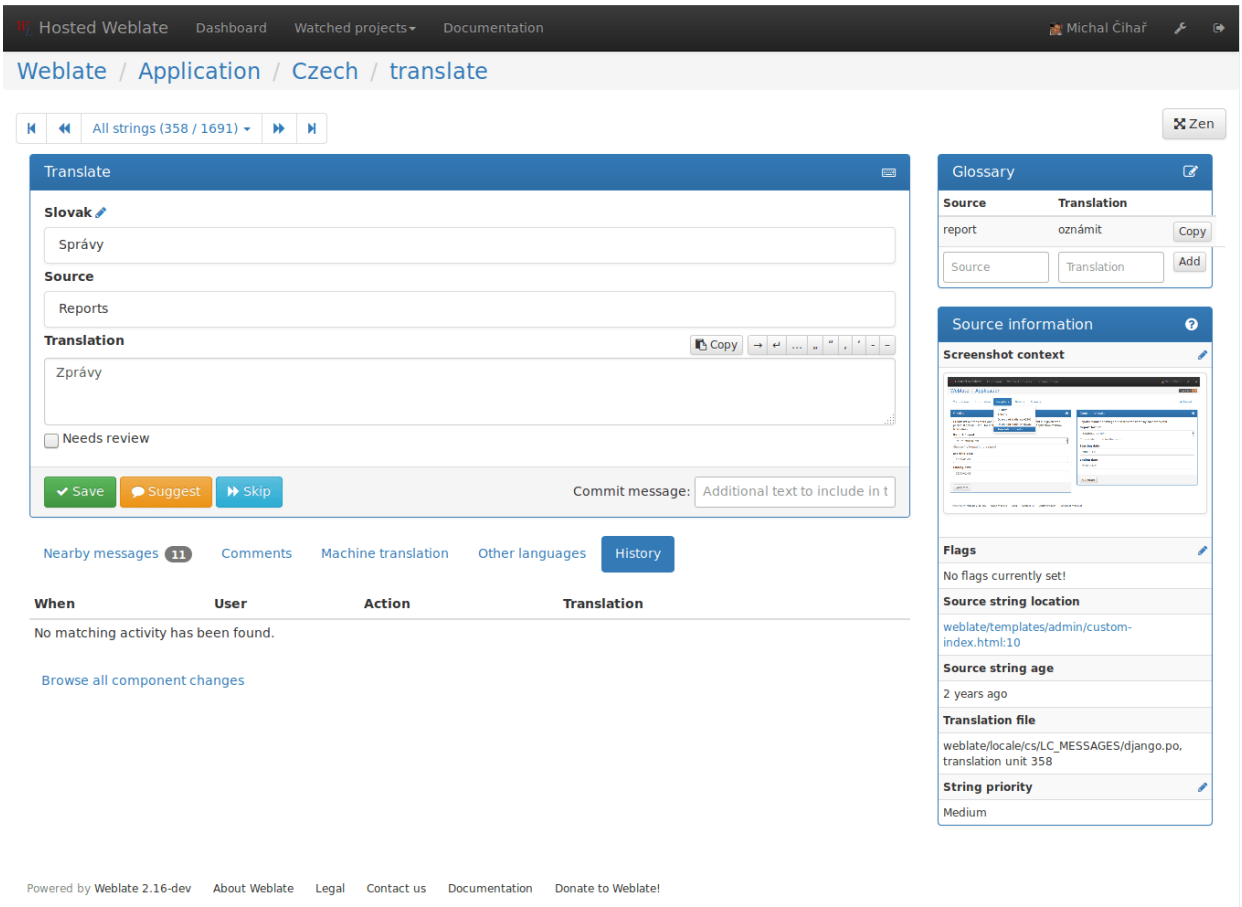
Translated languages

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.



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:



Default dashboard view

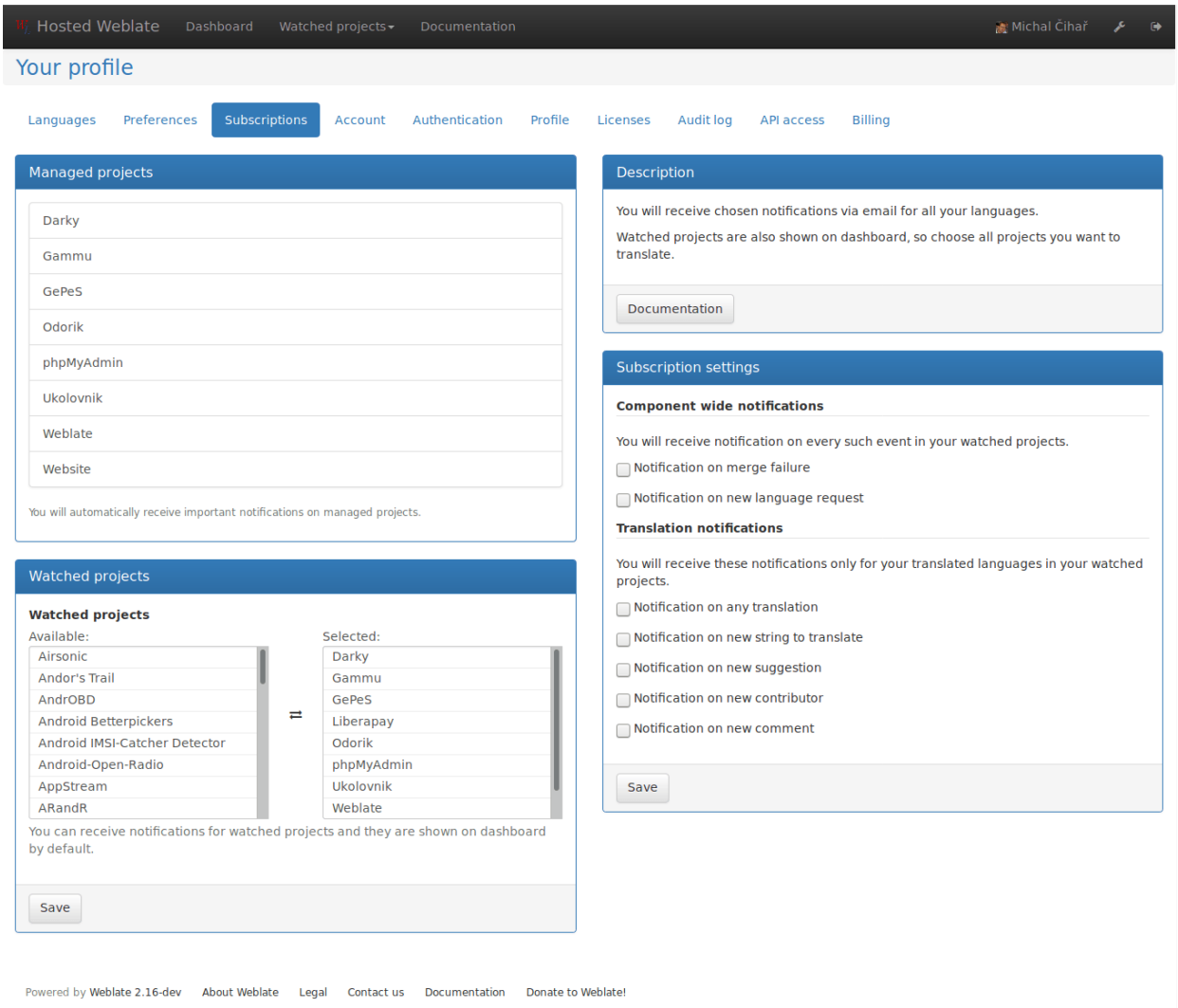
On the *Preferences* tab, you can pick which of the available dashboard views will be displayed by default. If you pick *Component list*, you have to select which component list will be displayed from the *Default component list* drop-down.

Subscriptions

You can subscribe to various notifications on *Subscriptions* tab. You will receive notifications for selected events on chosen projects for languages you have indicated for translation (see above).

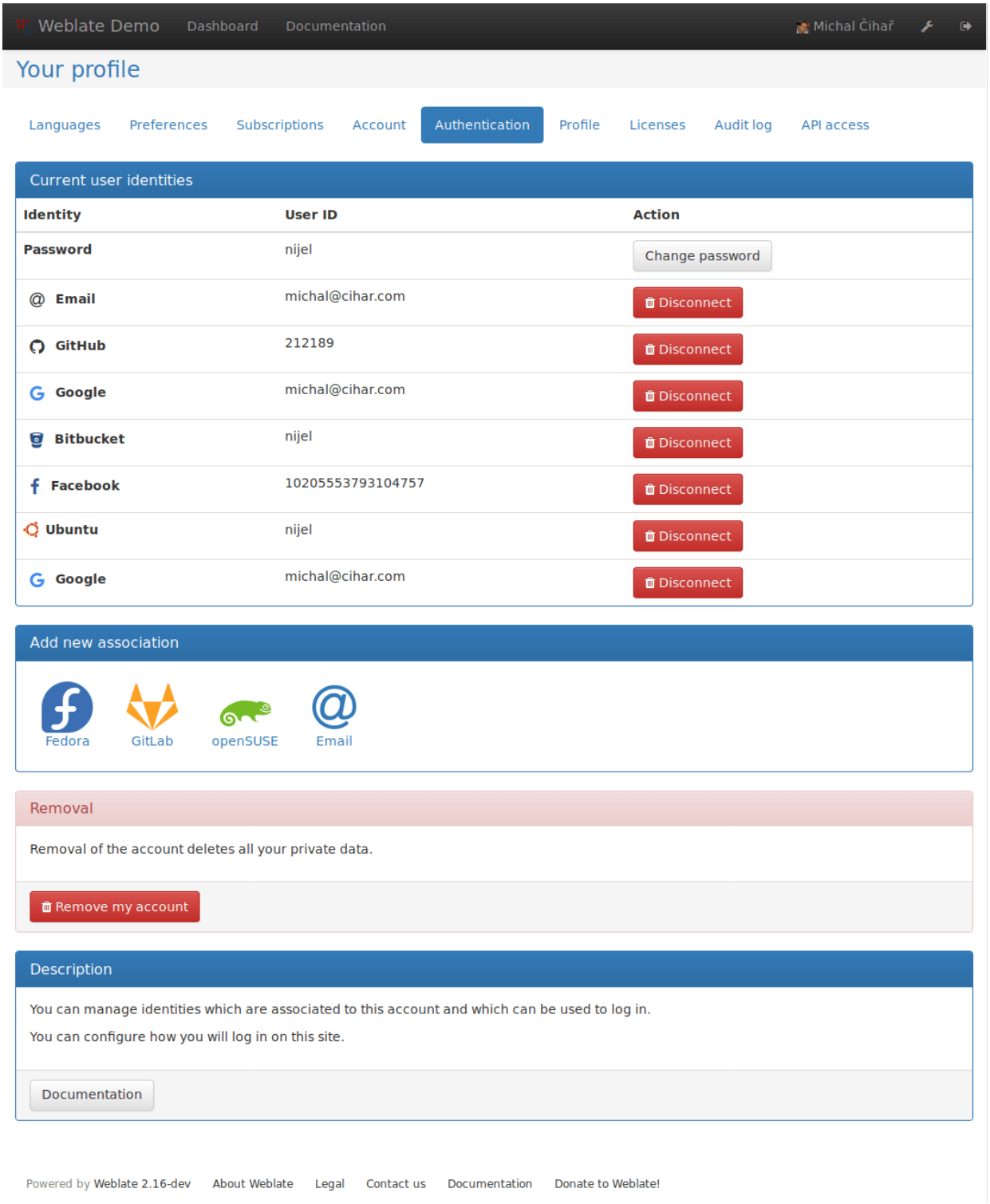
If you are an owner of some project, you will always receive some important notifications, like merge failures or new language requests.

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



Authentication

On the *Authentication* tab you can connect various services which you can use to login into Weblate. List of services depends on Weblate configuration, but can include popular sites such as Google, Facebook, GitHub or Bitbucket.



Avatar

Weblate can be configured to show avatar for each user (depending on *ENABLE_AVATARS*). These images are obtained using libravatar protocol (see https://www.libravatar.org/) or using https://gravatar.com/.

Translating using Weblate

Thank you for interest in translating using Weblate. Weblate can be used to translate many projects and every project can have different settings which influence whether you can translate or add suggestions only.

Overall there are following possibilities for translating:

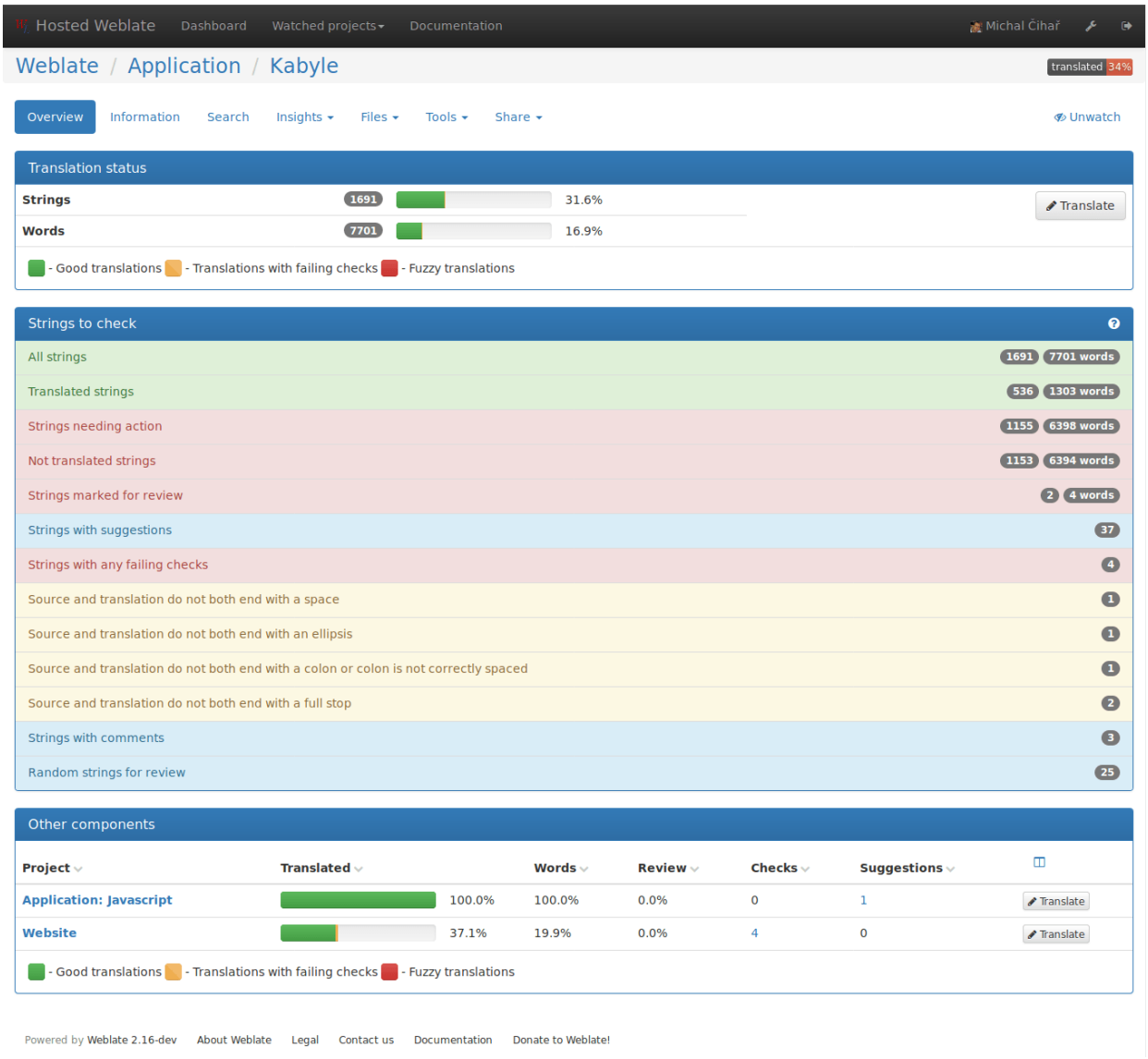
- Projects accepts direct translations
- Projects accepts only suggestions and those are accepted once they get defined number of votes

There are also some options for translation project visibility:

- It can be publicly visible and anybody can contribute
- It can be visible only to certain group of translators

Translation links

Once you navigate to a translation, you will be shown set of links which lead to translation. These are results of various checks, like untranslated or strings needing review. Should no other checks fire, there will be still link to all translations. Alternatively you can use search field to find translation you need to fix.



Suggestions

As an anonymous user, you have no other choice than making a suggestion. However if you are logged in you can still decide to make only a suggestion instead of saving translation, for example in case you are unsure about the translation and you want somebody else to review it.

Note: Permissions might vary depending on your setup, what is described is default Weblate behaviour.

Translating

On translate page, you are shown source string and edit area for translating. Should the translation be plural, multiple source strings and edit areas are shown, each described with label for plural form.

Any special whitespace chars are underlined in red and indicated with grey symbols. Also more than one space is underlined in red to allow translator to keep formatting.

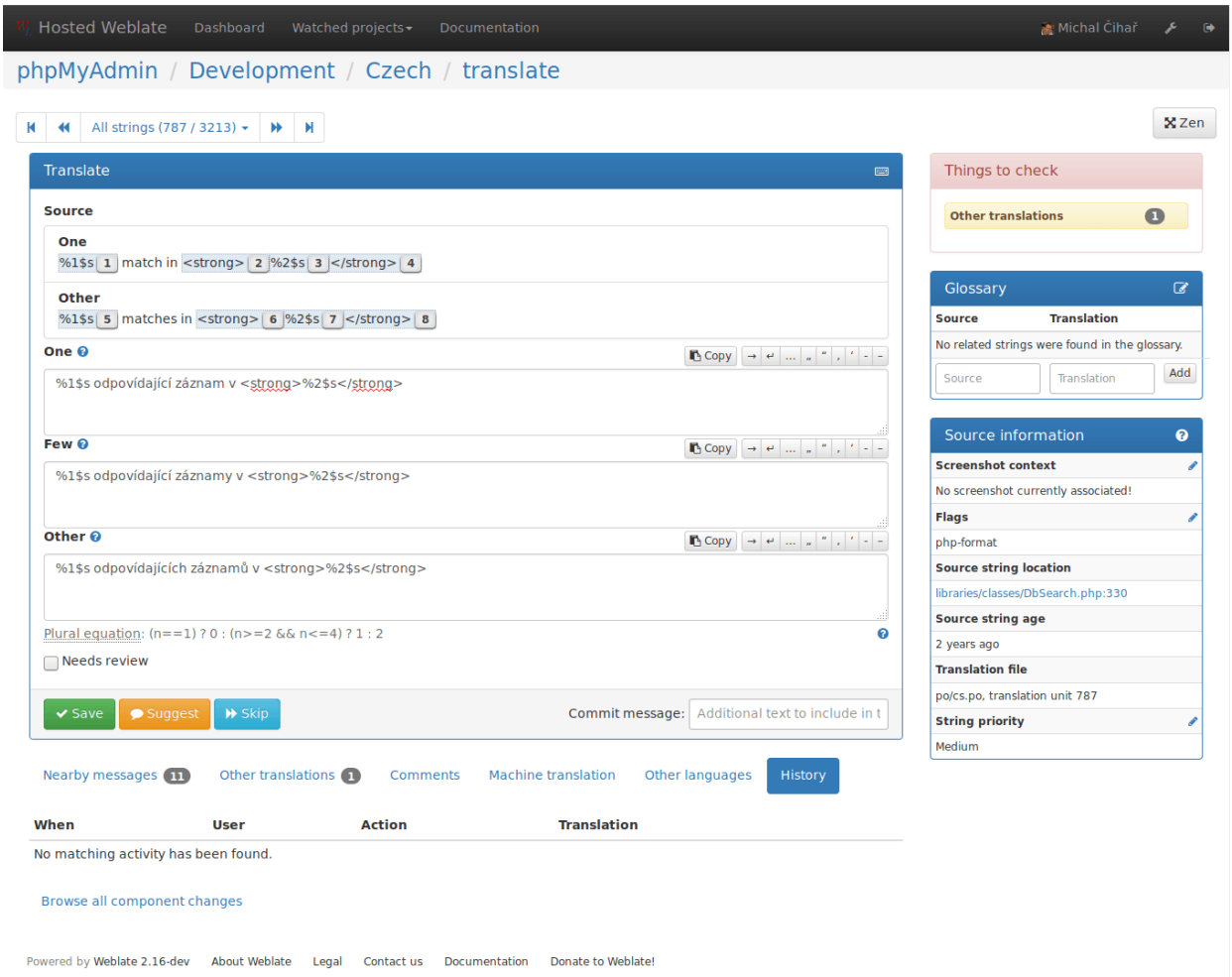
There are various extra information which can be shown on this page. Most of them are coming from the project source code (like context, comments or where the message is being used). When you configure secondary languages in your preferences, translation to these languages will be shown (see *Secondary languages*).

Bellow translation can be also shown suggestions from other users, which you can accept or delete.

Plurals

What are plurals? Generally spoken plurals are words which take into account numeric meanings. But as you may imagine each language has its own definition of plurals. English, for example, supports one plural. We have a singular definition, for example "car", which means implicit one car, and we have the plural definition, "cars" which could mean more than one car but also zero cars. Other languages like Czech or Arabic have more plurals and also the rules for plurals are different.

Weblate does have support for translating these and offers you one field to translate every plural separately. The number of fields and how it is used in the translated application depends on plural equation which is different for every language. Weblate shows the basic information, but you can find more detailed description in the Language Plural Rules from the Unicode Consortium.



Keyboard shortcuts

While translating you can use following keyboard shortcuts:

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.

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

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

Alt+E Focus translation editor.

Alt+C Focus comment editor.

Alt+M Shows machine translation tab.

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

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

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

Alt+N Shows nearby strings tab.

Alt+S Shows search tab.

Alt+V Copies source string

Alt+F Toggles fuzzy flag.

Translation context

Translation context part allows you to see related information about current string.

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

Screenshots Screenshots can be uploaded to Weblate to better show translators where the string is used, see *Visual context for strings*.

Nearby messages Displays messages which are located nearby in translation file. These usually are also used in similar context and you might want to check them to keep translation consistent.

Similar messages Messages which are similar to currently one, which again can help you to stay consistent within translation.

All locations In case message appears in multiple places (eg. multiple components), this tab shows all of them and for inconsistent translations (see *Inconsistent*) you can choose which one to use.

Glossary Displays words from project glossary which are used in current message.

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

Project Project information like instructions for translators or information about VCS repository.

If translation format supports it, you can also follow links to source code which contains translated strings.

Translation history

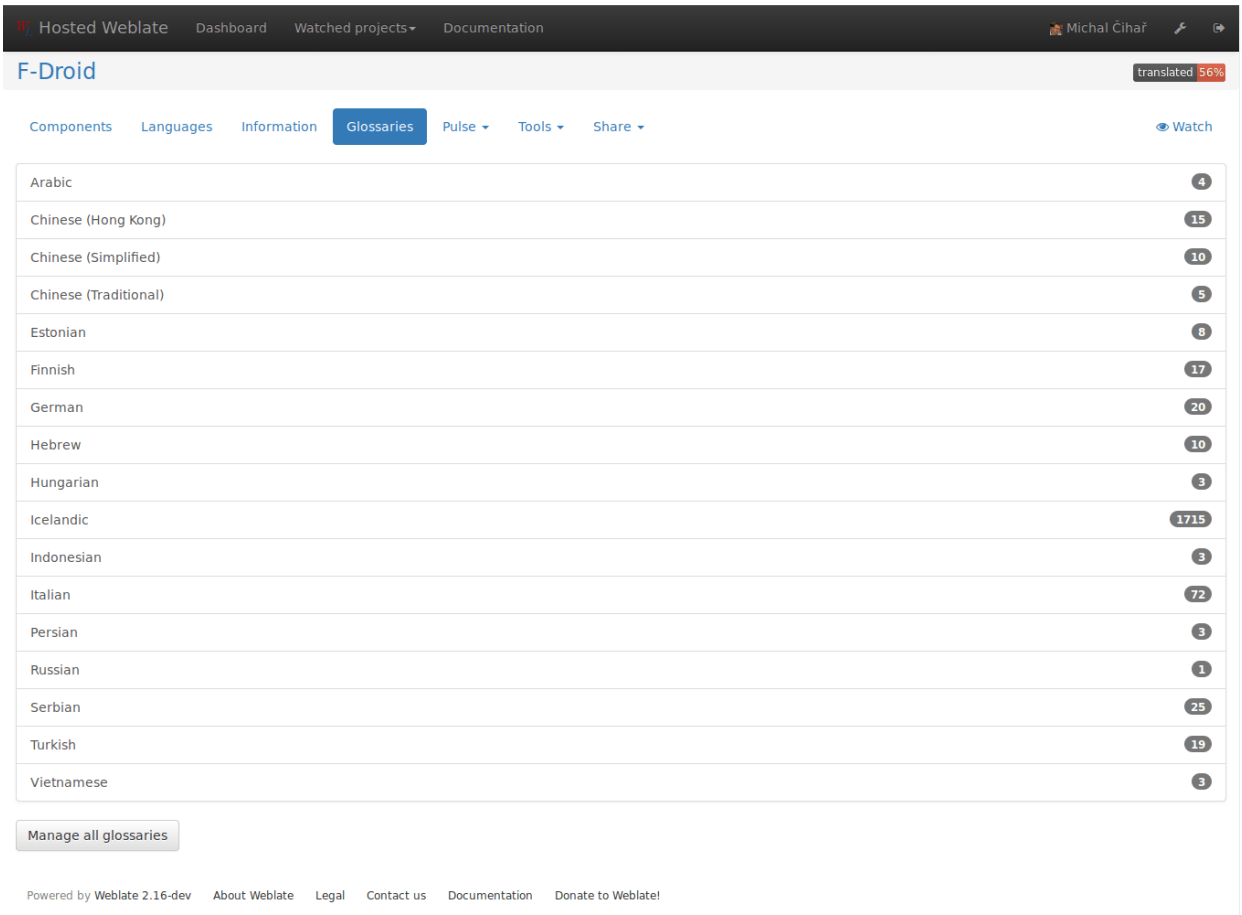
Every change is by default (unless disabled in component settings) saved in the database and can be reverted. Of course you can still also revert anything in underlaying version control system.

Glossary

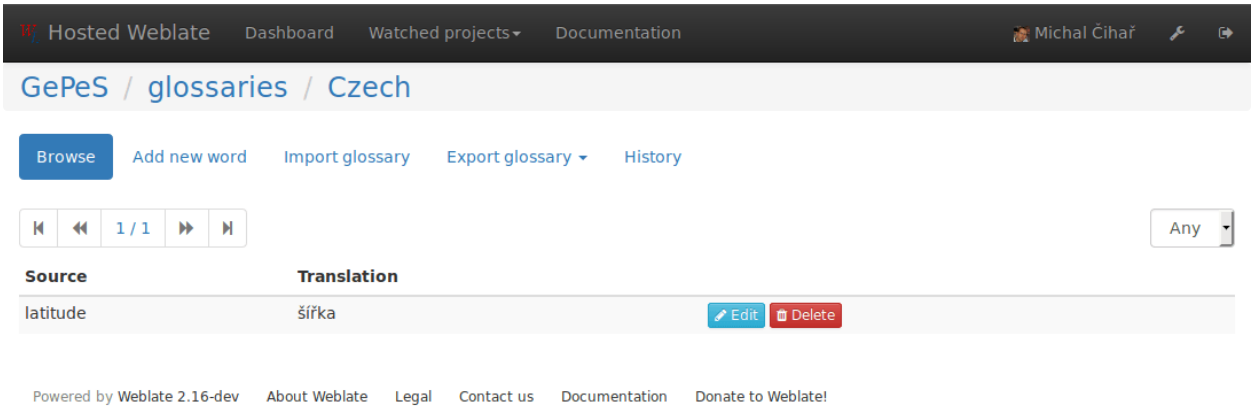
Each project can have assigned glossary for any language. This could be used for storing terminology for given project, so that translations are consistent. You can display terms from currently translated string in bottom tabs.

Managing glossaries

On project page, on *Glossaries* tab, you can find link *Manage all glossaries*, where you can start new glossaries or edit existing ones. Once glossary is existing, it will also show up on this tab.



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



Machine translation

Based on configuration and your language, Weblate provides buttons for following machine translation tools.

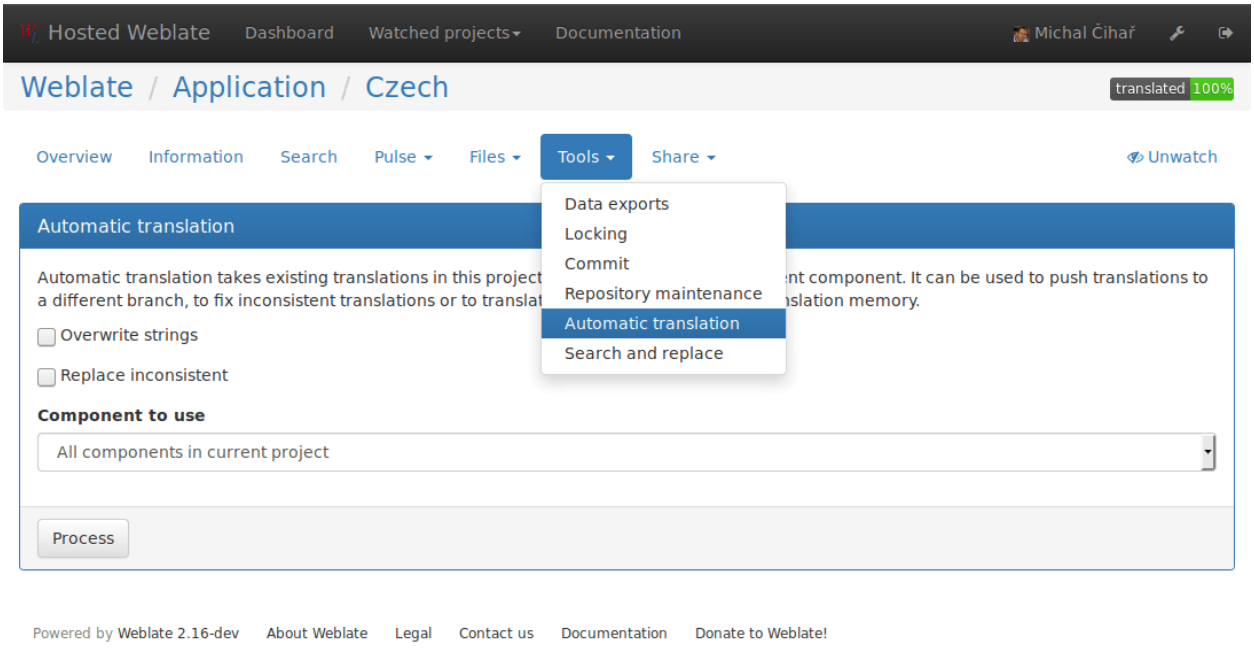
All machine translations are available on single tab on translation page.

See also:

Machine translation

Automatic translation

Weblate can be used for merging or copying translations from one component to another. This tool is called *Automatic translation* and is accessible in the *Tools* menu:



You can choose which components you want to use as source and how to handle conflicts.

This feature can be useful in several situations like consolidating translation between different components (eg. website and application) or when bootstrapping translation for new component using existing translations (translation memory).

Downloading and uploading translations

Weblate supports both export and import of translation files. This allows you to work offline and then merge changes back. Your changes will be merged within existing translation (even if it has been changed meanwhile).

Note: This available options might be limited by *Access control*.

Downloading translations

You can download translatable file using the *Download source file* action in the *Files* menu. This will give you the file as is stored in upstream version control system.

For some formats you can also download compiled file to use withing application (for example `.mo` files for GNU Gettext) using the *Download compiled translation*.

Uploading translations

You can upload translated files using the *Upload translation* action in the *Files* menu.

Weblate accepts any file format it understands on upload, but it is still recommeded to use same file format as is used for translation, otherwise some features might not be translated properly.

See also:

Supported formats

The uploaded file is merged to current translation, overwriting existing entries by default (this can be changed in the upload dialog).

Import methods

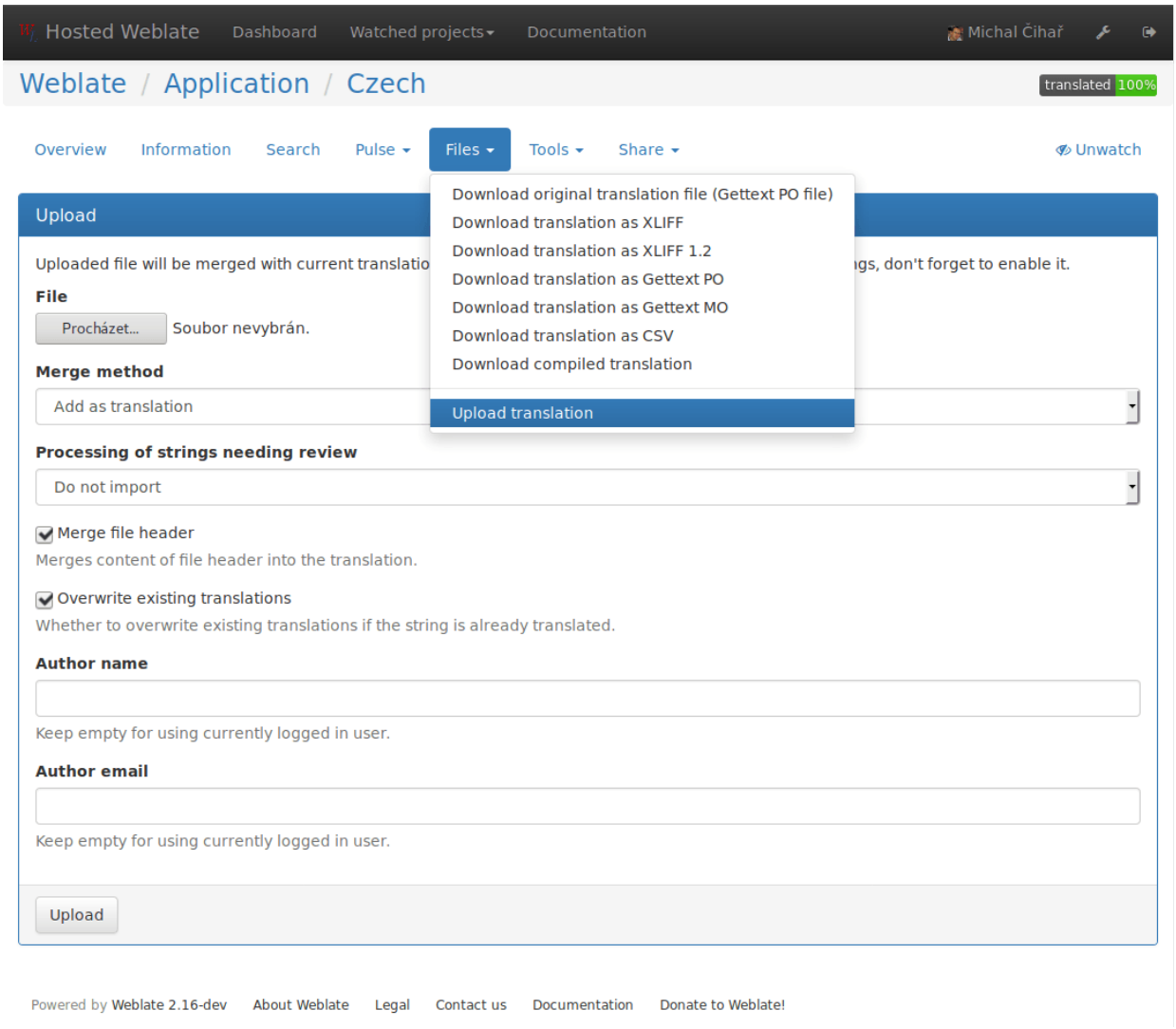
You can choose how imported strings will be merged out of following options:

Add as translation Imported translations are added as translation. This is most usual and default behavior.

Add as a suggestion Imported translations are added as suggestions, do this when you want to review imported strings.

Add as translation needing review Imported translations are added as translations needing review. This can be useful for review as well.

Additionally, when adding as a translation, you can choose whether to overwrite already translated strings or not or how to handle strings needing review in imported file.



Checks and fixups

Automatic fixups

In addition to *Quality checks*, Weblate can also automatically fix some common errors in translated strings. This can be quite powerful feature to prevent common mistakes in translations, however use it with caution as it can cause silent corruption as well.

See also:

AUTOFIX_LIST

Quality checks

Weblate does wide range of quality checks on messages. The following section describes them in more detail. The checks take account also special rules for different languages, so if you think the result is wrong, please report a bug.

See also:

CHECK_LIST, *Customizing checks*

Translation checks

These are executed on every translation change and help translators to keep good quality of translations.

Unchanged translation

The source and translated strings are the same at least in one of the plural forms. This check ignores some strings which are quite usually same in all languages and strips various markup, which can occur in the string, to reduce number of false positives.

This check can help finding strings which were mistakenly not translated .

Starting or trailing newline

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

Newlines usually appear in source string for a good reason, so omitting or adding it can lead to formatting problems when the translated text is used in the application.

Starting spaces

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

Space in beginning is usually used for indentation in the interface and thus is important to keep.

Trailing space

Source and translated do not both end with a space.

Trailing space is usually used to give space between neighbouring elements, so removing it might break application layout.

Trailing stop

Source and translated do not both end with a full stop. Full stop is also checked in various language variants (Chinese, Japanese, Devanagari or Urdu).

When the original string is a sentence, the translated one should be a sentence as well to be consistent within the translated content.

Trailing colon

Source and translated do not both end with a colon or the colon is not correctly spaced. This includes spacing rules for languages like French or Breton. Colon is also checked in various language variants (Chinese or Japanese).

Colon is part of a label and should be kept to provide consistent translation. Weblate also checks for various typographic conventions for colon, for example in some languages it should be preceded with space.

Trailing question

Source and translated do not both end with a question mark or it is not correctly spaced. This includes spacing rules for languages like French or Breton. Question mark is also checked in various language variants (Armenian, Arabic, Chinese, Korean, Japanese, Ethiopic, Vai or Coptic).

Question mark indicates question and this semantics should be kept in translated string as well. Weblate also checks for various typographic conventions for question mark, for example in some languages it should be preceded with space.

Trailing exclamation

Source and translated do not both end with an exclamation mark or it is not correctly spaced. This includes spacing rules for languages like French or Breton. Exclamation mark is also checked in various language variants (Chinese, Japanese, Korean, Armenian, Limbu, Myanmar or Nko).

Exclamation mark indicates some important statement and this semantics should be kept in translated string as well. Weblate also checks for various typographic conventions for exclamation mark, for example in some languages it should be preceded with space.

Trailing ellipsis

Source and translation do not both end with an ellipsis. This only checks for real ellipsis (`...`) not for three dots (`...`).

Ellipsis is usually rendered nicer than three dots, so it's good to keep it when the original string was using that as well.

See also:

Ellipsis on wikipedia

Trailing semicolon

Source and translation do not both end with a semicolon. This can be useful to keep formatting of entries such as desktop files.

Maximum Length

Translation is too long to accept. This only checks for the length of translation characters.

Source and translation usually do not have same amount of characters, but if translation is too long, it can be affect a rendered shape. For example, in some UI widget, it should be kept in a specific length of characters in order to show complete translation within limited space.

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

Format strings

Format string does not match source. Weblate supports following formats:

- Python format
- Python brace format
- PHP format
- C format
- Perl format
- Javascript format
- AngularJS interpolation string

Omitting format string from translation usually cause severe problems, so you should really keep the format string matching the original one.

See also:

Python string formatting, Python brace format, PHP format strings, C printf format, Perl sprintf, AngularJS: API: $interpolate

Missing plurals

Some plural forms are not translated. Check plural form definition to see for which counts each plural form is being used.

Not filling in some plural forms will lead to showing no text in the application in case this plural would be displayed.

Same plurals

Some plural forms are translated same. In most languages the plural forms have to be different, that's why this feature is actually used.

Inconsistent

More different translations of one string in a project. This can also lead to inconsistencies in displayed checks. You can find other translations of this string on *All locations* tab.

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

Has been translated

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

Mismatched \n

Number of \n in translation does not match source.

Usually escaped newlines are important for formatting program output, so this should match to source.

Mismatched BBcode

BBcode in translation does not match source.

This code is used as a simple markup to highlight important parts of a message, so it is usually a good idea to keep them.

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

Zero-width space

Translation contains extra zero-width space (<U+200B>) character.

This character is usually inserted by mistake, though it might have legitimate use. 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 invalid.

XML tags mismatch

XML tags in translation do not match source.

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

Source checks

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

Optional plural

The string is optionally used as plural, but not using plural forms. In case your translation system supports this, you should use 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

The string uses three dots (`...`) instead of an ellipsis character (`…`).

Using Unicode character is in most cases the better approach and looks better when rendered.

See also:

Ellipsis on wikipedia

Multiple failing checks

More translations of this string have some failed quality checks. This is usually an indication that something could be done about improving the source string.

This check 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 translations, while it would be better to fix it in a source string.

CHAPTER 3

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.

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 usual 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 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 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 translator would get string `Sun` to translate. Without context most people would translate that as our closest star, but you might be actually used as abbreviated name of day of week...

Translating software using GNU Gettext

GNU Gettext is one of the most widely used tool for internationalization of free software. It provides 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 it's usage really simple, instead of manual invoking of Gettext described below, you might want to try one of them, for example intltool.

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 message from it:

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

This creates 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 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"
"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 first translation:

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

The just created `cs.po` has already some information filled in. Most importantly it got 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] ""
```

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

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 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	`po/*.po`
Base file for new translations	`po/hello.pot`
File format	Choose *Gettext PO file*
New language	Choose *Automatically add language file*

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 Quick Guide. 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 it's 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 *`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 direcly 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/dir1/**.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.

Managing translations

Adding new translations

Weblate can add new language files to your project automatically for most of the *Supported formats*. This feature needs to be enabled in the *Component configuration*. In case this is not enabled (or available for your file format) the files have to be added manually to the VCS.

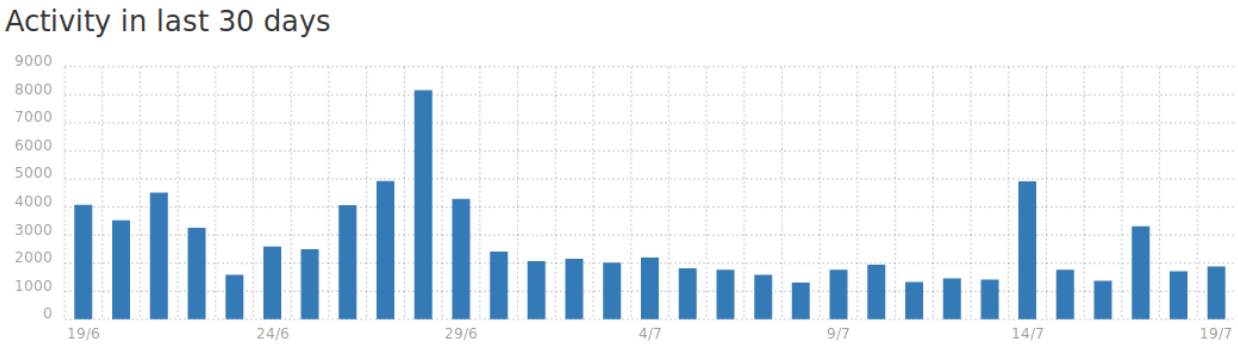
Weblate will automatically detect new languages which are added to the VCS repository and makes them available for translation. This makes adding new translations incredibly easy:

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

Reviewing source strings

Activity reports

You can check activity reports for translations, project or individual users.





Source strings checks

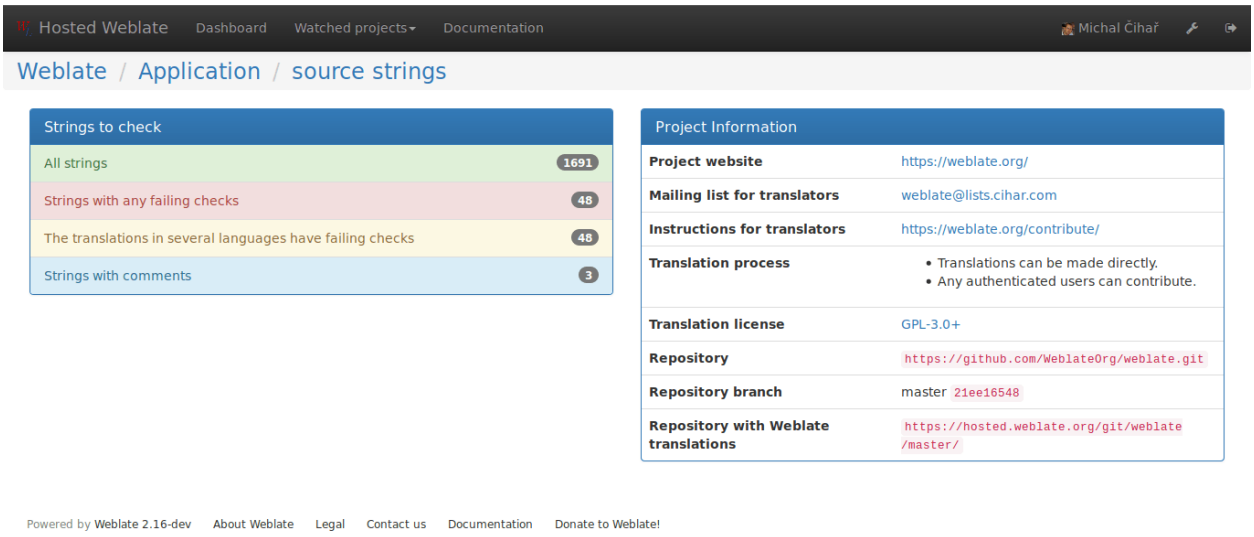
Weblate includes quite a lot of *Quality checks*. Some of them also focus on quality of source strings. These can give you some hints for making strings easier to translate. You can check failing source checks on *Source* tab of every component.

Failing checks on translation

On the other side, failing translation checks might also indicate problem in the source strings. Translators often tend to fix some mistakes in translation instead of reporting it - typical example is missing full stop at the end of sentence, but there are more such cases.

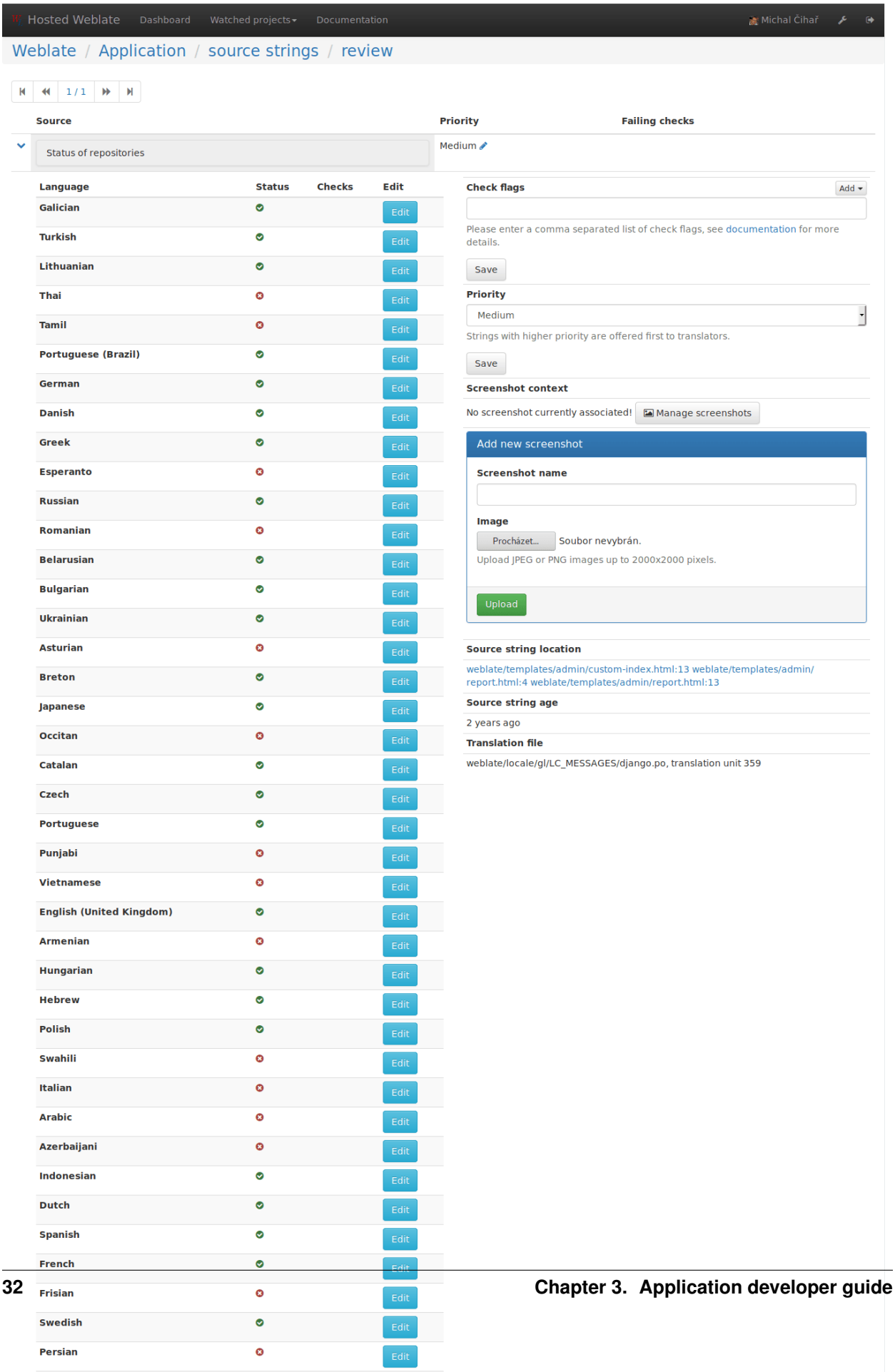
Reviewing all failing checks on your translation can bring you valuable feedback for improving source strings as well.

You can find the *Source strings review* in the *Tools* menu of a translation component. You will get similar view when opening translation, with slightly different checks being displayed:



One of the most interesting check here is the *Multiple failing checks* - it fires whenever there is failure on multiple translations of given string. Usually this is something to look for as this is string where translators have problems doing the translation properly. It might be just wrong punctation at the end of sentence or something more problematic.

The detailed listing then shows you overview per language:


Hosted Weblate
Dashboard
Watched projects
Documentation
Michal Čihař
Weblate / Application / source strings / review
1 / 1
Source
Priority
Failing checks
Status of repositories
Medium
Language
Status
Checks
Edit
Galician
Turkish
Lithuanian
Thai
Tamil
Portuguese (Brazil)
German
Danish
Greek
Esperanto
Russian
Romanian
Belarusian
Bulgarian
Ukrainian
Asturian
Breton
Japanese
Occitan
Catalan
Czech
Portuguese
Punjabi
Vietnamese
English (United Kingdom)
Armenian
Hungarian
Hebrew
Polish
Swahili
Italian
Arabic
Azerbaijani
Indonesian
Dutch
Spanish
French
Frisian
Swedish
Persian
Check flags
Add
Please enter a comma separated list of check flags, see documentation for more details.
Save
Priority
Medium
Strings with higher priority are offered first to translators.
Save
Screenshot context
No screenshot currently associated!
Manage screenshots
Add new screenshot
Screenshot name
Image
Procházet...
Soubor nevybrán.
Upload JPEG or PNG images up to 2000x2000 pixels.
Upload
Source string location
weblate/templates/admin/custom-index.html:13 weblate/templates/admin/report.html:4 weblate/templates/admin/report.html:13
Source string age
2 years ago
Translation file
weblate/locale/gl/LC_MESSAGES/django.po, translation unit 359

String comments

Weblate allows translators to comment on both translation and source strings. Each *Component configuration* can be configured to receive such comments on email address and sending this to developers mailing list is usually best approach. This way you can monitor when translators find problems and fix them quickly.

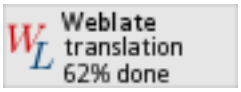
Promoting the translation

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. Example of status badges for Weblate itself are show below.

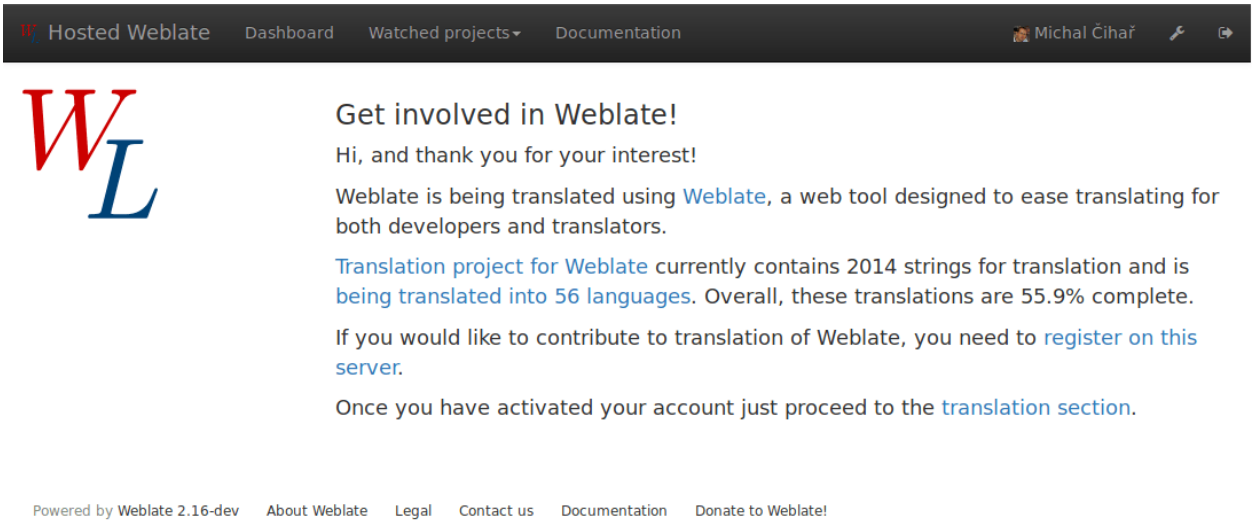
Shields.IO badge often used to quickly see status of a project: 

Small badge often used to quickly see status of a project: 

Big badge with status details useful for inclusion on a web page: 

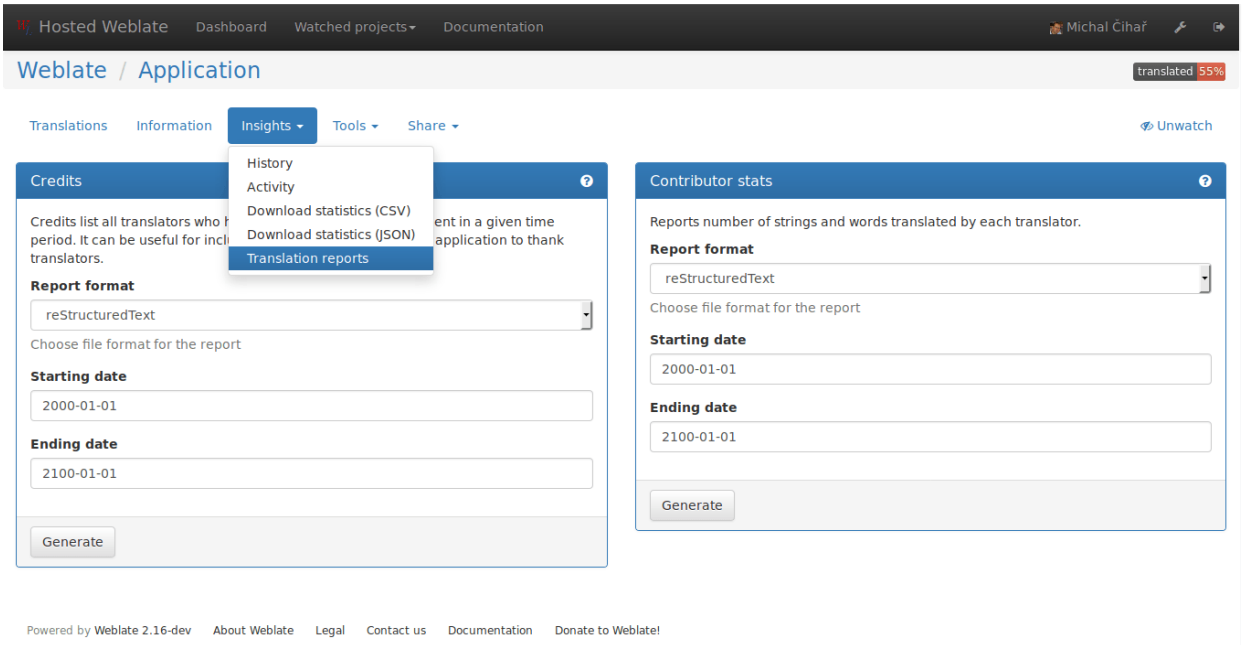
Small badge with status useful for inclusion on a web page: 

All these badges come with links to simple page which explains users how to translate using Weblate:



Translation progress reporting

It is often useful to be able to see how translation progresses over given period. For this purpose Weblate includes reporting features, where you can obtain summaries of contributions to given component over time. You can find the reporting tool in the *Insights* menu for a translation component:



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 into existing documentation, while JSON is useful for further processing of the data.

Translator credits

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

```
* Czech

    * Michal Čihař <michal@cihar.com>
    * Weblate Admin <admin@example.com>

* Dutch

    * Weblate Admin <admin@example.com>
```

And it will get rendered as:

- Czech
  - Michal Čihař <michal@cihar.com>
  - Weblate Admin <admin@example.com>
- Dutch
  - Weblate Admin <admin@example.com>

Contributor stats

Generates number of words and units translated by translators:

```
======================================== ========================================
 ↪========== ==========
Name                                     Email
                                                                                 
 ↪Words      Count
======================================== ========================================
 ↪========== ==========
Michal Čihař                             michal@cihar.com
                                                                                 
 ↪  2332        421
Weblate Admin                            admin@example.com
                                                                                 
 ↪    25          8
======================================== ========================================
 ↪========== ==========
```

And it will get rendered as:

Name	Email	Words	Count
Michal Čihař	michal@cihar.com	2332	421
Weblate Admin	admin@example.com	25	8

CHAPTER 4

Administrators guide

Quick setup guide

Note: This is just a quick guide for installing and starting to use Weblate for testing purposes. Please check *Installation instructions* for more real world setup instructions.

Installing from sources

1. Install all required dependencies, see *Software requirements*.
2. Grab Weblate sources (either using Git or download a tarball) and unpack them, see *Installing Weblate*.
3. Copy `weblate/settings_example.py` to `weblate/settings.py` and adjust it to match your setup. You will at least need to configure database connection (possibly adding user and creating the database). Check *Configuration* for Weblate specific configuration options.
4. Create database which will be 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*.

Using prebuilt appliance

1. Download the appliance and start it. You need to choose format depending on your target environment.
2. Everything should be set up immediately after boot, though you will want to adjust some settings to improve security, see *Weblate as a SUSE Studio appliance*.

Installing on OpenShift

1. You can install Weblate on OpenShift PaaS directly from its git repository using the OpenShift Client Tools:

   ```
   rhc -aweblate 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 to add a translation project as described below. For more information, including on how to retrieve the generated admin password, see *Running Weblate on OpenShift*.

Adding translation

1. Open admin interface (http://localhost/admin/) and create project you want to translate. See *Project configuration* for more details.

   All you need to specify here is project name and its website.

2. Create component which is the real object for translating - it points to VCS repository and selects which files to translate. See *Component configuration* for more details.

   The important fields here being component name, VCS repository address and mask for finding translatable files. Weblate supports wide range of formats including Gettext PO files, Android resource strings, OS X string properties, Java properties or Qt Linguist files, see *Supported formats* for more details.

3. Once above is completed (it can be lengthy process depending on size of your VCS repository and number of messages to translate), you can start translating.

Installation instructions

Hardware requirements

Weblate should run on any contemporary hardware without problems, following are minimal configuration required to run Weblate on single host (Weblate, database and web server):

- 1 GB of RAM memory
- 2 CPU cores
- 1 GB of storage space

The more memory you have, the better - it will be used for caching on all levels (filesystem, database and Weblate).

Note: The actual requirements for your installation heavily vary on size of translations managed by Weblate.

Software requirements

Python dependencies

Weblate is written in Python and support Python 2.7, 3.4 or newer. Following dependencies can be installed using pip or your distribution packages:

Django (>= 1.10) https://www.djangoproject.com/

siphashc3 https://github.com/carlopires/siphashc3

Translate-toolkit (>= 2.0.0) http://toolkit.translatehouse.org/

Six (>= 1.7.0) https://pypi.python.org/pypi/six

Mercurial (>= 2.8) (optional for Mercurial repositories support) https://www.mercurial-scm.org/

social-auth-core (>= 1.3.0) https://python-social-auth.readthedocs.io/

social-auth-app-django (>= 1.2.0) https://python-social-auth.readthedocs.io/

django-appconf (>= 1.0) https://github.com/django-compressor/django-appconf

Whoosh (>= 2.7.0) https://bitbucket.org/mchaput/whoosh/wiki/Home

PIL or Pillow library https://python-pillow.org/

lxml (>= 3.1.0) http://lxml.de/

PyYaML (>= 3.0) (optional for YAML support) http://pyyaml.org/wiki/PyYAML

defusedxml (>= 0.4) https://bitbucket.org/tiran/defusedxml

dateutil https://labix.org/python-dateutil

django_compressor (>= 2.1.1) https://github.com/django-compressor/django-compressor

django-crispy-forms (>= 1.6.1) https://django-crispy-forms.readthedocs.io/

Django REST Framework (>=3.4) http://www.django-rest-framework.org/

libravatar (optional for federated avatar support) You need to additionally install pydns (on Python 2) or py3dns (on Python 3) to make libravatar work.

https://pypi.python.org/pypi/pyLibravatar

pyuca (>= 1.1) (optional for proper sorting of strings) https://github.com/jtauber/pyuca

babel (optional for Android resources support) http://babel.pocoo.org/

Database backend Any database supported in Django will work, see *Database setup for Weblate* and backends documentation for more details.

pytz (optional, but recommended by Django) https://pypi.python.org/pypi/pytz/

python-bidi (optional for proper rendering of badges in RTL languages) https://github.com/MeirKriheli/python-bidi

tesserocr (>= 2.0.0) (optional for screenshots OCR) https://github.com/sirfz/tesserocr

Other system requirements

Following dependencies have to be installed on the system:

Git (>= 1.6) https://git-scm.com/

hub (optional for sending pull requests to GitHub) https://hub.github.com/

git-review (optional for Gerrit support) https://pypi.python.org/pypi/git-review

git-svn (>= 2.10.0) (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 and please consult individual packages for domentation. You won't need those if using prebuilt Wheels while installing using pip or when you use distribution packages.

Installing Weblate

Choose installation method that best fits your environment.

First choices include complete setup without relying on your system libraries:

- *Installing in virtualenv*
- *Running Weblate in the Docker*
- *Running Weblate on OpenShift*
- *Weblate as a SUSE Studio appliance*

You can also install Weblate directly on your system either fully using distribution packages (as of now available for openSUSE only) or mixed setup.

Choose installation method:

- *Installing Weblate by pip*
- *Installing Weblate from Git* (if you want to run bleeding edge version)
- Alternatively you can use released archives. You can download them from our website <https://weblate.org/>.

And install dependencies according your platform:

- *Requirements on Debian or Ubuntu*
- *Requirements on openSUSE*
- *Requirements on OSX*
- *Requirements using pip installer*

Installing in virtualenv

This is recommended method if you don't want to dig into details. This will create separate Python environment for Weblate, possibly duplicating some system Python libraries.

1. Install development files for libraries we will use during building Python modules:

```
# Debian/Ubuntu:
apt install libxml2-dev libxslt-dev libfreetype6-dev libjpeg-dev libz-dev libyaml-
↪dev python-dev

# openSUSE/SLES:
zypper install libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-
↪devel libyaml-devel python-devel

# Fedora/RHEL/CentOS:
dnf install libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel␣
↪libyaml-devel python-devel
```

2. Install pip and virtualenv. Usually they are shipped by your distribution or with Python:

```
# Debian/Ubuntu:
apt-get install python-pip python-virtualenv

# openSUSE/SLES:
zypper install python-pip python-virtualenv

# Fedora/RHEL/CentOS:
dnf install python-pip python-virtualenv
```

3. Create and activate virtualenv for Weblate (the path in `/tmp` is really just an example, you rather want something permanent):

```
virtualenv /tmp/weblate
. /tmp/weblate/bin/activate
```

4. Install Weblate including all dependencies, you can also use pip to install optional dependecies:

```
pip install Weblate
# Optional deps
pip install pytz python-bidi PyYaML Babel pyuca pylibravatar pydns
```

5. Create your settings (in our example it would be in `/tmp/weblate/lib/python2.7/site-packages/weblate/settings.py` based on the `settings_example.py` in same directory).
6. You can now run Weblate commands using **`weblate`** command, see *Management commands*.
7. To run webserver, use the wsgi wrapper installed with Weblate (in our case it is `/tmp/weblate/lib/python2.7/site-packages/weblate/wsgi.py`). Don't forget to set Python search path to your virtualenv as well (for example using `virtualenv = /tmp/weblate` in uwsgi).

Installing Weblate from Git

You can also run latest version from Git. It is maintained stable and production ready. You can usually find it running on Hosted Weblate.

To get latest sources using Git use:

```
git clone https://github.com/WeblateOrg/weblate.git
```

Note: If you are running version from Git, you should also regenerate locale files every time you are upgrading. You can do this by invoking script `./scripts/generate-locales`.

Installing Weblate by pip

If you decide to install Weblate using 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, either by environment variable *DJANGO_SETTINGS* or on the command line, for example:

```
weblate --settings=yourproject.settings migrate
```

See also:

Invoking management commands

Requirements on Debian or Ubuntu

On recent Debian or Ubuntu, most of requirements are already packaged, to install them you can use apt-get:

```
apt-get install python-pip python-django translate-toolkit \
    python-whoosh python-pil python-libravatar \
    python-babel git mercurial \
    python-django-compressor python-django-crispy-forms \
    python-djangorestframework python-dateutil

# Optional packages for database backend:

# For PostgreSQL
apt-get install python-psycopg2
# For MySQL on Ubuntu (if using Ubuntu package for Django)
apt-get install python-pymysql
# For MySQL on Debian (or Ubuntu if using upstream Django packages)
apt-get install python-mysqldb
```

On older versions, some required dependencies are missing or outdated, so you need to install several Python modules manually using pip:

```
# Dependencies for python-social-auth
apt-get install python-requests-oauthlib python-six python-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 python-django-crispy-forms package is missing
pip install django-crispy-forms

# In case python-whoosh package is misssing or older than 2.7
pip install Whoosh

# In case your python-django-compressor package is missing,
# try installing it by older name or using pip:
apt-get install python-compressor
pip install django_compressor

# Optional for OCR support
apt-get install tesseract-ocr libtesseract-dev libleptonica-dev cython
pip install tesserocr
```

For proper sorting of a 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-get install nginx uwsgi uwsgi-plugin-python

# Web server option 2: Apache with mod_wsgi
```

```
apt-get install apache2 libapache2-mod-wsgi

# Caching backend: memcached
apt-get install memcached

# Database option 1: postgresql
apt-get install postgresql

# Database option 2: mariadb
apt-get install mariadb-server

# Database option 3: mysql
apt-get install mysql-server

# SMTP server
apt-get 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 python-Django translate-toolkit \
    python-Whoosh python-Pillow \
    python-social-auth-core python-social-auth-app-django \
    python-babel Git mercurial python-pyuca \
    python-dateutil

# Optional for database backend
zypper install python-psycopg2      # For PostgreSQL
zypper install python-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-python

# Web server option 2: Apache with mod_wsgi
zypper install apache2 apache2-mod_wsgi

# Caching backend: memcached
zypper install memcached

# 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 OSX

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/python2.7/site-packages:$PYTHONPATH"
```

This configuration makes the installed libraries available to Python.

Requirements using pip installer

Most requirements can be also installed using pip installer:

```
pip install -r requirements.txt
```

For building some of the extensions devel files for several libraries are required, see *Installing in virtualenv* for instructions how to install these.

All optional dependencies (see above) can be installed using:

```
pip install -r requirements-optional.txt
```

Filesystem permissions

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 same tree as Weblate sources, however you might prefer to move these to 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 run under same user as Weblate itself is running, otherwise permissions on some files might be wrong.

See also:

Serving static files

Database setup for Weblate

It is recommended to run Weblate on some database server. Using SQLite backend is really good for testing purposes only.

See also:

Use powerful database engine, Databases

PostgreSQL

PostgreSQL is usually best choice for Django based sites. It's the reference database using for implementing Django database layer.

See also:

PostgreSQL notes

Creating database in PostgreSQL

It is usually good idea to run Weblate in separate database and separate user:

```
# If PostgreSQL was not installed before, set the master password
sudo -u postgres psql postgres -c "\password postgres"

# Create database user called "weblate"
sudo -u postgres createuser -D -P weblate

# Create 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_psycopg2',
        # 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 choice to run Weblate. However when using MySQL you might hit some problems caused by it.

See also:

MySQL notes

Unicode issues in MySQL

MySQL by default uses something called `utf8`, what 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 MySQL and Python bindings version, the error might look like:

- `OperationalError: (1366, "Incorrect string value: '\\xF0\\xA8\\xAB\\xA1' 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` (what is again subset of Unicode, but this time which can be stored in four bytes in `utf-8` encoding, thus covering all chars currently defined in Unicode).

This can be achieved at database creation time by creating it with this character set (see *Creating database in MySQL*) and specifying the character set in connection settings (see *Configuring Weblate to use MySQL*).

In case you have existing database, you can change it to `utf8mb4` by, but this won't change collation of existing fields:

```
ALTER DATABASE weblate CHARACTER SET utf8mb4;
```

See also:

Using Innodb_large_prefix to Avoid ERROR 1071

Transaction locking

MySQL by default uses has 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 database in MySQL

Create `weblate` user to access the `weblate` database:

```
# Grant all privileges to  weblate user
GRANT ALL PRIVILEGES ON weblate.* TO 'weblate'@'localhost'  IDENTIFIED BY 'password';

# Create 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 older MySQL server which has default MariaDB
            # 'init_command': 'SET storage_engine=INNODB',
            # If your server supports it, see Unicode issues above
           'charset': 'utf8mb4',
        }

    }
}
```

Other configurations

Configuring outgoing mail

Weblate sends out emails on various occasions - for account activation and on various notifications configured by users. For this it needs access to the SMTP server, which will handle this.

The mail server setup is configured using settings `EMAIL_HOST`, `EMAIL_HOST_PASSWORD`, `EMAIL_HOST_USER` and `EMAIL_PORT`. Their names are quite self-explaining, but you can find our more information in the Django documentation.

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 following options: `ADMINS`

> List of site administrators to receive notifications when something goes wrong, for example notifications on failed merge or Django errors.
>
> **See also:**
>
> `ADMINS`

`ALLOWED_HOSTS`

If you are running Django 1.5 or newer, you need to set this to list of 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 default database backed engine you should schedule **./manage.py clearsessions** to remove stale session data from the database.

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 production server. With debug mode enabled, Django will show backtraces in case of error to users, when you disable it, errors will go by email to ADMINS (see above).

Debug mode also slows down Weblate as Django stores much more information 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 information in cookies, see *Django secret key* for more information.

SERVER_EMAIL

Email used as sender address for sending emails to administrator, for example notifications on failed merge.

See also:

SERVER_EMAIL

Filling up the database

After your configuration is ready, you can run `./manage.py migrate` to create database structure. Now you should be able to create translation projects using admin interface.

In case you want to run installation non interactively, you can use `./manage.py migrate --noinput` and then create admin user using *createadmin* command.

You should also login to admin interface (on `/admin/` URL) and adjust default site name to match your domain by clicking on *Sites* and there changing the `example.com` record to match your real domain name.

Once you are done, you should also check *Performance report* in the admin interface which will give you hints for non optimal configuration on your site.

See also:

Configuration, *Access control*, *Why does links contain example.com as domain?*, *Set correct site name*

Production setup

For production setup you should do following adjustments:

Disable debug mode

Disable Django's debug mode (*DEBUG*) by:

```
DEBUG = False
```

With debug mode Django stores all executed queries and shows users backtraces of errors what is not desired in production setup.

See also:

Installation

Properly configure admins

Set correct admin addresses to *ADMINS* setting for defining who will receive mail in case something goes wrong on the server, for example:

```
ADMINS = (
    ('Your Name', 'your_email@example.com'),
)
```

See also:

Installation

Set correct site name

Adjust site name in admin interface, otherwise links in RSS or registration emails will not work.

Please open admin interface and edit default site name and domain under the *Sites › Sites* (or you can do that directly at `/admin/sites/site/1/` URL under your Weblate installation). You have to change the *Domain name* to match your setup.

Note: This setting should contain only 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 command line using *changesite*. For example for using built in server:

```
./manage.py changesite --set-name 127.0.0.1:8000
```

For production site, you want something like:

```
./manage.py changesite --set-name weblate.example.com
```

See also:

Why does links contain example.com as domain?, `changesite`, The "sites" framework

Enable indexing offloading

Enable `OFFLOAD_INDEXING` to prevent locking issues and improve performance. Don't forget to schedule indexing in background job to keep the index up to date.

See also:

Fulltext search, `OFFLOAD_INDEXING`, *Running maintenance tasks*

Use powerful database engine

Use powerful database engine (SQLite is usually not good enough for production environment), see *Database setup for Weblate* for more information.

See also:

Database setup for Weblate, *Installation*, Databases

Enable caching

If possible, use memcache from Django by adjusting `CACHES` configuration variable, for example:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

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 separate, file backed cache for this purpose:

```
CACHES = {
    'default': {
        # Default caching backend setup, see above
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    },
    'avatar': {
```

```
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': os.path.join(BASE_DIR, 'avatar-cache'),
        'TIMEOUT': 604800,
        'OPTIONS': {
            'MAX_ENTRIES': 1000,
        },
    }
```

See also:

ENABLE_AVATARS, *Avatars*, *Enable caching*, Django's cache framework

Configure email addresses

Weblate needs to send out emails on several occasions and these emails should have 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, DEFAULT_FROM_EMAIL, SERVER_EMAIL

Allowed hosts setup

Django 1.5 and newer require *ALLOWED_HOSTS* to hold list of domain names your site is allowed to serve, having it empty will block any request.

See also:

ALLOWED_HOSTS

Federated avatar support

By default, Weblate relies on <https://www.libravatar.org/> for avatars. When you install pyLibavatar, you will get proper support for federated avatars.

pyuca library

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 with accented letters.

Django secret key

The *SECRET_KEY* setting is used by Django to sign cookies and you should really use own value rather than using the one coming from example setup.

You can generate new key using `examples/generate-secret-key` shipped with Weblate.

See also:

SECRET_KEY

Static files

If you see purely designed admin interface, the CSS files required for it are not loaded. This is usually if you are running in non-debug mode and have not configured your web server to serve them. Recommended setup is described in the *Serving static files* chapter.

See also:

Running server, *Serving static files*

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 user which is running Weblate should be existing and 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 (depends on Git version you use).

You can change the directory used by Weblate in `settings.py`, for example to set it to `configuration` directory under 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 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 different user or change this setting.

See also:

Accessing repositories

Template loading

It is recommended to use cached template loader for Django. It caches parsed templates and avoids need to do the parsing with every single request. You can configure it using following snippet:

```
TEMPLATE_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.

On Unix system, this can be scheduled using cron:

```
# Fulltext index updates
*/5 * * * * cd /usr/share/weblate/; ./manage.py update_index

# Cleanup stale objects
@daily cd /usr/share/weblate/; ./manage.py cleanuptrans

# Commit pending changes after 96 hours
@hourly cd /usr/share/weblate/; ./manage.py commit_pending --all --age=96 --
↪verbosity=0
```

See also:

Enable indexing offloading, `update_index`, `cleanuptrans`, `commit_pending`

Running server

Running Weblate is not different from running any other Django based application. Django is usually executed as uwsgi or fcgi (see examples for different webservers below).

For testing purposes, you can use Django builtin web server:

```
./manage.py runserver
```

Serving static files

Changed in version 2.4: Prior to version 2.4 Weblate didn't properly use Django static files framework and the setup was more complex.

Django needs to collect its static files to a single directory. To do so, execute `./manage.py collectstatic --noinput`. This will copy the static files into directory specified by `STATIC_ROOT` setting (this default to `static` directory inside `DATA_DIR`).

It is recommended to serve static files directly by your web server, you should use that for following paths:

`/static/` Serves static files for Weblate and admin interface (from defined by `STATIC_ROOT`).

`/media/` Used for user media uploads (eg. screenshots).

`/favicon.ico` Should be rewritten to rewrite rule to serve `/static/favicon.ico`

`/robots.txt` Should be rewritten to rewrite rule to serve `/static/robots.txt`

See also:

Deploying Django, Deploying static files

Content security policy

Default Weblate configuration enables `weblate.middleware.SecurityMiddleware` middleware which sets security related HTTP headers like `Content-Security-Policy` or `X-XSS-Protection`. These are set to work with Weblate and it's configuration, but this might clash with your customization. If that is your case, it is recommended to disable this middleware and set these headers manually.

Sample configuration for Apache

Following configuration runs Weblate as WSGI, you need to have enabled mod_wsgi (available as `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/python2.7/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

Following configuration runs Weblate in gunicorn and Apache 2.4 (available as `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

Following configuration runs Weblate as uwsgi under nginx webserver.

Configuration for nginx (also available as `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 `examples/weblate.uwsgi.ini`):

```
[uwsgi]
plugins       = python
master        = true
protocol      = uwsgi
socket        = 127.0.0.1:8080
wsgi-file     = /path/to/weblate/weblate/wsgi.py
python-path   = /path/to/weblate
# In case you're using virtualenv uncomment this:
# virtualenv = /path/to/weblate/virtualenv
# Needed for OAuth/OpenID
buffer-size   = 8192
# Increase number of workers for heavily loaded sites
#workers       = 6
# Needed for background processing
enable-threads = true
# Child processes do not need file descriptors
close-on-exec = true
```

```
# Avoid default 0000 umask
umask = 0022
```

See also:

How to use Django with uWSGI

Running Weblate under path

Changed in version 1.3: This is supported since Weblate 1.3.

Sample Apache configuration to serve Weblate under `/weblate`. Again using mod_wsgi (also available as `examples/apache-path.conf`):

```
# Example Apache configuration for running Weblate under /weblate path

WSGIPythonPath /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/python2.7/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

    # 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
    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'
```

Monitoring Weblate

Weblate provides `/healthz/` URL to be used in simple health checks, for example using Kubernetes.

Collecting error reports

It is good idea to collect errors from any Django application in structured way and Weblate is not an exception from this. You might find several services providing this, for example:

- Sentry
- Rollbar

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_CLASSES = (
    # ... 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.

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 migrated 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 most effective (eg. **`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 to generate fresh one using *rebuild_index*.

Other notes

Don't forget to move other services which Weblate might have been using like memcached, cron jobs or custom authentication backends.

Weblate deployments

Weblate comes with support for deployment using several technologies. This section brings overview of them.

Running Weblate in the Docker

With dockerized weblate deployment you can get your personal weblate instance up an running in seconds. All of Weblate's dependencies are already included. PostgreSQL is configured as default database.

Deployment

Following examples assume you have working Docker environment, with docker-compose installed. Please check Docker documentation for instructions on this.

1. Clone weblate-docker repo:

```
git clone https://github.com/WeblateOrg/docker.git weblate-docker
cd weblate-docker
```

2. Create a `docker-compose.override.yml` file with your settings. See *Docker environment variables* full list of environment vars

```
version: '2'
services:
  weblate:
    environment:
      - WEBLATE_EMAIL_HOST=smtp.example.com
      - WEBLATE_EMAIL_HOST_USER=user
      - WEBLATE_EMAIL_HOST_PASSWORD=pass
      - WEBLATE_ALLOWED_HOSTS=your hosts
      - WEBLATE_ADMIN_PASSWORD=password for admin user
```

Note: If *WEBLATE_ADMIN_PASSWORD* is not set, admin user is created with random password printed out on first startup.

3. Build Weblate containers:

```
docker-compose build
```

4. 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, prior there was separate web server container, since 2.15-2 the web server is embedded in weblate container.

See also:

Invoking management commands

Upgrading Docker container

Usually it is good idea to update weblate container only and keep PostgreSQL one at version you have as upgrading PostgreSQL is quite painful and in most cases it does not bring much benefits.

You can do this by sticking with existing docker-compose and just pulling latest images and restarting:

```
docker-compose down
docker-compose pull
docker-compose build --pull
docker-compose up
```

The Weblate database should be automatically migrated on first start and there should be no need for additional manual actions.

Maintenance tasks

There are some cron jobs to run. You should set *WEBLATE_OFFLOAD_INDEXING* to `1` when these are setup

```
*/5 * * * * cd /usr/share/weblate/; docker-compose run --rm weblate update_index
@daily cd /usr/share/weblate/; docker-compose run --rm weblate cleanuptrans
@hourly cd /usr/share/weblate-docker/; docker-compose run --rm weblate commit_pending␣
↪--all --age=96
```

Docker environment variables

Many of Weblate *Configuration* can be set in Docker container using environment variables:

Generic settings

WEBLATE_DEBUG

Configures Django debug mode using *DEBUG*, see *Disable debug mode*.

Example:

```
environment:
  - WEBLATE_DEBUG=1
```

WEBLATE_LOGLEVEL
Configures verbosity of logging.

WEBLATE_SITE_TITLE
Configures site title, see *Set correct site name*.

WEBLATE_ADMIN_NAME

WEBLATE_ADMIN_EMAIL
Configures site admins name and email, see *Properly configure admins*.

Example:

```
environment:
  - WEBLATE_ADMIN_NAME=Weblate Admin
  - WEBLATE_ADMIN_EMAIL=noreply@example.com
```

WEBLATE_ADMIN_PASSWORD
Sets password for admin user. If not set, admin user is created with random password printed out 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 address for outgoing mails, see *Configure email addresses*.

WEBLATE_ALLOWED_HOSTS
Configures allowed HTTP hostnames using *ALLOWED_HOSTS*, see *Allowed hosts setup*

Example:

```
environment:
  - WEBLATE_ALLOWED_HOSTS=weblate.example.com,example.com
```

WEBLATE_SECRET_KEY
Configures secret for cookies signing, see *Django secret key*.

Deprecated since version 2.9: The secret is now generated automatically on first startup, there is no need to set it manually.

WEBLATE_REGISTRATION_OPEN
Configures whether registrations are open, see *REGISTRATION_OPEN*.

Example:

```
environment:
  - WEBLATE_REGISTRATION_OPEN=0
```

WEBLATE_TIME_ZONE
Configures used time zone.

WEBLATE_OFFLOAD_INDEXING
Configures offloaded indexing, see *Enable indexing offloading*.

Example:

```
environment:
  - WEBLATE_OFFLOAD_INDEXING=1
```

WEBLATE_ENABLE_HTTPS
Configures when use https in email and API links, see *Set correct site name*.

Example:

```
environment:
  - WEBLATE_ENABLE_HTTPS=1
```

WEBLATE_REQUIRE_LOGIN
Configures login required for whole Weblate using *LOGIN_REQUIRED_URLS*.

Example:

```
environment:
  - WEBLATE_REQUIRE_LOGIN=1
```

WEBLATE_GOOGLE_ANALYTICS_ID
Configures ID for Google Analytics, see *GOOGLE_ANALYTICS_ID*.

WEBLATE_GITHUB_USERNAME
Configures github username for GitHub pull requests, see *GITHUB_USERNAME*.

See also:

Pushing changes to GitHub as pull request, *Setting up hub*

Machine translation settings

WEBLATE_MT_GOOGLE_KEY
Enables Google machine translation and sets *MT_GOOGLE_KEY*

WEBLATE_MT_MICROSOFT_COGNITIVE_KEY
Enables Microsoft machine translation and sets *MT_MICROSOFT_COGNITIVE_KEY*

Authentication settings

WEBLATE_SOCIAL_AUTH_GITHUB_KEY

WEBLATE_SOCIAL_AUTH_GITHUB_SECRET
Enables *GitHub authentication*.

WEBLATE_SOCIAL_AUTH_BITBUCKET_KEY

WEBLATE_SOCIAL_AUTH_BITBUCKET_SECRET
Enables *Bitbucket authentication*.

WEBLATE_SOCIAL_AUTH_FACEBOOK_KEY

WEBLATE_SOCIAL_AUTH_FACEBOOK_SECRET
Enables *Facebook OAuth2*.

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET
Enables *Google OAuth2*.

WEBLATE_SOCIAL_AUTH_GITLAB_KEY

WEBLATE_SOCIAL_AUTH_GITLAB_SECRET

WEBLATE_SOCIAL_AUTH_GITLAB_API_URL
Enables *Gitlab OAuth2*.

Processing hooks

All these processing hooks should get comma separaated list of available scripts, for example:

```
WEBLATE_POST_UPDATE_SCRIPTS=/usr/local/share/weblate/examples/hook-cleanup-android
```

See also:

Processing repository with scripts

WEBLATE_POST_UPDATE_SCRIPTS
Sets *POST_UPDATE_SCRIPTS*.

WEBLATE_PRE_COMMIT_SCRIPTS
Sets *PRE_COMMIT_SCRIPTS*.

WEBLATE_POST_COMMIT_SCRIPTS
Sets *POST_COMMIT_SCRIPTS*.

WEBLATE_POST_PUSH_SCRIPTS
Sets *POST_PUSH_SCRIPTS*.

WEBLATE_POST_ADD_SCRIPTS
Sets *POST_ADD_SCRIPTS*.

PostgreSQL database setup

The database is created by `docker-compose.yml`, so this settings affects 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 adress. Defaults to *database*.

POSTGRES_PORT
PostgreSQL server port. Default to empty (use default value).

Email server setup

To make outgoing email work, you need to provide mail server.

See also:

Configuring outgoing mail

WEBLATE_EMAIL_HOST
Mail server, the server has to listen on port 587 and understand TLS.

WEBLATE_EMAIL_PORT
Mail server port, use if your cloud provider or ISP blocks outgoing connections on port 587.

WEBLATE_EMAIL_HOST_USER
Email authentication user, do NOT use quotes here.

WEBLATE_EMAIL_HOST_PASSWORD
Email authentication password, do NOT use quotes here.

Hub setup

In order to use the Github pull requests feature, you must initialize hub configuration by entering the weblate container and executing an arbitrary hub command. For example:

```
docker-compose exec weblate bash
cd
HOME=/app/data/home hub clone octocat/Spoon-Knife
```

The username passed for credentials must be the same than *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, Digitalocean and many more providers.

Running Weblate on OpenShift

This repository contains a configuration for the OpenShift platform as a service product, which facilitates easy installation of Weblate on OpenShift Online (https://www.openshift.com/), OpenShift Enterprise (https://enterprise.openshift.com/) and OpenShift Origin (https://www.openshift.org/).

Prerequisites

1. OpenShift Account

   You need an account for OpenShift Online (https://www.openshift.com/) or another OpenShift installation you have access to.

   You can register a free account on OpenShift Online, which allows you to host up to 3 applications free of charge.

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://developers.openshift.com/en/managing-client-tools.html

   While there are other possibilities to create and configure OpenShift applications, 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 HEAD
rhc -aweblate app create -t python-2.7 --from-code https://github.com/WeblateOrg/
↪weblate.git --no-git

# Install Weblate 2.10
rhc -aweblate app create -t python-2.7 --from-code https://github.com/WeblateOrg/
↪weblate.git#weblate-2.10 --no-git
```

The `-a` option defines the name of your weblate installation, `weblate` in this instance. You are free to specify a different name.

The above example installs latest development version, you can optionally specify tag identifier right of the `#` sign to identify the version of Weblate to install. For a list of available versions see 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 to use and preconfigured as follows:

- SQLite embedded database (*DATABASES*)
- Random admin password
- Random Django secret key (*SECRET_KEY*)
- Indexing offloading if the cron cartridge is installed (*OFFLOAD_INDEXING*)
- Committing of pending changes if the cron cartridge is installed (*commit_pending*)
- Weblate machine translations for suggestions bases on previous translations (*MACHINE_TRANSLATION_SERVICES*)
- Weblate directories (STATIC_ROOT, *DATA_DIR*, *TTF_PATH*, Avatar cache) set according to OpenShift requirements/conventions
- Django site name and ALLOWED_HOSTS set to DNS name of your OpenShift application

- 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's rhcloud.com.

See also:

Customize Weblate Configuration

Retrieve Admin Password

You can retrieve the generated admin password with the following command:

```
rhc -aweblate ssh credentials
```

Indexing Offloading

To enable the preconfigured indexing offloading you need to add the cron cartridge to your application and restart it:

```
rhc -aweblate add-cartridge cron
rhc -aweblate app stop
rhc -aweblate app start
```

The fulltext search index will then be updated every 5 minutes. Restarting with `rhc restart` instead will not enable indexing offloading in Weblate. You can verify that indexing offloading is indeed enabled by visiting the URL `/admin/performance/` of your application.

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 Weblate Configuration

You can customize the configuration of your Weblate installation on OpenShift through environment variables. Override any of Weblate's setting documented under *Configuration* using `rhc env set` by prepending the settings name with `WEBLATE_`. The variable content is put verbatim to the configuration file, so it is parsed as Python string, after replacing environment variables in it (eg. `$PATH`). To put literal `$` you need to escape it as `$$`.

For example override the *ADMINS* setting like this:

```
rhc -aweblate env set WEBLATE_ADMINS='(("John Doe", "jdoe@example.org"),)'
```

To change site title, do not forget to include additional quotes:

```
rhc -aweblate env set WEBLATE_SITE_TITLE='"Custom Title"'
```

New settings will only take effect after restarting Weblate:

```
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_PRE_COMMIT_SCRIPTS='("${OPENSHIFT_DATA_DIR}/examples/
↪hook-generate-mo",)'
```

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 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 right of the # sign identifies the version of Weblate to install. For a list of available versions see here: https://github.com/WeblateOrg/weblate/tags. Please note that the update process will not work if you modified the git repository of you weblate installation. You can force an update by specifying the `--force` option to 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 under *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

Bitnami Weblate stack

Bitnami provides 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.

Weblate as a SUSE Studio appliance

Weblate appliance provides preconfigured Weblate running with PostgreSQL database as backend and Apache as web server. It is provided in many formats suitable for any form of virtualization, cloud or hardware installation.

It comes with standard set of passwords you will want to change:

Username	Password	Scope	Description
root	linux	System	Administrator account, use for local or SSH login
weblate	weblate	PostgreSQL	Account in PostgreSQL database for storing Weblate data
admin	admin	Weblate	Weblate/Django admin user

The appliance is built using SUSE Studio and is based on openSUSE 42.1.

You should also adjust some settings to match your environment, namely:

- *Disable debug mode*
- *Set correct site name*
- *Configure email addresses*

Upgrading Weblate

Generic upgrade instructions

Before upgrading, please check 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 smooth migration. Upgrading across multiple releases should work, but is not that well tested as single version upgrades.

Note: It is recommended to perform database backup prior to upgrade so that you can roll back the database in case upgrade fails.

To upgrade database structure, you should run:

```
./manage.py migrate --noinput
```

To collect new static files, run:

```
./manage.py collectstatic --noinput
```

To upgrade default set of privileges definitions (optional), run:

```
./manage.py setupgroups
```

To upgrade default set of language definitions (optional), run:

```
./manage.py setuplang
```

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
```

Changed in version 1.2: Since version 1.2 the migration is done using South module, to upgrade to 1.2, please see *Version specific instructions*.

Changed in version 1.9: Since version 1.9, Weblate also supports Django 1.7 migrations, please check *Upgrading to Django 1.7* for more information.

Changed in version 2.3: Since version 2.3, Weblate supports only Django native migrations, South is no longer supported, please check *Upgrading to Django 1.7* for more information.

Changed in version 2.11: Since version 2.11, there is reduced support for migrating from older not released versions. In case you hit problem in this, please upgrade first to closest released version and then continue in upgrading to latest one.

Changed in version 2.12: Since version 2.12 upgrade is not supported for versions prior to 2.2. In case you are upgrading from such old version, please upgrade to 2.2 first and then continue in upgrading to current release.

Version specific instructions

Upgrade from 0.5 to 0.6

On upgrade to version 0.6 you should run `./manage.py syncdb` and `./manage.py setupgroups --move` to setup access control as described in installation section.

Upgrade from 0.6 to 0.7

On upgrade to version 0.7 you should run `./manage.py syncdb` to setup new tables and `./manage.py rebuild_index` to build index for fulltext search.

Upgrade from 0.7 to 0.8

On upgrade to version 0.8 you should run `./manage.py syncdb` to setup new tables, `./manage.py setupgroups` to update privileges setup and `./manage.py rebuild_index` to rebuild index for fulltext search.

Upgrade from 0.8 to 0.9

On upgrade to version 0.9 file structure has changed. You need to move `repos` and `whoosh-index` to `weblate` folder. Also running `./manage.py syncdb`, `./manage.py setupgroups` and `./manage.py setuplang` is recommended to get latest updates of privileges and language definitions.

Upgrade from 0.9 to 1.0

On upgrade to version 1.0 one field has been added to database, you need to invoke following SQL command to adjust it:

```
ALTER TABLE `trans_subproject` ADD `template` VARCHAR(200);
```

Upgrade from 1.0 (1.1) to 1.2

On upgrade to version 1.2, the migration procedure has changed. It now uses South for migrating database. To switch to this new migration schema, you need to run following commands:

```
./manage.py syncdb
./manage.py migrate trans 0001 --fake
./manage.py migrate accounts 0001 --fake
./manage.py migrate lang 0001 --fake
```

Also please note that there are several new requirements and version 0.8 of django-registration is now being required, see *Software requirements* for more details.

Once you have done this, you can use *Generic upgrade instructions*.

Upgrade from 1.2 to 1.3

Since 1.3, `settings.py` is not shipped with Weblate, but only example settings as `settings_example.py` it is recommended to use it as new base for your setup.

Upgrade from 1.4 to 1.5

Several internal modules and paths have been renamed and changed, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

- Many modules lost their `weblate.` prefix.
- Checks were moved to submodules.
- Locales were moved to top level directory.

The migration of database structure to 1.5 might take quite long, it is recommended to put your site offline, while the migration is going on.

Note: If you have update in same directory, stale `*.pyc` files might be left around and cause various import errors. To recover from this, delete all of them in Weblate's directory, for example by `find . -name '*.pyc' -delete`.

Upgrade from 1.6 to 1.7

The migration of database structure to 1.7 might take quite long, it is recommended to put your site offline, while the migration is going on.

If you are translating monolingual files, it is recommended to rerun quality checks as they might have been wrongly linked to units in previous versions.

Upgrade from 1.7 to 1.8

The migration of database structure to 1.8 might take quite long, it is recommended to put your site offline, while the migration is going on.

Authentication setup has been changed and some internal modules have changed name, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

Also please note that there are several new requirements, see *Software requirements* for more details.

Upgrade from 1.8 to 1.9

Several internal modules and paths have been renamed and changed, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

See also:

If you are upgrading to Django 1.7 in same step, please consult *Upgrading to Django 1.7*.

Upgrade from 1.9 to 2.0

Several internal modules and paths have been renamed and changed, please adjust your `settings.py` to match that (consult `settings_example.py` for correct values).

This upgrade also requires you to upgrade python-social-auth from 0.1.x to 0.2.x series, what will most likely to need to fake one of their migrations (see Upgrading PSA with South for more information):

```
./manage.py migrate --fake default
```

See also:

If you are upgrading to Django 1.7 in same step, please consult *Upgrading to Django 1.7*.

Upgrade from 2.0 to 2.1

The filesystem paths configuration has changed, the *GIT_ROOT* and *WHOOSH_INDEX* are gone and now all data resides in *DATA_DIR*. The existing data should be automatically migrated by supplied migration, but in case of non standard setup, you might need to move these manually.

See also:

If you are upgrading to Django 1.7 in same step, please consult *Upgrading to Django 1.7*.

Upgrade from 2.1 to 2.2

Weblate now supports fulltext search on additional fields. In order to make it work on existing data you need to update fulltext index by:

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

If you have some monolingual translations, Weblate now allows to edit template (source) strings as well. To see them, you need to reload translations, what will either happen automatically on next repository update or you can force it manually:

```
./manage.py loadpo --all
```

See also:

If you are upgrading to Django 1.7 in same step, please consult *Upgrading to Django 1.7*.

Upgrade from 2.2 to 2.3

If you have not yet performed upgrade to Django 1.7 and newer, first upgrade to 2.2 following instructions above. Weblate 2.3 no longer supports migration from Django 1.6.

If you were using Weblate 2.2 with Django 1.6, you will now need to fake some migrations:

```
./manage.py migrate --fake accounts 0004_auto_20150108_1424
./manage.py migrate --fake lang 0001_initial
./manage.py migrate --fake trans 0018_auto_20150213_1447
```

Previous Weblate releases contained bug which made some monolingual translations behave inconsistently for fuzzy and not translated strings, if you have such, it is recommended to run:

```
./manage.py fixup_flags --all
```

See also:

Generic upgrade instructions

Upgrade from 2.3 to 2.4

Handling of static content has been rewritten, please adjust configuration of your webserver accordingly (see *Serving static files* for more details). Most importantly:

- `/media/` path is no longer used
- `/static/` path now holds both admin and Weblate static files

There is now also additional dependency - `django_compressor`, please install it prior to upgrading.

See also:

Generic upgrade instructions

Upgrade from 2.4 to 2.5

The fulltext index has been changed, so unless you rebuild it, the fulltext search will not work. To rebuild it, execute:

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

See also:

Generic upgrade instructions

Upgrade from 2.5 to 2.6

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- new dependecy on Django REST Framework, see *Software requirements*
- example configuration now configures Django REST Framework, please adjust your settings accordingly

- the USE_TZ settings is now enabled by default

Note: Weblate now much more relies on correct site name in the database, please see *Set correct site name* for instructions how to set it up.

See also:

Generic upgrade instructions

Upgrade from 2.6 to 2.7

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- new optional dependency on python-bidi, see *Software requirements*
- Google Web Translation was removed, remove it from your configuration

See also:

Generic upgrade instructions

Upgrade from 2.7 to 2.8

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- new dependency on defusedxml, see *Software requirements*
- there is new quality check: *Invalid XML markup*

See also:

Generic upgrade instructions

Upgrade from 2.8 to 2.9

Please follow generic upgrade instructions, the only notable change is addition of media storage to *DATA_DIR*.

See also:

Generic upgrade instructions

Upgrade from 2.9 to 2.10

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- The `INSTALLED_APPS` now should include `weblate.utils`.
- There is new check in default set (`SamePluralsCheck`).
- There is change in `SOCIAL_AUTH_PIPELINE` default settings.
- You might want to enable optional *Git exporter*.
- There is new `RemoveControlChars` in default *AUTOFIX_LIST*.

- If you are using Microsoft Translator, please replace *Microsoft Translator* with *Microsoft Cognitive Services Translator*, Microsoft has changed authentication scheme.

See also:

Generic upgrade instructions

Upgrade from 2.10 to 2.11

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- There is new recommended value for `SOCIAL_AUTH_SLUGIFY_FUNCTION`.
- There is change in `MIDDLEWARE_CLASSES` setting.
- The `python-social-auth` module has been deprecated upstream, Weblate now uses `social-auth-core` and `social-auth-app-django` instead. You also have to adjust `settings.py` as several modules have been moved from `social` to either `social_core` or `social_django`. Please consult `settings_example.py` for correct values.

Warning: If you were using python-social-auth 0.2.19 or older with Weblate 2.10, you should first upgrade Weblate 2.10 to python-social-auth 0.2.21 and then perform upgrade to Weblate 2.11. Otherwise you end up with non applicable database migrations.

See Migrating from python-social-auth to split social for more information.

If you are upgrading from older version, you should first upgrade to Weblate 2.10 and python-social-auth 0.2.21 and then continue in upgrading.

See also:

Generic upgrade instructions

Upgrade from 2.11 to 2.12

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- The database migration will take quite long on this update as all translation units stored in database have to be updated. Expect about 1 hour of migration for 500000 translation units (depends on hardware and database).
- There is new dependency on `django-appconf` and `siphashc3`.
- The setting for `UNAUTHENTICATED_USER` for `REST_FRAMEWORK` has been changed to properly handle anonymous user permissions in REST API.
- The `INSTALLED_APPS` now should include `weblate.screenshots`.
- There is new optional dependency on tesserocr, see *Software requirements*.

See also:

Generic upgrade instructions

Upgrade from 2.12 to 2.13

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- There is new quality check: *Has been translated*.
- The `INSTALLED_APPS` now should include `weblate.permissions`.
- The per project ALCs are now implemented using Group ACL, you might need to adjust your setup if you were using Group ACLs before, see *Group-based access control* for more information about the setup.
- There are several new permissions which should be assigned to default groups, you should run `./manage.py setupgroups` to update them. Alternatively you might want to add following permissions where applicable (see *Extra privileges* for their default setup):
  - Can access VCS repository
  - Can access project

Note: If you have update in same directory, stale `*.pyc` files might be left around and cause various import errors. To recover from this, delete all of them in Weblate's directory, for example by `find . -name '*.pyc' -delete`.

See also:

Generic upgrade instructions

Upgrade from 2.13 to 2.14

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- There is new middleware `weblate.middleware.SecurityMiddleware` in the default configuration, see *Content security policy* for more details.
- Weblate now uses Django password validation, it's controlled by `AUTH_PASSWORD_VALIDATORS` setting.
- Weblate now customizes disconnect pipeline for Python Social Auth, the `SOCIAL_AUTH_DISCONNECT_PIPELINE` setting is now needed.
- There is change in `SOCIAL_AUTH_PIPELINE` default settings.
- All pending email verifications will be invalid due to validation change.
- The authentication attempts are now rate limited, see *Rate limiting* for more details.

See also:

Generic upgrade instructions

Upgrade from 2.14 to 2.15

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- The `AUTHENTICATION_BACKENDS` setting should be changed to include `social_core.backends.email.EmailAuth` as shipped by Python Social Auth. Weblate no longer uses own email auth backend.

See also:

Generic upgrade instructions

Upgrade from 2.15 to 2.16

Follow generic upgrade instructions, there is no special change.

Notable configuration or dependencies changes:

- There is change in `SOCIAL_AUTH_PIPELINE` default settings.
- The `weblate.wladmin` should now be first in the `INSTALLED_APPS` settings.

See also:

Generic upgrade instructions

Upgrading to Django 1.7

Changed in version 2.3: This migration is supported only in Weblate 2.2, in case you are upgrading from some older version, you will have to do intermediate update to 2.2.

Django 1.7 has a new feature to handle database schema upgrade called "migrations" which is incompatible with South (used before by Weblate).

Before migrating to Django 1.7, you first need to apply all migrations from South. If you already have upgraded Django to 1.7, you can do this using virtualenv and `examples/migrate-south` script:

```
examples/migrate-south --settings weblate.settings
```

Once you have done that, you can run Django migrations and work as usual. For the initial setup, you might need to fake some of the migrations though:

```
./manage.py migrate --fake-initial
```

Upgrading from Python 2.x to 3.x

The upgrade from Python 2.x to 3.x, should work without major problems. Take care about some changed module names when installing dependencies (eg. pydns vs. py3dns).

The Whoosh index has to be rebuilt as it's encoding depends on Python version, you can do that using following command:

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

Migrating from Pootle

As Weblate was originally written as replacement from Pootle, it is supported to migrate user accounts from Pootle. All you need to do is to copy `auth_user` table from Pootle, user profiles will be automatically created for users as they log in and they will be asked to update their settings. Alternatively you can use *importusers* to import dumped user credentials.

Authentication

User registration

The default setup for Weblate is to use python-social-auth for handling new users. This allows them to register using form on the website and after confirming their email they can contribute or by using some third party service to authenticate.

You can also completely disable new users registration using *REGISTRATION_OPEN*.

Rate limiting

New in version 2.14.

The password based authentication is subject to rate limiting. At most *AUTH_MAX_ATTEMPTS* attempts are allowed within *AUTH_CHECK_WINDOW* seconds. The user is then blocked for *AUTH_LOCKOUT_TIME*.

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.

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 case Weblate is running behind reverse proxy to work it properly.

See also:

IP_BEHIND_REVERSE_PROXY, *IP_PROXY_HEADER*, *IP_PROXY_OFFSET*

Authentication backends

By default Weblate uses Django built-in authentication and includes various social authentication options. Thanks to using Django authentication, you can also import user database from other Django based projects (see *Migrating from Pootle*).

Django can be additionally configured to authenticate against other means as well.

Social authentication

Thanks to python-social-auth, Weblate support authentication using many third party services such as Facebook, GitHub, Google or Bitbucket.

Please check their documentation for generic configuration instructions in Django Framework.

Note: By default, Weblate relies on third-party authentication services to provide validated email address, in case some of services you want to use do not support this, please remove `social_core.pipeline.social_auth.associate_by_email` from `SOCIAL_AUTH_PIPELINE` settings.

Enabling individual backends is quite easy, it's just a matter of adding entry to `AUTHENTICATION_BACKENDS` setting and possibly adding keys needed for given authentication. 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 users contributions.

OpenID authentication

For OpenID based services it's usually just a matter of enabling them. Following section enables OpenID authentication for OpenSUSE, Fedora and Ubuntu:

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.email.EmailAuth',
    'social_core.backends.suse.OpenSUSEOpenId',
    'social_core.backends.ubuntu.UbuntuOpenId',
    'social_core.backends.fedora.FedoraOpenId',
    'weblate.accounts.auth.WeblateUserBackend',
)
```

GitHub authentication

You need to register application on GitHub and then tell Weblate all the 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']
```

See also:

Python Social Auth backend

Bitbucket authentication

You need to register application on Bitbucket and then tell Weblate all the 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:

Python Social Auth backend

Google OAuth2

For using Google OAuth2, you need to register application on <https://console.developers.google.com/> and enable 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'
```

Facebook OAuth2

As usual with OAuth2 services, you need to register your application with Facebook. Once this is done, you can configure 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']
```

Gitlab OAuth2

For using Gitlab OAuth2, you need to register application on <https://gitlab.com/profile/applications>.

The redirect URL is `https://WEBLATE SERVER/accounts/complete/gitlab/` and ensure to mark the *read_user* scope.

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    'social_core.backends.gitlab.GitLabOAuth2',
    'social_core.backends.email.EmailAuth',
    'weblate.accounts.auth.WeblateUserBackend',
)

# Social auth backends setup
SOCIAL_AUTH_GITLAB_KEY = 'Application ID'
SOCIAL_AUTH_GITLAB_SECRET = 'Secret'
SOCIAL_AUTH_GITLAB_SCOPE = ['api']
```

LDAP authentication

LDAP authentication can be best achieved using *django-auth-ldap* package. You can install it by usual means:

```
# Using PyPI
pip install django-auth-ldap

# Using apt-get
apt-get install python-django-auth-ldap
```

Once you have the package installed, you can hook it to 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 different DN
# like:
# AUTH_LDAP_USER_DN_TEMPLATE = 'ou=users,dc=example,dc=com'

# List of attributes to import from LDAP on login
# Weblate stores full user name in the first_name attribute
AUTH_LDAP_USER_ATTR_MAP = {
    'first_name': 'name',
    'email': 'mail',
}
```

See also:

Django Authentication Using LDAP

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 Django one if you want to be able to login
# even without LDAP for 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 inside your 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 higher)
- at the end of your `models.py` file (Django 1.6 and lower)
- 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
    # 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

Access control

Weblate uses privileges system based on Django, but is extended in several ways to allow managing access at more fine grained level. See *Per project access control* and *Group-based access control* for more detailed information on those extensions.

The default setup (after you run `setupgroups`) consists of three groups *Guests*, *Users* and *Managers* which have privileges as described above. All new users are automatically added to *Users* group (thanks to *Automatic group assignments*). The *Guests* groups is used for not logged in users.

To customize this setup, it is recommended to remove privileges from *Users* group and create additional groups with finer privileges (eg. *Translators* group, which will be allowed to save translations and manage suggestions) and add selected users to this group. You can do all this from Django admin interface.

To completely lock down your Weblate installation you can use `LOGIN_REQUIRED_URLS` for forcing users to login and `REGISTRATION_OPEN` for disallowing new registrations.

Warning: Never remove Weblate predefined groups (*Guests*, *Users* and *Managers*). If you do not want to use these features, just remove all privileges from them.

Extra privileges

Weblate defines following extra privileges:

Can upload translation [Users, Managers] Uploading of translation files.

Can overwrite with translation upload [Users, Managers] Overwriting existing translations by uploading translation file.

Can define author of translation upload [Managers] Allows to define custom authorship when uploading translation file.

Can force committing of translation [Managers] Can force VCS commit in the web interface.

Can see VCS repository URL [Users, Managers, Guests] Can see VCS repository URL inside Weblate

Can update translation from VCS [Managers] Can force VCS pull in the web interface.

Can push translations to remote VCS [Managers] Can force VCS push in the web interface.

Can do automatic translation using other project strings [Managers] Can do automatic translation based on strings from other components

Can lock whole translation project [Managers] Can lock translation for updates, useful while doing some major changes in the project.

Can reset translations to match remote VCS [Managers] Can reset VCS repository to match remote VCS.

Can access VCS repository [Users, Managers, Guests] Can access the underlying VCS repository (see *Git exporter*).

Can save translation [Users, Managers] Can save translation (might be disabled with *Suggestion voting*).

Can save template [Users, Managers] Can edit source strings (usually English)

Can accept suggestion [Users, Managers] Can accept suggestion (might be disabled with *Suggestion voting*).

Can delete suggestion [Users, Managers] Can delete suggestion (might be disabled with *Suggestion voting*).

Can delete comment [Managers] Can delete comment.

Can vote for suggestion [Users, Managers] Can vote for suggestion (see *Suggestion voting*).

Can override suggestion state [Managers] Can save translation, accept or delete suggestion when automatic accepting by voting for suggestions is enabled (see *Suggestion voting*).

Can import dictionary [Users, Managers] Can import dictionary from translation file.

Can add dictionary [Users, Managers] Can add dictionary entries.

Can change dictionary [Users, Managers] Can change dictionary entries.

Can delete dictionary [Users, Managers] Can delete dictionary entries.

Can lock translation for translating [Users, Managers] Can lock translation while translating (see *Translation locking*).

Can add suggestion [Users, Managers, Guests] Can add new suggestions.

Can use machine translation [Users, Managers] Can use machine translations (see *Machine translation*).

Can manage ACL rules for a project [Managers] Can add users to ACL controlled projects (see *Per project access control*)

Can access project [Users, Managers, Guests] Can access project (see *Per project access control*)

Can edit priority [Managers] Can adjust source string priority

Can edit check flags [Managers] Can adjust source string check flags

Can download changes [Managers] Can download changes in a CSV format.

Can display reports [Managers] Can display detailed translation reports.

Can add translation [Users, Managers] Can start translations in new language.

Can mass add translation [Managers] Can start translations in several languages at once.

Can delete translation [Managers] Can remove translation.

Can change sub project [Managers] Can edit component settings.

Can change project [Managers] Can edit project settings.

Can upload screenshot [Managers] Can upload source string screenshot context.

Per project access control

New in version 1.4: This feature is available since Weblate 1.4.

Changed in version 2.13: Since Weblate 2.13 the per project access control uses *Group-based access control* under the hood. You might need some adjustments to your setup if you were using both features.

Note: By enabling ACL, all users are prohibited to access anything within given project unless you add them the permission to do that.

Additionally you can limit users access to individual projects. This feature is enabled by *Enable ACL* at Project configuration. This automatically creates *Group-based access control* for this project

To allow access to this project, you have to add the privilege to do so either directly to given user or group of users in Django admin interface. Or using user management on project page as described in *Managing per project access control*.

See also:

Managing users in the admin

Note: Even with ACL enabled some summary information will be available about your project:

- Site wide statistics includes counts for all projects
- Site wide languages summary includes counts for all projects

---

Automatic group assignments

New in version 2.5.

You can configure Weblate to automatically add users to groups based on their email. This automatic assignment happens only at time of account creation.

This can be configured in the Django admin interface (in the *Accounts* section).

Group-based access control

New in version 2.5: This feature is available since Weblate 2.5.

You can designate groups that have exclusive access to a particular language, project or component, or a combination thereof. This feature is also used to implement *Per project access control* by automatically created groups for each project. For example, you can use this feature to designate a language-specific translator team with full privileges for their own language.

This works by "locking" given permission for the group(s) in question to the object, the effect of which is twofold.

Firstly, groups that are locked for some object are the *only* groups that have given privileges on that object. If a user is not a member of the locked group, they cannot edit the object, even if their privileges or group membership allows them to edit other (unlocked) objects.

Secondly, privileges of the locked group don't apply on objects other than those to which the group is locked. If a user is a member of the locked group which grants them edit privileges, they can only edit the object locked to the group, unless something else grants them a general edit privilege.

This can be configured in the Django admin interface. The recommended workflow is as follows:

1. Create a new *group ACL* in the *Group ACL* section. Pick a project, subproject, language, or a combination, which will be locked to this group ACL.
2. Define permissions you want to limit by this *group ACL*.
3. Use the + (plus sign) button to the right of *Groups* field to create a new group. In the pop-up window, fill out the group name and assign permissions.
4. Save the newly created group ACL.
5. In the *Users* section of the admin interface, assign users to the newly created group.

For example, you could create a group called `czech_translators`, assign it full privileges, and lock it to Czech language. From that point on, all users in this groups would get full privileges for the Czech language in all projects and components, but not for any other languages. Also, users who are not members of the `czech_translators` group would get no privileges on Czech language in any project.

In order to delete a group ACL, make sure that you first delete the group (or remove its privileges), and only then delete the group ACL. Otherwise, there will be a window of time in which the group is "unlocked" and its permissions apply to all objects. In our example, members of `czech_translators` group would have full privileges for everything that is not locked to other groups.

It is possible to lock multiple groups within a single group ACL. One group can also be locked to multiple objects through multiple group ACLs. As long as a group is recorded in at least one group ACL, it's considered to be "locked", and its privileges do not apply outside the locks.

Group ACLs apply in order of specificity. "Component" is considered most specific, "Language" is least specific. Combinations follow the most specific part of the combination: a group ACL that is locked to a particular component is more specific than a group ACL locked to this component's project and a particular language. That means that members of the component-specific groups will have privileges on the component, and members of the project-and-language-specific groups will not. The latter will, of course, have privileges on their language in all other components of the project.

For project-level actions (such as pushing upstream, setting priority, etc.), you must create a group ACL locked to *only* the project. Combinations, such as project plus language, only apply to actions on individual translations.

Managing users and groups

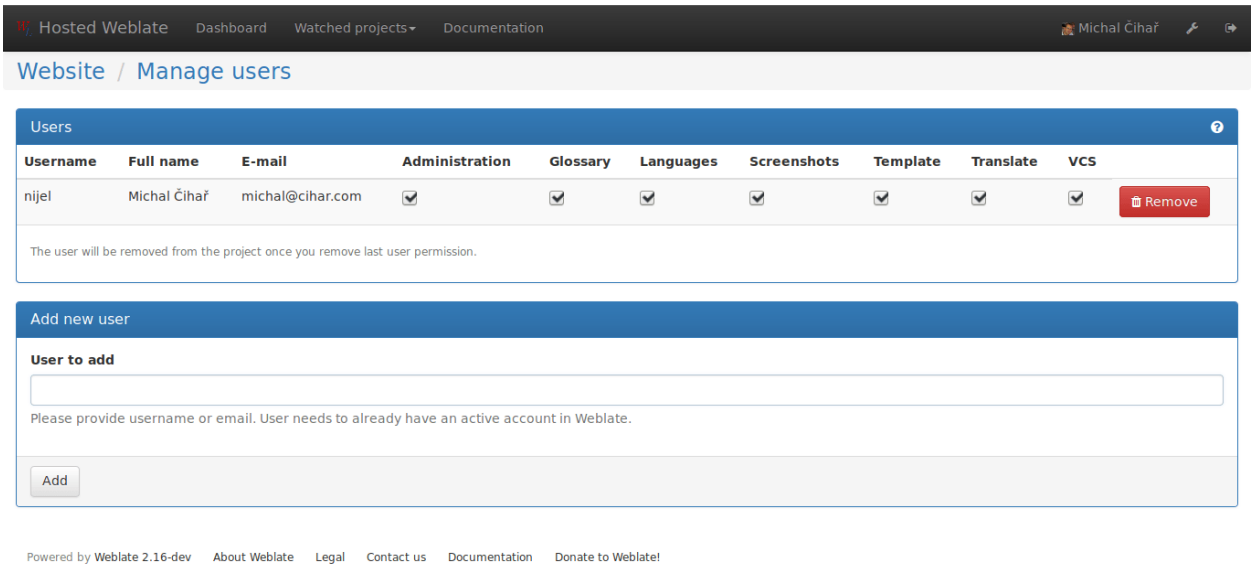
All users and groups can be managed using Django admin interface, which is available under `/admin/` URL.

Managing per project access control

Note: This feature only works for ACL controlled projects, see *Per project access control*.

Users with *Can manage ACL rules for a project* privilege (see *Access control*) can also manage users in projects with access control enabled on the project page. You can add or remove users to the project or make them owners.

The user management is available in *Tools* menu of a project:



See also:

Per project access control

Predefined groups

Weblate comes with predefined set of groups where you can assign users.

`Administration`
: Has all permissions on the project.

`Glossary`
: Can manage glossary (add or remove entries or upload glossary).

`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 template in *Monolingual components*.

`Translate`
: Can translate project, including upload of offline translatoins.

`VCS`
: Can manage VCS and access exported repository.

Translation projects

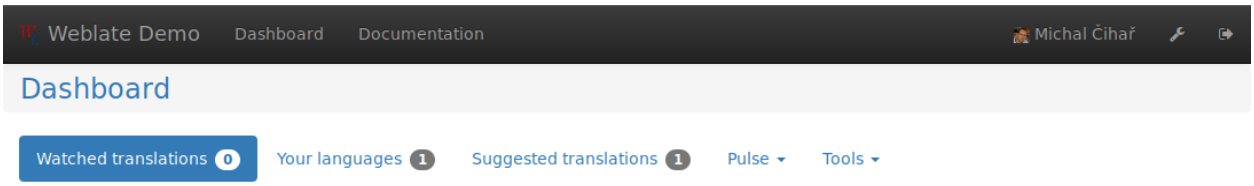
Translation organization

Weblate organizes translatable content into tree like structure. The toplevel object is *Project configuration*, which should hold all translations which belong together (for example translation of an application in several versions and/or documentation). On the next level, there is *Component configuration*, which is actually the component to translate. Here you define VCS repository to use and mask of files to translate. Bellow *Component configuration* there are individual translations, which are handled automatically by Weblate as the translation files (matching mask defined in *Component configuration*) appear in VCS repository.

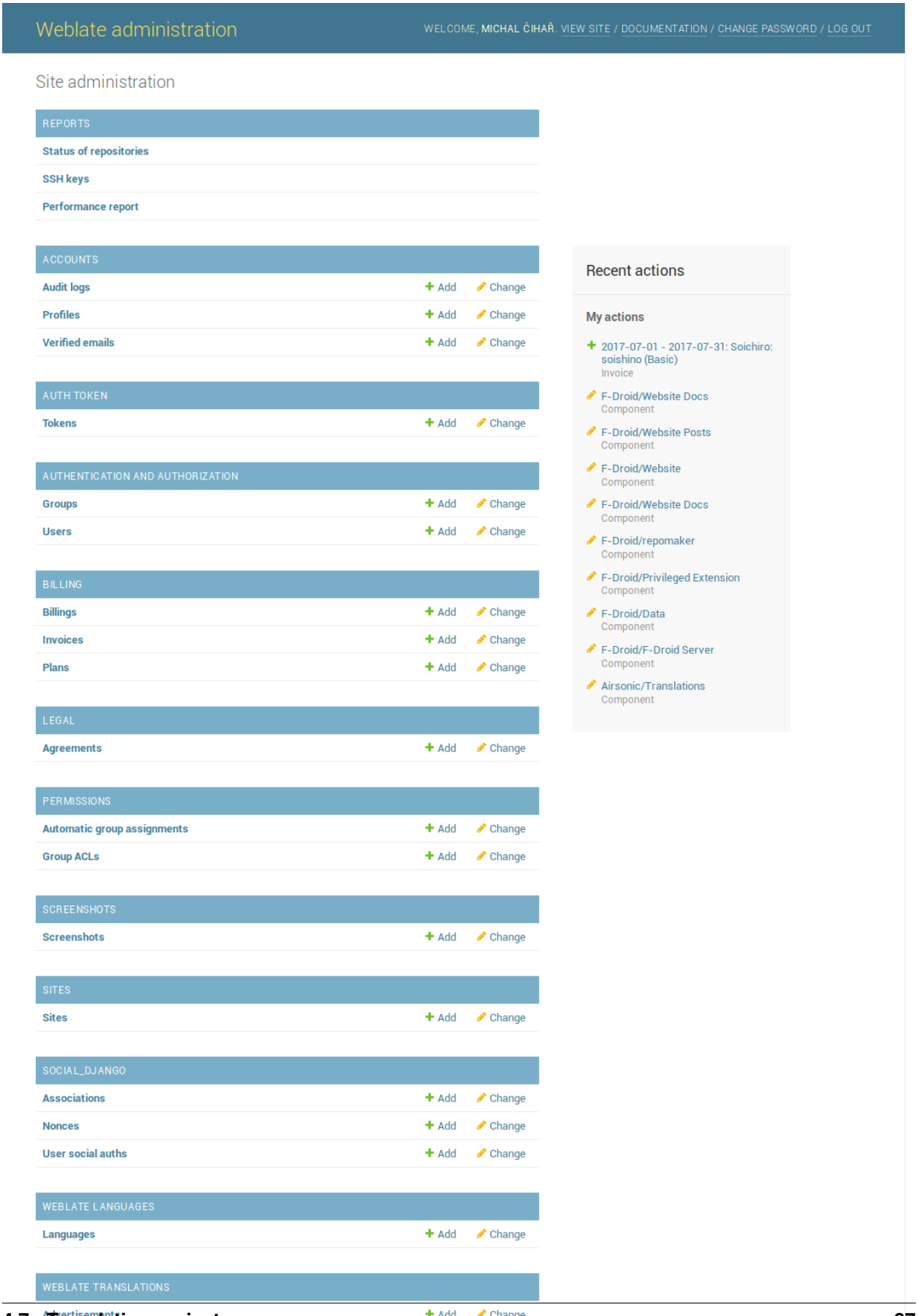
Note: You can share cloned VCS repositories using *Weblate internal URLs*. Using shared repositories feature is highly recommended when you have many components that use the same VCS. It will improve performance and use less disk space.

Administration

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 wrench icon in top navigation:



Here you can manage objects stored in the database, such as users, translations and other settings:


Weblate administration
WELCOME, MICHAL ČIHAŘ. VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT
Site administration
REPORTS
Status of repositories
SSH keys
Performance report
ACCOUNTS
Audit logs Add Change
Profiles Add Change
Verified emails Add Change
AUTH TOKEN
Tokens Add Change
AUTHENTICATION AND AUTHORIZATION
Groups Add Change
Users Add Change
BILLING
Billings Add Change
Invoices Add Change
Plans Add Change
LEGAL
Agreements Add Change
PERMISSIONS
Automatic group assignments Add Change
Group ACLs Add Change
SCREENSHOTS
Screenshots Add Change
SITES
Sites Add Change
SOCIAL_DJANGO
Associations Add Change
Nonces Add Change
User social auths Add Change
WEBLATE LANGUAGES
Languages Add Change
WEBLATE TRANSLATIONS
Advertisements Add Change
Component lists Add Change
Components Add Change
Projects Add Change
Recent actions
My actions
2017-07-01 - 2017-07-31: Soichiro: soishino (Basic)
Invoice
F-Droid/Website Docs
Component
F-Droid/Website Posts
Component
F-Droid/Website
Component
F-Droid/Website Docs
Component
F-Droid/repomaker
Component
F-Droid/Privileged Extension
Component
F-Droid/Data
Component
F-Droid/F-Droid Server
Component
Airsonic/Translations
Component

In the *Reports* section you can check status of your site, tweak it for *Production setup* or manage SSH keys to access *Accessing repositories*.

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 defintions, see *Language definitions* for more details.

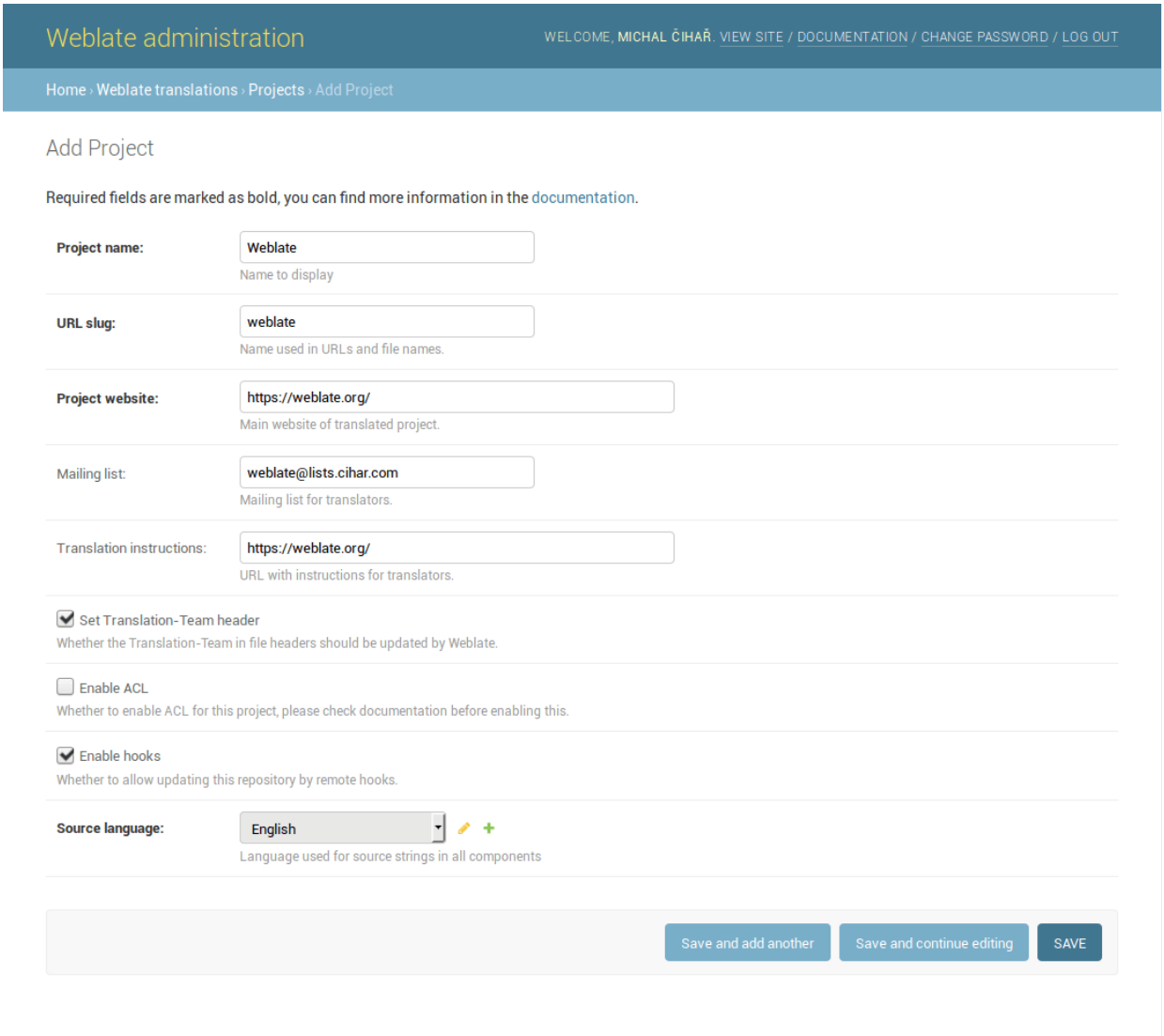
Adding new components

All translation components need to be available as VCS repositories and are organized as project/component structure.

Weblate supports wide range of translation formats (both bilingual and monolingua) supported by translate toolkit, see *Supported formats* for more information.

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):

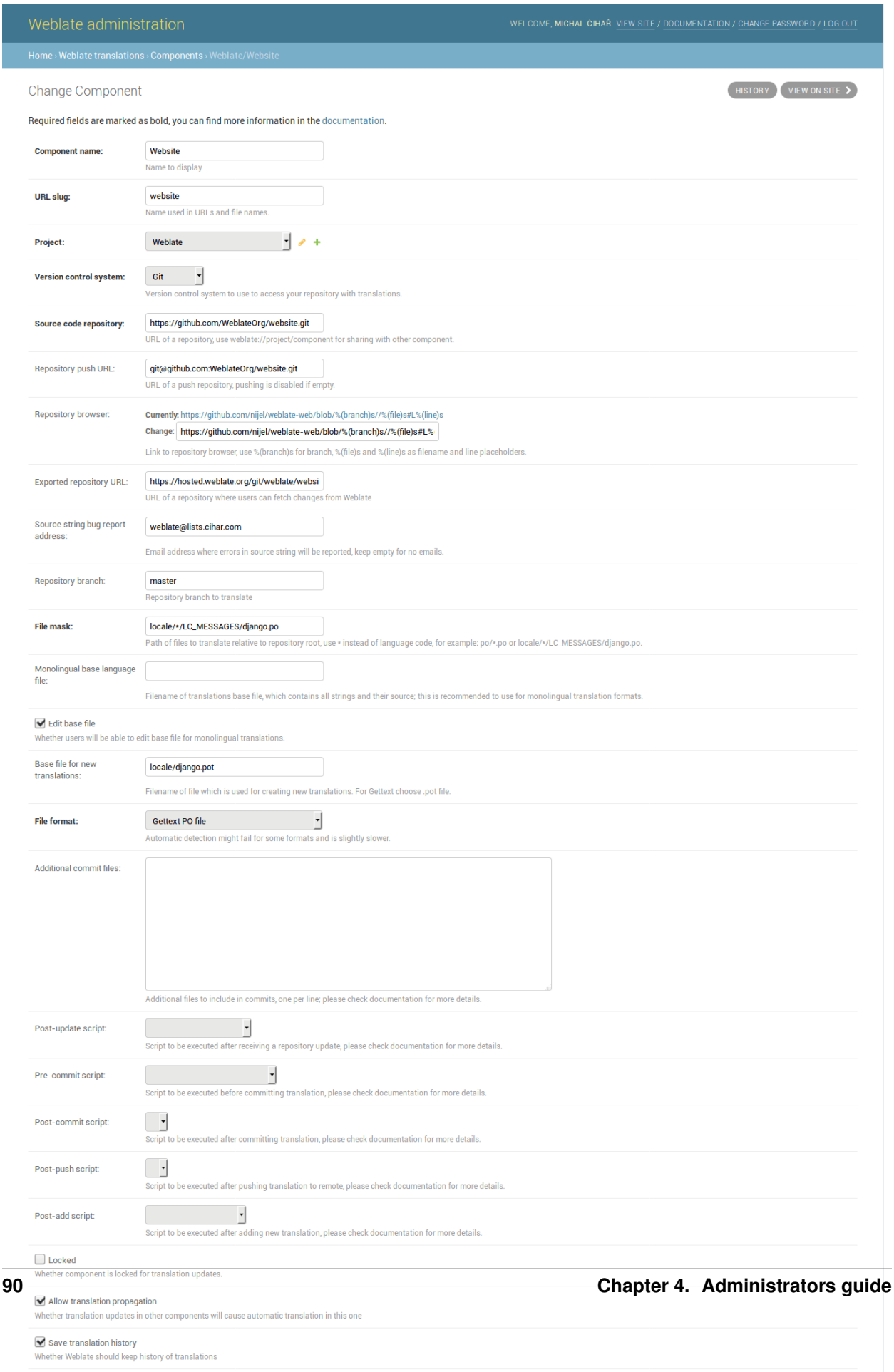


See also:

Project configuration

Bilingual components

Once you have added a project, you can add translation compontents to it (see *Component configuration* for information on individual parameters):

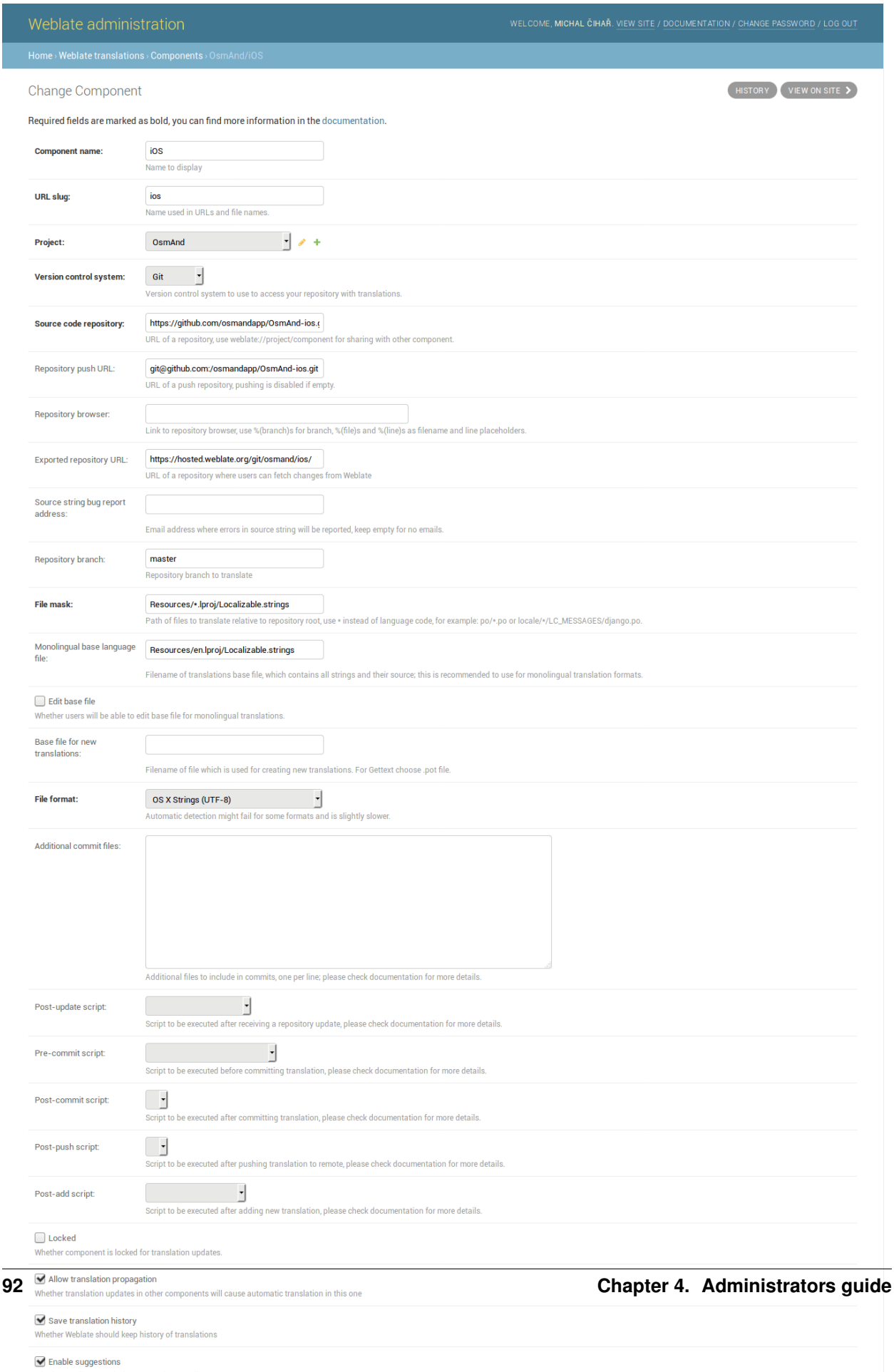

Weblate administration
WELCOME, MICHAL ČIHAŘ. VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT
Home › Weblate translations › Components › Weblate/Website
Change Component
HISTORY
VIEW ON SITE
Required fields are marked as bold, you can find more information in the documentation.
Component name:
Website
Name to display
URL slug:
website
Name used in URLs and file names.
Project:
Weblate
Version control system:
Git
Version control system to use to access your repository with translations.
Source code repository:
https://github.com/WeblateOrg/website.git
URL of a repository, use weblate://project/component for sharing with other component.
Repository push URL:
git@github.com:WeblateOrg/website.git
URL of a push repository, pushing is disabled if empty.
Repository browser:
Currently: https://github.com/nijel/weblate-web/blob/%(branch)s//%(file)s#L%(line)s
Change:
https://github.com/nijel/weblate-web/blob/%(branch)s//%(file)s#L%
Link to repository browser, use %(branch)s for branch, %(file)s and %(line)s as filename and line placeholders.
Exported repository URL:
https://hosted.weblate.org/git/weblate/websi
URL of a repository where users can fetch changes from Weblate
Source string bug report address:
weblate@lists.cihar.com
Email address where errors in source string will be reported, keep empty for no emails.
Repository branch:
master
Repository branch to translate
File mask:
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 translations base file, which contains all strings and their source; this is recommended to use for monolingual translation formats.
Edit base file
Whether users will be able to edit base file for monolingual translations.
Base file for new translations:
locale/django.pot
Filename of file which is used for creating new translations. For Gettext choose .pot file.
File format:
Gettext PO file
Automatic detection might fail for some formats and is slightly slower.
Additional commit files:
Additional files to include in commits, one per line; please check documentation for more details.
Post-update script:
Script to be executed after receiving a repository update, please check documentation for more details.
Pre-commit script:
Script to be executed before committing translation, please check documentation for more details.
Post-commit script:
Script to be executed after committing translation, please check documentation for more details.
Post-push script:
Script to be executed after pushing translation to remote, please check documentation for more details.
Post-add script:
Script to be executed after adding new translation, please check documentation for more details.
Locked
Whether component is locked for 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 history of translations

See also:

Component configuration

Monolingual components

For easier translating of monolingual formats, you should provide template file, which contains mapping of message IDs to source language (usually English) (see *Component configuration* for information on individual parameters):


Weblate administration
WELCOME, MICHAL ČIHAŘ. VIEW SITE / DOCUMENTATION / CHANGE PASSWORD / LOG OUT
Home › Weblate translations › Components › OsmAnd/iOS
Change Component
HISTORY
VIEW ON SITE
Required fields are marked as bold, you can find more information in the documentation.
Component name:
iOS
Name to display
URL slug:
ios
Name used in URLs and file names.
Project:
OsmAnd
Version control system:
Git
Version control system to use to access your repository with translations.
Source code repository:
https://github.com/osmandapp/OsmAnd-ios.g
URL of a repository, use weblate://project/component for sharing with other component.
Repository push URL:
git@github.com:/osmandapp/OsmAnd-ios.git
URL of a push repository, pushing is disabled if empty.
Repository browser:
Link to repository browser, use %(branch)s for branch, %(file)s and %(line)s as filename and line placeholders.
Exported repository URL:
https://hosted.weblate.org/git/osmand/ios/
URL of a repository where users can fetch changes from Weblate
Source string bug report address:
Email address where errors in source string will be reported, keep empty for no emails.
Repository branch:
master
Repository branch to translate
File mask:
Resources/*.lproj/Localizable.strings
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:
Resources/en.lproj/Localizable.strings
Filename of translations base file, which contains all strings and their source; this is recommended to use for monolingual translation formats.
Edit base file
Whether users will be able to edit base file for monolingual translations.
Base file for new translations:
Filename of file which is used for creating new translations. For Gettext choose .pot file.
File format:
OS X Strings (UTF-8)
Automatic detection might fail for some formats and is slightly slower.
Additional commit files:
Additional files to include in commits, one per line; please check documentation for more details.
Post-update script:
Script to be executed after receiving a repository update, please check documentation for more details.
Pre-commit script:
Script to be executed before committing translation, please check documentation for more details.
Post-commit script:
Script to be executed after committing translation, please check documentation for more details.
Post-push script:
Script to be executed after pushing translation to remote, please check documentation for more details.
Post-add script:
Script to be executed after adding new translation, please check documentation for more details.
Locked
Whether component is locked for 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 history of translations
Enable suggestions

See also:

Component configuration

Project configuration

To add new component to translate, you need to create translation project first. The project is sort of shelf, in which real translations are folded. All components in same project share suggestions and dictionary, also the translations are automatically propagated through the all component in single project (unless disabled in component configuration).

The project has only few attributes giving translators information about project:

Project website URL where translators can find more information about the project.

Mailing list Mailing list where translators can discuss or comment translations.

Translation instructions URL where you have more detailed instructions for translators.

Set Translation-Team header Whether Weblate should manage Translation-Team header (this is *GNU Gettext* only feature right now).

Enable ACL Enable per project access control, see *Per project access control* for more details.

Enable hooks Whether unauthenticated *Notification hooks* will be enabled 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 maintaining of Translation-Team header.

Component configuration

Component is real component for translating. You enter VCS repository location and file mask which files to translate and Weblate automatically fetches the VCS and finds all matching translatable files.

Note: It is recommended to have translation components of reasonable size - split the translation by anything what makes sense in your case (individual applications or addons, book chapters or websites).

Weblate handles just fine translations with 10000 of units, but it is harder to split work and coordinate amongh translators with such big translation. Also when one translator is working on a component, this translation is locked for others, see *Translation locking*.

Should the language definition for translation be missing, empty definition is created and named as "cs_CZ (generated)". You should adjust the definition and report this back to Weblate authors so that missing language can be included in next release.

The component contains all important parameters for working with VCS and 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 be either 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 will be disabled when this is empty. See *Accessing repositories* for more details on how to specify repository URL.

Repository browser URL of repository browser to display source files (location where messages are used). When empty no such links will be generated.

You can use following format strings:

- `%(branchs)s` - current branch
- `%(line)s` - line in file
- `%(file)s` - filename
- `%(../file)s` - filename in parent directory
- `%(../../file)s` - filename in grandparent directory

For example on GitHub, you would use something like `https://github.com/WeblateOrg/hello/blob/%(branch)s/%(file)s#L%(line)s`.

Exported repository URL URL where changes made by Weblate are exported. This is important when *Continuous translation* is not used or when there is 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 information how this is processed). In case your repository contains more than one translation files (eg. more Gettext domains), you need to create separate component for each.

For example `po/*.po` or `locale/*/LC_MESSAGES/django.po`.

Monolingual base language file Base file containing strings definition for *Monolingual components*.

Edit base file Whether to allow editing of base file for *Monolingual components*.

Base file for new translations Base file used to generate new translations, eg. `.pot` file with Gettext.

File format Translation file format, see also *Supported 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 disable 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 same string used.

It's usually good idea to disable this for monolingual translations unless you are using same IDs across whole project.

Post-update script One of scripts defined in *POST_UPDATE_SCRIPTS* which is executed after receiving update. This can be used to update the translation files.

Pre-commit script One of scripts defined in *PRE_COMMIT_SCRIPTS* which is executed before commit. This can be used to generate some metadata about translation or to generate binary form of a translation.

Post-commit script One of scripts defined in *POST_COMMIT_SCRIPTS* which is executed after commit. This can be used to notify external parties about the change.

Post-push script One of scripts defined in *POST_PUSH_SCRIPTS* which is executed after push to remote repository. This can be used to generate notify external parties about the change in repository (i.e. create pull request).

Post-add script One of scripts defined in *POST_ADD_SCRIPTS* which is executed when new translation has been added. This can be used to adjust additional files in the repository when adding new translation.

Additional commit files Additional files to include in the commit (separated by newline), usually this one is generated by the pre commit or post add scripts described above.

Supply the `%(language)s` in the path like this: `path/to/addditinal/%(language)s_file.example`

Save translation history Whether to store history of translation changes in database.

Suggestion voting Enable voting for suggestions, see *Suggestion voting*.

Autoaccept suggestions Automatically accept voted suggestions, see *Suggestion voting*.

Quality checks flags Additional flags to pass to quality checks, see *Customizing checks*.

Translation license License of this translation.

License URL URL where users can find full text of a license.

New language How to handle requests for creating new languages. Please note that availability of choices depends on the file format, see *Supported formats*.

Merge style You can configure how the updates from upstream repository are handled. This might not be supported for some VCS. See *Merge or rebase* for more details.

Commit message Message used when committing translation, see *Commit message formatting*.

Committer name Name of commiter used on Weblate commits, the author will be always the real translator. On some VCS this might be not supported. Default value can be changed by *DEFAULT_COMMITER_NAME*.

Committer email Email of commiter used on Weblate commits, the author will be always the real translator. On some VCS this might be not supported. Default value can be changed by *DEFAULT_COMMITER_EMAIL*.

Push on commit Whether any committed changes should be automatically pushed to upstream repository.

Age of changes to commit Configures how old changes (in hours) will be committed by *commit_pending* management command (usually executed by cron). Default value can be changed by *COMMIT_PENDING_HOURS*.

Language filter Regular expression which is used to filter translation when scanning for file mask. This can be used to limit list of languages managed by Weblate (eg. `^(cs|de|es)$` will include only those there 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.

Commit message formatting

The commit message on each commit Weblate does, it can use following format strings in the message:

`%(language)s` Language code

`%(language_name)s` Language name

`%(component)s` Component name

`%(project)s` Project name

`%(url)s` Translation URL

%(total)s Total strings count

%(fuzzy)s Count of strings needing review

%(fuzzy_percent)s Percent of strings needing review

%(translated)s Translated strings count

%(translated_percent)s Translated strings percent

See also:

Does Weblate support other VCS than Git and Mercurial?, *Processing repository with scripts*

Importing speed

Fetching VCS repository and importing translations to Weblate can be lengthy process depending on size of your translations. Here are some tips to improve this situation:

Clone Git repository in advance

You can put in place Git repository which will be used by Weblate. The repositories are stored in `vcs` directory in path defined by *DATA_DIR* in `settings.py` in `<project>/<component>` directories.

This can be especially useful if you already have local clone of this repository and you can use `--reference` option while cloning:

```
git clone \
    --reference /path/to/checkout \
    https://github.com/WeblateOrg/weblate.git \
    weblate/repos/project/component
```

Optimize configuration

The default configuration is useful for testing and debugging Weblate, while for production setup, you should do some adjustments. Many of them have quite big impact on performance. Please check *Production setup* for more details, especially:

- *Enable indexing offloading*
- *Enable caching*
- *Use powerful database engine*
- *Disable debug mode*

Disable not needed checks

Some quality checks can be quite expensive and if you don't need them, they can save you some time during import. See *CHECK_LIST* for more information how to configure this.

Automatic creation of components

In case you have project with dozen of translation files, you might want to import all at once. This can be achieved using `import_project` or `import_json`.

First you need to create project which will contain all components and then it's just a matter of running `import_project` or `import_json`.

See also:

Management commands

Fulltext search

Fulltext search is based on Whoosh. You can either allow Weblate to directly update index on every change to content or offload this to separate process by `OFFLOAD_INDEXING`.

The first approach (immediate updates) allows more up to date index, but suffers locking issues in some setup (eg. Apache's mod_wsgi) and produces more fragmented index.

Offloaded indexing is always better choice for production setup - it only marks which items need to be reindexed and you need to schedule background process (`update_index`) to update index. This leads to faster response of the site and less fragmented index with cost that it might be slightly outdated.

See also:

`update_index`, `OFFLOAD_INDEXING`, *Fulltext search is too slow*, *I get "Lock Error" quite often while translating*, *Rebuilding index has failed with "No space left on device"*

Language definitions

In order to properly present different translation Weblate needs to know some information about used languages. Currently it comes with definitions for about 200 languages and the definition include language name, text direction, plural definitions and language code.

Parsing language codes

While parsing translations, Weblate attempts to map language code (usually ISO 639-1 one) to existing language object. If it can not find exact match, it tries to find best fit in existing languages (eg. it ignores default country code for given language - choosing `cs` instead of `cs_CZ`). Should this fail as well, it will create new language definition using the defaults (left to right text direction, one plural) and naming the language :guilabel:`xx_XX (generated)`. You might want to change this in the admin interface (see *Changing language defintions*) and report it to our issue tracker (see *Contributing*).

Changing language defintions

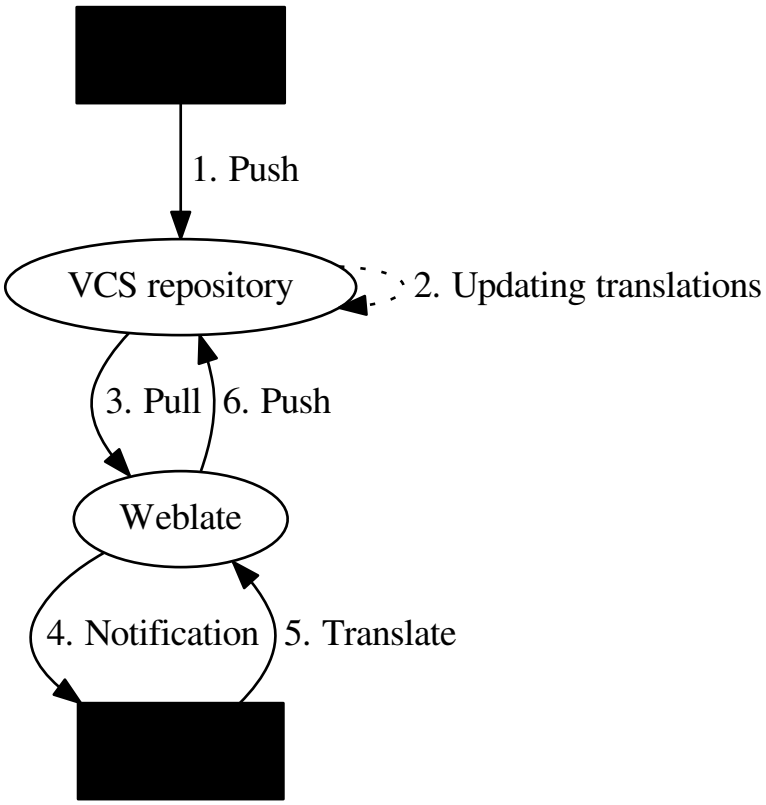
You can change language definitions in the admin interface (see *Administration*). The *Weblate languages* section allows you to change or add language definitions. While editing make sure that all fields are correct (especially plurals and text direction), otherwise the translators won't be able to properly edit those translations.

Continuous translation

Weblate provides you great infrastructure for translation to closely follow your development. This way translators can work on translations whole time and are not forced to translate huge amount of new texts before release.

The complete process can be described in following steps:

1. Developers make some 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 shows 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 will be notified based on their subscription settings.
5. Translators make translations using Weblate web interface.
6. Once translators are done, Weblate commits the changes to the local repository (see *Lazy commits*) and pushes them back if it has permissions to do that (see *Pushing changes*).



Updating repositories

You should set up some way how backend repositories are updated from their source. You can either use hooks (see *Notification hooks*) or just regularly run `updategit` (with selection of project or *–all* for updating all).

Whenever Weblate updates the repository, the *Post-update script* hooks are executed.

With Gettext po files, you might be often bitten by conflict in PO file headers. To avoid it, you can use shipped merge driver (`examples/git-merge-gettext-po`). To use it just put following configuration to your `.gitconfig`:

```
[merge "merge-gettext-po"]
  name = merge driver for gettext po files
  driver = /path/to/weblate/examples/git-merge-gettext-po %O %A %B
```

And enable its use by defining proper attributes in given repository (eg. in `.git/info/attributes`):

```
*.po merge=merge-gettext-po
```

Note: This merge driver assumes the changes in POT files always are done in branch we're trying to merge.

Changed in version 2.9: This merge driver is automatically installed in case Weblate finds it (this currently works only for Git checkout as distutils package does not include examples).

Avoiding merge conflicts

To avoid merge conflicts you should control when to update translation files in upstream repository to avoid Weblate having changes on same file.

You can achieve this using *Weblate's Web API* to force Weblate push all pending changes and lock translation while you are doing changes on your side.

The script for doing updates can look like:

```
# Lock Weblate translation
wlc lock
# Push changes from Weblate to upstream repository
wlc push
# 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
```

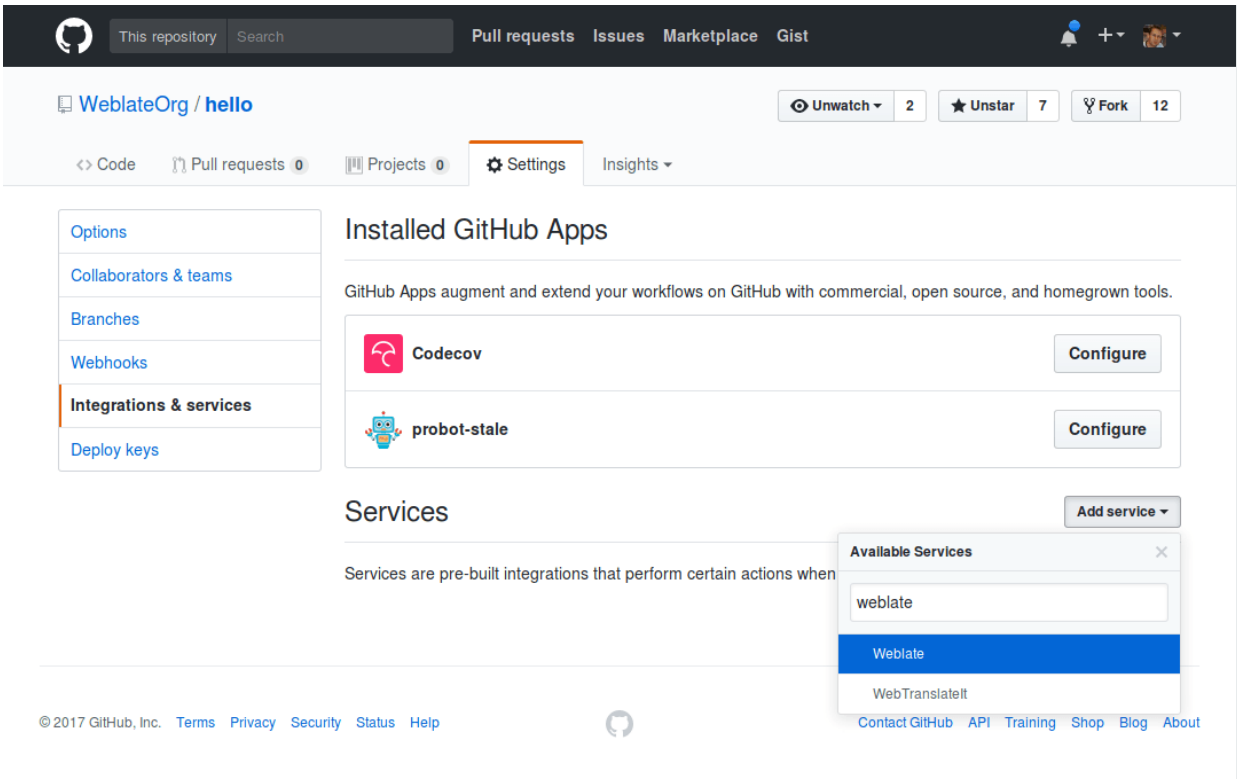
If you have multiple components sharing same repository, you need to lock them all separatey:

```
wlc lock foo/bar
wlc lock foo/baz
wlc lock foo/baj
```

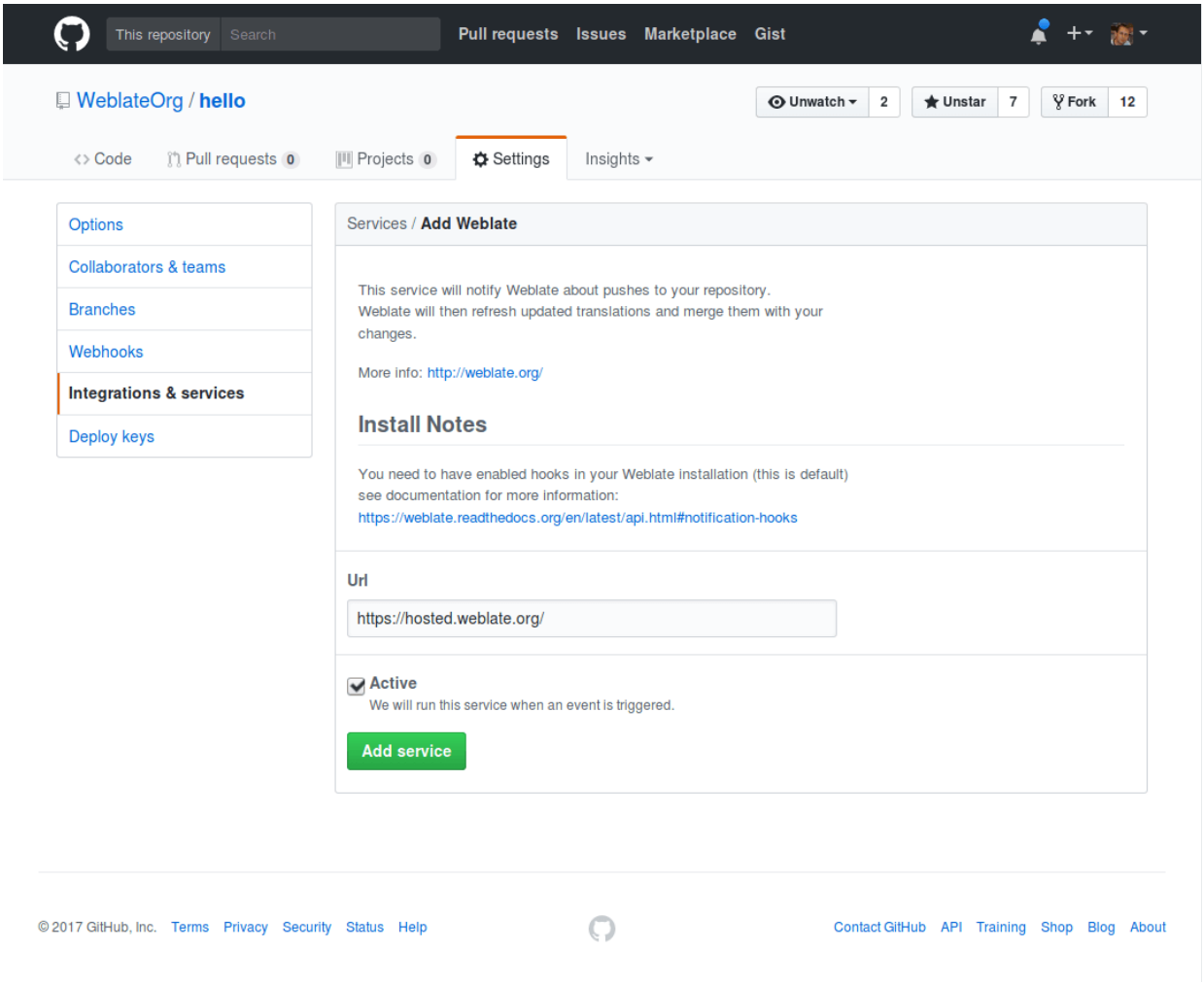
Note: The example uses *Weblate Client*, which will need configuration (API keys) to be able to control Weblate remotely. You can achieve same using any HTTP client instead of wlc, eg. curl, see *Weblate's Web API*.

Automatically receiving changes from GitHub

Weblate comes with native support for GitHub. To receive notifications on every push to GitHub repository, you just need to enable Weblate Service in the repository settings (*Integrations & services*) as shown on the image below:



To set the base URL of your Weblate installation (for example `https://hosted.weblate.org`) and Weblate will be notified about every push to GitHub repository:



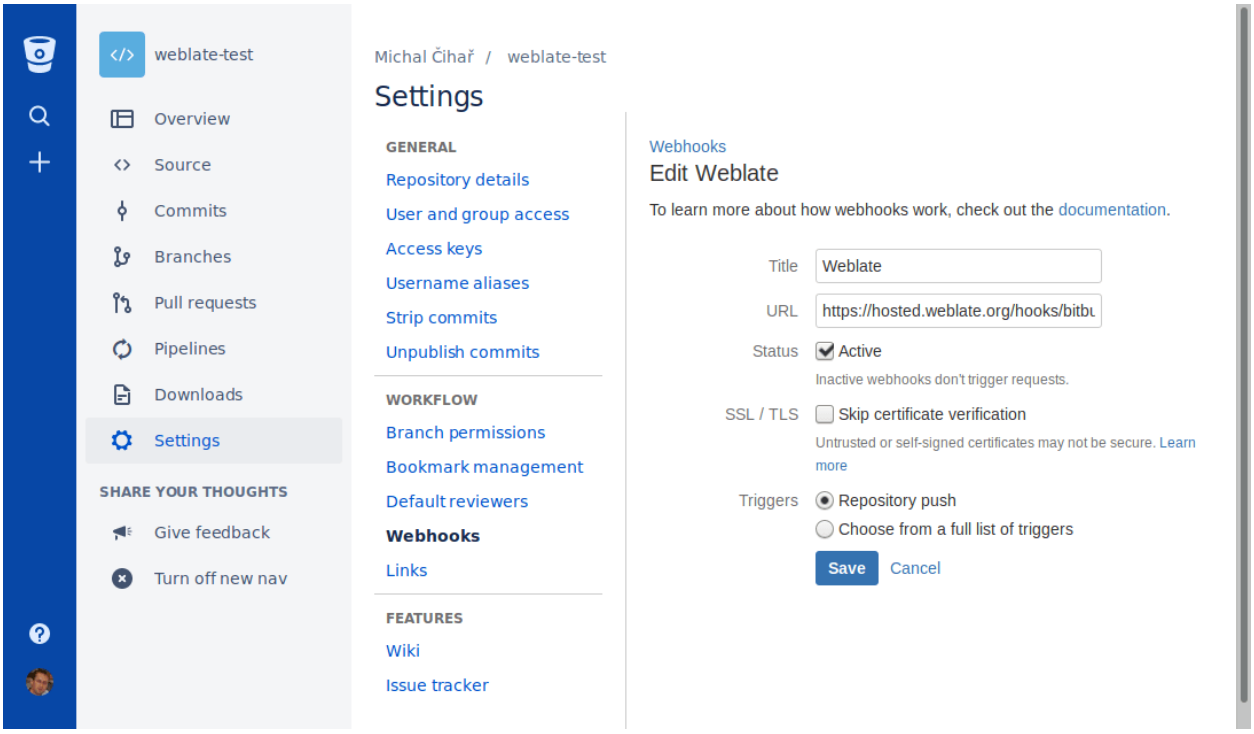
You can also use generic *Webhook*, in that case the *Payload URL* would have to be full path to the handler, for example `https://hosted.weblate.org/hooks/github/`.

See also:

`POST /hooks/github/`, *Pushing changes from Hosted Weblate*

Automatically receiving changes from Bitbucket

Weblate has support for Bitbucket webhooks, all you need to do is add webhook which triggers on 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, all you need to do is add project web hook 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*

Pushing changes

Each project can have configured push URL and in such case Weblate offers button to push changes to remote repository in web interface. Weblate can be also configured to automatically push changes on every commit.

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 used *Version control integration*, please check that chapter for more details.

Note: You can also enable automatic pushing changes on commit, this can be done in *Component configuration*.

See also:

See *Accessing repositories* for setting up SSH keys and *Lazy commits* for information about when Weblate decides to commit changes.

Pushing changes from Hosted Weblate

For Hosted Weblate there is dedicated push user registered on GitHub, Bitbucket and GitLab (with username *weblate* and named *Weblate push user*). You need to add this user as a collabolator and give him permissions to push to your repository. Let us know when you've done so and we will enable pushing changes from Hosted Weblate for you.

Merge or rebase

By default Weblate merges upstream repository into its own. This is safest way in case you also access underlying repository by other means. In case you don't need this, you can enable rebasing of changes on upstream, what will produce history with less merge commits.

Note: Rebasing can cause you troubles in case of complicated merges, so carefully consider whether you want to enable them or not.

Interacting with others

Weblate makes it easy to interact with others using its API.

See also:

Weblate's Web API

Lazy commits

Default behaviour (configured by `LAZY_COMMITS`) of Weblate is to group commits from same author into one if possible. This heavily reduces number of commits, however you might need to explicitly tell to do the commits in case you want to get VCS repository in sync, eg. for merge (this is by default allowed for Managers group, see *Access control*).

The changes are in this mode committed once any of following conditions is fulfilled:

- somebody else works on the translation
- merge from upstream occurs
- import of translation happens
- translation for a language is completed
- explicit commit is requested

You can also additionally set a cron job to commit pending changes after some delay, see `commit_pending` and *Running maintenance tasks*.

Processing repository with scripts

You can customize way how Weblate manipulates with repository by set of scripts. These include *Post-update script*, *Pre-commit script*, *Post-commit script*, *Post-add script* and *Post-push script* and are briefly described in *Component configuration*.

Their naming quite clearly tells when given script is executed. The commit related scripts always get one parameter with full path to the translation file which has been changed.

The script is executed with the current directory set to root of VCS repository for given component.

Additionally following environment variables are available:

WL_VCS
Used version control system.

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 for *POST_UPDATE_SCRIPTS*).

See also:

POST_UPDATE_SCRIPTS, *PRE_COMMIT_SCRIPTS*, *POST_COMMIT_SCRIPTS*, *POST_PUSH_SCRIPTS*, *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 translation before it is committed to the repository. The pre commit script is exactly the place to achieve this.

Before using any scripts, you need to list them in *PRE_COMMIT_SCRIPTS* configuration variable. Then you can enable them at *Component configuration* configuration as *Pre commit script*.

It is passed single parameter consisting of file name of current translation.

The script can also generate additional file to be included in the commit. This can be configured as *Extra commit file* at *Component configuration* configuration. You can use following format strings in the filename:

%(language)s Language code

Example - generating mo files in repository

Allow usage of the hook in the configuration

```
PRE_COMMIT_SCRIPTS = (
    '/usr/share/weblate/examples/hook-generate-mo',
)
```

To enable it, choose now *hook-generate-mo* as *Pre commit script*. You will also want to add path to generated files to be included in VCS commit, for example `po/%(language)s.mo` as *Extra commit file*.

You can find more example scripts in `examples` folder within Weblate sources, their name start with `hook-`.

Translation process

Suggestion voting

New in version 1.6: This feature is available since Weblate 1.6.

In default Weblate setup, everybody can add suggestions and logged in users can accept them. You might however want to have more eyes on the translation and require more people to accept them. This can be achieved by suggestion voting. You can enable this on *Component configuration* configuration by *Suggestion voting* and *Autoaccept suggestions*. The first one enables voting feature, while the latter allows you to configure threshold at which suggestion will gets automatically accepted (this includes own vote from suggesting user).

Note: Once you enable automatic accepting, normal users lose privilege to directly save translations or accept suggestions. This can be overriden by *Can override suggestion state* privilege (see *Access control*).

You can combine these with *Access control* into one of following setups:

- Users can suggest and vote for suggestions, limited group controls what is accepted - enable voting but not automatic accepting and remove privilege from users to save translations.
- Users can suggest and vote for suggestions, which get automatically accepted once defined number of users agree on this - enable voting and set desired number of votes for automatic accepting.
- Optional voting for suggestions - you can also only enable voting and in this case it can be optionally used by users when they are not sure about translation (they can suggest more of them).

Translation locking

To improve collaboration, it is good to prevent duplicate effort on translation. To achieve this, translation can be locked for single translator. This can be either done manually on translation page or is done automatically when somebody starts to work on translation. The automatic locking needs to be enabled using *AUTO_LOCK*.

The automatic lock is valid for *AUTO_LOCK_TIME* seconds and is automatically extended on every translation made and while user has opened translation page.

User can also explicitly lock translation for *LOCK_TIME* seconds.

Additional information on source strings

Weblate allows you to enhance translation process with information available in the translation files. This includes strings prioritization, check flags or providing visual context. All these features can be set on the *Reviewing source strings*:

Check flags Add

Please enter a comma separated list of check flags, see documentation for more details.

Save

Priority

Medium

Strings with higher priority are offered first to translators.

Save

Screenshot context

No screenshot currently associated! Manage screenshots

Add new screenshot

Screenshot name

Image

Procházet... Soubor nevybrán.

Upload JPEG or PNG images up to 2000x2000 pixels.

Upload

Source string location

weblate/templates/admin/custom-index.html:13 weblate/templates/admin/report.html:4 weblate/templates/admin/report.html:13

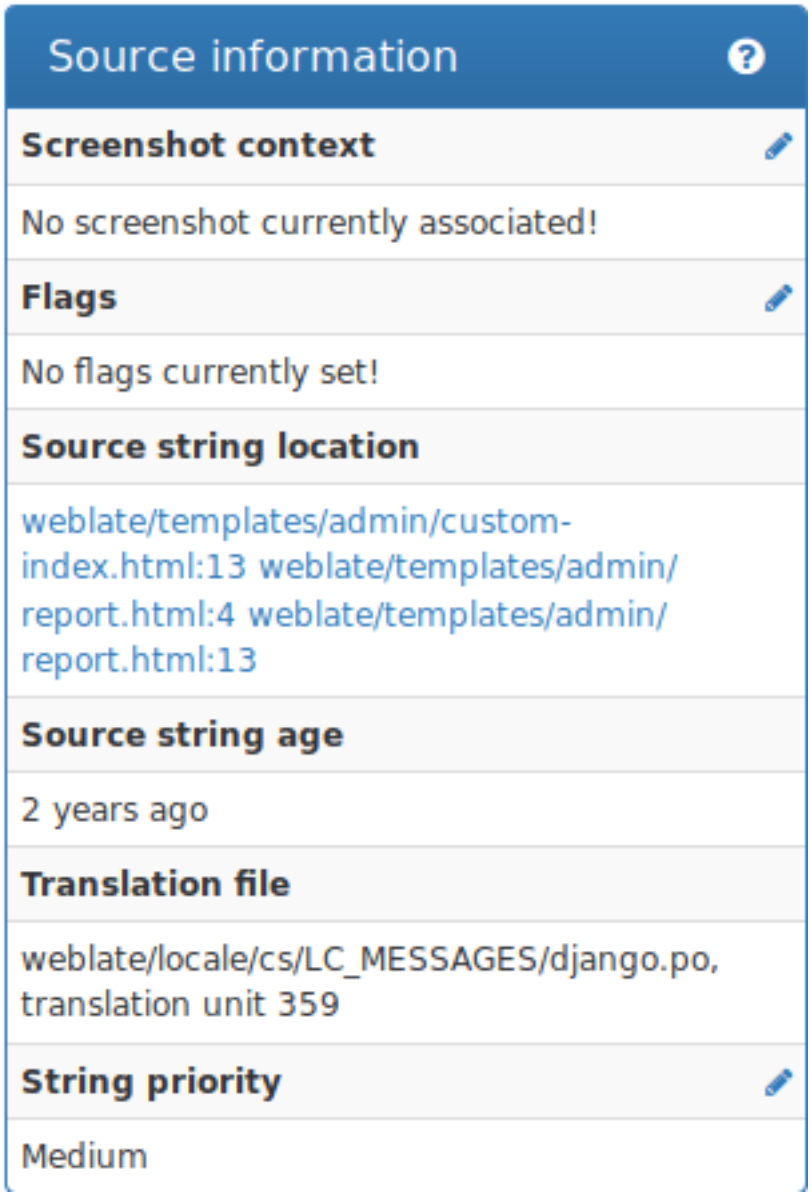
Source string age

2 years ago

Translation file

weblate/locale/LC_MESSAGES/django.po, translation unit 359

You can access this also directly from translating interface when clicking on the edit icon next to *Screenshot context*, *Flags* or *String priority*:



Strings prioritization

New in version 2.0.

You can change string priority, strings with higher priority are offered first for translation. This can be useful for prioritizing translation of strings which are seen first by users or are otherwise important.

Quaity check flags

New in version 2.4.

Default set of quality check flags is determined from the translation *Component configuration* and the translation file. However you might want to customize this per source string and you have the option here.

See also:

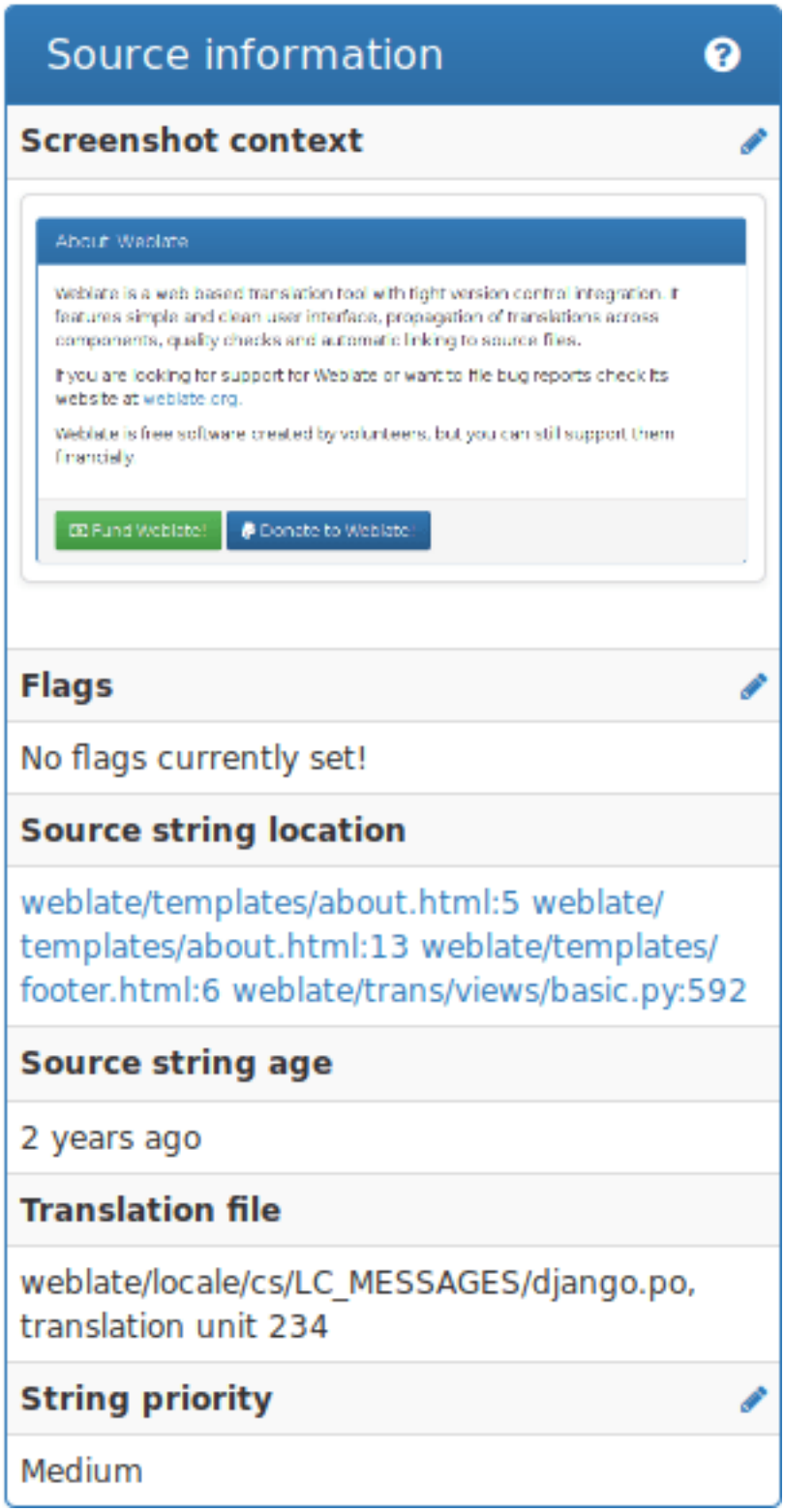
Quality checks

Visual context for strings

New in version 2.9.

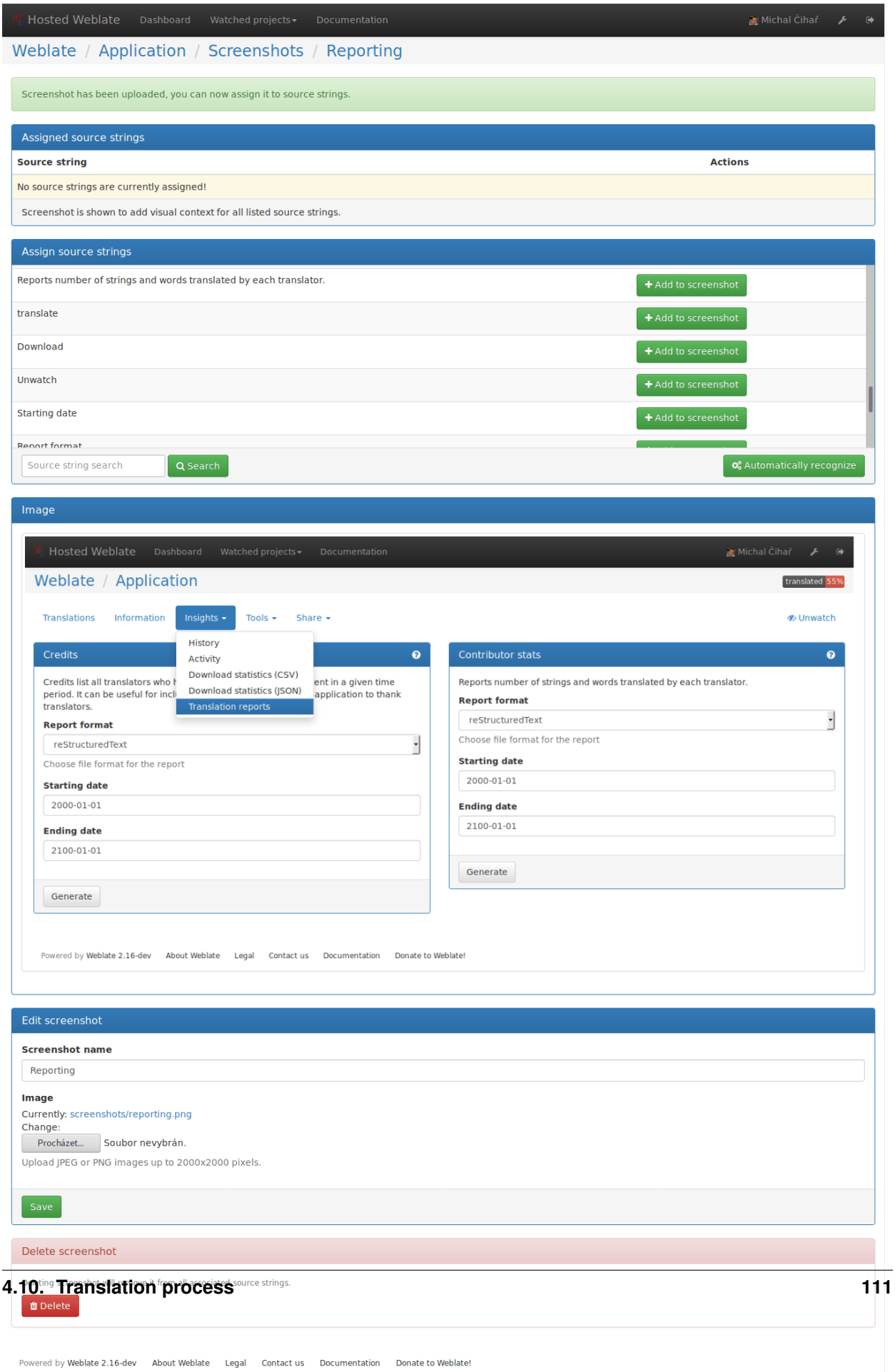
You can upload screenshot showing usage of given source string within your application. This can help translators to understand where it is used and how it should be translated.

Uploaded screenshot is shown in the translation context sidebar:



In addition to *Reviewing source strings*, screenshots have separate management interface. You can find it under *Tools* menu. This allows you to upload screenshots, assign them to source strings manually or using OCR.

Once screenshot is uploaded, you will be presented following interface to manage it and assign to source strings:


Hosted Weblate
Dashboard
Watched projects
Documentation
Michal Čihař
Weblate / Application / Screenshots / Reporting
Screenshot has been uploaded, you can now assign it to source strings.
Assigned source strings
Source string
Actions
No source strings are currently assigned!
Screenshot is shown to add visual context for all listed source strings.
Assign source strings
Reports number of strings and words translated by each translator.
translate
Download
Unwatch
Starting date
Report format
Add to screenshot
Source string search
Search
Automatically recognize
Image
Weblate / Application
translated 55%
Translations
Information
Insights
Tools
Share
Unwatch
History
Activity
Download statistics (CSV)
Download statistics (JSON)
Translation reports
Credits
Credits list all translators who h
ent in a given time
period. It can be useful for incl
application to thank
translators.
Report format
reStructuredText
Choose file format for the report
Starting date
2000-01-01
Ending date
2100-01-01
Generate
Contributor stats
Reports number of strings and words translated by each translator.
Report format
reStructuredText
Choose file format for the report
Starting date
2000-01-01
Ending date
2100-01-01
Generate
Powered by Weblate 2.16-dev
About Weblate
Legal
Contact us
Documentation
Donate to Weblate!
Edit screenshot
Screenshot name
Reporting
Image
Currently: screenshots/reporting.png
Change:
Procházet...
Soubor nevybrán.
Upload JPEG or PNG images up to 2000x2000 pixels.
Save
Delete screenshot
Delete

Checks and fixups

Custom automatic fixups

You can also implement own automatic fixup in addition to 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 following automatic fixup would replace every occurrence of string `foo` in translation with `bar`:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2017 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 weblate.trans.autofixes.base import AutoFix
from django.utils.translation import ugettext_lazy as _


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 fully-qualified path to Python class in the *AUTOFIX_LIST*, see *Using custom modules and classes*.

Customizing checks

Fine tuning existing checks

You can fine tune checks for each source string (in source strings review, see *Additional information on source strings*) or in the *Component configuration* (*Quality checks flags*), here is current list of flags accepted:

skip-review-flag Ignore whether unit is marked for review when importing from VCS. This can be useful for *XLIFF*.

add-source-review Whether to mark all new string in source language for review. This can be useful if you want to proofread the source language. This flag has no meaning for bilingual translations.

add-review Whether to mark all new string for review. This can be useful if you want to proofread the translations done by developers.

rst-text Treat text as RST document, affects *Unchanged translation*.

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*.

python-format, c-format, php-format, python-brace-format, javascript-format Treats all string like format strings, affects *Format strings*, *Format strings*, *Format strings*, *Format strings*, *Format 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.

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).

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.trans.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 fully-qualified path to Python class in the `CHECK_LIST`, see *Using custom modules and classes*.

Checking translation text does not contain "foo"

This is pretty simple check which just checks whether translation does not contain string "foo".

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2017 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/>.
#
"""Simple quality check example."""

from weblate.trans.checks.base import TargetCheck
from django.utils.translation import ugettext_lazy as _


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 - 2017 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 weblate.trans.checks.base import TargetCheck
from django.utils.translation import ugettext_lazy as _


class PluralCzechCheck(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_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
```

Using custom modules and classes

You have implemented code for *Custom automatic fixups* or *Customizing checks* and now it's time to install it into Weblate. That can be achieved by adding its fully-qualified path to Python class to appropriate settings.

This means that the module with class needs to be placed somewhere where Python interpreter can import it - either in system path (usually something like `/usr/lib/python2.7/site-packages/`) or in Weblate directory, which is also added to the interpreter search path.

Assuming you've created `mahongo.py` containing your custom quality check. You can place it among Weblate checks in `weblate/trans/checks/` folder and then add it as following:

```
CHECK_LIST = (
    'weblate.trans.checks.mahongo.MahongoCheck',
)
```

As you can see, it's comma separated path to your module and class name.

Alternatively, you can create 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_custom_checks* 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_custom_checks",
    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_custom_checks'],
)
```

4. Now you can install it using **`python setup.py install`**
5. Once installed into system Python path, you can use it from there:

```
CHECK_LIST = (
    'weblate_custom_checks.mahongo.MahongoCheck',
)
```

Overall your module structure should look like:

```
weblate_custom_checks
- setup.py
- weblate_custom_checks
    - __init__.py
    - mahongo.py
```

Machine translation

Weblate has builtin support for several machine translation services and it's up to administrator to enable them. The services have different terms of use, so please check whether you are allowed to use them before enabling in Weblate. The individual services are enabled using *MACHINE_TRANSLATION_SERVICES*.

The source language can be configured at *Project configuration*.

Amagama

Special installation of *tmserver* run by Virtaal authors.

To enable this service, add `weblate.trans.machine.tmserver.AmagamaTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

Amagama Translation Memory server

Apertium

A free/open-source machine translation platform providing translation to limited set of languages.

The recommended way how to use Apertium is to run own Apertium APy server.

Alternatively you can use https://www.apertium.org/apy if you don't expect to make too many requests.

To enable this service, add `weblate.trans.machine.apertium.ApertiumAPYTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_APERTIUM_KEY, Apertium website, Apertium APy documentation

Glosbe

Free dictionary and translation memory for almost every living language.

API is free to use, regarding indicated data source license. There is a limit of call that may be done from one IP in fixed period of time, to prevent from abuse.

To enable this service, add `weblate.trans.machine.glosbe.GlosbeTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

Glosbe website

Google Translate

Machine translation service provided by Google.

This service uses Translation API and you need to obtain API key and enable billing on Google API console.

To enable this service, add `weblate.trans.machine.google.GoogleTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_GOOGLE_KEY, Google translate documentation

Microsoft Translator

Deprecated since version 2.10.

Note: This service is deprecated by Microsoft as needs to be replaced by *Microsoft Cognitive Services Translator*.

Machine translation service provided by Microsoft, it's known as Bing Translator as well.

You need to register at Azure market and use Client ID and secret from there.

To enable this service, add `weblate.trans.machine.microsoft.MicrosoftTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_MICROSOFT_ID, *MT_MICROSOFT_SECRET*, Bing Translator, Azure datamarket

Microsoft Cognitive Services Translator

New in version 2.10.

Note: This is replacement service for *Microsoft Translator*.

Machine transation service provided by Microsoft in Azure portal as a one of Cognitive Services.

You need to register at Azure portal and use key you obtain there.

To enable this service, add `weblate.trans.machine.microsoft.MicrosoftCognitiveTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_MICROSOFT_COGNITIVE_KEY, Cognitive Services - Text Translation API, Microsoft Azure Portal

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.trans.machine.mymemory.MyMemoryTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_MYMEMORY_EMAIL, *MT_MYMEMORY_USER*, *MT_MYMEMORY_KEY*, MyMemory website

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 enhanced version of tmserver.

First you will want to import some data to the translation memory:

To enable this service, add `weblate.trans.machine.tmserver.TMServerTranslation` to *MACHINE_TRANSLATION_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, a Translation Memory service, Amagama Translation Memory

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.trans.machine.yandex.YandexTranslation` to *MACHINE_TRANSLATION_SERVICES*.

See also:

MT_YANDEX_KEY, Yandex Translate API

Weblate

Weblate can be source of machine translation as well. There are two services to provide you results - one does exact search for string, the other one finds all similar strings.

First one is useful for full string translations, the second one for finding individual phrases or words to keep the translation consistent.

To enable these services, add `weblate.trans.machine.weblatetm.WeblateSimilarTranslation` (for similar string matching) and/or `weblate.trans.machine.weblatetm.WeblateTranslation` (for exact string matching) to *MACHINE_TRANSLATION_SERVICES*.

Note: For similarity matching, it is recommended to have Whoosh 2.5.2 or later, earlier versions can cause infinite looks under some occasions.

Custom machine translation

You can also implement own machine translation services using few lines of Python code. Following example implements translation to fixed list of languages using `dictionary` Python module:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2017 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/>.
#
"""Machine translation example."""

from weblate.trans.machine.base import MachineTranslation
import dictionary


class SampleTranslation(MachineTranslation):
    """Sample machine translation interface."""
    name = 'Sample'

    def download_languages(self):
        """Return list of languages your machine translation supports."""
        return set(('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 *MACHINE_TRANSLATION_SERVICES* and Weblate will start using that.

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 check also Django's documentation for parameters which configure Django itself.

ANONYMOUS_USER_NAME

User name of user for defininining privileges of not logged in user.

See also:

Access control

AUTH_LOCK_ATTEMPTS

New in version 2.14.

Maximal number of failed authentication attempts before rate limiting is applied.

This is currently applied in following locations:

- On login, the acccount 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,

AUTH_MAX_ATTEMPTS

New in version 2.14.

Maximal number of authentication attempts before rate limiting applies.

Defaults to 5.

See also:

Rate limiting, *AUTH_CHECK_WINDOW*, *AUTH_LOCKOUT_TIME*

AUTH_CHECK_WINDOW

New in version 2.14.

Length of authentication window for rate limiting in seconds.

Defaults to 300 (5 minutes).

See also:

Rate limiting, *AUTH_MAX_ATTEMPTS*, *AUTH_LOCKOUT_TIME*

AUTH_LOCKOUT_TIME

New in version 2.14.

Length of authentication lockout window after rate limit is applied.

Defaults to 600 (10 minutes).

See also:

Rate limiting, AUTH_MAX_ATTEMPTS, AUTH_CHECK_WINDOW

AUTH_TOKEN_VALID

New in version 2.14.

Validity of token in activation and password reset mails in seconds.

Defaults to 3600 (1 hour).

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.

AUTO_LOCK

Enables automatic locking of translation when somebody is working on it.

See also:

Translation locking

AUTO_LOCK_TIME

Time in seconds for how long the automatic lock for translation will be active.

See also:

Translation locking

AUTOFIX_LIST

List of automatic fixups to apply when saving the message.

You need to provide fully-qualified path to 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 traling 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*

BACKGROUND_HOOKS

Whether to run hooks in background. This is generally recommended unless you are debugging.

CHECK_LIST

List of quality checks to perform on translation.

You need to provide fully-qualified path to Python class implementing the check interface.

Some of the checks are not useful for all projects, so you are welcome to adjust list of performed on your installation.

For example you can enable only few of them:

```
CHECK_LIST = (
    'weblate.trans.checks.same.SameCheck',
    'weblate.trans.checks.chars.BeginNewlineCheck',
    'weblate.trans.checks.chars.EndNewlineCheck',
    'weblate.trans.checks.chars.BeginSpaceCheck',
    'weblate.trans.checks.chars.EndSpaceCheck',
    'weblate.trans.checks.chars.EndStopCheck',
    'weblate.trans.checks.chars.EndColonCheck',
    'weblate.trans.checks.chars.EndQuestionCheck',
    'weblate.trans.checks.chars.EndExclamationCheck',
    'weblate.trans.checks.chars.EndEllipsisCheck',
    'weblate.trans.checks.chars.EndSemicolonCheck',
    'weblate.trans.checks.chars.MaxLengthCheck',
    'weblate.trans.checks.format.PythonFormatCheck',
    'weblate.trans.checks.format.PythonBraceFormatCheck',
    'weblate.trans.checks.format.PHPFormatCheck',
    'weblate.trans.checks.format.CFormatCheck',
    'weblate.trans.checks.format.PerlFormatCheck',
    'weblate.trans.checks.format.JavascriptFormatCheck',
    'weblate.trans.checks.consistency.SamePluralsCheck',
    'weblate.trans.checks.consistency.PluralsCheck',
    'weblate.trans.checks.consistency.ConsistencyCheck',
    'weblate.trans.checks.consistency.TranslatedCheck',
    'weblate.trans.checks.chars.NewlineCountingCheck',
    'weblate.trans.checks.markup.BBCodeCheck',
    'weblate.trans.checks.chars.ZeroWidthSpaceCheck',
    'weblate.trans.checks.markup.XMLTagsCheck',
    'weblate.trans.checks.source.OptionalPluralCheck',
    'weblate.trans.checks.source.EllipsisCheck',
```

```
    'weblate.trans.checks.source.MultipleFailingCheck',
)
```

Note: Once you change this setting the existing checks will be still stored in the database, only newly changed translation will be affected by the change. To apply change to already stored translations, you need to run *updatechecks*.

See also:

Quality checks, *Customizing checks*

COMMIT_PENDING_HOURS

New in version 2.10.

Default interval for commiting pending changes using *commit_pending*.

See also:

Running maintenance tasks, *commit_pending*

DATA_DIR

New in version 2.1: In previous versions the directories were configured separately as *GIT_ROOT* and *WHOOSH_INDEX*.

Directory where Weblate stores all data. This consists of VCS repositories, fulltext index and various configuration files for external tools.

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.

vcs Version control repositories.

whoosh Fulltext search index using Whoosh engine.

DEFAULT_COMMITER_EMAIL

New in version 2.4.

Default commiter email when creating translation component (see *Component configuration*), defaults to noreply@weblate.org.

See also:

DEFAULT_COMMITER_NAME, *Component configuration*

DEFAULT_COMMITER_NAME

New in version 2.4.

Default commiter name when creating translation component (see *Component configuration*), defaults to `Weblate`.

See also:

DEFAULT_COMMITER_EMAIL, *Component configuration*

DEFAULT_TRANSLATION_PROPAGATION

New in version 2.5.

Default setting for translation propagation (see *Component configuration*), defaults to `True`.

See also:

Component configuration

ENABLE_AVATARS

Whether to enable libravatar/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, *Avatars*

ENABLE_HOOKS

Whether to enable anonymous remote hooks.

See also:

Notification hooks

ENABLE_HTTPS

Whether to send links to the Weblate as https or http. This setting affects sent mails and generated absolute URLs.

See also:

Set correct site name

ENABLE_SHARING

Whether to show links to share translation progress on social networks.

GIT_ROOT

Deprecated since version 2.1: This setting is no longer used, use *DATA_DIR* instead.

Path where Weblate will store cloned VCS repositories. Defaults to `repos` subdirectory.

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*

GOOGLE_ANALYTICS_ID

Google Analytics ID to enable monitoring of Weblate using Google Analytics.

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`.

IP_BEHIND_REVERSE_PROXY

New in version 2.14.

Indicates whether Weblate is running behind 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*

IP_BEHIND_REVERSE_PROXY

New in version 2.14.

Indicates from which header Weblate should obtain IP address when *IP_BEHIND_REVERSE_PROXY* is enabled.

Defaults to `HTTP_X_FORWARDED_FOR`.

See also:

Rate limiting, *IP address for rate limiting*

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.

Defaults to 0.

See also:

Rate limiting, *IP address for rate limiting*

LAZY_COMMITS

Delay creating VCS commits until this is necessary. This heavily reduces number of commits generated by Weblate at expense of temporarily not being able to merge some changes as they are not yet committed.

See also:

Lazy commits

LOCK_TIME

Time in seconds for how long the translation will be locked for single translator when locked manually.

See also:

Translation locking

LOGIN_REQUIRED_URLS

List of URL which require login (besides standard rules built into Weblate). This allows you to password protect whole installation using:

```
LOGIN_REQUIRED_URLS = (
    r'/(.*)$',
)
```

LOGIN_REQUIRED_URLS_EXCEPTIONS

List of exceptions for *LOGIN_REQUIRED_URLS*, in case you won't specify this list, the default value will be used, which allows users to access 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
)
```

MACHINE_TRANSLATION_SERVICES

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.

```
MACHINE_TRANSLATION_SERVICES = (
    'weblate.trans.machine.apertium.ApertiumAPYTranslation',
    'weblate.trans.machine.glosbe.GlosbeTranslation',
    'weblate.trans.machine.google.GoogleTranslation',
    'weblate.trans.machine.microsoft.MicrosoftTranslation',
    'weblate.trans.machine.mymemory.MyMemoryTranslation',
    'weblate.trans.machine.tmserver.TMServerTranslation',
    'weblate.trans.machine.weblatetm.WeblateSimilarTranslation',
    'weblate.trans.machine.weblatetm.WeblateTranslation',
)
```

See also:

Machine translation, *Machine translation*

MT_APERTIUM_APY

URL of the Apertium APy server, see http://wiki.apertium.org/wiki/Apertium-apy

See also:

Apertium, *Machine translation*, *Machine translation*

MT_APERTIUM_KEY

API key for Apertium Web Service, currently not used.

Not needed at all when running own Apertium APy server.

See also:

Apertium, *Machine translation*, *Machine translation*

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*

MT_MICROSOFT_ID

Cliend ID for Microsoft Translator service.

See also:

Microsoft Translator, *Machine translation*, *Machine translation*, Azure datamarket

MT_MICROSOFT_SECRET

Client secret for Microsoft Translator service.

See also:

Microsoft Translator, *Machine translation*, *Machine translation*, Azure datamarket

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, Microsfot Azure Portal

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

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

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

MT_TMSERVER

URL where tmserver is running.

See also:

tmserver, *Machine translation*, *Machine translation*, tmserver, a Translation Memory service

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*

NEARBY_MESSAGES

How many messages around current one to show during translating.

OFFLOAD_INDEXING

Offload updating of fulltext index to separate process. This heavily improves responsiveness of online operation on expense of slightly outdated index, which might still point to older content.

While enabling this, don't forget scheduling runs of `update_index` in cron or similar tool.

This is recommended setup for production use.

See also:

Fulltext search

PIWIK_SITE_ID

ID of a site in Piwik you want to track.

See also:

`PIWIK_URL`

PIWIK_URL

URL of a Piwik installation you want to use to track Weblate users. For more information about Piwik see <https://piwik.org/>.

See also:

`PIWIK_SITE_ID`

POST_ADD_SCRIPTS

New in version 2.4.

List of scripts which are allowed as post add scripts. The script needs to be later enabled in the *Component configuration*.

Weblate comes with few example hook scripts which you might find useful:

`examples/hook-update-linguas` Updates LINGUAS file or ALL_LINGUAS in confiugure script.

See also:

Processing repository with scripts

POST_UPDATE_SCRIPTS

New in version 2.3.

List of scripts which are allowed as post update scripts. The script needs to be later enabled in the *Component configuration*.

Weblate comes with few example hook scripts which you might find useful:

`examples/hook-update-resx` Updates resx file to match template by adding new translations and removing obsolete ones.

`examples/hook-cleanup-android` Removes obsolete units from Android resource strings.

See also:

Processing repository with scripts

PRE_COMMIT_SCRIPTS

List of scripts which are allowed as pre commit scripts. The script needs to be later enabled in the *Component configuration*.

For example you can allow script which does some cleanup:

```
PRE_COMMIT_SCRIPTS = (
    '/usr/local/bin/cleanup-translation',
)
```

Weblate comes with few example hook scripts which you might find useful:

examples/hook-generate-mo Generates MO file from a PO file

examples/hook-unwrap-po Unwraps lines in a PO file.

examples/hook-sort-properties Sort and cleanups Java properties file.

examples/hook-replace-single-quotes Replaces single quotes in a file.

See also:

Processing repository with scripts

POST_COMMIT_SCRIPTS

New in version 2.4.

List of scripts which are allowed as post commit scripts. The script needs to be later enabled in the *Component configuration*.

See also:

Processing repository with scripts

POST_PUSH_SCRIPTS

New in version 2.4.

List of scripts which are allowed as post push scripts. The script needs to be later enabled in the *Component configuration*.

See also:

Processing repository with scripts

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 not logged in users.

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.

SELF_ADVERTISEMENT

Enables self advertisement of Weblate in case there are no configured ads.

See also:

Advertisement

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.

SITE_TITLE

Site title to be used in website and emails as well.

TTF_PATH

Path to Droid fonts used for widgets and charts.

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.

WHOOSH_INDEX

Deprecated since version 2.1: This setting is no longer used, use *DATA_DIR* instead.

Directory where Whoosh fulltext indices will be stored. Defaults to `whoosh-index` subdirectory.

Sample configuration

Following example is shipped as `weblate/settings_example.py` with Weblate:

```
# -*- coding: utf-8 -*-
#
# Copyright © 2012 - 2017 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 platform
import os
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_psycopg2', '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',
        # 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': {
            # Uncomment for MySQL older than 5.7:
```

```
            # 'init_command': "SET sql_mode='STRICT_TRANS_TABLES'"
        },
    }
}

BASE_DIR = 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.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
# If running in a Windows environment this must be set to the same as 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 = (
    ('az', 'Azrbaycan'),
    ('be', ''),
    ('be@latin', 'Biełaruskaja'),
    ('bg', ''),
    ('br', 'Brezhoneg'),
    ('ca', 'Català'),
    ('cs', 'Čeština'),
    ('da', 'Dansk'),
    ('de', 'Deutsch'),
    ('en', 'English'),
    ('en-gb', 'English (United Kingdom)'),
    ('el', 'Ελληνικ'),
    ('es', 'Español'),
    ('fi', 'Suomi'),
    ('fr', 'Français'),
    ('fy', 'Frysk'),
    ('gl', 'Galego'),
    ('he', ''),
    ('hu', 'Magyar'),
    ('id', 'Indonesia'),
    ('it', 'Italiano'),
    ('ja', ''),
    ('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.
    # 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 examples/generate-secret-key
```

```
SECRET_KEY = 'jm8fqjlg+5!#xu%e-oh#7!$aa7!6avf7ud*_v=chdrb9qdco6('  # noqa

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        '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',
)

# 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_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.reauthenticate',
    'social_core.pipeline.social_auth.associate_by_email',
    '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',
    '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/#auth'.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',
    },
    {
        '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',
    },
]

# Middleware
MIDDLEWARE_CLASSES = (
    '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'

INSTALLED_APPS = (
    # Has to be first to override Django admin templates:
    'weblate.wladmin',

    '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.trans',
    'weblate.lang',
```

```
    'weblate.permissions',
    'weblate.screenshots',
    'weblate.accounts',
    'weblate.utils',

    # Optional: Git exporter
    # 'weblate.gitexport',

    # This application has to be placed last!
    'weblate',
)

# Path to locales
LOCALE_PATHS = (os.path.join(BASE_DIR, '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:
        SysLogHandler(address='/dev/log', facility=SysLogHandler.LOG_LOCAL2)
        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': False,
    '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'
        },
    },
    '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'
        },
        '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,
        },
        # Logging database queries
        # 'django.db.backends': {
        #     'handlers': [DEFAULT_LOG],
        #     'level': 'DEBUG',
        # },
        'weblate': {
            'handlers': [DEFAULT_LOG],
            'level': 'DEBUG',
        },
        # Logging VCS operations
        # 'weblate-vcs': {
        #     'handlers': [DEFAULT_LOG],
        #     'level': 'DEBUG',
        # },
        # 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
# MACHINE_TRANSLATION_SERVICES = (
#     'weblate.trans.machine.apertium.ApertiumAPYTranslation',
#     'weblate.trans.machine.glosbe.GlosbeTranslation',
#     'weblate.trans.machine.google.GoogleTranslation',
#     'weblate.trans.machine.microsoft.MicrosoftCognitiveTranslation',
#     'weblate.trans.machine.mymemory.MyMemoryTranslation',
#     'weblate.trans.machine.tmserver.AmagamaTranslation',
#     'weblate.trans.machine.tmserver.TMServerTranslation',
#     'weblate.trans.machine.yandex.YandexTranslation',
#     'weblate.trans.machine.weblatetm.WeblateSimilarTranslation',
#     'weblate.trans.machine.weblatetm.WeblateTranslation',
# )

# Machine translation API keys

# URL of the Apertium APy server
MT_APERTIUM_APY = None

# Microsoft Translator service, register at
# https://datamarket.azure.com/developer/applications/
MT_MICROSOFT_ID = None
MT_MICROSOFT_SECRET = None

# Microsoft Cognitive Services Translator API, register at
# https://portal.azure.com/
MT_MICROSOFT_COGNITIVE_KEY = None

# 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

# API key for Yandex Translate API
MT_YANDEX_KEY = None

# tmserver URL
MT_TMSERVER = None
```

```
# 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:
# http://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 (since Django 1.11)
CSRF_USE_SESSIONS = True
SESSION_COOKIE_SECURE = ENABLE_HTTPS
# Session cookie age (in seconds)
SESSION_COOKIE_AGE = 1209600

# 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_BEHIND_REVERSE_PROXY = False
IP_PROXY_HEADER = 'HTTP_X_FORWARDED_FOR'
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

# Whether to run hooks in background
BACKGROUND_HOOKS = True

# Number of nearby messages to show in each direction
NEARBY_MESSAGES = 5

# Enable lazy commits
LAZY_COMMITS = True

# Offload indexing
```

```
OFFLOAD_INDEXING = False

# Translation locking
AUTO_LOCK = True
AUTO_LOCK_TIME = 60
LOCK_TIME = 15 * 60

# Render forms using bootstrap
CRISPY_TEMPLATE_PACK = 'bootstrap3'

# List of quality checks
# CHECK_LIST = (
#     'weblate.trans.checks.same.SameCheck',
#     'weblate.trans.checks.chars.BeginNewlineCheck',
#     'weblate.trans.checks.chars.EndNewlineCheck',
#     'weblate.trans.checks.chars.BeginSpaceCheck',
#     'weblate.trans.checks.chars.EndSpaceCheck',
#     'weblate.trans.checks.chars.EndStopCheck',
#     'weblate.trans.checks.chars.EndColonCheck',
#     'weblate.trans.checks.chars.EndQuestionCheck',
#     'weblate.trans.checks.chars.EndExclamationCheck',
#     'weblate.trans.checks.chars.EndEllipsisCheck',
#     'weblate.trans.checks.chars.EndSemicolonCheck',
#     'weblate.trans.checks.chars.MaxLengthCheck',
#     'weblate.trans.checks.format.PythonFormatCheck',
#     'weblate.trans.checks.format.PythonBraceFormatCheck',
#     'weblate.trans.checks.format.PHPFormatCheck',
#     'weblate.trans.checks.format.CFormatCheck',
#     'weblate.trans.checks.format.PerlFormatCheck',
#     'weblate.trans.checks.format.JavascriptFormatCheck',
#     'weblate.trans.checks.consistency.PluralsCheck',
#     'weblate.trans.checks.consistency.SamePluralsCheck',
#     'weblate.trans.checks.consistency.ConsistencyCheck',
#     'weblate.trans.checks.consistency.TranslatedCheck',
#     'weblate.trans.checks.chars.NewlineCountingCheck',
#     'weblate.trans.checks.markup.BBCodeCheck',
#     'weblate.trans.checks.chars.ZeroWidthSpaceCheck',
#     'weblate.trans.checks.markup.XMLValidityCheck',
#     'weblate.trans.checks.markup.XMLTagsCheck',
#     'weblate.trans.checks.source.OptionalPluralCheck',
#     'weblate.trans.checks.source.EllipsisCheck',
#     'weblate.trans.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 scripts to use in custom processing
# POST_UPDATE_SCRIPTS = (
# )
# PRE_COMMIT_SCRIPTS = (
# )
```

```
# 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 to use memcached for caching
# CACHES = {
#     'default': {
#         'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
#         'LOCATION': '127.0.0.1:11211',
#     },
#     'avatar': {
#         'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
#         'LOCATION': os.path.join(BASE_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.
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticatedOrReadOnly'
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.TokenAuthentication',
        '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.accounts.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/(.*)$', # Required for login
#     r'/static/(.*)$',   # 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
# )

# Force sane test runner
TEST_RUNNER = 'django.test.runner.DiscoverRunner'
```

Management commands

Note: Running management commands under different user than is running your webserver can cause wrong permissions on some files, please check *Filesystem permissions* for more details.

Django comes with 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.

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 istalled 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 <container> weblate list_versions
```

With **`docker-compose`** this is quite similar, you just have to use **`docker-compose run`**:

```
docker-compose run <container> weblate list_versions
```

See also:

Running Weblate in the Docker, *Installing Weblate by pip*

add_suggestions

`manage.py add_suggesstions <project> <component> <language> <file>`

New in version 2.5.

Imports translation from the file as a suggestions to given translation. It skips translations which are 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
```

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.

Example:

```
./manage.py --user nijel --inconsistent --source phpmyadmin/master phpmyadmin 4-5 cs
```

See also:

Automatic translation

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 site name

checkgit

manage.py checkgit <project|project/component>

Prints current state of backend git repository.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

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.

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.

This is most useful if executed periodically from cron or similar tool:

```
./manage.py commit_pending --all
```

See also:

Running maintenance tasks, *COMMIT_PENDING_HOURS*

cleanuptrans

manage.py cleanuptrans

Cleanups orphaned checks and translation suggestions.

See also:

Running maintenance tasks

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.

--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`.

dumpuserdata

manage.py dumpuserdata <file.json>

Dumps userdata to file for later use by *importuserdata*

This is useful when migrating of merging Weblate instances.

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"
    },
    {
        "name": "Android",
        "filemask": "android/values-*/strings.xml",
        "template": "android/values/strings.xml",
        "repo": "weblate://test/test"
```

```
    }
]
```

See also:

import_project

import_project

manage.py import_project <project> <gitrepo> <branch> <filemask>

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 repository is searched for directories matching a double wildcard (**) in the <*filemask*>. Each of these is then added as a component, named after the matched directory. Existing components will be skipped.

--name-template TEMPLATE
: Customise the component's name, its parameter is a python formatting string, which will expect the match from <*filemask*>.

--base-file-template TEMPLATE
: Customize base file for monolingual translations.

--file-format FORMAT
: You can also specify file format to use (see *Supported 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.

--component-regexp REGEX
: You can override parsing of component name from matched files here. This is a regular expression which will be matched against file name (as matched by <*filemask*>) and has to contain named groups *name* and *language*. This can be also used for excluding files in case they do not match this expression. For example: `(?P<language>.*)/(?P<name>[^-]*)\.po`

--no-skip-duplicates
: By default the import does skip already existing projects. This is to allow repeated importing of same repository. However if you want to force importing additional components even if name or slug matches existing one, you can do it by passing `--no-skip-duplicates`. This is generally useful for components with long names, which will get truncated on import and many of them will get same name or slug.

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 \
    master \
    web-app/tgol-web-app/src/main/resources/i18n/**-I18N_*.properties
```

Example of more complex parsing of filenames to get correct component and language out of file name like `src/security/Numerous_security_holes_in_0.10.1.de.po`:

```
./manage.py import_project \
    --component-regexp 'wiki/src/security/(?P<name>.*)\.([^.]*)\.po$' \
    tails \
    git://git.tails.boum.org/tails master \
    'wiki/src/security/**.*.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*.

importuserdata

manage.py importuserdata <file.json>

Imports userdata from file created by *dumpuserdata*

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
```

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.

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@exemple.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.

list_versions

manage.py list_versions

Lists versions of Weblate dependencies.

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 languaguage.

You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

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 underlaying 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

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.

rebuild_index

manage.py rebuild_index <project|project/component>

Rebuilds index for fulltext search. This might be lengthy operation if you have huge set of translation units.

--clean
Removes all words from database prior updating.

--optimize
The index will not be processed again, only it's content will be optimized (removing stale entries and merging possibly split index files).

See also:

Fulltext search

update_index

manage.py update_index

Updates index for fulltext search when *OFFLOAD_INDEXING* is enabled.

It is recommended to run this frequently (eg. every 5 minutes) to have index uptodate.

See also:

Fulltext search, *Running maintenance tasks*, *Enable indexing offloading*

unlock_translation

manage.py unlock_translation <project|project/component>

Unnocks given component for translating. This is useful in case you want to do some maintenance on underlaying 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

setupgroups

manage.py setupgroups

Configures default groups and optionally assigns all users to default group.

--move
Assigns all users to the default group.

--no-privs-update
Disables update of existing groups (only adds new ones).

See also:

Access control

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).

updatechecks

manage.py updatechecks <project|project/component>

Updates all check for all units. 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.

updategit

manage.py updategit <project|project/component>

Fetches remote VCS repositories and updates internal cache.

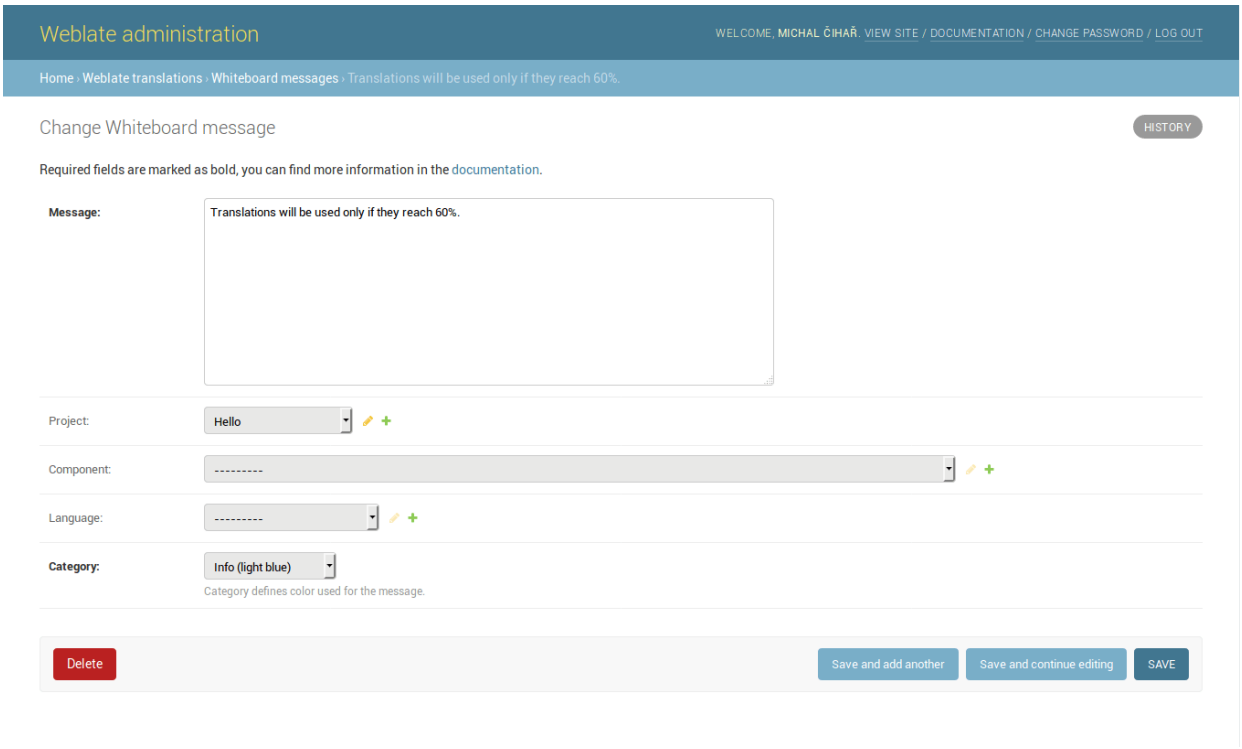
You can either define which project or component to update (eg. `weblate/master`) or use `--all` to update all existing components.

Whiteboard messages

You can use whiteboard messages to give some information to your translators. The message can be site-wide or targeted on translation component or language.

This can be useful for various things from announcing purpose of the website to specifying targets for translations.

The whiteboard can currently be specified only in the admin interface:



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.

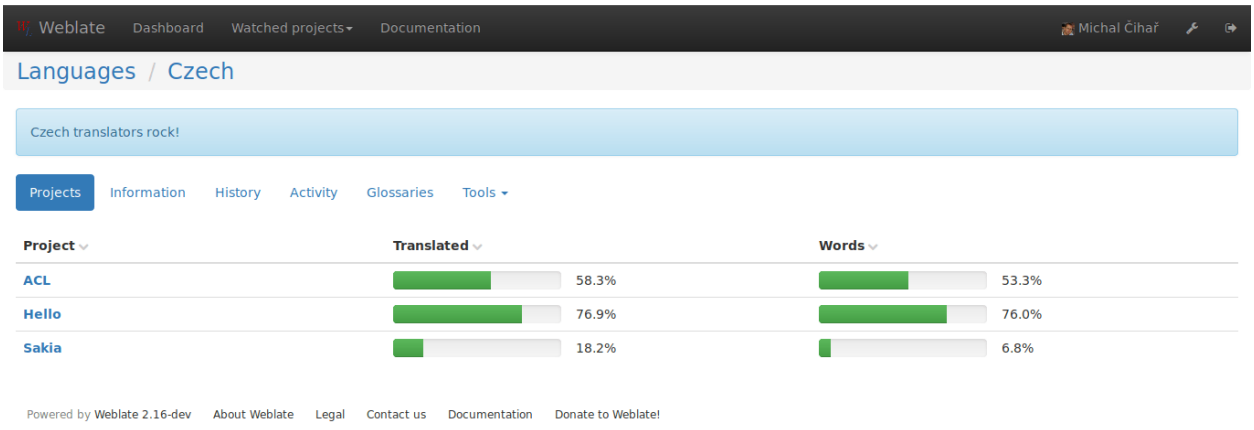
Component specified

Shown on component and all its translations.

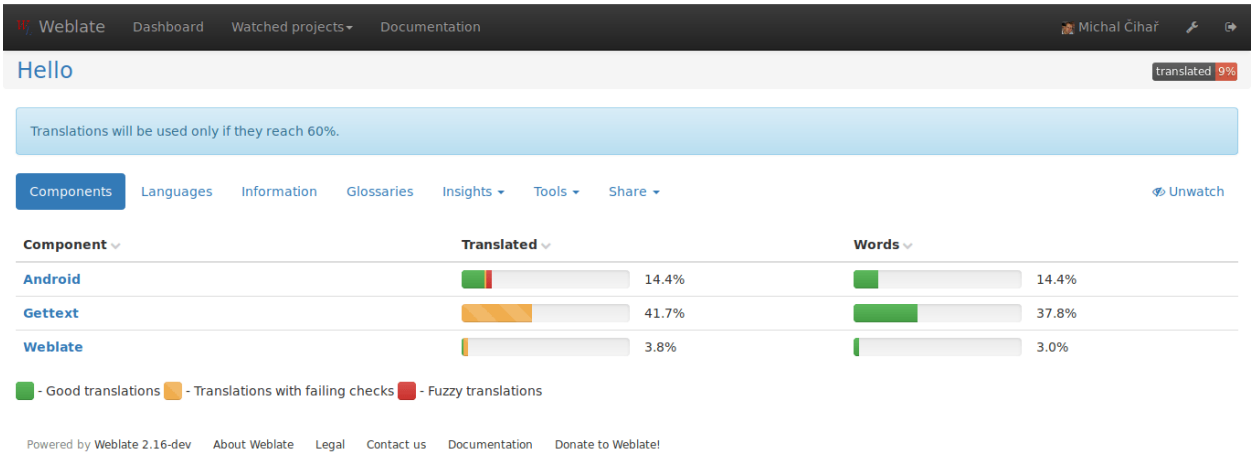
Language specified

Shown on language overview and all translations in this language.

You can see how it looks on the language overview page:



And on the project page:



Advertisement

Weblate allows you to place advertisements inside emails and web pages. You can add them in the admin interface, defining placement and timespan, when it should be shown.

Active advertisements are chosen randomly to be displayed inside defined placement.

When no advertisements are enabled, you can enable *SELF_ADVERTISEMENT* to show advertisements for Weblate.

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.

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.

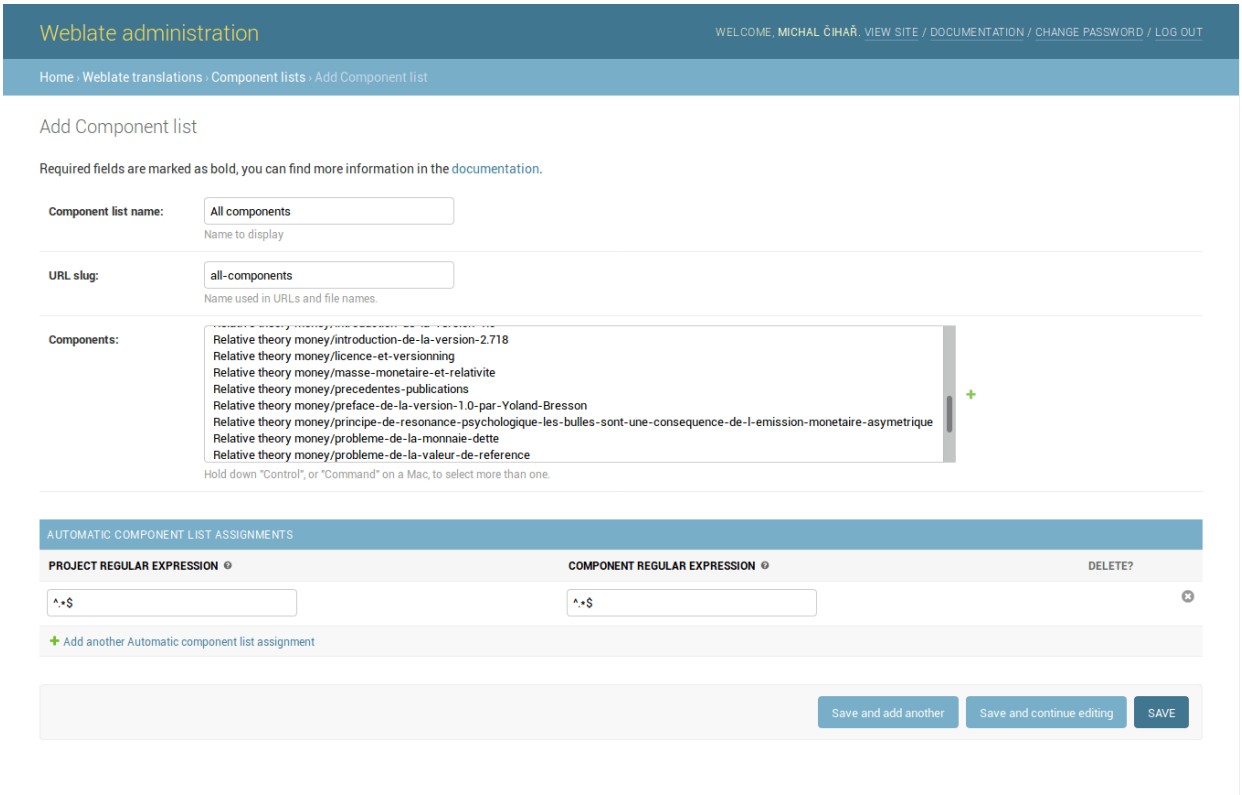
Note: Since version 2.13 you can also change the dashboard settings for the anonymous user in the admin interface, this will change what dashboard is visible to unauthenticated users.

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 expresion on both project and component as shown on following image:



Optional Weblate modules

Weblate comes with several optional modules which might be useful for your setup.

Git exporter

New in version 2.10.

The Git exporter provides you read only access to underlaying 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/'
```

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*.

Legal

New in version 2.15.

Legal module is used on Hosted Weblate and is used to provide required legal documents.

Note: The module ships legal documents for Hosted Weblate service. You are required to adjust the templates to match your use case.

Installation

To install, simply add `weblate.legal` to installed applications in `settings.py`:

```
INSTALLED_APPS += (
    'weblate.legal',
)
```

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.

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 two backends:

- Libravatar, what is federated avatar service with fallback to Gravatar. Libravatar is used automatically when pyLibravatar is installed.
- Gravatar can be also used directly by Weblate, that is used if the pyLibravatar library is not found.

See also:

Avatar caching, *ENABLE_AVATARS*

CHAPTER 5

Frequently Asked Questions

Configuration

How to create automatic workflow?

Weblate can handle all the translation things semi-automatically for you. If you will give it push access to your repository, the translations can live 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.
4. Optionally setup cron job for `commit_pending`.

See also:

Continuous translation, *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 time to time when the translation file is changed in both Weblate and upstream repository. You can usually avoid this by merging Weblate translations prior to doing some changes in the translation files (eg. before executing msgmerge). Just tell Weblate to commit all pending tranlslations (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 to the merge conflict, the easiest way is to solve all conflicts locally at your workstation - simply add Weblate as remote repository, merge it into upstream and fix conflicts. Once you push changes back, Weblate will be able to use merged version without any other special actions.

```
# Add weblate as remote
git remote add weblate https://hosted.weblate.org/git/weblate/master/

# 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 respository, Weblate will fetch merge from there
git push
```

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

```
# Add and update remotes
git remote add weblate-4.7 https://hosted.weblate.org/git/phpmyadmin/4-7/
git remote add weblate https://hosted.weblate.org/git/phpmyadmin/master/
git remote update weblate-4.7 weblate

# Merge QA_4_7 branch
git checkout QA_4_7
git merge weblate-4.7/QA_4_7
... # Resolve conflicts
git commit

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

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

See also:

How to export Git repository weblate uses?

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 Git repository weblate uses?

There is nothing special about the repository, it lives under *DATA_DIR* directory and is named as `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 git server and let it serve the repository to outside world.

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

What are options of 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 automatic workflow?*)
- You tell manually Weblate to push (it needs push access to upstream repository)
- Somebody manually merges changes from Weblate git repository into upstream repository
- Somebody rewrites history produced by Weblate (eg. by eliminating merge commits), merges changes and tells Weblate to reset contet on upstream repository.

Of course your 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 repository with your translation files.
2. Add this as 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 repository with your source code.
4. You can update the main repository by 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 set of configuration checks, which you can see in admin interface, just follow *Performace report* link in admin interface or directly open `/admin/performance/` URL.

Why does links contain example.com as domain?

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

See also:

Set correct site name

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

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

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

See also:

Component configuration

Why do I get warning about not reflected changes on database migration?

When running **`./manage.py migrate`**, you can get following warning:

```
Your models have changes that are not yet reflected in a migration, and so won't be␣
↪applied.
Run 'manage.py makemigrations' to make new migrations, and then re-run 'manage.py␣
↪migrate' to apply them.
```

This is expected as Weblate generates choices for some fields and Django migrations can not reflect this. You can safely ignore this warning.

Usage

How do I review others translations?

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

How do I provide feedback on source string?

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

How can I use existing translations while translating?

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

- You can use import functionality to load compendium as translations, suggestions or translations needing review. This is best approach for one time translation using compedium or similar translation database.

- You can setup *tmserver* with all databases you have and let Weblate use it. This is good for case when you want to use it for several times during translating.
- Another option is to translate all related projects in single Weblate instance, what will make it automatically pick up translation 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 minimum. For some file formats it might unfortunately lead to reformatting the file. If you want to keep the file formattted in your way, please use pre commit hook for that.

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

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 on your own.

See also:

Processing repository with scripts

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

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 language in administrative interface, you just need to provide information about it.

Can Weblate highlight change in a fuzzy string?

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

For Gettext PO files, you have to pass 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 shows old translation strings when I've updated the template?

Weblate does not try to manipulate with the translation files in any other way than allowing translators to translate. So it also does not update the translatable files when the template or source code has 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 good idea to merge changed 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 can want to do the update automatically, you can add custom script to handle this to *POST_UPDATE_SCRIPTS* and enable it in the *Component configuration*.

Troubleshooting

Requests sometimes fail with too many open files error

This happens sometimes when your Git repository grows too much and you have more 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), fulltext index might get too fragmented over time. It is recommended to optimize it time to time:

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

In case it does not help (or if you have removed 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 fulltext index. In case you are running multi threaded server (eg. mod_wsgi), this happens quite often. For such setup it is recommended to enable *OFFLOAD_INDEXING*.

See also:

Fulltext search

Rebuilding index has failed with "No space left on device"

Whoosh uses temporary directory to build indices. In case you have small /tmp (eg. using ramdisk), this might fail. Change used temporary directory by passing 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 with 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 powerful database engine* for more information.

See also:

Use powerful database engine, Databases

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

This is most likely caused by not properly 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

Features

Does Weblate support other VCS than Git and Mercurial?

Weblate currently does not have native support for anything else than *Git* (with extended support for *GitHub* 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 else 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 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 translation file format supports it, the file headers are updated to include translator name.

See also:

list_translators

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

Weblate was designed in a way that every po file is represented as single component. This is beneficial for translators, that 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.

CHAPTER 6

Supported formats

Weblate supports any translation format understood by Translate-toolkit, however each format being slightly different, there might be some issues with not well tested formats.

See also:

Supported formats in translate-toolkit

Note: When choosing file format for your application, it's better to stick some well established format in 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.

Weblate does support both monolingual and bilingual formats. Bilingual formats store two languages in single file - source and translation (typical examples is *GNU Gettext*, *XLIFF* or *Apple OS X 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 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.

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).

GNU Gettext

Most widely used format in translating free software. This was first format supported by Weblate and still has 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"
```

See also:

Gettext on Wikipedia, Gettext in translate-toolkit documentation

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"

#: 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"
```

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.

Note: If the translation unit doesn't have `approved="yes"` it will be imported into Weblate as needing review (what matches XLIFF specification).

You can override this by adding `skip-review-flag` flag to the component, see *Component configuration*, what will make Weblate ignore this and all strings will appear as approved.

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

See also:

XLIFF on Wikipedia, XLIFF in translate-toolkit documentation

Java properties

Native Java format for translations.

Java properties are usually used as monolingual.

This format supports creating new languages. When a new languages is created, a new empty file will be added to the repository. Only keys that are defined will be written to the newly created file. The Weblate maintainer needs to make sure that this is the expected behaviour with the framework in use.

Weblate supports ISO-8859-1, UTF-8 and UTF-16 variants of this format.

See also:

Java properties on Wikipedia, Java properties in translate-toolkit documentation

Joomla translations

New in version 2.12.

Native Joomla format for translations.

Joomla translation are usually used as monolingual.

This format supports creating new languages. When a new languages is created, a new empty file will be added to the repository. Only keys that are defined will be written to the newly created file. This should work fine since Joomla 3.0.

Note: You need translate-toolkit 2.1.0 or newer for Joomla support.

See also:

Specification of Joomla language files, Properties in translate-toolkit documentation

Qt Linguist .ts

Translation format used in Qt based applications.

Qt Linguist files are used as both bilingual and monolingual.

See also:

Qt Linguist manual, Qt .ts in translate-toolkit documentation

Android string resources

Android specific file format for translating applications.

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

See also:

Android string resources documentation, Android string resources in translate-toolkit documentation

---

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

Apple OS X strings

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

Apple OS X strings are usually used as bilingual.

See also:

Apple Strings Files documentation, Apple strings in translate-toolkit documentation

Note: You need translate-toolkit 1.12.0 or newer for proper support of Apple OS X strings. Older versions might produce corrupted files.

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';
```

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

See also:

PHP files in translate-toolkit documentation

JSON and nested structure 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 is format used mostly for translating applications implemented in Javascript.

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 as:

```
{
    "weblate": {
        "hello": "Ahoj světe!\n",
```

```
        "orangutan": "",
        "try": "",
        "thanks": ""
    }
}
```

See also:

JSON in translate-toolkit documentation

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 by 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"
    }
}
```

See also:

Google chrome.i18n, Mozilla Extensions Internationalization

.Net Resource files

New in version 2.3.

.Net Resource (.resx) file is a monolingual XML file format used in Microsoft .Net Applications.

Note: You need translate-toolkit 1.13.0 or newer to include support for this format.

CSV files

New in version 2.4.

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

- Files with header defining fields (source, translation, location, ...)
- Files with two fileld - 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

Example file:

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

YAML files

New in version 2.9.

There are several variants of using YAML as a translation format. Weblate currently supports following:

- Plain YAML files with string keys and values
- Ruby i18n YAML files with language as root node

Note: You currently need patched version of translate-toolkit to support YAML. Check translate-toolkit issue tracker for more details.

Example YAML file:

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

Example Ruby i18n YAML file:

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

Others

As already mentioned, all Translate-toolkit formats are supported, but they did not (yet) receive deeper testing.

See also:

Supported formats in translate-toolkit

CHAPTER 7

Version control integration

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

Accessing repositories

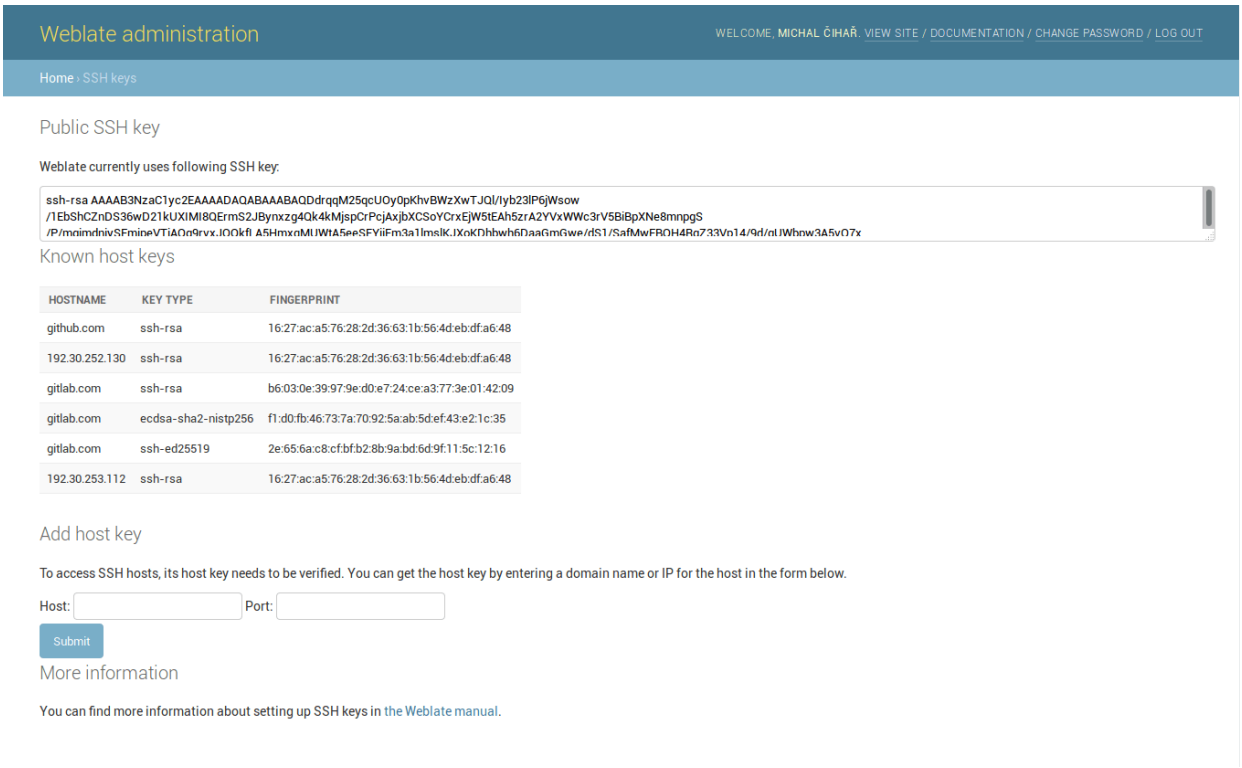
The VCS repository you want to use has to be accessible to Weblate. With publicly available repository you just need to enter correct URL (for example `git://github.com/WeblateOrg/weblate.git` or `https://github.com/WeblateOrg/weblate.git`), but for private repositories the setup might be more complex.

Weblate internal URLs

To share one repository between different components you can use 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

Most frequently used method to access private repositories is based on SSH. To have access to such 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:



Generating SSH keys

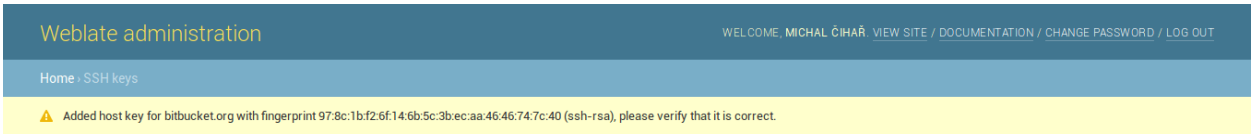
You can generate or display key currently used by Weblate in the admin interface (follow *SSH keys* link on main admin page). Once you've done this, Weblate should be able to access your repository.

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 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:



HTTPS repositories

To access protected HTTPS repositories, you need to include user and password in the URL. Don't worry, Weblate will strip this information when showing 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`.

Using proxy

If you need to access http/https VCS repositories using a proxy server, you need to configure 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 context which is used to execute Weblate. For the environment it should be set for both server and cron jobs. The VCS configuration has to be set for the user which is running Weblate.

See also:

curl manpage, git config documentation

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 GitHub limitation on the SSH key usage (one key can be used only for one repository). There are several ways to workaround this limitation.

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

For 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.

Git remote helpers

You can also use Git remote helpers for supporting other VCS as well, but this usually leads to smaller or bigger 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 incovenience when using Git remote helpers, for example with Mercurial, the remote helper sometimes tends to create new tip when pushing changes back.

GitHub

New in version 2.3.

This just adds 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 same, the only difference is how pushing to repository is handled. With *Git* changes are pushed directly to the repository, while *GitHub* creates pull requests.

Note: This feature is currently not available on Hosted Weblate due to technical limitations. See *Pushing changes from Hosted Weblate* for available options.

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:

```
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`.

Note: Use the username you configured *hub* with as *GITHUB_USERNAME*.

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.

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 supports only subversion repositories with standard layout (branches/, tags/ and trunk/). See git-svn documentation for more information.

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

CHAPTER 8

Weblate's Web API

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 is using 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
Autorization: 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/"
}
```

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/
```

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
Autorization: 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 listing 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 language.

Parameters

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

Response JSON Object

- **code** (*string*) – Language code
- **direction** (*string*) – Text direction
- **nplurals** (*int*) – Number of plurals
- **pluralequation** (*string*) – Gettext plural equation

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",
    "nplurals": 2,
    "pluralequation": "n != 1",
    "url": "http://example.com/api/languages/en/",
    "web_url": "http://example.com/languages/en/"
}
```

Projects

GET /api/projects/
Returns listing of 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 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",
        "nplurals": 2,
        "pluralequation": "n != 1",
        "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 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 overall summary for all repositories for 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`

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 list of translation components in 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*) **/**
string: *component***/statistics/** Returns paginated statistics for all languages within project.

New in version 2.10.

Parameters

- **project** (*string*) – Project URL slug
- **component** (*string*) – Component 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 listin 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",
            "nplurals": 2,
            "pluralequation": "n != 1",
            "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/",
    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
}
```


GET **/api/components/** (**string:** *project*) **/** **string:** *component***/changes/** Returns 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, 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 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

Request JSON Object

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

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 list of translation objects in 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 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 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 units with failing check
- **failing_checks_percent** (*float*) – percetage of failing check units

- **failing_checks_words** (*int*) – number of words with failing check
- **filename** (*string*) – translation filename
- **fuzzy** (*int*) – number of units marked for review
- **fuzzy_percent** (*float*) – percetage of units marked for review
- **fuzzy_words** (*int*) – number of words marked for review
- **have_comment** (*int*) – number of units with comment
- **have_suggestion** (*int*) – number of units 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 units
- **total_words** (*int*) – total number of words
- **translate_url** (*string*) – URL for translating
- **translated** (*int*) – number of translated units
- **translated_percent** (*float*) – percentage of translated units
- **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 units 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",
                "nplurals": 2,
                "pluralequation": "n != 1",
                "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": {
        "code": "cs",
        "direction": "ltr",
        "name": "Czech",
        "nplurals": 3,
        "pluralequation": "(n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2",
        "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 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 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 converted to one of standard formats (currently supported are Gettext PO, MO, Xliff or 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` parameters differ 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`, `xliff12`, `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

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`

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
- **souce_info** (*string*) – Source string information link, see *GET /api/sources/(int:pk)/*

Changes

New in version 2.10.

GET /api/changes/
Returns 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 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 list of screenshot string informations.

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 asssociated source string information, see *GET /api/sources/(int:pk)/*

GET /api/screenshots/ (**int:** *pk*) **/file/**
Download the screenshots 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/
```

Notification hooks

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

You can use repository endpoints for project, component and translation 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, just enable Weblate service hook in repository settings and set URL to 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

Exports

Weblate provides various exports to allow you 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 to access 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 \u010diha\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,
        "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/"
    },
]
```


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

CHAPTER 9

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 API might occur.

Installation

The Weblate Client is shipped separately, you need to install wlc to have it, it also includes Python module *wlc*:

```
pip3 install wlc
```

Synopsis

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

Commands actually indicate which operation should be performed.

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 build on *wlc*.

Global options

The program accepts 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 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).

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 compontent from translating in Weblate.

unlock
New in version 0.5: Supported since wlc 0.5.

Unlocks compontent 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.

Files

.weblate Per project configuration file

~/.config/weblate User configration file

/etc/xdg/weblate Global configration 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.

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
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
```

CHAPTER 10

Weblate's Python API

Instalation

The Python API is shipped separately, you need to install wlc to have it, it also include *Weblate Client*:

```
pip install wlc
```

wlc

WeblateException

exception wlc.**WeblateException**
Base class for all exceptions.

Weblate

class wlc.**Weblate**(*key=''*, *url=None*, *config=None*)

Parameters

- **key** (*string*) – User key
- **url** (*string*) – API server URL, if not specified default is used
- **config** (WeblateConfig) – Configuration object, overrides any other parameters.

Access class to the API, define API key and optionally API URL.

get(*path*)

Parameters **path** (*string*) – Request path

Return type object

Performs single API GET call.

post (*path*, ***kwargs*)

Parameters **path** (*string*) – Request path

Return type object

Performs single API GET call.

wlc.config

WeblateConfig

class wlc.config.**WeblateConfig** (*section='wlc'*)

Parameters **section** (*string*) – Configuration section to use

Configuration file parser following XDG specification.

load (*path=None*)

Parameters **path** (*string*) – Path where 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`).

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 argumets 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.

CHAPTER 11

Changes

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.

weblate 2.15

Released on June 30th 2017.

- Show more related translations in other translations.
- Add option to see translations of current unit 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.

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.

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 imporant 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.

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.

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 ALCs are now implemented using Group ACL.
- Added more fine grained privileges control.
- Various minor UI improvements.

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.

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.

weblate 2.10.1

Released on Jan 20th 2017.

- Do not leak account existence on password reset form (CVE-2017-5537).

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 units.
- Added support for Apertium APy server for machine translations.

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.

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.

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 traslations.
- Show full suggestion text, not only a diff.
- Extend API (detailed repository status, statistics, ...).
- Testsuite no longer requires network access to test repositories.

weblate 2.6

Released on Apr 28th 2016.

- Fixed validation of subprojects 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.

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 transations 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 erorrs.
- 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.

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 descriping 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 confiugure default committer.
- Add hook after adding new translation.
- Add option to specify multiple files to add to commit.

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.

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 file names.
- 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.

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.

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 possibilites.
- 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.

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.

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 units.
- Better tracking of source strings.
- Captcha protection for registration.

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 units.
- 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.

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.

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 commiting 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.

weblate 1.4

Released on January 23rd 2013.

- Fixed deleting of checks/comments on unit 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 unit tests coverage.

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.

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 units with failing checks.
- Checking of requirements during setup.
- Documentation improvements.

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.

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.

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.

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.

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.

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.

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).

weblate 0.4

Released on February 8th 2012.

- Added usage guide to documentation.
- Fixed API hooks not to require CSRF protection.

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.

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.

weblate 0.1

Released on February 6th 2012.

- Initial release.

CHAPTER 12

Contributing

There are dozens of ways to contribute to Weblate. We welcome any help, be it coding help, graphics design, documentation or sponsorship.

Code and development

Weblate is being developed on GitHub <https://github.com/WeblateOrg/weblate>. You are welcome to fork the code and open pull requests. Patches in any other form are welcome as well.

See also:

Check out *Internals* to see how Weblate looks from inside.

Coding standard

The code should follow PEP-8 coding guidelines.

It is good idea to check your contributions using **`pep8`**, **`pylint`** and **`pyflages`**. You can execute all checks by script `ci/run-lint`.

Developer's Certificate of Origin

If you would like to make a contribution 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:
>
> 1. 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

2. 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
3. The contribution was provided directly to me by some other person who certified (a) or (b) and I have not modified it.
4. 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.
5. 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

And please confirm your certification to 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 who is authorized to contribute to Weblate on behalf of your employer, then please use your corporate email address in the Signed-off-by tag. If not, then please use a personal email address.

Testsuite

We do write testsuite for our code, so please add testcases for any new functionality and verify that it works. You can see current test results on Travis <https://travis-ci.org/WeblateOrg/weblate> and coverage on Codecov <https://codecov.io/github/WeblateOrg/weblate>.

To run testsuite locally use:

```
./manage.py test --settings weblate.settings_test
```

You can also specify individual tests to run:

```
./manage.py test --settings weblate.settings_test weblate.gitexport
```

See also:

See Testing in Django for more information on running and writing tests for Django.

Reporting issues

Our issue tracker is hosted at GitHub: <https://github.com/WeblateOrg/weblate/issues>

Feel welcome to report any issues or suggestions to improve Weblate there. In case you have found security issue in Weblate, please consult section below.

Security issues

In order to give the community time to respond and upgrade we strongly urge you report all security issues privately. We're currently using HackerOne to handle security issues, so you are welcome to report issues directly at <https://hackerone.com/weblate>.

Alternatively you can report them 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 his PGP key *9C27B31342B7511D*.

Note: We're heavily depending on third party components for many things. In case you find vulnerability which is affecting those components in general, please report it directly to them.

See following sites for some of these:

- Django
- Django REST Framework
- Python Social Auth

Starting with our codebase

If you are looking for some bugs which should be good for starting with our codebase, you can find them labelled with *newbie* tag:

https://github.com/WeblateOrg/weblate/labels/newbie

Earning money by coding

We're using Bountysource to fund our development, you can participate on this as well by implementing issues with bounties:

https://github.com/WeblateOrg/weblate/labels/bounty

Translating

Weblate is being translated using Weblate on <https://hosted.weblate.org/>, feel free to join us in effort to make Weblate available in as many world languages as possible.

Funding Weblate development

You can fund further Weblate development on Bountysource. Funds collected there are used to fund free hosting for free software projects and further development of Weblate. Please check the Bountysource page for details such as funding goals and rewards you can get for funding.

Backers who have funded Weblate

List of Weblate supporters from Bountysource:

- Yashiro Ccs
- Cheng-Chia Tseng
- Timon Reinhard

- Cassidy James

CHAPTER 13

Internals

Note: This chapter will give you basic overview of Weblate internals.

Weblate is based on Django and most of it's code structure comes from that. If you are not familiar with Django, you might want to check Django at a glance to get basic understanding of files structure.

Modules

Weblate consists of several Django applications (some of them are optional, see *Optional Weblate modules*):

`accounts`
: User account, profiles and notifications.

`api`
: API based on Django REST framework.

`billing`
: The optional *Billing* module.

`gitexport`
: The optional *Git exporter* module.

`lang`
: Module defining language parameters.

`legal`
: The optional *Legal* module.

`permissions`
: The *Group-based access control* code with various helpers.

`screenshots`

Screenshots management and OCR module.

`trans`

Main module handling translations.

`utils`

Various helper utilities.

CHAPTER 14

License

Copyright (C) 2012 - 2017 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/>.

CHAPTER 15

Indices and tables

- genindex
- search

HTTP Routing Table

/

ANY /, 181

/api

GET /api/, 183
GET /api/changes/, 198
GET /api/changes/(int:pk)/, 199
GET /api/components/, 188
GET /api/components/(string:project)/(string:component)/, 188
GET /api/components/(string:project)/(string:component)/changes/, 189
GET /api/components/(string:project)/(string:component)/lock/, 190
GET /api/components/(string:project)/(string:component)/monolingual_base/, 191
GET /api/components/(string:project)/(string:component)/new_template/, 191
GET /api/components/(string:project)/(string:component)/repository/, 190
GET /api/components/(string:project)/(string:component)/statistics/, 192
GET /api/components/(string:project)/(string:component)/translations/, 191
GET /api/languages/, 184
GET /api/languages/(string:language)/, 184
GET /api/projects/, 185
GET /api/projects/(string:project)/, 185
GET /api/projects/(string:project)/changes/, 185
GET /api/projects/(string:project)/components/, 187
GET /api/projects/(string:project)/repository/, 186
GET /api/screenshots/, 200
GET /api/screenshots/(int:pk)/, 200
GET /api/screenshots/(int:pk)/file/, 200
GET /api/sources/, 199
GET /api/sources/(int:pk)/, 199
GET /api/translations/, 192
GET /api/translations/(string:project)/(string:component)/(string:language)/, 192
GET /api/translations/(string:project)/(string:component)/(string:language)/changes/, 195
GET /api/translations/(string:project)/(string:component)/(string:language)/file/, 195
GET /api/translations/(string:project)/(string:component)/(string:language)/units/, 196
GET /api/translations/(string:project)/(string:component)/(string:language)/units/(int:pk)/, 197
GET /api/translations/(string:project)/(string:component)/(string:language)/units/(int:pk)/, 198
POST /api/components/(string:project)/(string:component)/new_template/, 190
POST /api/components/(string:project)/(string:component)/repository/, 191
POST /api/projects/(string:project)/repository/, 186
POST /api/screenshots/(int:pk)/file/, 200
POST /api/translations/(string:project)/(string:component)/(string:language)/, 196
POST /api/translations/(string:project)/(string:component)/(string:language)/file/, 196

/exports

GET /exports/rss/, 203
GET /exports/rss/(string:project)/, 203
GET /exports/rss/(string:project)/(string:component)/, 203
GET /exports/rss/(string:project)/(string:component)/(string:language)/, 203
GET /exports/rss/language/(string:language)/, 203
GET /exports/stats/(string:project)/(string:component)/, 202

/hooks

GET /hooks/update/(string:project)/, 201
GET /hooks/update/(string:project)/(string:component)/, 201
POST /hooks/bitbucket/, 201
POST /hooks/github/, 201
POST /hooks/gitlab/, 201

Python Module Index

W

wlc, 209
wlc.config, 210
wlc.main, 210

Index

Symbols

–add
auto_translate command line option, 146
–age HOURS
commit_pending command line option, 147
EXAMPLE.COM
add_suggesstions command line option, 146
–base-file-template TEMPLATE
import_project command line option, 149
–check
importusers command line option, 150
–clean
rebuild_index command line option, 152
–component-regexp REGEX
import_project command line option, 149
–config PATH
wlc command line option, 206
–config-section SECTION
wlc command line option, 206
–convert
wlc command line option, 207
EXAMPLE.COM
createadmin command line option, 148
–file-format FORMAT
import_project command line option, 149
–force
loadpo command line option, 151
–force-commit
pushgit command line option, 152
–format {csv,json,text,html}
wlc command line option, 205
–get-name
changesite command line option, 146
–ignore
import_json command line option, 148
–inconsistent
auto_translate command line option, 146
–key KEY
wlc command line option, 206
–lang LANGUAGE
loadpo command line option, 151
–language-code
list_translators command line option, 151
–language-regex REGEX
import_project command line option, 149
–license NAME
import_project command line option, 149
–license-url URL
import_project command line option, 149
–main-component
import_project command line option, 149
–main-component COMPONENT
import_json command line option, 148
–move
setupgroups command line option, 153
–name
createadmin command line option, 148
–name-template TEMPLATE
import_project command line option, 149
–no-privs-update
setupgroups command line option, 153
–no-skip-duplicates
import_project command line option, 149
–no-update
setuplang command line option, 153
–optimize
rebuild_index command line option, 152
–output
wlc command line option, 207
–overwrite
auto_translate command line option, 146
–password PASSWORD
createadmin command line option, 147
–project PROJECT
import_json command line option, 148
–set-name NAME
changesite command line option, 146
–source PROJECT/COMPONENT
auto_translate command line option, 146

–update
createadmin command line option, 148
import_json command line option, 148
–url URL
wlc command line option, 205
–user USERNAME
auto_translate command line option, 146
–username USERNAME
createadmin command line option, 147
–vcs NAME
import_project command line option, 149
.Net Resource
file format, 172

A

add_suggesstions
django-admin command, 145
add_suggesstions command line option
EXAMPLE.COM, 146
ADMINS
setting, 47
ALLOWED_HOSTS
setting, 47
Android
file format, 170
ANONYMOUS_USER_NAME
setting, 120
API, 181, 203, 208
Apple strings
file format, 170
AUTH_CHECK_WINDOW
setting, 121
AUTH_LOCK_ATTEMPTS
setting, 121
AUTH_LOCKOUT_TIME
setting, 121
AUTH_MAX_ATTEMPTS
setting, 121
AUTH_TOKEN_VALID
setting, 122
AUTO_LOCK
setting, 122
AUTO_LOCK_TIME
setting, 122
auto_translate
django-admin command, 146
auto_translate command line option
–add, 146
–inconsistent, 146
–overwrite, 146
–source PROJECT/COMPONENT, 146
–user USERNAME, 146
AUTOFIX_LIST
setting, 122

B

BACKGROUND_HOOKS
setting, 123
bilingual
translation, 167

C

changes
wlc command line option, 207
changesite
django-admin command, 146
changesite command line option
–get-name, 146
–set-name NAME, 146
CHECK_LIST
setting, 123
checkgit
django-admin command, 147
cleanuptrans
django-admin command, 147
Comma separated values
file format, 172
Command (class in wlc.main), 210
commit
wlc command line option, 206
commit_pending
django-admin command, 147
commit_pending command line option
–age HOURS, 147
COMMIT_PENDING_HOURS
setting, 124
commitgit
django-admin command, 147
createadmin
django-admin command, 147
createadmin command line option
EXAMPLE.COM, 148
–name, 148
–password PASSWORD, 147
–update, 148
–username USERNAME, 147
CSV
file format, 172

D

DATA_DIR
setting, 124
DATABASES
setting, 48
DEBUG
setting, 48
DEFAULT_COMMITER_EMAIL
setting, 124
DEFAULT_COMMITER_NAME

setting, 124
DEFAULT_FROM_EMAIL
setting, 48
DEFAULT_TRANSLATION_PROPAGATION
setting, 125
django-admin command
add_suggesstions, 145
auto_translate, 146
changesite, 146
checkgit, 147
cleanuptrans, 147
commit_pending, 147
commitgit, 147
createadmin, 147
dumpuserdata, 148
import_json, 148
import_project, 149
importuserdata, 150
importusers, 150
list_ignored_checks, 151
list_translators, 151
list_versions, 151
loadpo, 151
lock_translation, 152
pushgit, 152
rebuild_index, 152
setupgroups, 153
setuplang, 153
unlock_translation, 153
update_index, 152
updatechecks, 153
updategit, 153
download
wlc command line option, 207
dumpuserdata
django-admin command, 148

E

ENABLE_AVATARS
setting, 125
ENABLE_HOOKS
setting, 125
ENABLE_HTTPS
setting, 125
ENABLE_SHARING
setting, 125
environment variable
POSTGRES_DATABASE, 63
POSTGRES_HOST, 63
POSTGRES_PASSWORD, 63
POSTGRES_PORT, 63
POSTGRES_USER, 63
WEBLATE_ADMIN_EMAIL, 61
WEBLATE_ADMIN_NAME, 61
WEBLATE_ADMIN_PASSWORD, 60, 61
WEBLATE_ALLOWED_HOSTS, 61
WEBLATE_DEBUG, 60
WEBLATE_DEFAULT_FROM_EMAIL, 61
WEBLATE_EMAIL_HOST, 64
WEBLATE_EMAIL_HOST_PASSWORD, 64
WEBLATE_EMAIL_HOST_USER, 64
WEBLATE_EMAIL_PORT, 64
WEBLATE_ENABLE_HTTPS, 62
WEBLATE_GITHUB_USERNAME, 62
WEBLATE_GOOGLE_ANALYTICS_ID, 62
WEBLATE_LOGLEVEL, 61
WEBLATE_MT_GOOGLE_KEY, 62
WEBLATE_MT_MICROSOFT_COGNITIVE_KEY, 62
WEBLATE_OFFLOAD_INDEXING, 60, 61
WEBLATE_POST_ADD_SCRIPTS, 63
WEBLATE_POST_COMMIT_SCRIPTS, 63
WEBLATE_POST_PUSH_SCRIPTS, 63
WEBLATE_POST_UPDATE_SCRIPTS, 63
WEBLATE_PRE_COMMIT_SCRIPTS, 63
WEBLATE_REGISTRATION_OPEN, 61
WEBLATE_REQUIRE_LOGIN, 62
WEBLATE_SECRET_KEY, 61
WEBLATE_SERVER_EMAIL, 61
WEBLATE_SITE_TITLE, 61
WEBLATE_SOCIAL_AUTH_BITBUCKET_KEY, 62
WEBLATE_SOCIAL_AUTH_BITBUCKET_SECRET, 62
WEBLATE_SOCIAL_AUTH_FACEBOOK_KEY, 62
WEBLATE_SOCIAL_AUTH_FACEBOOK_SECRET, 62
WEBLATE_SOCIAL_AUTH_GITHUB_KEY, 62
WEBLATE_SOCIAL_AUTH_GITHUB_SECRET, 62
WEBLATE_SOCIAL_AUTH_GITLAB_API_URL, 63
WEBLATE_SOCIAL_AUTH_GITLAB_KEY, 62
WEBLATE_SOCIAL_AUTH_GITLAB_SECRET, 63
WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY, 62
WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET, 62
WEBLATE_TIME_ZONE, 61
WL_BRANCH, 104
WL_FILE_FORMAT, 104
WL_FILEMASK, 104
WL_LANGUAGE, 104
WL_NEW_BASE, 104
WL_PATH, 104
WL_PREVIOUS_HEAD, 104

WL_REPO, 104
WL_TEMPLATE, 104
WL_VCS, 104

F

file format
.Net Resource, 172
Android, 170
Apple strings, 170
Comma separated values, 172
CSV, 172
Gettext, 167
Java properties, 169
Joomla translations, 169
JSON, 171
PHP strings, 171
PO, 167
Qt, 170
RESX, 172
string resources, 170
TS, 170
XLIFF, 169
YAML, 173
YAML Ain't Markup Language, 173

G

get() (wlc.Weblate method), 209
Gettext
file format, 167
GIT_ROOT
setting, 125
GITHUB_USERNAME
setting, 125
GOOGLE_ANALYTICS_ID
setting, 126

H

HIDE_REPO_CREDENTIALS
setting, 126

I

import_json
django-admin command, 148
import_json command line option
–ignore, 148
–main-component COMPONENT, 148
–project PROJECT, 148
–update, 148
import_project
django-admin command, 149
import_project command line option
–base-file-template TEMPLATE, 149
–component-regexp REGEX, 149
–file-format FORMAT, 149
–language-regex REGEX, 149
–license NAME, 149
–license-url URL, 149
–main-component, 149
–name-template TEMPLATE, 149
–no-skip-duplicates, 149
–vcs NAME, 149
importuserdata
django-admin command, 150
importusers
django-admin command, 150
importusers command line option
–check, 150
IP_BEHIND_REVERSE_PROXY
setting, 126
IP_PROXY_HEADER
setting, 126
IP_PROXY_OFFSET
setting, 126
iPad
translation, 170
iPhone
translation, 170

J

Java properties
file format, 169
Joomla translations
file format, 169
JSON
file format, 171

L

LAZY_COMMITS
setting, 127
list-components
wlc command line option, 206
list-languages
wlc command line option, 206
list-projects
wlc command line option, 206
list-translations
wlc command line option, 206
list_ignored_checks
django-admin command, 151
list_translators
django-admin command, 151
list_translators command line option
–language-code, 151
list_versions
django-admin command, 151
load() (wlc.config.WeblateConfig method), 210
loadpo

django-admin command, 151
loadpo command line option
–force, 151
–lang LANGUAGE, 151
lock
wlc command line option, 207
lock-status
wlc command line option, 206
LOCK_TIME
setting, 127
lock_translation
django-admin command, 152
LOGIN_REQUIRED_URLS
setting, 127
LOGIN_REQUIRED_URLS_EXCEPTIONS
setting, 127
ls
wlc command line option, 206

M

MACHINE_TRANSLATION_SERVICES
setting, 127
main() (in module wlc.main), 210
monolingual
translation, 167
MT_APERTIUM_APY
setting, 128
MT_APERTIUM_KEY
setting, 128
MT_GOOGLE_KEY
setting, 128
MT_MICROSOFT_COGNITIVE_KEY
setting, 128
MT_MICROSOFT_ID
setting, 128
MT_MICROSOFT_SECRET
setting, 128
MT_MYMEMORY_EMAIL
setting, 129
MT_MYMEMORY_KEY
setting, 129
MT_MYMEMORY_USER
setting, 129
MT_TMSERVER
setting, 129
MT_YANDEX_KEY
setting, 129

N

NEARBY_MESSAGES
setting, 129

O

OFFLOAD_INDEXING
setting, 129

P

PHP strings
file format, 171
PIWIK_SITE_ID
setting, 130
PIWIK_URL
setting, 130
PO
file format, 167
post() (wlc.Weblate method), 210
POST_ADD_SCRIPTS
setting, 130
POST_COMMIT_SCRIPTS
setting, 131
POST_PUSH_SCRIPTS
setting, 131
POST_UPDATE_SCRIPTS
setting, 130
PRE_COMMIT_SCRIPTS
setting, 131
pull
wlc command line option, 206
push
wlc command line option, 206
pushgit
django-admin command, 152
pushgit command line option
–force-commit, 152
Python, 208

Q

Qt
file format, 170

R

rebuild_index
django-admin command, 152
rebuild_index command line option
–clean, 152
–optimize, 152
register_command() (in module wlc.main), 210
REGISTRATION_CAPTCHA
setting, 131
REGISTRATION_OPEN
setting, 132
repo
wlc command line option, 206
reset
wlc command line option, 206
REST, 181
RESX
file format, 172

S

SECRET_KEY
  setting, 48
SELF_ADVERTISEMENT
  setting, 132
SERVER_EMAIL
  setting, 48
SESSION_ENGINE
  setting, 48
setting
  ADMINS, 47
  ALLOWED_HOSTS, 47
  ANONYMOUS_USER_NAME, 120
  AUTH_CHECK_WINDOW, 121
  AUTH_LOCK_ATTEMPTS, 121
  AUTH_LOCKOUT_TIME, 121
  AUTH_MAX_ATTEMPTS, 121
  AUTH_TOKEN_VALID, 122
  AUTO_LOCK, 122
  AUTO_LOCK_TIME, 122
  AUTOFIX_LIST, 122
  BACKGROUND_HOOKS, 123
  CHECK_LIST, 123
  COMMIT_PENDING_HOURS, 124
  DATA_DIR, 124
  DATABASES, 48
  DEBUG, 48
  DEFAULT_COMMITER_EMAIL, 124
  DEFAULT_COMMITER_NAME, 124
  DEFAULT_FROM_EMAIL, 48
  DEFAULT_TRANSLATION_PROPAGATION, 125
  ENABLE_AVATARS, 125
  ENABLE_HOOKS, 125
  ENABLE_HTTPS, 125
  ENABLE_SHARING, 125
  GIT_ROOT, 125
  GITHUB_USERNAME, 125
  GOOGLE_ANALYTICS_ID, 126
  HIDE_REPO_CREDENTIALS, 126
  IP_BEHIND_REVERSE_PROXY, 126
  IP_PROXY_HEADER, 126
  IP_PROXY_OFFSET, 126
  LAZY_COMMITS, 127
  LOCK_TIME, 127
  LOGIN_REQUIRED_URLS, 127
  LOGIN_REQUIRED_URLS_EXCEPTIONS, 127
  MACHINE_TRANSLATION_SERVICES, 127
  MT_APERTIUM_APY, 128
  MT_APERTIUM_KEY, 128
  MT_GOOGLE_KEY, 128
  MT_MICROSOFT_COGNITIVE_KEY, 128
  MT_MICROSOFT_ID, 128
  MT_MICROSOFT_SECRET, 128
  MT_MYMEMORY_EMAIL, 129
  MT_MYMEMORY_KEY, 129
  MT_MYMEMORY_USER, 129
  MT_TMSERVER, 129
  MT_YANDEX_KEY, 129
  NEARBY_MESSAGES, 129
  OFFLOAD_INDEXING, 129
  PIWIK_SITE_ID, 130
  PIWIK_URL, 130
  POST_ADD_SCRIPTS, 130
  POST_COMMIT_SCRIPTS, 131
  POST_PUSH_SCRIPTS, 131
  POST_UPDATE_SCRIPTS, 130
  PRE_COMMIT_SCRIPTS, 131
  REGISTRATION_CAPTCHA, 131
  REGISTRATION_OPEN, 132
  SECRET_KEY, 48
  SELF_ADVERTISEMENT, 132
  SERVER_EMAIL, 48
  SESSION_ENGINE, 48
  SIMPLIFY_LANGUAGES, 132
  SITE_TITLE, 132
  TTF_PATH, 132
  URL_PREFIX, 132
  WHOOSH_INDEX, 132
setupgroups
  django-admin command, 153
setupgroups command line option
  –move, 153
  –no-privs-update, 153
setuplang
  django-admin command, 153
setuplang command line option
  –no-update, 153
show
  wlc command line option, 206
SIMPLIFY_LANGUAGES
  setting, 132
SITE_TITLE
  setting, 132
statistics
  wlc command line option, 206
string resources
  file format, 170

T

translation
  bilingual, 167
  iPad, 170
  iPhone, 170
  monolingual, 167
TS
  file format, 170
TTF_PATH
  setting, 132

U

unlock
  wlc command line option, 207
unlock_translation
  django-admin command, 153
update_index
  django-admin command, 152
updatechecks
  django-admin command, 153
updategit
  django-admin command, 153
URL_PREFIX
  setting, 132

V

version
  wlc command line option, 206

W

Weblate (class in wlc), 209
WEBLATE_ADMIN_EMAIL, 61
WEBLATE_ADMIN_NAME, 61
WEBLATE_ADMIN_PASSWORD, 60, 61
WEBLATE_OFFLOAD_INDEXING, 60
WeblateConfig (class in wlc.config), 210
WeblateException, 209
WHOOSH_INDEX
  setting, 132
wlc, 203
wlc (module), 209
wlc command line option
  –config PATH, 206
  –config-section SECTION, 206
  –convert, 207
  –format {csv,json,text,html}, 205
  –key KEY, 206
  –output, 207
  –url URL, 205
  changes, 207
  commit, 206
  download, 207
  list-components, 206
  list-languages, 206
  list-projects, 206
  list-translations, 206
  lock, 207
  lock-status, 206
  ls, 206
  pull, 206
  push, 206
  repo, 206
  reset, 206
  show, 206
  statistics, 206
  unlock, 207
  version, 206
wlc.config (module), 210
wlc.main (module), 210

X

XLIFF
  file format, 169

Y

YAML
  file format, 173
YAML Ain't Markup Language
  file format, 173