
Pootle Documentation

Release 2.9.0rc1

Pootle contributors

Apr 13, 2018

Contents

1	All you need to know	3
1.1	Getting started	3
1.2	Features	3
1.3	Installation	31
1.4	Upgrading	35
1.5	Administering a server	40
1.6	Developers	91
1.7	Frequently Asked Questions (FAQ)	126
2	Additional Notes	129
2.1	Release Notes	129
2.2	License	170

Pootle is an online tool that makes the process of translating so much simpler. It allows crowd-sourced translations, easy volunteer contribution and gives statistics about the ongoing work.

Pootle is built using the powerful API of the *Translate Toolkit* and the *Django framework*. If you want to know more about these, you can dive into their own documentation.

- [Translate Toolkit Documentation](#)
- [Django Documentation](#)

All you need to know

The following pages cover the documentation of Pootle from a wide variety of perspectives, server administrator's, and developer's view.

1.1 Getting started

1.1.1 Setting up the docker environment

1.1.2 Running the demo

1.1.3 Setting up a development environment

1.2 Features

1.2.1 Pootle FS

Pootle FS

Pootle FS is Pootle's integration with version control plugin systems. It allows Pootle to synchronize with an external repository containing your translations, keep them synchronised and manage and resolve any conflicts either automatically or via user input.

Aims

- Allow Pootle translations to be stored in version control
- Abstract diverse version control systems into a standard set of Pootle commands across all systems
- Ensure that we don't lose any data

- Ensure that changes made on Pootle and the filesystem can seamlessly move from one to the other

Core concepts

Stores and files Pootle contains stores of translation units. The filesystem contains files.

Tracked and untracked When a store is associated with a file, it is tracked, if it is not yet associated then it is untracked. And vice versa.

States Tracked and untracked files and stores will be in various states depending on a number of things. Have they just appeared, have they changed, have they been removed, etc.

Actions Based on the states we can determine what actions might be applicable to the stores and files.

Staging We use Pootle FS commands to stage an action. Staging is not execution of those actions but merely preparing these actions for execution.

Synchronisation This is the act of executing the staged actions.

Understanding operations

At any time we are able to query the state of Pootle FS using `fs state` command. The results of this operation will indicate if there are any actions you need to specify to resolve any conflicts or if there are untracked files.

You specify *Actions* that need to be taken to resolve conflicts or to ensure that files are tracked. This could be adding a file, removing a file or merging conflicting translations. This is the process of staging actions.

The final step is to synchronise Pootle and your filesystem. This operation takes your staged actions and executes them.

What is a filesystem

A filesystem is actually itself a Pootle FS plugin. Currently two exist:

1. **localfs** - allowing synchronization with the filesystem on which Pootle is running
2. **git** - synchronization with a Git repository

You can write a plugin for any version control system, Pootle FS will ensure that the same commands and operations are used to ensure Pootle and your filesystem stay synchronized.

How does Pootle FS relate to `update_stores`/`sync_stores`

Note: Read this if you have used previous versions of Pootle.

Previous versions of Pootle made use of two commands, `update_stores` and `sync_stores`, to allow translations to be pushed into Pootle or pulled from Pootle.

These two commands still exist but are now deprecated. We will be phasing these out in the long term to make everything use Pootle FS, but you can *replace these commands with Pootle FS equivalents*.

You can find an outline of how to use Pootle FS in the *using Pootle FS* instructions. If you have a existing project you want to migrate to Pootle FS please read the *migrating your projects to Pootle FS* instructions. You might also have an already correctly setup project that you want to *integrate directly with a version control system*.

Migrating to Pootle FS

When upgrading Pootle your projects will be automatically migrated to use the Pootle FS `localfs` backend.

Note: Before continuing *ensure all projects were properly migrated to Pootle FS* when upgrading Pootle.

While Pootle will continue to support `update_stores` and `sync_stores` these are now deprecated, so it is advisable you start to adjust your workflow to use Pootle FS.

Adjust existing automation

If you have scripts using `sync_stores` and `update_stores` then you might want to continue using those until you can migrate them to Pootle FS commands.

`sync_stores` and `update_stores` make use of Pootle FS infrastructure so they are in fact still using Pootle FS. The difference is that they mimic the monodirectional behaviour of the old commands. Pootle FS will synchronise in both directions at a unit level, while `update_stores` will only load new and changed units and `sync_stores` will only synchronise Pootle changes to disk.

The advantage of this monodirectional mode is that you can add scripts to adapt files after synchronising or before loading into Pootle. Your scripts changing files on disk will likely mess with direct Pootle FS change detection.

You may want to look at the format adaptors for future massaging or formats.

Replacing `update_stores` and `sync_stores`

`update_stores` can be replaced with the following set of Pootle FS commands:

```
(env) $ pootle fs fetch my-project
(env) $ pootle fs resolve my-project --overwrite
(env) $ pootle fs sync my-project --update=pootle
```

Note: To narrow down the execution to a particular language in a project you must append the `--fs-path` argument for each of command in the previous snippet. For example `--fs-path=my-project/fr/*` constrains to the project's French filesystem files.

`sync_stores` can be replaced with the following set of Pootle FS commands:

```
(env) $ pootle fs fetch my-project
(env) $ pootle fs resolve my-project --overwrite --pootle-wins
(env) $ pootle fs sync my-project --update=fs
```

Note: To narrow down the execution to a particular language in a project you must append the `--pootle-path` argument for each of command in the previous snippet. For example `--pootle-path=/de/my-project/*` constrains to the project's German database stores.

Integrate with version control system

Note: Pootle FS will work out of the box when synchronizing with the local file system. If this is the case you can safely skip the integration with version control.

If the translations for your project are stored in a version control system (VCS in short), then might be a good idea to directly integrate with the VCS. The following instructions work either if you project is already setup to use the `localfs` Pootle FS backend, or if you are creating and setting a new project to directly work with the VCS.

Install Pootle FS plugins for VCS

Pootle FS provides support for different VCS systems through plugins, so in order for Pootle to work with a specific VCS it is necessary to install its plugin. For example for Git:

- Install the plugin:

```
(env) $ pip install --pre --process-dependency-links Pootle[git]
```

- Add the plugin to `INSTALLED_APPS` in your custom Pootle settings:

```
INSTALLED_APPS += ['pootle_fs_git']
```

This is done once for the whole Pootle server.

Connect Pootle FS with VCS repository

The version control system also must provide access for Pootle FS to synchronize:

- Create a SSH key:

```
$ sudo -u USER-RUNNING-POOTLE ssh-keygen -b 4096
```

- Tell your upstream repository about the public key, allowing Pootle to be able to push to the repository. For example for GitHub:
 - Either use the public key as a **Deploy key** for the repository on GitHub,
 - Or (**preferred**) add the public key to a GitHub user's **SSH and GPG Keys**. In most cases you want to create a specific user in GitHub for Pootle.

Configure the project to use VCS

After installing the necessary Pootle FS plugin and connecting Pootle FS with the VCS repository, it is now necessary to alter the project configuration:

- Deactivate any existing automatic synchronization (like **cron** entries).
- Disable the project to prevent changes from translators.
- Ensure you have synchronized all the translation files to disk.
- Ensure you have committed all the translation files to your version control system.
- Set the project's **Filesystem backend** to the appropriate VCS backend.
- Set the URL to your version control repository in the project's **Path or URL**, e.g. `git@github.com:user/repo.git`.
- Synchronize as follows:

```
(env) $ pootle fs fetch $MYPROJECT
(env) $ pootle fs sync $MYPROJECT
```

- Enable the project again.
- Enable automatic synchronization again.

Your project is now ready to synchronize with the configured repository using Pootle FS. You might want to learn more about how to [use Pootle FS](#).

Using Pootle FS

The task of Pootle FS is to keep the filesystem and Pootle in sync. There are scenarios where items are not in sync and Pootle FS requires your intervention, these are the commands you will use to bring things back into sync and to resolve conflicts.

Pootle FS background

To clarify the terminology that we use in Pootle FS:

- `file` - a translation file on disk
- `store` - a translation file in the Pootle database

Files and stores are usually associated and thus we are able to keep them synchronised. But there might be files with no store (the store for a new file has not yet been created in the Pootle database), and stores with no file (the file has been removed from the filesystem).

Pootle FS works in these stages:

1. Actions are staged. An action is chosen to resolve each issue.
2. The system is synchronized. The staged action are actually performed.

Files that have never been synced are untracked, need to be explicitly staged. Files previously synced are tracked, will be automatically staged if there are any changes. In the case of conflicts (changes both on disk and in Pootle) it is also necessary to manually stage these to resolve which version should prevail.

When staging it is possible to specify specific stores or files, or groups of them using the `-P` and `-p` options. It is also possible to limit which staged actions are executed by using these same options on the `sync` command.

Syncing tracked stores or files

When a store and its corresponding file are tracked and previously synced, then they are automatically staged for syncing if either changes.

If both have changed then we will need to specify how to [resolve the conflict](#).

To re-sync stores and files run:

```
(env) $ pootle fs sync MYPROJECT
```

Adding new files and stores

When new files appear on the filesystem that we want to bring into Pootle we use `add`. And when new stores have appeared on Pootle that we want to push to the filesystem we also use `add`:

```
(env) $ pootle fs add MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

Where `add` will stage the previously untracked files or stores. While `sync` will synchronize, pulling the translations in the file into the Pootle database or pushing translations from the stores in the database to files on the filesystem.

Following this the file and store are now tracked.

Removing files or stores

A store or file can be missing from Pootle or the filesystem because it has been removed, we use `rm` to remove such files and stores:

```
(env) $ pootle fs rm MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

This will remove the store or file, depending on whether it is the file or store that remains.

Following this there is no such file or store on the filesystem or on Pootle.

Resolving conflicts

Conflicts can occur if a tracked Pootle store and its corresponding file have both changed. They can also arise if a new Pootle store is added and a matching file has been added in the filesystem simultaneously.

Using the `resolve` command we have four possible ways to resolve such conflicts:

1. Keep the filesystem version and discard all Pootle translations
2. Keep the Pootle version and discard all filesystem translations
3. Merge translations and for unit conflicts choose Pootle's version and turn the filesystem version into a suggestion
4. Merge translations and for unit conflicts choose the filesystem version and turn the Pootle translation into a suggestion

The merge options are most useful where you need translators to resolve the conflict.

The default options for `resolve` ensure that the filesystem always wins and that translators will be given an opportunity to resolve the conflict. This ensures that changes on version control are authoritative, a fix can land there without anyone knowing Pootle is involved. It also means that we have no data loss in that any conflicts will be presented to translators as suggestions, allowing them to resolve the conflicts by reviewing the suggestions.

Overwrite Pootle with filesystem version

You want to keep the version that is currently on the filesystem, discarding all changes in Pootle:

```
(env) $ pootle fs resolve --overwrite MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

Overwrite filesystem with Pootle version

You wish to keep the version that is currently in Pootle, discarding all changes in the filesystem:

```
(env) $ pootle fs resolve --overwrite --pootle-wins MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

Use filesystem version and convert Pootle version into a suggestion

You want to retain all translations and allow translators to resolve conflicts. This will merge any non-conflicting units. For conflicting units, keep the filesystem translation and convert the Pootle translation into a suggestion:

```
(env) $ pootle fs resolve MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

The result is that all non-conflicting units have been synchronised. For any unit where both the store unit and file unit changed the translation is set to the file unit translation with the store unit translation converted into a suggestion. You can now review these suggestions to resolve the conflicts.

Use Pootle version and convert filesystem version into a suggestion

You want to retain all translations and allow translators to resolve conflicts. This will merge any non-conflicting units. For conflicting units, keep the Pootle translation (due to the `--pootle-wins` option) and convert the filesystem translations into a suggestion:

```
(env) $ pootle fs resolve --pootle-wins MYPROJECT
(env) $ pootle fs sync MYPROJECT
```

Pootle FS statuses

Pootle FS uses a set of statuses for the files or stores it manages, using these it is able to determine what to do to resolve the scenario.

There are two groups of statuses:

- Unstaged - these need to be resolved by the user, once resolved they become staged. To resolve the user will use the `resolve`, `rm` and `add` commands.
- Staged - staged and ready to be `sync`'ed.

Unstaged statuses

Statuses that need an action to be specified in order for Pootle FS to be able to resolve them:

conflict Both the Pootle store and the filesystem file have changed. To resolve this conflict use `resolve` to merge the two and manage conflict resolution. Or if you wish to discard one or the other use `resolve --overwrite`.

conflict_untracked Conflict can also arise if both the store and file are untracked. In this case you can use `resolve` to combine the translation and manage conflict resolution for each unit. Or to force taking the whole file or store use either `resolve --overwrite` and optionally `--pootle-wins` depending on whether you want to keep the filesystem file or the Pootle store.

pootle_untracked A new store has been added in Pootle but does not have any matching file on the filesystem. You can use either `add` to create the file on the filesystem and push the translations on the store to it, or alternatively use `rm --force` to stage the store for removal.

fs_untracked A new file has been added in the filesystem but does not have any matching store in Pootle. You can use either `add` to pull the file into Pootle or alternatively use `rm --force` to stage the file for removal.

pootle_removed A tracked store has been removed. Either use `add --force` to restore the filesystem version, or use `rm` to stage for removal from filesystem.

fs_removed A tracked file has been removed from the filesystem. Either use `add --force` to restore the Pootle version, or use `rm` to stage for removal from Pootle.

Staged statuses

These statuses reflect changes that can be either unstaged by using `unstage` or executed with `sync`:

pootle_ahead A Pootle store has changed since the last synchronization. Running `sync` will push the changes to the filesystem.

fs_ahead A file has changed in the filesystem since the last synchronization. Running `sync` will pull the changes to Pootle.

pootle_staged A new store (with no associated file on the filesystem) has been created in Pootle and has been staged to be added to the filesystem. Running `sync` will create the file on the filesystem.

fs_staged A new file (with no associated store on Pootle) has been created in the filesystem and has been staged to be added to Pootle. Running `sync` will create the store on Pootle.

merge_pootle_wins Merge stores or files that have both been updated. If there are conflicts use the translation from Pootle and turn the translation from the file into a suggestion.

merge_fs_wins Merge stores or files that have both been updated. If there are conflicts use the translation from the filesystem and convert the translation from Pootle into a suggestion.

remove A file or store, whose corresponding store or file is missing, has been staged for removal. Running `sync` will remove the file or store.

both_removed A previously tracked file has been staged to be removed from both the filesystem and Pootle. Running `sync` will remove both the file and store.

1.2.2 Backends and storage

File formats

Pootle supports many file formats through the powerful *Translate Toolkit* API. The Toolkit also provides several format converters for other formats, this will allow you to host a lot of translatable content on Pootle.

All these formats can be downloaded for offline use/or translation (for example in *Virtaal*). We recommend *Virtaal* for offline translation. They can also be downloaded in XLIFF format.

Bilingual formats

These formats are translation files that include the source and target language in one file.

- Gettext PO
- XLIFF
- Qt TS
- TBX

- [TMX](#)

Translation statistics

Pootle gives translators and project developers an easy way to see progress on the translation work. Progress is indicated by a coloured graph to easily see how much work is complete, and how much work remains. Pootle can also give detailed statistics about the progress in translation work.

Statistics report on both the progress in the number of messages, and in the number of words. The number of words gives a much better impression of how much work is involved, and allows for more accurate time estimation.

Pootle also assists in translation quality assurance, by performing several [Quality checks](#) on the translations which can help in review. These quality checks correspond to the [quality checks](#) performed by [pofilter](#) from the Translate Toolkit.

Translation templates

Translation templates are translation files that contain only the source text (original text). These files are used as a template to create target files for each language.

Users familiar with Gettext know translation templates as POT files. For other bilingual formats (like XLIFF) untranslated files with the same extension are used as templates.

The “*Templates*” language

Pootle can manage a special language called *Templates*. This is not strictly speaking a language but rather a place to store translation templates for a project.

If the *Templates* language is present then Pootle will initialise brand new languages from the *Templates* files present in Pootle.

If the *Templates* language is absent from a project, Pootle will assume all initialisation of files for new languages happens outside of Pootle.

Starting a new translation

It is helpful to understand in more detail how a new language is created or added to Pootle.

When adding a new language to a project from the Pootle interface, and the *Templates* language exists for the project in Pootle then a fresh copy will be generated based on those template files.

If there is no *Templates* language it is necessary to manage all initialisation of languages from the Pootle command line. When using [update_stores](#) new languages will be initialised if they are present on the filesystem. You are responsible for initialisation of these new languages from template files as required.

Updating existing translations

Pootle will not update existing translations if new template files are added to Pootle. Updating of translations is managed outside of Pootle. You can update your translations as follows:

1. Use [sync_stores](#) to sync all translations to the filesystem. These files will now contain the latest translations from Pootle users.
2. Use [pot2po](#) or similar to update the translations.
3. Use [update_stores](#) to push the updated translations to Pootle.

A detailed example can be found in *Updating strings for existing project*.

1.2.3 Online translation editor

Alternative source language

Pootle has the ability to display alternative source languages while translating. Thus, translators who know another language better than English can take part in the translation project. Also, it provides a way to disambiguate terminology by seeing how other languages have translated the same string.

The screenshot shows the Pootle online translation editor interface. At the top, the browser address bar displays the URL: `Basque → Firefox → firefox/browser/profile/bookmarks.inc.po → Unit #8707666`. The main interface has a light blue background. On the left, it shows 'Priority: 6.0' and 'Locations: bookmarks_heading'. The central area displays three alternative source languages: 'French' with the string 'Marque-pages', 'Spanish' with 'Marcadores', and 'English' with 'Bookmarks'. Each language has a small icon to its right. Below the 'English' section, there is a text input field containing 'Laster-markak'. To the right of the input field is a green 'Submit' button and a blue 'Suggest' button. Below the input field, there is a checkbox labeled 'Needs work'. At the bottom left, there are three small icons: a speech bubble, a magnifying glass, and a document.

Setup

Users who want to use the functionality need to specify the desired alternative source languages in their account configuration. Alternatively, Pootle will try to guess the user's alternative source language by looking at the browser's `Accept-Lang` header.

Note: If the selected project doesn't have translations in the alternative source language then no alternative will be displayed.

This feature is enabled by default.

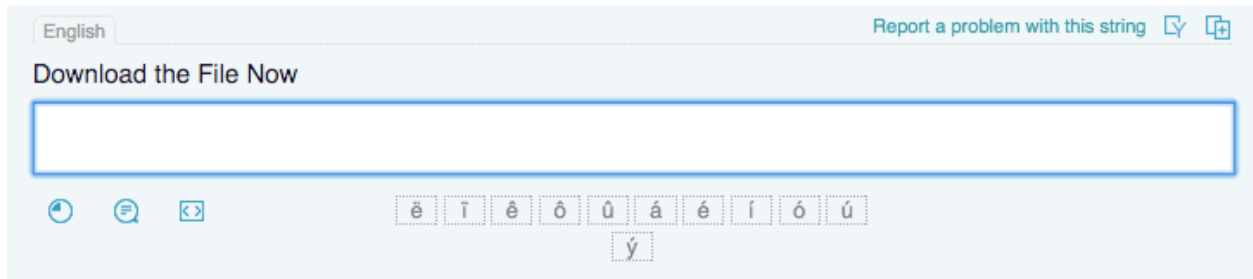
Matching criteria

In order to show suggestions from another language, the following is needed:

- The alternative languages must be visible in Pootle and added to the same project.
- The string must be translated in the alternative language (not incomplete or untranslated).
- The file names need to be identical (identical strings from different files are not matched).
- The source text for both translations need to be identical.

Special characters

Pootle can display clickable characters which might be difficult to type or unavailable to the translator on their keyboard. These appear below the translation widget as we see below for Afrikaans.



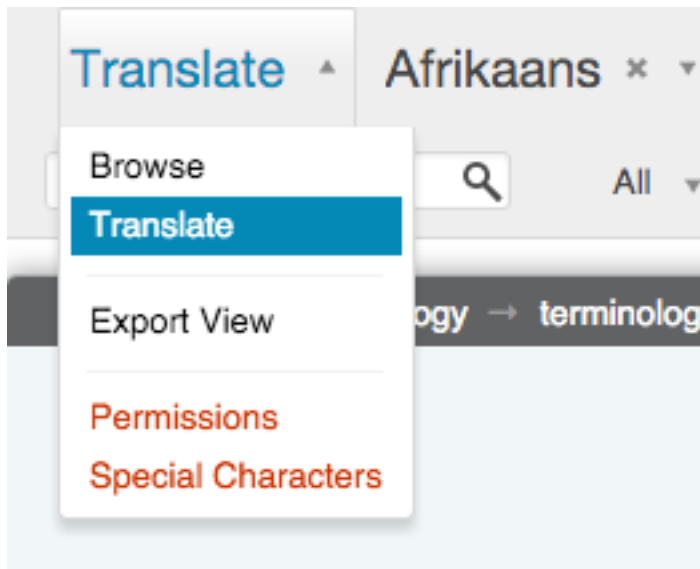
Clicking on any of the ãĩêôûáéíóúý characters will insert that character into the translation.

Many languages, e.g. those using the Latin script with diacritics, will find this very helpful, especially where keyboard layouts are not readily available.

Adding or altering special characters

Anyone with admin rights for Pootle or for a specific language can adjust the special characters.

To adjust the characters open the Special characters page accessed via the admin dropdown in the navigation bar.



Adjust the needed characters by adding and deleting characters.

Special Characters ▾ Afrikaans × ▾ All Projects ▾

Special Characters

Enter any special characters that users might find difficult to type

Save

When to use special characters

Special characters do not solve the input needs for all languages, but has been a very useful help for many languages, especially in *translate@thons*.

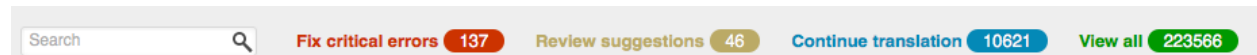
For people using non-Latin scripts, consider if it will be useful to perhaps include things that can't be easily typed by translators in your language. You will probably need to limit the number of characters, but hopefully you can find a reasonable compromise that will help many people.


Quality checks

Pootle provides a powerful way of reviewing translations for quality. It exposes most of the [pofilter checks](#) that can check for several issues that can affect the quality of your translations.

If Pootle indicates a possible problem with a translation, it doesn't mean that the translation is necessarily wrong, just that you might want to review it. Pootle administrators should indicate the correct project type (GNOME, KDE, Mozilla, etc.) in the administration pages. This will improve the accuracy of the quality checks.

Critical checks are prominently displayed through the browsing UI.

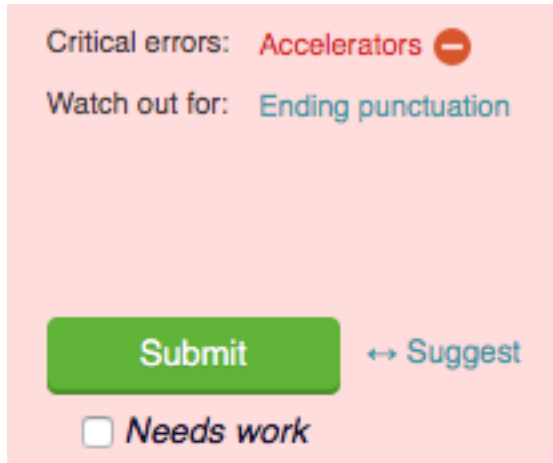


Any extra failing checks can be accessed by clicking the  button located below the navigation breadcrumbs. Clicking on the name of a check will step you through the translations that fail that check.

While in the translation editor, submissions resulting in critical failing checks will be immediately reported, preventing you from automatically continuing until the issues have been resolved or muted as false positives by using the mute





button. Non-critical checks flag potential problems but can be ignored or muted as needed.



To understand the meaning of each check, Pootle displays the failing checks right on top of the submission button, with a link to the online documentation. You can also read the detailed descriptions of the [pofilter checks](#).

Muting Quality Checks

It is possible to mute the quality check if the translation is correct. Reviewers are able to remove the check for a certain string, by clicking on the  button, to indicate that the string is correctly translated. This avoids having to review the same check multiple times. You can unmute any muted check using the  button.

If the source text of the translation is changed then the mute is discarded to ensure that the new translation is checked again for any possible issues.

Translation Memory

Pootle provides matching translations to the current string. Translator can use these matches as their translation or to aid their translation.

Matches are based on previous translations of similar strings. These Translation Memory (TM) matches mean that you can speed up your translation and ensure consistency across your work.

Using Translation Memory

Translation Memory matches are automatically retrieved when you enter a new translation unit. These are displayed below the editing widget. You can insert a TM match by clicking on a match row.

The differences between the current string and the match are highlighted, this allows you to see how the two differ and helps you make changes to the match to adapt it for use with the current string.

Configuring Translation Memory

Translation Memory will work out of the box with a default Pootle installation. There are three methods of getting Translation Memory.

1. Amagama - for remote Translation Memory
2. Elasticsearch - for local Translation Memory

3. Elasticsearch - for external Translation Memory

Amagama based remote TM

By default Pootle will query Translate's [Amagama](#) Translation Memory server, which hosts translations of an extensive collection of Opensource software.

If you want to setup and connect to your own TM server then the [AMAGAMA_URL](#) will allow you to point to a private TM server.

To disable Amagama set [AMAGAMA_URL](#) to ' '.

Elasticsearch-based TMs

New in version 2.7.

Pootle can also retrieve TM matches stored on Elasticsearch-based TM servers. These TM servers require [Elasticsearch](#) to be installed and running.

Note: Elasticsearch depends on Java. Note that some systems may ship with OpenJDK, however [elasticsearch recommends using Oracle JDK](#).

Install the required Python libraries:

```
(env) $ pip install --pre --process-dependency-links Pootle[es5]
```

Note: Elasticsearch TM should work with any version of Elasticsearch, our tests run against Elasticsearch 5.x. For support for Elasticsearch 1.x and Elasticsearch 2.x, simply replace es5 with es1 and es2 respectively in the above command.

Pootle supports two types of Elasticsearch-based TMs:

- **Local TM:** (just one, named `local`) is populated using translations stored in Pootle database and every new translation gets automatically imported to it.
- **External TMs:** (several) are populated from translation files specifically provided by the server admins, and are not automatically updated.

Both local and external TM settings can be adjusted in [POOTLE_TM_SERVER](#). A configuration example for local and external TM can be found in the default `~/.pootle/pootle.conf`, and can be enabled by uncommenting the example.

Please see the [POOTLE_TM_SERVER-WEIGHT](#) for a full example of the configuration necessary to set up local/external TM.

Both Amagama and Elasticsearch based TMs can operate together. Though you may want to disable Amagama.

Elasticsearch-based local TM

New in version 2.7.

To use it, the `local` TM must be enabled in [POOTLE_TM_SERVER](#) and will need to be populated using the [update_tmserver](#) command:

```
(env) $ pootle update_tmserver
```

Once populated Pootle will keep Local TM up-to-date.

Elasticsearch-based external TMs

New in version 2.7.3.

In order to use them they must be enabled in `POOTLE_TM_SERVER` and you will need to populate them using the `update_tmserver` command specifying the TM to use with `--tm` and the display name with `--display-name`:

```
(env) $ pootle update_tmserver --tm=external --display-name=Pidgin af.po gl.tmx
```

A display name is a label used to group translations within a TM. A given TM can host translations for several labels. Just specify them with `--display-name`:

```
(env) $ pootle update_tmserver --tm=external --display-name=GNOME pt.tmx eu.po xh.po
```

It is possible to have several Elasticsearch-based external TM servers working at once, along with the Elasticsearch-based local TM server. In order to do so just add new entries to `POOTLE_TM_SERVER`:

```
POOTLE_TM_SERVER = {
    ...

    'libreoffice': {
        'ENGINE': 'pootle.core.search.backends.ElasticSearchBackend',
        'HOST': 'localhost',
        'PORT': 9200,
        'INDEX_NAME': 'whatever',
        'WEIGHT': 0.9,
        'MIN_SCORE': 'AUTO',
    },
}
```

Make sure `INDEX_NAME` is unique. You might also want to tweak `WEIGHT` to change the score of the TM results in relation to other TM servers (valid values are between 0.0 and 1.0).

To use these additional external TMs you will need to populate them using the `update_tmserver` command specifying the TM server with `--tm`:

```
(env) $ pootle update_tmserver --tm=libreoffice --display-name=LibreOffice af.po gl.
↪tmx
```

Check `update_tmserver` for more options.


Note that Pootle will not push new translations to these TM servers unless you explicitly use the `update_tmserver` command, giving you full control of which translations make into them.

Machine Translation

Pootle has the ability to use online Machine Translation (MT) Services to give suggestions to translators. This feature has to be enabled by the server administrators.

Using Machine Translation

Note: Machine Translations are not meant to replace human translations but to give a general idea or understanding of the source text. It can be used as suggestion of a translation, but don't forget to review the suggestion given.

If the server administrator has enabled machine translation then an icon  will be displayed for each source text (English or alternative source language) next to the Copy button. Clicking the relevant buttons will retrieve translation suggestions from the online services and will mark the current string as fuzzy to indicate that review is required.

Enabling Machine Translations


To enable a certain Machine Translation Service, edit *your configuration file* and add the desired service within the `POOTLE_MT_BACKENDS` setting.

Each line is a tuple which has the name of the service and an optional API key. Some services may not require API keys but others do, so please take care of getting an API key when necessary.

Available Machine Translation Services

Supported Services:

 Google Translate

 Yandex.Translate

New in version 2.7: Yandex.Translate

Google Translate is widely used and supports a number of [languages](#). It is a [paid service](#) requiring an account and API key.

[Yandex.Translate](#) is the free alternative to Google.

Searching in Pootle

Pootle provides search functionality that allows translators and reviewers to search through translations for some text. The search box is shown close to the top of the page. Searching can be used to find specific things you want to work on, see how issues were solved before, or to verify consistency in your translations.

Search results are up to date, and will reflect the current translations in Pootle.

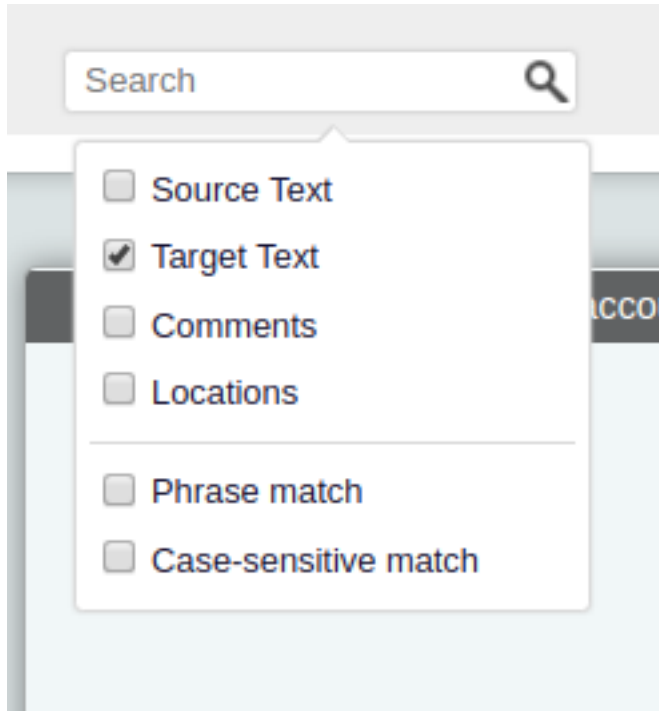
Search domain

It is important to realize that when a new search term is entered, **searching will take place inside the currently viewed domain**. If you are currently at the top level of your project, the whole project will be searched. If you are viewing a directory, only files under that directory will be searched. If you are already viewing/translating a file, only that file will be searched.

The first result will be shown in context in the file where it is found. When you click “Skip”, “Suggest” or “Translate” it will provide the next match to the search (in the original domain) until all matches were presented. Remember that if you edit the search query while viewing search results in a specific file, your new query will only search in that specific file.

Advanced search

When you enter a search box a dropdown will open allowing you to limit or expand your **search to specific fields**. Any combination of these fields and options is accepted.



Fields that you can search in include:

- Source Text – the original reference text.
- Target Text (**default**) – the translations.
- Comments – any comments with the translation.
- Location – any location, key or ID value.

Options:

- Phrase Match (**default: off**) – search matching specified phrase. With this option on searching for “file table” will not find “table file”.
- Case-sensitive Match (**default: off**) – search in a case sensitive manner. With this option on searching for “File” will not find “file”.

Keyboard shortcuts

Editing

Action	Current shortcut	Proposed shortcut
Submit and move to next translation	Ctrl+Enter	
Toggle the ‘Needs work’ flag	Ctrl+Space	
Toggle the suggest/submit mode	Ctrl+Shift+Space	
Copy the contents from the original language		Alt+Down
Focus on comments field		Ctrl+Shift+C

Navigation

Action	Shortcut	Alternative Shortcut
Move to previous string	Ctrl+Up	Ctrl+,
Move to next string	Ctrl+Down	Ctrl+.
Move to the first string	Ctrl+Shift+Home	
Move to the last string	Ctrl+Shift+End	
Move up 10 strings	Ctrl+Shift+Page Up	Ctrl+Shift+,
Move down 10 strings	Ctrl+Shift+Page Down	Ctrl+Shift+.
Select search box	Ctrl+Shift+S	
Select page number	Ctrl+Shift+N	

Translation suggestions

Pootle has the ability to optionally allow users to provide suggestions that need to be reviewed before they are accepted into the real translation files. Who is allowed to do what, is determined by the configuration of *User permissions* for the project or the server.

This allows for a team to form with different roles for different team members, and makes it possible to have a more explicit review step that requires suggestions to be checked before they become the real translations. This also allows the collection of different ideas for translating a single string.

Viewing and making suggestions

When translating, suggestions are shown inline so they're always visible. If a user wants to view all the suggestions within a project scope, it just needs to go to the “Review” tab and click on the “View Suggestions” link. Users with rights to translate will be shown a “Review Suggestions” link and will be able to accept and reject suggestions.

Users with rights for making suggestions will see a “Suggest” button next to “Submit”. Making a suggestions is as easy as clicking the button – hey, did you expect more steps involved?

Reviewing suggestions

In order to review suggestions, users must have privileges to translate. There are two ways for reviewing suggestions: going through all of them, or reviewing while translating.

To go through all of them, the reviewer must click on “Review Suggestions” within the “Review” tab of the project. This would guide her/him through all the suggestions available for the current view.

The second method is straightforward, since suggestions are shown throughout the translation process. Additionally, buttons for accepting and rejecting the suggestions are displayed.

While reviewing a suggestion, a coloured difference between the current translation and the suggestion is displayed. If available, the username is provided of the user that gave the suggestion.

A click on the green tick icon approves the selected suggestion while the red cross rejects the selected suggestion. A suggestion approval doesn't imply the rejection of the remaining suggestions.

Terminology

Pootle can help translators with terminology. Terminology can be specified to be global per language, and can be overridden per project for each language. A project called *terminology* (with any full name) can contain any files that

will be used for terminology matching. Alternatively a file with the name *pootle-terminology.po* (in a PO project) can be put in the directory of the project, in which case the global one (in the terminology project) will not be used. Matching is done in real time.

Ideally, the source term should be the shortest, simplest form of a word. Therefore *cat*, *dog*, *house* are good, but *cats*, *dogged* and *housing* are bad.

Context indicators are allowed in the source text, in brackets after the term, but keep them short, eg *file (noun)*, *view (verb)*, etc.

The ideal is therefore that the target term be something that you'd like the translator to be able to insert... but strictly speaking the target text can be anything, including a definition.

If the terminology PO file has translator comments, they will be displayed as a tooltip in Pootle.

What does it do?

If our glossary has an entry: *file->lêr*, and we translate a sentence like *The file was not found*, we can suggest the glossary entry *file->lêr* as relevant to the translation, even if we don't have any TM entry that is related to the complete sentence that is available for translation.

Say our glossary has an entry *category->kategorie* and we translate a sentence like *Please enter the categories for this photo*, we can suggest the glossary entry *category->kategorie*, even though the letters *category* doesn't occur anywhere in the original string.

Limits

Currently a single term entry can be up to 30 characters long (including context information), and the first 500 characters of each translation are scanned. Terms can consist of many words, but consider making them as general or simple as possible for maximum impact.

If these limits prove too restrictive, feel free to point out use cases where this is not sufficient.

Since the terminology matching is performed in real-time, you might want to keep an eye on the size of your terminology project to ensure that performance is not affected too much by having too many terms. This is highly dependent on your server abilities and the nature of what you are translating.

Virtual Folders

New in version 2.7.

Virtual folders provide a way to group translations based on any criteria, including a file across all the languages in a project, or files on specific locations. Virtual folders have priority, so they can be used to allow translators to focus on the most important work.

Virtual folders' attributes

Virtual folders have several attributes:

- A mandatory lowercase name,
- A mandatory location,
- An optional priority,
- An optional publicness flag,

- An optional description,
- A field accepting several optional filtering rules.

The location indicates the root place where the virtual folder applies. It can use placeholders for language (`{LANG}`) and project (`{PROJ}`).

Note: The `/` location is not valid and must be replaced by `/ {LANG} / {PROJ} /`. The locations starting with `/ projects /` are also not valid and must be changed so they instead start with `/ {LANG} /`.

Each virtual folder must have a unique combination of name and location. This means that there can exist two different virtual folders with the same name if they have different locations.

The priority defaults to 1 and accepts any value greater than 0, including numbers with decimals, like 0.75. Higher numbers means higher priority.

By default virtual folders are public. If they are not public then they won't be displayed, but they are still used for sorting.

Also the virtual folders can have a description which might be useful to explain the contents of the folder or provide additional instructions. This might be handy when using the virtual folders as goals.

The filtering rules specify which translation units are included within a virtual folder. Currently the only supported filtering rule consists of a list of file or directory paths relative to the virtual folder location. Note that it is required to set some filtering rule.

Adding and updating virtual folders

To add or modify the properties of virtual folders use the `add_vfolders` management command.

This command imports a JSON file holding a list of virtual folders, and the files included on each virtual folder along with all their attributes. Check the specs for the *JSON format* in order to know how to craft a JSON file that fits your needs.

Virtual folders stats

Translation stats of virtual folders are automatically calculated.

Translating virtual folders

If a virtual folder applies in the current location, then clicking on the links on the overview page will provide the units in priority order when translating in the editor. The priority sorting on the translation editor is calculated taking into account all the applicable virtual folders in the current location, including the not public ones.

Format for the JSON file

The JSON file used to import virtual folders consists of a list of virtual folder definitions with the *same fields* as the virtual folders, except for two differences:

- If the **description** includes newlines those must be escaped.

The following example depicts a basic JSON file:

```
[
  {
    "name": "user1",
    "location": "{LANG}/firefox/browser/",
    "priority": 999.99,
    "is_public": true,
    "description": "Most visible strings for the user.",
    "filters": {
      "files": [
        "branding/official/brand.dtd.po",
        "chrome/browser/aboutDialog.dtd.po"
      ]
    }
  },
  {
    "name": "user2",
    "location": "gl/firefox/",
    "priority": 7.5,
    "is_public": false,
    "filters": {
      "files": [
        "browser/chrome/browser/aboutSessionRestore.dtd.po",
        "browser/chrome/browser/downloads/downloads.dtd.po"
      ]
    }
  },
  {
    "name": "user3",
    "location": "ru/{PROJ}/",
    "priority": 0.3,
    "filters": {
      "files": [
        "browser/chrome/browser/engineManager.dtd.po"
      ]
    }
  },
  {
    "name": "directories-for-lang",
    "location": "{LANG}/",
    "filters": {
      "files": [
        "firefox/browser/profile/",
        "firefox/browser/chrome/browser/"
      ]
    }
  },
  {
    "name": "directories-and-files-for-tp",
    "location": "{LANG}/firefox/",
    "filters": {
      "files": [
        "browser/updater/",
        "browser/chrome/browser/devtools/appcacheutils.properties.po",
        "browser/chrome/browser/migration/"
      ]
    }
  }
],
```

(continues on next page)

(continued from previous page)

```
{
  "name": "default",
  "location": "{LANG}/{PROJ}",
  "description": "All files in all projects for all languages.",
  "filters": {
    "files": [
      "/"
    ]
  }
},
{
  "name": "other",
  "location": "/af/firefox/",
  "is_public": true,
  "filters": {
    "files": [
      "browser/chrome/browser/aboutCertError.dtd.po"
    ]
  }
},
{
  "name": "developer",
  "location": "/af/firefox/",
  "priority": 0.9,
  "description": "As you can see this\\n description spans\\n several lines.",
  "filters": {
    "files": [
      "browser/chrome/browser/devtools/appcacheutils.properties.po",
      "browser/chrome/browser/devtools/debugger.dtd.po"
    ]
  }
},
{
  "name": "install",
  "location": "/ru/{PROJ}/",
  "priority": 5,
  "is_public": true,
  "description": "Installation related strings.",
  "filters": {
    "files": [
      "browser/chrome/browser/migration/migration.dtd.po",
      "browser/chrome/browser/migration/migration.properties.po"
    ]
  }
}
]
```

Offline Translation

You can export files for offline translation. Once translated you can import them again and Pootle will manage updating the translation in Pootle based on your changes.

This feature is ideal for teams who have poor connectivity or if you prefer to use an offline translation tool.

To export, simply click on the “Download for offline translation” link on the sidebar in Pootle’s overview page. To import simply click the “Upload translations” link and select the file you wish to upload.

Changed in version 2.7.1.

If a string has been translated on Pootle and changed in your uploaded file then your change will still be uploaded but it will be converted into a suggestion which you can resolve in Pootle.

Note: If there are any errors in the upload then Pootle will warn you and the file will be rejected.

1.2.4 Administrative features

Static Pages

Pootle makes it easy to setup additional custom content without too much effort.

There are three types of static pages:

1. Regular – these work like normal web pages and are able to present additional content such as a “Getting Started” page. You will want to add these into other *UI customisation*.
2. Legal – in addition to presenting content like a Regular page, Legal pages require that users agree to the content otherwise they are logged off the system. Use these pages for presenting terms of service or changes in licensing terms that user must accept before they can use or continue to use Pootle.
3. Announcements – these appear in the sidebar and can present special instructions about projects. If you change these pages then they will be presented to users on their next visit. Announcements can also make use of link rewriting to allow URLs to vary based on the language being browsed.

Use **Admin – Static Pages** to create and manage static pages.

The static pages are by default formatted using HTML. But you can use Markdown or RestructuredText by setting `POOTLE_MARKUP_FILTER` correctly.

Links in static pages

When linking to a static page externally or in any customisations, your links would be pointing to `/pages/$slug`, such as `/pages/getting-started`.

For linking to another static page from within a static page use the `#$slug` syntax. Thus, if you created a *Getting Started* page as a static page which pointed to your *Licence Statement* legal page we’d use this `#/licence_statement` in the URL.

Special features of announcement pages

Naming your slug

When creating an announcement page use a slug `projects/$project` so that the page will be used on the `$project` project.

Other slug names may be used:

- `$lang` - for an announcement page that will appear on every single project enabled for the `$lang` languages.
- `$lang/$project` - for an announcement page that is specific to the `$project` project in the `$lang` language.

The preferred model though is to use the `projects/$project` convention for a single easy-to-maintain page.

Language link rewriting in Announcement pages

In many cases you have URLs in announcement pages that would be identical except for variations in the language code. Examples would include links to team wiki pages, signoff pages, progress dashboards, live test versions, etc.

Any link within your announcement page that uses a fake language code of `/xx/` will be rewritten with the language code for this translation. Thus if you insert a link such as `http://example.com/signoff/xx/` then that will be rewritten to `http://example.com/signoff/af/` for a user viewing this announcement page for the Afrikaans language translation.

Captcha Support

With Pootle's flexible [permissions](#) several ways of interacting with your translation community are possible. If you have a very open Pootle server, you might want to ensure that spammers don't abuse it by enabling [captchas](#).

Configuration

If you have no need for captchas, e.g. at a translation sprint, you might want to remove captcha support. To disable it, set `POOTLE_CAPTCHA_ENABLED` in your configuration file to `False`. Restart your server for the setting to take effect.

Customization

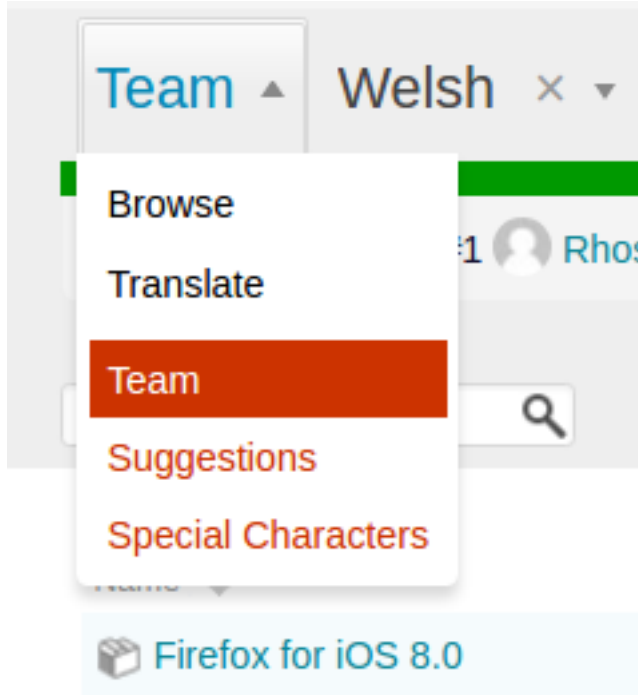
The captchas can be customized. Look at the captcha template and code:

- `pootle/templates/captcha.html` and
- `pootle/middleware/captcha.py`

and make the changes you need.

Teams

Pootle's team page is designed to assist in managing teams. These includes team members and their roles within the team, and other team related aspects and features. The team page also provides overall statistics and access to the bulk handling for pending suggestions.



Language administrators will find the team interface on the first dropdown in their language's browse page.

The team page is divided in two parts, on the left the management of the team members and on the right a list of the projects enabled for the language, and the overall translation statistics with links to the editor and the bulk suggestions manager.

Managing team members

The UI for team members management is divided in two parts:

1. Top left: a form to add new members to the team,
2. Bottom left: the team's members listed within their assigned roles.

Roles

There are four different roles, which provide incremental permissions within the team:

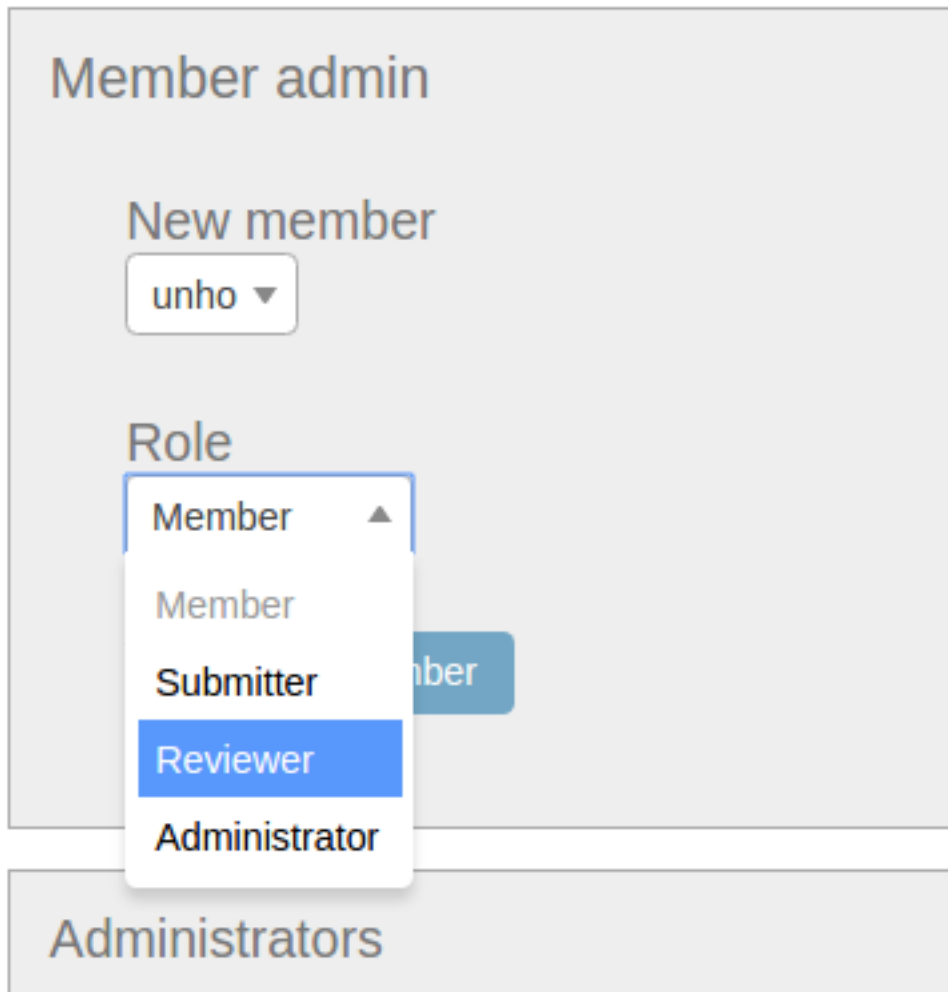
Member May submit suggestions.

Submitter Can translate and make suggestions.

Reviewer In addition to translate and submit suggestions, can also review suggestions.

Administrator Can administer the team, adding team members and adjusting roles. May edit the announcement of the team. Plus all rights of the Reviewer.

Adding a member



The screenshot shows a web interface titled "Member admin". Below the title is a section labeled "New member" containing a dropdown menu with the text "unho" and a downward arrow. Below this is a section labeled "Role" with a dropdown menu that is open, showing a list of roles: "Member" (with an upward arrow), "Member", "Submitter", "Reviewer" (highlighted in blue), and "Administrator". To the right of the "Role" dropdown is a blue button with the text "Add team member". Below the "New member" section is a section labeled "Administrators".

Add new members using the form at the top left. Simply, select the user, choose a role and click **Add team member** button.

Removing a member

A screenshot of a web interface titled "Administrators" in a light blue header. Below the header, there is a list of three users: "admin 1", "admin 2", and "admin007". Each user name is preceded by a small square checkbox. The checkbox for "admin 2" is checked, while the others are unchecked. At the bottom of the list, there is a blue button with the text "Remove selected" in white.

To remove members check the box next to their username and then click on the **Remove selected** button.

Changing a member's role

To change a member's role first remove the member from the team, then add the member back with the desired role.

User permissions

There are several rights which can be assigned to users or to a group of users, such as to all logged in users. The default site-wide permissions are configured by the server administrator. These are the permissions that will be used in each project unless other permissions are configured.

Permissions precedence

Permissions can be customized server-wide, per-language, per-project or language/project combination (translation project).

Permissions apply recursively, so server-wide permissions will apply to all languages and projects unless there is a more specific permission. Language permission applies to all translation projects under that language, etc.

Special users

Pootle has two special users, *nobody* and *default*, which are used to assign permissions to more than one user at once. The user *nobody* represents any non-logged in user, and *default* represents any logged in user.

If a user has permissions assigned to her user account they override any default permissions even those applied to more specific objects (i.e. a user who has specific rights on a language will override default rights on translation projects).

Server administrators can be specified in the users page of the admin section. Server administrators have full rights on all languages and projects and override all permissions.

Available permissions

Access Permissions

Access rights can be set server-wide or for projects. Bear in mind that when limiting access to projects the permissions affect to *all the languages* available in the project.

view Gives access to a project.

hide Forbids access to a project.

Action Permissions

Permissions restricting actions can be set server-wide, per language, or language-project combination:

suggest The right to *suggest* a translation for a specific string, also implies the right to upload file using suggest only method.

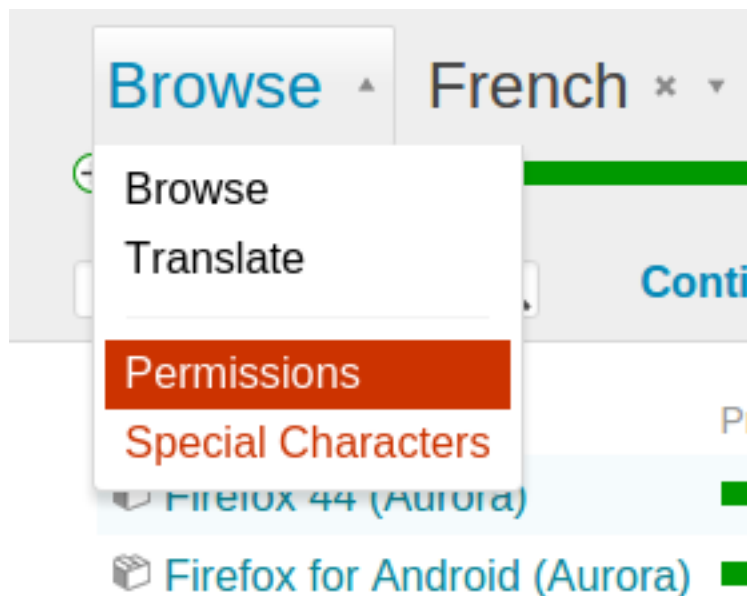
review The right to review the suggested translations and accept or reject them, as well as the right to reject false positive quality checks

translate The right to supply a translation for a specific string or to replace the existing one. This implies the right to upload files using the merge method.

administrate The right to administrate the project or language including administer permissions and delegating rights to users (this is not the same as the site administrator)

Permissions interface

Users with administrative rights for projects, languages or translation projects can access the permissions interface by selecting “Permissions” in the navigation dropdown on the project, language or translation project browsing pages.



Pootle administrators will find the default permissions interface on the administration page, at the “Permissions” tab, where they can set server-wide permissions.

The current rights are listed as they are assigned. The user “nobody” refers to any user that is not logged in (an anonymous, unidentified user). The user “default” refers to the rights that all logged in users will have by default, unless other specific rights were assigned to them. The rest of the users are users of the Pootle server for which non-default rights were assigned.

Changing permissions

In the list of permissions, you can simply select which rights must be assigned to each user by picking new permissions or unassigning them. Changes will be updated when you submit the form.

Permissions ▾
French × ▾
All Projects ▾

Below you can add, edit, and delete the permissions for this language.

The user “nobody” refers to any user that is not logged in (an anonymous, unidentified user). The user “default” refers to the rights that all logged in users will have by default, unless other specific rights were assigned to them.

Username ↕	Add Permissions	<input type="checkbox"/> Delete:
random	<div> × Can administrate a translation project × Can review suggestions × Can make a suggestion for a translation × Can submit a translation </div>	<input type="checkbox"/>
----- ▾	Select one or more permissions	
Username	Add Permissions	Delete

Save Changes

Adding a user

To set permissions for a specific user, select the user in the dropdown list and set the specific rights for that user. This is only necessary if the user does not yet have their own set of rights defined.

Removing a user

To reset some user’s rights to the default rights, select the “Delete” tick box next to their name and permissions list. When you submit, their rights will be reset to the default rights.

Warning: A user with administrative rights can remove his own administrative rights.

1.3 Installation

These instructions will guide you through installing Pootle and its requirements in a virtual environment.

```

bash-3.2$ cd ~/dev/pootle
bash-3.2$ virtualenv env
New python executable in env/bin/python
Installing setuptools, pip...done.
bash-3.2$ source env/bin/activate
(env)bash-3.2$ pip install --upgrade pip setuptools
Downloading/unpacking pip from https://pypi.python.org/packages/b6/ac/7015eb97dc749283ffdec1c3a88ddb8ae0
3b8fad0f0e611408f196358da3/pip-9.0.1-py2.py3-none-any.whl#md5=297dbd16ef53bcef0447d245815f5144
Downloading pip-9.0.1-py2.py3-none-any.whl (1.3MB): 1.3MB downloaded
Downloading/unpacking setuptools from https://pypi.python.org/packages/3b/7e/293d19ccd106119e35db4bf3e11
1b1895098f618b455b758aa636496cf03/setuptools-28.8.0-py2.py3-none-any.whl#md5=4a5e6857bd49f1e95e916d7ca5d
5a161
Downloading setuptools-28.8.0-py2.py3-none-any.whl (472kB): 472kB downloaded
Installing collected packages: pip, setuptools
Found existing installation: pip 1.5.4
Uninstalling pip:
  Successfully uninstalled pip
Found existing installation: setuptools 2.2
Uninstalling setuptools:
  Successfully uninstalled setuptools
Successfully installed pip setuptools
Cleaning up...
(env)bash-3.2$ pip install Pootle

```



Fig. 1: Pootle installation tutorial (Click to play)

lation video tutorial, which *starts after some basic setup*, to see the installation steps in action and expected results.

For a production deployment we **strongly** recommend that you set up the following:

- *Install optional optimization packages*
- Use either a *MySQL* or *PostgreSQL* database.
- *Make use of a front-end web server*

Note: Before installing please ensure that you have all the *necessary requirements*.

Warning: It is important to install Pootle into a virtual environment to ensure the correct packages and permissions. It's even more important not to install Pootle as the root user on your system. **Installing or running Pootle as the root user will expose your system to many potential security vulnerabilities**

1.3.1 Setup assumptions

We've made some assumptions in these instructions, adjust as needed:

1. All of the *Pootle requirements* have been installed.
2. We're installing into `~/dev/pootle`.
3. We're using SQLite as it's easy to setup.
4. We're setting up the essential parts of Pootle including Redis, and RQ Workers.
5. This is a test installation on a single server, and not optimised for production use.
6. We're installing using **pip**.

1.3.2 Setting up the virtual environment

In order to install Pootle first create a virtual environment. The virtual environment allows you to install dependencies independent of your system packages.

Please install **virtualenv** from your system packages, e.g. on Debian:

```
$ sudo apt-get install python-virtualenv
```

Otherwise you can install **virtualenv** using **pip**:

```
$ sudo pip install virtualenv
```

Now create a virtual environment on your location of choice by issuing the **virtualenv** command:

```
$ cd ~/dev/pootle
$ virtualenv env
```

Note: for versions of **virtualenv** prior to 1.10, you may need to call **virtualenv** with the `--setuptools` option, to ensure the correct environment.

To activate the virtual environment run the **activate** script:

```
$ source env/bin/activate
```

Once activated the virtual environment name will be prepended to the shell prompt.

Lastly, we want to make sure that we are using the latest version of **pip** and **setuptools**:

```
(env) $ pip install --upgrade pip setuptools
```

1.3.3 Installing Pootle

Use **pip** to install Pootle into the virtual environment:

```
(env) $ pip install --pre --process-dependency-links Pootle
```

This will also fetch and install Pootle's dependencies.

Note: pip requires `--pre` to install pre-release versions of Pootle, i.e. alpha, beta and rc versions. You may require `--process-dependency-links` if Pootle depends on unreleased versions of third-party software.

To verify that everything installed correctly, you should be able to access the **pootle** command line tool within your environment.

```
(env) $ pootle --version
Pootle 2.9.0rc1 (Django 1.11.12, Translate Toolkit 2.3.0)
```

1.3.4 Initializing the Configuration

Once Pootle has been installed, you will need to initialize a configuration file:

```
(env) $ pootle init
```

By default the configuration file is saved as `~/.pootle/pootle.conf`. You can pass an alternative path as an argument if required - see the `init` command for all of the options.

Warning: This default configuration is enough to experiment with Pootle. **Don't use this configuration in a production environment.**

The initial configuration includes the settings that you're most likely to change. For further customization, see the [full list of available settings](#).

1.3.5 Running RQ worker

Statistics tracking and various other background processes are managed by [RQ](#). The `rqworker` command needs to be run continuously in order to process the jobs.

If you have not already done so you should [install and start a Redis server](#).

You can start the worker in the background with the following command:

```
(env) $ pootle rqworker &
```

In a production environment you may want to [run RQ workers as services](#).

See here for [further information about RQ jobs in Pootle](#).

1.3.6 Populating the Database

Before you run Pootle for the first time, you need to create the schema for the database and populate it with initial data. This is done by executing the `migrate` and `initdb` management commands:

Note: You will need to have an [RQ worker running](#) to complete this. Alternately, you can use the `--no-rq`.

```
(env) $ pootle migrate
(env) $ pootle initdb
```

Running `initdb` will take some time as it will create the default projects and stores.

1.3.7 Creating an admin user

Pootle needs at least one user with superuser rights which we create with the `createsuperuser` command.

```
(env) $ pootle createsuperuser
```

All users are required to verify their email before logging in. If you wish to bypass this step you can use the `verify_user` command.

For example to allow a user named `admin` to log in without having to verify their email address:

```
(env) $ pootle verify_user admin
```

1.3.8 Running Pootle

The Django default server will be enough for quickly testing the software. To run it, just issue:

```
(env) $ pootle runserver --insecure
```

Warning: There are [serious drawbacks](#) to using **runserver**. Never use it in production.

And the server will start listening on port 8000. Pootle can then be accessed from your web browser at `localhost:8000`.

1.3.9 Next steps

Now that you have Pootle up and running you may want to consider some of the following in order to build a production environment.

- *Create your first localisation project*
- *Run Pootle and RQ workers as services*
- *Set up a reverse-proxy web server for static files*
- *Use a wsgi server to serve dynamic content*
- *Check out the available settings*
- *Check out Pootle management commands*
- *Optimize your setup*
- *Set up a Translation Memory Server*
- *Customize the Pootle UI*

1.4 Upgrading

These are the instructions for upgrading Pootle from an older version to the current release.

1.4.1 Stop your running Pootle

Stop your running Pootle while you upgrade to prevent updates to your data during the migration process. If you have RQ workers running stop those also.

1.4.2 Backup your system

Warning: Before upgrading we **strongly** recommend that you *backup your current system*.

You **must** synchronize all your translations to disk:

```
(env) $ pootle sync_stores
```

1.4.3 Migrate your database

If you are currently using SQLite for your database you will need to *migrate to either MySQL (InnoDB) or PostgreSQL* before you upgrade.

1.4.4 Latest changes

Before upgrading Pootle familiarize yourself with *important changes* since the version that you are upgrading from.

1.4.5 Check Pootle requirements

You should check that you have all of the necessary *Pootle requirements* and have installed all required *system packages*.

Warning: Pootle 2.7.0 or newer requires **Python 2.7**

If you are upgrading from a virtual environment using an earlier Python version, you **must** upgrade or rebuild your virtual environment first.

1.4.6 Activate virtualenv

These instructions assume that you are using **virtualenv** and you have activated a virtual environment named `env` as follows:

```
$ source env/bin/activate
(env) $
```

1.4.7 Update pip and setuptools

You should now upgrade **pip** and **setuptools** to the latest version:

```
(env) $ pip install --upgrade pip setuptools
```

1.4.8 Upgrading from a version older than 2.6

If you are upgrading from a version older than 2.6 you will need to first upgrade to the latest 2.6.x version and then you will be able to upgrade to the latest version.


```
(env) $ pip install --upgrade "Pootle>=2.6,<2.7"
(env) $ pootle setup
```

Warning: The 2.6.x releases are meant only as a migration step.
You must upgrade immediately to the latest version once setup has completed.

1.4.9 Clean up stale Python bytecode

You should remove any stale Python bytecode files before upgrading.

Assuming you are in the root of your virtualenv folder you can run:

```
(env) $ pyclean .
```

1.4.10 Upgrading from version 2.6.x or later

Upgrade to the latest Pootle version:

```
(env) $ pip install --pre --process-dependency-links --upgrade Pootle
```

1.4.11 Update and check your settings

You should now update your custom Pootle settings to add, remove or adjust any settings that have changed. You may want to view the latest *available settings*.

You can check to see if there are any issues with your configuration settings that need to be resolved:

```
(env) $ pootle check
```

Note: If you are upgrading from a version of Pootle that uses `localsettings.py` then you may want to merge your old custom settings with your *settings conf file* (default location `~/ .pootle/pootle.conf`).

1.4.12 Start an RQ Worker

Statistics tracking and various other background processes are managed by [RQ](#). The `rqworker` command needs to be run continuously in order to process the jobs.

If you have not already done so you should *install and start a Redis server*.

You can start the worker in the background with the following command:

```
(env) $ pootle rqworker &
```

In a production environment you may want to *run RQ workers as services*.

See here for *further information about RQ jobs in Pootle*.

1.4.13 Review your database configuration

Review the *MySQL* or *PostgreSQL* installation instructions for any changes that you need to make to your database.

If you run MySQL you will almost certainly need to make sure you have [Time zone definition files](#) loaded into the database.

1.4.14 Migrate your database schema

Once you have updated your settings you can perform the database schema and data upgrade by running. This is done as follows:

```
(env) $ pootle migrate --fake-initial
```

You will also need to update the stats data held in Pootle

```
(env) $ pootle update_data
```

1.4.15 Refreshing checks

You must now update the translation checks. You will need to have an *RQ worker running* to complete this.

```
(env) $ pootle calculate_checks
```

This command will dispatch jobs to the RQ worker and may take some time.

If you wish to run *calculate_checks* in the foreground without using the RQ worker you can use the *--no-rq* option.

1.4.16 Refreshing scores

If you are upgrading from a version prior to 2.8rc6, you will need to update user scores using *refresh_scores*.

```
(env) $ pootle refresh_scores --reset
(env) $ pootle refresh_scores
```

1.4.17 Drop cached snippets

Redis might have cached HTML snippets referring to outdated static assets. In order for Pootle to return references to the newest assets these cached snippets must go away:

```
(env) $ pootle flush_cache --django-cache
```

1.4.18 Set up users

Any accounts that do not have an email address registered will not be able to log in. You can set the email for a user using the *update_user_email* command.

For example to set the email for user admin to admin@example.com:

```
(env) $ pootle update_user_email admin admin@example.com
```

As of Pootle 2.7 users must now verify their email before they can log in.

You can use the `verify_user` command to bypass email verification for a specific user.

For example to automatically verify the admin user:

```
(env) $ pootle verify_user admin
```

If you wish to verify all of your existing users please see the `verify_user` command for further options.

1.4.19 Check Pootle FS migration

Your projects should have been automatically migrated to use the Pootle FS `localfs` backend, so you need to check that everything was migrated correctly:

```
(env) $ pootle fs
```

This command lists all the available Pootle FS projects. Make sure that all the existing projects in Pootle are listed here.

```
(env) $ pootle fs state MYPROJECT
```

This command will show the state of tracked files for a specific project. Run it for each of the projects listed by the previous command.

Ideally we want the state to show no results, i.e. that all files on disk and in Pootle are in sync and are being tracked. If the migration to Pootle FS was not able to fully understand your layout then there may be untracked files.

If there are untracked files you will want do some of these steps:

1. `fs add` or `fs rm` any files that should be tracked but are currently untracked.
2. Moving and renaming files on the filesystem could resolve missing files.
3. Adding language mappings could correctly map Filesystem and Pootle stores. This is explained in the [Enable translation to a new language](#) instructions.

1.4.20 Next steps

Now that you have Pootle up and running you may want to consider some of the following in order to build a production environment.

- [Run Pootle and RQ workers as a service](#)
- [Re-apply customisations](#)
- [Optimize your setup](#)
- [Set up a Translation Memory Server](#)
- [Check out any new settings](#)
- [Check out Pootle management commands](#)

1.5 Administering a server

1.5.1 Installation

Pootle requirements

Hardware Requirements

Your Pootle installation will need to be flexible enough to handle the translation load. The recommended hardware depends highly on the performance you expect, the number of users you want to support, and the number and size of the files you want to host.

Whatever hardware you have, you will still benefit from performance improvements if you can *optimize your system*.

Your disk space should always be enough to store your files and your Pootle database, with some extra space available.

System Requirements

To run Pootle you need a computer running:

- Linux
- Mac OS X

Or, any other Unix-like system.

Note: Pootle will not run on Windows since it uses RQ, whose workers cannot run on [Windows](#).

Some developers do develop on Windows so these problems can be worked around for some of the development tasks.

Pootle should be able to run on any system that implements `fork()`.

Python version

Python 2.7 is required, 3.x is not yet supported.

Installing system packages

You will need a C compiler and the development libraries for Python and XML to be available on your system before you will be able to install your virtual environment. You will also need pip.

Eg. on a Debian-based system:

```
$ sudo apt-get install build-essential libxml2-dev libxslt-dev python-dev python-pip_
↪ zlib1g-dev
```

You will also need to access to a working Redis server to provide caching to Pootle and for managing asynchronous workers.

To install and run Redis on a Debian-based system:

```
$ sudo apt-get install redis-server
$ sudo service redis-server start
```

Note: Pootle requires a minimum Redis server version of 2.8.4. If you are using Debian Wheezy you will need to install *redis-server* from backports.

Database requirements

Make sure to install the requirements for your chosen database, either *MySQL* or *PostgreSQL*.

System requirements for customising static resources

In order to customise static resources such as CSS or JavaScript, you must install Node.js and npm.

On a Debian-based system you can install these with:

```
$ sudo apt-get install nodejs npm
```

On Debian Jessie and perhaps other distributions you also need to link the `nodejs` command to `node`:

```
$ sudo update-alternatives --install /usr/bin/node node /usr/bin/nodejs 99
```

Create a Project

Now that you have the server running, you can setup a project to translate on the server.

Our assumptions

To simplify the example we assume that:

- The project uses PO files.
- You can copy these files to the Pootle server.
- There is a template file in POT format containing the strings that need to be translated.
- The project follows the GNU layout (More information on this is provided below).
- Pootle is correctly set up and running.
- There is at least one **rqworker** thread running. This is important.
- You are logged into the Pootle server using your newly created administrator user.

Adding a new project

Placing translation files

You need to place the translation files for your new project in a location where Pootle can find, read and write them. Pootle uses the `POOTLE_TRANSLATION_DIRECTORY` setting to find out where translation files are stored on the server.

Our example project uses a GNU layout.

A GNU layout means that our project contains translation files named using language codes. Within the project there are no directories, just files. There can only be a single translation file per language in a project using this layout.

This is the simplest layout possible and the reason we are using it in our example.

Below you can see an example with two projects using the GNU layout:

```
`-- translations
  |-- project1
  |   |-- po
  |       |-- de.po
  |       |-- fr.po
  |       |-- gl.po
  |       |-- pt_BR.po
  |       |-- templates.pot
  |-- project2
  |   |-- po
  |       |-- af.po
  |       |-- eu.po
  |       |-- pt_BR.po
  |       |-- templates.pot
  |       |-- zu.po
```

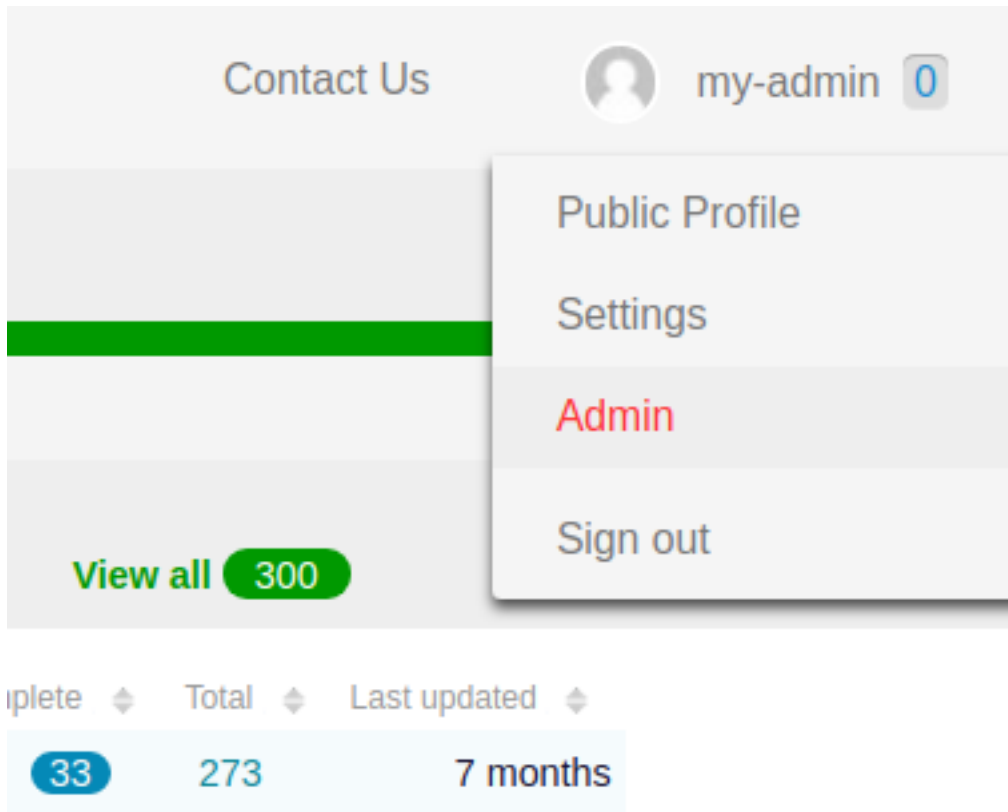
Among the regular translation files there are two files named `templates.pot`. These are the template (master or reference) files that contain the original strings. Usually these files contain only English strings, however it is much less confusing to use the term `templates` than e.g. `en` or `English`.

To get started, create a `my-project` directory in the location pointed to by [POOTLE_TRANSLATION_DIRECTORY](#) and place within it the translation files for your new project. Make sure you have a `templates.pot` among those project translation files.

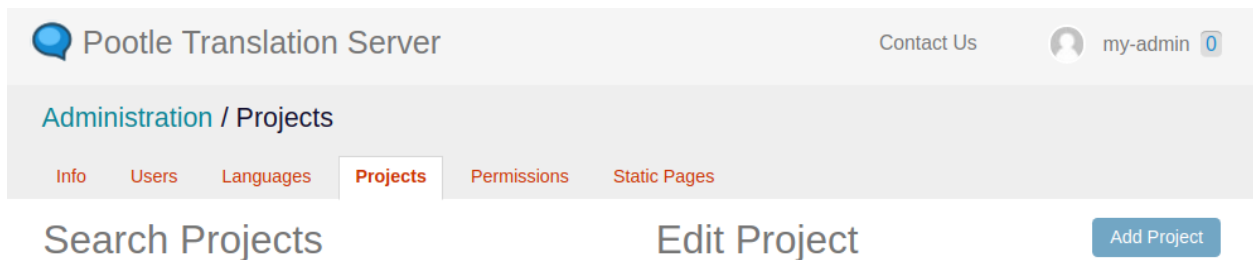
Creating the project

Note: If you want to integrate your project with a version control system you need to *install the Pootle FS plugins for the VCS* and *connect Pootle FS with the VCS repository* before creating the project in Pootle.

At the top of the user interface, you will see your newly created administrator username. Click on it and the main top menu will be displayed, then click on **Admin** (highlighted in red):



Now you are in the administration interface. Go to the **Projects** tab and you will see a **New Project** button:



Click on that button and the **Add Project** form will be displayed. Enter the new project's details.

Code is a unique string that identifies the project. Also you must specify the **File Types** used in this new project.

The **Filesystem backend** tells Pootle FS how to synchronize the translations in the project. The default option is `localfs` which tells to synchronize with the disk. Other backends allow to synchronize with version control systems and they might require the installation of additional Pootle FS plugins.

Path or URL either is a path pointing to the translation files on the disk if you are using the `localfs` **Filesystem backend**, or is a URL pointing to the VCS repository. The path can be an absolute path like `/path/to/translations/my-project/` or it can use the `{POOTLE_TRANSLATION_DIRECTORY}` placeholder if you are using the `localfs` **Filesystem backend**, like `{POOTLE_TRANSLATION_DIRECTORY}my-project` (the placeholder will be transparently replaced by the value of the `POOTLE_TRANSLATION_DIRECTORY` setting). If you are using a **Filesystem backend** that uses a URL instead of a Path, for example for the **Git** backend, this would be like `git@github.com:user/repo.git` for a GitHub repository.

The **Path mapping** field specifies the project layout using a glob like `/path/to/translation/files/<language_code>/<dir_path>/<filename>.<ext>` that must start with `/`, end with `.<ext>`, and con-

tain `<language_code>` (the rest of the placeholders are optional). If you are using the `localfs` **Filesystem backend** the **Path mapping** will be combined with the path specified in the **Path or URL** field. For other backends it will be relative to the root of the repository. Note you can easily fill this field by selecting one of the available **Path mapping presets**.

You can also provide a **Full Name** easily readable for humans, if not the **Code** will be used. You don't need to change the rest of the fields unless you need to further customize your project.

Note: If you are creating a project that is integrated with a version control repository then configure as follows:

- Set the **Filesystem backend** to the required VCS backend.
 - Set the **Path or URL** to point to the repository, e.g. `git@github.com:user/repo.git` for a GitHub repository.
-

In our example set the following:

- **Code** to `my-project`.
- **Full Name** to `My project`
- **File Types** to `Gettext PO (po/pot)`.
- **Filesystem backend** to `localfs`.
- **Path or URL** to `{POOTLE_TRANSLATION_DIRECTORY}my-project`.
- **Path mapping preset** to `GNU style`.

Add Project

[Cancel](#)

Code	<input type="text" value="my-project"/>
Full Name	<input type="text" value="My Project"/>
Quality Checks	<input type="text" value="standard"/>
File types	<div>× Gettext PO (po/pot)</div>
Filesystem backend	<input type="text" value="localfs"/>
Path or URL	<input type="text" value="{POOTLE_TRANSLATION_DIRECTORY}my-project"/>
Path mapping preset	<input type="text" value="GNU style"/>
Path mapping	<input type="text" value="/po/<language_code>.<ext>"/>
Template name	<input type="text" value="templates"/>
Source Language	<input type="text" value="English - en"/>
Ignore Files	<input type="text"/>
String Errors Contact	<input type="text"/>
Screenshot Search Prefix	<input type="text"/>
Disabled	<input type="checkbox"/>

Save

Once you are done click on the **Save** button below the form to create the project and save its Pootle FS configuration.

Specify language codes mapping

If not all of your project's language codes **do not match** those available in Pootle, then you must add language mapping configurations for those languages. You can do that by clicking on the **Languages** link that is displayed below your project form:

Source Language	<div>English - en</div>
Ignore Files	<div></div>
String Errors Contact	<div></div>
Screenshot Search Prefix	<div></div>
Disabled	<div><input type="checkbox"/></div>
	<div>Save</div>
	<div>Overview</div>
	<div>Languages</div>
	<div>Permissions</div>

☐

Delete

The existing languages enabled for the project are listed, including an **optional** mapping to the language code used in the filesystem. In the screenshot below you can see that `fr_FR` on filesystem maps to `fr` on Pootle:

Language Filesystem language code Delete:

German - de	<input type="text"/>	<input type="checkbox"/>
French - fr	fr-FR	<input type="checkbox"/>
Galician - gl	<input type="text"/>	<input type="checkbox"/>
Portuguese (Brazil) - pt_BR	<input type="text"/>	<input type="checkbox"/>
Templates - templates	<input type="text"/>	<input type="checkbox"/>

Arabic - ar

Language Filesystem language code Delete

Save Changes

In our example this form won't list any language, so you must add new entries for the languages that need to be mapped specifying their **Filesystem language code** field.

Import the translations

Creating the project doesn't actually import all the translations to Pootle. To do that you need to run the following on the command line of the Pootle server:

```
(env) $ pootle fs fetch my-project
(env) $ pootle fs state my-project
(env) $ pootle fs add my-project
(env) $ pootle fs sync my-project
```

Note: Read the [using Pootle FS](#) instructions in order to learn more about Pootle FS usage.

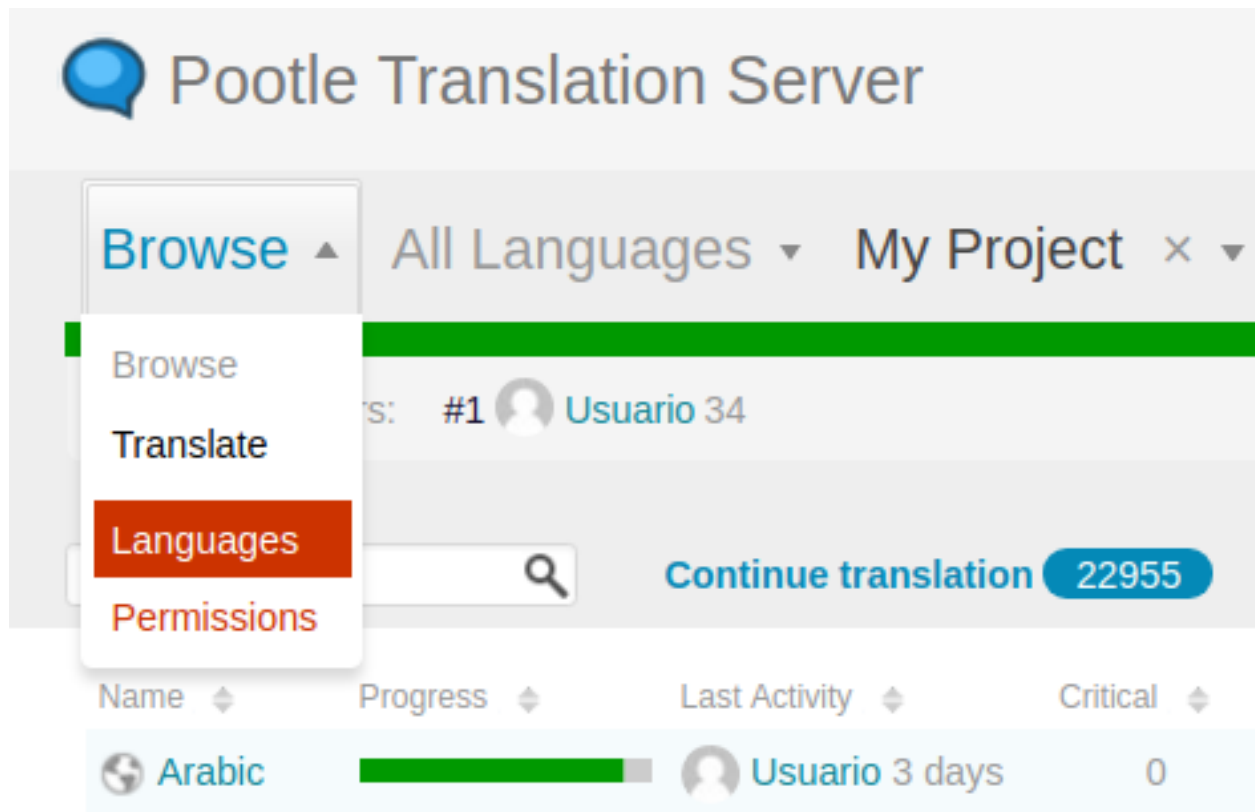
This will import all the translations from disk into Pootle, calculate the translation statistics and calculate the quality check failures. This might take a while for a large project.

Looking at your new project you will see that Pootle has imported all the existing translations for the existing languages that you copied to the `my-project` directory within `POOTLE_TRANSLATION_DIRECTORY`.

Enable translation to a new language

When you want to add a new language to the project, follow these steps.

Go to your project overview and select **Languages** in the navigation dropdown:



Note: Alternatively you can get the same result by clicking on the **Languages** link that is displayed below your project form in the administration interface:

Source Language

Ignore Files

String Errors Contact

Screenshot Search Prefix

Disabled ☐



[Overview](#)

[Languages](#)

[Permissions](#)

☐

The existing languages enabled for the project are listed, including an **optional** mapping to the language code used in the filesystem (in the screenshot below you can see that `fr_FR` on filesystem maps to `fr` on Pootle). In our example we are adding **Arabic** to the project:

 Pootle Translation Server Contact Us  My Admin 34

[Languages](#) ▾ [All Languages](#) ▾ [My Project](#) × ▾

Here you can add, edit, or delete the languages for this project.

Language ↕	Filesystem language code ↕	<input type="checkbox"/> Delete:
German - de	<input type="text"/>	<input type="checkbox"/>
French - fr	<input type="text" value="fr-FR"/>	<input type="checkbox"/>
Galician - gl	<input type="text"/>	<input type="checkbox"/>
Portuguese (Brazil) - pt_BR	<input type="text"/>	<input type="checkbox"/>
Templates - templates	<input type="text"/>	<input type="checkbox"/>
<input type="text" value="Arabic - ar"/>	<input type="text"/>	

Language Filesystem language code Delete

When you click the **Save** button the new language will be added for translation. In large projects it may take some time to create the new translation files from the templates.

Note: If you want to enable translation to a language that doesn't yet exist in your Pootle instance, then you will first have to add the language in the **Languages** tab in the administration interface, in a similar way to [creating a new project](#).

Once the language is created you can enable translation to that new language in any project by following the instructions above.

Updating strings for existing project

Whenever developers introduce new strings, deprecate older ones, or change some of them this impacts Pootle and the languages being translated.

When any of these changes occur, you will need to generate a new `templates.pot` and use it to bring the translations in Pootle up-to-date with the new templates.

Once you have created the new `templates.pot` place it within your project's directory in `POOTLE_TRANSLATION_DIRECTORY`, replacing the file with the same name. After that, invoke the following command which will update the template translations in the Pootle database.

```
(env) $ pootle update_stores --project=my-project --language=templates
```

This command will ensure that new strings are added to the project and any strings which have been removed are marked as deprecated, and thus will not be available for translation.

Now each of the languages will need to be brought into sync with the template language. The first step is to save all the Pootle translations to disk:

```
(env) $ pootle sync_stores --project=my-project
```

Then update all those translations on disk against the newer templates. We recommend you to update them on disk using the `pot2po` command line tool because it can handle other formats besides Gettext PO.

```
(env) $ cd $POOTLE_TRANSLATION_DIRECTORY # Use the actual path!
(env) $ cd my-project
(env) $ pot2po -t af.po templates.pot af.po # Repeat for each language by changing_
↪the language code.
```

Note: To preserve the existing translations we pass the previous translation file to the `-t` option.

When all the languages in the project have been updated you can push them back to Pootle:

```
(env) $ pootle update_stores --project=my-project
```

Note: If your project languages contain many translations you might want to perform the update against newer templates on a language by language basis.

Installation with MySQL

These instructions provide additional steps for setting up Pootle with MySQL.

You should read the [full installation instructions](#) in order to install Pootle.

Pootle supports the [versions of MySQL supported by Django](#), make sure that your installed version is supported.

Setting up the database

Use the **mysql** command to create the user and database:

```
$ mysql -u root -p # You will be asked for the MySQL root password to log in

> CREATE DATABASE pootledb CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci;
> GRANT ALL PRIVILEGES ON pootledb.* TO pootle@localhost IDENTIFIED BY 'secret';
> FLUSH PRIVILEGES;
```

System software requirements

In addition to the [system packages](#) set out in the general installation requirements you will also require the MySQL client development headers in order to build the Python bindings, e.g. on a Debian-based system:

```
$ sudo apt-get install libmysqlclient-dev
```

Installing MySQL Python driver

Once you have [set up and activated your virtual environment](#), you will need to install the MySQL driver.

You can do so as follows:

```
(env) $ pip install --pre --process-dependency-links Pootle[mysql]
```

Initializing the Configuration

When [initializing your configuration](#) you can specify params to set up your database, e.g.:

```
(env) $ pootle init --db mysql --db-name pootledb --db-user pootle
```

This will create a configuration file to connect to a MySQL database named `pootledb` hosted on localhost as the user `pootle`. Please see the `init` command for all of the available options.

You will most likely want to edit your Pootle configuration (default location: `~/.pootle/pootle.conf`) to set your password.

Adding timezone definitions

Pootle makes use of time zones, follow Django's instructions to [load time zone tables into the MySQL database](#). This needs to be done just once for your MySQL server, not per database.

Database backend

A Note on Persistent Connections

MySQL terminates idle connections after `wait_timeout` seconds. Thus setting `CONN_MAX_AGE` to a lower value will be fine (it defaults to 0). Persistent connections where `CONN_MAX_AGE` is `None` can't be used with MySQL.

To learn more please check Django's docs on [persistent connections and connection management](#).

```
DATABASES = {
    'default': {
        ...
        'CONN_MAX_AGE': 0,
        ...
    }
}
```

Installation with PostgreSQL

These instructions provide additional steps for setting up Pootle with PostgreSQL.

You should read the [full installation instructions](#) in order to install Pootle.

Pootle supports the [versions of PostgreSQL supported by Django](#), make sure that your installed version is supported.

Setting up the database

As the postgres user you must create a database and database user:

```
$ sudo su postgres # On Ubuntu, may be different on your system
postgres@ $ createuser -P pootle # This will ask you to define the users password.
postgres@ $ createdb --encoding='utf8' --locale=en_US.utf8 --template=template0 --
➔owner=pootle pootledb
```

System software requirements

In addition to the [system packages](#) set out in the general installation requirements you will also require the PostgreSQL client development headers in order to build the Python bindings, e.g. on Debian Jessie:

```
$ sudo apt-get install postgresql-server-dev-9.4
```

Installing PostgreSQL Python bindings

Once you have [set up and activated your virtual environment](#), you will need to install the PostgreSQL bindings.

You can do so as follows:

```
(env) $ pip install --pre --process-dependency-links Pootle[postgresql]
```

Initializing the Configuration

When [initializing your configuration](#) you can specify params to set up your database, e.g.:

```
(env) $ pootle init --db postgresql --db-name pootledb --db-user pootle
```

This will create a configuration file to connect to a PostgreSQL database named `pootledb` hosted on localhost as the user `pootle`. Please see the [init](#) command for all of the available options.

You will most likely want to edit your Pootle configuration (default location: `~/.pootle/pootle.conf`) to set your password.

Database backend

A Note on Persistent Connections

The default value for `CONN_MAX_AGE` is 0. It means that Django creates a connection before every request and closes it at the end. PostgreSQL supports persistent connections, and it will be fine to set `CONN_MAX_AGE` to `None`.

To learn more please check Django's docs on [persistent connections and connection management](#).

```
DATABASES = {
    'default': {
        ...
        'CONN_MAX_AGE': None,
        ...
    }
}
```

Running under a Web Server

Running Pootle with a front end web server will improve performance, give you more flexibility, and might be better for security. It is strongly recommended to run Pootle under Apache, Nginx, or a similar web server.

Running Pootle and RQ workers as a Service

If you plan to run Pootle and/or RQ workers as system services, you can use whatever software you are familiar with for that purpose. For example [Supervisor](#), [Circus](#) or [daemontools](#) might fit your needs.

Running under Apache

You can use Apache either as a reverse proxy or straight with `mod_wsgi`.

Proxying with Apache

If you want to reverse proxy through Apache, you will need to have `mod_proxy` installed for forwarding requests and configure it accordingly.

```
ProxyPass / http://localhost:8000/
ProxyPassReverse / http://localhost:8000/
ProxyPreserveHost On
```

Apache with `mod_wsgi`

Make sure to review your global Apache settings (something like `/etc/apache2/httpd.conf` or `/etc/httpd/conf/httpd.conf`) for the server-pool settings. The default settings provided by Apache are too high for running a web application like Pootle. The ideal settings depend heavily on your hardware and the number of users you expect to have. A moderate server with 1GB memory might set `MaxClients` to something like 20, for example.

Make sure Apache has read access to all of Pootle's files and write access to the `POOTLE_TRANSLATION_DIRECTORY` directory.

Note: Most of the paths present in the examples in this section are the result of deploying Pootle using a Python virtualenv as told in the *Setting up the Environment* section from the *Quickstart installation* instructions.

If for any reason you have different paths, you will have to adjust the examples before using them.

For example the path `/var/www/pootle/env/lib/python2.7/site-packages/` will be different if you have another Python version, or if the Python virtualenv is located in any other place.

First it is necessary to create a WSGI loader script:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os
import site
import sys

# You probably will need to change these paths to match your deployment,
# most likely because of the Python version you are using.
ALLDIRS = [
    '/var/www/pootle/env/lib/python2.7/site-packages',
    '/var/www/pootle/env/lib/python2.7/site-packages/pootle/apps',
]

# Remember original sys.path.
prev_sys_path = list(sys.path)

# Add each new site-packages directory.
for directory in ALLDIRS:
    site.addsitedir(directory)

# Reorder sys.path so new directories at the front.
new_sys_path = []

for item in list(sys.path):
    if item not in prev_sys_path:
        new_sys_path.append(item)
        sys.path.remove(item)

sys.path[:0] = new_sys_path

# Set the Pootle settings module as DJANGO_SETTINGS_MODULE.
os.environ['DJANGO_SETTINGS_MODULE'] = 'pootle.settings'

# Set the WSGI application.
def application(environ, start_response):
    """Wrapper for Django's WSGIHandler().

    This allows to get values specified by SetEnv in the Apache
    configuration or interpose other changes to that environment, like
    installing middleware.
    """
    try:
        os.environ['POOTLE_SETTINGS'] = environ['POOTLE_SETTINGS']
    except KeyError:
```

(continues on next page)

(continued from previous page)

```

pass

from django.core.wsgi import get_wsgi_application
_wsgi_application = get_wsgi_application()
return _wsgi_application(environ, start_response)

```

Place it in `/var/www/pootle/wsgi.py`. If you use a different location remember to update the Apache configuration accordingly.

A sample Apache configuration with `mod_wsgi` might look like this:

```

WSGIRestrictEmbedded On
WSGI PythonOptimize 1

<VirtualHost *:80>
    # Domain for the Pootle server. Use 'localhost' for local deployments.
    #
    # If you want to deploy on example.com/your-pootle/ rather than in
    # my-pootle.example.com/ you will have to do the following changes to
    # this sample Apache configuration:
    #
    # - Change the ServerName directive to:
    #   ServerName example.com
    # - Change the WSGIScriptAlias directive to (note that /your-pootle must
    #   not end with a slash):
    #   WSGIScriptAlias /your-pootle /var/www/pootle/wsgi.py
    # - Change the Alias directive for 'assets' to include the '/your-pootle'.
    # - Include the following settings in your custom Pootle settings:
    #   STATIC_URL = '/your-pootle/assets/'
    #   FORCE_SCRIPT_NAME = '/your-pootle'
    # - If you have previously calculated the stats:
    #   - Restart the RQ workers.
    #   - Run refresh_stats to recalculate the stats data.
    ServerName my-pootle.example.com

    # Set the 'POOTLE_SETTINGS' environment variable pointing at your custom
    # Pootle settings file. An initial settings file can be created using
    # 'pootle init'
    #
    # This might require enabling the 'env' module.
    SetEnv POOTLE_SETTINGS /var/www/pootle/your_custom_settings.conf

    # The following two optional lines enable the "daemon mode" which
    # limits the number of processes and therefore also keeps memory use
    # more predictable.
    WSGIDaemonProcess pootle processes=2 threads=3 stack-size=1048576 maximum-
    ↪requests=500 inactivity-timeout=300 display-name=%{GROUP} python-path=/var/www/
    ↪pootle/env/lib/python2.7/site-packages
    WSGIProcessGroup pootle

    # Point to the WSGI loader script.
    WSGIScriptAlias / /var/www/pootle/wsgi.py

    # Turn off directory listing by default.
    Options -Indexes

```

(continues on next page)

(continued from previous page)

```

# Compress before being sent to the client over the network.
# This might require enabling the 'deflate' module.
SetOutputFilter DEFLATE
AddOutputFilterByType DEFLATE text/html text/css text/plain text/xml application/
↪x-javascript

# Set expiration for some types of files.
# This might require enabling the 'expires' module.
ExpiresActive On

ExpiresByType image/jpg "access plus 10 years"
ExpiresByType image/png "access plus 10 years"
ExpiresByType text/css "access plus 10 years"
ExpiresByType application/x-javascript "access plus 10 years"

# Optimal caching by proxies.
# This might require enabling the 'headers' module.
Header set Cache-Control "public"

# Directly serve static files like css and images, no need to go
# through mod_wsgi and Django. For high performance consider having a
# separate server.
Alias /assets /var/www/pootle/env/lib/python2.7/site-packages/pootle/assets
<Directory /var/www/pootle/env/lib/python2.7/site-packages/pootle/assets>
    Require all granted
    # For apache 2.2, comment the line directly above, and uncommend the two
↪lines directly below
    #Order deny,allow
    #Allow from all
</Directory>

</VirtualHost>

```

You can find more information in the [Django docs](#) about Apache and mod_wsgi.

.htaccess

If you do not have access to the main Apache configuration, you should still be able to configure things correctly using the `.htaccess` file.

[More information](#) on configuring `mod_wsgi` (including `.htaccess`)

Running under Nginx

Running Pootle under a web server such as Nginx will improve performance. For more information about Nginx and WSGI, visit [Nginx's WSGI module page](#)

A Pootle server is made up of static and dynamic content. By default Pootle serves all content, and for low-latency purposes it is better to get other webserver to serve the content that does not change, the static content. It is just the issue of low latency and making the translation experience more interactive that calls you to proxy through Nginx. The following steps show you how to setup Pootle to proxy through Nginx.

Proxying with Nginx

The default Pootle server runs at port 8000 and for convenience and simplicity does ugly things such as serving static files — you should definitely avoid that in production environments.

By proxying Pootle through nginx, the web server will serve all the static media and the dynamic content will be produced by the app server.

```
server {
    listen 80;
    server_name pootle.example.com;

    access_log /path/to/pootle/logs/nginx-access.log;
    gzip on; # Enable gzip compression

    charset utf-8;

    location /assets {
        alias /path/to/pootle/env/lib/python2.6/site-packages/pootle/assets/;
        expires 14d;
        access_log off;
    }

    location / {
        proxy_pass http://localhost:8000;
        proxy_redirect off;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

1.5.2 Upgrading

Database Migration

Note: Please note that the database migration must be performed before upgrading Pootle.

Using `dumpdata` and `loaddata` commands to migrate between databases is no longer supported.

The MySQL MyISAM backend is no longer supported. Use [InnoDB](#) instead.

There are several tools available to migrate between databases. We recommend having a look through this list for the following supported backends:

- PostgreSQL
- SQLite
- MySQL/MariaDB (InnoDB)

1.5.3 Performance tuning and managing the server

Settings

You will find all the Pootle-specific settings in this document.

If you have upgraded, you might want to compare your previous copy to the one distributed with the Pootle version for any new settings you might be interested in.

Customizing Settings

When starting Pootle with the **pootle** runner script, by default it will try to load custom settings from the `~/ .pootle/pootle.conf` file. These settings will override the defaults set by Pootle.

An alternative location for the settings file can be specified by setting the `-c </path/to/settings.conf/>` flag when executing the runner. You can also set the `POOTLE_SETTINGS` environment variable to specify the path to the custom configuration file. The environment variable will take precedence over the command-line flag.

Available Settings

This is a list of Pootle-specific settings grouped by the file they're contained and ordered alphabetically.

10-base.conf

This file contains base configuration settings.

POOTLE_INSTANCE_ID Instance ID. This is to differentiate multiple instances of the same app (e.g. development, staging and production). By default this value is exposed as a global `<html>` class name to allow overriding CSS rules based on the instance type.

POOTLE_TITLE Default: 'Pootle Translation Server'

The name of the Pootle server.

POOTLE_SQL_MIGRATIONS Default: True

New in version 2.8.

Some migrations have been optimized by using SQL directly and bypassing the Django ORM (Currently only for mysql migrations).

This could cause problems if, for example, you have developed custom models with foreign keys to Pootle models.

20-backends.conf

Backend and caching settings.

POOTLE_CACHE_TIMEOUT Default: 604800 (a week)

New in version 2.7.

Time in seconds to keep certain objects cached in memory (template fragments, language and project lists, permissions, etc.).

Note that for anonymous users Pootle also uses [Django's caching middleware](#), and its settings can be configured separately.

25-logging.conf

POOTLE_LOG_DIRECTORY Default: `working_path('log')`

New in version 2.7.

The directory where Pootle writes event logs to. These are high-level logs of events on store/unit changes and **pootle** commands executed.

30-site.conf

Site-specific settings.

POOTLE_CONTACT_ENABLED Default: `True`

Controls whether users will be able to use the contact form. The address to receive messages is controlled by [POOTLE_CONTACT_EMAIL](#).

POOTLE_CONTACT_EMAIL Default: `info@YOUR_DOMAIN.com`

Address to receive messages sent through the contact form. This will only have effect if [POOTLE_CONTACT_ENABLED](#) is set to `True`.

POOTLE_CONTACT_REPORT_EMAIL Default: `POOTLE_CONTACT_EMAIL`

New in version 2.7.

Email address to report errors on strings.

POOTLE_EMAIL_FEEDBACK_ENABLED Default: `False`

New in version 2.8.

Controls whether emails are sent to suggesters when a reviewer accepts or rejects their suggestions providing some comment for the suggester.

POOTLE_CANONICAL_URL Default: `http://localhost`

New in version 2.8.

Canonical URL, **without trailing slash**, used when deriving the URLs to send out emails. If you use the `django.contrib.sites` framework set this to blank string.

POOTLE_CUSTOM_LOGO Default: `" "`

New in version 2.8.

Custom logo URL - this can be an absolute or relative URL.

POOTLE_FAVICONS_PATH Default: `"/assets/favicon"`

New in version 2.8.

Customisable favicon path. Should not end with trailing slash.

40-apps.conf

Configuration settings for applications used by Pootle.

POOTLE_SIGNUP_ENABLED Default: True

Changed in version 2.7.

Controls whether user sign ups are allowed or not. If set to False, administrators will still be able to create new user accounts.

POOTLE_CUSTOM_TEMPLATE_CONTEXT Default: {}

Changed in version 2.7.

Custom template context dictionary. The values will be available in the templates as `{{ custom.<key> }}`.

POOTLE_LEGALPAGE_NOCHECK_PREFIXES Default: ('/about', '/accounts', '/admin', '/contact', '/jsi18n', '/pages',)

Changed in version 2.7.

List of path prefixes where the `LegalAgreementMiddleware` will check if the current logged-in user has agreed all the legal documents defined for the Pootle instance. Don't change this unless you know what you're doing.

POOTLE_META_USERS Default: ()

New in version 2.7.

Additional meta, or non-human, accounts. Pootle already manages the 'system' and 'nobody' users who own system updates to translations and submissions by anonymous users. These meta accounts have their own simple public profiles and won't track scores.

POOTLE_MARKUP_FILTER Default: (None, {})

Two-tuple defining the markup filter to apply in certain textareas.

- Acceptable values for the first element are `markdown` and `html` (deprecated).
- The second element should be a dictionary of keyword arguments that will be passed to the markup function

Changed in version 2.8: Support for `textile`, `restructuredtext` and `html` formats has been deprecated.

Examples:

```
POOTLE_MARKUP_FILTER = ('markdown', {})
```

```
POOTLE_MARKUP_FILTER = ('markdown', {
    'clean': {
        'extra_tags': ['div'],
        'extra_attrs': {
            '*': ['style']
            'img': ['alt'],
        },
        'extra_styles': [
            'color',
            'font-weight'
        ],
    },
})
```


POOTLE_CAPTCHA_ENABLED Default: True

Enable spam prevention through a captcha.

POOTLE_REPORTS_MARK_FUNC Default: '' (empty string)

New in version 2.7.

The graph of a user's activity, within reports, can be **marked** to indicate events by using this function. The setting must contain an import path to such a marking function (string).

The function receives the user and graph ranges and returns an array of applicable marks.

Parameters:

- `username` - user for whom we're producing this graph
- `start` (datetime) - start date of the graph
- `end` (datetime) - end date of the graph

The function must return an **array of dictionaries** (marks), where every mark has the following properties:

- `position`, specifying the point in the x-axis where the mark should be set (UNIX timestamp multiplied by 1000), and
- `label` specifying the text that will be displayed next to the mark.

POOTLE_SCORES Default:

```
{
    'suggestion_add': 0,
    'suggestion_accept': .1,
    'suggestion_reject': .1,
    'comment_updated': .1,
    'target_updated': .3,
    'state_translated': .6,
    'state_fuzzy': .1,
    'state_unfuzzy': .1,
}
```

New in version 2.8.

The default score is based on the wordcount of the source text. The values of the various parameters are used as a multiplier to arrive at the score attributed to the translators and reviewers.

Thus:

```
score = source wordcount * multiplier
```

When a translator translates a 10 word string they would get a score using the `state_translated` multiplier of 0.6.

```
6 = 10 words * 0.6
```

Parameters:

- `suggestion_*` - scoring for the events of adding a new suggestions, accepting or rejecting existing suggestions. By default adding a suggestion gives no score, to prevent users from gaming the system.
- `comment_*` - making changes to the string comment.
- `target_*` - adjusting the existing translation.
- `state_*` - changing the state of the string, this includes translation and fuzzy setting.

POOTLE_FS_WORKING_PATH Default: `working_path('.pootle_fs/tmp/')`

New in version 2.8.

The directory that Pootle FS uses to store temporary data for handling the projects.

Warning: This directory can potentially get very large, so you need to place it somewhere with plenty of room.

60-translation.conf

Translation environment configuration settings.

AMAGAMA_URL Default: `https://amagama-live.translatehouse.org/api/v1/`

URL to an amaGama Translation Memory server. The default service should work fine, but if you have a custom server set it here.

This URL must point to the public API URL which returns JSON. Don't forget the trailing slash.

AMAGAMA_SOURCE_LANGUAGES Default: `('en', 'en-US', 'en-US')`

List of available source languages in amaGama server pointed to by `AMAGAMA_URL`.

POOTLE_SYNC_FILE_MODE Default: `0o644`

Changed in version 2.7.

On POSIX systems, files synchronized to disk will be assigned this permission. Use `0o644` for publically-readable files or `0o600` if you want only the Pootle user to be able to read them.

POOTLE_TM_SERVER New in version 2.7.

Changed in version 2.7.3: Added the *WEIGHT* and *MIN_SIMILARITY* options. Also added another default TM used to import external translations from files.

Default: `{}` (empty dict)

Example configuration for local/external TM server:

```
{
  'local': {
    'ENGINE': 'pootle.core.search.backends.ElasticSearchBackend',
    'HOST': 'localhost',
    'PORT': 9200,
    'INDEX_NAME': 'translations',
    'WEIGHT': 1,
  },
  'external': {
    'ENGINE': 'pootle.core.search.backends.ElasticSearchBackend',
    'HOST': 'localhost',
    'PORT': 9200,
    'INDEX_NAME': 'external-translations',
    'WEIGHT': 0.9,
  },
}
```

This is configured to access a standard Elasticsearch setup. Change the settings for any non-standard setup. Change `HOST` and `PORT` settings as required.

The default `local` TM is automatically updated every time a new translation is submitted. The other TMs are not automatically updated so they can be trusted to provide selected high quality translations. Every TM server must have its own unique `INDEX_NAME`. `WEIGHT` provides a weighting factor to alter the final score for TM results from this TM server. Valid values are between 0.0 and 1.0, both included. Defaults to 1.0 if not provided. `MIN_SIMILARITY` serves as a threshold value to filter out results that are potentially too far from the source text. The Levenshtein distance is considered when measuring how similar the text is from the source text, and this represents a real value in the (0..1) range, 1 being 100% similarity. The default value (0.7) should work fine in most cases, although your mileage might vary.

POOTLE_MT_BACKENDS Default: [] (empty list)

This setting enables translation suggestions through several online services.

The elements for the list are two-element tuples containing the name of the service and an optional API key.

Available options are:

GOOGLE_TRANSLATE: Google Translate service. For this service you need to obtain an API key. Note that Google Translate API is a [paid service](#).

YANDEX_TRANSLATE: Yandex.Translate service. For this service you need to [obtain a Yandex API key](#).

PARSE_POOL_CULL_FREQUENCY Default: 4

When the pool fills up, 1/PARSE_POOL_CULL_FREQUENCY number of files will be removed from the pool.

PARSE_POOL_SIZE Default: 40

To avoid rereading and reparsing translation files from disk on every request, Pootle keeps a pool of already parsed files in memory.

Larger pools will offer better performance, but higher memory usage (per server process).

POOTLE_TRANSLATION_DIRECTORY Default: `working_path('translations')`

The directory where projects hosted on Pootle store their translation files. `sync_stores` will write to this directory and `update_stores` will read from this directory.

POOTLE_WORDCOUNT_FUNC Default: `translate.storage.statsdb.wordcount`

New in version 2.7.

The import path to a function that provides wordcounts for Pootle.

Current options:

- Translate Toolkit (default) - `translate.storage.statsdb.wordcount`

Adding a custom function allows you to alter how words are counted.

Warning: Changing this function requires that you recalculate the associated wordcounts.

Deprecated Settings

ENABLE_ALT_SRC Deprecated since version 2.5: Alternate source languages are now on by default. This ensures that translators have access to as much useful information as possible when translating.

POOTLE_TOP_STATS_CACHE_TIMEOUT Deprecated since version 2.7: The overview page statistics rewrite has removed these statistics and the RQ based statistics has also removed the load of this type of data so this setting has been removed.

VCS_DIRECTORY Deprecated since version 2.7: Version Control Support has been removed from Pootle. We feel we can support version control better in future. You can currently make use of *sync_stores* and *update_stores* to automate your own integration.

CONTRIBUTORS_EXCLUDED_NAMES Deprecated since version 2.7: The contributors page has been removed and is being replaced with better user statistics.

CONTRIBUTORS_EXCLUDED_PROJECT_NAMES Deprecated since version 2.7: The contributors page has been removed and is being replaced with better user statistics.

MIN_AUTOTERMS Deprecated since version 2.7: Terminology auto-extraction feature has been removed.

MAX_AUTOTERMS Deprecated since version 2.7: Terminology auto-extraction feature has been removed.

DESCRIPTION Deprecated since version 2.7: Pootle no longer displays site description on the landing page, but rather makes use of static pages to convey information to users in the sidebar. Use *static pages* and *customization* if you want to give users information about the Pootle site.

FUZZY_MATCH_MAX_LENGTH Deprecated since version 2.7: Update against templates feature has been removed.

FUZZY_MATCH_MIN_SIMILARITY Deprecated since version 2.7: Update against templates feature has been removed.

EXPORTED_DIRECTORY_MODE Deprecated since version 2.7: Offline translation support was rewritten and the setting was unused.

POOTLE_QUALITY_CHECKER Deprecated since version 2.8: To simplify checker code, the ability to have a custom quality checker was removed. To create custom checks, write them within the Translate Toolkit.

POOTLE_SCORE_COEFFICIENTS Removed in version 2.8: A rewrite of the scoring in Pootle now uses `POOTLE_SCORE` to store score adjustment coefficients.

Logging

Pootle's default logging has configurations for all important aspects of the server that we want to log. Pootle also logs to the 'action' logger that will log every user, system and command action executed on the server.

Log directory

You can override the default logging directory by specifying the `POOTLE_LOG_DIRECTORY` setting.

Action logger

The action logger logs each activity related to translation, units changes, store changes, command execution and other activities.

The generic log message is as follows (though some actions do produce slightly different log entries):

```
[2015-05-04T15:06:39]    system    X    pootle update_tmserver
```

That is:

```
[date] user type message
```

Action types

Current action types are as follows:

Action	Group	Description
A	Translation	Translation submission added a translation
C	Translation	An existing translation was changed
D	Translation	An existing translation was deleted
UA	Unit	A new unit was added
UO	Unit	An existing unit was made obsolete
UR	Unit	An obsolete unit was resurrected i.e. reinstated
UD	Unit	An existing unit was deleted
SA	Store	A new store was added
SO	Store	An existing store was made obsolete
SR	Store	An obsolete store was reinstated
SD	Store	An existing store was deleted
X	Command	A <code>pootle</code> command was executed
QM	Quality check	A quality check was muted (marked as a false positive)
QU	Quality check	A quality check was unmuted (reenabled after having been muted)
SC	Score	A users score has changed because of an action

Score Translation Actions

In addition the SC action type also has its own actions which track the actual type of activity that leads to changes in translation. These are used to track scores for the translators.

Action	Description
TA	unit translated
TE	unit edited after someone else
TX	unit edited after themselves
TD	translation deleted by admin
R	translation reviewed
TF	translation's fuzzy flag is set by admin
XE	translation penalty [when translation deleted]
XR	translation penalty [when review canceled]
S	suggestion added
SA	suggestion accepted (counted towards the suggestion author)
SR	suggestion rejected (counted towards the suggestion author)
RA	suggestion accepted (counted towards the reviewer)
RR	suggestion rejected (counted towards the reviewer)

Action messages

Various of the action groups have different message structures as outlined here:

Translation:

```
date  user  action  lang    unit    path      translation
[2015-05-19T14:11:18]  admin   C        af       2        /af/tutorial/stats-test.po  ↵
↪ Twee
```

(continues on next page)

(continued from previous page)

[2015-05-19T14:12:17]	admin	A	af	3	/af/tutorial/stats-test.po	↵
↪ Drie						
[2015-05-19T14:13:05]	admin	D	af	1	/af/tutorial/stats-test.po	

Unit:

date	user	action	language	unit	file	translation	
[2015-05-06T16:25:20]	system	UA	am	4109	/am/terminology/gnome/am.po		↵
↪ MSDOS							
[2015-05-06T16:37:05]	system	UA	cs	12043	/cs/terminology/gnome/cs.po		↵
↪ přepínač							

Store:

date	user	action	path	store	
[2015-05-05T20:23:37]	system	SA	/templates/tutorial/tutorial.pot	1	

Command:

date	user	action	command
[2015-05-06T11:24:28]	system	X	pootle update_stores --project=vfolders
[2015-05-05T20:22:46]	system	X	pootle migrate

Quality check:

date	user	action	lang	unit	path	translation
[2015-05-19T14:16:36]	admin	QM	af	855	/af/terminology/gnome-	
↪ terminologie.po	lug					
[2015-05-19T14:17:44]	admin	QU	af	855	/af/terminology/gnome-	
↪ terminologie.po	lug					

Score:

date	user	SC	score_delta	score_action	#unit	NS=wordcount	S=similarity	↵
↪ total								
[2015-05-19T14:19:11]	admin	SC	1.0	TA	#1	NS=1	S=0.0	↵
↪ (total: 2.28571428571)								

Sync and Update messages

The `sync_stores` and `update_stores` commands will produce a number of logs to report any activity that results from those commands.

update_stores:

[\$date] [update] updated \$number units in \$store_path [revision: \$revision]
[2015-05-19T21:06:24] [update] updated 1 units in /an/libo_ui/dictionaries/pt_PT.po ↵
↪ [revision: 58]

sync_stores:

[\$date] [sync] File saved; updated \$number units in \$store_path [revision: ↵
↪ \$revision]
[2015-05-19T23:11:50] [sync] File saved; updated 1 units in /an/libo_ui/avmedia/ ↵
↪ source/viewer.po [revision: 0]

Optimization

This page lists extra optional software you can install to improve Pootle's performance. Some configuration tips are given too.

Optional Software

By installing optional software you can gain performance and extra features.

Database Backends

You should really switch to a real database backend in production environments. Adjust the `DATABASES` setting accordingly.

- *MySQL*
- *PostgreSQL*

Web Servers

You should really run Pootle behind a *real web server*, at least to serve static content. For generating the dynamic content, you can also use alternative WSGI servers that might better suit your environment.

Apache Apache web server.

Nginx Nginx web server.

gunicorn Python WSGI HTTP server.

Speed-ups and Extras

iso-codes Enables translated language and country names.

raven Enables logging server exceptions to a *Sentry server*. If installed and configured, Pootle will automatically use the raven client.

Tips

With a few extra steps, you can support more users and more data. Here are some tips for performance tuning on your Pootle installation.

- Ensure that Pootle runs under a proper *web server*.
- Be sure to use a proper database server like *MySQL* instead of the default SQLite. You can *migrate an existing installation* if you already have data you don't want to lose.
- Install the latest recommended version of all dependencies. Django and the Translate Toolkit might affect performance. Later versions of Pootle should also give better performance. You can *upgrade* to newer versions of Pootle easily.
- Ensure `DEBUG` mode is disabled.
- Increase the cache timeout for users who are not logged in.
- Increase your `PARSE_POOL_SIZE` if you have enough memory available.

- Enable `'django.contrib.sessions.backends.cached_db'`.
- Disable swap on the server. Things should be configured so that physical memory of the server is never exceeded. Swapping is bound to seriously degrade the user experience.
- Ensure gzip compression is enabled on your web server. For Apache, `mod_deflate` and for Nginx, `ngx_http_gzip_module`.
- Serve your `robots.txt` file statically. By default Pootle will serve this file as a static template, but that means it is still going through a small layer of Django. On larger sites you likely want to have your webserver serve this file.

Apache

For Apache, review your server settings so that you don't support too many or too few clients. Supporting too many clients increases memory usage, and can actually reduce performance.

No specific settings can be recommended, since this depends heavily on your users, your files, and your hardware. However the default value for the `MaxClient` directive (usually 256) is almost always too high. Experiment with values between 10 and 80.

MySQL

Using MySQL with `InnoDB backend` is well tested. MyISAM is no longer supported. You can *migrate your current database* if you already have data you don't want to lose.

Caching System

Pootle uses a caching system to improve performance. It is an essential part of your Pootle installation. It is based on *Django's caching system*, and is used for various things:

- To serve cached (possibly slightly outdated) versions of most pages to anonymous users to reduce their impact on server performance.
- To cache bits of the user interface, even for logged in users. Data will not be out of date but Pootle still tries to use the cache to reduce the impact on server performance.
- To store the result of expensive calculations like translation statistics.
- To keep track of last update timestamps to avoid unnecessary and expensive file operations (for example don't attempt to save translations before a download if there are no new translations).

Without a well functioning cache system, Pootle could be slow.

Named Caches

Pootle is configured with a these named caches:

- `'default'` – all non specified cache data and all Django cache data.
- `'redis'` – all RQ data related and revision counter.
- `'lru'` – all lru cache data.

In large installations you may want to setup separate caches to improve cache performance. You can then setup caching parameters for each cache separately.

Cache Backends

Django supports [multiple cache backends](#) (methods of storing cache data). However, Redis is the only cache backend supported by Pootle. We use some custom features of Redis so cannot support other backends. You can customise the Redis cache settings by overriding the value of `CACHES` in your configuration file.

User Authentication and Authorization

Pootle's backend for authenticating and authorizing users is provided by [django-allauth](#), and it comes with a heavily-customized client-side user interface.

Note that while Allauth supports local and social sign-in flows, not all of them have been equally-tested on Pootle, so your mileage might vary.

At the same time, Allauth also provides [tons of settings](#) which deployments can configure to their needs, although some of them clash directly with how our workflow has been designed. For instance, leaving `UNIQUE_EMAIL = True` becomes a hard requirement.

Setting Up a Social Provider

Each third party social authentication provider has its own requirements, although most of them implement similar protocols (OAuth, OAuth2, OpenID etc.).

Usually providers require consumers to register their apps on the provider website. On the Pootle side of things, your provider might need to be registered as a social app against your host. In order to do this you will need to insert a few records into your SQL database.

An example with GitHub follows.

GitHub

1. [Register your app](#) against your host.

Application name Some descriptive name

Homepage URL URL to your Pootle server, e.g. `http://foo.bar.tld`

Application description Some descriptive text

Authorization callback URL URL to the callback endpoint of your provider in the Pootle server, e.g. `http://foo.bar.tld/accounts/github/login/callback/`

2. Let Allauth know about your social provider.

```
UPDATE django_site SET DOMAIN = 'foo.bar.tld', name = 'Site name' WHERE id=1;

INSERT INTO socialaccount_socialapp (provider, name, secret, client_id, 'key')
VALUES ('github', 'GitHub', '---Client-Secret-from-above---',
       '---Client-ID-from-above---', '');

INSERT INTO socialaccount_socialapp_sites (socialapp_id, site_id)
VALUES (1,1);
```

Note the first line simply sets the domain name for the default site; you can omit it if it's already up-to-date.

Management commands

The management commands are administration commands provided by Django, Pootle or any external Django app being used with Pootle. You will usually run these commands by issuing `pootle <command> [options]`.

For example, to get information about all available management commands, you will run:

```
(env) $ pootle help
```

Managing Pootle projects

These commands will go through all existing projects performing maintenance tasks. The tasks are all available through the web interface but on a project by project or file by file basis.

--project, --language

The commands target can be limited in a more flexible way using the `--project --language` command line options. They can be repeated to indicate multiple languages or projects. If you use both options together it will only match the files that match both languages and projects selected.

For example, to `calculate_checks` for the tutorial project only, run:

```
(env) $ pootle calculate_checks --project=tutorial
```

To only calculate the Zulu and Basque language files within the tutorial project, run:

```
(env) $ pootle calculate_checks --project=tutorial --language=zu --language=eu
```

Running commands with --no-rq option

--no-rq

New in version 2.7.1.

Some of the commands work asynchronously and will schedule jobs to RQ workers, rather than running them in the command process. You can change this behaviour using the `--no-rq` command line option.

This can be useful for running pootle commands in bash scripts or automating installation/upgrade/migration. It can also be useful for debugging otherwise asynchronous jobs.

For example, to run `calculate_checks` in the command process and wait for the process to terminate:

```
(env) $ pootle calculate_checks --no-rq
```

It is *not* generally safe to run commands in this mode if you have RQ workers active at the same time, as there is a risk that they conflict with other jobs dispatched to the workers.

--atomic

New in version 2.8: Default: `tp`. Available choices: `tp`, `all`, `none`.

This option allows you to run CLI commands with atomic transactions.

The default is to commit changes on per-translation-project basis.

For example to run `update_stores` against all translation projects in a single transaction.

```
(env) $ pootle update_stores --atomic=all
```

--noinput

If there are RQ workers running, the command will ask for confirmation before proceeding. This can be overridden using the `--noinput` flag, in which case the command will run even if there are.

retry_failed_jobs**retry_failed_jobs**

New in version 2.7.

Requeue failed RQ jobs.

Background RQ jobs can fail for various reasons. To push them back into the queue you can run this command.

Examine the RQ worker logs for tracebacks before trying to requeue your jobs.

update_data**update_data**

New in version 2.8.

This command updates the stats data. The stats data update can be triggered for specific languages or projects.

--store

Use the `--store` option to narrow the stats data calculation to a specific store:

```
(env) $ pootle update_data --store=/fr/proj/mydir/mystore.po
```

Note this will also trigger the update of the stats data for items above the store, like for example directories above it, its language and its project.

calculate_checks**calculate_checks**

New in version 2.7.

This command will create a background job to go through all units and recalculate quality checks.

Note: Disabled projects are processed.

`calculate_checks` will flush existing caches and update the quality checks cache.

It's necessary to run this command after upgrading Pootle if new quality checks are added.

The time it takes to complete the whole process will vary depending on the number of units you have in the database. If a user hits a page that needs to display stats but they haven't been calculated yet, then a message will be displayed indicating that the stats being calculated.

--check

Use the `--check` option to force calculation of a specified check. To recalculate only the `date_format` quality checks, run:

```
(env) $ pootle calculate_checks --check=date_format
```

Multiple checks can be specified in one run as well:

```
(env) $ pootle calculate_checks --check=date_format --check=accelerators
```

flush_cache

flush_cache

New in version 2.8.0.

Flush cache.

Warning: You must first **stop the workers** if you flush *redis* cache.

--django-cache

Use the `--django-cache` to flush the default cache which keeps Django templates, project permissions etc.

--rqdata

Use the `--rqdata` to flush all data contained in *redis* cache: pending jobs, revision (which will be automatically restored), all data from queues.

--lru

Use the `--lru` to flush all lru cache data contained in *lru* cache.

--all

Use the `--all` to flush all caches (default, *redis*, *lru*) data.

refresh_scores

refresh_scores

New in version 2.7.

Recalculates the scores for all users. It is possible to narrow down the calculation to specific projects and/or languages.

Warning: It is advisable to run this command while Pootle server is offline since the command can fail due to data being changed by users.

--reset

When the `--reset` option is used , all score log data is removed and *zero* score is set for all users.

sync_stores

sync_stores

Deprecated since version 2.9: Deprecated in favor of *Pootle FS equivalent*.

Note: Since version 2.9 all projects are managed by Pootle FS and therefore this command is now able to work with those projects.

Save all translations currently in the database to the file system, thereby bringing the files under the `POOTLE_TRANSLATION_DIRECTORY` directory in sync with the Pootle database.

Note: Disabled projects are skipped.

You must run this command before taking backups or running scripts that modify the translation files directly on the file system, otherwise you might miss out on translations that are in the database but not yet saved to disk. In other words, **translations are saved to disk only when you explicitly do so** using this command.

For every file being synced, the in-DB `Store` will be updated to reflect the latest revision across the units in the file at the time of syncing. This allows Pootle to make optimizations when syncing and updating files, ignoring files that haven't change.

The default behavior of `sync_stores` can be altered by specifying these parameters:

--force

Changed in version 2.9: This option has no effect anymore.

--overwrite

Changed in version 2.9: This option has no effect anymore.

--skip-missing

Ignores files missing on disk, and no new files will be created.

update_stores

update_stores

Deprecated since version 2.9: Deprecated in favor of *Pootle FS equivalent*.

Note: Since version 2.9 all projects are managed by Pootle FS and therefore this command is now able to work with those projects.

Load translation files currently on the file system into the database, thereby bringing the Pootle database in sync with the files under the `POOTLE_TRANSLATION_DIRECTORY` directory. Pootle will not detect changes in the file system on its own. This is the opposite of `sync_stores`.

Note: Disabled projects are skipped.

Note: `update_stores` does not manage the updating of translations against templates, it simply loads translation files and translation templates into Pootle. For a full understanding of the role of templates and updating translations against templates read the *templates* section.

It also discovers new units, files and translation projects that were added on disk:

- Projects that exist in the DB but ceased to exist on disk will be **disabled** (not deleted). If a project is recovered on disk it can be enabled via the admin UI only.
- Translation projects will be scanned for new files and directories. In-DB files and directories that no longer exist on disk will be **marked as obsolete**. Also any in-DB directory will be **marked as obsolete** if this directory is empty or contains empty directories only.
- In-DB stores will be updated with the contents of the on-disk files. New units will be **added** to the store, units that ceased to exist will be **marked as obsolete**. Translations that were updated on-disk will be reflected in the DB.

You must run this command after running scripts that modify translation files directly on the file system.

`update_stores` accepts several options:

--force

Changed in version 2.9: This option has no effect anymore.

--overwrite

Mirrors the on-disk contents of the file. If there have been changes in the database **since the last sync operation**, these will be overwritten.

Warning: If files on the file system are corrupt, translations might be deleted from the database. Handle with care!

`list_serializers`

`list_serializers`

New in version 2.8.0.

List the installed serializers and deserializers on your system.

Available options:

-m, --model

List serializers for specified model. The model should be expressed as a contenttype label - eg `app_name.“model_name“`

-d, --deserializers

List available deserializers set up for our system.

`list_languages`

`list_languages`

Lists all the language codes for languages hosted on the server. This can be useful for automation.

--modified-since

Accepts the `--modified-since` parameter to list only those languages modified since the revision given by `revision`.

`list_projects`

list_projects

Lists all the project codes on the server. This might can be useful for automation.

--modified-since

Accepts the *--modified-since* parameter to list only those projects modified since the revision given by *revision*.

contributors

contributors

New in version 2.7.1.

Lists the contributors to a language, project or overall and the amount of contributions they have.

Available options:

--sort-by

Changed in version 2.8.0.

Specifies the sorting to be used. Valid options are *contributions* (sort by decreasing number of contributions) and *username* (sort by user name, alphabetically).

Default: *username*.

--mailmerge

New in version 2.8.0.

Specifies to only output user names and emails. Users with no email are skipped.

--mailmerge and *--include-anonymous* are mutually exclusive.

--include-anonymous

New in version 2.8.0.

Specifies to include anonymous contributions.

--include-anonymous and *--mailmerge* are mutually exclusive.

--since

New in version 2.8.0.

Only consider contributions since the specified date or datetime.

Date or datetime can be in any format accepted by *python-dateutil* library, for example ISO 8601 format (2016-01-24T23:15:22+0000 or 2016-01-24) or a string formatted like "2016-01-24 23:15:22+0000" (quotes included).

--until

New in version 2.8.0.

Only consider contributions until the specified date or datetime.

Date or datetime can be in any format accepted by *python-dateutil* library, for example ISO 8601 format (2016-01-24T23:15:22+0000 or 2016-01-24) or a string formatted like "2016-01-24 23:15:22+0000" (quotes included).

set_filetype

set_filetype

New in version 2.8.

This command sets file formats for projects, and also allows to convert files to another format.

--from-filetype

Convert stores of this file type.

--matching

Convert stores matching this path glob within the project.

For example, to add the *properties* format to a project, run:

```
(env) $ pootle set_filetype --project=myproj properties
```

To convert stores of *po* format to *properties*, run:

```
(env) $ pootle set_filetype --project=myproj --from-filetype=po properties
```

To convert stores matching a given path glob to *properties* format, run:

```
(env) $ pootle set_filetype --project=myproj --matching=mydir/myfile-* properties
```

revision

revision

New in version 2.7.

Print the latest revision number.

The revision is a common system-wide counter for units. It is incremented with every translation action made from the browser. Zero length units that have been auto-translated also increment the unit revision.

--restore

The revision counter is stored in the database but also in cache for faster retrieval. If for some reason the revision counter was removed or got corrupted, passing the *--restore* flag to the command will restore the counter's value based on the revision data available on the relational DB backend. You shouldn't need to ever run this, but if for instance you deleted your cache you will need to restore the counter to ensure correct operation.

test_checks

test_checks

New in version 2.7.

Tests any given string pair or unit against all or certain checks from the command line. This is useful for debugging and developing new checks.

--source, --target

String pairs can be specified by setting the values to be checked in the *--source=<"source_text">* and *--target=<"<target_text">* command-line arguments.

--unit

Alternatively, `--unit=<unit_id>` can be used to reference an existing unit from the database.

--check

By default, `test_checks` tests all existing checks. When `--check=<checkname>` is set, only specific checks will be tested against.

dump

dump

New in version 2.7.

Prints data or stats data (depending on `--data` or `--stats` option) in specific format.

--data

```
object_id:class_name
8276:Directory      name=android      parent=/uk/      pootle_path=/uk/android/
24394:Store    file=android/uk/strings.xml.po  translation_project=/uk/android/
↳pootle_path=/uk/android/strings.xml.po  name=strings.xml.pstate=2
806705:Unit    source=Create Account  target=      source_wordcount=2      target_
↳wordcount=2      developer_comment=create_account      translator_
↳commentlocations=File:\nstrings.xml\nID:\ne82a8ea14a0b9f92b1b67ebfde2c16e9
↳isobsolete=False      isfuzzy=False      istranslated=True
115654:Suggestion      target_f=      user_id=104481
```

--stats

```
pootle_path total,translated,fuzzy,suggestions,criticals,is_dirty,last_action_unit_id,
↳last_updated_unit_id
/uk/android/strings.xml.po  11126,10597,383,231,0,False,4710214,4735242
/uk/android/widget/strings.xml.po  339,339,0,26,0,False,2277376,3738609
/uk/android/widget/  339,339,0,26,0,False,2277376,3738609
/uk/android/  11465,10936,383,257,0,False,4710214,4735242
```

This command can be used by developers to check if all data kept after migrations or stats calculating algorithm was changed.

config

config

New in version 2.8.

Gets, sets, lists, appends and clears pootle configuration settings.

content_type

Optional positional argument to specify a model to manage configuration for.

object

Optional positional argument to specify the primary key of an object to manage configuration for. You can use a field other than the primary key by specifying `-o`, but the field must be unique for the request object when doing so.

`-o <field>`, `--object-field <field>`

Specify a field other than the primary key when specifying an object. It must be unique to the object specified.

- g** <key>, **--get** <key>
Get value for specified key.
- l** <key>, **--list** <key>
List values for specified key(s). This option can be specified multiple times.
- s** <key> <value>, **--set** <key> <value>
Set value for specified key. The key must be unique or not exist already.
- a** <key> <value>, **--append** <key> <value>
Append value for specified key.
- c** <key>, **--clear** <key>
Clear value(s) for specified key.
- j**, **--json**
Treat data as JSON when getting, setting, or appending values.

schema

schema

New in version 2.8.

Dumps a JSON representation for the Pootle database schema, currently only MySQL, for debugging and comparison to a reference database schema.

Translation Memory

These commands allow you to setup and manage *Translation Memory*.

update_tmserver

update_tmserver

New in version 2.7.

Changed in version 2.7.3: Renamed `--overwrite` to `--refresh`. Disabled projects' translations are no longer added by default. It is also possible to import translations from files.

Updates the `local` server in `POOTLE_TM_SERVER`. The command reads translations from the current Pootle install and builds the TM resources in the TM server.

If no options are provided, the command will only add new translations to the server.

--refresh

Use `--refresh` to also update existing translations that have been changed, besides adding any new translation.

--rebuild

To completely remove the TM and rebuild it adding all existing translations use `--rebuild`.

--tm

If no specific TM server is specified using `--tm`, then the default `local` TM will be used. If the specified TM server doesn't exist it will be automatically created for you.

--include-disabled-projects

By default translations from disabled projects are not added to the TM, but this can be changed by specifying `--include-disabled-projects`.

--dry-run

To see how many units will be loaded into the server use `--dry-run`, no actual data will be loaded or deleted (the TM will be left unchanged):

```
(env) $ pootle update_tmserver --dry-run
(env) $ pootle update_tmserver --refresh --dry-run
(env) $ pootle update_tmserver --rebuild --dry-run
```

This command also allows to read translations from files and build the TM resources in the external TM server. In order to do so it is mandatory to provide the `--tm` and `--display-name` options, along with some files to import.

--display-name

The display name is a label used to group translations within a TM. A given TM can host translations for several display names. The display name can be used to specify the name of the project from which the translations originate. The display name will be shown on TM matches in the translation editor. To specify a name use `--display-name`:

```
(env) $ pootle update_tmserver --tm=libreoffice --display-name="LibreOffice 4.3 UI"
↳TM_LibreOffice_4.3.gl.tmx
```

By default the command will only add new translations to the server. To rebuild the server from scratch use `--rebuild` to completely remove the TM and rebuild it before importing the translations:

```
(env) $ pootle update_tmserver --rebuild --tm=mozilla --display-name="Foo 1.7" foo.po
```

Option `--refresh` doesn't apply when adding translations from files on disk.

To see how many units will be loaded into the server use `--dry-run`, no actual data will be loaded:

```
(env) $ pootle update_tmserver --dry-run --tm=mozilla --display-name="Foo 1.7" foo.po
175045 translations to index
```

This command is capable of importing translations in multiple formats from several files and directories at once:

```
(env) $ pootle update_tmserver --tm=mozilla --display-name="Foo 1.7" bar.tmx foo.
↳xliff fr/
```

--target-language

Use `--target-language` to specify the target language ISO code for the imported translations in case it is not possible to guess it from the translation files or if the code is incorrect:

```
(env) $ pootle update_tmserver --target-language=af --tm=mozilla --display-name="Foo
↳1.7" foo.po bar.tmx
```

Virtual Folders

These commands allow you to perform tasks with virtual folders from the command line.

add_vfolders

add_vfolders

New in version 2.7.

Creates *virtual folders* from a JSON file. If the specified virtual folders already exist then they are updated.

The *vfolder format* defines how to specify a virtual folder that fits your needs.

This command requires a mandatory filename argument.

```
(env) $ pootle add_vfolders virtual_folders.json
```

Import and Export

Export and Import translation files in Pootle. The operation can be thought of best as offline operations to assist with offline translation, unlike *sync_stores* and *update_stores* the operations here are designed to cater for translators working outside of Pootle.

The *import* and *export* commands are designed to mimic the operations of Download and Upload from the Pootle UI.

export

export

New in version 2.7.

Download a file for offline translation.

Note: This mimics the editor's download functionality and its primary purpose is to test the operation of downloads from the command line.

A file or a .zip of files is provided as output. The file headers include a revision counter to assist Pootle to determine how to handle subsequent uploads of the file.

Available options:

--tmx

New in version 2.8.0.

Export every translation project as one zipped TMX file into `MEDIA_ROOT` directory.

--rotate

New in version 2.8.0.

Remove old exported zipped TMX files (except previous one) from `MEDIA_ROOT` directory after current exported file is saved.

import

import

New in version 2.7.

Upload a file that was altered offline.

Note: This mimics the editor's upload functionality and its primary purpose is to test the operation of uploads from the command line.

A file or a .zip is submitted to Pootle and based on the revision counter of the `Store` on Pootle it will be uploaded or rejected. If the revision counter is older than on Pootle, that is someone has translated while the file was offline, then it will be rejected. Otherwise the translations in the file are accepted.

Available options:

--user

New in version 2.7.3.

Import file(s) as given user. The user with the provided username must exist.

Default: `system`.

Manually Installing Pootle

These commands expose the database installation and upgrade process from the command line.

`init`

`init`

Create the initial configuration for Pootle.

Available options:

--config

The configuration file to write to.

Default: `~/pootle/pootle.conf`.

--db

New in version 2.7.1.

The database backend that you are using

Default: `sqlite`. Available options: `sqlite`, `mysql`, `postgresql`.

--db-name

New in version 2.7.1.

The database name or path to database file if you are using `sqlite`.

Default for `sqlite`: `db/pootle.db`. Default for `mysql/postgresql`: `pootledb`.

--db-user

New in version 2.7.1.

Name of the database user. Not used with `sqlite`.

Default: `pootle`.

--db-host

New in version 2.7.1.

Database host to connect to. Not used with `sqlite`.

Default: `localhost`.

--db-port

New in version 2.7.1.

Port to connect to database on. Defaults to database backend's default port. Not used with sqlite.

--dev

New in version 2.8.

Creates a development configuration instead.

--yes

New in version 2.9.

Answer 'yes' to any questions blocking overwrite of existing config files.

initdb

initdb

Initializes a new Pootle install.

This is an optional part of Pootle's install process, it creates the default *admin* user, populates the language table with several languages, initializes the terminology project, and creates the tutorial project among other tasks.

initdb can only be run after *migrate*.

initdb accepts the following option:

New in version 2.7.3.

--no-projects

Don't create the default terminology and tutorial projects.

Note: *initdb* will import translations into the database, so can be slow to run. You should have an *rqworker* running or run with the *-no-rq*.

collectstatic

Running the Django admin *collectstatic* command finds and extracts static content such as images, CSS and JavaScript files used by the Pootle server, so that they can be served separately from a static webserver. Typically, this is run with the *--clear --noinput* options, to flush any existing static data and use default answers for the content finders.

assets

Pootle uses the Django app *django-assets* interface of *webassets* to minify and bundle CSS and JavaScript; this app has a management command that is used to make these preparations using the command *assets build*. This command is usually executed after the *collectstatic* one.

webpack

webpack

New in version 2.7.

The **webpack** tool is used under the hood to bundle JavaScript scripts, and this management command is a convenient wrapper that sets everything up ready for production and makes sure to include any 3rd party customizations.

--dev

When the `--dev` flag is enabled, development builds will be created and the process will start a watchdog to track any client-side scripts for changes. Use this only when developing Pootle.

Pootle FS

fs

fs

To interact with Pootle FS we use multiple subcommands:

- Admin:
 - `info` - Display filesystem info
 - `state` - Show current state
- Action:
 - `fetch` - Add a file from the filesystem to Pootle
 - `add` - Add a store from Pootle to the filesystem
 - `rm` - Remove a store and file from both Pootle and the filesystem
 - `resolve` - Handle conflicts in stores and files
 - `unstage` - Revert a staged action
- Execute:
 - `sync` - Execute staged actions

Note: The **action** staging commands require that you run `sync` in order to actually perform the staged actions.

Common options

Pootle FS **action** and **execution** subcommands take the `-p` and `-P` options which allow you to specify a glob to limit which files or stores are affected by the command.

-p --fs-path

Only affect files whose filesystem path matches a given glob.

```
(env) $ pootle fs add --fs-path=MYPROJECT/af/directory/file.po MYPROJECT
(env) $ pootle fs add --fs-path=MYPROJECT/af/* MYPROJECT
(env) $ pootle fs add --fs-path=MYPROJECT/af/*/file.po MYPROJECT
(env) $ pootle fs add --fs-path=MYPROJECT/af/directory/*.po MYPROJECT
```

Note: The path should be relative to the Pootle FS URL setting for the project.

-P `--pootle-path`
Only affect files whose Pootle path matches a given glob.

```
(env) $ pootle fs add --pootle-path=/af/MYPROJECT/directory/file.po MYPROJECT
(env) $ pootle fs add --pootle-path=/af/MYPROJECT/* MYPROJECT
(env) $ pootle fs add --pootle-path=/af/MYPROJECT/*/file.po MYPROJECT
(env) $ pootle fs add --pootle-path=/af/MYPROJECT/directory/*.po MYPROJECT
```

Note: Keep in mind that Pootle paths always start with `/`.

Pootle FS subcommands

add

fs add

New in version 2.8.0.

Stage for adding any new or changed stores from Pootle to the filesystem:

```
(env) $ pootle fs add MYPROJECT
```

This command is the functional opposite of the *fetch* command.

--force

Conflicting files on the filesystem will be staged to be overwritten by the Pootle store.

```
(env) $ pootle fs add --force MYPROJECT
```

fetch

fs fetch

New in version 2.8.0.

Stage for fetching any new or changed files from the filesystem to Pootle:

```
(env) $ pootle fs fetch MYPROJECT
```

This command is the functional opposite of the *add* command.

--force

Conflicting stores in Pootle to be overwritten with the filesystem file.

```
(env) $ pootle fs fetch --force MYPROJECT
```

info

fs info

New in version 2.8.0.

Retrieve the filesystem info for a project.

```
(env) $ pootle fs info MYPROJECT
```

resolve

fs resolve

New in version 2.8.0.

Stage for merging any stores/files that have either been updated both in Pootle and filesystem.

When merging, if there are conflicts in any specific translation unit the default behavior is to keep the filesystem version and convert the Pootle version into a suggestion. Suggestions can then be reviewed by translators to ensure any corrections are correctly incorporated.

When there are no conflicts in unit *resolve* will handle the merge without user input:

```
(env) $ pootle fs resolve MYPROJECT
```

--pootle-wins

Alter the default conflict resolution of filesystem winning to instead use the Pootle version as the correct translation and converting the filesystem version into a suggestion.

```
(env) $ pootle fs resolve --pootle-wins MYPROJECT
```

--overwrite

Discard all translations. Use only those translations from the filesystem, by default, or from Pootle if used together with *--pootle-wins*

```
(env) $ pootle fs resolve --overwrite MYPROJECT
```

rm

fs rm

New in version 2.8.0.

Remove any matched:

- Store that do not have a corresponding file in filesystem.
- File that do not have a corresponding store in Pootle.

```
(env) $ pootle fs rm MYPROJECT
```

--force

Stage for removal conflicting/untracked files and/or stores.

```
(env) $ pootle fs rm --force MYPROJECT
```

state

fs state

New in version 2.8.0.

List the status of stores in Pootle and files on the filesystem.

```
(env) $ pootle fs state MYPROJECT
```

-t --type

Restrict to specified *Pootle FS status*.

```
(env) $ pootle fs state -t pootle_staged MYPROJECT
```

sync

fs sync

New in version 2.8.0.

Commit any staged changes, effectively synchronizing the filesystem and Pootle. This command is run after other Pootle FS commands have been used to stage changes.

```
(env) $ pootle fs sync MYPROJECT
```

unstage

fs unstage

New in version 2.8.0.

Unstage any staged Pootle FS actions. This allows you to remove any staged actions which you might have added erroneously.

```
(env) $ pootle fs unstage MYPROJECT
```

Managing users

find_duplicate_emails

find_duplicate_emails

New in version 2.7.1.

As of Pootle version 2.8, it will no longer be possible to have users with duplicate emails. This command will find any user accounts that have duplicate emails. It also shows the last login time for each affected user and indicates if they are superusers of the site.

```
(env) $ pootle find_duplicate_emails
```

merge_user

merge_user

New in version 2.7.1.

This can be used if you have a user with two accounts and need to merge one account into another. This will re-assign all submissions, units and suggestions, but not any of the user's profile data.

This command requires 2 mandatory arguments, `src_username` and `target_username`, both should be valid usernames for users of your site. Submissions from the first are re-assigned to the second. The users' profile data is not merged.

--no-delete

By default `src_username` will be deleted after the contributions have been merged. You can prevent this by using the `--no-delete` option.

```
(env) $ pootle merge_user src_username target_username
```

purge_user

purge_user

New in version 2.7.1.

This command can be used if you wish to permanently remove a user and revert the edits, comments and reviews that the user has made. This is useful for removing a spam account or other malicious user.

This command requires a mandatory `username` argument, which should be a valid username for a user of your site.

Changed in version 2.7.3: `purge_user` can accept multiple user accounts to purge.

```
(env) $ pootle purge_user username [username ...]
```

update_user_email

update_user_email

New in version 2.7.1.

```
(env) $ pootle update_user_email username email
```

This command can be used if you wish to update a user's email address. This might be useful if you have users with duplicate email addresses.

This command requires a mandatory `username`, which should be a valid username for a user of your site, and a mandatory valid `email` address.

```
(env) $ pootle update_user_email username email
```

verify_user

verify_user

New in version 2.7.1.

Verify a user without the user having to go through email verification process.

This is useful if you are migrating users that have already been verified, or if you want to create a superuser that can log in immediately.

This command requires either mandatory `username` arguments, which should be valid username(s) for user(s) on your site, or the `--all` flag if you wish to verify all users of your site.

Changed in version 2.7.3: `verify_user` can accept multiple user accounts to verify.

```
(env) $ pootle verify_user username [username ...]
```

Available options:

--all

Verify all users of the site

Running WSGI servers

There are multiple ways to run Pootle, and some of them rely on running WSGI servers that can be reverse proxied to a proper HTTP web server such as nginx or lighttpd.

There are many more options such as [uWSGI](#), [Gunicorn](#), etc.

Deprecated commands

The following are commands that have been removed or deprecated:

refresh_stats

refresh_stats

Removed in version 2.8.

With the new stats infrastructure this is not needed anymore.

clear_stats

clear_stats

Removed in version 2.8.

With the new stats infrastructure this is not needed anymore.

last_change_id

last_change_id

Deprecated since version 2.7.

With the change to revisions the command you will want to use is `revision`, though you are unlikely to know a specific revision number as you needed to in older versions of `update_stores`.

commit_to_vcs

commit_to_vcs

Deprecated since version 2.7.

Version Control support has been removed from Pootle and will reappear in a later release.

update_from_vcs

update_from_vcs

Deprecated since version 2.7.

Version Control support has been removed from Pootle and will reappear in a later release.

run_cherryipy

run_cherryipy

Deprecated since version 2.7.3.

Run the CherryPy server bundled with the Translate Toolkit.

start

start

Removed in version 2.7.3.

Use `runserver` instead.

Run Pootle using the default Django server.

Running Commands in cron

If you want to schedule certain actions on your Pootle server, using management commands with cron might be a solution.

The management commands can perform certain batch commands which you might want to have executed periodically without user intervention.

For the full details on how to configure cron, read your platform documentation (for example `man crontab`). Here is an example that runs the `calculate_checks` command daily at 02:00 AM:

```
00 02 * * * www-data source /var/www/sites/pootle/env/bin/activate; pootle calculate_
→checks
```

Test your command with the parameters you want from the command line. Insert it in the cron table, and ensure that it is executed as the correct user (the same as your web server) like `www-data`, for example. The user executing the command is specified in the sixth column. Cron might report errors through local mail, but it might also be useful to look at the logs in `/var/log/cron/`, for example.

If you are running Pootle from a virtualenv, or if you set any custom `PYTHONPATH` or similar, you might need to run your management command from a bash script that creates the correct environment for your command to run from. Call this script then from cron. It shouldn't be necessary to specify the settings file for Pootle — it should automatically be detected.

RQ Job Queues

Pootle makes use of RQ to manage background jobs.

Some tasks are performed using background jobs and we expect more components to use it in future.

The RQ queue is managed by Redis and it is setup in the `RQ_QUEUES` and `CACHES` settings.

Running job workers

The queue is processed by Workers. Any number of workers may be started and will process jobs in the default queue. The `rqworker` command is used to start a Worker.

Monitoring the queue

At the simplest level the Admin dashboard will tell you if the queue is active and how many workers are available to service the queue. It also lists the number of pending jobs and the number of failed jobs. This gives you a quick way to see if anything is wrong.

Working with failed jobs

If a job fails it needs to be investigated. In most cases a traceback will indicate why the job failed.

The simplest way to work with queues and jobs is to use `rq-dashboard`, though you likely don't want to deploy that on a production server. With this you can see the jobs in the queue, you can check the tracebacks and you can retry failed jobs.

In the case of a production server you can make use of the following commands to manage jobs:

```
$ redis-cli -n 2 lrange rq:queue:default 0 -1
03135097-00f8-46eb-b084-6f34a16d9940
a07309b3-f056-47e7-856c-c608bda2f171
3df6a559-2e3c-4c0c-b09c-1948b4bacda2
```

This will display all pending job IDs in the default queue. We're using the Redis DB number 2, the default RQ queue on a standard Pootle install.

```
$ redis-cli -n 2 lrange rq:queue:failed 0 -1
60ed13df-0ce5-4b98-96f0-f8e0294ba421
3240527f-58b9-40fe-b0c5-b8d3fcaa06b6
```

This will display the failed job IDs.

To investigate a failed job simply add `rq:job:` prefix to a job ID and use a command such as this:

```
$ redis-cli -n 2 hgetall rq:job:60ed13df-0ce5-4b98-96f0-f8e0294ba421
```

This will allow you to see any traceback and investigate and solve them.

To push failed jobs back into the queue we simply run the `retry_failed_jobs` management command.

Delete all failed jobs

Sometimes failed jobs no longer apply since they refer to removed items, so no matter how many times you run them they will keep failing. Note that sometimes those unrecoverable failed jobs are in company of other failed jobs that can be re-run by using the [retry_failed_jobs](#) management command:

```
(env) $ pootle retry_failed_jobs
```

In order to delete all the failed jobs you must first **stop the workers**.

Once the workers are stopped make sure that there are no failed jobs that you don't want to remove. In case there is any restart the workers to re-run them with [retry_failed_jobs](#). Stop the workers again once those jobs are completed. Check again that all the failed jobs are the ones you want to remove.

In order to perform a bulk delete of all failed jobs run the following commands:

```
$ redis-cli -n 2 LRANGE "rq:queue:failed" 0 -1 | perl -nE 'chomp; `redis-cli DEL_`
↪rq:job:$`;'
```

Now remove the list of failed jobs:

```
$ redis-cli -n 2 DEL "rq:queue:failed"
```

Do not forget to **restart the workers**.

Backup your Pootle system

In particular you should backup:

- All your translation files (your whole [POOTLE_TRANSLATION_DIRECTORY](#)). Use the [sync_stores](#) command to synchronize all your translation files to disk before making any backup.
- Your settings, to avoid losing any settings customizations.
- Your complete database using the appropriate *dump* command for your database system. For example **mysqldump** for MySQL, or **pg_dump** for PostgreSQL.
- Any code, templates or styling customization that you have done to your installation.

1.6 Developers

If you are a developer and are willing to hack on Pootle or contribute in some other way, make sure to read through this part.

1.6.1 Contributing

There are several ways you can contribute to improve Pootle, even if you don't know programming! Want to know how? Please keep reading.

- You can give us feedback about things that annoy you or about areas you see for improvement. You can reach us in [our mailing list](#) or in the [Pootle channel](#).
- Found a bug? Report it in our [Issue tracker](#). You can also always contact us on Pootle channel. Make sure to read more about [how to report bugs](#).

- Translate the User Interface into your own language. Pootle is translated into [nearly 50 languages](#). Is your language missing? Have you found any errors in the translation? Learn [how to contribute translating](#).
- Suggest [documentation improvements](#) by fixing mistakes and adding new sections.
- In case you have coding skills and are willing to contribute patches, fixes, or new features, read how you can [hack on Pootle](#).

Requesting features

Sometimes Pootle doesn't quite meet your expectations or you have an idea for a great new feature.

It might help to understand how Pootle developers evaluate new features:

1. Is it generally useful? *i.e.* will it be useful for a large number of people?
2. Does it follow the ethos of Pootle? *e.g.* does it keep the interface clean, is it intuitive and non-technical?
3. How long would it take to implement?
 - (a) Does it require fundamental changes to how Pootle works? *i.e.* long, or
 - (b) Is this just a simple change of layout or a simple feature? *i.e.* short
4. Is this something a developer is passionate about? Does this meet their itch or are they convinced it is a winning feature?

How can I make a winning feature request?

If you really do want your feature to succeed here are some options to help you when reporting or requesting the feature.

1. Have you thought about this and provided a clear use case?
 - Using a real use case would be good.
 - Make it clear why you think this feature is important, don't assume it is obvious.
2. Have you made some mockups of the UI?
 - Isn't it a bit unfair that you expect a volunteer coder to create the mockup for your feature?
3. Did you have some discussion on the mailing list or on the Pootle channel?
 - Drive-by feature requests usually don't get attention. But if you have built a case and some links to developers, *i.e.* they know you, then they will listen. Proposing your idea in these forums could be helpful for your case.
4. Can you code?
 - If you can code the feature yourself that will always win some acceptance. But realise that someone does need to review your code and your code still needs to meet the acceptance criterion. So discuss early.
 - If you can't code, commission someone to write it for you. Or spend a lot more time making sure that you use the volunteers' free time to your best advantage, *i.e.* you need to work hard to make the feature clear and easy to implement.

Reporting bugs

In order to best solve the problem we need good bug reports. Reports that do not give a full picture or which coders are unable to reproduce, end up wasting a lot of time. If you, the expert in your bug, spend a bit of time you can make sure your bug gets fixed.

First **see if the bug is not already reported**. Perhaps someone already reported it and you can provide some extra information in that bug report. You can also add yourself in the CC field so that you get notified of any changes to the bug report.

If you could not find the bug, you should report it. Look through each of the following sections and make sure you have given the information required.

Be verbose

Tell us exactly how came to see this bug. Don't say:

```
Suggesting doesn't work
```

Rather say:

```
In a translation project with proper permissions when I try to suggest I  
get a 404 error.
```

So we need to know:

1. What procedure you followed
2. What you got, and
3. What you expected to get

Steps to reproduce

Tell us exactly how to reproduce the error. Mention the steps if needed, or give an example. Without being able to reproduce the error, it will not easily get fixed.

Include tracebacks

If you are a server administrator you can get this information from the web server's error log. In case you're hacking on Pootle, the traceback will be displayed both in the console and the browser.

A traceback will give a much better clue as to what the error might be and send the coder on the right path. It may be a very simple fix, may relate to your setup or might indicate a much more complex problem. Tracebacks help coders get you information quicker.

Be available

If you can be on [Pootle channel](#) or the [mailing list](#) to answer questions and test possible fixes then this will help to get your problem fixed quickly.

Translating

Pootle's User Interface translations are kept in the [official Pootle server](#). If you have a user in that server, you can start translating right away. Otherwise, just create a new user and start translating.

If your language already has a translation and you want to further improve or complete it, you can contribute suggestions that will later be reviewed by the language administrators.

If you can't find your language and want to have that added or have concerns of any other means, contact us on the [Pootle channel](#).

Although desirable, it's not mandatory to use the official Pootle server to translate Pootle itself. In case you feel more comfortable working with files and offline tools, just head to the [code repository at GitHub](#), create your localization based on the latest template and submit it to us by [opening a bug](#) or by sending us a pull request.

There are some [additional localization requirements](#) beyond translation, so please review those to ensure that your language is 100% translated.

Documentation

You can help us documenting Pootle by just mentioning typos, providing reworded alternatives or by writing full sections.

Pootle's documentation is written using [reStructuredText](#) and [Sphinx](#).

If you intend to build the documentation yourself (it's converted from reST to HTML using Sphinx), you may want to *setup a development environment* for that.

1.6.2 Pootle Development Roadmap

We track the development roadmap using [Github Milestones](#). These match the version numbers of future releases.

Project specific milestones are tracked using `mstone-$name` milestones.

Note: Development partners may use their own systems to track their roadmaps.

1.6.3 Hacking

Want to fix a bug in Pootle? Want to change the behaviour of an existing feature or add new ones? This section is all about hacking on Pootle, so if you are interested on the topic, keep reading.

Before doing anything

Before starting any actual work on the source code, make sure that:

- There is nobody working on the bug you are trying to fix. See the [existing bug reports](#) and the [existing pull requests](#). In the situation where somebody else is working on a fix, you can always offer your help.
- If you plan to develop a new feature and want to include it upstream, please first discuss it with the developers on the [Pootle development channel](#) or in the [translate-pootle mailing list](#) so that it doesn't interfere in current development plans. Also note that adding new features is relatively easy, but keeping them updated is harder.

Environment setup

Although Pootle should only be deployed to production on a Linux server, it is possible to get a viable development environment up and running on Windows with some slightly different steps.

- [Environment setup on Linux](#)
- [Environment setup on Windows](#)

Workflow

Any time you want to fix a bug or work on a new feature, create a new local branch:

```
$ git checkout -b <my_new_branch>
```

Then safely work there, create the needed commits and once the work is ready for being incorporated upstream, either:

- Push the changes to your own GitHub fork and send us a pull request, or
- Create a patch against the HEAD of the master branch using `git diff` or `git format-patch` and attach it to the affected issue.

Commits

When creating commits take into account the following:

What to commit As far as possible, try to commit individual changes in individual commits. Where different changes depend on each other, but are related to different parts of a problem / solution, try to commit them in quick succession.

If a change in the code requires some change in the documentation then all those changes must be in the same commit.

If code and documentation changes are unrelated then it is recommended to put them in separate commits, despite that sometimes it is acceptable to mix those changes in the same commit, for example cleanups changes both in code and documentation.

Commit messages Begin the commit message with a single short (less than 50 character) line summarizing the change, followed by a blank line and then a more thorough (and sometimes optional) description.

```
Cleanups
```

Another example:

```
Factor out common behavior for whatever

These reduces lines of code to maintain, and eases a lot the maintenance
work.

Also was partially reworked to ease extending it in the future.
```

If your change fixes a bug in the tracker, mention the bug number. This way the bug is automatically closed after merging the commit.

```
Docs: Update code for this thing

Now the docs are exact and represent the actual behavior introduced in
```

(continues on next page)

(continued from previous page)

```
commits ef4517ab and abc361fd.
```

```
Fixes #2399
```

If you are reverting a previous commit, mention the sha1 revision that is being reverted.

```
Revert "Fabric: Cleanup to use the new setup command"
```

```
This reverts commit 5c54bd4.
```

1.6.4 Linux Development Environment Setup

The minimum software packages you need for setting up a development environment include [git](#) and a [Python interpreter](#) along with the [pip installer](#). Consult the specifics for your operating system in order to get each package installed successfully.

Once you have the basic requirements in place, you will need to install Pootle's dependencies, which come in shape of Python packages. Instead of installing them system-wide, we recommend using [virtualenv](#) (and [virtualenvwrapper](#) for easing the management of multiple virtualenvs). This way you can install all the dependencies at specific versions without interfering with system-wide packages. You can test on different Python/Django versions in parallel as well.

Detailed setup

For installing the dependencies in an isolated environment, we will use [virtualenv](#) – more specifically [virtualenvwrapper](#), which eases the process of managing and switching between multiple virtual environments. Install [virtualenvwrapper](#) as follows for bash (examine [platform specific installation instructions](#) for other environments):

```
$ sudo pip install virtualenvwrapper
```

[virtualenvwrapper](#) will need to be configured in order to specify where to store the created environments:

```
$ export WORKON_HOME=~/.envs
$ mkdir -p $WORKON_HOME
$ source /usr/local/bin/virtualenvwrapper.sh # Or /usr/bin/virtualenvwrapper.sh
```

Note: You may want to add the above-mentioned commands and environment variables to your `.bashrc` file (or whatever file your shell uses for initializing user customizations).

Now that the commands provided by [virtualenv](#) and [virtualenvwrapper](#) are available, we can create our virtual environment.

```
$ mkvirtualenv pootle_env
```

Replace `pootle_env` with a meaningful name that describes the environment you are creating. [mkvirtualenv](#) accepts any options that [virtualenv](#) accepts. We could for example specify to use the Python 3.3 interpreter by passing the `-p python3.3` option.

Note: After running [mkvirtualenv](#), the newly created environment is activated. To deactivate it just run:

```
(pootle_env) $ deactivate
```

To activate a virtual environment again use `workon` as follows:

```
$ workon pootle_env
```

First, upgrade the version of `pip` and `setuptools` as follows:

```
(pootle_env) $ pip install --upgrade pip setuptools
```

Time to clone Pootle's source code repository. The main repository lives under [translate/pootle](https://github.com/translate/pootle) in GitHub.

Note: If you have a GitHub account, fork the main `translate/pootle` repository and replace the repository URL with your own fork.

```
(pootle_env) $ git clone https://github.com/translate/pootle.git
```

Install Pootle and its development dependencies into your virtualenv. This makes it easy to run Pootle locally and is needed for various development activities. The `[dev]` target will install some extra packages to aid development (you can examine these in `requirements/dev.txt`).

```
(pootle_env) $ pip install -e .[dev]
```

Note: Some requirements may depend on external packages. For these you may need to install extra packages on your system in order to complete their installation.

With all the dependencies installed within the virtual environment, Pootle is almost ready to run. In development environments you will want to use settings that differ from those used in production environments.

```
(pootle_env) $ pootle init --dev
```

Note: To learn more about how settings work in Pootle read the [settings](#) documentation.

Once the configuration is in place, you'll need to setup the database schema and add initial data.

```
(pootle_env) $ pootle migrate
(pootle_env) $ pootle initdb
```

Now ensure that you have built the assets by following the instructions for [frontend development](#).

Finally, run the development server.

```
(pootle_env) $ pootle runserver
```

Once all is done, you can start the development server anytime by enabling the virtual environment (using the `workon` command) and running the `pootle runserver` command.

Happy hacking!!

1.6.5 Windows Development Environment Setup

Note: Ensure that you are executing all of the following steps with Administrator privileges!

Install prerequisites

Download the latest Redis installer from: <https://github.com/MSOpenTech/redis/releases>

During the installation you will be asked to set what port Redis should listen on; leave it at the default (6379).

Download the latest Nodejs installer from: <https://nodejs.org/en/>

Detailed setup

Note: For convenience these instructions consistently specify paths `C:\pootle_venv`, `C:\git\pootle` and `C:\temp`, but you can change these to suit your environment and needs.

Note: Depending on how correctly your environment is set up (depending on factors beyond your control such as virus scanners, Windows system health, and so on), you may need to use the command `python -m pip` for the following steps if the basic `pip` commands fail. Similarly, any other Python command that should ‘just work’ might need to be invoked with `python -m` to avoid issues.

For installing the dependencies in an isolated environment, set up a fresh virtualenv.

```
> pip install virtualenv
> virtualenv C:\pootle_venv
```

Activate the new virtualenv and upgrade pip:

```
> C:\pootle_venv\Scripts\activate
(pootle_venv)> pip install --upgrade pip setuptools
```

Clone your fork of the Pootle master using your favourite Windows implementation of Git so that you have a working copy somewhere accessible on your computer (e.g. `C:\git\pootle`).

Then go to the Pootle `requirements\base.txt` and comment out the following packages:

```
# lxml
# python-levenshtein
# scandir
```

These three packages are difficult to build on Windows, so we will download pre-built versions to install manually, saving them into your temporary folder:

- <http://www.lfd.uci.edu/~gohlke/pythonlibs/#lxml>
- <http://www.lfd.uci.edu/~gohlke/pythonlibs/#python-levenshtein>
- <http://www.lfd.uci.edu/~gohlke/pythonlibs/#scandir>

Now install them explicitly:

```
(pootle_venv)> pip install C:\temp\lxml-3.6.4-cp27-cp27m-win32.whl
(pootle_venv)> pip install C:\temp\python-Levenshtein-0.12.0-cp27-none-win32.whl
(pootle_venv)> pip install C:\temp\scandir-1.2-cp27-none-win32.whl
```

At this point, you may be able to install Pootle and its requirements using the following command. However, pip installation of requirements may fail with “directory was not empty” or “file not found” issues, in which case you need to use the commands in the next note block.

```
(pootle_venv)> cd C:\git\pootle
(pootle_venv)> pip install -e .[dev]
```

Note: “Directory was not empty” and “file not found” issues come from modern versions of Windows’ tighter control over permissions for special folders. By default, pip stores temporary files in your user\AppData folder which may not allow access in all circumstances. By downloading the packages to a folder with no special permissions and building and installing them from there we can circumvent these problems:

```
(pootle_venv)> pip download -d C:\temp -r requirements\dev.txt -b C:\temp
(pootle_venv)> pip install -r requirements\dev.txt -b C:\temp -t C:\pootle_
→venv\Lib\site-packages\ --no-index --find-links="C:\temp"
(pootle_venv)> cd C:\git\pootle
(pootle_venv)> pip install -e .
```

Now that all the requirements are lined up, we are ready to initialise Pootle. You should be able to initialise the Pootle demo database the same way as on a Linux system.

Note: Depending on how successfully your system has engaged the virtual environment, you may have to execute pootle commands with `python pootle/runner.py` from the pootle root folder instead (e.g. `python pootle/runner.py migrate` instead of `pootle migrate`).

```
(pootle_venv)> pootle init --dev
(pootle_venv)> pootle migrate
(pootle_venv)> pootle initdb
```

Next, you will need to set up the client-side bundles with NPM. It might be necessary to deactivate the virtual environment or use a separate command window to perform this step, but it might also ‘just work’ from within the venv.

```
C:\git\pootle> cd pootle\static\js
C:\git\pootle\pootle\static\js> npm install
```

Once NPM install has completed, the actual javascript bundles can be compiled:

```
(pootle_venv)> cd C:\git\pootle
(pootle_venv)> pootle webpack --dev
```

The `webpack` command will keep running after it’s completed, to monitor your javascript files for changes so that it can auto-recompile as you work. You’ll need to either exit it with `Ctrl+C` once it has settled down, or else open up a new command prompt and activate your virtual environment there too.

One last javascript pack needs to be compiled to complete the client-side preparations:

```
(pootle_venv)> pootle compilejsi18n
```

Now create and verify a super-user as normal:

```
(pootle_venv)> pootle createsuperuser  
[Follow on-screen prompts.]  
(pootle_venv)> pootle verify_user [username]
```

Pootle is now ready to be fired up!

You will need to run one RQWorker and one Pootle server, so you'll need two command prompt windows (as both will remain active until you disable the server):

```
(pootle_venv)> pootle rqworker
```

```
(pootle_venv)> pootle runserver
```

Congratulations, Pootle should now be running comfortably! Happy hacking on Windows!!

1.6.6 Front-end Development

Parts of Pootle front-end development require a Node.js run-time and packages installed via [npm](#). This is only the case for developing or building Pootle.

Setting Things Up

In order to setup the front-end development environment, it's necessary to have Node.js installed. Please check the [installation instructions for your OS](#).

Warning: If you are using versions provided by your system then you need at least *npm* $\geq v1.4.3$ for installation to work correctly. To upgrade, use `[sudo] npm install npm@latest -g`.

Once Node.js is available, Pootle dependencies need to be installed.

```
$ cd pootle/static/js  
$ npm install
```

This will read the `package.json` file and install the development dependencies.

Building Scripts

Simply run:

```
(env) $ pootle compilejsi18n  
(env) $ pootle webpack --dev
```

This will make sure to build all the necessary scripts and create the relevant bundles with source maps support. It will also watch for changes in scripts so you don't need to constantly be running this.

For creating a production-ready build, use:

```
(env) $ pootle webpack
```

This will also run the output through [UglifyJS](#), making the output build considerably lighter in size.

Note that this step is also done as part of the `make assets` command, so you may only want to run the latter.

1.6.7 Customizing Pootle

In some cases it might be desirable to customize the styling of Pootle to fit in with your other websites or other aspects of your identity. It might also be required to add a common header or footer for proper visual integration and even adjust and enhance existing functionality.

It's highly recommended to put any custom changes separate from the distributed files, so that upgrades are unlikely to affect your customizations.

Customizing templates

In case you need to change a template, copy it into your custom `TEMPLATE_DIRS` directory with the same path name as it had before.

Warning: If you edit any templates, keep in mind that changes to the text could result in untranslated text for users of the non-English user interface.

You can customize specific blocks of templates by indicating which template the current template is customizing. Use the `{% overextends %}` template tag for that (requires to install the `django-overextends` package). This must be the first tag in the template.

```
{% overextends 'browser/overview.html' %}

{% block pre_content %}
{{ block.super }}
<h1>My custom content</h1>
{% endblock %}
```

Check the original templates in order to know which blocks can be customized.

On upgrades, you will want to check if the templates and the contained blocks differ.

Customizing JavaScript

You can place any custom scripts in your custom `STATICFILES_DIRS` directory and make them part of the default Pootle bundles by adding a very simple `manifest.json` file under the `js/` directory of your custom `STATICFILES_DIRS`.

This file must contain an object of key-values where the keys correspond to the entry points defined by Pootle and the values are arrays of module names to include in the output bundle. Check out the `pootle/static/js/webpack.config.js` file to see the existing entry points.

Example:

```
{
  "common": ["login.js", "extra_module.js"]
}
```

In the example above, the `login.js` and the `extra_module.js` JavaScript modules will be added as part of the `common` bundle. If `common` didn't exist as an entry point before, a new bundle will be output.

Note that the `manifest.json` file has to be valid JSON, otherwise it will be omitted.

Custom scripts can `require()` Pootle modules that are part of the core bundles by prefixing paths with `pootle/`. For instance the `require('pootle/models')` call will make Pootle's own `models` module available in the scope of a 3rd party script.

Needless to say, you can refer to your custom scripts the same way as you would refer to any other static asset, i.e. by using the `{% static %}` template tag.

Customizing CSS

Create any needed files under your custom `STATICFILES_DIRS` and reference them from your custom templates using the `{% static %}` template tag. You can also inline styles in your templates as usual.

Customizing images

You should put your custom images in your custom `STATICFILES_DIRS`. From CSS you would just reference them using a relative path.

On the contrary, if you want to reference images from HTML code or inline CSS, you should use the `{% static %}` template tag.

Customising robots.txt

The site `robots.txt` file uses a static template. There is no dynamic content. Place your custom `robots.txt` file in your custom templates folder and this will be served instead of Pootle's default `robots.txt` file.

Larger installs will want to serve this file statically, bypassing Django completely. You can safely serve the `robots.txt` template file as is, statically, via your configured webserver.

Installing JS build libraries

Before you can rebuild your static assets with any CSS or JavaScript customisations, you will need to install some Node.js libraries.

Before proceeding please make sure you have *Node.js and npm installed in your system*.

```
(env) $ cd $pootle_dir
(env) $ cd pootle/static/js/
(env) $ npm install
```

`$pootle_dir` is the directory where Pootle is installed.

Rebuilding assets after customization

Before rebuilding your assets for the first time you must *install the JavaScript build libraries*.

After doing any customizations, you will need to regenerate any modified bundles and gather all the static assets in a single place for public consumption.

You will need to activate your virtual environment before running these commands.

```
(env) $ pootle webpack
(env) $ pootle collectstatic --noinput --clear -i node_modules
(env) $ pootle assets build
```

1.6.8 Supported Browsers

Pootle targets the **latest stable** versions of major modern web browsers.

Pootle should not only work correctly, but it should also look great in *Firefox*, *Chrome* and *Safari*.

Internet Explorer is an exception, where we support the latest two stable versions (as of today, IE11+). Here Pootle should work well, but might look imperfect.

Older browser versions might work properly too, but we are not committed to ensure such support.

A nice to have goal is making Pootle usable in smaller screens such as iPads. But this is not a hard requirement.

If you are about to use a feature which might not be available in the set of supported browsers, check the [Can I Use...](#) website first.

1.6.9 Testing

Warning: Work in progress. For now only Python testing is being added. Future coverage will include JavaScript code too.

Pootle tests use the full-featured [pytest testing tool](#) and its integration with Django via [pytest-django](#).

To test simply run:

```
.. code-block:: console
```

```
(env) $ py.test
```

from the root of the repository. Note that you need to install the testing requirements into your virtualenv first (*requirements/tests.txt*).

Note: Since the test runner automatically sets the `DEBUG` setting to `False`, the static assets need to be collected before running the view tests. You can run `make assets` for building them.

The `py.test` runner command offers several options which are extended by plugins as well. Check [its documentation](#) for further details.

Settings for Tests

Some testing-specific settings are loaded from the `tests/settings.py` file and override any previous setting you might have set in the `settings/*.conf` files.

Writing Tests

Writing new tests is easy. Just write a function whose name starts with `test_` and place it in an appropriate module under the `tests/` subdirectory.

You'll need to use plain Python assertions in test functions. Check [pytest's documentation](#) for [more information on assertions](#).

In order to use a fixture, you simply need to reference its name as a function argument. Pytest does the rest of the magic. There are [other ways to reference and use fixtures](#) as well, but most of the time you'll find yourself passing them as function arguments.

What to Test

You'll usually want to test model behavior. These tests should test one function or method in isolation. If you end up needing to test for multiple things, then you might need to split the function/method into more specific units. This allows to structure the code better.

When testing models, it's a suggested practice to avoid DB access because it makes the tests run slower, so think twice if your test actually needs DB access. At the same time, `pytest-django` encourages you to follow these best practices and disables DB access by default. If your test needs DB access, you need to explicitly request it by using the `@pytest.mark.django_db` marker.

While testing views/integration tests can also help catch regressions, they're slower to run and end up in less useful failures, so better to write fewer of these.

Fixtures

Pootle tests include some `pytest` fixtures you can reuse. They're located in `tests/fixtures/` and are loaded when the test runner is being set up.

If you have a fixture which is very specific you can place it in a usual `conftest.py` file in its proper context, whereas the aforementioned directory is meant to be for storing shared or general-purpose fixtures.

Model Fixtures

Model fixtures are stored under `tests/fixtures/models/`, and they are basically factory functions returning an instance of the desired model. Note that these might depend on other fixtures too.

For now these model fixtures require DB access, but since that's not what every single test might need, we might want to combine this with other more complete solutions like `factory_boy` in the future.

1.6.10 Release Process

This document describes the release process Pootle follows starting from version 2.5.

Principles

- *Align Pootle releases with Django releases*, keeping compatibility with the latest version of the framework and avoiding the use, and maintenance headache, of deprecated code.
- *Time-based feature releases every six months*, this ensures that users, who don't want to run from *master*, and packagers have regular features updates.
- *Master is always stable*, this ensures that anyone can run a production server from *master*. It also reduces our effort of maintaining multiple branches in development. Lastly, it helps create a discipline of landing stable features.

Rules

The principles above extended into these rules.

1. Feature releases are made every six months.
2. Feature releases (as distinct from a bug fix release) are only against the latest Django version that Pootle supports i.e. we won't backport features.

3. Security fixes are made to the last two time-based releases.
4. Older time-based releases are no longer supported.
5. Pootle should run on Django N and N-1.
6. *master* is always releasable.
7. All schema related and major changes are made in feature branches.
8. One month before a time-based release, when *master* is in a stabilising period, schema and feature changes should not landed on *master*.

Version Numbering

A Pootle version number consists of Major-Minor-Point-Bugfix as in 2.5.0 or 2.6.1.2

Pootle's minor number is changed to indicate the latest version of Django that is supported. Thus when the latest version of Django is released, and Pootle gains support for this version, then the Pootle minor number will change.

Note: Pootle 2.5.0 is an exception to this rule. It did not support Django 1.5 at the time of release.

Every six months, when a new release train is ready to be shipped, Pootle's point version will be incremented.

Any critical security fixes will automatically result in a new bugfix release.

Examples

Understanding the number and release train with some examples:

Django 1.5 is the latest version of Django:

- Pootle is named 2.5 and should support *Django 1.5*.
- Pootle 2.5.0 is released as the first time-based release.
- Next time-based release would be 2.5.1.

A security issue is detected in Pootle 2.5.0

- The first security release 2.5.0.1 is made
- Next time-based release is still 2.5.1

Django 1.6 is released:

- Current Pootle release is 2.5.4, next Pootle release will be 2.6.0
- When 2.6.0 is out we will support Pootle 2.6.0 and 2.5.4, all previous versions will be unsupported.

A security issue is discovered which impacts all our supported time-based releases:

- We release 2.6.0.1 and 2.5.4.1

Time-based release 2.6.1 is released six months after 2.6.0

- We now support 2.6.1 and 2.6.0
- Support is dropped for 2.5.4 which is now a year old.

The Release Train: Point Releases Every Six Months

Within the principle that *master* is always deployable we aim to ensure a period of stability to allow easier release in the month prior to a time-based release.

First-Fifth month All major work and features are allowed, strings may be broken.

Sixth month Feature work that doesn't change the DB schema, bug fixes, refinements and translations. Strings are frozen.

If for some reason there's feature work that changes the schema during month six of the release train, the feature will go in its own branch and won't be merged until the next release train starts.

Security fixes are applied anytime in the release train.

Branching Strategy

The next Pootle version is always baked in the *master* branch. Exceptions are security fixes which are committed in *master* and cherry-picked to the currently supported time-based release branches.

A new time-based release is made off of *master*, incrementing the point version. Every time a new release happens, a new branch is created. These branches are named after their version numbers: if *master* is to become version 2.6.2, then the new branch will be named *stable/2.6.2*. The actual release is also tagged, in this case as 2.6.2.

Security fixes are made on the relevant release branches. So the first security release on *stable/2.6.2* would be tagged as 2.6.2.1.

Features that produce schema changes or are quite invasive go into feature branches named *feature/<feature-name>*. Once the feature is ready to be integrated within the first phase of the release train, they're merged into *master*.

1.6.11 Glossary

Translation Store A file that stores translations (e.g. a PO file) — although it could also be used to refer to other ways of storing translations.

Contains a number of Translation Units, which contain messages.

Translation Unit At the simplest level contains a single **source** string (the original message) and a single **target** string (the translated message).

XLIFF refers to this as a unit, Gettext calls it a message or string. Some industry tools talk of segments. To maintain consistency we refer to **string** in the GUI and **unit** in the code.

Monolingual formats (like .properties, OpenOffice SDF, DTD, HTML, etc.) only contain a source strings.

However when handling plurals the source may actually contain different variants of a message for different plural forms (e.g. in English, the singular and plural), and the target as well (the number of variants in source and target strings are often different because different languages handle plurals differently).

Language They refer to the languages translated into.

Project They refer to the different programs/sets of messages we translate.

Translation Project A set of translation stores translating a project into a language.

Template A translation file that contains only the source or original texts.

Translation States

Untranslated A unit that is not translated i.e. blank.

Incomplete See: Needs Attention i.e. Untranslated + Fuzzy

Translated The unit has a translation.

Fuzzy In Gettext PO fuzzy means that a unit will need to be reviewed and will not be used in production. On Pootle for the user we call this ‘Needs Work’ as the term fuzzy is either technical for some users, or confusing to those who use the term fuzzy for Translation Memory, as in ‘fuzzy match’.

Needs work See: Fuzzy

Needs review Currently see: Fuzzy In the future this will actually mean that the translated string still requires review.

Needs attention Untranslated + Fuzzy

Pootle internals

Context object (`ctx_obj`) An object representing the context that encloses the current view.

If we are navigating through the files for an existing translation project, the context object will refer to the current translation project.

Similarly, if we are in the overview page for a language, the context will point to the current language object. In the overview page for a project, the context object points to the current project.

At a higher level, the root directory is considered the context object.

Resource object (`resource_obj`) An object representing the resource that the current view is referring to.

For example, if we are navigating through the files and directories for an existing translation project, the resource object will refer to the current file or directory object.

If the current view refers to multiple resources, the resource object is the same as the context object.

1.6.12 Styleguide

Pootle developers try to stick to some development standards that are gathered in this document.

Python and documentation

For Python code and documentation Pootle follows the [Translate Styleguide](#) adding extra clarifications listed below.

- [Python style conventions](#)
- [Documentation style conventions](#)

Pootle-specific Python guidelines

Pootle has specific conventions for Python coding style.

Imports

Like in [Python import conventions](#) in Translate styleguide, but imports should be grouped in the following order:

1. `__future__` library imports
2. Python standard library imports
3. Third party libraries imports (Including Translate Toolkit ones)
4. Django imports
5. Django external apps imports
6. Other Pootle apps imports
7. Current package (or app) imports, using explicit relative imports (See [PEP 328](#))

Check [Python import conventions](#) in Translate styleguide for other conventions that the imports must follow.

```
import re
import sys.path as sys_path
import time
from datetime import timedelta
from os import path

from lxml.html import fromstring
from translate.storage import versioncontrol

from django.contrib.sites.models import Site
from django.db import models
from django.db.models import Q
from django.db.models.signals import post_save

from tastypie.models import ApiKey

from pootle_language.models import Language
from pootle_translationproject.models import TranslationProject

from .forms import GoalForm
from .models import Tag
```

Order in models

Model's inner classes and methods should keep the following order:

- Database fields
- Non database fields
- Default objects manager
- Custom manager attributes (i.e. other managers)
- `class Meta`
- `def natural_key()` (Because it is tightly related to model fields)
- Properties
- Any method decorated with `@classmethod`
- `def __unicode__()`

- `def __str__()`
- Any other method starting with `__` (for example `__init__()`)
- `def save()`
- `def delete()`
- `def get_absolute_url()`
- `def get_translate_url()`
- Any custom methods

Fields in models and forms

- If the field declaration fits in one line:
 - Put all the options on that line,
 - Don't put a comma after the last option,
 - The parenthesis that closes the field declaration goes just after the last option.
- If the field declaration spans to several lines:
 - Each option goes on its own line (including the first one),
 - The options are indented 4 spaces,
 - The last option must have a comma after it,
 - The closing parenthesis in the field declaration goes on its own line, aligned with the first line in the field declaration.

```
class SampleForm(forms.Form):
    # Field declaration that spans to several lines.
    language = forms.ChoiceField(
        label=_('Interface Language'),
        initial="",
        required=False,
        widget=forms.Select(attrs={
            'class': 'js-select2 select2-language',
        }),
        help_text=_('Default language for using on the user interface.'),
    )
    # One line field declaration.
    project = forms.ModelChoiceField(Project, required=True)
```

URL patterns

When writing the URL patterns:

- URL patterns can be grouped by putting a blank line between the groups.
- On each URL pattern:
 - Specify the URL pattern using the `url()` function, not a tuple.
 - Each parameter must go on its own line in all cases, indenting them one level to allow easily seeing the different URL patterns.

- In URLs:
 - * Use hyphens, never underscores.
 - * To split long URLs use implicit string continuation. Note that URLs are raw strings.
- URL pattern names must be named like `pootle-{app}-{view}` (except in some specific cases):
 - `{app}` is the app name, which sometimes can be shortened, e.g. using **tp** to avoid the longish **translationproject**. The chosen app name must be used consistently across all the URL patterns for the app.
 - `{view}` is a unique string which might consist on several words, separated with hyphens, that might not match the name of the view that is handled by the URL pattern.
 - The exceptions to this naming convention are:
 - * URL patterns for AJAX views must be named like `pootle-xhr-{view}`.
 - * URL patterns in `pootle_app` app must be named like:
 - `pootle_app` admin URLs must be named like `pootle-admin-{view}`
 - Other `pootle_app` URLs must be named like `pootle-{view}`.

```
urlpatterns = patterns('pootle_project.views',
    # Listing of all projects.
    url(r'^$',
        'projects_index'),

    # Whatever URLs.
    url(r'^incredibly-stupid/randomly-long-url-with-hyphens-that-is-split-'
        r'and-continued-on-next-line.html$',
        'whatever',
        name='pootle-project-whatever'),

    # Admin URLs.
    url(r'^(?P<project_code>[/]*)/admin.html$',
        'project_admin'),
    url(r'^(?P<project_code>[/]*)/permissions.html$',
        'project_admin_permissions',
        name='pootle-project-admin-permissions'),
)
```

Variables naming

In order to have a more consistent code the use of specific names for some heavily used variables is encouraged:

- `ctx`: Name for the dictionary with the context passed to a template for rendering. Also known as *context*, *template variables* or *template vars*.

```
# Good.
ctx = {
    'language': language,
}

# Bad.
context = {
    ...
```

(continues on next page)

(continued from previous page)

```
templatevars = {
    ...

template_vars = {
    ...
```

Settings naming

Pootle specific settings must be named like `POOTLE_*`, for example: `POOTLE_ENABLE_API`, `POOTLE_VCS_DIRECTORY` or `POOTLE_MARKUP_FILTER`

Pootle-specific documentation guidelines

For documenting several things, Pootle defines custom Sphinx roles.

- Settings:


```
.. setting:: POOTLE_TITLE
```

To link to a setting, use `:setting:`POOTLE_TITLE``.

- Icons:

```
Some reference to |icon:some-icon| in the text.
```

This allows you to easily add inline images of icons used in Pootle. The icons are all files from `pootle/static/images/sprite`. If you were referring to an icon `icon-edit.png` then you would use the syntax `|icon:icon-edit|`. The icon reference is always prefixed by `icon:` and the name of the icon is used without the extension.

E.g. `|icon:icon-google-translate|` will insert this  icon.

- Pootle and Django commands:

```
.. django-admin:: sync_stores
```

To link to a command, use `:djadmin:`sync_stores``

JavaScript

Follow the great [Airbnb JavaScript Style Guide](#). Go check it out for all the details.

As a summary, that includes:

- 2-space indent.
- Single quotes.
- `pascalCase` variable naming.

In addition to that:

- Try to be in the 80 (+4) soft character limit, but be wise to know when to make exceptions.
- Use [ES2015](#).

- Avoid `jQuery`.

When dealing with existing or legacy code, also keep in mind to:

- Prefix with `$` Variables holding `jQuery` objects.
- Use `js-` to prefix selectors for elements queried via JavaScript.

React + JSX

For React + JSX code also follow the [Airbnb React/JSX Style Guide](#), with the following exceptions:

- Naming extensions: Use `.js` extension for React components (not `.jsx`).
- Use `React.createClass({})` over extending `React.Component`.

Also bear in mind the following:

- Event handler naming: `handle*()` for methods, `on*()` for props.
- `propTypes`: sort them alphabetically, but also group them to place `isRequired` types first.

HTML

Indenting

- Indent using 2 spaces. Don't use tabs.
- Although it's desirable to avoid lines longer than 80 characters, most of the time the templating library doesn't easily allow this. So try not to extend too much the line length.

Template naming

- If a template name consists on several words they must be joined using underscores (never hyphens), e.g. *my_precious_template.html*
- If a template is being used in AJAX views, even if it is also used for including it on other templates, the first word on its name must be *xhr*, e.g. *xhr_tag_form.html*.
- If a template is intended to be included by other templates, and it is not going to be used directly, start its name with an underscore, e.g. *_included_template.html*.

Quoting

- Always use double quotes for HTML attribute values.
- Always use single quotes for Django template tags and template filters located inside HTML attribute values.

```
<!-- Good -->
<a href="{% url 'whatever' %}" class="highlight">
```

CSS

Indenting

- Indent using 4 spaces. Don't use tabs.
- Put selectors and braces on their own lines.

Good:

```
.foo-bar,
.foo-bar:hover
{
    background-color: #eee;
}
```

Bad:

```
.foo-bar, .foo-bar:hover {
    background-color: #eee;
}
```

Naming

- Selectors should all be in lowercase and consequent words should be separated using dashes. As an example, rather use `.tm-results` and not `.TM_results`.

1.6.13 Deprecation

From time to time features, commands, configurations will be deprecated. We deprecate and manage backward compatibility within the following guidelines:

1. Our priority is the movement of Pootle development forward. Thus:
 - (a) We don't want to have to maintain backward compatibility for too long as it hampers forward mobility.
 - (b) We won't maintain backward compatibility if that prevents or impacts the needs of the new feature, refactoring, etc.
 - (c) We won't maintain backward compatibility if the cost of that far outweighs the effort of reconfiguring Pootle.
2. We don't want there to be major disruptions that we can avoid with point release. That is it shouldn't be painful as we shift features.
3. Nothing is forever. We won't maintain deprecation or backward compatibility for long.

The “rules” of deprecation

So some rough “rules”. These apply to features, management commands and settings.

1. If it's **not released**. Drop it and tell others on [Pootle development channel](#). If it has settings add them to the settings deprecation infrastructure to force removal if required.
2. If it is **obsolete or replaced** with an equivalent then drop with no fanfare and add settings to the deprecation infrastructure so that an admin will remove settings from their settings files. Add to release notes if needed.
3. If it has been **renamed**. Put that in the release notes and allow fallback for one version. Use deprecation infrastructure for settings to allow old settings to continue to work until the N+1 release. After that its a hard failure. For commands simply rename.
4. If things **changed**. For settings put that in release notes and do a hard failure to ensure that admins will reconfigure. For commands, just put those notes in the release notes and in the command features.
5. If **removed**. Put in release notes. For settings choose either hard or soft failure depending on whether something needs to be done by the admin. Put in release notes together with a guide on how to work around the missing feature if its possible. for commands simply make sure they are highlighted as removed in the release notes.

Implementing a deprecated setting

1. Add the newly deprecated setting to `pootle/core/utils/deprecation.py` DEPRECATIONS.
2. Move the deprecated setting to the deprecated section in the *settings* document. With the needed `.. deprecated:: N.M` marker.
3. Add to the release notes.

1.6.14 Making a Pootle Release

This page is divided in four sections. The first one lists the tasks that must be performed before creating a package. The second section includes a list of tasks to get a valid package. The third one to get the package published and the release announced. The last one lists and suggests some possible cleanup tasks to be done after releasing.

Note: Please note that this is not a complete list of tasks. Please feel free to improve it.

Pre-release tasks

Before starting the release process it is necessary to perform some previous tasks.

Upload and announce new translations

We need to give localizers enough time to localize Pootle. They need time to do the actual translation and to feedback on any errors that they might encounter.

First upload the new translations:

1. Create the new templates:

```
$ git clone git@github.com:translate/pootle.git pootle-translations
$ cd pootle-translations
$ make pot
```

2. Upload the templates to Pootle for translation.
3. Update current translations against templates either on Pootle or in code and commits these updated files to Git.

Announce the new translations on the following channels:

- The News tab on Pootle – for those not on any mailing list
- The translate-announce@lists.sourceforge.net and the translate-pootle@lists.sourceforge.net mailing lists – for those who might miss the news.

String freeze

A string freeze would normally run between an RC and a final version. We want to give a string freeze at least 2-4 weeks before a release. They must be announced, explicitly stating the duration, on the translate-announce@lists.sourceforge.net and the translate-pootle@lists.sourceforge.net mailing lists.

Note: If we do have a string freeze break then announce it to people. The string freeze breaks usually are only allowed to fix mistakes on the translatable strings.

Create the package

The first steps are to create and validate a package for the next release.

Get a clean checkout

We work from a clean checkout to ensure that everything you are adding to the build is what is in the repository and doesn't contain any of your uncommitted changes. It also ensures that someone else could replicate your process.

```
$ git clone git@github.com:translate/pootle.git pootle-release
$ cd pootle-release
$ git submodule update --init
```

Create a working branch

It is extremely advisable to create a Pull Request with the changes necessary for the release. This means that it is best to create a working branch to clearly separate those changes from *master* branch, which can be altered while doing the release, and ease rebasing if needed.

```
$ git remote add my-fork git@github.com:my-fork/pootle.git
$ git fetch my-fork
$ git checkout -b releasing
```

Update requirements versions

Update the minimum version number for the requirements in:

- requirements/
- pootle/checks.py
- docs/server/requirements.rst

Make sure version numbers displayed on documentation examples match the latest requirements on the above files.

Check copyright dates

Update any copyright dates in docs/conf.py:copyright and anywhere else that needs fixing.

```
$ git grep 2013 # Should pick up anything that should be examined
```

Set build settings

Create ~/.pootle/pootle_build.conf with the following content:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""Configuration file to build Pootle.

Must be placed in ~/.pootle/pootle_build.conf
"""

# Django now requires to set some secret key to be set.
SECRET_KEY = '__BuildingPootle_1234567890__'

# Silence some checks so the build output is cleaner.
SILENCED_SYSTEM_CHECKS = [
    'pootle.W004', # Pootle requires a working mail server
    'pootle.W006', # sqlite database backend is unsupported
    'pootle.W010', # DEFAULT_FROM_EMAIL has default setting
    'pootle.W011', # POOTLE_CONTACT_EMAIL has default setting
]
```

Update checks descriptions

The quality checks descriptions are kept as a static HTML page that has to be regenerated in order to ensure the descriptions match the currently available quality checks.

```
$ mkvirtualenv build-checks-templates
(build-checks-templates)$ pip install --upgrade setuptools pip
(build-checks-templates)$ pip install -r requirements/build.txt
(build-checks-templates)$ export POOTLE_SETTINGS=~/.pootle/pootle_build.conf
(build-checks-templates)$ DJANGO_SETTINGS_MODULE=pootle.settings ./setup.py build_
↪checks_templates
(build-checks-templates)$ deactivate
$ unset POOTLE_SETTINGS
$ rmvirtualenv build-checks-templates
```

Update translations

Update the translations from the [Pootle server](#)

1. Download all translations

```
$ make get-translations
```

2. Update `pootle/locale/LINGUAS` to list the languages we would like to ship. While we package all PO files, this is an indication of which ones we want packagers to use. The requirement is roughly 80% translated with no obvious variable errors. Languages with a small userbase can be included.

```
$ make linguas
```

Check the output and make any adjustments such as adding back languages that don't quite make the target but you wish to ship.

3. Build translations to check for errors:

```
$ ./setup.py build_mo # Build all LINGUAS enabled languages
$ ./setup.py build_mo --check # Not all of these are errors
```


Create release notes

We create our release notes in reStructured Text, since we use that elsewhere and since it can be rendered well in some of our key sites.

First we need to create a log of changes in Pootle, which is done generically like this:

```
$ git log $previous_version..HEAD > docs/releases/$version.rst
```

Or a more specific example:

```
$ git log 2.5.0..HEAD > docs/releases/2.5.1.rst
```

Edit this file. You can use the commits as a guide to build up the release notes. You should remove all log messages before the release.

Note: Since the release notes will be used in places that allow linking we use links within the notes. These should link back to products websites ([Virtaal](#), [Pootle](#), etc), references to [Translate](#) and possibly bug numbers, etc.

Read for grammar and spelling errors.

Note: When writing the notes please remember:

1. The voice is active. ‘Translate has released a new version of Pootle’, not ‘A new version of Pootle was released by Translate’.
 2. The connection to the users is human not distant.
 3. We speak in familiar terms e.g. “I know you’ve been waiting for this release” instead of formal.
-

We create a list of contributors using this command:

```
$ git log 2.5.0..HEAD --format='%aN, ' | awk '{arr[$0]++} END{for (i in arr){print_  
→arr[i], i;}}' | sort -rn | cut -d\ -f2-
```

Cane caches

Bump the version for each of the apps so the caches are caned after upgrade:

- `apps.py` for each of the apps that have a *version*

Up version numbers

Update the version number in:

- `pootle/__init__.py:VERSION`
- Documentation examples, especially `docs/server/installation.rst` and `docs/server/upgrading.rst`

The version tuple should follow the pattern:

```
(major, minor, micro, candidate, extra)
```

E.g.

```
(1, 10, 0, 'final', 0)
(2, 7, 0 'alpha', 1)
```

When in development we use ‘alpha’ with extra of 0. The first release of a minor version will always have a micro of .0. So 2.6.0 and never just 2.6.

Install nvm

Most likely your system will provide a nodejs version older than the one that is required. nvm is a tool that allows to quickly install and switch nodejs versions.

Follow the [nvm installation instructions](#).

Build the package

Building is the first step to testing that things work. From your clean checkout run:

```
$ mkvirtualenv build-pootle-release
(build-pootle-release)$ nvm install stable
(build-pootle-release)$ pip install --upgrade setuptools pip
(build-pootle-release)$ pip install -r requirements/build.txt
(build-pootle-release)$ pip install -e .[dev]
(build-pootle-release)$ export PYTHONPATH="${PYTHONPATH}:"`pwd` "
(build-pootle-release)$ export POOTLE_SETTINGS=~/.pootle/pootle_build.conf
(build-pootle-release)$ ./setup.py build_mo          # Build all LINGUAS enabled_
↪languages
(build-pootle-release)$ ./setup.py build_mo --all    # If we are shipping an RC
(build-pootle-release)$ make build
(build-pootle-release)$ deactivate
$ unset POOTLE_SETTINGS
```

This will create a tarball in `dist/` which you can use for further testing.

Note: We use a clean checkout just to make sure that no inadvertant changes make it into the release.

Test install and other tests

The easiest way to test is in a virtualenv. You can test the installation of the new release using:

```
$ mkvirtualenv test-pootle-release
(test-pootle-release)$ pip install --upgrade setuptools pip
(test-pootle-release)$ pip install dist/Pootle-$version.tar.bz2
(test-pootle-release)$ pip install mysqlclient
(test-pootle-release)$ pootle init
```

You can then proceed with other tests such as checking:

1. Documentation is available in the package
2. Assets are available in the package
3. Quick SQLite installation check:

```
(test-pootle-release)$ pootle migrate
(test-pootle-release)$ pootle initdb
(test-pootle-release)$ pootle runserver --insecure
(test-pootle-release)$ # Browse to localhost:8000
```

4. MySQL installation check:

- (a) Create a blank database on MySQL:

```
mysql -u $db_user -p$db_password -e 'CREATE DATABASE `test-mysql-pootle`
↳DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;'
```

- (b) Change the database settings in the settings file created by `pootle init` (by default `~/.pootle/pootle.conf`) to use this new MySQL database

- (c) Run the following:

```
(test-pootle-release)$ pootle migrate
(test-pootle-release)$ pootle initdb
(test-pootle-release)$ pootle runserver --insecure
(test-pootle-release)$ # Browse to localhost:8000
```

- (d) Drop the MySQL database you have created:

```
mysql -u $db_user -p$db_password -e 'DROP DATABASE `test-mysql-pootle`;'
```

5. MySQL upgrade check:

- (a) Download a database dump from [Pootle Test Data repository](#)
- (b) Create a blank database on MySQL:

```
mysql -u $db_user -p$db_password -e 'CREATE DATABASE `test-mysql-pootle`
↳DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;'
```

- (c) Import the database dump into the MySQL database:

```
mysql -u $db_user -p$db_password test-mysql-pootle < $db_dump_file
```

- (d) Run the following:

```
(test-pootle-release)$ pootle migrate
(test-pootle-release)$ pootle runserver --insecure
(test-pootle-release)$ # Browse to localhost:8000
```

- (e) Drop the MySQL database you have created:

```
mysql -u $db_user -p$db_password -e 'DROP DATABASE `test-mysql-pootle`;'
```

6. Check that the instructions in the [Installation guide](#) are correct
7. Check that the instructions in the [Upgrade guide](#) are correct
8. Check that the instructions in the [Hacking guide](#) are correct
9. Meta information about the package is correct. This is stored in `setup.py`, to see some options to display meta-data use:

```
$ ./setup.py --help
```

Now you can try some options like:

```
$ ./setup.py --name
$ ./setup.py --version
$ ./setup.py --author
$ ./setup.py --author-email
$ ./setup.py --url
$ ./setup.py --license
$ ./setup.py --description
$ ./setup.py --long-description
$ ./setup.py --classifiers
```

The actual long description is taken from `/README.rst` with some tweaking for releasing.

Finally clean your test environment:

```
(test-pootle-release)$ deactivate
$ rmvirtualenv test-pootle-release
```

Publish the new release

Once we have a valid package it is necessary to publish it and announce the release.

Create a Pull Request

Before merging the changes you must create a Pull Request to ensure that all tests and checks pass:

```
$ git push my-fork releasing
```

Note: Of course you must wait until all automatic checks pass.

Branch the release

You should branch only when you are releasing the first stable version of a new version series, since betas and release candidates can be developed in the *master* branch. To branch do:

```
$ git checkout -b stable/2.8.x
$ git push origin stable/2.8.x
```

If you branch you will want to update the `README.rst` file so that it points correctly to branched versions of badges and documentation. Review and test the actual links created, you don't need to commit everything.

```
$ workon build-pootle-release
(build-pootle-release)$ ./setup.py update_readme -w
(build-pootle-release)$ git diff README.rst
(build-pootle-release)$ git commit README.rst -m "Adjust README to branch"
(build-pootle-release)$ deactivate
```

Also if you branch you will want to limit `requires.io` to requirements in the branch. To do so check how it was done in [this commit](#):

```
$ nano .requires.yml
$ git add .requires.yml
$ git commit -m "Requirements: Limit requires.io to branch requirements"
```

Tag the release

You should only tag once you are happy with your release as there are some things that are difficult to undo:

```
$ git tag -a 2.8.0 -m "Tag version 2.8.0"
$ git push --tags
```

Release documentation

We need a tagged release or branch before we can do this. The docs are published on Read The Docs.

- <https://readthedocs.org/dashboard/pootle/versions/>

Use the admin pages to flag a version that should be published. When we have branched the stable release we use the branch rather than the tag i.e. `stable/2.5.x` rather than `2.5.0` as that allows any fixes of documentation for the 2.5 releases to be immediately available.

Change all references to docs in the Pootle code to point to the branched version as apposed to the latest version.

Deactivate documentation that is no longer applicable.

Publish on PyPI

Note: You need a username and password on [Python Package Index \(PyPI\)](#) and have rights to the project before you can proceed with this step.

These can be stored in `$HOME/.pypirc` and will contain your username and password. Check [Create a PyPI account](#) for more details.

Run the following to publish the package on PyPI:

```
$ workon build-pootle-release
(build-pootle-release)$ pip install --upgrade pyopenssl ndg-httpsclient pyasn1 twine
(build-pootle-release)$ twine upload dist/Pootle-*
(build-pootle-release)$ deactivate
$ rmvirtualenv build-pootle-release
```

Create a release on Github

Do the following to create the release:

1. Go to <https://github.com/translate/pootle/releases/new>
2. Draft a new release with the corresponding tag version
3. Convert the major changes (no more than five) in the release notes to Markdown with [Pandoc](#). Bugfix releases can replace the major changes with *This is a bugfix release for the X.X.X branch.*

4. Add the converted major changes to the release description
5. Include at the bottom of the release description a link to the full release notes at Read The Docs
6. Attach the tarball to the release
7. Mark it as pre-release if it's a release candidate

Update Pootle website

We use github pages for the website. First we need to checkout the pages:

```
$ git checkout gh-pages
```

1. In `_posts/` add a new release posting. Use the same text used for the *Github release* description, including the link to the full release notes.
2. Change `$version` as needed. See `_config.yml` and `git grep $old_release`
3. `git commit` and `git push` – changes are quite quick so easy to review.

Announce to the world

Let people know that there is a new version:

1. Announce on mailing lists **using plain text** emails using the same text (adjusting what needs to be adjusted) used for the *Github release* description:

Warning: This has to be explicitly reviewed and approved by Dwayne so **we don't repeat the same email over and over**.

- translate-announce@lists.sourceforge.net
 - translate-pootle@lists.sourceforge.net
 - translate-devel@lists.sourceforge.net
2. Adjust the *Pootle channel* notice. Use `/topic [new topic]` to change the topic. It is easier if you copy the previous topic and adjust it.
 3. Email important users
 4. Tweet about it
 5. Update *Pootle's Wikipedia page*

Post-Releasing Tasks

These are tasks not directly related to the releasing, but that are nevertheless completely necessary.

Bump version to N+1-alpha1

If this new release is a stable one, bump the version in `master` to `{N+1}-alpha1`. The places to be changed are the same ones listed in *Up version numbers*. This prevents anyone using `master` being confused with a stable release and we can easily check if they are using `master` or `stable`.

Add release notes for dev

After updating the release notes for the about to be released version, it is necessary to add new release notes for the next release, tagged as `dev`.

Other possible steps

Some possible cleanup tasks:

- Remove your `pootle-release` checkout.
- Update and fix these releasing notes:
 - Make sure these releasing notes are updated on `master`.
 - Discuss any changes that should be made or new things that could be added.
 - Add automation if you can.
- Add new sections to this document. Possible ideas are:
 - Pre-release checks
 - Change URLs to point to the correct docs: do we want to change URLs to point to the `$version` docs rather than `latest`?
 - Building on Windows, building for other Linux distros.
 - Communicating to upstream packagers.

1.6.15 Plugins

Warning: Pootle’s plugin system is currently in an early stage of development, and may be subject to change in the future. If you have any questions or are intending to use it in your own applications you can chat with us on the [Translate development channel](#).

You can customize or extend Pootle using plugins.

A Pootle plugin is a Django application that hooks into the core functionality in Pootle.

Signals, providers and getters

Pootle emits `Signals` when key events happen. You can listen to these signals using a `receiver` to trigger custom behaviour. Pootle uses Django’s `Signals` framework for handling these types of events.

Pootle allows plugins to override the default behaviour using a `Getter` function, which are decorated with the `pootle.core.plugin.getter` decorator. Once Pootle has received a response from a plugin for a `Getter` function it stops processing any further configured functions.

Pootle allows developers to change or extend the data used by the system, by adding `Provider` functions, which are decorated with the `pootle.core.plugin.provider` decorator. With `Provider` functions Pootle will gather data from all plugins configured to provide for a given `Provider` function.

Application file structure

- `__init__.py`
- `apps.py` - Django application configuration
- `receivers.py` - receivers for signals
- `getters.py` - getter functions
- `providers.py` - provider functions

Creating a plugin application

Your application requires a [Django application configuration](#)

For an application named `pootle_custom` you need to add lines similar to the following in the `__init__.py`:

```
default_app_config = 'pootle_custom.apps.PootleCustomConfig'
```

With the above configuration you should add an `apps.py`.

At a minimum this should define the `PootleCustomConfig` class with its `name` and `verbose_name`.

It can also be used to activate receivers, providers and getters. The following application configuration activates all of them for the “custom” application.

```
import importlib

from django.apps import AppConfig

class PootleCustomConfig(AppConfig):

    name = "pootle_custom"
    verbose_name = "Pootle Custom"

    def ready(self):
        importlib.import_module("pootle_custom.receivers")
        importlib.import_module("pootle_custom.providers")
        importlib.import_module("pootle_custom.getters")
```

Setting up a provider

The following is an example of providing custom `context_data` to the Pootle `LanguageView`.

Add a file called `providers.py` with the following:

```
from pootle.core.delegate import context_data
from pootle.core.plugin import provider

from pootle_language.views import LanguageView

@provider(context_data, sender=LanguageView)
def provide_context_data(**kwargs):
    return dict(
```

(continues on next page)

(continued from previous page)

```
custom_var1="foo",
custom_var2="bar")
```

Setting up a getter

The following is an example of customizing the `Unit search_backend` for an application.

Add a file called `getters.py` with the following:

```
from pootle.core.delegate import search_backend
from pootle.core.plugin import getter

from pootle_store.models import Unit
from pootle_store.unit.search import DBSearchBackend

class CustomSearchBackend(DBSearchBackend):
    pass

@getter(search_backend, sender=Unit)
def get_search_backend(**kwargs):
    return CustomSearchBackend
```

Setting up a receiver

Pootle uses the `django.core.signals` module to handle events.

The following is an example of a receiver that emits a log warning whenever a `Store` cache is expired.

Add a file called `receivers.py` with the following code:

```
import logging

from django.core.signals import receiver

from pootle.core.signals import cache_cleared
from pootle_store.models import Store

@receiver(cache_cleared, sender=Store)
def handle_cache_cleared(**kwargs):
    logging.warning(
        "Store cache cleared: %s"
        % kwargs["instance"].pootle_path)
```

1.6.16 Docker

You can install a development and test environment using docker.

Assumptions

- You have docker installed and working on your system

- The user you are using has permissions to use docker

Create a virtualenv (optional)

Installing with virtualenv keeps your host environment clean, and separates your development environment from the host.

For example:

```
$ mkvirtualenv pootle
$ cd ~/virtualenvs/pootle
```

Clone the pootle repository

You most likely want to clone *your* fork of the pootle repository, so you can easily create Pull Requests for your changes.

```
(pootle): git clone git@github.com:$USER/pootle
(pootle): cd pootle/
```

Install host requirements

```
(pootle): pip install -r requirements/host.txt
```

Install the pootle database

The default installer will create a postgresql database with

```
(pootle): make install-dev
```

This will take some time as it loads the default projects

Run the development server

Once Pootle is installed you can run the development server with

```
(pootle): make runserver
```

1.7 Frequently Asked Questions (FAQ)

Caught out by a problem installing or running Pootle? We hope you'll find some answers here. Ideal candidates are specific installation issues that we can't integrate into the main docs. Feel free to provide updates with your own findings.

1.7.1 Installation

Does Pootle run under Python 3?

Pootle does not, yet, support Python 3 but it definitely is a goal.

Our first priority has been cleaning up the code and getting onto the latest version of Django. We've achieved that with Pootle 2.8.0.

We also want to be Django warning free, we've also achieved that in Pootle 2.8.0.

All of these were needed to ease the migration to Python 3.

Currently, we're trying to eliminate Python 2 specific changes and we're coding pylint checks to prevent any regression.

If you want to help make this happen sooner, patches are welcome.

ModuleNotFoundError: No module named 'syspath_override'

```
File "/home/pootle/env/lib/python3.6/site-packages/pootle/runner.py", line 19, in
↳ <module>
    import syspath_override # noqa
ModuleNotFoundError: No module named 'syspath_override'
```

You are running Pootle using Python 3, change your virtual environment to Python 2 and try again.

Something like this will be needed to setup your virtual environment.

```
$ mkvirtualenv --python=/path/to/python2 pootle
```

locale.Error: unsupported locale setting

Pootle assumes that you have the `en_US.utf8` locale installed on your server. If for some reason your server does not include this (you're not American or you are using a very minimal server) then you need to install that locale.

On a Debian based server simply run:

```
$ sudo dpkg-reconfigure locales
```

Installing missing system dependencies

Pootle may require you to install additional system dependencies. The majority of these relate to the installation of `lxml`, required by Pootle for XLIFF and other XML based support.

`lxml` requires compilation so we depend on build components as well as libraries for `libxml2`, `libxslt` and Python.

On Debian based system the following will install all additional system requirements:

```
$ sudo apt-get install build-essential libxml2-dev libxslt-dev python-dev python-pip_
↳ zlib1g-dev
```

What is the optimal size for translation files?

There are too many variables to give a definitive numbers.

In terms of a servers ability to handle large files, this will depend on the size or shape of the database, available system resources, the database configuration and the activity on the site.

What is helpful to be aware of is that Pootle does work on a file level. So really large translation files might become unwealdy to process, and queries to find untranslated units in the file may take longer then expected.

Our general advice is to keep related translations in the same file and this should work fine. If performance does appear to be a problem then break the large files into logical divisions to create smaller files.

2.1 Release Notes

The following are release notes used on PyPI and mailing lists for Pootle releases.

These are the changes that have happened in Pootle and may affect your server. Also be aware of the [important changes in the Translate Toolkit](#) as many of these also affect Pootle.

If you are upgrading Pootle, you might want to see some tips to ensure your *upgrade goes smoothly*.

2.1.1 Upcoming releases

2.9 release

Pootle 2.9 release notes

Not yet released

Welcome to Pootle 2.9!

If you want to try it, check one of the following:

- [Installation instructions](#)
- [Upgrading instructions](#)

Changes

- Django>=1.10.8,<1.11
- Pootle FS is enabled by default:
 - All projects are now handled by Pootle FS.

- When creating new projects:
 - * It is mandatory to set filesystem configuration right after creating it,
 - * Pootle no longer automatically imports translations from disk,
 - * Pootle no longer creates project directory on disk if missing,
 - * Name for templates in disk is now configurable.
- When adding new languages to a project:
 - * Pootle no longer tries to first import existing translations from disk,
 - * Pootle initializes the new languages using the templates in the database.
- Merged previously separated forms for handling Pootle FS specific settings of projects into other existing forms:
 - * Merged project's Pootle FS backend configuration into project form.
 - * Merged project's languages mapping configuration into project's languages form.
- Several bugfixes for Pootle FS:
 - Fix handling of obsolete and unsynced units,
 - Allow to sync to empty file,
 - Several fixes for initializing new languages from templates,
 - When language mappings change now the disk files that are tracked are immediately detected,
 - Fix for correctly stage filesystem overwrite on conflict,
 - Adjustments in migrations.
- Serialization:
 - Don't cache on serialization,
 - Allow serializing obsolete units,
 - Serialization and sync bugfixes.
- Editor:
 - No longer shows red background if all critical checks are muted.
 - Don't redirect to browse view after translating last unit unless all previous units have been translated.
- Improved performance on permissions forms by using a live search field for users.
- Fixed issues with variables in translations.
- Updated UI language discovery to try simpler language codes before trying a fallback.
- Limited several text fields in the models to 4096 characters.
- Overall documentation review and updates.

Command changes and additions

- `pootle` command can now be run with no `VIRTUAL_ENV` environment variable set.
- `sync_stores`:
 - Has been deprecated in favor of Pootle FS commands,

- Can now work with projects managed by Pootle FS,
 - Warns if there is any conflict between disk and database changes,
 - The `--force` and `--overwrite` arguments no longer have any effect in the command execution.
- `update_stores`:
 - Has been deprecated in favor of Pootle FS commands,
 - Can now work with projects managed by Pootle FS,
 - The `--force` argument no longer has any effect in the command execution.
- Removed `changed_languages` command. Use `list_languages` instead.
- Added `--yes` argument to `init` command.

Credits

This release was made possible by the following people:

Ryan Northey, Leandro Regueiro, Dwayne Bailey, Taras Semenenko, boite.

And to all our bug finders, testers and translators, a Very BIG Thank You.

2.1.2 Final releases

2.8 series

Pootle 2.8.2 release notes

Released on 15 September 2017

Welcome to Pootle 2.8.2!

If you want to try it, check one of the following:

- [*Installation instructions*](#)
- [*Upgrading instructions*](#)

Changes

- Fix for deployments without `git` command.
- Updated UI language discovery to try simpler language codes before trying a fallback.
- Updated documentation.

Credits

This release was made possible by the following people:

Leandro Regueiro, Dwayne Bailey, Ryan Northey.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Pootle 2.8.1 release notes

Released on 31 August 2017

Welcome to Pootle 2.8.1!

If you want to try it, check one of the following:

- [Installation instructions](#)
- [Upgrading instructions](#)

Changes

- **pootle** command can now be run with no `VIRTUAL_ENV` environment variable set.
- Updated documentation

Credits

This release was made possible by the following people:

Leandro Regueiro, boite.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Pootle 2.8 release notes

Released on 16 August 2017

Welcome to Pootle 2.8!

If you want to try it, check one of the following:

- [Installation instructions](#)
- [Upgrading instructions](#)

Changes in Requirements

- Django>=1.10.5,<1.11
- zlib1g-dev required for lxml
- MySQL support uses mysqlclient instead of MySQLdb
- django-transaction-hooks is no longer used

Major Changes

- Performed a security audit. Made various changes to close potential security vectors including DOS type attacks.
- Browse view optimisation. The performance of the brose mode now scales well for large projects. Statistics are now in real time.
- Improved editor performance. Retrieval of editable units now correctly retrieves pages and prevents massive database load.

- Timeline data. The timeline data has been migrated and improved to store more data changes and clean up past data. This is a step to providing more detailed change history and is needed to do unit level comparisons for Pootle FS.
- Statistics generation and display. We no longer use rqworkers to calculate statistics. Instead these are live and always correct being updated on unit changes. The system is faster then previous statistics calculations.
- Pootle FS (beta) - provides bidirectional version control synchronisation.

Details of changes

Below we provide much more detail. These are by no means exhaustive, view the [git log](#) for complete information.

- MySQL now uses the `mysqlclient` database driver instead of `MySQLdb`, this is Django's [preferred database driver](#).
- Performed a security audit
- JavaScript fixes addressing performance and memory leaks in the editor
- Improved editor performance
- Extensive review and fixes to RTL layout
- Refactored code for syncing stores - as part of the internal rework of core functionality in Pootle we reworked synchronisation code. This will eventually be dropped when we move fully onto Pootle FS.
- Changed the default `robots.txt`. It is now a static file which you can adjust for your site. The site is `allow` by default.
- Editor:
 - Suggestions:
 - * Support rejecting/accepting suggestions with comments
 - Suggesters are emailed based on new `POOTLE_EMAIL_FEEDBACK_ENABLED` setting
 - * Accept suggestion without `can review` permissions if submitted translation is 100% match.
 - * Users cannot submit suggestions that match a previously submitted suggestion or the current translation.
 - Current unit is shown at the top of the unit list with one context row, this makes it is easier to use the navigation bar and prevents UI jumpiness.
 - Faster terminology matching yielding more results by using stemming.
 - Special non-display characters are depicted by symbols from a custom-built font:
 - * Properly display whitespace as special character e.g. non breaking space, newline, tab.
 - * Technical details of escape sequences are omitted in the output displayed to end users, making it easier to enter correct data.
 - Buttons to add language specific special characters display helpful information in tooltips.
 - Errors fixed:
 - * Fixed bugs in muting/unmuting checks
 - * Fixed filtering translations by month
 - * Fixed `TypeError` error when filter in editor gets no units
 - * Fixed bug that prevented translators from clicking on context rows to edit those context units

- Translation memory:
 - * Display original and translations side-by-side for TM results
 - * AmaGama Translation Memory is now queried using CORS
 - * AmaGama is only queried if the source language is supported by amaGama
- Perform all highlighting in the client.
- Clearly present plurals to translator.
- Allow text from the similar translations area to be selected.
- Force word wrapping on long strings with no spaces.
- Improvements on timeline.
- Cross-language translation is now restricted to admins. It heavily impacts performance and translators are unlikely to require it while admins may have a valid reason to use it.
- Removed ability to clear language or project dropdown to prevent performance degradation. This prevents users inadvertently hitting very expensive queries.
- Alternate source language translations are no longer displayed for anonymous users to prevent performance costs for users who aren't able to translate.
- Editor is disabled for users without the required permissions. Reviewing suggestions is similarly disabled without required permissions.
- Check categories can now be used in dropdown to filter units. You can review all 'critical' check failures at once.
- Auto-matched translations are now highlighted to indicate that they came from translation memory to prevent confusing users.
- Incomplete plural translations may now be submitted.
- String error reporting form is now clearer and prevents empty reports from being submitted.
- Added suggestion bulk management:
 - Provides filtering by user, etc.
 - Allows to review multiple suggestions at once and reject/accept them at once optionally providing a comment for the suggesters
- Configuration system - a generic system to store configuration information for Pootle.
- Plugin framework - allowing Pootle to use plugins to expand its functionality.
- Comment system
- Removed Plurr format checks
- Removed *ENChecker*
- Added support to have several formats in the same project
- Browse pages:
 - Refactored stats backend:
 - * We now store statistics in the database and have removed the need for rqworkers to calculate stats.
 - * The stats refresh notice has been removed as all stats are now up-to-date, always.
 - * Faster stats retrieval is now possible as stats are always up-to-date and we can get it directly from the database.

- Disabled items are hidden by default but admin users can select to shown them.
- Changed order of columns to highlight latest activity and pending work:
 - * Last updated data is now only shown to admin users
- Altered order in which some items are listed by default:
 - * Projects and languages are sorted by most recent translators changes to highlight activity
 - * Virtual folders are sorted by priority to highlight most important strings to translate
- Hid most of the special ‘templates’ language data as it is unnecessary and can be confusing.
- Got rid of fat cookies:
 - * Increases responsiveness and removes security issue
 - * Most data is now stored in user session instead
 - * Sidebar is no longer automatically open for anonymous users when an announcement changes.
- Leaderboard on top panel and expanded stats panel:
 - * The top panel will show the three users with the highest score.
 - * Expanded stats shows the top contributors scores and other detailed information about current location.
- Numbers are rendered in a locale aware fashion.
- Improvements to the statistics table for overly long filenames and smaller screens.
- Files dropdown no longer keeps references to empty directories.
- Fixed issue where the “Back” button would sometimes not work.
- Fixed issue with project dropdown when there are projects without a name.
- Search:
 - No longer autocompletes
 - Added pluggable search backend
 - Search widget is disabled if user cannot translate. This is to prevent any load on the server for users who are not able to contribute.
 - Old ‘Exact Match’ was separated into ‘Case-sensitive match’ and ‘Phrase match’ allowing finer-grained searches and removing the previous confusion about the actual intent of the options.
- Added team page:
 - Only for languages so far, and only available to language managers
 - Replaces permissions with roles
 - Provides direct access to suggestion bulk management
- Revamped user profile page. The aim is to slowly draw more information onto this page and make it a hub for translators.
- Removed for performance reasons:
 - Removed statistics from user profiles. Will be brought back in the future.
 - Removed export view. This has been replaced with TMX export functionality, download still remains.
 - Removed performance hogging “More stats” in admin dashboard. While it has some useful information there are better ways to get this data.

- Removed reports feature. This was a potential security area and data leak. We will bring this back now that we have finer grained change tracking.
- Pootle's own localization changes:
 - Updated translations. You can still [contribute translation updates for your language](#).
 - Now `compilejsi18n` is used to compile JavaScript translations into assets, thus requiring `django-statici18n` app.
 - Password reset email is now localizable in Pootle.
 - Multiple changes in localizable strings to ease translation.
 - Select2 localization is bundled to ensure Select2 is shown localized.
- Upload and download:
 - Disabled upload for non-PO projects as conflict handling currently only works in PO.
 - Admins can upload translations as other user allowing correct crediting for translations.
 - Fixed error for stores with no revision.
 - Added the ability to download TMX exports.
- New Machine Translation providers:
 - [Caighdeán](#) - Irish
 - [Welsh](#)
- Refactoring of models to increase performance, including dropping unnecessary indices.
- User input is sanitized for outgoing emails
- Usernames using latin1 characters are now allowed
- Improved RQ usage and new management commands
- Changed Pootle logo and styling
- Added the ability to use a custom logo with `POOTLE_CUSTOM_LOGO`
- Documentation updates

Pootle FS (beta)

Pootle FS enables synchronization of Pootle against a filesystem, or version control system, handling conflict resolution and other situations of two files being out of sync.

Pootle FS follows a git like command execution. We've designed it such that we expect there to be no data loss when conflicts are discovered. Any conflicts are turned into suggestions which can be resolved in Pootle.

Pootle FS is still in beta as we'd like to make sure that all the bugs are washed out before making it an official and default part of Pootle.

`sync_stores` and `update_stores` are still the default method of interacting with Pootle. We expect these to remain for some time, but expect the next version of Pootle to use to Pootle FS infrastructure to manage and handle these commands.

- Added admin UI to set up projects configuration and language mapping
- CLI - adds `info`, `fetch`, `resolve`, `sync`, `add` and `rm` commands
- LanguageMapper - allows differing codes on the filesystem vs Pootle

- FileMapper - maps the file layout on the filesystem to the expected Pootle layout
- Store de/serialization - makes it possible to customise and adapt file serialisation, most likely for slight deviations from the official file format specification.
- Removed the ability to add new TPs from the admin UI for Pootle FS projects, we will initialise new TPs differently in Pootle FS.

Development changes

- Updated and pinned PyPI requirements:
 - From now on requirements will be pinned in order to simplify support and development.
- Tests:
 - Massive improvement in test framework.
 - Coverage increased from 55% to 94%.
 - Moved to tox.
 - Travis caching and optimisations.
 - Added JavaScript testing.
- Code sanity:
 - Python code cleanup/linting pep8/pyflakes/pep257 to increase code health.
 - Javascript code linting and cleanups.
 - CSS code linting and cleanups.
- Code polishing:
 - Moved all commands to argparse.
 - Moved shortcuts to Mousetrap.
 - JS improvements, move to React components.
- Triage meetings are now held on a weekly basis.

Command changes and additions

- Running Pootle commands using **manage.py** is no longer supported, use **pootle** instead.
- **pootle** command warns if configuration is missing.
- Changed commands:
 - `verify_user` and `purge_user` now accept multiple usernames.
 - `refresh_scores` now recalculates user scores and accepts multiple usernames. It can be run across projects and/or languages.
 - `contributors` command has been refactored in order to return more accurate results and has new options `--since`, `--until` and `--mailmerge`. The `--from-revision` option has been removed.
 - `flush_cache` flushes default, redis caches, accepts `--rqdata`, `--django-cache` options.
 - `export` is now able to export zipped TMX files per translation project with the `--tmx` option. `--rotate` option allows old files to be removed.

- `init` now creates a development configuration with `--dev` option.
- Added new commands:
 - `list_serializers` allows to view serializers and deserializers installed on your system.
 - `config` allows to get, set, list, append and clear configuration settings.
 - `init_fs_project`.
 - `set_filetype`.
 - `schema` allows to dump the database schema on MySQL which is useful for diagnosing differences in database schema.
 - `update_data` allows to update the stats data.
- Removed commands:
 - `run_cherry.py`.
 - `start` has been removed, use `runserver` instead.
 - `refresh_stats`.
 - `clear_stats`.

Changes in settings

- Changes in settings:
 - MySQL database connections should now use `STRICT_TRANS_TABLES`.
 - `POOTLE_TM_SERVER` no longer receives the `MIN_SCORE` parameter, as it was misleading and had questionable effects.
 - `POOTLE_TM_SERVER` now accepts a `MIN_SIMILARITY` parameter, to filter out results which might be irrelevant. To learn more, check the documentation on `MIN_SIMILARITY`.
 - Changed the default value for `ACCOUNT_SESSION_REMEMBER` so now sessions are always remembered.
 - `POOTLE_MARKUP_FILTER` defaults to `'markdown'`, and `None`, `'html'`, `'textile'` and `'restructuredtext'` values have been deprecated. Deployments using any deprecated markup must migrate manually to Markdown. This setting will be removed in the future since Markdown will be the only available markup.
- Added new settings:
 - `POOTLE_SCORES` accepts custom settings for user scores calculation.
 - `POOTLE_SEARCH_BACKEND` to allow configuring the search backend to be used.
 - `POOTLE_EMAIL_FEEDBACK_ENABLED` to allow disabling sending emails to suggesters when suggestions are accepted or rejected.
 - `POOTLE_CUSTOM_LOGO`, `POOTLE_FAVICONS_PATH`, `POOTLE_FS_WORKING_PATH` and `POOTLE_CANONICAL_URL` settings to allow easy customisations.
 - `POOTLE_SQL_MIGRATIONS`.
 - `AMAGAMA_SOURCE_LANGUAGES`.
- Removed settings:
 - `POOTLE_QUALITY_CHECKER` since the custom quality checkers feature is gone.

- `POOTLE_SCORE_COEFFICIENTS` has been removed and replaced with `POOTLE_SCORES`.

Credits

This release was made possible by the following people:

Ryan Northey, Dwayne Bailey, Julen Ruiz Aizpuru, Taras Semenenko, Leandro Regueiro, Igor Afanasyev, Claude Paroz, Safa Alfulaij, Rene Ladan, Kevin Scannell, Jason P. Pickering, Eamonn Lawlor, Alexander Lakhin, Robbie Cole, Rhoslyn Prys, Prasasto Adi, Nootan Ghimire, Mikhail Paulyshka, Mike Robinson, leonardcj, Henrik Feldt, Francesc Ortiz, Allan Nordhøy, Christian Lohmaier, Burhan Khalid, benbankes, Arash Mousavi, Andy Kittner, Adam Chainz.

And to all our bug finders, testers and translators, a Very BIG Thank You.

2.7 series

Welcome to the new Pootle 2.7.6

Released on 20 June 2016

Bugfix release for 2.7.5.

2.7.6 vs 2.7.5

Changes since 2.7.5:

- Fixed assets

View the [git log](#) for complete information.

Credits

This release was made possible by the following people:

Leandro Regueiro.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.5

Released on 21 May 2016

Bugfix release for 2.7.4.

2.7.5 vs 2.7.4

Changes since 2.7.4:

- Fixed build process
- Expanded list of RTL languages

View the [git log](#) for complete information.

Credits

This release was made possible by the following people:

Leandro Regueiro, Dwayne Bailey, Ryan Northey, Jason P. Pickering.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.4

Released on 19 May 2016

Bugfix release for 2.7.3.

2.7.4 vs 2.7.3

Changes since 2.7.3:

- Updated some requirements to prevent failures on rebuilding assets.
- Requirements use ranges to prevent installing broken versions ([issue 4737](#))

View the [git log](#) for complete information.

Credits

This release was made possible by the following people:

Leandro Regueiro, Taras Semenenko, Mikhail Paulyshka, Dwayne Bailey.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.3

Released on 27 April 2016

Bugfix release for 2.7.2.

Major Changes

- Several critical security fixes that prevent potential XSS attacks
- Pootle no longer supports the MySQL's MyISAM database backend, users are strongly encouraged to convert their database to InnoDB.
- The editor for static pages now highlights the content's markup and displays a live preview.
- Added support for Elasticsearch-based external Translation Memory servers.
- Changed connection logic and added checks for misconfigured Translation Memory servers.
- Significant speed up when importing files.

Below we provide much more detail. These are by no means exhaustive, view the [git log](#) for complete information.

Changes not reported on previous releases

There are some changes that haven't been reported on their corresponding release notes at the time:

- In release 2.7.0 support was dropped for allowing to allow users to specify their preferred UI language on Pootle. Pootle now uses the preferred languages as reported by the user's browser and falls back to English if the specified languages can't be used. See [Setting language preferences in a browser](#) for more information. The ability to specify Pootle UI language will be added back ([issue 4230](#)).

Details of changes

- Several critical security fixes that prevent potential XSS attacks
- Store update has been refactored which has brought a significant speed up when importing files.
- Static pages and announcements:
 - The editor for static pages now highlights the content's markup and displays a live preview of the rendered contents ([issue 3346](#), [issue 3766](#)).
 - Project and language announcements are now also displayed on their respective overview pages.
- Translation memory:
 - `update_tmserver`:
 - * Renamed `--overwrite` to `--refresh`.
 - * Translations from disabled projects must be explicitly included with `--include-disabled-projects`.
 - * Added support for Elasticsearch-based external Translation Memory servers, which can be populated from translation files or directories on disk. This effectively brings the ability to display TM results from different TM servers, sorting them by their score.
 - * Translations saved from Pootle now include a timestamp.
 - * Fixed missing index error [issue 4120](#).
 - `POOTLE_TM_SERVER`:
 - * The default TM server has been renamed to `local`. Make sure to adjust your settings.
 - * Added a new `WEIGHT` option to raise or lower the TM results score for each specific TM server.
 - * Added several checks to ensure this setting is not misconfigured.
 - Changed connection logic for Translation Memory servers to handle connection issues and misconfigurations on the settings.
- Database:
 - InnoDB is the supported MySQL backend. Deployments using MyISAM must *migrate to either MySQL (InnoDB) or PostgreSQL*.
 - Close a database connection before and after each rqworker job once it exceeds the maximum age to imitate Django's request/response cycle [issue 4094](#).
- Editor:
 - Non-critical checks can once again be muted/unmuted.
 - Fixed units sorting issue for admin users [issue 4116](#).
- Import/export and upload/download:

- Fixed running `export` command without options.
- Added a new `--user` to `import` to attribute changes to specified user on file import.
- Ignore non project filetypes when uploading zip files [issue 4124](#).
- Only authenticated users with translate rights can upload translations.
- Any authenticated user can now download translations.
- Translations from *Terminology* project can now also be downloaded.
- `initdb`:
 - Now has an `--no-projects` option to prevent creating the default projects at set up.
 - Now loads the translations for the default projects and languages and triggers their stats calculation.
 - Doesn't throw errors when accidentally being run more than once.
- The Apertium MT backend has been dropped.
- Report string errors form subject and body can be overridden.
- Language managers can now edit their language's special characters by using the *Special Characters* page accessible through the browse dropdown in the language overview page.
- Added extra data to reports.
- Added more languages for Yandex machine translation.
- Fixed `test_checks` errors when being run with no options and without the `--check` option.
- Pulled latest translations.

...and lots of refactoring, new tests, cleanups, improved documentation and of course, loads of bugs were fixed.

Credits

This release was made possible by the following people:

Julen Ruiz Aizpuru, Leandro Regueiro, Ryan Northey, Dwayne Bailey, Taras Semenenko.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.2 final

Released on 22 September 2015

Bugfix release for 2.7.1.

Changes in Requirements

- Django $\geq 1.7.10$, < 1.8
- Translate Toolkit $\geq 1.13.0$
- Python ≥ 2.7 , < 3.0
- Redis $\geq 2.8.4$
- Django transaction hooks

- Unix-based operating system.

Major Changes

- Bugfixes for some important issues.
- Pulled latest translations.

Below we provide much more detail. These are by no means exhaustive, view the [git log](#) for complete information.

Details of changes

- Prevent local TM from crashing if `elasticsearch` is unavailable. `elasticsearch` version must now be 1.6.0 at most.
- Prevent regular users from seeing disabled projects translation stats.
- If disabled, do not the display contact form on sign in and sign up.
- Fixed regression for admin users on export view.
- Fixed issue with translatable extraction tools that prevented several texts from being translated.
- Pulled latest translations.

... and cleanups, improved documentation.

Credits

This release was made possible by the following people:

Dwayne Bailey, Leandro Regueiro, Julen Ruiz Aizpuru, Ryan Northey, Taras Semenenko.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.1 final

Released on 18 September 2015

Bugfix release for 2.7.0.

Changes in Requirements

- Django `>= 1.7.10, < 1.8`
- Translate Toolkit `>= 1.13.0`
- Python `>= 2.7, < 3.0`
- Redis `>= 2.8.4`
- Django transaction hooks
- Unix-based operating system.

Major Changes

- Updated translations.
- Added django-transaction-hooks
- Changed user delete behaviour
- Lots of command changes and additions
- Improved upload

Below we provide much more detail. These are by no means exhaustive, view the [git log](#) for complete information.

Details of changes

Translation statistics

- Last activity snippets for stats are not kept in the cache anymore. The markup is now built on the client. This requires refreshing all server stats using the `refresh_stats` command ([issue 3835](#)).
- Renamed `refresh_stats_rq` command to `refresh_stats`. The old `refresh_stats` command is now gone.
- POT files are no longer included in the translations stats. This allows to achieve a 100% translation status for a project if all the languages are completely translated.
- Fixed issue with empty directories preventing stats from being fully calculated.
- Public virtual folders with pending suggestions are now always displayed.

Django transaction hooks

- To ensure async jobs are scheduled at the correct time `django-transaction-hooks` is now required. This dependency will be unnecessary once Django 1.9 becomes Pootle's minimum requirement.
- You must update your database connection to use one of the django-transaction-hooks backends:
 - mysql: `transaction_hooks.backends.mysql`
 - postgres: `transaction_hooks.backends.postgresql_psycopg2`

Changed user delete behaviour

On deleting a user account their submissions, suggestions and reviews are now re-assigned to the “nobody” user.

If you wish to remove the user's contributions also, you can use the `purge_user` command, or call `user.delete(purge=True)` to delete the user programmatically.

File uploads

- The uploading user now receives the credit for the upload.
- Handling of upload errors have been improved, displaying more useful messages now.
- In case of upload conflict the new translations are turned into suggestions.

Command changes and additions

- Added a `contributors` command to get the list of contributors ([issue 3867](#)).
- Added a `find_duplicate_emails` command to find duplicate emails.
- Added a `merge_user` command to get merge submissions, comments and reviews from one user account to another. This is useful for fixing users that have multiple accounts and want them to be combined. No profile data is merged. By default it removes the original user account after successful merge.
- Added a `purge_user` command to purge a user from the site and revert any submissions, comments and reviews that they have made. This is useful to revert spam or a malicious user.
- Added a `verify_user` command to automatically verify a user account
- Renamed `refresh_stats_rq` command to `refresh_stats`, replacing the old command of the same name. `refresh_stats` is able to calculate the stats for disabled projects (old `refresh_stats_rq` was unable to do it).
 - Errata: the removal of the old `refresh_stats` has removed the following options:
 - * `--calculate-checks` and `--check` – Use `calculate_checks` instead.
 - * `--calculate-wordcount`
- Added a `update_user_email` command to update a user's email address.
- Added a `--no-rq` option to run commands in a single process without using RQ workers.
- Now it is possible specify the parameters to set up your database directly through `init` command.

Editor

- Editor now request confirmation before navigating away from modified units in order to prevent data loss. This also includes non-saved comments. Going to the previous, next, and a specific unit will trigger the prompt, as well as changing filters or searching. It is also triggered by typing a different URL, reloading the page or closing the browser window.
- Fixed issue that didn't allow users with only just suggestion rights to send suggestions.
- Suggestion related events are now displayed on the timeline.
- Critical and not critical failing checks are now displayed separately in the editor.
- Potential errors when managing the suggestions are now displayed to users.
- Fixed a regression that prevented users from rejecting their own suggestions even if they don't have enough permissions to reject suggestions.

Misc changes

- Disabled projects are visually differentiated in the projects drop-down ([issue 3996](#)). Since the in-cache data structure supporting this changed, it's necessary to clear the cache. Assuming your default cache lives in the DB number 1, you can clear it as follows:

```
$ redis-cli -n 1 KEYS "*method-cache:Project:cached_dict:*" | xargs redis-cli -n 1
→ 1 DEL
```

- Admins can now always see and navigate disabled projects.

- Pulled latest translations.
- Scores now include suggestions.
- A link is now displayed on the sidebar so admin users can quickly edit the announcements.
- Now previously hidden errors during login and sign up are displayed to the user.
- Improved usage of system checks so sysadmins get better feedback on whether something is wrong with Pootle.

...and lots of refactoring, new tests, cleanups, improved documentation and of course, loads of bugs were fixed.

Credits

This release was made possible by the following people:

Julen Ruiz Aizpuru, Ryan Northey, Taras Semenenko, Leandro Regueiro, Dwayne Bailey, Jerome Leclanche, Kevin Scannell, Daniel Widerin.

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the new Pootle 2.7.0 final

Released on 4 August 2015

This is the first release of Pootle that combines the work of Evernote and Translate.

Changes in Requirements

- Django ≥ 1.7 , < 1.8
- [Translate Toolkit](#) $\geq 1.13.0$
- Python ≥ 2.7 , < 3.0
- Redis ≥ 2.8
- Unix-based operating system.

Major Changes

- Switched license from GPLv2 to GPLv3.
- The [Evernote Pootle fork](#) and Translate Pootle are now merged into the same code base and are being actively developed together.
- Major UI revamp - browsing pages are consistent across all views. Navigation is now much easier, more consistent and more powerful. The editor is cleaner and works to prevent errors early.
- Backgrounding statistics calculations - so Pootle and its users are never bogged down or delayed by real time stats calculations. Instead these are backgrounded and updated when available.
- A number of features have been *removed* and will be recovered in future releases.

Below we provide much more detail. These are by no means exhaustive, view the [git log](#) for complete information.

Details of changes

Major user interface revamp

Browsing

- Pootle browsing pages now have a single column and wide stats table that shows the same data in the different views. This create a consistent look while browsing through languages and projects.
- No home page. Users are redirected to their preferred language pages instead, falling back to the project listings page.
- Critical errors and pending suggestions are prominently displayed on the browsing page and not hidden.
- New navigation scheme:
 - All directories/files for a project are displayed in a new drop-down.
 - Directories/files can be navigated and translated across multiple languages in a project.
 - Tabs have been replaced in favor of drop-down menus.
 - The editor search box is now displayed in the action links section, keeping its positioning consistent with the browsing page.
 - A new action link in the editor, *Go back to browsing*, allows users to go back to the same place they entered translation mode from.
- When there are failing checks, browsing tables now display the number of units which have failing checks, not the total number of failing checks.
- Table sorting is now remembered across browsing pages, and not separately in project, language and translation project pages.
- User actionable items in the navbar have been moved to a drop-down.
- When selecting languages, redirect logic is now smarter.
- Different last activity messages for new translations and edits.
- Filters allow sorting units according to their last action date.
- Implemented project specific announcements in a sidebar. These make use of static pages. Notifications are per-project and are displayed across languages (automatically adapting any hyperlinks).
- Announcements for language and translation project have been also implemented.
- Major speed improvements when calculating last action information.

Editor

- New features:
 - Added context search URL prefix for projects, which allow integrating screenshots for units. This is the ability to provide a search URL (link to a search engine) where to look up the text of the current unit in.
 - Translation similarities are calculated in the client and sent to the server to calculate the user's score. The score changes are logged over time. Along with this, the best matches are highlighted in the editor.
- Navigation:
 - The editor now displays the numbering for units, not pages.

- When going through all units in the translation editor, users will be automatically redirected back to browsing.
- If the currently-submitted unit has failing checks then the editor won't advance to the next unit and it will be updated displaying the unresolved checks. The same behavior applies when a suggestion is accepted.
- The *Submit/Suggest* button is disabled until a change, over the initial state of the unit, is detected.
- Checks:
 - Quality checks are always displayed and can be individually muted/unmuted.
 - When a users mutes or unmutes a quality check, the action will be recorded in the unit's timeline.
 - A custom set of new quality checks has been incorporated. It is still possible to instead use the old quality checks.
 - It is possible filter the strings using the checks categories.
- Usability improvements and other fixes:
 - Location comments are always displayed, providing a shortcut link to display the same source unit across languages.
 - The same string can't be suggested more than once at the same time, avoiding duplicated suggestions pending review.
 - TM diffs now display what has been removed and what's being added.
 - Latest translator comments can be "removed" or, in other words, can be blanked. The action is recorded in the timeline.

Users

- New welcome page for anonymous users, which displays the top scorers over the last 30 days.
- User score is displayed in the header and dynamically updated when translation actions are made.
- Revamped user profile pages. These now display user's latest activity and their personal properties.

Miscellaneous

- Rewritten contact form. It used to both contact the site owners from any page as well as to report any issues with strings.
- Support for old web browsers has been dropped, despite this change Pootle might work on such browsers. More information on Pootle's [supported browsers](#).
- Several layout improvements that take into account different screen sizes.
- Implemented export view for cross-language and cross-project views.
- Implemented global search. This allows to perform searches and edit units in collections that span multiple projects across languages, multiple languages across projects, or even the whole server.
- Timeline tracks all changes done to units.
- Uploads and downloads have been changed slightly. There are no options to overwrite or suggest. Your files will be accepted if no changes have been made online while you translated offline. If changes were made while offline then the upload will be rejected. In such case download the file again, use your offline tools or TM to retranslate and try another upload.

- Added the initial implementation of virtual folders. This feature is meant to replace the now gone goals.

Statistics calculations

- Statistics handling has received a major rewrite (in fact 3 rewrites). [RQ](#) is used to queue and manage the calculation of invalidated statistics. On the frontend, stats are now loaded asynchronously, thus any pending calculations no longer block page loads. This results in a major usability improvement for Pootle users.
- `POOTLE_WORDCOUNT_FUNC` allows a custom word counting method to be used.
- A new counter `pootle.core.utils.wordcount.wordcount` has been incorporated (it omits placeholders and words that shouldn't be translated). Non-empty units with 0 words are immediately translated and marked as fuzzy.
- Browsing pages now report the last time a unit was added to a store/project. In the browsing tables a *Last Updated* column is displayed and in the extended stats *Created* and *Last Updated* dates.
- Statistics are also available for the *All Projects* view.

Administrators

- Only admins can set the fuzzy flag on the unit. Non-admin users are not shown the fuzzy checkbox by default, but in case it's displayed (because the unit was already fuzzy, or some other action turned it fuzzy, such as using MT), they are always forced to clear the flag.
- Projects can be disabled from the administration page, allowing them to be hidden temporarily while retaining statistics.
- The `view` permission is now project-centric, it can be set server-wide or for projects, but not for individual language-project combinations.

Sysadmins

- Direct upgrade is now only possible from 2.6.0. Thus to upgrade from older releases first upgrade to 2.6.0
- The ability to deploy using Fabric has been dropped. We advise to use the recommended upgrade and install instructions in the documentation.
- Redis is now required for all caching, memcached and other alternatives will not work.
- Exports no longer work if they are directly served by the web server. Apache users can refer to [Apache and mod_wsgi](#) for a suggested configuration.
- Registration and authentication is now handled by [django-allauth](#). gives Pootle implicit support for OpenID, OAuth, OAuth2 and Persona sign-in protocols. Check out the [documentation on users auth](#) for further details.
- Integrated Elasticsearch-based local TM server into Pootle. Unit submissions update the index instantly. To configure adjust `POOTLE_TM_SERVER` and to load the TM use the `update_tmserver` management command.
- The report target for reporting string errors has been dropped in favor of a report email address. The report is now sent using an automatically pre-filled contact form. If the project doesn't have a report email then the reports are sent to `POOTLE_CONTACT_REPORT_EMAIL`.
- Using the Django `dumpdata` and `loaddata` commands to move between databases is no longer supported. If you need to move, please use proper SQL scripts instead.
- Captcha implementation details have been refined.

- Yandex.Translate is now available as a Machine Translation backend.
- `POOTLE_QUALITY_CHECKER` can be used to point to a custom quality check handler.
- Xapian and Lucene are no longer required for searching and Pootle will not make use of them. You can safely remove supporting libraries and packages if these services were used only for Pootle.
- `POOTLE_REPORTS_MARK_FUNC` allows a site wide function to provide marks to user graphs.
- Pootle no longer runs on Windows. Pootle uses RQ which makes use of `fork()` therefore Pootle will only run on systems that implement `fork()`. Importantly that means that Pootle is no longer supported on Windows. It would be possible to run Pootle on Windows if the rqworkers are run on a system that supports `fork()`.

Command changes and additions

- Improved the way `update_stores` inserts and deletes units in the store ([issue 3802](#)).
- In `update_stores` if a directory doesn't exist while running the command, the project will be disabled. Thus the `update_translation_projects` command has been removed, it's functionality has been merged into `update_stores` with this change.
- Added the `changed_languages` management command.
- Individual quality checks can now be recalculated via the `--check` flag passed to the `refresh_stats` management command.
- Added `--calculate-checks` parameter to the `refresh_stats` command.
- `refresh_stats_rq` was added to allow statistics to be refresh when running with multiple RQ workers.
- Added a new `system` user to attribute changes done by the management commands.
- Added command and store action logging.
- Added `test_checks` management command.
- Removed `--directory` and `--path-prefix` parameters from management commands. `--project` and `--language` should be used instead to reduce the scope of commands.
- Removed the `--modified-since` flag from `sync_stores` and `update_stores`. Optimizations will automatically be done based on the latest sync revision.
- New management commands: `revision`, `refresh_scores`, `retry_failed_jobs`, `import`, `export`, `dump` and `calculate_checks`.

Deprecated settings

- All Pootle specific settings have been renamed and prefixed with `POOTLE_`. The following settings are impacted and should be renamed accordingly in your settings file:
 - `TITLE` -> `POOTLE_TITLE`
 - `CAN_CONTACT` -> `POOTLE_CONTACT_ENABLED`
 - `CAN_REGISTER` -> `POOTLE_SIGNUP_ENABLED`
 - `CONTACT_EMAIL` -> `POOTLE_CONTACT_EMAIL`
 - `PODDIRECTORY` -> `POOTLE_TRANSLATION_DIRECTORY`
 - `MARKUP_FILTER` -> `POOTLE_MARKUP_FILTER`
 - `USE_CAPTCHA` -> `POOTLE_CAPTCHA_ENABLED`

- MT_BACKENDS -> *POOTLE_MT_BACKENDS*
- POOTLE_CONTACT_REPORT_EMAIL -> *POOTLE_REPORT_STRING_ERRORS_EMAIL*
- EXPORTED_FILE_MODE -> *POOTLE_SYNC_FILE_MODE*
- OBJECT_CACHE_TIMEOUT -> *POOTLE_CACHE_TIMEOUT*
- LEGALPAGE_NOCHECK_PREFIXES -> *POOTLE_LEGALPAGE_NOCHECK_PREFIXES*
- CUSTOM_TEMPLATE_CONTEXT -> *POOTLE_CUSTOM_TEMPLATE_CONTEXT*
- *POOTLE_TOP_STATS_CACHE_TIMEOUT* has been removed with the old top stats rendering and is replaced by the new browsing UI.
- *VCS_DIRECTORY* is now deprecated as the integrated Version Control feature has been removed to come back at a later date.
- *CONTRIBUTORS_EXCLUDED_PROJECT_NAMES* and *CONTRIBUTORS_EXCLUDED_NAMES* have been removed along with the contributors' page.
- *DESCRIPTION* has been removed, use *static pages* instead.
- *ENABLE_ALT_SRC* has been removed
- *MIN_AUTOTERMS* has been removed
- *MAX_AUTOTERMS* has been removed
- *FUZZY_MATCH_MAX_LENGTH* has been removed
- *FUZZY_MATCH_MIN_SIMILARITY* has been removed
- *EXPORTED_DIRECTORY_MODE* has been removed

Internal changes

- Switched to a custom user model. This merges the data and functionality available in `auth.User` and `PootleProfile` before, and has allowed to remove the dependency on deprecated third party apps that were bundled in the code.
- The multiple `Suggestion` models have been merged into a single model.
- Changed the way units needing to be sync'ed to disk is determined. Units now have a unique revision number within the store they belong to and they'll be synchronized based on the `last_sync_revision` field of the store.
- Tests have been resurrected.
- Upgraded jQuery to 2.x and applied a bunch of fixes to the Tippy plugin, avoiding ad-hoc hacks to remove dangling tips.
- Translation projects now have a `creation_time` field.
- Dropped code for several external apps from Pootle codebase. Also upgraded to newer versions of those apps.
- Fixed and avoided any inconsistencies in the unit's submitter information.
- URLs have been unified and all follow the same scheme. URLs ending in `.html` have been removed altogether. `reverse()` and `{% url %}` are used almost everywhere.
- All templates are gathered in a single location (*pootle/templates*), and have been reorganized and sorted.
- Targetting modern browsers has allowed some CSS prefixes to be removed.
- Ability to list top scorers over a period of time.

Infrastructure

- All bugs have moved from Bugzilla to [Github issues](#).

Removed features

There are two groups of features that have been dropped:

1. Those removed that we will likely recover in future Pootle releases.
2. Legacy features that will not be coming back

Recoverable features

The following features are removed from Pootle since 2.5.1.3 and will be recovered at some time. Where possible we provide alternate approaches that can be used.

Note: sysadmins should take note of these changes and determine if this prevents use of Pootle within their environment. Essentially you will need to evaluate the use and need for each missing feature.

- Extension actions.
- Tags and Goals.
- Placeables support in the editor
- SQLite support.
- LDAP support.
- Monolingual file format support - perform file conversion to and from bilingual files outside of Pootle.
- Support for Version Control Systems - automate your version control integration outside of Pootle.
- News, notifications and RSS feeds - make use of announcement pages or use other channels of communication.
- Update against templates - do template updates outside of Pootle and use `update_stores` to load the changed files.
- Public API.
- The Wikipedia lookup backend
- No *Top Contributors* tables - user scores likely provide the information you are looking for.
- Project/Language/Translation Project descriptions - these are migrated to announcements.
- Users can no longer specify their preferred Pootle UI language on their settings nor using the language picker. Pootle UI now uses the preferred language as specified by user's browser. See [Setting language preferences in a browse](#) for more information. The ability to specify Pootle UI language will be added back ([issue 4230](#)).

Legacy features

We have dropped these features, some of which have been kept around to allow easy upgrades in the past:

Note: The removal of some of these features required extensive changes to the upgrading code, which means that upgrading directly from very old Pootle versions is no longer possible. In case you are trying to upgrade you must first upgrade to 2.6 before continuing the upgrade process.

- .pending and .tm files support: Not necessary since the `updatetm` tool was removed in Pootle 2.5.0.
- Live translation: Rarely enabled, and its use was actively discouraged.
- Autosync: It was recommended to never use it. The files can be synced using `sync_stores` instead.
- The voting feature for terminology suggestions has also been removed, due to its low popularity and high maintenance cost.
- Removed the zoom feature. Users should use their browsers zooming features.
- Hooks.
- Automatic terminology extraction. It's encouraged to use an external tool to generate any glossaries, then load them up on Pootle.
- Management commands: `update_translation_projects`, `updatedb`, `upgrade`, `setup`, `assign_permissions`.

...and lots of refactoring, upgrades of upstream code, cleanups to remove old Django versions specifics, improved documentation and of course, loads of bugs were fixed.

Credits

This release was made possible by the following people:

Julen Ruiz Aizpuru, Taras Semenenko, Dwayne Bailey, Leandro Regueiro, Igor Afanasyev, Jerome Leclanche, Khaled Hosny, pfennig59, Zahim Anass, Trejkaz (pen name), safaafulaij, Peter Bengtsson, msaad, Mikhail Paulyshka, Miha Vrhovnik, Kevin Scannell, Edmund Huber, Dídac Rios, Andras Timar.

And to all our bug finders, testers and translators, a Very BIG Thank You.

2.6 series

Welcome to the Pootle 2.6.4 Interim

Released on 04 April 2017

The 2.6.4 release is an interim release. It is used to migrate from Pootle 2.5.0 or newer to Pootle 2.7.x or newer releases.

Warning: Do not run a Pootle instance using this version.

Changes in Requirements

- Django `>= 1.6.11, < 1.7`
- `Translate Toolkit` `== 1.13.0`
- Python `>= 2.7, < 3.0`

Major Changes

This release fixes issues that some users experienced upgrading via 2.6.3

Warning: If you are upgrading from Pootle 2.1.0 or older you must first upgrade to 2.1.6 before upgrading to this version.

Warning: If you are upgrading from Pootle older than 2.5.0 you must first upgrade to 2.5.1.3 before upgrading to this version.

Credits

This release was made possible by the following people:

Ryan Northey, Leandro Regueiro

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the Pootle 2.6.3 Interim

Released on 30 March 2017

The 2.6.3 release is an interim release. It is used to migrate from Pootle 2.5.0 or newer to Pootle 2.7.x or newer releases.

Warning: Do not run a Pootle instance using this version.

Changes in Requirements

- Django $\geq 1.6.11$, < 1.7
- [Translate Toolkit](#) $== 1.13.0$
- Python ≥ 2.7 , < 3.0

Major Changes

This release fixes issues that some users experienced upgrading via 2.6.2

Warning: If you are upgrading from Pootle 2.1.0 or older you must first upgrade to 2.1.6 before upgrading to this version.

Warning: If you are upgrading from Pootle older than 2.5.0 you must first upgrade to 2.5.1.3 before upgrading to this version.

Credits

This release was made possible by the following people:

Ryan Northey, Leandro Regueiro

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the Pootle 2.6.2 Interim

Released on 28 September 2015

The 2.6.2 release is an interim release. It is used to migrate from Pootle 2.5.0 or newer to Pootle 2.7.x or newer releases.

Warning: Do not run a Pootle instance using this version.

Major Changes

This release fixes issue [issue 4101](#) that some users experienced upgrading via 2.6.1.

Warning: If you are upgrading from Pootle 2.1.0 or older you must first upgrade to 2.1.6 before upgrading to this version.

Warning: If you are upgrading from Pootle older than 2.5.0 you must first upgrade to 2.5.1.3 before upgrading to this version.

Credits

This release was made possible by the following people:

Ryan Northey, Leandro Regueiro

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the Pootle 2.6.1 Interim

Released on 15 September 2015

The 2.6.1 release is an interim release. It is used to migrate from Pootle 2.5.0 or newer to Pootle 2.7.x or newer releases.

Warning: Do not run a Pootle instance using this version.

Major Changes

This release fixes issues that some users experienced upgrading via 2.6.0

Warning: If you are upgrading from Pootle 2.1.0 or older you must first upgrade to 2.1.6 before upgrading to this version.

Warning: If you are upgrading from Pootle older than 2.5.0 you must first upgrade to 2.5.1.3 before upgrading to this version.

Credits

This release was made possible by the following people:

Ryan Northey, Leandro Regueiro

And to all our bug finders, testers and translators, a Very BIG Thank You.

Welcome to the Pootle 2.6.0 Interim

Released on 29 June 2015

The 2.6.0 release is an interim release. It is used to migrate from Pootle 2.5.0 or newer to Pootle 2.7.0.

Warning: Do not run a Pootle instance using this version.

Changes in Requirements

- Django $\geq 1.6.5 < 1.7$
- Translate Toolkit $\geq 1.12.0$
- Python $\geq 2.6 < 3.0$

Major Changes

Direct upgrade is now only possible from 2.5.0 and later.

We have dropped some legacy upgrade features. The removal of some of these feature means that upgrading directly from ancient Pootle versions is no longer possible.

Warning: If you are upgrading from Pootle 2.1.0 or older you must first upgrade to 2.1.6 before upgrading to this version.

Warning: If you are upgrading from Pootle older than 2.5.0 you must first upgrade to 2.5.1.3 before upgrading to this version.

Credits

This release was made possible by the following people:

Leandro Regueiro, Julen Ruiz Aizpuru, Jerome Leclanche, Igor Afanasyev, Taras Semenenko, Dwayne Bailey, Khaled Hosny, Arky, Peter Bengtsson, , Sebastian Silva, ricordisamoa, Miha Vrhovnik, Kevin KIN-FOO, Henrik Saari, Greg Slepak, Folkert van Heusden, Clement Wong, Alexandre Segura, afan.

And to all our bug finders, testers and translators, a Very BIG Thank You.

2.5 series

Pootle bugfix release 2.5.1.3

Released on 2015-06-03

This is a bugfix release for the 2.5.1 branch. It is meant to provide a newer stable version until Pootle 2.7.0 is released.

Installation and Upgrade

- *Installation*
- *Upgrade*

Bugfixes

For a full list of changes, please check the [git log](#).

- Added support for `xliff` extension for XLIFF files
- Fixed the missing assets issue with the provided package
- Fixed submission of untrusted input from editor
- Fixed upgrading from version 2.5.0
- Fixed notification when saving units
- Assorted documentation updates and fixes

Credits

The following people have made this release possible:

Dwayne Bailey, Leandro Regueiro, Miha Vrhovnik, Kevin KIN-FOO, Julen Ruiz Aizpuru.

Pootle bugfix release 2.5.1.2

Released on 2015-06-01

The 2.5.1.2 release is a bugfix release for the 2.5.1 branch. It is meant to provide a newer stable version until Pootle 2.7.0 is released.

Installation and Upgrade

- *Installation*
- *Upgrade*

Bugfixes

For a full list of changes, please check the [git log](#).

- Added support for `xliff` extension for XLIFF files
- Fixed the missing assets issue with the provided package
- Fixed submission of untrusted input from editor
- Fixed upgrading from version 2.5.0
- Fixed notification when saving units
- Assorted documentation updates and fixes

Credits

The following people have made Pootle 2.5.1.2 possible:

Dwayne Bailey, Leandro Regueiro, Miha Vrhovnik, Kevin KIN-FOO, Julen Ruiz Aizpuru.

Pootle bugfix release 2.5.1.1

Released on 2014-04-29

The 2.5.1.1 release is a bugfix release for the 2.5.1 branch.

Installation and Upgrade

- *Installation*
- *Upgrade*

Bugfixes

For a full list of changes, please check the [git log](#).

- Top stats are now cached for a much longer time and are configurable using `POOTLE_TOP_STATS_CACHE_TIMEOUT`.

- Updated Google Translate support to work with the updated Google Translate API
- Fixed potential failures with zip exports
- Fixed several requirements issues with newer versions of Python and some libraries
- Fixed an obscure crash caused by pagination queries
- Fixed a potential crash when calculating statistics for a submission
- Fixed some javascript issues for users with corrupt cookies
- Assorted documentation updates and fixes

Credits

The following people have made Pootle 2.5.1.1 possible:

Julen Ruiz Aizpuru, Leandro Regueiro, Dwayne Bailey, Khaled Hosny, Jerome Leclanche, Igor Afanasyev and @qd-inar.

Welcome to the new Pootle 2.5.1

Released on 24 January 2014

Yes, we did miss our *6 month release cycle*! Many changes have gone into Pootle 2.5.1 which follows on from 2.5.0 released in May.

Pootle 2.5.1 has been in production for a number of users, so although it is a new official release, we've had many people running their production Pootle server off this code. This includes [Mozilla](#) and [Evernote](#). So you are in good company.

For those who can't wait you might be interested to know what we've got planned on our *roadmap* for Pootle 2.5.2.

Changes in Requirements

- Django >= 1.4.10 (note that Django 1.5 and 1.6 are not yet supported)
- [Translate Toolkit](#) >= 1.11.0
- Python >= 2.6

Installation and Upgrade

- *[Installation](#)*
- *[Upgrade](#)*

Major Changes

These are by no means exhaustive, check the [git log](#) for more details.

- Tags – You can now tag and filter translation projects, making it easy to focus on a set of languages.
- Goals – you can now group files within a project to ensure that translators focus on the most important tasks first.

- Extension Actions – you can create custom actions using Python scripts. These are displayed with current actions and allow you to extend Pootle’s functionality.
- API – an initial Pootle API is in place (disabled by default).

Important server admin changes

- The minimum required Python version is now 2.6.x. While Django 1.4.x supports Python 2.5, it is no longer supported by the Python Foundation neither by several third party apps.
- The database schema upgrade procedure has been redefined:
 - The `updatedb` management command has been phased out in favor of South’s own `migrate` command.
 - Post schema upgrade actions have been moved to the `upgrade` command.
 - The automatic update has been removed.
- The `setup` management command was added to hide the complexities in the altering of the DB when installing or upgrading Pootle.
- Fabric deployment scripts have been improved to make deployment easier.
- Security fixes identified by a Mozilla security audit have been implemented.
- Optimisations of asset caching such as Expires headers have been enabled.
- LDAP authentication backend moved to `pootle.core.auth.ldap_backend.LdapBackend` and received various fixes.
- Static pages can now be used to track the acceptance of terms of use.
- The quality check for spell checking has been globally disabled. It wasn’t properly advertised nor documented, and it didn’t perform well enough to be considered useful.
- `css/custom/custom.css` is now served as part of the common bundle.

Visual Changes

- User contribution are displayed in the users profile page.
- Breadcrumbs now follow the way a translator would interact with Pootle and are unified across all views of the project.
- Global search allows you to search across all projects and all languages.
- Last activity messages show quickly what last change was made to the translations.
- The export view allows for easier proofreading by translators.
- Various RTL fixes.

... and lots of refactoring, upgrades of upstream code, cleanups to remove Django 1.3 specifics, missing documentation and of course, loads of bugs were fixed

Credits

The following people have made Pootle 2.5.1 possible:

Julen Ruiz Aizpuru, Leandro Regueiro, Dwayne Bailey, Alexander Dupuy, Khaled Hosny, Arky, Fabio Pirola, Christian Hitz, Taras Semenenko, Chris Oelmueller, Peter Bengtsson, Yasunori Mahata, Denis Parchenko, Henrik Saari, Hakan Bayindir, Edmund Huber, Dmitry Rozhkov & Darío Hereñú

Welcome to the new Pootle 2.5.0

Released on 18 May 2013

Finally! Translate has a new baby and we're pretty proud of her. Many changes have gone into 2.5.0 which follows on from 2.1.6 released more than two years ago. So many changes that it's quite hard to list them all.

Why so long? Well we had the [Egyptian revolution](#), a complete change in UI, and a load of features we wanted you to have. It took much longer to stabilise it for you to enjoy.

Pootle 2.5.0 has been in production with many users, so although it is a new official release, we've had many people running their production server off this code. This includes [LibreOffice](#), [Mozilla](#) and [Evernote](#). So you are in good company.

Requirements

- Django 1.3 or 1.4
- [Translate Toolkit](#) >= 1.10.0
- lxml (now a runtime requirement)

Installation and Upgrade

- [Installation](#)
- [Upgrade](#)

Changes

These are by no means exhaustive, check the git log for more details

User Experience

We undertook a major UI rework – we now have a clean new translation interface, and overview page.

In the editor:

- We follow a new approach when you edit translations, you will see a list of units that meet some criterion.
- Translation Memory is displayed for the current unit – results are from Translate's public [Amagama](#) server.
- Filters are easily accessible while you translate, so you can quickly change these within the translation interface.
- Context rows are provided in the translation interface when you are filtering and these can be hidden or expanded.
- A timeline is provided for a unit. This provides a history of the changes in translation text, state changes, translator and dates of changes.
- Gravatars give credit to translators and suggesters.

In the overview page:

- The overview page allows you to drill down into certain types of units matching a translation state or with an error.
- It is now easier to see what work needs attentions, as we highlight next actions for your project.
- With editable project and language descriptions you can supply description for projects. These are editable using Markdown, reStructuredText or HTML.
- News alerts can now be sent via email to project participants.
- The overview page provides an expanded checks page that highlights all failing checks.
- Checks are classified into categories so that more urgent ones are highlighted to translators

Version Control

- Update the whole project at once avoiding slow file by file updates
- A separate `VCS_DIRECTORY` for VCS checkout is where Pootle now does all VC related work – this ensures that we can work well with DVCS like Git.
- Detect new and removed files after a VCS update
- Management commands for VCS actions [Stuart Prescott]
- Add new files to VCS after updating from templates

Commands

New and changed commands:

- `list_languages`
- `list_projects`
- `latest_change_id`
- `--modified-since` flag for `update_stores` and `sync_stores`
- `commit_to_vcs`
- `update_from_vcs`

Infrastructure

- All documentation is now on [Read The Docs](#)
- We have a [new website](#) for Pootle
- We're using Travis for [Continuous Integration](#)
- All our [code](#) is now on Github

Other important changes

- Static files are now handled by the `django.contrib.staticfiles` module. This means you will need to run the `pootle collectstatic` command on production and serve the `pootle/assets/` directory from your webserver at `/assets/`. If you are upgrading from a previous version, you will need to replace the occurrences of `static` with `assets` within your web server configuration.

- Static files are bundled into assets by using `django-assets`.
- Several features from translation projects have been merged into the *Overview* tab, including quality check failures and directory- and file-level actions. As a consequence the *Review* tab has been dropped and the *Translate* tab serves solely to display the actual translation editor.
- Settings have been migrated from `localsettings.py` into `settings/*.conf` files. Your customizations now go in a *separate configuration file* (or in `settings/90-local.conf` if running from a repository clone).
- The `PootleServer` script has been phased out in favor of a `pootle` runner script.
- If you will be using Pootle with Django 1.3, you *have* to keep the timezone on UTC, unless you are using PostgreSQL. Users of PostgreSQL or Django 1.4 or later are free to set the time zone as they prefer. Also make sure to use the minimum required South version when performing database upgrades.
- The `update_from_templates` management command has been renamed to `update_against_templates`.

...and of course, loads of bugs were fixed

Credits

The following people have made Pootle 2.5.0 possible:

Julen Ruiz Aizpuru, Friedel Wolff, Alaa Abd el Fattah, Igor Afanasyev, Dwayne Bailey, Leandro Regueiro, Claude Paroz, Chris Oelmueller, Taras Semenenko, Kevin Scannell, Christian Hitz, Thomas Kinnen, Alexander Dupuy, khangaroth, dvinella, Stuart Prescott, Roman Imankulov, Peter Bengtsson, Nagy Akos, Michael Tänzer, Gregory Oschwaldi & Andy Nicholson.

2.1 series

Pootle 2.1.6

Released on 13 April 2011

It's been 3 months since our last bug fix releases, it's about time we give you [Pootle 2.1.6](#).

Pootle is a web based system for translation and translation management.

Main focus of the release is incompatibility issues with the latest versions of Django (1.2.5 and 1.3.0).

Apart from that, version 2.1.6 has a handful of fixes. Here are the highlights:

- Fixed another bug with GNU style projects language detection.
- Added a separate project type for UTF-8 encoded Java properties.
- Fixed a bug that would under rare conditions hide some strings from translate page.
- Fixed a bug that caused some translation project level statistics to be miscalculated.
- Fix for Qt TS format based on changes in Translate Toolkit 1.9.0

On the first visit after upgrading upgrade screen will flash for a short period while translation statistics are recalculated, if running under [Translate Toolkit](#) version 1.9.0 it might last longer as Qt TS files will be reparsed to benefit from improvements to the format support.

Django 1.2.5 and 1.3.0 compatibility depends on Translate Toolkit version 1.9.0 or above but all users are encouraged to upgrade their versions of Translate Toolkit. As always Pootle will benefit from fixes and performance improvements in the latest versions.

- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)
- [More information](#)

Pootle 2.1.5 released

Released on 18 Jan 2011

A quick bug fix release to celebrate the new Year. Please welcome [Pootle 2.1.5](#)!

Pootle is a web based system for translation and translation management.

This release fixes a couple of regressions introduced in the previous 2.1.4 release. Including a build mistake where the files in the 2.1.4 tarball had very restrictive permissions.

Apart from that, version 2.1.5 has a handful of fixes. Here are the highlights:

- Fix regression causing update from templates to fail for GNU Style projects with subdirectories.
- Fix regression in handling obsolete units while committing to version control (reported by Mozilla).
- Clean stale file locks left in cases of external kills which running expensive commands.
- Fix security bug where project names would leak to users without view access on the server via news summary on front page or profile edit form.
- Fix a bug that prevented Project level permissions from overriding very restrictive server wide permissions.

As always Pootle will benefit from fixes and performance improvements in the latest versions of Translate Toolkit.

- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)
- [More information](#)

Enjoy it, The Translate Team

Pootle 2.1.4 Released

Released on 17 Dec 2010

We thought we'd wrap up the year with one more bug fix release, Please welcome [Pootle 2.1.4](#)

Pootle is a web based system for translation and translation management.

This release fixes a nasty bug where quality checks failed to update on file uploads. the upgrade screen will flash on first visit after upgrade for a minute or two to correct this problem (might take longer if you used the quality checks feature extensively).

Apart from that, version 2.1.4 has a handful of fixes. Here are the highlights:

- Once and for all Qt ts plurals should now work correctly.
- Fixed a bug where obsolete units could not be updated when uploading a new version of the file.

- Fixed a bug that affected some GNU/Linux systems causing server errors when using Turkish Locale.
- Fixed a bug in GNU style projects with a prefix where pt_BR would be detected as Breton instead of Brazilian Portuguese

As always Pootle will benefit from fixes and performance improvements in the latest versions of Translate Toolkit.

- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)
- [More information](#)

Pootle 2.1.3 released

Released on 26 Nov 2010

It's been less than three weeks since the we released Pootle 2.1.2 but we've fixed a couple of critical bugs affecting many users so it's time for another bug fix release. Please welcome [Pootle 2.1.3](#)

Pootle is a web based system for translation and translation management.

This release includes a fix to a data loss bug, where recent translations are lost when updating from version control. Users who depend on version control support are encouraged to upgrade immediately.

We've added support for CSV format. This will hopefully make it easier for less technical users to get their strings inside Pootle by exporting from spreadsheet or similar office software. But it should not be treated as a replacement for more solid formats like PO, Qt ts or XLIFF.

By popular demand we've improved Java properties support to accept properties files in any encoding. including UTF-8.

Improved format support depends on the recently release Translate Toolkit 1.8.1

We also bring you translations for Chiga and Latvian.

Apart from that, version 2.1.2 has many bug fixes. Here are the highlights:

- Fix for database migration failing for some users
- Fix for errors on upgrades for users who deleted the English language
- Fix for errors on filenames with spaces and memcached
- Many fixes to language detection in GNU Style projects
- Various fixes to handling of escaped characters in translate page

As always Pootle will benefit from fixes in any the latest versions of Translate Toolkit, the recently released 1.8.1 includes many fixes specifically for Pootle 2.1.3 so upgrading translate toolkit is highly recommended.

- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)
- [More information](#)

Pootle 2.1.2 Released including security fix

Released on 15 Nov 2010

<https://sourceforge.net/projects/translate/files/Pootle/2.1.2/Pootle-2.1.2.tar.bz2>

This release includes an important security fix to a cross site scripting vulnerability in the translate page. All users are encouraged to upgrade immediately.

The release also includes many improvements to the support of monolingual translation formats (like subtitles files and Java properties) and to “GNU style” projects.

We also bring you translations for five new language (Zulu, Greek, Danish, Acoli and Fulah) and six more translations are now 100% complete (Uighur, Chinese (China), Catalan, Asturian, Akan and Ganda).

Highlighted fixed and improvements:

- Fixed a PostgreSQL incompatibility bug.
- Fixed a regression where plural units in Qt ts were not parsed correctly.
- A new manage.py command `update_translation_projects` allows for detecting new languages added to projects on the file system.
- More flexible options to all manage.py commands allowing users to limit commands to a set of projects and languages.
- Pootle now supports GNU Style projects where filenames have a prefix preceding language codes.
- Pootle will ignore case differences when matching filenames to language codes.
- Improvements to fuzzy matching when updating monolingual projects from templates.
- Pootle will no longer modify templates files, translations to these files will be stored in database only to avoid propagating these translations on update from templates.
- Users with administer permissions on a language or project now have all the other rights implied automatically for that language or project.
- Users with only suggest right will be able to upload files using the “suggest only” merge method.
- URLs in developer comments are now displayed as links.
- Fixed bug that caused unnecessary diffs to PO files tracked in version control.
- Local terminology no longer blocks suggestions from the server-wide terminology project.
- Pootle is now less fascistic about what language codes should look like, but users should try to stick to GNU locale names when possible.
- Removed confusing initialize checkbox from Project admin page. No one knew what it was for, those who do can uncomment a single line of code to bring it back.

Pootle 2.1.1 depends on at least version 1.8.0 of Translate Toolkit, and as always will benefit from fixes in any later versions. so always use the latest.

This work was made possible by many volunteers and our funders:

- ANLoc, funded by IDRC <http://africanlocalisation.net/>
- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)

- [More information](#)

Pootle 2.1.1 Released including security fix

Released on 03 September 2010

With the coming of spring we thought it's a good time to make the first bug fix release of the exciting new Pootle. We bring you Pootle 2.1.1 get it while it is blooming from <https://sourceforge.net/projects/translate/files/Pootle/2.1.1/>

Pootle is a web based system for translation and translation management.

This release finally brings the ability to migrate data between different database engines. This means all of you stuck with the default sqlite3 can now move to a database engine that scales better like MySQL or PostgreSQL.

Note that database migration depends on Django 1.2 or later.

As an added bonus we added database migration to the 2.0 branch and quietly slipped in the last bug fix release for that series <https://sourceforge.net/projects/translate/files/Pootle/2.0.6/> we made this bonus release so users still on the 2.0 branch using sqlite can migrate databases before they upgrade to 2.1 since the upgrade process is slow and the database size under 2.1 is considerably larger.

For instructions and more details check [Database migration](#) docs.

We noticed some users running Pootle under apache fail to use memcached for caching and stick to the default local memory cache backend. This causes buggy behavior as the default is not compatible with multiprocess servers. So for 2.1.1 we changed the default to a database cache backend. We still recommend using memcached but if for any reason you can't please update your localsettings.py.

Users upgrading from 2.1.0 will see the upgrade screen appear for a few seconds while Pootle prepares the database for the new cache backend.

For more information check [Caching System](#) docs.

Apart from these two major changes 2.1.1 includes four new translations (Slovenian, Songhai, Tamil and Faroese) and many fixes and performance improvements. Here are the highlights:

- Translation progress tables now show icons to indicate ability to change table sorting.
- Apertium machine translation improved their Javascript APIs with the help of our Julien, Pootle has been updated to use these new apis which make apertium a much more attractive option (specially for translation between European languages).
- Pootle no longer attempts to save translations to disk when there are no new translations. Speeds up downloads.
- Pootle now keeps a cached copy of exported ZIP archives and XLIFF files to improve performance.
- Correct From header for emails sent by contact form.
- Fixed a bug where Pootle kept files open even when not needed. May make us more Windows friendly (but no promises).
- Better handling of invalid file types on upload.
- Expensive serverwide stats on admin dashboard are not calculated on demand only. Should make admin page loading more snappy.
- Don't accept empty suggestions.
- Thanks to Terin Stock (terinjokes) it is now possible to send registration email as HTML emails.

Pootle 2.1.1 depends on at least version 1.8.0 of Translate Toolkit, and as always will benefit from fixes in any later versions. So always use the latest.

This work was made possible by many volunteers and our funders:

- ANLoc, funded by IDRC <http://africanlocalisation.net/>
- [Feature list](#)
- [Download](#)
- [Installation instructions](#)
- [Bugs](#)
- [More information](#)

Older releases

Older releases

Older release more for your entertainment and to track Pootle's history.

Version 2.1

Released on August 17th 2010.

- Pootle no longer depends on statsdb and SQLite.
- Files on disk are only synced with the database on download or commit. The old behaviour can be restored at the cost of performance. A `manage.py command` can sync to files on the command line.
- The database is now much larger. This should have no negative impact on performance, but we strongly suggest using MySQL or PostgreSQL for the best performance.
- Pootle 2.1 will upgrade the database automatically from Pootle 2.0 installations. You need to have South installed. Install it from your distribution, or <http://south.aeracode.org/> or with `easy_install South` (the upgrade could take quite a while, depending on your installation size).
- Pending files are not used for suggestions any more, and will also be migrated to the database during upgrade.
- New settings are available in `localsettings.py` – compare your existing one to the new one.
- Pootle 1 installations can easily migrate everything excluding project permissions. We encourage administrators to configure permissions with the new permission system which is much simpler to use, since permissions on the language and project level are now supported.
- Have a look at the optimization guide to ensure your Pootle runs well.

Version 2.0

Released on December 7th 2009.

- Pootle now uses the Django framework and data that previously was stored in flat files (projects, languages, users and permissions) is now stored in a database. Migration scripts are provided.
- Review all suggestions before migrating, and note that assignments are not yet supported in Pootle 2.0.

Version 1.2.0

Released on October 8th 2008.

- The name of the directory for indexing databases changed from *.poinde-PROJECT-LANGUAGE* to *.translation_index*. Administrators may want to remove the old indexing directories manually.
- The enhanced search function needs all indexing databases to be regenerated, otherwise it won't find anything. To achieve this, just remove all *.translation_index* directories under your projects:

```
find /path/to/projects/ -type d -name ".translation_index" -exec rm -rf {} \;
```

- If you used testing versions of Pootle 1.2, you almost definitely need to regenerate your statistics database. Pootle might be able to do it automatically, but if not, delete *~/translate_toolkit/stats.db*.

Version 1.0

Released on May 25th 2007.

XLIFF support Pootle 1.0 is the first version with support for XLIFF based projects. In the admin interface the project type can be specified as PO / XLIFF (this really just tells Pootle for which type of files it should look - it won't convert your project for you). This property is stored in *pootle.prefs* in the variable *localfiletype* for each project.

Configurable logos You are now able to configure the logos to use in *pootle.prefs*. At the moment it will probably be easiest to ensure that the same image sizes are used as the standard images.

Localized language names Users can now feel more at home with language names being localized. This functionality is actually provided by the toolkit and your system's iso-codes package.

Treestyle: gnu vs nongnu Pootle automatically detects the file layout of each project. If you want to eliminate the detection process (which can be a bit slow for big projects) or want to override the type that Pootle detected, you can specify the *treestyle* attribute for the project in *pootle.prefs*. Currently this can not be specified through the admin interface.

Version 0.11

Released on March 8th 2007.

- If the user has the appropriate privileges (overwrite right) he/she will be able to upload a file and completely overwrite the previous one. Obviously this should be done with care, but was a requested feature for people that want to entirely replace existing files on a Pootle server.
- The server administrator can now specify the default access rights (permissions) for the server. This is the rights that will be used for all projects where no other setup has been given. See *pootle.prefs* for some examples.
- The default rights in the default Pootle setup has changed to only allow suggesting and to not allow translation. This means that the default server setup is not configured to allow translation, and that users must be specifically assigned the translate (and optionally review) right, or alternatively, the default rights must be configured to allow translation (see the paragraph above).
- The *baseurl* will now be used, except for the */doc/* directory, that currently still is offered at */doc/*.
- The default installation now uses English language names in preparation for future versions that will hopefully have language names translated into the user interface language. To this end the language names must be in English, and names with country codes must have the country code in simple noun form in brackets. For example *Portuguese (Brazil)*; in other words, not *Portuguese (Brazilian)*.

Version 0.10

Released on August 29th 2006.

Statistics The statistics pages are greatly reworked. We now have a page that shows a nice table, that you can sort, with graphs of the completeness of the files. This is the default view. What is confusing is that the stats page does not work directly with editing. To get the editing features, click on the editing link in the top bar.

The quick statistics files (*pootle-projectname-zu.stats*) now also store the fuzzy stats that are needed to render the statistics tables. Your previous files from 0.9 can not supply this information. Pootle 0.10 will automatically update these files, but if you (for some reason) want/need to go back to Pootle 0.9, you will have to delete these files. Not all *.stats* files need to be deleted, only the ones starting with *pootle-projectname*.

SVN and CVS committing You can now commit to SVN or CVS. A default commit message is added, you cannot edit this message. Your ability to commit depends on the rights you have on the checkout and since you cannot supply a password it needs to be a non-blocking method. This feature is probably not useful for a very public server unless it is managing multiple translations of your own project and you have direct control over it and CVS/SVN accounts. It will work well in a standalone situation like a [Translate@thon](#) etc, where it is a public event but the server is controlled by yourself for the event and then you can simply commit changes at the end. For more information, see version control information.

Terminology Pootle can now aid translators with terminology. Terminology can be specified to be global per language, and can be overridden per project for each language. A project called “terminology” (with any full name) can contain any files that will be used for terminology matching. Alternatively a file with the name *pootle-terminology.po* can be put in the directory of the project, in which case the global one (in the terminology project) will not be used. Matching is done in real time. Note that this does not work with GNU-style projects (where all the files are in one directory and have names according to the language code).

Translation Memory Pootle can now aid translators by means of a translation memory. The suggestions are not generated realtime – it is done on the server by means of a commandline program (*updatetm*). Files with an appended *.tm* will be generated and read by Pootle to supply the suggestions. For more information see *updatetm*.

2.2 License

The Pootle code and documentation is released under the [GNU General Public License \(GPL\)](#), version 3 or later.

Symbols

- all
 - flush_cache command line option, 72
 - verify_user command line option, 88
- atomic
 - command line option, 70
- check
 - calculate_checks command line option, 71
 - test_checks command line option, 77
- config
 - init command line option, 81
- data
 - dump command line option, 77
- db
 - init command line option, 81
- db-host
 - init command line option, 81
- db-name
 - init command line option, 81
- db-port
 - init command line option, 81
- db-user
 - init command line option, 81
- dev
 - init command line option, 82
 - webpack command line option, 83
- display-name
 - update_tmserver command line option, 79
- django-cache
 - flush_cache command line option, 72
- dry-run
 - update_tmserver command line option, 79
- force
 - add command line option, 84
 - fetch command line option, 84
 - rm command line option, 85
 - sync_stores command line option, 73
 - update_stores command line option, 74
- from-filetype
 - set_filetype command line option, 76
- include-anonymous
 - contributors command line option, 75
- include-disabled-projects
 - update_tmserver command line option, 78
- lru
 - flush_cache command line option, 72
- mailmerge
 - contributors command line option, 75
- matching
 - set_filetype command line option, 76
- modified-since
 - list_languages command line option, 74
 - list_projects command line option, 75
- no-delete
 - merge_user command line option, 87
- no-projects
 - initdb command line option, 82
- no-rq
 - command line option, 70
- noinput
 - command line option, 70
- overwrite
 - resolve command line option, 85
 - sync_stores command line option, 73
 - update_stores command line option, 74
- pootle-wins
 - resolve command line option, 85
- project, –language
 - command line option, 70
- rebuild
 - update_tmserver command line option, 78
- refresh
 - update_tmserver command line option, 78
- reset
 - refresh_scores command line option, 72
- restore
 - revision command line option, 76
- rotate
 - export command line option, 80

- rqdata
 - flush_cache command line option, 72
- since
 - contributors command line option, 75
- skip-missing
 - sync_stores command line option, 73
- sort-by
 - contributors command line option, 75
- source, -target
 - test_checks command line option, 76
- stats
 - dump command line option, 77
- store
 - update_data command line option, 71
- target-language
 - update_tmserver command line option, 79
- tm
 - update_tmserver command line option, 78
- tmx
 - export command line option, 80
- unit
 - test_checks command line option, 76
- until
 - contributors command line option, 75
- user
 - import command line option, 81
- yes
 - init command line option, 82
- P -pootle-path
 - fs command line option, 84
- a <key> <value>, -append <key> <value>
 - config command line option, 78
- c <key>, -clear <key>
 - config command line option, 78
- d, -deserializers
 - list_serializers command line option, 74
- g <key>, -get <key>
 - config command line option, 77
- j, -json
 - config command line option, 78
- l <key>, -list <key>
 - config command line option, 78
- m, -model
 - list_serializers command line option, 74
- o <field>, -object-field <field>
 - config command line option, 77
- p -fs-path
 - fs command line option, 83
- s <key> <value>, -set <key> <value>
 - config command line option, 78
- t -type
 - state command line option, 86

A

- add
 - django-admin command, 84
- add command line option
 - force, 84
- add_vfolders
 - django-admin command, 79
- AMAGAMA_SOURCE_LANGUAGES
 - setting, 62
- AMAGAMA_URL
 - setting, 62

C

- calculate_checks
 - django-admin command, 71
- calculate_checks command line option
 - check, 71
- clear_stats
 - django-admin command, 88
- command line option
 - atomic, 70
 - no-rq, 70
 - noinput, 70
 - project, -language, 70
- commit_to_vcs
 - django-admin command, 88
- config
 - django-admin command, 77
- config command line option
 - a <key> <value>, -append <key> <value>, 78
 - c <key>, -clear <key>, 78
 - g <key>, -get <key>, 77
 - j, -json, 78
 - l <key>, -list <key>, 78
 - o <field>, -object-field <field>, 77
 - s <key> <value>, -set <key> <value>, 78
 - content_type, 77
 - object, 77
- content_type
 - config command line option, 77
- contributors
 - django-admin command, 75
- contributors command line option
 - include-anonymous, 75
 - mailmerge, 75
 - since, 75
 - sort-by, 75
 - until, 75
- CONTRIBUTORS_EXCLUDED_NAMES
 - setting, 64
- CONTRIBUTORS_EXCLUDED_PROJECT_NAMES
 - setting, 64

D

DESCRIPTION

setting, 64

django-admin command

add, 84

add_vfolders, 79

calculate_checks, 71

clear_stats, 88

commit_to_vcs, 88

config, 77

contributors, 75

dump, 77

export, 80

fetch, 84

find_duplicate_emails, 86

flush_cache, 72

fs, 83

import, 80

info, 84

init, 81

initdb, 82

last_change_id, 88

list_languages, 74

list_projects, 74

list_serializers, 74

merge_user, 86

purge_user, 87

refresh_scores, 72

refresh_stats, 88

resolve, 85

retry_failed_jobs, 71

revision, 76

rm, 85

run_cherrypy, 89

schema, 78

set_filetype, 75

start, 89

state, 85

sync, 86

sync_stores, 72

test_checks, 76

unstage, 86

update_data, 71

update_from_vcs, 89

update_stores, 73

update_tmserver, 78

update_user_email, 87

verify_user, 87

webpack, 82

dump

django-admin command, 77

dump command line option

-data, 77

-stats, 77

E

ENABLE_ALT_SRC

setting, 63

environment variable

POOTLE_SETTINGS, 58

PYTHONPATH, 89

export

django-admin command, 80

export command line option

-rotate, 80

-tmx, 80

EXPORTED_DIRECTORY_MODE

setting, 64

F

fetch

django-admin command, 84

fetch command line option

-force, 84

find_duplicate_emails

django-admin command, 86

flush_cache

django-admin command, 72

flush_cache command line option

-all, 72

-django-cache, 72

-lru, 72

-rqdata, 72

fs

django-admin command, 83

fs command line option

-P -pootle-path, 84

-p -fs-path, 83

FUZZY_MATCH_MAX_LENGTH

setting, 64

FUZZY_MATCH_MIN_SIMILARITY

setting, 64

I

import

django-admin command, 80

import command line option

-user, 81

info

django-admin command, 84

init

django-admin command, 81

init command line option

-config, 81

-db, 81

-db-host, 81

-db-name, 81

-db-port, 81

- db-user, 81
- dev, 82
- yes, 82
- The configuration file to write to., 81
- initdb
 - django-admin command, 82
- initdb command line option
 - no-projects, 82

L

- last_change_id
 - django-admin command, 88
- list_languages
 - django-admin command, 74
- list_languages command line option
 - modified-since, 74
- list_projects
 - django-admin command, 74
- list_projects command line option
 - modified-since, 75
- list_serializers
 - django-admin command, 74
- list_serializers command line option
 - d, -deserializers, 74
 - m, -model, 74

M

- MAX_AUTOTERMS
 - setting, 64
- merge_user
 - django-admin command, 86
- merge_user command line option
 - no-delete, 87
- MIN_AUTOTERMS
 - setting, 64

O

- object
 - config command line option, 77

P

- PARSE_POOL_CULL_FREQUENCY
 - setting, 63
- PARSE_POOL_SIZE
 - setting, 63
- POOTLE_CACHE_TIMEOUT
 - setting, 58
- POOTLE_CANONICAL_URL
 - setting, 59
- POOTLE_CAPTCHA_ENABLED
 - setting, 60
- POOTLE_CONTACT_EMAIL
 - setting, 59

- POOTLE_CONTACT_ENABLED
 - setting, 59
- POOTLE_CONTACT_REPORT_EMAIL
 - setting, 59
- POOTLE_CUSTOM_LOGO
 - setting, 59
- POOTLE_CUSTOM_TEMPLATE_CONTEXT
 - setting, 60
- POOTLE_EMAIL_FEEDBACK_ENABLED
 - setting, 59
- POOTLE_FAVICONS_PATH
 - setting, 59
- POOTLE_FS_WORKING_PATH
 - setting, 61
- POOTLE_INSTANCE_ID
 - setting, 58
- POOTLE_LEGALPAGE_NOCHECK_PREFIXES
 - setting, 60
- POOTLE_LOG_DIRECTORY
 - setting, 59
- POOTLE_MARKUP_FILTER
 - setting, 60
- POOTLE_META_USERS
 - setting, 60
- POOTLE_MT_BACKENDS
 - setting, 63
- POOTLE_QUALITY_CHECKER
 - setting, 64
- POOTLE_REPORTS_MARK_FUNC
 - setting, 61
- POOTLE_SCORE_COEFFICIENTS
 - setting, 64
- POOTLE_SCORES
 - setting, 61
- POOTLE_SETTINGS, 58
- POOTLE_SIGNUP_ENABLED
 - setting, 60
- POOTLE_SQL_MIGRATIONS
 - setting, 58
- POOTLE_SYNC_FILE_MODE
 - setting, 62
- POOTLE_TITLE
 - setting, 58
- POOTLE_TM_SERVER
 - setting, 62
- POOTLE_TM_SERVER-INDEX_NAME
 - setting, 63
- POOTLE_TM_SERVER-MIN_SIMILARITY
 - setting, 63
- POOTLE_TM_SERVER-WEIGHT
 - setting, 63
- POOTLE_TOP_STATS_CACHE_TIMEOUT
 - setting, 63
- POOTLE_TRANSLATION_DIRECTORY

setting, 63
 POOTLE_WORDCOUNT_FUNC
 setting, 63
 purge_user
 django-admin command, 87
 PYTHONPATH, 89

R

refresh_scores
 django-admin command, 72
 refresh_scores command line option
 –reset, 72
 refresh_stats
 django-admin command, 88
 resolve
 django-admin command, 85
 resolve command line option
 –overwrite, 85
 –pootle-wins, 85
 retry_failed_jobs
 django-admin command, 71
 revision
 django-admin command, 76
 revision command line option
 –restore, 76
 rm
 django-admin command, 85
 rm command line option
 –force, 85
 run_cherrypy
 django-admin command, 89

S

schema
 django-admin command, 78
 set_filetype
 django-admin command, 75
 set_filetype command line option
 –from-filetype, 76
 –matching, 76
 setting
 AMAGAMA_SOURCE_LANGUAGES, 62
 AMAGAMA_URL, 62
 CONTRIBUTORS_EXCLUDED_NAMES, 64
 CONTRIBUTORS_EXCLUDED_PROJECT_NAMES, 64
 DESCRIPTION, 64
 ENABLE_ALT_SRC, 63
 EXPORTED_DIRECTORY_MODE, 64
 FUZZY_MATCH_MAX_LENGTH, 64
 FUZZY_MATCH_MIN_SIMILARITY, 64
 MAX_AUTOTERMS, 64
 MIN_AUTOTERMS, 64
 PARSE_POOL_CULL_FREQUENCY, 63

 PARSE_POOL_SIZE, 63
 POOTLE_CACHE_TIMEOUT, 58
 POOTLE_CANONICAL_URL, 59
 POOTLE_CAPTCHA_ENABLED, 60
 POOTLE_CONTACT_EMAIL, 59
 POOTLE_CONTACT_ENABLED, 59
 POOTLE_CONTACT_REPORT_EMAIL, 59
 POOTLE_CUSTOM_LOGO, 59
 POOTLE_CUSTOM_TEMPLATE_CONTEXT, 60
 POOTLE_EMAIL_FEEDBACK_ENABLED, 59
 POOTLE_FAVICONS_PATH, 59
 POOTLE_FS_WORKING_PATH, 61
 POOTLE_INSTANCE_ID, 58
 POOTLE_LEGALPAGE_NOCHECK_PREFIXES, 60
 POOTLE_LOG_DIRECTORY, 59
 POOTLE_MARKUP_FILTER, 60
 POOTLE_META_USERS, 60
 POOTLE_MT_BACKENDS, 63
 POOTLE_QUALITY_CHECKER, 64
 POOTLE_REPORTS_MARK_FUNC, 61
 POOTLE_SCORE_COEFFICIENTS, 64
 POOTLE_SCORES, 61
 POOTLE_SIGNUP_ENABLED, 60
 POOTLE_SQL_MIGRATIONS, 58
 POOTLE_SYNC_FILE_MODE, 62
 POOTLE_TITLE, 58
 POOTLE_TM_SERVER, 62
 POOTLE_TM_SERVER-INDEX_NAME, 63
 POOTLE_TM_SERVER-MIN_SIMILARITY, 63
 POOTLE_TM_SERVER-WEIGHT, 63
 POOTLE_TOP_STATS_CACHE_TIMEOUT, 63
 POOTLE_TRANSLATION_DIRECTORY, 63
 POOTLE_WORDCOUNT_FUNC, 63
 VCS_DIRECTORY, 63

start
 django-admin command, 89
 state
 django-admin command, 85
 state command line option
 –t –type, 86
 sync
 django-admin command, 86
 sync_stores
 django-admin command, 72
 sync_stores command line option
 –force, 73
 –overwrite, 73
 –skip-missing, 73

T

test_checks
 django-admin command, 76
 test_checks command line option

- check, [77](#)
- source, –target, [76](#)
- unit, [76](#)

The configuration file to write to.
init command line option, [81](#)

U

unstage

- django-admin command, [86](#)

update_data

- django-admin command, [71](#)

update_data command line option

- store, [71](#)

update_from_vcs

- django-admin command, [89](#)

update_stores

- django-admin command, [73](#)

update_stores command line option

- force, [74](#)

- overwrite, [74](#)

update_tmserver

- django-admin command, [78](#)

update_tmserver command line option

- display-name, [79](#)

- dry-run, [79](#)

- include-disabled-projects, [78](#)

- rebuild, [78](#)

- refresh, [78](#)

- target-language, [79](#)

- tm, [78](#)

update_user_email

- django-admin command, [87](#)

V

VCS_DIRECTORY

- setting, [63](#)

verify_user

- django-admin command, [87](#)

verify_user command line option

- all, [88](#)

W

webpack

- django-admin command, [82](#)

webpack command line option

- dev, [83](#)