
Pootle Documentation

Release 2.5.1.3

Translate.org.za

September 16, 2015

1	All you need to know	3
1.1	User's guide	3
1.2	Features	5
1.3	Administering a server	28
1.4	Developers	66
1.5	Pootle API	90
2	Additional Notes	113
2.1	Changelog	113
2.2	Release Notes	115
2.3	External documentation	129
2.4	License	130

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, including user's, server administrator's, and developer's view.

1.1 User's guide

1.1.1 Getting started

Pootle is a web portal that allows you to translate more easily. The name stands for PO-based Online Translation / Localization Engine, but you may need to read [this](#). Pootle is GPL licensed Free Software, and you can download it and run your own copy if you like.

You can also help participate in the development in many ways (you don't have to be able to program).

The Pootle project itself is hosted at <http://pootle.translatehouse.org> where you can find details about source code, mailing lists, etc.

Registration

While everybody can view the files, only registered users can edit them and receive credit for their effort, so unless your server uses LDAP authentication, the first thing you should do in order to translate, is to register.

You can register in the register page (accessible by clicking *Register* in the menubar) following two simple steps, providing you have a current e-mail address.

1. Fill in your desired user name, a valid e-mail account, and enter your password twice for verification. Then choose *Register* and an you will receive a message with an activation link in the e-mail address you have provided.
2. When you receive the activation message by e-mail, just click on the activation link and your account will be activated.

Login and user settings setup

Now that you are a registered user, you can login to Pootle by following the *Login* link on the top menubar and filling in your credentials.

Once you have logged in, your account's dashboard will be shown, which includes links to your selected languages and projects.

The first time you log in, no links will be shown since you haven't chosen any before. You can click on the link provided to change your settings and select your preferred languages and projects. You can set more settings within the same page as well.

Browsing the files tree

Now that your dashboard is set up, you can reach the desired files for translation directly through the links in it.

Another way to find the file you wish to translate is through the main page. The main page displays two categories: languages and projects. Choosing a language will give you the list of projects available for translation into this language; choosing a project will give you the list of languages to which it can be translated.

Once you have chosen both the project and the language, you'll be presented with the files and directories available for translation.

Heading to translating

If you click on a filename it will start showing all the entries on the file, independently whether the entries are translated or not. This is also known as *Translate All*.

Alternatively, if the translation for a file is not complete, you can click on the summary text (eg, *27 words need attention*), which will give you through all the untranslated or fuzzy entries on the file. This mode is also named as *Quick Translate*.

In both cases you'll be presented with a two-column table, with the strings to be translated on the left, and the current translation on the right.

The current edited entry will appear as a text box with the options *Skip* and *Suggest* or *Submit* below it. Naturally you can enter text in the text box and submit it or skip to the next entry.

You can also directly access any of the other entries presented by clicking on the numbers on the left-hand side.

Other aids from Pootle

Pootle's editor also helps translators by displaying *Terminology* related to the entry currently being edited.

Another helpful feature is the *Alternative source language*, which displays how the current entry has been translated into other languages. In order for this to work, you must select your desired alternative source languages in your account's settings.

In addition to the above, you can also download the translation file, *work offline* with your favourite editor and upload the file. For multi-file projects, you can download a ZIP file with all the files for a directory and upload the ZIP file with the translated files.

1.1.2 Administration

In default Pootle installations, an *admin* account (the password matches the username) is created with superuser privileges which can be used to administer the whole site.

Note: It's highly recommended that you change the password for the default *admin* account on your first login, or even delete the account and assign superuser rights to another user.

Administration scope

Users with administration privileges will have an extra *Admin* element within the main navigation bar, which gives direct access to the administration functions of the site.

Administrators can change the general settings for the site (such as the server title or description), as well as add, edit, and remove users, languages, and translation projects. They are also able to define the default permissions that will apply to the whole site unless otherwise noted.

Apart from that, administrators have full rights over all the translation projects, so they can do whatever can be done: translate, suggest, upload new files, update VCS checkouts, ...

Adding new site administrators

If you want to assign site administration permissions to an already existing user, just go to the *Users* tab within the *Administration* page, check the *Superuser status* checkbox for the user you want to make administrator and click on *Save Changes*. That's all!

Language administrators

Administration rights can be delegated for all tasks within a language. This allows the language administrator to set permissions in the language, set permissions in certain projects in the language, manage files, etc.

To appoint a language administrator, visit the language page, and add the administration right on the *Permissions* tab.

1.2 Features

1.2.1 Backends and storage

Authentication Backends

LDAP Authentication

LDAP configuration can be enabled by appending the `'pootle.core.auth.ldap_backend.LdapBackend'` to the list of `AUTHENTICATION_BACKENDS`. The settings page lists all the *configuration keys available for LDAP*.

Below a brief example of a working configuration is showcased.

The mail addresses are *john.doe@website.org*, the LDAP server is *your.ldapserver.org*. In this case, we need a specific user account to search in our LDAP server, this user/password is *admin/pootle*. The LDAP accounts are based on the mail addresses: these are the uids. Finally, *John Doe* is part of the branch *employees* on the LDAP.

```
# Authenticate first with an LDAP system and then fall back to Django's
# authentication system.
AUTHENTICATION_BACKENDS = [
    #: Uncomment the following line for enabling LDAP authentication
    'pootle.core.auth.ldap_backend.LdapBackend',
    'django.contrib.auth.backends.ModelBackend',
]

# The LDAP server. Format: protocol://hostname:port
AUTH_LDAP_SERVER = 'ldap://your.ldapserver.org:389'
# Anonymous Credentials : if you don't have a super user, don't put cn=...
AUTH_LDAP_ANON_DN = 'cn=admin,dc=website,dc=org'
```

```
AUTH_LDAP_ANON_PASS = 'pootle'
# Base DN to search
AUTH_LDAP_BASE_DN = 'ou=employees,dc=website,dc=org'
# What are we filtering on? %s will be the username (must be in the string)
# In this case, we filter on mails, which are the uid.
AUTH_LDAP_FILTER = 'uid=%s'

# This is a mapping of Pootle field names to LDAP fields. The key is
# Pootle's name, the value should be your LDAP field name. If you don't use the
# field or don't want to automatically retrieve these fields from LDAP comment
# them out. The only required field is 'dn'. givenName, sn and uid are the names
# of the LDAP fields.
AUTH_LDAP_FIELDS = {
    'dn': 'dn',
    'first_name': 'givenName',
    'last_name': 'sn',
    'email': 'uid'
}
```

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](#)

New in version 2.0.3.

- [Qt TS](#)
- [TBX](#)
- [TMX](#)

Monolingual formats

New in version 2.1.

These files contain only one language in the file. Pootle supports formats without conversion.

- [Java properties](#)
- [Mac OSX strings](#)
- [PHP arrays](#)
- [Subtitles](#) in many formats

Monolingual files need special attention in order to provide translators with good workflow and to assist to perform good translation. Read more in the localization guide.

The main difference between monolingual and bilingual projects in Pootle is that for monolingual projects a translation template is required. Pootle cannot meaningfully import strings from monolingual files unless the original text is present.

Either the source language or the *special Templates language* must be added to the project and their files uploaded before other languages are added. Files found in either will be considered template files (in the case where both templates and source language exist templates will be used).

What users will see when translating monolingual file is a matching between strings in the templates file and strings in the target language files. The matching is format specific (for example in subtitles the matching is based on timestamps, for Java properties it is based on keys, etc.)

While Pootle supports uploading translations in the monolingual format this should be limited to importing old translations. Users who want to translate offline should download the XLIFF version.

When tracking monolingual files with version control, if the file structure changes (e.g. new strings are added) then source files must be updated first.

Apart from these considerations monolingual projects will feel and behave the same as bilingual projects, all of Pootle's features are available to administrators and translators.

You can still use the format converters from the Translate Toolkit to host these monolingual file formats as a Gettext PO project. This has the advantage that files in version control always have the source and target strings together and you are able to integrate with external PO tools.

Hooks

Pootle supports hooks to customize its behavior at various points in its interaction with [Version Control Systems](#), translation update, and translation initialization.

Hooks are Python scripts and can do things like checking or converting formats before commit.

Note: Changed in version 2.5.

Because VCS checkouts and commits are performed in a separate *VCS_DIRECTORY*, hooks performing VCS operations themselves may need to use functions in `apps.pootle_misc.versioncontrol` to copy files between that directory and the *PO_DIRECTORY* containing translation files. As part of this change, pathnames passed to hooks are relative to *PO_DIRECTORY*, not absolute paths.

Implementing a hook

Hooks are Python scripts stored in the *pootle/scripts* directory with the same name as a project. Thus, `hello.py` for a project called **hello**.

The project hook should implement functions for each needed hooktype.

Available hooktypes

Hooktype	Arguments	Return
initialize	projectdir, languagecode	<i>unused</i>
precommit	file, author, message	array of strings indicating what files to commit
postcommit	file, success	<i>unused</i>
preupdate	file	pathname of file to update
postupdate	file	<i>unused</i>
pretemplateupdate	file	boolean indicating whether file should be updated from template

initialize

This hook is called when a language is added to a project. It can be used to set up any additional files that may be needed (e.g. alternate formats) or even handle repositories with special directory layouts, by adding appropriate symlinks in the *VCS_DIRECTORY*.

The first parameter is the path to the project directory. It's up to this script to know any internal structure of the directory (in particular whether standard, GNU, or a special *tree style* is used).

The second parameter is the code for the language (e.g. nl, pt-BR, sr_RS@latin etc.) that is being added to the project.

precommit

This hook is called when a translation file for a project is *committed* to VCS (possibly as a result of an *update against templates* that added a new file, in which case it is called after the `pretemplateupdate` hook). It can be used to perform conversion to another format and other pre-commit checks and fixups.

The first parameter is the path to the file that will be committed. The second parameter is the author what will be used for the commit, and the third argument is the commit message.

This hook should return an array (possibly empty) of filenames that should also be committed together with the translation file.

postcommit

This hook is called under the same circumstances as the `precommit` hook, after the commit operation has been attempted. It can be used to do logging or cleanup of resources created by the `precommit` hook.

The first parameter is the path to the file that will be committed. The second parameter is a boolean indicating whether the commit was successful.

preupdate

This hook is called when a translation file for a project is *updated* from VCS. It can be used to set up for conversion from another format in the project source files.

The first (and only) parameter is the path to the translation file that will be updated.

This hook should return the filename that will be updated. Normally this should be identical to the filename parameter passed to the hook, but if format conversion is needed, this can be another (project source) file to be updated instead, so that the `postupdate` hook can use it to generate the Pootle translation file.

`postupdate`

This hook is called under the same circumstances as the `preupdate` hook, after the update has been completed (if the update fails, a `VersionControlError` is raised and this hook is not called). It can be used to do format conversion to generate Pootle translation files from project source formats as well as other logging.

The first parameter is the path to the file that was updated.

Note: If a `preupdate` hook changes the file to be updated (by returning a string other than the filename it is passed) the original filename, not the modified one it returns, will be passed to the `postupdate` hook, if there is one.

`pretemplateupdate`

New in version 2.5.1.

This hook is called when a translation file for a project is *updated against templates* to get new source strings (and mark removed strings as obsolete). It can be used to customize the handling of new or obsolete strings or to prevent updating against templates for any reason.

The first parameter is the path to the template file that will be used to update the translation file. If the hook returns false, that template file will not be used to generate updates for translation files

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 will be used as templates. For *Monolingual formats* the files representing the source language are used as templates.

The “*Templates*” language

Pootle has 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 absent from a project, Pootle will assume files under the project's source language are translation templates.

Gettext PO projects should always use a templates project where POT files can be uploaded. For simple projects (and most monolingual formats) it will be simpler to use the source language.

Starting a new translation

When adding a new language to a project, Pootle will first scan the file system and look for translation files for that language. If none are present a fresh copy will be generated based on the templates files (in a manner similar to [pot2po](#)).

Updating against templates

When the document or software being translated is updated, Pootle helps you retain old translation through the translation templates feature.

The templates files should be replaced with new versions (i.e. upload the new versions to the templates language). Users with admin permission in the project can use the *Update against templates* checkbox in the project admin page to update languages to the newer version.

Users with admin permissions over a language can update this single language from the files tab for the translation project.

This will update both the files and the database retaining old translations and using fuzzy matching to match translations when the source text had minor changes (in a manner similar to [pot2po](#)). Fuzzy matched strings will be marked as fuzzy.

Integration with Version Control Systems

Pootle has the ability to integrate with version control systems (also called revision control systems). Read more on Wikipedia for a general overview of what a [Version Control System](#) is.

Supported systems

System	Status
CVS	Supported
Subversion	Supported
Darcs	Supported
Git	Supported
Bazaar	Supported
Mercurial	Supported

It should be possible to add other systems fairly easily. Interested programmers can look at the [versioncontrol](#) module.

Preparation

Note: Changed in version 2.5: [VCS_DIRECTORY](#) was introduced for separating version control directories. Previously your [PO_DIRECTORY](#) contained your files from version control. Separation allows Pootle to work reliably on Distributed Version Control Systems (Git, Mercurial, etc).

Note: The setup of version control has to be done outside of the Pootle admin interface.

To have any sort of integration with version control from within Pootle, it is necessary to construct the correct file system structure in the *VCS_DIRECTORY* as defined in the settings. Any projects integrating with a version control system have to follow a layout that corresponds to the *PODIRECTORY*. The *VCS_DIRECTORY* is *pootle/repos* by default and should contain one directory for each project on the server that is either a clone/checkout for the corresponding Pootle project, or a symlink (or a directory with symlinks) to the repo somewhere else on the file system.

The *PODIRECTORY* therefore contains the translation files used during normal operation of Pootle, and the *VCS_DIRECTORY* contains “clean” versions (with no uncommitted changes) that enables the version control integration. The meta files for the version control system (CVS/, *.svn*/, *.hg*, *.git*, etc.) therefore should be present in *VCS_DIRECTORY* for Pootle to perform the integration.

An example layout:

```
.../
|-- po
|   |-- project1
|       |-- de.po
|       |-- fr.po
|       |-- pt_BR.po
|-- repos
    |-- project1
        |-- de.po
        |-- fr.po
        |-- pt_BR.po
```

Here *VCS_DIRECTORY* is *.../repos* and *PODIRECTORY* is *.../po*. The directory *.../repos/project1* contains a clean checkout of the translations from version control. This is where Pootle will perform any version control actions such as updates and commits.

The *VCS_DIRECTORY* should never contain uncommitted changes. Pootle will bring in changes from the upstream VCS and rely on it succeeding without conflicts.

Example

```
$ cd pootle/repos/
$ svn co https://translate.svn.sourceforge.net/svnroot/translate/src/trunk/Pootle/po/pootle
```

Now you have the directory *pootle* containing a translation project. If that directory is now one of your projects registered on the server, the version control functions should appear for all users with the necessary privileges. Look for the functions under the actions on the overview page.

Note: The summary of steps to add a new project which will use a VCS are:

1. Create a local copy of the repository in *VCS_DIRECTORY* (for example using `svn checkout` in Subversion, or `git clone` in Git),
2. Copy the newly created directory, which holds the translation files for the new project, from *VCS_DIRECTORY* to *PODIRECTORY*,
3. Add the project via the administration panel. Remember that the project code should match the project directory name both in *VCS_DIRECTORY* and *PODIRECTORY*.

The project will be automatically imported by Pootle.

How to treat special directory layouts

There exists some conventions for directories.

Convention	Directory structure
Standard convention	<code>PODIRECTORY/project_name/language_code/files.po</code>
GNU convention	<code>PODIRECTORY/project_name/language_code.po</code>

Is the directory structure for the language files of your project different from the default structure found in the source project?

If yes, then you might consider using symlinking every single language file to the expected location. The version control support of Pootle will follow these links. Thus the meta directories of your version control system (e.g.: `.svn/` or `CVS/`) do not necessarily have to be below your `VCS_DIRECTORY` (see your settings for the value of this setting). In this case, everything under `VCS_DIRECTORY` for this project must be outside of the clone/checkout for the project.

You can use an *initialize* hook script to automate the creation of these symlinks whenever languages are added to your project.

If you use symlinks, ensure that the resulting structure under `VCS_DIRECTORY` corresponds to the structure under `PODIRECTORY`.

Working with VCS integrated projects

Once you have added a project with VCS integration to Pootle, if you have the necessary privileges, you will be able to perform the different version control functions from the actions section on the translation project overview page.

Updating If you want to update the Pootle copy of the translations with the version that is currently in version control, a contributor with the ‘update’ right can click on the *Update* link for a file which will then perform the update process. The project administrator needs to assign the “update” right.

When updating from version control there is the possibility that a third party could have changed the file (another translator accessing the version control directly could have made a change). Traditionally in version control this would create a file with conflicts. Those familiar with version control conflicts will understand that we can’t afford to have that as we won’t be able to resolve them. Therefore Pootle will be conservative and will consider the version control system to be the authority and any conflicts in the local file get be converted to suggestions. These suggestions then need to be reviewed by a translator with *review* rights.

Committing You can commit translation files from within Pootle. In the case where authentication is required to submit the translation to version control (probably almost all relevant systems), there needs to be a non-blocking authentication method. Pootle will not be able to commit if a password is necessary to complete the action. This unfortunately means that it will probably not be realistic for Pootle to commit with the project admin’s credentials, as this will require his/her private key(s) to be on the Pootle server.

This usage scenario is more useful for people hosting their own Pootle server where they are able to setup one commit account on the version control server, or perhaps one account for each team. A typical commit message when committing from Pootle will look something like this:

```
Commit from GNOME Pootle by user Sipho. 80 of 100 messages translated (7
fuzzy).
```

So it is still possible to see who submitted what and when, and actually provides some useful statistics in the commit message. A user must be assigned ‘commit’ privileges by the project administrator. If the user has the correct privileges, they will see a “submit” link next to each file.

Version Control Authentication To access the repository of version controlled files (especially for submitting), it is necessary to configure a non-interactive authentication. This enables the Pootle server to connect to the version control server and to submit changes with the appropriate privileges.

The following examples should help the pootle administrator to configure this authentication properly.

Subversion (HTTP)

- Add a new user with appropriate privileges to the subversion server, if necessary (e.g. read [subversion authentication](#))
- Make sure, that the *pootle user* has write access for `~/.subversion/` to store authentication tokens. The *pootle user* is whichever user is running the Pootle application. When running behind a webserver this might be the webserver user. Thus on some systems using Apache that user is *www-data*.
- Do a real `svn commit` with the uid of the *pootle user* in order to:
 - Import (possibly) an SSL certificate
 - Store the username and password in the subversion authentication cache (by default, the option `store-passwords` is enabled in `~/.subversion/config`)
- If you start Pootle from an init script, make sure that all necessary environment variables are set. `$HOME` will be needed to obtain your cached authentication information, for example.

From now on, the *pootle user* should use these stored access credentials when uploading commits for this repository.

Adding New in version 2.5.

When a language is initialized from templates, Pootle will check if it is inside a version control checkout/clone. If it is, it will add the new files as initialized from the templates, and commit these initial versions. The same is done when updating against templates at a later stage – if this introduced any new files, these will be added to the configured version control system.

A typical commit message when adding from Pootle will look something like this:

```
New files added from Labs Translation Server based on templates
```

1.2.2 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.

94 Terminology

translation itzulpena

translate itzuli

project proiektua

local_apps/pootle_app/templates/index/about.html:19

Spanish

Este sitio usa Pootle — un potente software web para potenciar a los equipos de traducción y la gestión de traducciones. Esta herramienta comunitaria es Software Libre, y cualquier equipo puede utilizarla y contribuir a la comunidad Pootle. Lea más en el <http://translate.sourceforge.net/> sitio web del proyecto.

French

Ce site utilise le logiciel Pootle — logiciel web puissant permettant le travail de traduction en équipe et la gestion des traductions. Cet outil communautaire est un logiciel libre et chacun peut contribuer à la communauté de Pootle. Plus d'informations sont disponibles sur <http://translate.sourceforge.net/> le site web du projet.

English

This site is running Pootle — powerful web software to empower teams to do translation and translation management. This community tool is Free Software, and any team can use it and contribute to the Pootle community. Read more on the <http://translate.sourceforge.net/> project website.

Gune hau Pootle, exekutatzen ari da — taldeak itzulpena eta itzulpenen kudeaketara bideratzen dituen web bidezko software boteretsua. Komunitatearen tresna hau Software Libre da eta edozein taldek erabili eta ekarpenak egin ditzake Pootle komunitatera. Irakurri gehiago <http://translate.sourceforge.net/> proiektuaren webgunean.

Submit

Suggest

Fuzzy

Previous Add Comment Next

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 to help people insert characters which might be difficult to type. For many languages using the Latin alphabet with diacritics, this helps a lot, especially where keyboard layouts are not always common.

This page allows people to specify which characters might be useful for translators. This does 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. We will probably need to limit the number of characters, but hopefully we can find a reasonable compromise that will help many people.

If you edit this page, please ensure that you use a browser that supports UTF-8 encoding so that the existing text is kept intact. Note that you might not be able to see all the characters on this page if you do not have the appropriate fonts installed. Please take care not to edit something inadvertently.

The characters

Afrikaans áéíóúý äëïöü âêîôû è

Northern Sotho šš

Tswana šš êô

Venda

old orthography: áéíóú aeïou

Vietnamese (vi) àáãääèèëèìíòóôùúýđđ«» ÀÃÄÅÂÊËÊÏÎÕÖÔÙÚÝÐ

You could prioritize by excluding the characters covered by the Latin-1 codeset, which are available via most standard keyboards.

Since Vietnamese is mostly composed of accented vowels as above, the priority should be to help users acquire the appropriate input systems and keyboard layouts. Relying on clicking each character from a palette would slow down translation severely. However, it would make translation possible in the short term for those who can't yet input our language, or for those accessing computers which for some reason won't use the correct input software.

Quality checks

Pootle provides a powerful way of reviewing translations for quality. It exposes most of the [pofilter tests](#) that can test 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.

To review the quality checks you need to have translation rights for the project. To find the results, click on the "Review" tab. Clicking on the name of a test will step you through the translations that fail the test.

To understand the meaning of each test, Pootle displays the failing tests on the top-right corner of the translation page with a link to the online documentation. You can also read the detailed descriptions of the [pofilter tests](#).

Overriding Quality Checks

New in version 2.1.

It is possible to override the quality check if the translation is correct. Reviewers are able to remove the check for a certain string to indicate that the string is correctly translated. This avoids having to recheck the same checks multiple times.

If the translation is changed, this information is discarded to ensure that the new translation is tested again for any possible issues.

Translation Memory

Changed in version 2.5.

Pootle provides suggested translations to the current string. Translator can use these suggestions for the translation.

Suggestions 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 suggestions are automatically retrieved when you enter a new translation unit. These are displayed below the editing widget. You can insert a TM suggestion by clicking on the suggestion row.

The differences between the current string and the suggested string are highlighted, this allows you to see how the two differ and helps you make changes to the suggestion to make it work as the current translation.

Configuring Translation Memory

Translation Memory will work out of the box with a default Pootle installation. 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.


Machine Translation

New in version 2.1.

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 `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

 Apertium

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

On the other hand, [Apertium](#) is best suited for close language pairs. Especially for those languages spoken in the Iberian Peninsula that are similar.

Offline translation

Pootle's strength is making translation management easy, allowing large teams to collaborate while ensuring quality and consistency through [features](#) like [quality checks](#) and [Terminology](#), opening the door for casual contributions and crowd-sourcing through [Translation suggestions](#).

However experienced translators might still prefer to use a dedicated desktop translation application.

Offline translation using whatever tool the user prefers can be integrated within Pootle's workflow easily through downloading and uploading translation files.

We recommend [Virtaal](#) for offline translation, as it supports the same formats that Pootle supports, has all of Pootle's features and power and more.

Downloading

From the Translate tab in the translation project page users can download files for offline translations.

Files are available in the original format and in [XLIFF](#) format. Bilingual formats are suitable for offline translation, but [Monolingual formats](#) should be treated as just an export. If you want to translate files offline from a monolingual project, it is best to download the file as XLIFF.

XLIFF export will include all of the information in Pootle's database (like suggestions, fuzzy, translators comment) even if the original format doesn't support representing this information.

In case you want to work with multiple files at once, you can download the whole translation project or the contents of a subdirectory as a ZIP archive.

Uploading

From the translation project page you can upload translations to Pootle. Translations from the uploaded file will be merged with existing translations in the database, the merge process depends on the merging method the user selects, and the [User permissions](#) the user has.

Merging methods

Merge Translations in the uploaded file are accepted for currently untranslated strings. In case of conflict between the current file and the uploaded file, the new translations are turned into suggestions. The file structure will not change, new strings will be ignored. This requires the "translate" permission.

Suggest No translation is accepted, all new translations are added as suggestions. This requires the *suggest* permission.

Overwrite The current file is replaced with the uploaded version. All the current translations on Pootle are lost. The structure of the file may change with new strings introduced and some existing strings deleted. This requires the “overwrite” permission.

Advanced uploading Most of the time, you’ll be uploading files you directly downloaded from Pootle — either in the original format or as XLIFF export. Pootle will match the uploaded file with an existing file based on filename.

In case a file got renamed or you want to merge translations from a different file (for example a translation compendium created by [pocompendium](#)) use the *Upload to* field to specify which existing file to merge with.

Users with admin permissions can introduce new files by just uploading them. When uploading a new file, the merge method is irrelevant.

Users can mass upload files using a ZIP archive. Pootle expects a ZIP file similar to the one it exports. The selected merge method will apply to the content of the archive on a file by file basis.

If a ZIP archive is selected, the *Upload to* field can be used to specify the subdirectory to merge with.

Caveats Users with admin permissions should be very careful when uploading ZIP archives. Mistakes in naming or incorrect *Upload to* choice can lead to introduction of many spurious files.

Users with overwrite permissions should be careful when uploading ZIP archives, as all files will be overwritten including ones that they may not have translated. When overwriting, it is better to do it one file at a time.

Files of a different format can be uploaded, and will be converted on the fly, but the merging behaviour is format specific and not always predictable. This also applies to XLIFF files generated outside of Pootle. It is preferable to always use the original format or the XLIFF file exported by Pootle.

Monolingual files can be uploaded, but this is not recommended for normal use. To merge translations, a corresponding template file is required. The uploaded file and the template file should have exactly the same structure. If their versions differ, incorrect translations may be introduced. This is why we recommend never using monolingual files for offline translations. Uploading monolingual files should be used only when initially importing existing work.

Searching in Pootle

Pootle provides a searching 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 in 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

The search function improved in Pootle 1.2. Next to the search textbox, there's an arrow icon that when clicked will toggle some options for the search to be done.

At this stage, the advanced search option allows **searching in specific fields**. It is possible to select to search for text in source and target texts as well as in comments and locations. Any combination of these options is accepted.

As a default, it will search for source and target strings. If a non-default search is performed, the search widget will slightly change its background colour.

Backend The basic searching uses [pogrep](#) which will look for the substring in the source and target text. It supports Unicode normalization. Alternatively, a Pootle server might be installed with an [indexing engine](#) (PyLucene or Xapian) to speed up searching. Search results can differ slightly from the normal search, based on the indexing that engine uses.

Some basic query in Lucene syntax is also possible. More information on [Lucene queries](#).

Keyboard shortcuts

New in version 2.5.

Global

Action	Shortcut
Content Zoom Out	Ctrl+Shift+-
Reset Content Zoom	Ctrl+Shift+0
Content Zoom In	Ctrl+Shift+"+"

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->l  er*, and we translate a sentence like *The file was not found*, we can suggest the glossary entry *file->l  er* 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.

Generating terminology

New in version 2.1.

Project administrators can generate a list of frequently occurring terms from the *Terminology* tab in the Pootle interface, which can help to quickly standardize some frequently occurring terms. It is also possible to add extra entries. This is based on the [poterminology](#) tool from the Translate Toolkit.

Extension Actions

New in version 2.5.1.

Extension actions are used to add custom action links to Pootle. Extension actions are Python scripts that can perform tasks such as generating language packs, provide downloads, or performing checks across several languages and returning a report.

Implementing an extension action

Extension actions are Python classes in module files stored in the *pootle/scripts/ext_actions* directory. The names of modules or classes are not important, but only classes derived from the subclasses of `ExtensionAction` will actually create extension actions.

There are several subclasses of `ExtensionAction` defined, depending on the scope of the action.

- `ProjectAction` – an action that will apply to all or part of a project.
- `LanguageAction` – an action that will apply to a language across multiple projects.
- `TranslationProjectAction` – an action that will apply to a single language within a project.
- `StoreAction` – an action that will apply only to a single translation file (store) within a specific language and project.

Extension Action properties

Every extension action has a category and a title. The category is used to group related actions together, and is displayed as a small text label on the left margin of the **Actions** section of the relevant Pootle screens. The built-in actions for downloading and uploading translation files and archives use the category ‘Translate offline’ but extension actions can use any category they wish to. The title of the extension action is the text of the link that is displayed to the right of the category label, and clicking on that link will invoke the action with the current project, language, and/or store.

Extension action tooltips

The docstring for an extension action class is used as the tooltip (mouseover text) for the link created. Like all the other strings in an extension actions module, it too is implicitly internationalized, and can be localized by creating translation files and importing them into Pootle.

Localization of categories and titles

It may be desirable to have the category and title text displayed in different languages depending on the preferences of users. If either property is a string that is already in the Pootle localization (e.g. ‘Translate offline’) it will be displayed using the standard Pootle localization for the current locale selected by the user preferences.

Titles and categories that are not already localized (or for which a different localization is desired) should be extracted from the extension action module using the `i18n` script located in the `pootle/scripts/ext_actions` directory:

```
$ cd $POOTLE_HOME/pootle/scripts/ext_actions
$ ls
hello.py      i18n
$ ./i18n hello
$ ls
hello.pot     hello.py      i18n
```

The generated `module.pot` file is a translation template file that can be moved to a templates subdirectory (and copied to a language subdirectory as a PO file). These directories would be `pootle/scripts/locale/templates` and e.g. `pootle/scripts/locale/fr`. The PO file can be localized using Pootle, Virtaal, or any other translation tool that works with PO files.



Tags


New in version 2.5.1.

Pootle supports tags that can be used as a way to group and filter related items.

Note: Currently tags are only available for translation projects and individual files. We expect this to expand in future versions.

Managing tags

Tags are hidden by default, and can be shown by clicking on the  tag icon near the top of the page (in pages that allow showing the tags like project overview, translation project overview, and so on). Clicking on it  again will hide the tags.



When tags are shown, the  add tag icon will be displayed (for users who have enough permissions to add tags). Clicking on the icon will allow you to add a new tag to the item. If the tag does not exist then one will be created and applied to the current item.

Note: Tag names are case insensitive (they will be automatically converted to lowercase).

Warning: Tags must be composed of only letters and numbers, and can have spaces, colons, hyphens, underscores, slashes or periods in the middle, but not at start or at the end.

Hovering over any of the tags with the mouse will show the “x” icon that can be clicked to remove the tag.

Filtering tags

On the project overview page, translation projects can be filtered based on their tags. Clicking on the  filter icon (near the top of the page), will activate tag filtering. Use the tag filtering area to filter the list of translation projects. Clicking on the  icon again will hide the filtering widget and reset the filters.

Goals

New in version 2.5.1.

Goals provide a way to group work based on any criteria. They can be prioritized to allow translators to focus on the most important work.

By using *project goals* goals can be applied to the same file across all the languages in the project.

Note: Currently goals can only be applied to individual files.

Regular goals vs project goals

Pootle supports two types of goals:

1. *Regular goals* (or just goals)
2. *Project goals*

Project goals are available in all languages. They are applied to files in the *template* translation project. This allows project managers to easily define a goal shared across all languages in the project.

The goal type can easily be changed using the *goal editing form*.

Project goals are shown below regular goals in the *goals tab*.

The statistics for a goal in a translation project will only include files that are part of that goal, and won't be displayed at all if the goal doesn't have any matching files in the current directory of the project.

Goals tab

The goals tab is shown on the overview page for any translation project with goals applied to any of its files. When shown, the goals tab provides a comprehensive list of all the goals in that translation project, including statistics for each goal and links for working on the translations.

The goals tab is also displayed on any directory, if there is any goal applied to files inside that directory and its subdirectories.

Drill down into a goal It is possible to drill down into each goal on the goals tab to see the files and directories that belong to a goal. This works like the regular files view with some small differences:

- In the upper level `..` will return you to the list of goals,
- Breadcrumbs includes a reference to the current goal,
- Every translate link in the table points to a translate view restricted to the goal that is currently being viewed.

Adding and removing files from a goal

Goals are special tags which start with **goal:** (including the colon) and that have some additional attributes.

Note: Like tag names, the goal names are case insensitive (they are automatically converted to lowercase), and must be composed of only letters and numbers, and can have spaces, colons, hyphens, underscores, slashes or periods in the middle, but not at start or at the end.

Note: If you create a goal without the **goal:** prefix then an ordinary tag will be created instead.

Goals can be added and removed from a file as you would *add and remove tags*. If the goal did not previously exist then a new goal is created. While if you remove a goal from a file it will simply remove the association of that file to the goal, the goal itself will not be removed.

Editing goals

To modify the properties of goals go to the goals tab and drill down into the goal. Use the form in the *Description* section to modify any of the goal properties.

Note: Remember that if the goal is not applied to any files then it is not possible to edit the goal, as you won't have access to it in the goals tab. Simply add a file to the goal and you will be able to edit the goal.

You can modify the goal description and turn it into a project wide goal as needed.

Translating goals

The goals tab and goals drill down views provide translation links as in the normal file view that will take you to the translation editor. Each link allows you to translate strings limited to the current goal.

Once in the translation editor the different filters are restricted to the stores in the given path that belong to the chosen goal, thus allowing you to focus on the work in the current goal.

1.2.3 Administrative features

Captcha Support

New in version 2.0.5.

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

Changed in version 2.1: Captchas are now enabled by default.

If you have no need for captchas, e.g. at a translation sprint, you might want to remove captcha support. To disable it, set `USE_CAPTCHA` 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.

News

Pootle provides the ability to access useful information about your language and projects. These can be read on Pootle, or through your RSS reader.

Project administrators can write notices, and Pootle will also generate entries automatically for these events:

- New languages
- New projects
- New projects added to languages
- Project updated from version control
- Project updated to templates
- File committed to version control
- File uploaded to project
- Archive uploaded to project
- File reached 100%

Notifications

Pootle has RSS feeds for notifications about concrete translation projects, languages or even the whole site. Pootle’s front page will show the latest events on the site.

Types of notifications

Notifications can either be manual or automatic.

Manual notifications are written by the language or translation project administrators and are shown in the relevant pages within the “News” tab.

When certain events occur, events will automatically be notified in the relevant feeds. The events that generate notices include:

- New languages added
- New projects added
- New projects added to languages
- Project updated from version control
- Project updated against templates
- File committed to version control
- File uploaded to project
- Archive uploaded to project
- File reached 100%
- User registers in a language
- User registers in a project

If you want to receive all events for a language (including sub-projects) or absolutely everything on the whole server, add `?all=True` to the end of the URL for the RSS feed. (This is not currently advertised on Pootle due to possible performance impact.)

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

The following permissions may be set for the server or per language, or language-project combination:

view Limits access to project of language but does not limit it's visibility.

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.

archive The right to download archives (ZIP files) of a project or directory.

overwrite The right to entirely overwrite a file at the time of upload (instead of the default behaviour of merging translations)

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)

commit The right to update or commit a file to the version control system (if the files are configured for [Integration with Version Control Systems](#) integration)

Permissions interface

Users with administrative rights for languages or translation projects can access the permissions interface by clicking on the *Permissions* tab on the language or translation project index pages.

Pootle administrators will find the default permissions interface on the administration page, at the “Permissions” tab.

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 that user or class of users. You might need to hold down the `Ctrl` key of you keyboard to select multiple rights. Changes will be updated when you submit the form.

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.

Users who selected the language or project in their profile settings will be listed as the project or language team. After that follows a list of all registered users.

Removing a user To reset a user’s rights to the default rights, select the 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 Administering a server

1.3.1 Installation

Installation

Want to try Pootle? This guide will guide you through installing Pootle and its requirements in a virtual environment.

Before proceeding, consider installing these first:

- At least Python 2.6
- `python-pip`

If you only want to have a sneak peek of Pootle, the default configuration and the built-in server will suffice. But in case you want to deploy a real world server, *installing optional packages*, using a real database and a *proper web server* is highly recommended.

Note: The easiest way to test and install Pootle is by using pip. However, installations can be done straight from git sources or be automated by using *fabric deployment scripts*.

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.

Setting up the Environment

In order to install Pootle you will first create a virtual environment. This allows you to keep track of dependencies without messing up with system packages. For that purpose you need to install the `virtualenv` package. You might already have it, but in case you haven't:

```
$ pip install virtualenv
```

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

```
$ virtualenv /var/www/pootle/env/
```

This will copy the system's default Python interpreter into your environment. For activating the virtual environment you must run the `activate` script:

```
$ source /var/www/pootle/env/bin/activate
```

Every time you activate this virtual environment, the Python interpreter will know where to look for libraries. Also notice the environment name will be prepended to the shell prompt.

Installing Pootle

After creating the virtual environment, you can safely ask pip to grab and install Pootle by running:


```
(env) $ pip install "Pootle==2.5.1.3"
```

This will fetch and install the minimum set of required dependencies as well.

Note: If you run into trouble while installing the dependencies, it's likely that you're missing some extra packages needed to build those third-party packages.

For example, `lxml` needs a C compiler.

`lxml` also require the development packages of `libxml2` and `libxslt`.

If everything went well, you will now be able to access the `pootle` command line tool within your environment.

```
(env) $ pootle --version
Pootle 2.5.1.2
Translate Toolkit 1.10
Django 1.4.10
```

Initializing the Configuration

Once Pootle has been installed, you will need to initialize a configuration file for it. This is as easy as running:

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

By default it writes the configuration file at `~/.pootle/pootle.conf` but if you want you can pass an alternative path as an argument to the `init` command. If the desired path exists, you will be prompted for whether to overwrite the old configuration. Passing the `--noinput` flag assumes a negative answer.

Warning: This default configuration is enough to initially experiment with Pootle but **it's highly discouraged and unsupported to use this configuration in a production environment**.

Also, the default configuration uses SQLite, which shouldn't be used for anything more than testing purposes.

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

Setting Up the Database

Before your 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 [setup](#) management command:

```
(env) $ pootle setup
```

Note: If you are installing directly from the code then you must also build the assets after running the previous command:

```
(env) $ pootle collectstatic --noinput
(env) $ pootle assets build
```

Running Pootle

By default Pootle provides a built-in [CherryPy server](#) that will be enough for quickly testing the software. To run it, just issue:

```
(env) $ pootle start
```

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

Setting up a Reverse Proxy

By default the Pootle server runs on port 8000 and you will probably be interested on binding it to the usual port 80. Also, it's highly recommended to have all the static assets served by a proper web server, and setting up a web proxy is the simplest way to go.

The *Running under a Web Server* section has further information on setting up a web server that proxies requests to the application server.

If you want to omit a reverse proxy and rather prefer to use a web server for serving both dynamic and static content, you can also setup such a scenario with *Apache and mod_wsgi* for example.

Running Pootle as a Service

If you plan to run Pootle as a system service, you can use whatever software you are familiar with for that purpose. For example *Supervisor*, *Circus* or *daemontools* might fit your needs.

Further Configuration and Tuning

This has been a quickstart for getting you up and running. If you want to continue diving into Pootle, you should first consider *making some optimizations to your setup* — don't forget to switch your database backend! After that you should also *adjust the application configuration* to better suit your specific needs.

For additional scripting and improved management, Pootle also provides a set of *management commands* to ease the automation of common administration tasks.

Automated deployment using Fabric

Pootle can be deployed using Fabric automation scripts. There are other methods to deploy Pootle, but using Fabric with these scripts allows automated deployments and simplifies maintenance tasks and the upgrade to newer versions.

The sample scripts bundled with Pootle allow you to deploy a Pootle server using a Python virtualenv, running on a **Apache** server with *mod_wsgi* using **MySQL** as database server on **Debian**-based systems. These sample scripts can be modified to perform different deployments.

To see a comprehensive list of all Fabric commands available to deploy Pootle check the *Fabric commands reference*.

Preparing the remote server

Before performing an automated deployment using Fabric, make sure the server where Pootle is going to be deployed has the required software.

Installing required software on the remote server Before proceeding, install the following software (if absent) on the remote server:

- Python 2.5, 2.6, or 2.7

- `python-pip`
- Git distributed version control system
- Apache web server
- MySQL database server
- OpenSSH server
- C compiler (to install Pootle's Python dependencies – can be removed later)

Note: Currently only Debian-based (e.g. Ubuntu) servers are supported.

Note: If you have problems installing the dependencies during the bootstrap you are probably missing other packages needed to build those third-party Python modules. For example, `lxml` needs development files for `libxml2` and `libxslt1` (as well as the C compiler mentioned above).

Note: Also consider *installing optional packages* for optimal performance.

Hardware requirements Check the *Hardware requirements* on Installation docs.

Preparing Fabric deployment

Before performing a deployment you will need to install some software on the local computer and configure the necessary settings to connect to the remote server.

Installing required software on the local computer The first step is to install the necessary software on the local computer.

Note: We strongly recommend using a virtual environment (virtualenv). Check the *Setting up the Environment* docs for information about virtualenvs.

```
$ pip install Fabric
```

Getting Pootle Fabric files Pootle is bundled with sample scripts for deploying using Fabric. The relevant files are:

- `fabfile.py`
- Files inside the `deploy/` directory

You can get those files from the [Pootle GitHub repository](#). The rest of the Pootle files are not necessary for this kind of deployment.

Setting up Fabric The `deploy/` directory contains sample files that can be used in combination with the `fabfile.py` file for deploying Pootle servers.

There are two different deployment environments. Each one has a directory inside `deploy/`:

- Staging environment: `deploy/staging/` directory
- Production environment: `deploy/production/` directory

This way server administrators can separate their testing and real-world Pootle servers.

For deploying a Pootle server using one of the environments it is necessary to put some configuration files in place:

- `deploy/pootle.wsgi` WSGI script that will be used to run Pootle.
- `deploy/ENVIRONMENT/fabric.py` Module with settings that will be used in Fabric.
- `deploy/ENVIRONMENT/settings.conf` Pootle-specific settings for the server (it will override the defaults). For example, the settings for connecting to the database will go here.
- `deploy/ENVIRONMENT/virtualhost.conf` Apache VirtualHost configuration file.

In the previous paths `ENVIRONMENT` is the directory name for the chosen environment (production or staging).

All the settings defined in the `deploy/ENVIRONMENT/fabric.py` module will populate the Fabric `env` dictionary, making the configuration keys available in the `deploy/ENVIRONMENT/settings.conf` and `deploy/ENVIRONMENT/virtualhost.conf` files. You can use basic Python string formatting to access the configuration values.

Note: Sample configuration files are provided for reference under the `deploy/` directory. Put them in the desired environment directory, and adapt them to your needs before running any Fabric commands.

Check `pootle/settings/90-local.conf.sample` to see settings that you might want to use in `deploy/ENVIRONMENT/settings.conf`.

Note: If it is necessary you can adapt the `deploy/pootle.wsgi` file to meet your needs.

Once you make your changes to the settings you are ready to run the Fabric commands.

Note: For security, please make sure you change the `db_password` setting – using the example one could make your server vulnerable to exploits. The `db_password` setting is used both to properly configure Pootle, as well as to set up the database user access for the deployment.

The `db_root_password` setting, on the other hand, is only used to configure the MySQL options file, if you choose to do this, and is only needed when creating the database (if the normal user does not have the necessary permissions). Leaving this with default setting will have no security impact.

How to run Fabric commands

In order to run Fabric commands for Pootle it is necessary that the directory containing the `fabfile.py` file and the `deploy` subdirectory be included in the `PYTHONPATH`. If it is not, then add it using:

```
$ export PYTHONPATH=`pwd`: $PYTHONPATH
```

The fabric commands need to know the type of environment in which they are going to work, e.g. if the deployment will be for the production environment. The Fabric commands for Pootle support two environments: `production` and `staging`. To select the environment for running a command just add it before the command like this:

```
$ fab production bootstrap # Use the 'production' environment
$ fab staging bootstrap    # Or use the 'staging' environment
```

Note: It is necessary to *install Fabric* in order to be able to run the **fab** command.

Provide arguments Some commands do accept arguments – the argument name is followed by a colon (:) and the value for the argument (with no spaces):

```
$ fab production load_db:dumpfile=backup_mysql.sql # Call load_db providing a database dump to load
```

The previous call runs the *load_db* command providing the value `backup_mysql.sql` for its `dumpfile` argument.

Tweak the environment One possible use for arguments is to tweak the environment when setting it, before calling the commands:

```
$ fab production:branch=stable/2.5.0 bootstrap # Run bootstrap for a branch
```

In the previous example *bootstrap* is run after setting the environment using *production* but changing the branch to work on, to be the value `stable/2.5.0` passed to the `branch` argument.

Run several commands in a row It is possible to run several commands in a row with a single call:

```
$ # Run several commands in a row using a single call to fab
$ fab production:branch=stable/2.5.0 bootstrap create_db load_db:dumpfile=backup_mysql.sql
```

The previous call will run *production* followed by *bootstrap*, *create_db* and *load_db*, in that exact order.

Note: If you want to know more about Fabric, please read [its documentation](#).

See the *Fabric commands reference* for a comprehensive list of all Fabric commands available for deploying Pootle, with detailed descriptions of each command.

Configuring passwordless access

While it is not required, it is much easier to perform deployment operations without interactive prompts for login, sudo, or MySQL database passwords:

- You can eliminate the need for an SSH login password by adding your public SSH key(s) to the `~/.ssh/authorized_hosts` file of the user on the remote server.
- You can eliminate the need for sudo passwords by adding in the `/etc/sudoers.d/` directory on the remote server, a file (with permissions mode 440) containing the line:

```
username ALL = (ALL) NOPASSWD: ALL
```

where *username* must be replaced with the user configured in the `fabric.py` settings file.

- You can eliminate the need for MySQL passwords by configuring the database password(s) in the `fabric.py` settings file, running the *mysql_conf* fabric command to create a MySQL options file for the remote user:

```
$ fab production mysql_conf # Set up MySQL options file
```

and then modifying the `fabric.py` settings file to un-comment the alternate value for `db_password_opt` (and optionally `db_root_password_opt`, if `db_root_password` is configured).

Typical Usage Example

A typical usage example is included here in order to provide a more easy to understand example on how to use this deployment method and the available commands.

Bootstrapping the environment You can install the Pootle software, configuration files, and directory tree(s) with the bootstrap command.

```
$ export PYTHONPATH=`pwd`: $PYTHONPATH
$ fab production:branch=stable/2.5.0 bootstrap # Install Pootle 2.5
```

Setting Up the Database When setting up the database there are several possible scenarios:

- If creating a new database from scratch:

```
$ fab production create_db # Creates Pootle DB on MySQL
$ fab production update_config # Uploads the settings
$ fab production setup # Creates the DB tables and populates the DB
```

- If creating a blank database and populating with a (local) database backup:

```
$ fab production create_db # Creates Pootle DB on MySQL
$ fab production load_db:dumpfile=backup_mysql.sql # Populate DB from local dump
```

Note: The `dumpfile` (for `load_db` and `dump_db`) is local to the system where Fabric runs, and is automatically copied to/from the remote server.

- If updating a previous version database (possibly just loaded with `load_db`) to the latest version of the schema:

```
$ fab production update_config # Uploads the settings
$ fab production setup # Updates the DB to the latest version
```

Enabling the web server

```
$ fab production:branch=stable/2.5.0 deploy
```

Available Fabric commands

New in version 2.5: Starting in this release, Pootle includes Fabric deployment scripts.

Introduction

The sample Fabric scripts provide several commands that you can use to easily deploy your Pootle site.

Note: Most of the examples in this section will use the `production` environment, but remember that other environments can be used as well.

Please read first the [How to run Fabric commands](#) section in order to know how this commands can be run. Reading the [Typical Usage Example](#) section might be helpful as well.

Available commands

bootstrap This command:

- Creates the required directories, asking to remove them first if they already exist
- Creates a virtual environment (virtualenv) and activates it
- Clones the Pootle repository from GitHub

- Checks out the specified branch, using master if no branch is specified
- Installs the deployment requirements as listed in `requirements/`, including the base requirements as well

Note: While running it may ask for the remote server `root` password, or more likely the `sudo` password, which is the standard password for the remote user configured in the environment.

Note: Changed in version 2.5.1: Added support for bootstrapping from a given branch on Pootle repository.

Examples:

```
$ fab production bootstrap # Call that will use the default 'master' branch
$ fab production:branch=stable/2.5.0 bootstrap # Call which provides a branch
```

compile_translations This command:

- Compiles the MO files for Pootle translations

Examples:

```
$ fab production compile_translations
```

create_db New in version 2.5.1.

This command:

- Creates a new blank database using the settings provided to Fabric in the chosen environment

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment) as well as the specified `db_user` and/or database root password. See the `mysql_conf` command for a way to eliminate the need for database password prompting.

Note: This command will try to create a database on MySQL, which will fail if MySQL is not installed or the settings don't provide configuration data for creating the database.

Examples:

```
$ fab production create_db
```

deploy This command:

- Calls the `update_code` command
- Calls the `syncdb` command
- Calls the `migratedb` command
- Calls the `deploy_static` command
- Calls the `install_site` command

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment).

Note: Changed in version 2.5.1: Added support for deploying from a given branch on Pootle repository.

Examples:

```
$ fab production deploy # Call that will use the default 'master' branch
$ fab production:branch=stable/2.5.0 deploy # Call which provides a branch
```

deploy_static This command:

- Creates `pootle/assets/` directory if it does not exist
- Runs `collectstatic --noinput --clear` to collect the static files
- Runs `assets build` to create the assets

Examples:

```
$ fab production deploy_static
```

disable_site This command:

- Disables the Pootle site on Apache using the Apache **a2dissite** command

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment).

Examples:

```
$ fab production disable_site
```

drop_db New in version 2.5.1.

This command:

- Drops a database (losing all data!) using the settings provided to Fabric in the chosen environment

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment) as well as the specified `db_user` and/or database root password. See the `mysql_conf` command for a way to eliminate the need for database password prompting.

Examples:

```
$ fab production drop_db
```

dump_db New in version 2.5.1.

This command:

- Dumps the database to the provided filename using the **mysqldump** command
- Downloads the dumpfile to the local computer

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment) as well as the specified `db_user` and/or database root password. See the `mysql_conf` command for a way to eliminate the need for database password prompting.

Note: This commands can be used to perform periodic backups, that can be imported later using the `load_db` com-

mand.

Available arguments:

dumpfile The local filename for the file where the database will be dumped.

Default: `pootle_DB_backup.sql`.

Examples:

```
$ fab production dump_db # Call that will use the default filename
$ fab production dump_db:dumpfile=backup_mysql.sql # Call which provides a filename
```

enable_site This command:

- Enables the Pootle site on Apache using the Apache **a2ensite** command

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment).

Examples:

```
$ fab production enable_site
```

initdb New in version 2.5.1.

This command:

- Runs *initdb* to initialize the database

Examples:

```
$ fab production initdb
```

install_site This command:

- Calls the *update_config* command
- Calls the *enable_site* command

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment).

Examples:

```
$ fab production install_site
```

load_db New in version 2.5.1.

This command:

- Uploads the given SQL dump file to the remote server
- Imports it to the database specified on Fabric settings using the **mysql** command

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment) as well as the specified `db_user` and/or database root password. See the `mysql_conf` command for a way to eliminate the need for database password prompting.

Note: You must first create the database you will import (e.g. using the `create_db` command) before calling this command,

Available arguments:

dumpfile The local SQL dump filename that will be uploaded to the remote server and imported into an existing database on the remote server. This file can be created using the `dump_db` command.

Note: This is a required argument.

Examples:

```
$ fab production create_db # Remember to create the DB first
$ fab production load_db:dumpfile=backup_mysql.sql
```

migratedb New in version 2.5.1.

This command:

- Runs `migrate` to update the 2.5 or later database schema to the latest version

Examples:

```
$ fab production migratedb
```

mysql_conf New in version 2.5.1.

This command creates a `.my.cnf` MySQL options file on the remote system with the password(s) for database access stored in them (the passwords are taken from the `fabric.py` settings file). Once you have done this, you can uncomment the alternate `db_password_opt` and `db_root_password_opt` settings in `fabric.py`, which will eliminate the need for password prompting on all MySQL operations.

Examples:

```
$ fab production mysql_conf
```

production This command:

- Sets up the configuration for the `production` environment in Fabric settings

Note: This command is useless unless it is called before another command or commands.

Note: This command allows changing the settings. To do so just pass it any of its arguments when calling it.

Note that some commands might require passing any or all of these arguments to this command in order to overwrite the default settings before calling those commands. For example the command **stage_feature** requires passing `branch`, `repo` and `feature`.

Note: Changed in version 2.5.1: Added support for altering the settings based on the passed arguments.

Available arguments:

branch A specific branch to check out in the repository.

repo A repository URL to clone from.

This allows to checkout from a fork repository (not necessarily on GitHub) and try new features developed on that repository. It must be an URL that the `git clone` command is able to clone.

feature Allows specifying if the deployment is for a feature-staging server. Such servers are used by Pootle developers in order to allow quick test of new features using a live Pootle server.

Examples:

```
$ fab production bootstrap
```

In the previous example **production** is called to set up the environment for calling **bootstrap** afterwards.

```
$ fab production:branch=feature/extension-actions bootstrap
```

In the previous example **production** is called to set up the environment for calling **bootstrap** afterwards.

The **branch** argument overwrites the default branch in the settings, which are then used for all the subsequent commands (just **bootstrap** in this example).

```
$ fab production:branch=feature/extension-actions,repo=git://github.com/unho/pootle.git bootstrap
```

In the previous example **production** is called to set up the environment for calling **bootstrap** afterwards.

The **branch** and **repo** arguments overwrite the default settings, which are then used for all the subsequent commands (just **bootstrap** in this example).

```
$ fab production:branch=feature/extension-actions,repo=git://github.com/unho/pootle.git,feature=yes s
```

This example is like the previous one, with the addition of the **feature** argument that triggers the altering of several settings. That altering is necessary for working with feature-staging servers.

setup New in version 2.5.1.

This command:

- Runs *setup* to create or upgrade the database as required

Examples:

```
$ fab production setup
```

setup_db New in version 2.5.1.

This command:

- Runs *syncdb --noinput* to create the database schema
- Runs *initdb* to populate the standard schema objects
- Runs *migrate* to bring the database schema up to the latest version

Examples:

```
$ fab production setup_db
```

stage_feature New in version 2.5.1.

This command:

- Calls the *bootstrap* command
- Calls the *create_db* command
- Copies the data in the specified source DB into the DB that will be used for the deployed Pootle
- Calls the *update_db* command
- Calls the *deploy_static* command
- Calls the *install_site* command

Note: While running it may ask for the remote server `root` password and the specified `db_user` password.

Note: This command is intended primarily for deploying ad-hoc Pootle servers for easing the test of feature branches during Pootle development.

Warning: This command might require changing the `source_db` field in the `deploy/ENVIRONMENT/fabric.py` file. Note that the database specified on this field must exist.

Warning: This command requires using the `staging` environment passing to it the `feature` argument, the desired branch and optionally a repository URL.

Examples:

```
$ fab staging:branch=feature/extension-actions,feature=yes stage_feature
$ fab staging:branch=feature/extension-actions,repo=git://github.com/unho/pootle.git,feature=yes sta
```

staging This command:

- Sets up the configuration for the `staging` environment in Fabric settings

Note: This command is useless unless it is called before another command or commands.

Note: This command allows changing the settings. To do so just pass it any of its arguments when calling it.

Note that some commands might require passing any or all of these arguments to this command in order to overwrite the default settings before calling those commands. For example the command **stage_feature** requires passing `branch`, `repo` and `feature`.

Note: Changed in version 2.5.1: Added support for altering the settings based on the passed arguments.

Available arguments:

branch A specific branch to check out in the repository.

repo A repository URL to clone from.

This allows to checkout from a fork repository (not necessarily on GitHub) and try new features developed on that repository. It must be an URL that the `git clone` command is able to clone.

feature Allows specifying if the deployment is for a feature-staging server. Such servers are used by Pootle developers in order to allow quick test of new features using a live Pootle server.

Examples:

```
$ fab staging bootstrap
```

In the previous example **staging** is called to set up the environment for calling **bootstrap** afterwards.

```
$ fab staging:branch=feature/extension-actions bootstrap
```

In the previous example **staging** is called to set up the environment for calling **bootstrap** afterwards.

The `branch` argument overwrites the default branch in the settings, which are then used for all the subsequent commands (just **bootstrap** in this example).

```
$ fab staging:branch=feature/extension-actions,repo=git://github.com/unho/pootle.git bootstrap
```

In the previous example **staging** is called to set up the environment for calling **bootstrap** afterwards.

The `branch` and `repo` arguments overwrite the default settings, which are then used for all the subsequent commands (just **bootstrap** in this example).

```
$ fab staging:branch=feature/extension-actions,repo=git://github.com/unho/pootle.git,feature=yes stage
```

This example is like the previous one, with the addition of the `feature` argument that triggers the altering of several settings. That altering is necessary for working with feature-staging servers.

syncdb New in version 2.5.1.

This command:

- Runs *syncdb --noinput* to create the database schema

Examples:

```
$ fab production syncdb
```

touch This command:

- Reloads daemon processes by touching the WSGI file

Examples:

```
$ fab production touch
```

unstage_feature New in version 2.5.1.

This command:

- Calls the *disable_site* command
- Calls the *drop_db* command
- Removes the configuration files created by the *update_config* command
- Removes the directories created during the deployment, including the ones holding the translation files and the repositories for those translation files

Note: While running it may ask for the remote server `root` password and the specified `db_user` password.

Note: This command is intended for removing Pootle deployments performed using the *stage_feature* command.

Warning: This command requires using the `staging` environment passing to it the `feature` argument and the desired branch.

Examples:

```
$ fab staging:branch=feature/extension-actions,feature=yes unstage_feature
```

update_code This command:

- Updates the Pootle repository from GitHub
- Checks out the specified branch, using `master` if no branch is specified
- Updates the deployment requirements as listed in `requirements/`, including the base requirements as well

Note: Changed in version 2.5.1: Added support for updating code from a given branch on Pootle repository.

Examples:

```
$ fab production update_code # Call that will use the default 'master' branch
$ fab production:branch=stable/2.5.0 update_code # Call which provides a branch
```

update_config This command:

- Will upload the configuration files included in the chosen environment to the remote server:
 - Configure VirtualHost using the provided `virtualhost.conf`
 - Configure WSGI application using the provided `pootle.wsgi`
 - Configure and install custom settings for Pootle using the provided `settings.conf`

Note: While running it may ask for the remote server `root` password or the `sudo` password (standard password for the remote user configured in the environment).

Examples:

```
$ fab production update_config
```

update_db This command:

- Runs *updatedb* and *migrate* to update the database schema to the latest version

Examples:

```
$ fab production update_db
```

upgrade New in version 2.5.1.

This command:

- Runs *upgrade* to apply any special post-schema-upgrade actions (including changes needed for updated Translate Toolkit version). This would typically be performed after running the *update_code* command. If you haven't just upgraded Pootle or the Translate Toolkit to a new release, this isn't generally required, so there is no need to run it unless release notes or other instructions direct you to do so.

Examples:

```
$ fab production upgrade
```

Running under a Web Server

Running Pootle under a proper 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.

Note: Note that translation files must be served directly by the web server. These files are in the location indicated by the Pootle `PODDIRECTORY` setting.

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/
```

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 `PODDIRECTORY` 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:
        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 and Location directives for 'export', and the Alias
    #   directive for 'assets' to include the '/your-pootle'.
    # - Include the following setting in your custom Pootle settings:
```



```

# STATIC_URL = '/your-pootle/assets/'
ServerName my-pootle.example.com

# Set the 'POOTLE_SETTINGS' environment variable pointing at your custom
# Pootle settings file.
#
# If you don't know which settings to include in this file you can use
# the file '90-local.conf.sample' as a starting point. This file can be
# found at '/var/www/pootle/env/lib/python2.7/site-packages/pootle/settings/'.
#
# Another way to specify your custom settings is to comment this
# directive and add a new '90-local.conf' file (by copying the file
# '90-local.conf.sample' and changing the desired settings) in
# '/var/www/pootle/env/lib/python2.7/site-packages/pootle/settings/'
# (default location for a pip-installed Pootle, having Python 2.7).
#
# 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
WSGIProcessGroup pootle

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

# Turn off directory listing by default.
Options -Indexes

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

ExpiresByType image/jpg "access plus 2 hours"
ExpiresByType image/png "access plus 2 hours"

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>
    Order deny,allow
    Allow from all
</Directory>

# Allow downloading translation files directly.
# This location must be the same in the Pootle 'PODIRECTORY' setting.
Alias /export /var/www/pootle/env/lib/python2.7/site-packages/pootle/po

```

```
<Directory /var/www/pootle/env/lib/python2.7/site-packages/pootle/po>
    Order deny,allow
    Allow from all
</Directory>

<Location /export>
    # Compress before being sent to the client over the network.
    # This might require enabling the 'deflate' module.
    SetOutputFilter DEFLATE

    # Enable directory listing.
    Options Indexes
</Location>

</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 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;

    charset utf-8;

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

    location /export {
        alias /path/to/pootle/env/lib/python2.6/site-packages/pootle/po/;
        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;
}
}

```

Proxying with Nginx (FastCGI) Run Pootle as a FastCGI application:

```
$ pootle runfcgi host=127.0.0.1 port=8080
```

There are more possible parameters available. See:

```
$ pootle help runfcgi
```

And add the following lines to your Nginx config file:

```

server {
    listen 80; # port and optionally hostname where nginx listens
    server_name example.com translate.example.com; # names of your site
    # Change the values above to the appropriate values

    location ^~ /assets/ {
        root /path/to/pootle;
    }

    location ^~ /export/ {
        root /path/to/pootle;
    }

    location / {
        fastcgi_pass 127.0.0.1:8000;
        fastcgi_param QUERY_STRING $query_string;
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_TYPE $content_type;
        fastcgi_param CONTENT_LENGTH $content_length;
        fastcgi_param REQUEST_URI $request_uri;
        fastcgi_param DOCUMENT_URI $document_uri;
        fastcgi_param DOCUMENT_ROOT $document_root;
        fastcgi_param SERVER_PROTOCOL $server_protocol;
        fastcgi_param REMOTE_ADDR $remote_addr;
        fastcgi_param REMOTE_PORT $remote_port;
        fastcgi_param SERVER_ADDR $server_addr;
        fastcgi_param SERVER_PORT $server_port;
        fastcgi_param SERVER_NAME $server_name;
        fastcgi_pass_header Authorization;
        fastcgi_intercept_errors off;
        fastcgi_read_timeout 600;
    }
}

```

Note: The `fastcgi_read_timeout` line is only relevant if you're getting Gateway Timeout errors and you find

them annoying. It defines how long (in seconds, default is 60) Nginx will wait for response from Pootle before giving up. Your optimal value will vary depending on the size of your translation project(s) and capabilities of the server.

Note: Not all of these lines may be required. Feel free to remove those you find useless from this instruction.

1.3.2 Upgrading

Upgrading

Here are some points to take into account while performing Pootle upgrades.

Checklist

Before upgrading Pootle to a newer version, make sure to go through this checklist.

- Familiarize yourself with [important changes](#) in Pootle over the versions.
- If you want to change databases, which might be needed when upgrading from Pootle 1.x to Pootle 2.x, or from Pootle 2.0.x to 2.1.x, then have a look at the [database migration](#) page first, although some of the issues on this page could still be relevant.
- Check the [installation](#) instructions for the newer version, and ensure that you have all the dependencies for the newer version.
- Always make backups of all your translation files (your whole [PODIRECTORY](#)) and your custom settings file. You can synchronize all your translation files with the database using the [sync_stores](#) command before you make your backups.
- Make a backup of your complete database using the appropriate *dump* command for your database system. For example **mysqldump** for MySQL, or **pg_dump** for PostgreSQL.
- If you are upgrading from a version of Pootle that uses *localsettings.py* then you want to make sure your configuration file is read when Pootle starts. For more information, read about [customizing settings](#).
- You might want to look for any new [available settings](#) in the new version that you might want to configure.
- After a successful upgrade, consider clearing your cache. For users of memcached it is enough to restart memcached. For users of the default database cache, you can drop the `pootlecache` table and recreate it with:

```
$ pootle createcachetable pootlecache
```

- Finally run the [collectstatic](#) and [assets build](#) commands to update the static assets:

```
$ pootle collectstatic --clear --noinput
$ pootle assets build
```

Performing the Database Upgrade

Changed in version 2.5.1.

Once you have the new code configured to in your server using the correct settings file, you will be ready to run the database schema and data upgrade procedure.

Since the database upgrade procedures have been growing in complexity in the last releases it was necessary to provide a simple way to upgrade Pootle using a single command. The old procedure is still available, mostly for debugging failing upgrades, but the new procedure is now the preferred one.

Simplified database upgrade**Warning:** Always make database backups before running any upgrades.

This is now the preferred way to upgrade the database.

The procedure is easy, just run:

```
$ pootle setup
```

Step by step database upgrade**Warning:** Always make database backups before running any upgrades.

Note: Use this procedure only if the *Simplified database upgrade* procedure doesn't work for you.

The step by step database upgrade procedure lets you control the upgrade process and tweak it. This is useful for debugging purposes.

Note: If you are upgrading from a Pootle version older than 2.5.0, you will need an extra step at the beginning (before running `syncdb --noinput`):

```
$ pootle updatedb
```

The *updatedb command* upgrades the database schema to the state of Pootle 2.5.0. This is necessary due to the changes made to the database schema migration mechanisms after the 2.5.0 release.

In the first step, the `syncdb` command will create any missing database tables that don't require any migrations:

```
$ pootle syncdb --noinput
```

For this specific version (Pootle 2.5.1), and due to Pootle's transitioning to South, you will need to run a fake migration action in order to let South know which is your current database schema. You can execute the fake migration by running the following commands:

```
$ pootle migrate pootle_app --fake 0001
$ pootle migrate pootle_language --fake 0001
$ pootle migrate pootle_notifications --fake 0001
$ pootle migrate pootle_profile --fake 0001
$ pootle migrate pootle_project --fake 0001
$ pootle migrate pootle_statistics --fake 0001
$ pootle migrate pootle_store --fake 0001
$ pootle migrate pootle_translationproject --fake 0001
```

Note: If you are upgrading from Pootle 2.5.0 you also have to run:

```
$ pootle migrate staticpages --fake 0001
```

The next step will perform any pending schema migrations. You can read more about the *migrate command* in South's documentation.

```
$ pootle migrate
```

Lastly, the *upgrade command* will perform any extra operations needed by Pootle to finish the upgrade and will record the current code build versions for Pootle and the Translate Toolkit. Before running this command please check if you are interested on running it using any of its available flags.

```
$ pootle upgrade
```

Custom Changes

If you made any changes to Pootle code, templates or styling, you will want to ensure that your upgraded Pootle contains those changes. How hard that is will depend entirely on the details of these changes.

Changes made to the base template are likely to work fine, but changes to details will need individual inspection to see if they can apply cleanly or have to be reimplemented on the new version of Pootle.

Since Pootle 2.5 [customization of style sheets and templates](#) have become much easier to separate from the standard code. If you are migrating to Pootle 2.5+ then use this opportunity to move your code to the correct customization locations.

Database Migration

The default configuration for Pootle uses SQLite which is not really suited for production use. If you started using SQLite and want to migrate to another database system such as MySQL or PostgreSQL without recreating existing data, you can perform a database migration using the steps described on this page.

Note: A database migration is possible since Pootle 2.1.1. It is possible to do the database migration using the version 2.0.6, which was specifically added to allow database migration. This migration will only work with Django 1.2 or later.

Detailed Migration Process

Note: Pootle 2.1.x and 2.5.x database can be very large. Dumping and loading data will take long and will require lots of RAM. If you have a 2.0 install it is better to migrate the database first and then upgrade to 2.5, since the 2.0 database is relatively small.

The steps to migrate between databases are as follows:

1. Make **complete backups** of all your files, settings and databases.
2. Ensure that you have:
 - (a) At least Pootle 2.0.* or Pootle 2.1.*.
 - (b) At least Django 1.2.0.
3. Don't change the version of Pootle at any stage during this process.
4. Read about how to run [management commands](#).
5. Stop the Pootle server to avoid data changing while you migrate.
6. Leave current settings intact.
7. Dump the data to a JSON file using the **dumpdata** command. Note the `-n` option.

```
$ pootle dumpdata -n > data.json
```

8. Create a new database for Pootle.
9. Change *your settings* to point at this new database.

10. Initialize the new database using the **syncdb** command.

Note: Sometimes not removing records introduced by **syncdb** can create problems. So if you experience any failure during **loaddata** execution that can't be solved by any other mean, then remove all the records from the new database while keeping the tables intact.

11. Load the data from the JSON file in the new database using the **loaddata** command:

```
$ pootle loaddata ./data.json
```

Note: If you experience any problem during **loaddata** execution, you may find it helps to instead export the data with:

```
$ pootle dumpdata > data.json
```

avoiding the use of `-n` or `--natural` options.

12. Restart the server; you should be running under the new database now.

Note: Some other problems reported during database migration may be solved by commenting all signal calls in Pootle code during the database migration process. If you do so, remember to uncomment them after the migration.

1.3.3 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.

If instead of an installation you deployed Pootle straight from the git repository, you can either set the `POOTLE_SETTINGS` environment variable or put a file under the `pootle/settings/` directory. Note that the files in this directory are read in alphabetical order, and **creating a 90-local.conf file is recommended** (files ending in `-local.conf` will be ignored by git).

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.

DESCRIPTION Description of the Pootle server.

TITLE The name of the Pootle server.

20-backends.conf Backend and caching settings.

OBJECT_CACHE_TIMEOUT Default: 2500000

Time in seconds the Pootle's statistics cache will last.

POOTLE_TOP_STATS_CACHE_TIMEOUT Default: 86400

Time in seconds the Pootle's top statistics cache will last.

30-site.conf Site-specific settings.

CAN_CONTACT Default: True

Controls whether users will be able to use the contact form. The address to receive messages is controlled by `CONTACT_EMAIL`.

CAN_REGISTER Default: True

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

CONTACT_EMAIL Default: `info@YOUR_DOMAIN.com`

Address to receive messages sent through the contact form. This will only have effect if `CAN_CONTACT` is set to `True`.

40-apps.conf Configuration settings for applications used by Pootle.

API_LIMIT_PER_PAGE Default: 0

New in version 2.5.1.

Number of records Pootle API will show in a list view. 0 means no limit.

CUSTOM_TEMPLATE_CONTEXT Default: `{ }`

New in version 2.5.

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

EMAIL_SEND_HTML Default: `False`

By default Pootle sends only text emails. If your organization would prefer to send mixed HTML/TEXT emails, set this to `True`, and update `activation_email.txt` and `activation_email.html` in the `templates/registration/` directory.

Note: Password reset emails will still be sent in plain text. This is a limitation of the underlying system.

FUZZY_MATCH_MAX_LENGTH Default: 70

New in version 2.5.

Maximum character length to consider when doing fuzzy matching. The default might not be enough for long texts. Please note this affects all fuzzy matching operations, so bear in mind this might affect performance.

FUZZY_MATCH_MIN_SIMILARITY Default: 75

New in version 2.5.

Minimum similarity to consider when doing fuzzy matching. Please note this affects all fuzzy matching operations, so bear in mind this might affect performance.

LEGALPAGE_NOCHECK_PREFIXES Default: ('/accounts', '/admin', '/api', '/contact', '/django_admin', '/jsi18n', '/pages',)

New in version 2.5.1.

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.

MIN_AUTOTERMS Default: 60

When building the terminology, the minimum number of terms that will be automatically extracted.

MARKUP_FILTER Default: (None, {})

New in version 2.5.

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

- Accepted values for the first element are `textile`, `markdown`, `restructuredtext` and `None`
- The second element should be a dictionary of keyword arguments that will be passed to the markup function

Examples:

```
MARKUP_FILTER = (None, {})  
  
MARKUP_FILTER = ('markdown', {'safe_mode': 'escape'})  
  
MARKUP_FILTER = ('restructuredtext', {'settings_overrides': {  
                                     'report_level': 'quiet',  
                                     }  
})
```

MAX_AUTOTERMS Default: 600

When building the terminology, the maximum number of terms that will be automatically extracted.

POOTLE_ENABLE_API Default: False

New in version 2.5.1.

Enable Pootle API.

TASTYPIE_DEFAULT_FORMATS Default: ['json']

New in version 2.5.1.

List defining the allowed serialization formats for Pootle API. Check [Tastypie docs](#) for all the available formats and [its dependencies](#) (see in Installation section).

TOPSTAT_SIZE Default: 5

The number of rows displayed in the top contributors table.

USE_CAPTCHA Default: True

Enable spam prevention through a captcha.

51-ldap.conf Optional LDAP configuration settings. To enable the LDAP authentication backend, you'll need to append `'pootle.core.auth.ldap_backend.LdapBackend'` to the list of `AUTHENTICATION_BACKENDS`.

AUTH_LDAP_ANON_DN Default: ''

Anonymous credentials: Distinguished Name.

AUTH_LDAP_ANON_PASS Default: ''

Anonymous credentials: password.

AUTH_LDAP_BASE_DN Default: ''

Base DN to search

AUTH_LDAP_FIELDS Default: {'dn': 'dn'}

A mapping of Pootle field names to LDAP fields. The key is Pootle's name, the value should be your LDAP field name. If you don't use the field or don't want to automatically retrieve these fields from LDAP comment them out. The only required field is `dn`.

AUTH_LDAP_FILTER Default: ''

What are we filtering on? `%s` will be the username, for example `'sn=%s'`, or `'uid=%s'`.

AUTH_LDAP_SERVER Default: ''

The LDAP server. Format: `protocol://hostname:port`

60-translation.conf Translation environment configuration settings.

AMAGAMA_URL Default: `http://amagama.locamotion.org/tmserver/`

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.

AUTOSYNC Default: `False`

Set this to `True` if you want translation files to be updated immediately.

Note: This negatively affects performance and should be avoided unless another application needs direct access to the files.

Warning: This feature is not maintained anymore, use it at your own risk.

EXPORTED_DIRECTORY_MODE Default: `0755`

On POSIX systems, exported directories will be assigned this permission. Use `0755` for publically-readable directories or `0700` if you want only the Pootle user to be able to read them.

EXPORTED_FILE_MODE Default: `0644`

On POSIX systems, exported files will be assigned this permission. Use `0644` for publically-readable files or `0600` if you want only the Pootle user to be able to read them.

LIVE_TRANSLATION Default: `False`

Live translation means that the project called *Pootle* is used to provide the localized versions of Pootle. Set this to `True` to enable live translation of Pootle's UI. This is a good way to learn how to use Pootle, but it has high impact on performance.

LOOKUP_BACKENDS Default: ['wikipedia'] (Wikipedia enabled)

Enables backends for web-based lookups.

Available options: wikipedia.

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:

APERTIUM: Apertium service. For this service you need to set the API key. Get your key at <http://api.apertium.org/register.jsp>

GOOGLE_TRANSLATE: Google Translate service. For this service you need to set the API key. Note that Google Translate API is a paid service. See more at <https://developers.google.com/translate/v2/pricing>

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).

PODDIRECTORY Default: `working_path('po')`

The directory where the translation files are kept.

VCS_DIRECTORY Default: `working_path('repos')`

New in version 2.5.

The directory where version control clones/checkouts are kept.

Deprecated Settings

ENABLE_ALT_SRC Default: True

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.

Display alternate source languages in the translation interface.

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-python MySQL adapter for Python.

Psycopg2 PostgreSQL adapter for Python.

Caching Fast and efficient caching avoids hitting the DB when it's not really needed. Adjust the `CACHES` setting accordingly.

python-memcached Efficient caching.

Indexing Engines Installing an `indexing engine` will speed-up searches. Pootle will automatically pick one from any of the available engines.

Xapian Python bindings for Xapian ¹.

PyLucene Python bindings for Lucene.

Note

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

zip and unzip Fast (un)compression of file archives.

python-Levenshtein Provides speed-up when updating against templates.

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.

python-ldap Enables *LDAP authentication*. Be sure to check the *LDAP settings*.

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 `memcached` and enable it in the settings file.

¹ Xapian versions before 1.0.13 are incompatible with Apache; Pootle will detect Xapian version and disable indexing when running under `mod_wsgi` if needed.

Checking for Xapian relies on the `xapian-check` command, which is found in the `xapian-tools` package in Debian-based systems.

- 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 `LIVE_TRANSLATION` is disabled.
- Ensure `DEBUG` mode is disabled.
- Ensure that the `zip` and `unzip` commands are installed on your server. These can improve the performance during upload and download of large ZIP files.
- Ensure that you have an [indexing engine](#) installed with its Python bindings. This will improve the performance of searching in big projects. PyLucene should perform better, although it might be harder to install.
- Ensure that you have `python-levenshtein` installed. This will improve the performance when updating against templates.
- 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.

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 is well tested and recommended. You can [migrate your current database](#) if you already have data you don't want to lose.

If using MySQL backend, for smaller installations it is suggested to go with [MyISAM backend](#) (which might result in smaller memory usage and better performance). If high concurrency is expected, [InnoDB](#) is suggested to avoid locking issues.

Fast PO implementation If you want better performance for your PO based operations, you can try to enable the fast PO implementation. This implementation will be used if `USECPO=2` is available in the operating system environment variables. Note that this is different from the web server's environment variables.

Your PO files will have to have character encodings specified, and be perfectly valid PO files (no duplicate messages or other format errors). Be sure to test this extensively before you migrate your live server to this setup.

Caching System

Pootle uses a caching system to improve performance. It is an essential part of [optimizing](#) 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.

Cache Backends

Django supports [multiple cache backends](#) (methods of storing cache data). You can specify which backend to use by overriding the value of `CACHES` in your configuration file.

Memcached

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

[Memcached](#) is the **recommended cache backend**, it provides the best performance. And works fine with multiprocessing servers like Apache. It requires the *python-memcached* package and a running memcached server. Due to extra dependencies it is not enabled by default.

Memcached on Unix sockets

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': 'unix:/path/to/memcached.sock',
    }
}
```

If you don't want Pootle using TCP/IP to access memcached then you can use Unix sockets. This is often a situation in hardened installations using SELinux.

You will need to ensure that memcached is running with the `-s` option:

```
$ memcached -u nobody -s /path/to/memcached.sock -a 0777
```

Database

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
    }
}
```

[Database caching](#) relies on a table in the main Pootle database for storing the cached data, which makes it suitable for multiprocessing servers, with the added benefit that the cached data remains intact after a server reboot (unlike memcached) but it is considerably slower than memcached.

Changed in version 2.1.1.

This is the default cache backend. On new installs and upgrades the required database will be created.

Users of older versions need to create the cache tables manually if they would like to switch to the database cache backend using this [management command](#):

```
$ pootle createcachetable pootlecache
```

Text indexing for Pootle

Pootle provides [searching](#) functionality, which is a great way to do searches over all files in a project. If there are many strings to search through, then performance can be slow, but installing a text indexing library will speed things up, allowing searching even in very large projects.

Supported indexing engines

The following indexing engines are supported:

- [Lucene](#): This should be the fastest, but is not packaged for many Linux distributions, and is a bit harder to build.
- [Xapian](#) (v1.0 or higher): Note that you need at least [version 1.0.13](#) to run under Apache with `mod_wsgi` or `mod_python`.

Pootle's usage of the indexing engine

The indexing database helps to speed up [search queries](#) performed from the Pootle interface.

Installation

If you want to use an indexing engine to speed up text searches, then you need to install one of the supported indexing engines and its Python binding.

The indexing engine will be used, as soon as the required Python bindings are available.

See below for details.

Lucene

- Install the [PyLucene](#) package
 - For debian: follow this [Howto](#)

Xapian

- Install the [Python bindings for Xapian](#)
 - Debian: `apt-get install python-xapian xapian-tools`
 - [Other distributions and platforms](#)

The Xapian tools packaged is required for the *xapian-check* command which is used to determines whether the Xapian version is compatible with Pootle.

Note: If you are deploying using a virtualenv and want to make use of Xapian then you will need to, either:

1. Install your virtualenv with access to the system packages using the `--system-site-packages` option:

```
virtualenv --system-site-packages ENV
```

2. [Compile the Python bindings](#)

3. Symbolically link the Xapian bindings into your virtualenv, as follows on an Ubuntu system:

```
$ mkdir ${ENV}/lib/python2.6/dist-packages
$ cd ${ENV}/lib/python2.6/dist-packages
$ ln -s /usr/lib/python2.6/dist-packages/xapian.py
$ ln -s /usr/lib/python2.6/dist-packages/_xapian.so
```

Debugging

If you want to check which indexing engines are currently detected on your system you can execute the self-tests of the indexing engine interface of Pootle:

```
python translate/search/indexing/test_indexers.py
```

This will display the installed engines and check if they work as expected.

Note: Please file a [bug report](#) if you encounter any errors when running these tests.

The actual test for xapian is `xapian-check --version`.

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:

```
$ pootle help
```

Note: If you run Pootle from a repository checkout you can use the *manage.py* file found in the root of the repository.

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.

The Translate Toolkit offers a bundled CherryPy server but there are many more options such as gunicorn, flup, paste, etc.

run_cherrypy New in version 2.5.

This command runs the CherryPy server bundled with the Translate Toolkit.

Available options:

--host The hostname to listen on.

Default: 127.0.0.1.

--port The TCP port on which the server should listen for new connections.

Default: 8080.

--threads The number of working threads to create.

Default: 1.

--name The name of the worker process.

Default: `socket.gethostname()`.

--queue Specifies the maximum number of queued connections. This is the the `backlog` argument to `socket.listen()`.

Default: 5.

--ssl_certificate The filename of the server SSL certificate.

--ssl_privatekey The filename of the server's private key file.

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.

All commands in this category accept a `--directory` command line option that limits its action to a path relative to the `po/` directory.

Changed in version 2.1.2.

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.

If you need to limit the commands to certain files or subdirectories you can use the `--path-prefix` option, path should be relative to project/language pair.

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

```
$ pootle refresh_stats --project=tutorial
```

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

```
$ pootle refresh_stats --project=tutorial --language=zu --language=eu
```

refresh_stats This command will go through all existing projects making sure calculated data is up to date. Running `refresh_stats` immediately after an install, upgrade or after adding a large number of files will make Pootle feel faster as it will require less on-demand calculation of expensive statistics.

`refresh_stats` will do the following tasks:

- Update the statistics cache (this only useful if you are using memcached).
- Calculate quality checks so that they appear on the expanded overview page without a delay.
- Update [full text search index](#) (Lucene or Xapian).

sync_stores This command will save all translations currently in the database to the file system, thereby bringing the files under the `PODIRECTORY` directory in sync with the Pootle database.

Note: For better performance Pootle keeps translations in database and doesn't save them to disk except on demand (before file downloads and major file level operations like version control updates).

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 database but not yet saved to disk.

When the `--overwrite` option is specified, the sync operation will not be conservative and it will overwrite the existing files on disk, making strings obsolete and updating the file's structure.

With the `--skip-missing` option, files that are missing on disk will be ignored, and no new files will be created.

New in version 2.5.

With the `--modified-since` option it is possible to give a change identifier (from the output of *latest_change_id*) to specifically indicate which changes need to be synced to disk. This will override Pootle on what has/hasn't been synced to disk, and specifically those changes will be synced. Note that bulk changes (from uploads and version control actions) don't yet record fine-grained changes, and these will therefore not be synced to disk. However, these should already be on disk, since those actions always sync to disk anyway.

update_stores This command is the opposite of *sync_stores*. It will update the strings in database to reflect what is on disk, as Pootle will not detect changes in the file system on its own.

It will also discover and import any new files added to existing languages within the projects.

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

`update_stores` has an extra command line option `--keep` that will prevent it from overwriting any existing translation in the database, thus only updating new translations, removing obsolete strings and discovering new files and strings.

Changed in version 2.5.1.

Note that `--keep` doesn't keep obsolete units around anymore, they are either deleted in case the string is untranslated or marked as obsolete in case the string was translated.

Changed in version 2.5.

Along with `--keep`, the `--modified-since` option can be used to control the set of translations that will be updated: translations with a change ID **greater than** the given value will be kept.

To illustrate the results of these two options, the following table emulates the behavior of a `pootle update_stores --modified-since=5 --keep` run:

File on disk	DB before (change ID)	DB after (result)
New string appeared in existing file	<none>	String added
Existing string changed in existing file	<none>	String updated
Existing string changed in existing file	2	String updated
Existing string changed in existing file	5	String updated
Existing string changed in existing file	8	String kept
New string in a new file	<none>	String added
String removed from the file	3	String removed
String removed from the file	10	String removed
File removed	4	Strings removed
File removed	12	Strings removed

By default, `update_stores` will only update files that appear to have changed on disk since the last synchronization with Pootle. To force all files to update, specify `--force`.

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

update_against_templates Changed in version 2.5: The name of the command has been renamed from `update_from_templates`.

Updates languages to match what is present in the translation templates.

This command is essentially an interface to the Translate Toolkit command `pot2po` with special Pootle specific routines to update the database and file system to reflect the latest version of translation templates for each language in a project.

During the process, translations existing in the database will first be synced to disk (only in bilingual formats), then they will be updated against the latest templates and after that the database will also be updated to reflect the latest changes.

When updating existing translated files under a given language, the command will retain any existing translations, fuzzy matching is performed on strings with minor changes, and unused translations will be marked as obsolete. New template files will initialize new untranslated files.

It is unlikely you will ever need to run this command for all projects at once. Use the `--directory`, `--project` or `--language` command line options to be specific about the project, language or project/language pair you want to target.

Warning: If the template files are corrupt translations might be lost. If you generate templates based on a script make sure they are in good shape.

update_translation_projects This command scans project directories looking for files matching languages not added to the project then adds them. It basically repeats the discovery process done by Pootle when you create a new project.

Using the `--cleanup` command line option, languages added to projects that no longer have matching files on the filesystem will be deleted.

update_from_vcs New in version 2.5.

This command updates the specified files from their [Version Control System\(s\)](#). It supports the `--directory`, `--project`, and `--language` parameters.

Pootle will take care to avoid version control conflicts, and will handle any conflicts on a string level, just like it would if the update was done through the web front-end.

The command first syncs database contents to disk.

commit_to_vcs New in version 2.5.

This command commits the specified files to their [Version Control System\(s\)](#). It supports the `--directory`, `--project`, and `--language` parameters.

A file needs to be up to date, otherwise the commit will fail. Files can be updated inside Pootle, or using the `update_from_vcs` command. This is not done automatically, otherwise the merged version of the file will be committed without review without anybody knowing.

list_languages New in version 2.5.

This command prints all the language codes on the server. This might be useful for automation.

Accepts the `--modified-since` parameter to list only those languages modified since the change id given by `latest_change_id`.

The option `--project` limits the output to one or more projects. Specify the option multiple times for more than one project.

list_projects New in version 2.5.

This command prints all the project codes on the server. This might be useful for automation.

Accepts the `--modified-since` parameter to list only those projects modified since the change id given by *latest_change_id*.

latest_change_id New in version 2.5.

This command prints the ID of the latest change (submission) made on the server. This is mostly useful in combination with other commands that operate with these IDs.

Goals

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

add_project_goals This command allows you to create **project goals** for a given project reading them from a phaselist file.

Such file has several lines where each line consists on two fields separated by a tab. The first field specifies a goal name and the second one is the path of a file:

user1	./browser/branding/official/brand.dtd.pot
other	./browser/chrome/browser/aboutCertError.dtd.pot
user1	browser/chrome/browser/aboutDialog.dtd.pot
user2	browser/chrome/browser/aboutSessionRestore.dtd.pot
developer	./browser/chrome/browser/devtools/appcacheutils.properties.pot
developer	browser/chrome/browser/devtools/debugger.dtd.pot
user2	browser/chrome/browser/downloads/downloads.dtd.pot
user3	browser/chrome/browser/engineManager.dtd.pot
install	browser/chrome/browser/migration/migration.dtd.pot
install	./browser/chrome/browser/migration/migration.properties.pot

The goals are created if necessary. If the goal exists and has any relationship to any store, that relationships are deleted to make sure that the goals specified on the phaselist file are only applied to the specified stores.

After all goals are created then they are tied to the files on template translation project for the project as they are specified on the phaselist file. If any specified file does not exist for the template translation project on the given project then it is skipped.

This command has two mandatory options: `--project` and `--filename`.

```
$ pootle add_project_goals --project=tutorial --filename=phaselist.txt
```

Manually Installing Pootle

These commands expose the database installation and upgrade process from the command line.

setup New in version 2.5.1.

This command either initializes a new DB or upgrades an existing DB, as required.

syncdb Originally, `syncdb` was a generic Django management command that creates empty database tables. It has been customized for Pootle to create everything required for a bare bones install for releases up to 2.5.0. This includes database tables, default permissions, some default objects used internally by Pootle (like the “*default*” and “*nobody*” user profiles) and the special *Terminology* project and *Templates language*.

For releases up to 2.5.0, if you just run `syncdb` you will have a usable Pootle install but you will need to create all languages manually, and you will not have a tutorial project to play with. For releases after 2.5.0, `syncdb` is not sufficient to create the database schema; it will remain incomplete and unusable until you apply all migrations to the database schema by running the *migrate* command.

migrate New in version 2.5.1.

Note: Since the addition of the *setup* management command it is not necessary to directly run this command. Please refer to the *Upgrading* or *Installation* instructions to see how to run the `setup` management command in those scenarios.

This is South’s *migrate command*, which applies migrations to bring the database up to the latest schema revision. It is required for releases after 2.5.0, even for a fresh install where you are not upgrading from a previous release.

initdb This is Pootle’s install process, it creates the default *admin* user, populates the language table with several languages with their correct fields, initializes several terminology projects, and creates the tutorial project.

`initdb` can only be run after *syncdb* and *migrate*.

Note: `initdb` will not import translations into the database, so the first visit to Pootle after `initdb` will be very slow. **It is best to run *refresh_stats* immediately after `initdb`.**

updatedb Changed in version 2.5.1.

This is a command line interface to Pootle’s database schema upgrade process.

This will only perform schema upgrades to version 2.5 from Pootle versions older than 2.5. To upgrade to version 2.5.1 and later South’s *migrate command* must be used, after upgrading to version 2.5.

For detailed instructions on upgrading, read the *Upgrading* section of the documentation.

upgrade New in version 2.5.1.

Performs post schema upgrade actions that are necessary to leave all the bits in place. It also serves as a trigger for any changes needed by Translate Toolkit version upgrades.

Optionally, the command accepts the `--calculate-stats` flag, which will calculate full translation statistics after doing the upgrade.

Also, the `--flush-checks` flag forces flushing the existing quality checks. This is useful when new quality checks have been added or existing ones have been updated, but take into account that **this operation is very expensive**.

For detailed instructions on upgrading, read the *Upgrading* section of the documentation.

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.

Useful Django commands

changepassword

```
$ pootle changepassword <username>
```

This can be used to change the password of any user from the command line.

createsuperuser This creates a new admin user. It will prompt for username, password and email address.

dbshell This opens a database command prompt with the Pootle database already loaded. It is useful if you know SQL.

Warning: Try not to break anything.

shell This opens a Python shell with the Django and Pootle environment already loaded. Useful if you know a bit of Python or the Django models syntax.

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 *refresh_stats* command daily at 02:00 AM:

```
00 02 * * * www-data /var/www/sites/pootle/manage.py refresh_stats
```

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.

1.4 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.

1.4.1 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 on IRC in the #pootle channel in FreeNode.
- Found a bug? Report it in our [Bugzilla tracker](#). If you don't want to create an account you can always contact us on IRC. 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 #pootle?
 - 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 IRC on #pootle 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 our [mailing list](#) or on IRC.

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.

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.

1.4.2 Pootle Development Roadmap

This is the Pootle roadmap for the next few iterations. Don't look here for small improvements, we're only tracking larger bits of work.

Estimated release April 2014

- Move to Django 1.5/1.6 – remove anything keeping us on Django 1.4.
- Live cross project Translation Memory.
- Stats speedup – work on Stats speedups.
- Concordance searching.
- amaGama – automate updating of resources.
- Translation editor improvements:
 - [Highlight placeable](#) – terms, variables and other things in source text and allow them to be copied easily using the keyboard.
 - Live Quality Assurance checks – at the moment these happen after the translation editor has left the unit, performing them while editing will help to reduce errors.
- Developer centric changes:
 - Adding a UI test framework.
 - Automatic tests for most important parts of Pootle to prevent the risk of regressions.
- Mozilla specific features:
 - Proper plural forms handling in Pootle for Firefox Desktop.
 - Integration of compare-locale errors to the translator error page.

- Contributions by a translator to a given project and language.

Estimated release October 2014

- Substring matching in TM.
- Variable abstraction so that we can leverage translations from other projects that might not match because of differences in variables placeable e.g. `%s` vs `&brandShortName;`.
- Management statistical reporting – project, language and user statistical reporting.
- A dashboard (health report) that allows 110n managers to check on the health of a language.
- Social interventions:
 - Social sharing of projects, strings, etc for community building and community input.
 - Social/Persona authentication to make it easier for users to login and contribute.
 - OpenBadges – implement badges to reward team members contributions.
- Team review of translations.
- Easing team management:
 - Improve our rights display.
 - Request a new language.
 - Request to join a translation team.

Sometime in the future

Things we'd love to do sooner but they are hard or need a sponsor.

- Get rid of actions for pushing, merging and retrieving translations. Do these actions in the background with no human intervention at all to reduce errors, improve scale.
- Manage all setup from version control files.
- Monolingual files – make Pootle work more reliably directly on monolingual files.

1.4.3 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 IRC 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.

Setting up the development environment

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. Installing `virtualenvwrapper` will pull in `virtualenv` as a dependency.

```
$ 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 in your `.bashrc` file (or whatever file your shell uses for initializing user customizations).

Now the commands provided `virtualenv` and `virtualenvwrapper` are available, so we can start creating our virtual environment.

```
$ mkvirtualenv <env-name>
```

Replace `<env-name>` 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 2.6 interpreter by passing the `-p python2.6` option.

Note: After running `mkvirtualenv`, the newly created environment is activated. To deactivate it just run:

```
(env-name) $ deactivate
```

To activate a virtual environment again simply run:

```
$ workon <env-name>
```

Time to clone Pootle's source code repository. The main repository lives under [translate/pootle in GitHub](#). If you have a GitHub account, the best idea is to fork the main repository and to clone your own fork for hacking. Once you know which way you want to continue forward, just move to a directory where you want to keep the development files and run `git clone` by passing the repository's URL.

```
(env-name) $ git clone https://github.com/translate/pootle.git
```

This will create a directory named `pootle` where you will find all the files that constitute Pootle's source code.

Note: If you have a GitHub account, fork the main `translate/pootle` repository and replace the repository URL by your own fork.

Before running the development server, it's necessary to install the software dependencies/requirements by using `pip`. For this matter there are some `pip requirements` files within the `requirements` directory. We will install the requirements defined in `requirements/dev.txt`, which apart from the minimum will pull in some extras that will ease the development process.

```
(env-name) $ cd pootle
(env-name) $ pip install -r requirements/dev.txt
```

Note: Some dependencies might need to build or compile source code in languages other than Python. You may need to install extra packages on your system in order to complete the build process and the installation of the required packages.

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 vastly differ from those used in production environments.

For that purpose there is a sample configuration file with settings adapted for development scenarios, `pootle/settings/90-dev-local.conf.sample`. Copy this file and rename it by removing the `.sample` extension:

```
(env-name) $ cp pootle/settings/90-dev-local.conf.sample pootle/settings/90-dev-local.conf
```

Note: To learn more about how settings work in Pootle head over the [Settings](#) section in the documentation.

Once the configuration is in place, you'll need to setup the database schema and add initial data.

```
(env-name) $ python manage.py syncdb --noinput
(env-name) $ python manage.py migrate
(env-name) $ python manage.py initdb
```

Finally, just run the development server.

```
(env-name) $ python manage.py runserver
```

Once all is done, you can start the development server anytime by enabling the virtual environment (using the **workon** command) and running the **manage.py runserver** command.

Happy hacking!!

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 bug.

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 Bugzilla, 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
commits ef4517ab and abc361fd.

Fixes bug #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.
```

1.4.4 Customizing the look

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. Before you start editing the CSS of Pootle, have a look at our [styling guidelines for developers](#).

Custom changes are kept separate from the distributed files, so that upgrades are unlikely to affect your customizations.

Rebuilding assets after customization

Warning: After doing any customization, please execute the following commands to collect and build static content such as images, CSS and JavaScript files that are served by Pootle server.

```
$ python manage.py collectstatic --noinput --clear
$ python manage.py assets build
```

Customizing CSS

Edit the file in `static/css/custom/custom.css` to override any rules from the main CSS file. That CSS file will be included in every page.

Customizing images

Any custom images can be placed in `static/css/custom/`. The `custom.css` file can refer to it directly by name, including any paths relative to `static/css` directory, for example: `url('custom/image.png')` to refer to `static/css/custom/image.png`.

Customizing the favicon

The favicon can be customized by editing the base template directly (`templates/base.html`). This has the downside that you have to reimplement this on upgrades if the base template is replaced. Alternatively the base template can be overridden as a whole with the favicon customized to your needs (see the next section).

Customizing templates

In case you need to change a template, copy it into `templates/custom/` with the same name as it had before. Make sure that you have a complete copy of the template and then make any changes you require.

If you edit any templates, keep in mind that any changes to the text could result in untranslated text for users of the non-English user interface.

On upgrades, it would be ideal to ensure that any changes to the distributed templates are reflected in your customized versions of them, to ensure all features and improvements are present.

1.4.5 Testing

Warning: This page needs expanding and updating.

This page contains notes about Pootle's unit tests and how they should be used, interpreted and expanded. See the [Translate Toolkit testing docs](#) for notes on writing tests.

Pootle's unit tests use the Django testing framework, and can be executed with:

```
$ python manage.py test pootle_store pootle_app pootle_translationproject
```

Although you can run tests for all applications, several of the external applications are not passing their tests which renders this less useful.

Tests could be run with `py.test` using [pytest-django](#) or alternately by adding a [django-pytest](#) app to Pootle (conceivably both could be done) – however this is not currently supported or implemented.

1.4.6 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*.

1.4.7 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

1.4.8 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.

```
from __future__ import absolute_import

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.auth.models import User
from django.db import models
from django.db.models import Q
from django.db.models.signals import post_save

from profiles.views import edit_profile
from tastypie import fields

from pootle.core.decorators import permission_required
from pootle_store.models import (FUZZY, TRANSLATED, UNTRANSLATED, Store,
                                Unit, count_words)
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
- All @cached_property 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. Avoid underscores at all costs.
 - * 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 cases, like URLs on `pootle_app` app):

- * {app} is the app name, which sometimes can be shortened, e.g. using **tp** to avoid the longish **translationproject**. If either a shortened app name or a full one is being used, the chosen app name must be used consistently across all the URL patterns for the app. The only exception to this are AJAX URL patterns which can use a different value for {app}, that must be consistently used among all the AJAX URL patterns in 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.

```
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'),
)
```

Settings naming Pootle specific settings must be named like `POOTLE_*`, for example: `POOTLE_ENABLE_API`, `POOTLE_VCS_DIRECTORY` or `POOTLE_MARKUP_FILTER`

Pootle-specific markup

For documenting several things, Pootle defines custom Sphinx roles.

- Settings:


```
.. setting:: PODIRECTORY
```

To link to a setting, use `:setting: 'PODIRECTORY'`.

- 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.

JavaScript

There are no “official” coding style guidelines for JavaScript, so based on several recommendations (1, 2, 3) we try to stick to our preferences.

Indenting

- We currently use 2-space indentation. Don't use tabs.
- Avoid lines longer than 80 characters. When a statement will not fit on a single line, it may be necessary to break it. Place the break after an operator, ideally after a comma.

Whitespace

- If a function literal is anonymous, there should be one space between the word `function` and the `(` (left parenthesis).
- In function calls, don't use any space before the `(` (left parenthesis).
- Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.
- Each `;` (semicolon) in the control part of a `for` statement should be followed with a space.
- Whitespace should follow every `,` (comma).

Naming

- Variable and function names should always start by a lowercase letter and consequent words should be CamelCased. Never use underscores.
- If a variable holds a jQuery object, prefix it by a dollar sign `$`. For example:

```
var $fields = $('js-search-fields');
```

Selectors

- Prefix selectors that deal with JavaScript with `js-`. This way it's clear the separation between class selectors that deal with presentation (CSS) and functionality (JavaScript).
- Use the same naming criterion as with CSS selector names, ie, lowercase and consequent words separated by dashes.

Control statements Control statements such as `if`, `for`, or `switch` should follow these rules:

- The enclosed statements should be indented.
- The `{` (left curly brace) should be at the end of the line that begins the compound statement.
- The `}` (right curly brace) should begin a line and be indented to align with the beginning of the line containing the matching `{` (left curly brace).
- Braces should be used around all statements, even single statements, when they are part of a control structure, such as an `if` or `for` statement. This makes it easier to add statements without accidentally introducing bugs.
- Should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

String

- A string literal should be wrapped in single quotes.
- `join` should be used to concatenate pieces instead of `+` because it is usually faster to put the pieces into an array and join them.

Number

- `radix` should be specified in the `parseInt` function to eliminate reader confusion and to guarantee predictable behavior.

Examples

- if statements

```
if (condition) {
    statements
}

if (condition) {
    statements
} else {
    statements
}

if (condition) {
    statements
} else if (condition) {
    statements
} else {
    statements
}
```

- for statements

```
for (initialization; condition; update) {
    statements;
}

for (variable in object) {
    if (condition) {
        statements
    }
}
```

- switch statements

```
switch (condition) {
    case 1:
        statements
        break;

    case 2:
        statements
        break;

    default:
        statements
}
```

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, its name must start with `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`.

CSS

Indenting

- Indent using 4 spaces. Don't use tabs.
- Put selectors and braces on their own lines.
- Right-align the CSS browser-prefixed properties.

Good:

```
.foo-bar,
.foo-bar:hover
{
    background-color: #eee;
    -webkit-box-shadow: 0 1px 4px #d9d9d9;
    -moz-box-shadow: 0 1px 4px #d9d9d9;
    box-shadow: 0 1px 4px #d9d9d9;
}
```

Bad:

```
.foo-bar, .foo-bar:hover {
    background-color: #eee;
    -webkit-box-shadow: 0 1px 4px #d9d9d9;
    -moz-box-shadow: 0 1px 4px #d9d9d9;
    box-shadow: 0 1px 4px #d9d9d9;
}
```

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`.

1.4.9 Making a Pootle Release

These instructions are the guidelines for anyone making a Pootle commit.

Summary

1. `git clone git@github.com:translate/pootle.git pootle-release`
2. Create release notes
3. Adjust the roadmap
4. Up version number
5. Update translations
6. make build
7. Test install and other tests

8. Tag the release
9. Publish on PyPI
10. Upload to Sourceforge
11. Add product version to Bugzilla
12. Release documentation
13. Update translate website
14. Update Pootle dashboard
15. Unstage sourceforge
16. Announce to the world
17. Cleanup

Other possible steps

We need to check and document these if needed:

- Pre-release checks
- Build docs: we need to check if we need to build the docs for the release tarball.
- 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. We have produced
- Communicating to upstream packagers

Pre-release instructions

Upload and announce 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.

To make a new template:

```
make pot
```

And upload the templates to Pootle for translation. Update current translations against templates either on Pootle or in code and commits these updated files to Git.

Announce the new translations using these two channels:

1. The News tab on Pootle – for those not on any mailing list
2. The translate-pootle and translate-devel mailing lists – for those who might miss the news.

String freeze

We want to give a string freeze at least 2-4 weeks before a release. Announce that on the mailing lists.

If we do have a string freeze break then announce those to people.

A string freeze would normally run between an RC1 and a released version.

Detailed release instructions

Get a clean checkout and new virtualenv

We work from a clean checkout to ensure that everything you are adding to the build is what is in VC and doesn't contain any of your uncommitted changes. It also ensure that someone else could replicate your process.

```
git clone git@github.com:translate/pootle.git pootle-release
mkvirtualenv pootle-release
pip install -r requirements/build.txt
```

Create release notes

The release notes will be used in these places:

- Pootle website – [download page](#) (used in gh-pages)
- Sourceforge download – README.rst (used to give user info)
- Email announcements – text version

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 $version-1..HEAD > docs/release/$version.rst
```

Or a more specific example:

```
git log 2.5.0..HEAD > docs/releases/2.5.1.rst
```

Edit this new 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 the toolkit', not 'A new version of the toolkit 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
```

Adjust the roadmap

The roadmap file needs to be updated. Remove things that are part of this release. Adjust any version numbering if for example we're moving to Django 1.6 we need to change the proposed release numbers.

Look at the actual roadmap commitments and change if needed. These will remain during the lifetime of this version so it is good to adjust them before we branch.

Up version numbers

Update the version number in:

- `pootle/__version__.py`
- `docs/conf.py`

In `__version__.py`, bump the build number if anybody used the toolkit with the previous number, and there have been any changes to code touching stats or quality checks. An increased build number will force a toolkit user, like Pootle, to regenerate the stats and checks.

For `conf.py` change `version` and `release`

Note: FIXME – We might want to automate the version and release info so that we can update it in one place.

The version string should follow the pattern:

```
$MAJOR-$MINOR-$MICRO[-$EXTRA]
```

E.g.

```
1.10.0
0.9.1-rc1
```

`$EXTRA` is optional but all the three others are required. The first release of a `$MINOR` version will always have a `$MICRO` of `.0`. So `1.10.0` and never just `1.10`.

Update requirements versions

Update the minimum version number for the requirements in:

- `requirements/`
- `pootle/depcheck.py`

Update the requirements files:

```
make requirements
```

Note: I'm still not 100% why or if we need these, but until we work it out lets make sure we ship with correct files.

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:

```
$ make mo # Build all LINGUAS enabled languages
```

Build the package

Building is the first step to testing that things work. From your clean checkout run:

```
make mo-all # if we are shipping an pre-release
make build
```

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 install the new toolkit using:

```
mkvirtualenv pootle-testing
pip install path/to/dist/Pootle-$version.tar.bz2
```

This will allow you test installation of the software.

You can then proceed with other tests such as checking:

1. Quick installation check:

```
pootle init
pootle setup
pootle start
# browse to localhost:8000
```

2. Documentation is available
3. Installation documentation is correct
 - Follow the [installation](#) and [hacking](#) guides to ensure that they are correct.
4. Meta information about the package is correct. See pypi section of reviewing meta data.

To cleanup:

```
deactivate
rmvirtualenv pootle-testing
```

Tag the release

You should only tag once you are happy with your release as there are some things that we can't undo.

```
git tag -a 2.5.0 -m "Tag version 2.5.0"
git push --tags
```

If this is the final release then there should be a stable branch e.g. `stable/2.5.0`, so create one if it does not already exist.

Publish on PyPI

Publish the package on the [Python Package Index \(PyPI\)](#)

- [Submitting Packages to the Package Index](#)

Note: You need a username and password on <https://pypi.python.org> 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. A first run of `./setup.py register` will create such a file. It will also actually publish the meta-data so only do it when you are actually ready.

Review the meta data. This is stored in `setup.py`, use `./setup.py --help` to see some options to display meta-data. The actual long description is taken from `/README.rst`.

To test before publishing run:

```
make test-publish-pypi
```

Then to actually publish:

```
make publish-pypi
```

Copy files to sourceforge

Publishing files to the Translate Sourceforge project.

Note: You need to have release permissions on sourceforge to perform this step.

- <http://sourceforge.net/projects/translate/files/Pootle/>

You will need:

- Tarball of the release
 - Release notes in reStructured Text
1. Create a new folder in the [Pootle Sourceforge release folder](#) using the ‘Add Folder’ button. The folder name must be the same as the release name e.g. `2.5.0-rc1`. Mark this as being for staging for the moment.
 2. `make publish-sourceforge` will give you the command to upload your tarball and `README.rst`.
 - (a) Upload tarball for release.
 - (b) Upload release notes as `README.rst`.
 - (c) Click on the info icon for `README.rst` and tick “Exclude Stats” to exclude the `README` from stats counting.
 3. Check `README.rst`. Since this is generated on Sourceforge, without reference to the docs folder, some of the links will be broken.
 - (a) Check all links
 - (b) If broken links exist then download `README.rst` from Sourceforge, make changes and upload your adjusted version. Don’t change the version in `releases/` as we want that to continue to work correctly.

4. Final checks:

- (a) Check that the `README.rst` for the parent `Pootle` folder is still appropriate, this text is the text from `/README.rst`.
- (b) Check all the links in `README.rst` files for existing releases, new release and the parent folders.

Add product version to Bugzilla

We need to allow users to report issues against the released version.

1. In Administration->Products add a product version.
2. Review existing versions that are available and disable older version from accepting bug reports.

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/2.5.0` rather than `2.5.0` as that allows any fixes of documentation for the `2.5.0` release 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.

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. This is in Markdown format (for now), so we need to change the release notes `.rst` to `.md`, which mostly means changing URL links from `'`xxx <link> `_` to `[xxx] (link)`.
2. Change `$version` as needed. See `download.html`, `_config.yml` and `git grep $old_release`
3. `git commit` and `git push` – changes are quite quick so easy to review.

Note: FIXME it would be great if gh-pages accepted `.rst`, maybe it can if we prerender just that page?

Update Pootle dashboard

The dashboard used in Pootle's dashboard is updated in its own project:

1. `git clone git@github.com:translate/pootle-dashboard.git`
2. Edit `index.html` to contain the latest release info
3. Add the same info in `alerts.xml` pointing to the release in RTD `release/$version.html`

Do a `git pull` on the server to get the latest changes from the repo.

Unstage on sourceforge

If you have created a staged release folder, then unstage it now.

Announce to the world

Let people know that there is a new version:

1. Announce on mailing lists: Send the announcement to the translate-announce mailing lists on translate-announce@lists.sourceforge.net translate-pootle@lists.sourceforge.net
2. Adjust the #pootle channel notice. Use `/topic` to change the topic.
3. Email important users
4. Tweet about it

Cleanup

Some possible cleanup tasks:

- Remove any RC builds from the sourceforge download pages and add redirects to Sourceforge Pootle top level download page.
- Checkin any release notes and such (or maybe do that before tagging).
- Remove your pootle-release checkout.
- Remove pootle-release virtualenv: `deactivate; rmvirtualenv pootle-release`
- Update and change things based on what you learnt, don't wait:
 - Update and fix these release notes and make sure they are on `master`.
 - Discuss any changes that should be made or new things that could be added
 - Add automation if you can

1.5 Pootle API

Changed in version 2.5.1.

Pootle provides a REST API for interacting with it using external tools, allowing those to retrieve data, for example translation stats, or save data to Pootle, e.g. translations. This reference document is written for those interested in:

- Developing software to use this API
- Integrating existing software with this API
- Exploring API features in detail

1.5.1 Enabling the Pootle API

Pootle API is disabled by default. To enable it just install `django-tastypie` and put the following line on your custom settings:

```
POOTLE_ENABLE_API = True
```

Warning: If you are running Pootle using Apache with `mod_wsgi` you will need to enable `WSGIPassAuthorization` On as told in [Tastypie authentication docs](#).

1.5.2 Pootle API usage

In order to interact with Pootle API it is necessary to know how to use it and some of its particularities.

Using Pootle API

In order to use the Pootle API it is necessary to know how some things, like the supported formats, available authentication methods or basic rules for performing queries.

Pootle API is created using [Tastypie](#) so you might need to refer to [its documentation](#) as well.

How to perform API queries

The structure of the API URLs is `<SERVER>/api/<API_VERSION>/<QUERY>` where:

Placeholder	Description
<code><SERVER></code>	The URL of the Pootle server
<code><API_VERSION></code>	Version number of the API
<code><QUERY></code>	Resource query URI

So the API can be queried using URLs like:

```
http://pootle.locamotion.org/api/v1/translation-projects/65/
```

List matching a criteria For some resources it is also possible to narrow down the list by providing a [query string](#) containing filters [provided by Tastypie](#) (that actually are [Django ORM Field Lookups](#)).

In this case the structure of the API URLs is `<SERVER>/api/<API_VERSION>/<RESOURCE>/?<CRITERIA>` where `<CRITERIA>` is the query string. For example:

```
http://pootle.locamotion.org/api/v1/units/?mtime__month=05&mtime__day=12&state__exact=200
```

Authentication

Pootle requires authentication for accessing its API.

The method used for authentication is [HTTP Basic Authentication](#) which requires providing a username and a password (the same ones used for Pootle login).

Note: Other authentication methods can be added in the future.

Authorization

The Pootle API allows to interact with resources that represent some of the data handled internally by Pootle. In order to avoid all users access or alter data they are not meant to, the Pootle API checks if the visitor has enough permissions to perform the requested actions on the resources. The permissions used for these checks are the same permissions used in Pootle for regular users.

For some particular resources some other checks can be done to allow or deny performing the requested action. For example the visitors can only see the *User resource* for the user that they used to log in the Pootle API.

Formats

By default Pootle API returns only **JSON** replies. It is possible to use all the **formats supported** by Tastypie.

Tools and libraries

Translate is currently developing a **client for Pootle API**, but there are several other **libraries and programs** capable of interacting with Pootle API. For example here is an example script that uses **Slumber** to retrieve and print the list of used languages in Pootle:

```
import slumber

# Change the following to match your Pootle URL, your username and password.
API_URL = "http://127.0.0.1:8000/api/v1/"
AUTH=('admin', 'admin')

api = slumber.API(API_URL, auth=AUTH)

# Get all languages data.
lang_data = api.languages.get()

for lang in lang_data["objects"]:
    print(lang["code"])
```

Note: Remember to **install Slumber** in order to run the previous code.

1.5.3 Available resources

The Pootle API exposes a number of resources. Next you have a complete list of them with data about the accepted HTTP methods, result limits, authentication requirements or other constraints.

Note: You might want to look at the *Glossary* to fully understand the resource names used in the API.

Language resources

The Pootle API exposes a number of resources. Next you have a complete list of *Language* specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List languages

URL /languages/

Description Returns the languages list.

API versions 1

Method GET

Returns List of languages.

```

{
  "meta": {
    "limit": 1000,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 132
  },
  "objects": [
    {
      "code": "af",
      "description": "",
      "fullname": "Afrikaans",
      "nplurals": 2,
      "pluralequation": "(n != 1)",
      "resource_uri": "/api/v1/languages/3/",
      "specialchars": "ëïêôûáéíóúý",
      "translation_projects": [
        "/api/v1/translation-projects/2/",
        "/api/v1/translation-projects/3/"
      ]
    },
    {
      "code": "ak",
      "description": "",
      "fullname": "Akan",
      "nplurals": 2,
      "pluralequation": "(n > 1)",
      "resource_uri": "/api/v1/languages/4/",
      "specialchars": "",
      "translation_projects": [
        "/api/v1/translation-projects/4/"
      ]
    }
  ]
}

```

Create a language

URL /languages/

Description Creates a new language.

API versions 1

Method POST

Returns HTTP 201 response with the relative URL for the newly created language on its `Location` header.

Get a language

URL `/languages/<LANG>/`

Description Returns the language with the `<LANG>` ID.

API versions 1

Method GET

Returns Language with `<LANG>` ID.

```
{
  "code": "gl",
  "description": "",
  "fullname": "Galician",
  "nplurals": 2,
  "pluralequation": "(n != 1)",
  "resource_uri": "/api/v1/languages/20/",
  "specialchars": "",
  "translation_projects": [
    "/api/v1/translation-projects/12/",
    "/api/v1/translation-projects/81/"
  ]
}
```

Change a language

URL `/languages/<LANG>/`

Description Changes the language with the `<LANG>` ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the language is going to be partially changed (just some of its fields)
 - **PUT** if the whole language is going to be changed
-

Delete a language

URL `/languages/<LANG>/`

Description Deletes the language with the `<LANG>` ID.

API versions 1

Method DELETE

Returns HTTP 204 NO CONTENT response.

Get statistics for a language

URL /languages/<LANG>/statistics/

Description Returns the language with the <LANG> ID, including an extra field with its statistics.

API versions 1

Method GET

Returns Language with <LANG> ID and its statistics.

```

{
  "code": "gl",
  "description": "",
  "fullname": "Galician",
  "nplurals": 2,
  "pluralequation": "(n != 1)",
  "resource_uri": "/api/v1/languages/20/",
  "specialchars": "",
  "statistics": {
    "errors": 0,
    "fuzzy": {
      "percentage": 1,
      "units": 1,
      "words": 1
    },
    "suggestions": 0,
    "total": {
      "percentage": 100,
      "units": 1191,
      "words": 1949
    },
    "translated": {
      "percentage": 91,
      "units": 1156,
      "words": 1767
    },
    "untranslated": {
      "percentage": 8,
      "units": 34,
      "words": 181
    }
  },
  "translation_projects": [
    "/api/v1/translation-projects/12/",
    "/api/v1/translation-projects/81/"
  ]
}

```

Project resources

The Pootle API exposes a number of resources. Next you have a complete list of *Project* specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List projects

URL /projects/

Description Returns the projects list.

API versions 1

Method GET

Returns List of projects.

```
{
  "meta": {
    "limit": 1000,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 4
  },
  "objects": [
    {
      "checkstyle": "standard",
      "code": "firefox",
      "description": "",
      "fullname": "Firefox 22 (Aurora)",
      "ignoredfiles": "",
      "localfiletype": "po",
      "resource_uri": "/api/v1/projects/4/",
      "source_language": "/api/v1/languages/2/",
      "translation_projects": [
        "/api/v1/translation-projects/71/",
        "/api/v1/translation-projects/72/",
        "/api/v1/translation-projects/73/",
        "/api/v1/translation-projects/74/"
      ],
      "treestyle": "nongnu"
    },
    {
      "checkstyle": "standard",
      "code": "lxde",
      "description": "",
      "fullname": "LXDE",
      "ignoredfiles": "",
      "localfiletype": "po",
      "resource_uri": "/api/v1/projects/5/",
      "source_language": "/api/v1/languages/2/",
      "translation_projects": [
        "/api/v1/translation-projects/88/",
        "/api/v1/translation-projects/89/",
        "/api/v1/translation-projects/90/"
      ],
      "treestyle": "nongnu"
    }
  ]
}
```

Create a project

URL /projects/

Description Creates a new project.

API versions 1

Method POST

Returns HTTP 201 response with the relative URL for the newly created project on its `Location` header.

Get a project

URL /projects/<PROJ>/

Description Returns the project with the <PROJ> ID.

API versions 1

Method GET

Returns Project with <PROJ> ID.

```
{
  "checkstyle": "standard",
  "code": "firefox",
  "description": "",
  "fullname": "Firefox 22 (Aurora)",
  "ignoredfiles": "",
  "localfiletype": "po",
  "resource_uri": "/api/v1/projects/4/",
  "source_language": "/api/v1/languages/2/",
  "translation_projects": [
    "/api/v1/translation-projects/71/",
    "/api/v1/translation-projects/72/",
    "/api/v1/translation-projects/73/",
    "/api/v1/translation-projects/74/"
  ],
  "treestyle": "nongnu"
}
```

Change a project

URL /projects/<PROJ>/

Description Changes the project with the <PROJ> ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the project is going to be partially changed (just some of its fields)
 - **PUT** if the whole project is going to be changed
-

Delete a project

URL /projects/<PROJ>/

Description Deletes the project with the <PROJ> ID.

API versions 1

Method DELETE

Returns HTTP 204 NO CONTENT response.

Get statistics for a project

URL /projects/<PROJ>/statistics/

Description Returns the project with the <PROJ> ID, including an extra field with its statistics.

API versions 1

Method GET

Returns Project with <PROJ> ID and its statistics.

```
{
  "checkstyle": "standard",
  "code": "firefox",
  "description": "",
  "fullname": "Firefox 22 (Aurora)",
  "ignoredfiles": "",
  "localfiletype": "po",
  "resource_uri": "/api/v1/projects/4/",
  "source_language": "/api/v1/languages/2/",
  "statistics": {
    "errors": 0,
    "fuzzy": {
      "percentage": 1,
      "units": 1,
      "words": 7
    },
    "suggestions": 5,
    "total": {
      "percentage": 100,
      "units": 289,
      "words": 1309
    },
    "translated": {
      "percentage": 99,
      "units": 284,
      "words": 1296
    },
    "untranslated": {
      "percentage": 0,
      "units": 4,
      "words": 6
    }
  },
  "translation_projects": [
    "/api/v1/translation-projects/71/",
    "/api/v1/translation-projects/72/",
```

```

    "/api/v1/translation-projects/73/",
    "/api/v1/translation-projects/74/"
  ],
  "treestyle": "nongnu"
}

```

Store resources

The Pootle API exposes a number of resources. Next you have a complete list of *Store* specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List stores in a translation project

URL /translation-projects/<TRPR>/

Description Returns the store (file) list on a given <TRPR> translation project.

API versions 1

Method GET

Returns Store (file) list on a given <TRPR> translation project.

```

{
  "description": "",
  "language": "/api/v1/languages/110/",
  "pootle_path": "/fr/Firefox/",
  "project": "/api/v1/projects/3/",
  "real_path": "Firefox/fr",
  "resource_uri": "/api/v1/translation-projects/65/",
  "stores": [
    "/api/v1/stores/77/",
    "/api/v1/stores/76/",
    "/api/v1/stores/75/"
  ]
}

```

Get a store

URL /stores/<STOR>/

Description Returns the store with the <STOR> ID.

API versions 1

Method GET

Returns Store with <STOR> ID.

```

{
  "file": "/media/Firefox/fr/chrome/global/languageNames.properties.po",
  "name": "languageNames.properties.po",
  "pending": null,
  "pootle_path": "fr/firefox/chrome/global/languageNames.properties.po",
  "resource_uri": "/api/v1/stores/76/",
}

```

```
{
  "state": 2,
  "sync_time": "2013-03-15T20:10:35.070238",
  "tm": null,
  "translation_project": "/api/v1/translation-projects/65/",
  "units": [
    "/api/v1/units/70316/",
    "/api/v1/units/70317/",
    "/api/v1/units/70318/",
    "/api/v1/units/70319/"
  ]
}
```

Get statistics for a store

URL /stores/<STOR>/statistics/

Description Returns the store with the <STOR> ID, including an extra field with its statistics.

API versions 1

Method GET

Returns Store with <STOR> ID and its statistics.

```
{
  "file": "/media/Firefox/fr/chrome/global/languageNames.properties.po",
  "name": "languageNames.properties.po",
  "pending": null,
  "pootle_path": "fr/firefox/chrome/global/languageNames.properties.po",
  "resource_uri": "/api/v1/stores/76/",
  "state": 2,
  "statistics": {
    "errors": 0,
    "fuzzy": {
      "percentage": 26,
      "units": 1,
      "words": 7
    },
    "suggestions": 1,
    "total": {
      "percentage": 100,
      "units": 4,
      "words": 27
    },
    "translated": {
      "percentage": 63,
      "units": 2,
      "words": 17
    },
    "untranslated": {
      "percentage": 11,
      "units": 1,
      "words": 3
    }
  },
  "sync_time": "2013-03-15T20:10:35.070238",
  "tm": null,
  "translation_project": "/api/v1/translation-projects/65/",
}
```



```

    "units": [
        "/api/v1/units/70316/",
        "/api/v1/units/70317/",
        "/api/v1/units/70318/",
        "/api/v1/units/70319/"
    ]
}

```

Suggestion resources

The Pootle API exposes a number of resources. Next you have a complete list of Suggestion specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List suggestions for a unit

Description Returns the suggestion list for a given <UNIT> unit.

API versions 1

Method GET

Returns Suggestion list on a given <UNIT> unit.

```

{
    "commented_by": null,
    "commented_on": "2013-03-15T20:10:35.017844",
    "context": "This is a phrase, not a verb.",
    "developer_comment": "Translators: name of the option in the menu.",
    "locations": "fr/firefox/chrome/global/languageNames.properties.po:62",
    "mtime": "2013-05-12T17:51:49.786611",
    "resource_uri": "/api/v1/units/70316/",
    "source_f": "New Tab",
    "source_length": 29,
    "source_wordcount": 3,
    "state": 0,
    "store": "/api/v1/stores/76/",
    "submitted_by": "/api/v1/users/3/",
    "submitted_on": "2013-05-21T17:51:16.155000",
    "suggestions": [
        "/api/v1/suggestions/1/",
        "/api/v1/suggestions/3/"
    ],
    "target_f": "",
    "target_length": 0,
    "target_wordcount": 0,
    "translator_comment": ""
}

```

Create a suggestion

URL /suggestions/

Description Creates a new suggestion.

API versions 1

Method POST

Returns HTTP 201 response with the relative URL for the newly created suggestion on its `Location` header.

Get a suggestion

URL /suggestions/<SUGG>/

Description Returns the suggestion with the <SUGG> ID.

API versions 1

Method GET

Returns Suggestion with <SUGG> ID.

```
{
  "resource_uri": "/api/v1/suggestions/1/",
  "target_f": "Nouvel onglet",
  "translator_comment_f": "",
  "unit": "/api/v1/units/70316/"
}
```

Change a suggestion

URL /suggestions/<SUGG>/

Description Changes the suggestion with the <SUGG> ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the suggestion is going to be partially changed (just some of its fields)
 - **PUT** if the whole suggestion is going to be changed
-

Delete a suggestion

URL /suggestion/<SUGG>/

Description Deletes the suggestion with the <SUGG> ID.

API versions 1

Method DELETE

Returns HTTP 204 NO CONTENT response.

Translation project resources

The Pootle API exposes a number of resources. Next you have a complete list of *Translation project* specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List translation projects in a project

URL /projects/<PROJ>/

Description Returns the translation projects (languages) list on a given <PROJ> project.

API versions 1

Method GET

Returns List of translation projects (languages) on a given <PROJ> project.

```
{
  "checkstyle": "standard",
  "code": "firefox",
  "description": "",
  "fullname": "Firefox 22 (Aurora)",
  "ignoredfiles": "",
  "localfiletype": "po",
  "resource_uri": "/api/v1/projects/4/",
  "source_language": "/api/v1/languages/2/",
  "translation_projects": [
    "/api/v1/translation-projects/71/",
    "/api/v1/translation-projects/72/",
    "/api/v1/translation-projects/73/",
    "/api/v1/translation-projects/74/"
  ],
  "treestyle": "nongnu"
}
```

List translation projects in a language

URL /languages/<LANG>/

Description Returns the translation projects (projects) list on a given <LANG> language.

API versions 1

Method GET

Returns List of translation projects (projects) on a given <LANG> language.

```
{
  "code": "gl",
  "description": "",
  "fullname": "Galician",
  "nplurals": 2,
  "pluralequation": "(n != 1)",
  "resource_uri": "/api/v1/languages/20/",
  "specialchars": "",
  "translation_projects": [
    "/api/v1/translation-projects/12/",
  ]
}
```

```
    "/api/v1/translation-projects/81/"
  ]
}
```

Create a translation project

URL /translation-projects/

Description Creates a new translation project.

API versions 1

Method POST

Returns HTTP 201 response with the relative URL for the newly created translation project on its Location header.

Get a translation project

URL /translation-projects/<TRPR>/

Description Returns the translation project with the <TRPR> ID.

API versions 1

Method GET

Returns Translation project with <TRPR> ID.

```
{
  "description": "",
  "language": "/api/v1/languages/110/",
  "pootle_path": "/fr/Firefox/",
  "project": "/api/v1/projects/3/",
  "real_path": "Firefox/fr",
  "resource_uri": "/api/v1/translation-projects/65/",
  "stores": [
    "/api/v1/stores/77/",
    "/api/v1/stores/76/",
    "/api/v1/stores/75/"
  ]
}
```

Change a translation project

URL /translation-projects/<TRPR>/

Description Changes the translation project with the <TRPR> ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the translation project is going to be partially changed (just some of its fields)

- PUT if the whole translation project is going to be changed

Delete a translation project

URL /translation-projects/<TRPR>/

Description Deletes the translation project with the <TRPR> ID.

API versions 1

Method DELETE

Returns HTTP 204 NO CONTENT response.

Get statistics for a translation project

URL /translation-projects/<TRAP>/statistics/

Description Returns the translation project with the <TRAP> ID, including an extra field with its statistics.

API versions 1

Method GET

Returns Translation project with <TRAP> ID and its statistics.

```
{
  "description": "",
  "language": "/api/v1/languages/110/",
  "pootle_path": "/fr/Firefox/",
  "project": "/api/v1/projects/3/",
  "real_path": "Firefox/fr",
  "resource_uri": "/api/v1/translation-projects/65/",
  "statistics": {
    "errors": 0,
    "fuzzy": {
      "percentage": 4,
      "units": 1,
      "words": 7
    },
    "suggestions": 3,
    "total": {
      "percentage": 100,
      "units": 39,
      "words": 167
    },
    "translated": {
      "percentage": 94,
      "units": 37,
      "words": 157
    },
    "untranslated": {
      "percentage": 2,
      "units": 1,
      "words": 3
    }
  }
},
```

```
"stores": [
    "/api/v1/stores/77/",
    "/api/v1/stores/76/",
    "/api/v1/stores/75/"
]
```

Unit resources

The Pootle API exposes a number of resources. Next you have a complete list of *Unit* specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

List units in a store

URL /stores/<STOR>/

Description Returns the unit list on a given <STOR> store.

API versions 1

Method GET

Returns Unit list on a given <STOR> store.

```
{
  "file": "/media/Firefox/fr/chrome/global/languageNames.properties.po",
  "name": "languageNames.properties.po",
  "pending": null,
  "pootle_path": "fr/firefox/chrome/global/languageNames.properties.po",
  "resource_uri": "/api/v1/stores/76/",
  "state": 2,
  "sync_time": "2013-03-15T20:10:35.070238",
  "tm": null,
  "translation_project": "/api/v1/translation-projects/65/",
  "units": [
    "/api/v1/units/70316/",
    "/api/v1/units/70317/",
    "/api/v1/units/70318/",
    "/api/v1/units/70319/"
  ]
}
```

List units matching a criteria

URL /units/?<CRITERIA>

Description Returns a unit list that match the <CRITERIA>.

API versions 1

Method GET

Returns Unit list that match a given <CRITERIA>.

<CRITERIA> is a *query string* where the fields are Django ORM Field Lookups. Some examples might help:

- `source_f` field contains, case insensitive, *button*: `/units/?source_f__icontains=button`
- `target_f` field starts with *window*: `/units/?target_f__startswith=window`
- `mtime` field (modification datetime) is on the *May* month: `/units/?mtime__month=05`
- Unit is translated: `/units/?state=200`

Multiple field lookups can be provided, even several lookups on the same field:

`/units/?mtime__month=05&mtime__day=12&developer_comment__icontains=verb`

Note: It is not possible to provide **OR** conditions using filters, nor negate the filters.

Fields	Available filters (field lookups)
<ul style="list-style-type: none"> • <code>context</code> • <code>developer_comment</code> • <code>locations</code> • <code>source_f</code> • <code>target_f</code> • <code>translator_comment</code> 	<ul style="list-style-type: none"> • <code>exact</code> • <code>icontains</code> • <code>startswith</code> • <code>istartswith</code> • <code>endswith</code> • <code>iendswith</code>
<ul style="list-style-type: none"> • <code>commented_on</code> • <code>mtime</code> • <code>submitted_on</code> 	<ul style="list-style-type: none"> • <code>year</code> • <code>month</code> • <code>day</code>
<ul style="list-style-type: none"> • <code>state</code> • <code>store</code> 	<ul style="list-style-type: none"> • <code>exact</code>

The available states are:

- **0** (untranslated): The unit is untranslated (empty)
- **50** (fuzzy): The unit is fuzzy (typically means translation needs more work)
- **200** (translated): The unit is fully translated
- **-100** (obsolete): The unit is no longer part of the store

Warning: It is possible to get all the units in a given store by requesting `/units/?store=<STOR>` but it is recommended to use the [List units in a store](#) method instead.

Filtering by store is only advisable when:

- You need to provide extra filters:
`/units/?store=74&developer_comment__icontains=verb`
- You want to get all the data for those units with a single request, despite the computational cost.

Get a unit

URL `/units/<UNIT>/`

Description Returns the unit with the `<UNIT>` ID.

API versions 1

Method GET

Returns Unit with <UNIT> ID.

```
{
  "commented_by": null,
  "commented_on": "2013-03-15T20:10:35.017844",
  "context": "This is a phrase, not a verb.",
  "developer_comment": "Translators: name of the option in the menu.",
  "locations": "fr/firefox/chrome/global/languageNames.properties.po:62",
  "mtime": "2013-05-12T17:51:49.786611",
  "resource_uri": "/api/v1/units/70316/",
  "source_f": "New Tab",
  "source_length": 29,
  "source_wordcount": 3,
  "state": 0,
  "store": "/api/v1/stores/76/",
  "submitted_by": "/api/v1/users/3/",
  "submitted_on": "2013-05-21T17:51:16.155000",
  "suggestions": [
    "/api/v1/suggestions/1/"
  ],
  "target_f": "",
  "target_length": 0,
  "target_wordcount": 0,
  "translator_comment": ""
}
```

Change a unit

URL /units/<UNIT>/

Description Changes the unit with the <UNIT> ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the unit is going to be partially changed (just some of its fields), for example when providing a translation
 - **PUT** if the whole unit is going to be changed
-

User resources

The Pootle API exposes a number of resources. Next you have a complete list of User specific resources.

Note: All URLs listed here should be *appended to the base URL of the API*.

Create a user

URL /users/

Description Creates a new user.

API versions 1

Method POST

Returns HTTP 201 response with the relative URL for the newly created user on its `Location` header.

Note: The consumer must have the appropriate permissions in order to be able to create new users.

Warning: The new user will have no password set, and therefore won't be able to log in until a password is set, either by an administrator or by the user requesting and setting a new password.

Get a user

URL `/users/<USER>/`

Description Returns the user with the `<USER>` ID.

API versions 1

Method GET

Returns User with `<USER>` ID.

Note: The consumer will get the user data only if it is authenticated as the user which is trying to get the data for.

```
{
  "date_joined": "2013-03-15T19:04:39.401505",
  "email": "admin@doesnotexist.com",
  "first_name": "Administrator",
  "last_name": "",
  "resource_uri": "/api/v1/users/3/",
  "username": "admin"
}
```

Change a user

URL `/users/<USER>/`

Description Changes the user with the `<USER>` ID.

API versions 1

Method PATCH or PUT

Returns HTTP 204 NO CONTENT response.

Note: The method used can be:

- **PATCH** if the user is going to be partially changed (just some of its fields)
 - **PUT** if the whole user is going to be changed
-

Note: The consumer will only be able to change the data for a given user if it:

- Is authenticated as the user which is trying to change the data for, and

- Has enough permissions to perform this action.
-

Delete a user

URL /users/<USER>/

Description Deletes the user with the <USER> ID.

API versions 1

Method DELETE

Returns HTTP 204 NO CONTENT response.

Note: The consumer will only be able to delete a given user if it:

- Is authenticated as the user which is trying to delete, and
 - Has enough permissions to perform this action.
-

Get statistics for a user

URL /users/<USER>/statistics/

Description Returns the user with the <USER> ID, including an extra field with its statistics.

API versions 1

Method GET

Returns User with <USER> ID and its statistics.

Note: If the consumer is authenticated as the same user for which the statistics are shown, then some extra fields are included in the response.

This fields are the same ones that can be accessed when the consumer *gets the data for a user*.

```
{
  "resource_uri": "/api/v1/users/3/",
  "statistics": [
    [
      "Portuguese (Brazil) - pt_BR",
      [
        [
          "/pt_BR/Firefox/",
          [
            {
              "count": 2,
              "id": "suggestions-pending",
              "url": "/pt_BR/Firefox/translate.html#filter=user-suggestions&user=admin",
            },
            {
              "count": 0,
              "id": "suggestions-accepted",
              "url": "/pt_BR/Firefox/translate.html#filter=user-suggestions-accepted&user=admin",
            },
            {
              "count": 0,
            }
          ]
        ]
      ]
    ]
  ]
}
```

```

        "id": "suggestions-rejected",
        "url": "/pt_BR/Firefox/translate.html#filter=user-suggestions-rejected&user=admin",
    },
    {
        "count": 10,
        "id": "submissions-total",
        "url": "/pt_BR/Firefox/translate.html#filter=user-submissions&user=admin",
    },
    {
        "count": 0,
        "id": "submissions-overwritten",
        "url": "/pt_BR/Firefox/translate.html#filter=user-submissions-overwritten&user=admin",
    }
]
]
],
[
    "Russian - ru",
    [
        ["/ru/LXDE/"],
        [
            {
                "count": 0,
                "id": "suggestions-pending",
                "url": "/ru/LXDE/translate.html#filter=user-suggestions&user=admin",
            },
            {
                "count": 0,
                "id": "suggestions-accepted",
                "url": "/ru/LXDE/translate.html#filter=user-suggestions-accepted&user=admin",
            },
            {
                "count": 0,
                "id": "suggestions-rejected",
                "url": "/ru/LXDE/translate.html#filter=user-suggestions-rejected&user=admin",
            },
            {
                "count": 34,
                "id": "submissions-total",
                "url": "/ru/LXDE/translate.html#filter=user-submissions&user=admin",
            },
            {
                "count": 0,
                "id": "submissions-overwritten",
                "url": "/ru/LXDE/translate.html#filter=user-submissions-overwritten&user=admin",
            }
        ]
    ]
]
],
    "username": "admin"
}

```

Additional Notes

2.1 Changelog

These are the critical 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*.

Note: For newer Pootle versions changes please check the [Release notes](#).

2.1.1 Version 2.1.1

Bugfix release, released on September 3rd 2010.

- The default cache backend is now a database backend. Memcached is still the preferred cache backend, but consider using the database cache if you are using the local memory backend and can't use memcached.
- You can perform a *database migration away from SQLite*.

2.1.2 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.

2.1.3 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.

2.1.4 Version 1.2.0

Released on October 8th 2008.

- The name of the directory for indexing databases changed from *.poindex-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*.

2.1.5 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.

2.1.6 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)*.

2.1.7 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 Release Notes

The following are release notes used on PyPI, Sourceforge and mailing lists for Pootle releases.

2.2.1 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.

2.2.2 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.

2.2.3 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*.
- 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.

2.2.4 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 \geq 1.4.10 (note that Django 1.5 and 1.6 are not yet supported)
- [Translate Toolkit](#) \geq 1.11.0
- Python \geq 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).

Changes since 2.5.1-rc1

- [Goals](#): more efficient cache flushing mechanism, sites with large projects took very long to submit new translations.
- [LDAP](#): explicit import of `ldap.filter`
- [Tags](#): restrict accepted taggit versions to those that will work with Pootle’s use of tags.

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.

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

2.2.5 Welcome to the new Pootle 2.5.1-rc1

Released on 1 December 2013

We almost missed 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`
- [Translate Toolkit](#) `>= 1.11.0-rc1`
- 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.

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

2.2.6 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

2.5.0 vs 2.5.0-rc1

Changes from 2.5.0 RC1 to 2.5.0 final release:

- Correct all Right-to-Left rendering issues
- Minor fixes: update translations, fixes to lightbox and some documentation corrections

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:

- 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.
- 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`
- `update_from_templates` has been renamed to `update_against_templates`

Important server admin 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`.
- 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.
- Make sure to use the minimum required South version when performing database upgrades.

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

...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, kharoth, dvinella, Stuart Prescott, Roman Imankulov, Peter Bengtsson, Nagy Akos, Michael Tänzer, Gregory Oschwaldi & Andy Nicholson.

2.2.7 Welcome to the new Pootle 2.5.0-rc1

Released on 16 March 2013

At Translate we're pretty proud of this baby. 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 enjoy. 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 the [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:

- 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.
- 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`
- `update_from_templates` has been renamed to `update_against_templates`

Important server admin 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`.
- 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.
- Make sure to use the minimum required South version when performing database upgrades.

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

...and of course, loads of bugs where 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.

2.2.8 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](#)

2.2.9 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

2.2.10 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](#)

2.2.11 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](#)

2.2.12 Pootle 2.1.2 Released including security fix

Released on 15 Nov 2010

<http://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 where 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](#)

2.3 External documentation

These are external sources of documentation, often relevant to a specific Pootle server for one of the many projects that use Pootle.

- <http://colivre.coop.br/Tech/TraducaoPootle>
- http://wiki.creativecommons.org/Translating_with_Pootle
- http://wiki.openoffice.org/wiki/Pootle_User_Guide
- http://wiki.openoffice.org/wiki/Pootle_Glossary_Guide
- <http://mr-dust.pe.kr/entry/How-to-translate-OpenOfficeorg>
- <http://docs.linux.org.ua/Dlou/index.php/Pootle> (Ukrainian,)
- <http://takanory.net/plone/japanese/pootle> (Japanese,)
- <http://www.moongift.jp/2006/09/2326/> (Japanese,)

- <http://gustavonarea.net/blog/posts/installing-pootle-on-debian-etch-the-easiest-way/>
- <http://wiki.laptop.org/go/Pootle> (OLPC project)
- <http://wiki.list.org/display/DEV/Pootle+primer> (mailman project)
- <http://wiki.squid-cache.org/Translations/Basics> (Squid project)

2.4 License

The Pootle documentation is released under the [GNU General Public License \(GPL\)](#), version 2 or later.

A

AMAGAMA_URL
 [setting, 54](#)
API_LIMIT_PER_PAGE
 [setting, 52](#)
AUTH_LDAP_ANON_DN
 [setting, 54](#)
AUTH_LDAP_ANON_PASS
 [setting, 54](#)
AUTH_LDAP_BASE_DN
 [setting, 54](#)
AUTH_LDAP_FIELDS
 [setting, 54](#)
AUTH_LDAP_FILTER
 [setting, 54](#)
AUTH_LDAP_SERVER
 [setting, 54](#)
AUTOSYNC
 [setting, 54](#)

C

CAN_CONTACT
 [setting, 52](#)
CAN_REGISTER
 [setting, 52](#)
CONTACT_EMAIL
 [setting, 52](#)
CUSTOM_TEMPLATE_CONTEXT
 [setting, 52](#)

D

DESCRIPTION
 [setting, 52](#)

E

EMAIL_SEND_HTML
 [setting, 52](#)
ENABLE_ALT_SRC
 [setting, 55](#)
EXPORTED_DIRECTORY_MODE

[setting, 54](#)

EXPORTED_FILE_MODE
 [setting, 54](#)

F

FUZZY_MATCH_MAX_LENGTH
 [setting, 52](#)
FUZZY_MATCH_MIN_SIMILARITY
 [setting, 52](#)

L

LEGALPAGE_NOCHECK_PREFIXES
 [setting, 53](#)
LIVE_TRANSLATION
 [setting, 54](#)
LOOKUP_BACKENDS
 [setting, 54](#)

M

MARKUP_FILTER
 [setting, 53](#)
MAX_AUTOTERMS
 [setting, 53](#)
MIN_AUTOTERMS
 [setting, 53](#)
MT_BACKENDS
 [setting, 55](#)

O

OBJECT_CACHE_TIMEOUT
 [setting, 52](#)

P

PARSE_POOL_CULL_FREQUENCY
 [setting, 55](#)
PARSE_POOL_SIZE
 [setting, 55](#)
PODIRECTORY
 [setting, 55](#)
POOTLE_ENABLE_API

setting, [53](#)

POOTLE_TOP_STATS_CACHE_TIMEOUT
setting, [52](#)

S

setting

AMAGAMA_URL, [54](#)
API_LIMIT_PER_PAGE, [52](#)
AUTH_LDAP_ANON_DN, [54](#)
AUTH_LDAP_ANON_PASS, [54](#)
AUTH_LDAP_BASE_DN, [54](#)
AUTH_LDAP_FIELDS, [54](#)
AUTH_LDAP_FILTER, [54](#)
AUTH_LDAP_SERVER, [54](#)
AUTOSYNC, [54](#)
CAN_CONTACT, [52](#)
CAN_REGISTER, [52](#)
CONTACT_EMAIL, [52](#)
CUSTOM_TEMPLATE_CONTEXT, [52](#)
DESCRIPTION, [52](#)
EMAIL_SEND_HTML, [52](#)
ENABLE_ALT_SRC, [55](#)
EXPORTED_DIRECTORY_MODE, [54](#)
EXPORTED_FILE_MODE, [54](#)
FUZZY_MATCH_MAX_LENGTH, [52](#)
FUZZY_MATCH_MIN_SIMILARITY, [52](#)
LEGALPAGE_NOCHECK_PREFIXES, [53](#)
LIVE_TRANSLATION, [54](#)
LOOKUP_BACKENDS, [54](#)
MARKUP_FILTER, [53](#)
MAX_AUTOTERMS, [53](#)
MIN_AUTOTERMS, [53](#)
MT_BACKENDS, [55](#)
OBJECT_CACHE_TIMEOUT, [52](#)
PARSE_POOL_CULL_FREQUENCY, [55](#)
PARSE_POOL_SIZE, [55](#)
PODIRECTORY, [55](#)
POOTLE_ENABLE_API, [53](#)
POOTLE_TOP_STATS_CACHE_TIMEOUT, [52](#)
TASTYPIE_DEFAULT_FORMATS, [53](#)
TITLE, [52](#)
TOPSTAT_SIZE, [53](#)
USE_CAPTCHA, [53](#)
VCS_DIRECTORY, [55](#)

T

TASTYPIE_DEFAULT_FORMATS
setting, [53](#)

TITLE
setting, [52](#)

TOPSTAT_SIZE
setting, [53](#)

U

USE_CAPTCHA
setting, [53](#)

V

VCS_DIRECTORY
setting, [55](#)