Core GIT Translations

This directory holds the translations for the core of Git. This document describes how you can contribute to the effort of enhancing the language coverage and maintaining the translation.

The localization (l10n) coordinator, Jiang Xin worldhello.net@gmail.com, coordinates our localization effort in the l10 coordinator repository:

```
https://github.com/git-l10n/git-po/
```

The two character language translation codes are defined by ISO_639-1, as stated in the gettext(1) full manual, appendix A.1, Usual Language Codes.

Contributing to an existing translation

As a contributor for a language XX, you should first check TEAMS file in this directory to see whether a dedicated repository for your language XX exists. Fork the dedicated repository and start to work if it exists.

Sometime, contributors may find that the translations of their Git distributions are quite different with the translations of the corresponding version from Git official. This is because some Git distributions (such as from Ubuntu, etc.) have their own l10n workflow. For this case, wrong translations should be reported and fixed through their workflows.

Creating a new language translation

If you are the first contributor for the language XX, please fork this repository, prepare and/or update the translated message file "po/XX.po" (described later), and ask the l10n coordinator to pull your work.

If there are multiple contributors for the same language, please first coordinate among yourselves and nominate the team leader for your language, so that the l10n coordinator only needs to interact with one person per language.

Core translation

The core translation is the smallest set of work that must be completed for a new language translation. Because there are more than 5000 messages in the template message file "po/git.pot" that need to be translated, this is not a piece of cake for the contributor for a new language.

The core template message file which contains a small set of messages will be generated in "po-core/core.pot" automatically by running a helper program named "git-po-helper" (described later).

```
git-po-helper init --core XX.po
```

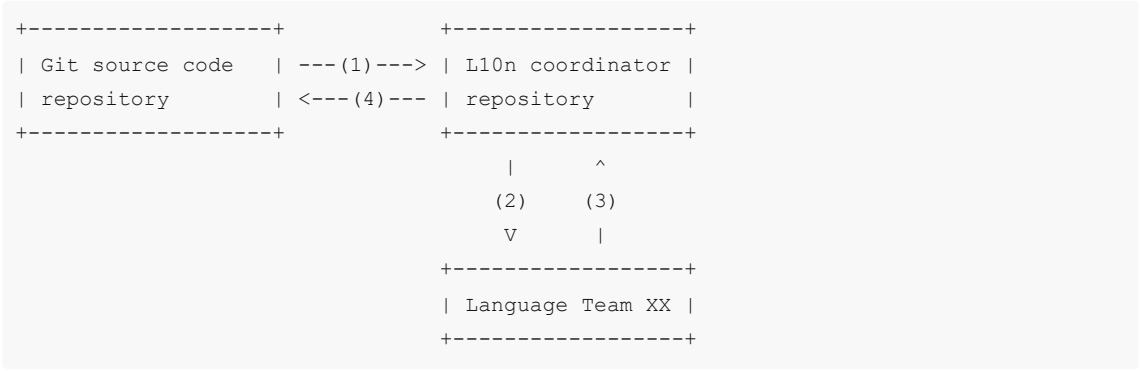
After translating the generated "po-core/XX.po", you can merge it to "po/XX.po" using the following commands:

```
msgcat po-core/XX.po po/XX.po -s -o /tmp/XX.po
mv /tmp/XX.po po/XX.po
git-po-helper update XX.po
```

Edit "po/XX.po" by hand to fix "fuzzy" messages, which may have misplaced translated messages and duplicate messages.

Translation Process Flow

The overall data-flow looks like this:



- Translatable strings are marked in the source file.
- L10n coordinator pulls from the source (1)
- L10n coordinator updates the message template "po/git.pot"
- Language team pulls from L10n coordinator (2)
- Language team updates the message file "po/XX.po"
- L10n coordinator pulls from Language team (3)
- L10n coordinator asks the result to be pulled (4).

Maintaining the "po/git.pot" file

(This is done by the l10n coordinator).

The "po/git.pot" file contains a message catalog extracted from Git's sources. The l10n coordinator maintains it by adding new translations with msginit(1), or update existing ones with msgmerge(1). In order to update the Git sources to extract the messages from, the l10n coordinator is expected to pull from the main git repository at strategic point in history (e.g. when a major release and release candidates are tagged), and then run "make pot" at the top-level directory.

Language contributors use this file to prepare translations for their language, but they are not expected to modify it.

Initializing a "XX.po" file

(This is done by the language teams).

If your language XX does not have translated message file "po/XX.po" yet, you add a translation for the first time by running:

```
msginit --locale=XX
```

in the "po/" directory, where XX is the locale, e.g. "de", "is", "pt_BR", "zh_CN", etc.

Then edit the automatically generated copyright info in your new "XX.po" to be correct, e.g. for Icelandic:

```
@@ -1,6 +1,6 @@
-# Icelandic translations for PACKAGE package.
-# Copyright (C) 2010 THE PACKAGE'S COPYRIGHT HOLDER
-# This file is distributed under the same license as the PACKAGE package.
```

```
+# Icelandic translations for Git.
+# Copyright (C) 2010 Ævar Arnfjörð Bjarmason <avarab@gmail.com>
+# This file is distributed under the same license as the Git package.
 # Ævar Arnfjörð Bjarmason <avarab@gmail.com>, 2010.
```

And change references to PACKAGE VERSION in the PO Header Entry to just "Git":

```
perl -pi -e 's/(?<="Project-Id-Version: )PACKAGE VERSION/Git/' XX.po
```

Once you are done testing the translation (see below), commit the result and ask the l10n coordinator to pull from you.

Updating a "XX.po" file

(This is done by the language teams).

If you are replacing translation strings in an existing "XX.po" file to improve the translation, just edit the file.

If there's an existing "XX.po" file for your language, but the repository of the l10n coordinator has newer "po/git.pot" file, you would need to first pull from the l10n coordinator (see the beginning of this document for its URL), and then update the existing translation by running:

```
msgmerge --add-location --backup=off -U XX.po git.pot
```

in the "po/" directory, where "XX.po" is the file you want to update.

Once you are done testing the translation (see below), commit the result and ask the l10n coordinator to pull from you.

Fuzzy translation

Fuzzy translation is a translation marked by comment "fuzzy" to let you know that the translation is out of date because the "msgid" has been changed. A fuzzy translation will be ignored when compiling using "msgfmt". Fuzzy translation can be marked by hands, but for most cases they are marked automatically when running "msgmerge" to update your "XX.po" file.

After fixing the corresponding translation, you must remove the "fuzzy" tag in the comment.

Testing your changes

(This is done by the language teams, after creating or updating "XX.po" file).

Before you submit your changes go back to the top-level and do:

```
make
```

On systems with GNU gettext (i.e. not Solaris) this will compile your changed PO file with `msgfmt --check`, the --check option flags many common errors, e.g. missing printf format strings, or translated messages that deviate from the originals in whether they begin/end with a newline or not.

Marking strings for translation

(This is done by the core developers).

Before strings can be translated they first have to be marked for translation.

Git uses an internationalization interface that wraps the system's gettext library, so most of the advice in your gettext documentation (on GNU systems `info gettext` in a terminal) applies.

General advice:

- Don't mark everything for translation, only strings which will be read by humans (the porcelain interface) should be translated.

  The output from Git's plumbing utilities will primarily be read by programs and would break scripts under non-C locales if it was translated. Plumbing strings should not be translated, since they're part of Git's API.

- Adjust the strings so that they're easy to translate. Most of the advice in `info '(gettext)Preparing Strings'` applies here.

- Strings referencing numbers of items may need to be split into singular and plural forms; see the Q_() wrapper in the C sub-section below for an example.

- If something is unclear or ambiguous you can use a "TRANSLATORS" comment to tell the translators what to make of it. These will be extracted by xgettext(1) and put in the "po/*.po" files, e.g. from git-am.sh:

  ```
  # TRANSLATORS: Make sure to include [y], [n], [e], [v] and [a]
  # in your translation. The program will only accept English
  # input at this point.
  gettext "Apply? [y]es/[n]o/[e]dit/[v]iew patch/[a]ccept all "
  ```

  Or in C, from builtin/revert.c:

  ```
  /* TRANSLATORS: %s will be "revert" or "cherry-pick" */
  die(_("%s: Unable to write new index file"), action_name(opts));
  ```

We provide wrappers for C, Shell and Perl programs. Here's how they're used:

C

Include builtin.h at the top, it'll pull in gettext.h, which defines the gettext interface. Consult with the list if you need to use gettext.h directly.

The C interface is a subset of the normal GNU gettext interface. We currently export these functions:

- _()

  Mark and translate a string. E.g.:

  ```
  printf(_("HEAD is now at %s"), hex);
  ```

- Q_()

Mark and translate a plural string. E.g.:

```
printf(Q_("%d commit", "%d commits", number_of_commits));
```

This is just a wrapper for the ngettext() function.

- N_()

  A no-op pass-through macro for marking strings inside static initializations, e.g.:

  ```
  static const char *reset_type_names[] = {
      N_("mixed"), N_("soft"), N_("hard"), N_("merge"), N_("keep"), NULL
  };
  ```

  And then, later:

  ```
  die(_("%s reset is not allowed in a bare repository"),
        _(reset_type_names[reset_type]));
  ```

  Here `_()` couldn't have statically determined what the translation string will be, but since it was already marked for translation with `N_()` the look-up in the message catalog will succeed.

Shell

The Git gettext shell interface is just a wrapper for gettext.sh. Import it right after git-sh-setup like this:

```
. git-sh-setup
. git-sh-i18n
```

And then use the `gettext` or `eval_gettext` functions:

```
# For constant interface messages:
gettext "A message for the user"; echo

# To interpolate variables:
details="oh noes"
eval_gettext "An error occurred: \$details"; echo
```

In addition we have wrappers for messages that end with a trailing newline. I.e. you could write the above as:

```
# For constant interface messages:
gettextln "A message for the user"

# To interpolate variables:
details="oh noes"
eval_gettextln "An error occurred: \$details"
```

More documentation about the interface is available in the GNU info page: `info '(gettext)sh'`. Looking at git-am.sh (the first shell command to be translated) for examples is also useful:

```
git log --reverse -p --grep=i18n git-am.sh
```

Perl

The Git::I18N module provides a limited subset of the Locale::Messages functionality, e.g.:

```
use Git::I18N;
print __("Welcome to Git!\n");
printf __("The following error occurred: %s\n"), $error;
```

Run `perldoc perl/Git/I18N.pm` for more info.

Testing marked strings

Git's tests are run under `LANG=C LC_ALL=C` . So the tests do not need be changed to account for translations as they're added.

PO helper

To make the maintenance of "XX.po" easier, the l10n coordinator and l10n team leaders can use a helper program named "git-po-helper". It is a wrapper to gettext suite, specifically written for the purpose of Git l10n workflow.

To build and install the helper program from source, see git-po-helper/README.

Usage for git-po-helper:

- To start a new language translation:

  ```
  git-po-helper init XX.po
  ```

- To update your "XX.po" file:

  ```
  git-po-helper update XX.po
  ```

- To check commit log and syntax of "XX.po":

  ```
  git-po-helper check-po XX.po
  git-po-helper check-commits
  ```

Run "git-po-helper" without arguments to show usage.

Conventions

There are some conventions that l10n contributors must follow:

- The subject of each l10n commit should be prefixed with "l10n: ".
- Do not use non-ASCII characters in the subject of a commit.

- The length of commit subject (first line of the commit log) should be less than 50 characters, and the length of other lines of the commit log should be no more than 72 characters.
- Add "Signed-off-by" trailer to your commit log, like other commits in Git. You can automatically add the trailer by committing with the following command:

  ```
  git commit -s
  ```

- Check syntax with "msgfmt" or the following command before creating your commit:

  ```
  git-po-helper check-po <XX.po>
  ```

- Squash trivial commits to make history clear.
- DO NOT edit files outside "po/" directory.
- Other subsystems ("git-gui", "gitk", and Git itself) have their own workflow. See Documentation/SubmittingPatches for instructions on how to contribute patches to these subsystems.

To contribute for a new l10n language, contributor should follow additional conventions:

- Initialize proper filename of the "XX.po" file conforming to iso-639 and iso-3166.
- Must complete a minimal translation based on the "po-core/core.pot" template. Using the following command to initialize the minimal "po-core/XX.po" file:

  ```
  git-po-helper init --core <your-language>
  ```

- Add a new entry in the "po/TEAMS" file with proper format, and check the syntax of "po/TEAMS" by running the following command:

  ```
  git-po-helper team --check
  ```