Internationalization and localization of shell scripts

Presentation

Purpose, scope and intended audience

This document is intended to help developers, maintainers and translators to write/maintain/translate internationalized then localized shell scripts, using the tools provided by GNU gettext.

The reference document is the manual entitled GNU 'gettext' utilities.

The manual encompasses all programming languages usable with gettext, with a special focus on the C language.

The POSIX specification is a recommended reading, in particular volumes Base Definitions and Shell and Utilities.

In contrast with the the manual, the present document's scope is restricted to shell scripts.

Theory of operation

The aim is that messages (usually text strings) output of shell scripts be displayed on user's screen in his or her preferred language.

The user indicates his or her preferences in setting the LANG or the LANGUAGE environment variable (the latter records a priorized list of languages for messages displaying).

Internationalization (abbreviated in I18N) process consists in:

- marking in the shell scripts that make up the software messages to be translated,
- then use gettext's tools to produce from the set of marked scripts a template messages catalog.

A template messages catalog is usually called a “Portable Object Template” or POT file.

A POT file, human readable, is mainly made up of extracted text strings, preceded by “msgid” for “message identifier”, each one to be followed by a translation of that message, which will be preceded by “msgstr”.

Localization (abbreviated in L10N) process consists in:

- generating from the POT file one “Portable Object” or PO file for each target language,
- feeding the “msgstr” strings with translated messages in each PO file,
- checking these PO files after translation,
- compiling each PO file to generate a “Machine Objet” or MO file.

The MO files, that are only machine readable hence the name, are customarily stored as:

```
/usr/share/locale/<locale>/LC_MESSAGES/<software name>.mo
```

In above path, <locale> is a locale code in the form <ll[_TT], where ll is the two letters code of the target language as defined in ISO 639-1 standard and TT, if present, the two letters country code of the locale as defined in ISO 3166.

Every marked script should include following command:

```
export TEXTDOMAIN=<software name>
```

During script's execution this allows 'gettext' to fetch the proper MO file in order to display each marked message in the preferred language as set by the LANG or LANGUAGE environment variable.

Processes diagrams

Let's assume that a given software comprises a set of shell scripts that we want to internationalize and localize.

Following diagrams give an overview of each of the involved processes: internationalization, localization, usage and maintenance.

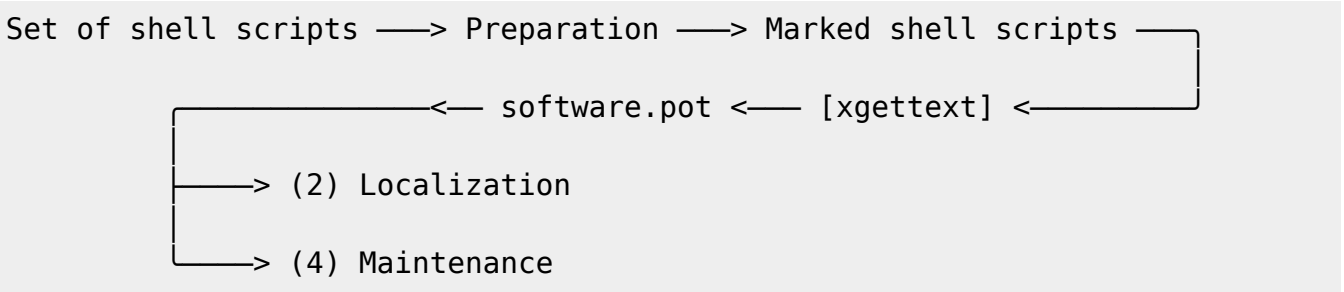
These diagrams are hybrid, i.e. they exhibit data as well as actions.

Among these actions are execution of some programs of the gettext suite:

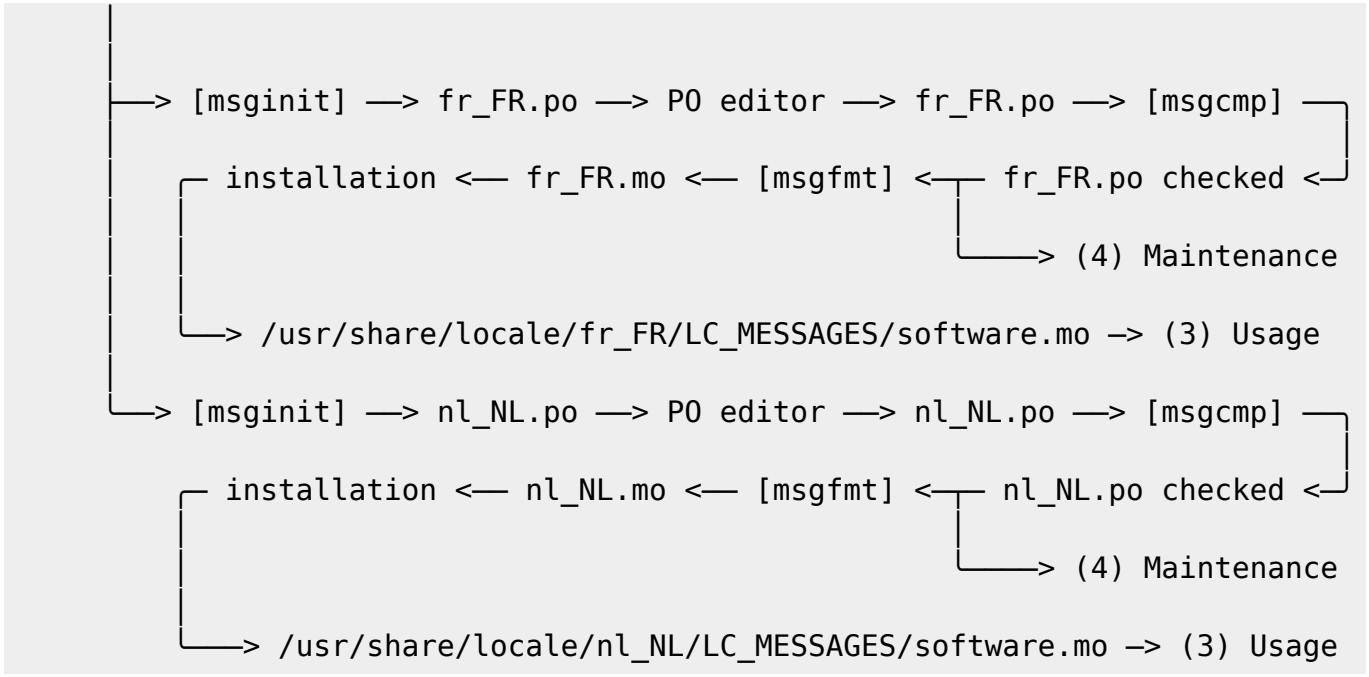
- gettetxt: marks strings to be internationalized, then displays localized messages during scripts' execution
- xgettext: extracts marked strings from a set of shell scripts to build a POT or a PO file
- msgcmp: checks a PO file against another PO or a POT file for consistency
- msginit: write a PO file using a POT file as its input
- msgfmt: format a MO file using a POI file as its input
- msgmerge: merge or update PO or POT files

In below diagrams gettext programs are surrounded by square brackets.

(1) Internationalization



(2) Localization (example for French and Dutch languages).



(3) Usage

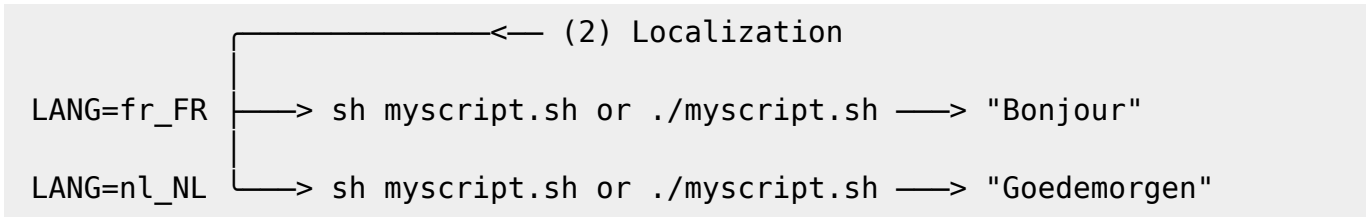
Let's assume that one of the scripts, “myscript.sh” includes following command:

```
gettext "Good morning"
```

and that “Good morning” is translated as follows in the message catalogs:

```
/usr/share/locale/fr_FR/LC_MESSAGES/PACKAGE.mo -> "Bonjour"
/usr/share/locale/nl/LC_MESSAGES/PACKAGE.mo -> "Goedemorgen"
```

Here is what user will see depending on LANG setting:

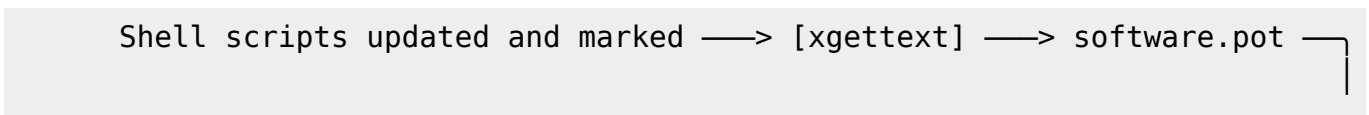


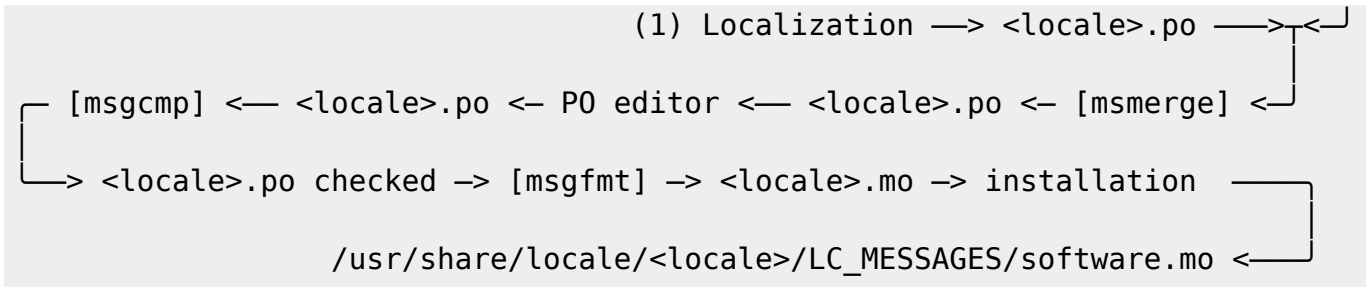
(4) Maintenance

Maintenance process can be triggered by a script's creation, modification or deletion.

In the diagram below, the part of the process beginning with the msmerge command should be repeated for each available PO file.

It is therefore advisable to keep an up to date list of available translations in the form of PO files.





Maintenance process can be triggered as well by a modification of a messages catalog for a specific language (to correct an error for instance).

This variant of the process is shorter:

Internationalization process

This chapter is intended for developers and maintainers.

The internationalization process comprises following tasks:

1. Prepare scripts for internationalization
2. Mark messages to be localized
3. Use 'xgettext' to produce a template catalog of messages

Prepare scripts for internationalization

This task is needed for shell scripts that do not yet fulfill requirements for internationalization.

Technical note: Gettext's requirements for shell scripts.

The list of requirements below is not complete.

It includes only the main ones that I recommend the developer or maintainer to check, based on my experience.

Gettext replaces at run time text strings output of:

- an “echo” command or
- a program (like 'dialog', for instance)

with translated text strings (found in a messages catalog for the language set by $LANG or

$LANGUAGE)

But the replacement only occurs if following conditions are fulfilled:

- A MO file is available in the path computed from the TEXTDOMAIN environment variable as <dir_name>/<locale>/LC_MESSAGES/text_domain.mo.
  For instance, if TEXTDOMAIN=software and $LANG=de_DE.utf8, gettext will look for:
  <dir_name>/de_DE/LC_MESSAGES/software.mo
  <dir_name> can be set through the value of the TEXTDOMAINDIR environment variable, otherwise a default value is used.
  In Slackware Linux for instance, the default value is /usr/share/locale.
  There are fall backs, for instance if <locale> is “de_DE” the mo file could be placed in <dir_name>/de/LC_MESSAGES/ instead of <dir_name>/de_DE/LC_MESSAGES/
- TEXTDOMAIN variable is exported before any *gettext command occurs.
- gettext.sh, which provides the eval_gettext and eval_ngettext functions, is sourced before any occurrence of one of these functions.
- A msgid string in the MO file matches exactly the argument of gettext (or eval_gettext if the text string includes a parameter expansion).
- The corresponding msgstr string does not include a backslash followed by a white space.
- The msgstr string begins and ends with a newline or not, as the msgid does.
- If the text string includes a parameter expansion, eval_gettext is used instead of gettext.
- “The variable names must consist solely of alphanumeric or underscore ASCII characters, not start with a digit and be nonempty; otherwise such a variable reference is ignored.” (gettext manual)
- Parameter expansions are escaped with a single backslash like this:
  \$parameter or \${parameter}
  unless the eval_gettext command be inside a command substitution like this:
  “`eval_gettext ”…“`” or “$(eval_gettext ”…“)“
  In the latter case, three backslashes are needed like this:
  \\\$parameter or \\\${parameter}.
- Only the forms $parameter and ${parameter} of parameter expansion are used inside an eval_gettext's argument (all other ones are forbidden).
- Positional parameters, special parameters and command substitutions are *not* used inside a gettext's or eval_gettext's argument.

As a practical consequence of the two last rules, it is advisable that all positional parameters, special parameters, command substitutions and not allowed forms of parameter substitutions be assigned upstream to named variables, then expanded in the text string argument of eval_gettext or eval_negettext.

Tip: if a text string has been included as a msgid in a catalog of messages and is assigned to a named variable in a script, then the commands: “gettext $parameter” and “gettext ${parameter}” will output the translated string at run time, even though 'xgettext' would discard that command when parsing the script, because 'gettext' is used instead of 'eval_gettext'. This can be handy. In this case the parameter expansion should not be escaped.

Mark messages to be localized

I recommend to mark messages:

- arguments of a not redirected 'echo' command
- arguments of redirected 'echo' commands whenever a further processing displays it on user's screen
- arguments of other commands which displays the message, for instance the 'dialog' program

On the contrary I recommend not to mark:

- comments intended for readers of the script,
- text string whose value will be processed later, for instance as arguments of a 'case' compound command, or <tag> arguments of a dialog –menu' command.

Sometimes the shell script writes other shell scripts.

Then the developer or maintainer have to decide on a case by case basis what to mark depending on the intended scope of internationalization.

Use 'xgettext' to produce a template catalog of messages

The choice to produce only one POT file for the software as a whole or to make one POT files per set of scripts have to be made, considering for instance which choice will minimize maintenance work, how localizations work can be organized, relative frequency of updates for the different sets of scripts which comprise the software, and the relevance of distinguishing groups of features like setup vs configuration vs package management.

I'm inclined to produce only one POT file, but the choice is yours.

If the software comprises of numerous scripts located in different places or included in several packages, it can be handy to collect a copy of all scripts in a single directory, and/or to register in a text file a list of all of them with their paths.

The POT file will be generated using the 'xgettext' command (see the manual or 'xgettext –help' for details).

Include following options in the command:

```
-L Shell (of course!)
--strict (to facilitate checks and management of the messages catalogs)
-c       (to include comments useful for the translators in the POT file)
-n       (to identify the source file and the line number of each message.
         This is the default.)
```

Once the POT file is generated you could check that it includes entries for all *gettext invocation in shell script(s).

Localization process

Once the POT file is available, the 'msginit' command writes a PO file for each target language.

In PO files the “msgid” strings should never be modified, otherwise the translation won't occur at run time.

The 'msgcmp' command allow to checks each PO file against the POT after translation, to make sure all messages are translated.

The translator can use the 'msgfmt' command to check the layout of the translated text.

The PO file should be carefully saved somewhere, as it will be needed for subsequent maintenance (it is still possible to 'msgunfmt' a MO file to re-create a PO file but then you would loose the context, which would make it almost useless).

The checked PO file is handed over to the maintainer, who runs 'msgfmt' to produce the MO file, then installs it.

Usage process

The only thing the user will have to take care of is set up his preferred language(s).

The primary way to do that is setting the LANG environment variable.

This can be done at run time, preceding the command used to run the script with LANG=<locale>, but usually the user will set it up permanently.

For instance in Slackware Linux this will be done in editing the file(s) /etc/profile.d/lang.sh and/or /etc/profile.d/lang.csh (see these files).

The changes will be effective at next reboot.

I suggest to use an UTF-8 locale, as for reading this document.

If the user is polyglot, another option is to set gettext's specific LANGUAGE environment variable to specify a prioritized list of languages.

For instance, if LANGUAGE is set to 'de:fr' then a Deutsch translation will be used if available, else a French translation will be used if available, else messages will be displayed in the original language, usually English. See gettext's manual for details.

Maintenance process

In most cases the maintenance process will be triggered by a script's creation, modification or deletion.

In such a case the maintainer will generate a new POT file with 'xgettext' then hand it hover to the translators.

The translators will use the new POT file to update their respective (saved) PO files with the 'msgmerge –update' command.

Then they will edit/complete the translations, focusing on the not yet translated messages and on those marked as “fuzzy” in the PO files, using a PO editor.

After that the PO file will be checked against the POT file with 'msgcmp', carefully saved, handed over to the maintainer who will generate the new MO file with 'msgfmt' and install it as in the initial localization process.

The maintenance process triggered by a needed modification of a PO file for a specific language is similar, only shorter: it will begin with the update of the relevant PO file by the translator. To minimize the workload caused by this type of maintenance, I suggest that the maintainer demand that he or she be provided only with complete and well reviewed translations.

Practical recommendations for developers and maintainers

Many English words are polysemous: their meaning can only be determined from the context of their usage. As a practical consequence, the more context you provide, the more accurate the translation can be.

Example: recently, while downloading a software I saw something like this:
31min gauche
Go figure? After a while I realized that “left” had been translated “gauche” (as in “left hand”).

Also, order of words in a sentence vary upon language, furthermore not all languages are written left to right. Thus, mark entire paragraphs, or at least entire sentences, not lines, let alone isolated words but in special cases.

For instance, if text paragraphs were split in lines displayed by 'echo' commands, replace all consecutive 'echo' commands by a single 'gettext' or 'eval_gettext' command.

Do not fear to include the variable substitutions in the sentences, PO editor will check that they be present as is in the translations.

Recommendations for 'dialog' program.

The 'dialog' program provides an UI taking the form of dialog boxes.

There are other programs with similar feature, to which I guess (only a guess), these recommendations are also applicable.

Bear in mind following considerations, when making or reviewing the design choices for dialog's boxes.

- Messages translated in other languages will often be significantly longer than the original (usually in English) ones.
- In situations where only VGA drivers are available (e.g. in text installers) screen display is generally restricted to 25 rows of 80 columns with most widely used fonts, but in practice word wrapping can occur if line's length is more than 74 characters.
  As a consequence, for static layouts text lines' length should be at most 74 characters.
- Vertical scrolling of text is widely accepted as frequently used to display web pages, and sometimes unavoidable.
  On the contrary, horizontal scrolling should be avoided as much as possible.

Therefore I suggest to:

- renounce to tightly adjust the dimensions of the boxes to the size of English text as the translation will probably break your carefully crafted layout, unless you impose unreasonable (IMO) constraints to the translators,
- in particular, not narrow boxes' width to what is strictly needed for displaying English texts, especially in tabular layouts where the text can't flow on next lines,
- favor a fluid layout of the displayed text over a fixed one to avoid too long lines in translations, whose complete display would then necessitate horizontal scrolling (which, moreover, is not always possible).

In particular, I recommend to favor options which take as first argument a text string instead of a file, to allow line wrapping. It is still possible to preserve the intended layout using white spaces for indentation.

For instance,

```
dialog <common-options> --textbox <file> <height> <width>
```

can be replaced with

```
dialog --no-collapse <common-options> --msgbox "`cat <file>`" <height>
<width>
```

Practical recommendations for translators

Depending on amount of work needed and available resources, there can be one translator or a team of translators per target language. In all cases, I recommend that at least one person be responsible for organizing the team's work, checking the translations and transmitting the checked PO file to the maintainer(s). Let's call this person the team coordinator.

Don't feel obliged to translate verbatim. Not only is this rarely the best way to convey the meaning, but in addition this often leads to sentences too long to fit in allowed space.

Use a specialized PO editor, 'not' a general text editor. This will not only prevent inadvertently editing 'msgid' strings but also facilitate their work and automatize additional checks, as the presence of a variable in the translation with the same spelling as in the original.

While translating, choose a serif fixed width (or “monospaced”) font, like Courier. That allow to visually distinguish characters that otherwise would look the same, and check line's length when that matters.

If possible, check the layout of the messages. You could do that looking at the context in the relevant source file. Even better, simply run the translated script.

This is especially important if you are translating dialog boxes. In particular, take care not to write too long sentences on one single line if it appears that the text can't flow on next one.

Bear in mind that in VGA mode (used in text installers, in particular), line's width is limited theoretically to 80 characters, but practically often to 74.

Do not add question marks that are not present in the original message.

If the message refers to tags (text on the buttons) of dialog boxes, like “OK”, “Yes”, “NO”, “Continue”, “Cancel”, check how theses tags are translated in your language in dialog's interface and use the same words.

Avoid colloquialisms and technical slang.

To “cut” (or end) a line inside a “dialog” box you should type \n: pressing [Enter] will 'not' insert a “new line” character in the text viewed by user.

In addition, you will have to comply to gettext's requirements for it to work:

- If a word beginning with a dollar sign is included in the original text it should be present in the translation with exactly the same spelling (case matters).
- The translation text should include a “new line” character (or line feed, represented by “\n”) at the beginning or at the end, exactly as the original text does. Conversely, if the original text doesn’t have the character, then the translation shouldn’t have it.
- A single backslash character “\” is not allowed in the translation.

To check your translation against gettext's requirements you could run following command:

```
msgfmt -c <name of the PO file>
```

Warning about translation of man pages

Preserve carefully syntax of man pages found in English markup. For instance don't replace:

- 'B<' with 'B <' (don't insert a space)
- 'B<' with 'b<' (keep the B as a capital letter - and don't replace it by the Greek capital letter BETA that looks the same on the screen)
- “I” with '|' (don't replace the capital letter I with a pipe symbol)

When translating shell commands, preserve English names of paths when needed. But you may and should translate arguments to be replaced by a value like 'packagename'

Didier Spaier