LilyPond -- Easy guide for translators

LilyPond Contributor's Guide - the reference for developers as well as for translators - is not always up-to-date and the information is scattered through many paragraphs. This guide is intended to be a slim and possibly complete guide for whoever wants to quickly get started to work on the translation of LilyPond website and documentation.

Support

The very first step is subscribing to the mailing list dedicated to the translation work: `translations@lilynet.net`. Send an email to `translations-request@lilynet.net` with subject `subscribe` (body message can be empty). This is the place where you can ask for help in case of problems and also where you'll submit your translations. Before asking, please search in the translation list archive: you may find the answer to your question!

Configuring Git

In order to get the source code, you need to have Git installed. Then use these commands:

```
cd ~
git clone git://git.sv.gnu.org/lilypond.git
cd lilypond-git
git checkout -b translation origin/translation
```

The development of LilyPond is organized in several branches; the translation takes place in the *translation* branch. This command will tell you the current working branch:

```
$ git status
# On branch translation
```

You should configure at least name and email for your Git commits, in order to be credited in the git logs:

```
git config --global user.name "John Smith"
git config --global user.email john@example.com
```

To configure Git to use colored output where possible, use:

```
git config --global color.ui auto
```

I also suggest using the pre-commit hook, which catches introduction of lines with trailing whitespaces and aborts the commit when such a line is found:

```
mv .git/hooks/pre-commit.sample .git/hooks/pre-commit
```

You can skip the warning and commit without changing the offending line using:

```
git commit -n
```

Suggested reading:

- http://git-scm.com/book
- https://help.github.com/

Adding a new translation

All the translation work is made easy by some scripts included in the LilyPond source. To add a new language, run these commands:

```
cd ~/lilypond-git
./autogen.sh
cd Documentation
make ISOLANG=LANG new-lang
```

and then add a language definition for your language in `python/langdefs.py`.

> ***Note***
>
> All the scripts used for documentation maintenance must be run inside Documentation/. In the following instructions always replace LANG with your language code (e.g. *fr* for french).

Translation work

In the Contributor's Guide there's a list of priority files. You may want to follow it or just translate each manual in this order: website, learning manual and usage manual first; then notation manual, which is huge.

Texinfo

Translators are not required to know much of texinfo syntax. What is essential is adding @translationof to every @node, e.g.:

```
@node Ritmi
@section Ritmi
@translationof Rhythms
```

Cross-references:

- @ref are links to nodes within current manual. You can leave them untranslated, because they will be automatically translated, because of @translationof.
- @rlearning, @ruser, @rprogram are links to nodes in external manuals. You should translate them only if you have translated the linked file.

> ***Warning***
>
> Some cross-references, mostly to the Internal Reference manual, are broken because of a bug, see issue 2266.

You can use any text editor to work on texinfo files, but consider that Emacs has an excellent support for Texinfo. For example, you can see the document structure and click on it to jump to each section (menu Texinfo»Show Structure).

Useful shortcuts in Emacs:

- Ctrl + K: remove line
- Ctrl + C and Ctrl + Y: copy and paste
- Ctrl + X and Ctrl + S: save file
- Alt + Shift + %: search and replace (y to replace, n to skip)

Snippets

The documentation includes some snippets that are contributed by users through the user-friendly interface of the Lilypond Snippet Repository (LSR). You should translate the title and the description. Let's see how it works with an example; if you find something like:

```
@snippets

@lilypondfile[verbatim,quote,ragged-right,texidoc,doctitle]
{alternative-breve-notes.ly}
```

you should create the file Documentation/LANG/texidocs/alternative-breve-notes.texidoc and use this skeleton:

```
%% Translation of GIT committish: 6e4e9c4eacb94d68c11fd6b9062da4f724114860
  texidocit = "

"
  doctitleit = ""
```

Replace the "*it* suffix" with your LANG code and use an up-to-date committish.

Now open Documentation/snippets/alternative-breve-notes.ly, search the texidoc and doctitle sections and add their translation to your .texidoc file.

Credits

Translator credits should be put at the beginning of the translated files:

```
@c Translators: John Doe
@c Translation checkers: Jack Black
```

These credits are visible in the translation status page.

Committishes

Every translated file must have a committish at the beginning. The committish is a 40 character string which refers to a commit recorded in the git repository. The maintenance scripts compare the committish in the translated file with the committishes of the original (english) file and decide if it is outdated or not. So you must update the committish once you've finished your translation, otherwise the scripts will keep marking the file as outdated.

Which committish to use? You can use any committish which is newer than the committish of the original file. For example, the Contributor Guide suggests using the last committish in your branch, which can be found using this command:

```
git rev-list HEAD |head -1
```

Another way is using the last committish of the original file:

```
git log path/to/file
```

so you can know easily which version you updated (even if you'll probably never need this information).

Most of the changes in the LSR snippets included in the documentation concern the syntax, not the description inside texidoc="". This implies that quite often you will have to update only the committish of the matching .texidoc file. This can be a tedious work if there are many snippets to be marked as up-to-date. You can use the following command to update the committishes at once:

```
cd Documentation/LANG/texidocs
sed -i -r 's/[0-9a-z]{40}/NEW-COMMITTISH/' *.texidoc
```

Spell checking

You should install Aspell and your language dictionary. For example, in debian and ubuntu the italian package is called *aspell-it*. You can run it on the command line:

```
aspell -l LANG --mode=texinfo check file.itely
```

-l option defaults to locale language, so it's not required if your locale is setted to your mother language.

You can also use it within Emacs: Alt+x and type `ispell-buffer`. Then you can use these commands:

- `space` to ignore once
- a to always ignore for this session (in any buffer/file) -- recommended
- `i` to insert in the personal dictionary
- `<digit>` to replace with one of the proposed substitutions

The personal dictionary is located in the home: `~/.aspell.LANG.pws`.

Updating translated files

Before starting the update of your translations, you may want to see the list of outdated files:

```
make ISOLANG=LANG check-translation | grep 'diff --git'
```

Then you can see the changes in the terminal:

```
make ISOLANG=LANG check-translation | less -R
```

or save them in a text file and then open it in your favorite text editor:

```
make ISOLANG=LANG NO_COLOR=1  check-translation > ~/lilypond-translation.diff
```

Building the doc

Building the doc is absolutely required only if you have push access to the repository, because you must check that *make doc* compiles before pushing. But it's still recommended because you can catch the errors and fix them; then you can write to the mailing list and ask if someone can test and push your patch.

If you are using LilyDev, you should be set. Otherwise, this command *should* install all the required dependencies (it works on Debian/Ubuntu distros):

```
sudo apt-get build-dep lilypond
```

If some packages are missing, check the Contributor's Guide or ask the mailing list.

You compile the documentation by running these two commands in the root directory of your git repository:

```
cd ~/lilypond-git
make
make -j3 doc
```

> ***Note***
>
> Use the -jX option to speed up the build, where X is one more the number of cores of your CPU.

Snippets

When you add a new .texidoc file in Documentation/LANG/texidoc, `make doc` won't re-compile the snippets automatically. You have to change the modify time of the snippets and then re-compile the doc:

```
touch Documentation/snippets/*
make doc
```

Debugging

If `make doc` fails, it should print an error which asks you to check in a specific log file, which is saved in Documentation/LANG.

Patch or push

When you've done, commit your changes using a short description:

```
git add .
git commit -m "Doc-LANG: add chapter 3 of Notation Reference"
```

> ***Note***
>
> If you have enabled pre-commit (see above) to catch the trailing spaces, you might need to use the -n option to skip the verify. In case you want to split your changes in two commits, you can use `git add -u .` to add all the updated files but the new translated files.

If you don't have push access - which is likely the case if you are reading this guide - create a patch file and send it to the translators mailing list:

```
git format-patch origin translation
```

This command will create as many .patch files as the number of commits that you've created.

If you have push access, you can use this command:

```
git push origin translation
```