

Lora's user manual

Laurent Claessens

January 15, 2016

Lora is a backup program and a small Git remainder utility¹.

Warning

This is a free software² manipulating your precious data. If you use it, I consider you as a consenting adult.

Contents

1	Installation and compilation	2
1.1	The manual way	2
1.2	The graphical way	2
1.2.1	Compile the installation program	2
1.2.2	The backup tab	3
1.2.3	The terminal tab	4
1.2.4	The compilation tab	4
1.3	Compilation	5
2	Testing	5
3	Playing with Lora	5
3.1	Backup	5
3.2	Purge	6
3.3	The Git helper	6

¹Did you committed everything you modified today ?

²In the sense of the GPL : no warranty, etc.

1 Installation and compilation

The first step is to download the whole on [github](https://github.com/LaurentClaessens/lora) with

```
git clone https://github.com/LaurentClaessens/lora
```

You compile this documentation with

```
pdflatex manual.tex
```

A \LaTeX distribution is needed distribution is needed.

1.1 The manual way

1. Create your `lora.cfg` looking at `example.cfg` for example and explanations.
2. Modify the path of `BOOST_THREAD_LIB` in `makefile`.
3. Test with

```
./tests.sh
```

This should compile everything and launch a small testing program.

4. Launch Lora with

```
./lora
```

1.2 The graphical way

1.2.1 Compile the installation program

1. Compile the installation program :


```
make installation
```

2. Launch the installation program :

```
./installation
```

1.2.2 The backup tab

Here it is :



1. The backup directory (A) is the directory in which you want your `$HOME` to be copied. Typically it will be something like

`/mnt/backup_partition/backup.lora`

where `/mnt/backup_partition` is the mount point of an encrypted partition on an external drive.

2. The purge directory (B) is the directory in which modified and removed files will be copied (from the backup directory) before to the respectively copied from the home to the backup and removed from the backup directory. The main user case is «I accidentally deleted a file, I perform the backup (thus the file is deleted from the backup) and *then* I note that the file was removed.» In this case, the removed file (in the version that was available in the backup) can be retrieved in the purge directory.

Typically it will be something like

`/mnt/backup_partition/purge.lora`

Note that there are no automatic process removing the files from the purge directory. The size is then always increasing and you should remove very old subdirectory by hand.

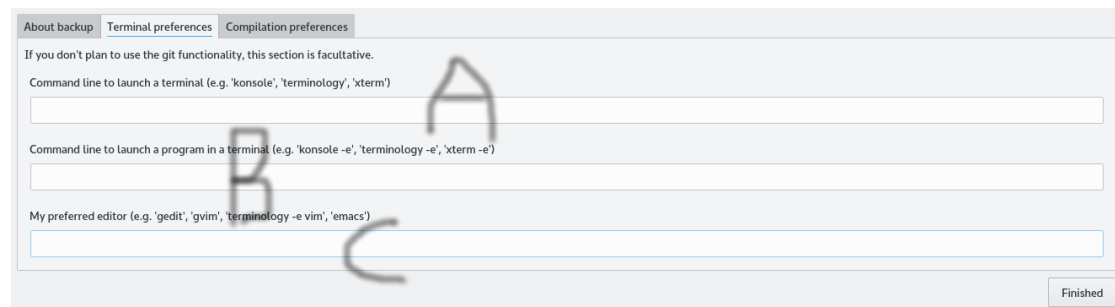
If you are using a program like Thunderbird, I let you know that it will **literally** produce **gibis** of data in the purge directory each day.

3. The excluded directories (C) are what they seem to be : they will be excluded from the backup. You can as example not backup the directory in which you save the `iso` files of your preferred Linux distribution. Press the `+` button to add an excluded directory.

1.2.3 The terminal tab

You can skip this if you don't use git.

The “Terminal tab” allows you to determine what kind of terminal and editor you prefer³. Here is the tab :



Terminal preferences

If you don't plan to use the git functionality, this section is facultative.

Command line to launch a terminal (e.g. 'konsole', 'terminology', 'xterm')

Command line to launch a program in a terminal (e.g. 'konsole -e', 'terminology -e', 'xterm -e')

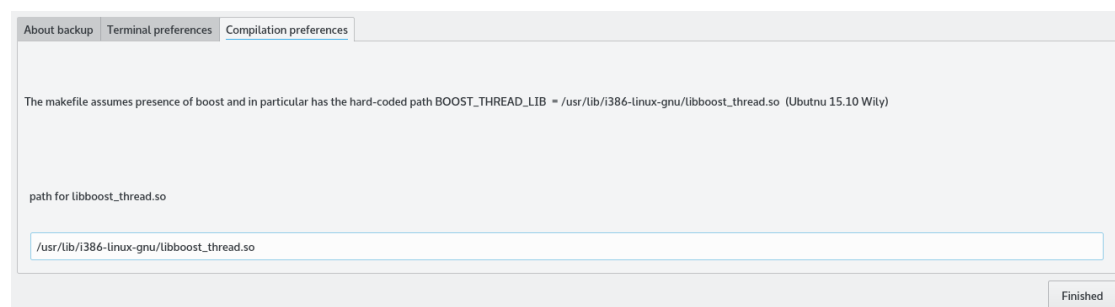
My preferred editor (e.g. 'gedit', 'gvim', 'terminology -e vim', 'emacs')

Finished

1. The line (A) ask you the command line that launch your favorite terminal. This will be `konsole`, `terminology`, `xterm` or something like that. This is used when clicking on “Open a terminal here”.
2. The line (B) ask you how to launch a process in your terminal. The point is that `git diff` has to be launched in a terminal in order to take into account you git preferences⁴.
3. The line (C) ask you your editor. This is used when clicking on “Edit .gitignore”. If your favorite editor works in a terminal, you need to ask for opening a new terminal, e.g. `konsole -e vim` instead of simply `vim`.

1.2.4 The compilation tab

Here it is :



Compilation preferences

The makefile assumes presence of boost and in particular has the hard-coded path BOOST_THREAD_LIB = /usr/lib/i386-linux-gnu/libboost_thread.so (Ubuntu 15.10 Wily)

path for libboost_thread.so

/usr/lib/i386-linux-gnu/libboost_thread.so

Finished

This asks you the path of the file `libboost_thread.so` that is mandatory for the compilation of Lora.

³Serious people use `gedit vim` or `notepademacs` inside `konsole command.com` terminology.

⁴By the way, you should add `quotepath=false` in your `.gitconfig` file.

1.3 Compilation

You need Boost to be installed somewhere on your system⁵. The path to the file `libboost_thread.so` has to be correct in the makefile (edit it if you are unsure).

For compiling everything you simply type

```
make all
```

2 Testing

You can test Lora by

```
./tests.sh
```

This will test some functionalities (for regressions) and allow you to see a Git window. If something gets wrong, send me an email at `laurent.claesens@studenti.unipd.it`

3 Playing with Lora

3.1 Backup

Launch

```
./lora -configuration=<configuration_filename> <starting_path>
```

Configuration Default is `lora.cfg`. The filename in which Lora has to read your configuration.

Starting path Default is `$HOME`. Giving the argument overrides the starting path written in the configuration file. If you want to backup only a part of your home because you only have 5 minutes to shut down your computer.

Most of time you only have to launch `./lora` with no arguments.

Lora will loop over your home directory (or `starting_path` if given in command line or in the configuration file) and compare⁶ each file with the corresponding one in your backup directory.

1. If `<home>/foo/bar.txt` differs from `<backup>/foo/bar.txt`.

⁵On my Ubuntu, I do `apt install libboost-all-dev libboost1.58-all-dev`

⁶Files are different if they size differ or if they `last_write_time` attribute differ.

- Move `<backup>/foo/bar.txt` to `<purge>/foo/bar.txt` Thus you will never loose data⁷. In this sense, Lora is more than a *synchronizing* program. It synchronizes and keeps the old data.
- Copy `<home>/foo/bar.txt` to `<backup>/foo/bar.txt`

If `<home>/foo/bar.txt` exists and `<backup>/foo/bar.txt` does not exist.

- Copy.

3.2 Purge

When the backup is finished, Lora will loop over your `<backup>` directory.

If `<backup>/foo/bar.txt` exists and `<home>/foo/bar.txt` does not exist, we deduce that you removed your file.

- Move `<backup>/foo/bar.txt` to `<purge>/foo/bar.txt`.

Note. The purge directory here is not exactly the same as the previous one. In fact each time you launch Lora, a new directory is created :

`<purge>/<date>/<hour-minutes>`

Inside that directory, Lora creates two directories : `modified` and `removed` which contain the files that were respectively seen to be modified and removed.

A general concept of Lora is that your data is more precious than your disk space and than everything⁸. If you understand well the backup/purge concept, you can imagine the extreme disk space waste when you just rename, say your music directory.

3.3 The Git helper

Since Lora loops over the whole `$HOME`, it also takes time to check for each directory if it is a git repository (has non trivial `.git` subdirectory) and if this repository is clean⁹.

A list of directories that are not clean git repository (untracked or modified files) is displayed. Clicking on one of them opens a dialog window that helps you to manage the situation.

Here is a window example :

⁷Never in the sense of the GPL no warranty stuff...

⁸Better to crash than to manage a borderline situation.

⁹I personally gitted directories like `.config` and `.kde`, so I am not always conscient that a git repository have been modified.

/home/moky/script/lora

add vim to gitignore

add latex to gitignore

add C++ to gitignore

add python to gitignore

See git diff

Edit .gitignore

Launch git commit -a

Open a terminal here

Exit

Modified files :
Configuration.cpp
README.txt
UnitTests.cpp
manual.tex

test1.txt ☐ add ☐ gitignore ☒ no action

test2.txt ☐ add ☐ gitignore ☒ no action

Apply these changes

Track these files

Ignore these files

Commit that

1. A list of modified files (A).
2. A list of untracked files (B). For each you can choice to track it (add) or ignore it (add to `.gitignore`). When clicking on “Apply these changes” (C), Lora

does `git add` for the files you checked “add” and appends to `.gitignore` the names that you checked as “gitignore”.

3. The “Track these files” line (D). Lora will perform `git add` with the given argument (like `*.cpp`, `*.tex`).
4. The “Ignore these files” line (E). Lora will append the files given in argument into `.gitignore` (like `*.tmp`, `*.eps`).
5. The “Commit that” line (F) performs `git commit -message=` with the given message.

Add <this> to gitignore Click there is this repository is of <this> type and Lora will add corresponding files to `.gitignore`. Example : with “add latex to gitignore”, it will add `*.aux`, `*.toc`, etc.

See git diff open a terminal and launch `git diff` inside.

Edit .gitignore open `.gitignore` in your preferred editor.

Launch git commit -a Open a terminal and launches `git commit -a` inside.

Open a terminal here does what you think it does.

Exit does what you think it does.