# Relazione di progetto
# Lora

Laurent Claessens

16 febbraio 2016

## Indice



# 1 Scopo et descrizione generale del progetto

## 1.1 Cosa fa ?

Lora è un programmo che fa due cose.

- Fa un backup della cartelle `$HOME`
- Cerca le cartelle che sono *git repository*, verifica se hanno bisogno di un `git add`, `git commit` oppure di aggiungere dei file dentro `.gitignore`. Un interfaccia grafica aiuta a questo tipo di manutenzione.

## 1.2 L'algoritmo di backup

Il programma si invoca con[1]

```
./lora
```

In questo esempio, supponiamo fare il backup di `<home>` (che sarà usualmente la cartella `$HOME`) dentro la cartella `<backup>` (che carà su un disco esterno[2]). Di più abbiamo bisogno di una cartelle `<purge>` accanto a `<backup>`.

Lora fa un *loop* su tutti i file di `$HOME`.

1. Se `<home>/foo/bar.txt` è diverso[3] da `<backup>/foo/bar.txt`.

[1] Vedere il manuale del'utente per maggiore informazioni.
[2] Che dovrebbe essere cifrato, però è un altra storia.
[3] L'attributo *size* o *last write time* è diverso.

- Sposta `<backup>/foo/bar.txt` verso `<purge>/foo/bar.txt` Quindi non ci sarà mai una perdita di dati : Lora è più di un programma di sincronizzazione. Lora fa la sincronizzationne di `<home>` con `<backup>`, ma guarda i viechi files in `<purge>`.
- Copia `<home>/foo/bar.txt` su `<backup>/foo/bar.txt`

Se `<home>/foo/bar.txt` ma `<backup>/foo/bar.txt` non esista.

- Copia.

Vantaggio di Lora su altri programmi di backup[4] : i dati si ritrovano con una semplice copia di `<backup>` si `<home>`. L'utente non ha bisogno si Lora per ritrovare i dati.

In particolare, l'utente può ritrovare i sui dati anche da una chiave USB minimale senza interfaccia grafica.

## 1.3 L'algoritmo di purge

Quando il backup è finito, Lora fa un *loop* su la cartella `<bakcup>`.

Se `<backup>/foo/bar.txt` esista ma `<home>/foo/bar.txt` non esista, Lora pensa che l'utente abbia eliminato questo file.

- Sposta `<backup>/foo/bar.txt` verso `<purge>/foo/bar.txt`.

Nota bene. 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 everything[5]. If you understand well the backup/purge concept, you can imagine the extreme disk space waste when you just rename, say your music directory.

## 1.4 I dati prima di tutto

Questo programmo è destinato al "power user" : l'utente dovrebbe guardate (un po) coda succede dentro il terminale per tenersi al corrente di cosa succede sul suo disco. Per esempio, vedere 2000 files passare dalla purge significa probabilmente che l'utente abbia soppresso una cartella.

Lora contiene ancora un certo numero di casi in cui va in *crash*. Per esempio quando un file è soppresso dal utente tra il momento in cui è visto come "da aggiornare dentro `<backup>`" e il momento in cui la copia è effettivamente fatta[6].

In questi casi, Lora preferisce lasciare al l'utente la responsabilità di capire cos'ha fatto con i suoi dati piuttosto che prendere un iniziativa. Quindi *crash* piuttosto che gestione.

Ci sono dunque un bel po di eccezione di tipo `std::string` che sono sollevate e gestite solo alla fine del'`main`, sotto la forma di stampa semplice.

In ogni casi, se succede qualcosa di strano con i dati, l'utente deve essere avvertito. Fina ora, l'utente è avvertito sotto la forma d'un *crash*. C'è però una proposta dentro `TODO.txt` di usare il sistema di log (vedere `Logging.cpp`) per scrivere i problemi dentro un file e stampare il file alla fine del'esecuzione.

[4] Sopratutto quelli che implementano un backup incrementale.

[5] Better to crash than to manage a borderline situation.

[6] Mi capita spesso con dei file ausiliari di LaTeX quando faccio una compilazione nello stesso tempo di un backup.

## 1.5 Multi-thread

Il backup stesso occupa 2 threads.

- Il primo è un *loop* sulla cartella `$HOME` che alla ricerca di files o cartelle di cui aggiogare il backup. Questo thread aggiunge delle *tasks* (vedere 2) a una lista.
- Il secondo thread legge la lista e esegue le copie da fare.

Questi threads non hanno interfaccia grafica, e questo è una scelta di design : devo poter fare un backup anche (e soprattutto) quando il computer va male. Per esempio voglio poter fare il backup da une chiave USB minimale[7].

Nello stesso tempo, un elenco dei *git repository* che hanno bisogno di pulizia si aggiorna in un interfaccia grafica.

Per queste ragione, i processi sono lanciate e chiuse in questo modo :

```
1  Configuration* config_ptr=arguments_to_configuration(argc,argv);
2
3  GitListWindow* git_list_window=new GitListWindow(config_ptr);
4  git_list_window->show();
5
6  boost::thread task_runner( run_tasks, config_ptr );
7  loops(config_ptr);
8
9  task_runner.join();
10 git_list_window->join();
```

- Leggere il file di configurazione.
- Creare e mostrare l'interfaccia grafica che elenca i *git repository* (`GitListWindow`).
- Lanciare il thread che legge l'elenco dei task da fare e esegue le task (`task_runner`). A l'inizio, l'elenco è vuota, ovviamente.
- Lanciare il backup e la purge (`loops`), quindi iniziare a popolare l'elenco delle task da fare.
- Aspettare che tutte le task siano finite (`task_runner.join()`).
- Aspettare che l'interfaccia grafica sia chiusa (`git_list_windows.join()`).

## 1.6 Perché è cosi grosso ?

Avevo cominciato il lavoro al'inizio di Ottobre 2015 perché avevo bisogno di un programmo di backup per uso personale. La parte di backup era già quasi finita quando abbiamo a parlare del progetto in classe.

Quindi per il progetto stesso, ho aggiunto la parte "git" e l'interfaccia grafica.

Di più, questo è un vero programmo che uso veramente per gestire i miei preziosi dati; quindi preferisco avere qualcosa che abbia le funzionalità utile. Storia interessante : avevo già gatto lo stesso programmo molti anni fa[8] in Python (senza interfaccia grafica). Per finire il lavoro, gli serviva a volte più di trenta minuti. Lora finisce raramente in più di 6 minuti.

[7] Dal punto di vista del codice, il backup è abbastanza ben separato delle dipendenze su Qt. Dovrei scrivere un giorno una versione che si può veramente eseguire fuori del'interfacia grafica.

[8] Le scelte di design sono basate su dei casi di uso reali.

## 2 Gerarchia polimorfa : `GenericTask`

## 3 Gerarchia polimorfa : `MainLoop`