Software Engineering at MPRI - Tutorial on the version control system git, and its extensions

Amélie Ledein
ledein@lsv.fr

October 9, 2020

Introduction

Don't forget ...

Introduction

Don't forget ...

- **Rigor**
  - $\rightarrow$ Code, methodology, specifications and tests

Introduction

Don't forget ...

- **Rigor**
  - $\rightarrow$ Code, methodology, specifications and tests
- **Adaptability**
  - $\rightarrow$ Components could be re-used in a (slightly different) context.

Introduction

Don't forget ...

- **Rigor**
  - $\rightarrow$ Code, methodology, specifications and tests
- **Adaptability**
  - $\rightarrow$ Components could be re-used in a (slightly different) context.
- **Modularity**
  - $\rightarrow$ Segment project in modules with clearly defined interfaces.
  - $\rightarrow$ Develop and test independently, facilitate re-use and change.

Introduction

Don't forget ...

- **Rigor**
  - $\rightarrow$ Code, methodology, specifications and tests
- **Adaptability**
  - $\rightarrow$ Components could be re-used in a (slightly different) context.
- **Modularity**
  - $\rightarrow$ Segment project in modules with clearly defined interfaces.
  - $\rightarrow$ Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - $\rightarrow$ Do not specify implementation details, i.e. things that can easily change.

Introduction

Don't forget ...

- **Rigor**
  - → Code, methodology, specifications and tests
- **Adaptability**
  - → Components could be re-used in a (slightly different) context.
- **Modularity**
  - → Segment project in modules with clearly defined interfaces.
  - → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

Introduction

Don't forget ...

- **Rigor**
  - → Code, methodology, specifications and tests
- **Adaptability**
  - → Components could be re-used in a (slightly different) context.
- **Modularity**
  - → Segment project in modules with clearly defined interfaces.
  - → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- USB-key

Introduction

Don't forget ...

- **Rigor**
  - → Code, methodology, specifications and tests
- **Adaptability**
  - → Components could be re-used in a (slightly different) context.
- **Modularity**
  - → Segment project in modules with clearly defined interfaces.
  - → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- ~~USB-key~~

Introduction

Don't forget ...

- **Rigor**
  - → Code, methodology, specifications and tests
- **Adaptability**
  - → Components could be re-used in a (slightly different) context.
- **Modularity**
  - → Segment project in modules with clearly defined interfaces.
  - → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- ~~USB-key~~
- Email

Introduction

Don't forget ...

- **Rigor**
  - → Code, methodology, specifications and tests
- **Adaptability**
  - → Components could be re-used in a (slightly different) context.
- **Modularity**
  - → Segment project in modules with clearly defined interfaces.
  - → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  - → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- ~~USB-key~~
- ~~Email~~

Introduction

Don't forget ...

- **Rigor**
  $\rightarrow$ Code, methodology, specifications and tests
- **Adaptability**
  $\rightarrow$ Components could be re-used in a (slightly different) context.
- **Modularity**
  $\rightarrow$ Segment project in modules with clearly defined interfaces.
  $\rightarrow$ Develop and test independently, facilitate re-use and change.
- **Abstraction**
  $\rightarrow$ Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- ~~USB-key~~
- ~~Email~~
- Dropbox, Google Drive, etc.

Introduction

Don't forget ...

- **Rigor**
  → Code, methodology, specifications and tests
- **Adaptability**
  → Components could be re-used in a (slightly different) context.
- **Modularity**
  → Segment project in modules with clearly defined interfaces.
  → Develop and test independently, facilitate re-use and change.
- **Abstraction**
  → Do not specify implementation details, i.e. things that can easily change.

OK, but how can you share your code?

- ~~USB-key~~
- ~~Email~~
- ~~Dropbox, Google Drive, etc.~~

The answer is ...

The answer is ...

or any version control systems (VCS).
(*systèmes de gestion de version*, in French)

Sommaire

Sommaire

What is a version control system?

What is a version control system?

- Software system that allows you to maintain and manage all versions of a set of files.

What is a version control system?

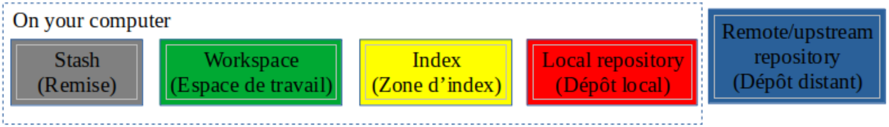
- Software system that allows you to maintain and manage all versions of a set of files.
- Why a version management system?
    - Revolve easily to a previous version.
    - Follow the evolution of the project over time.
    - Allow parallel work on disjointed parts of the project and manage the competing modifications.
    - Facilitate the detection and correction of errors.
    - etc.

What is a version control system?

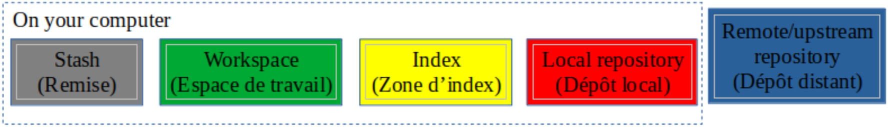
- Software system that allows you to maintain and manage all versions of a set of files.
- Why a version management system?
  - Revolve easily to a previous version.
  - Follow the evolution of the project over time.
  - Allow parallel work on disjointed parts of the project and manage the competing modifications.
  - Facilitate the detection and correction of errors.
  - etc.
- An example:

$\rightarrow$ A free and open source distributed version control system (DVCS)
$\rightarrow$ Designed to handle everything from small to very large projects with speed and efficiency

Git in detail

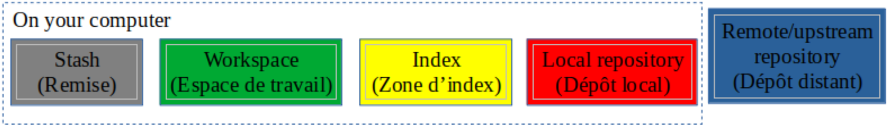

On your computer
Stash (Remise)
Workspace (Espace de travail)
Index (Zone d'index)
Local repository (Dépôt local)
Remote/upstream repository (Dépôt distant)

Git in detail


On your computer
Stash (Remise)
Workspace (Espace de travail)
Index (Zone d'index)
Local repository (Dépôt local)
Remote/upstream repository (Dépôt distant)

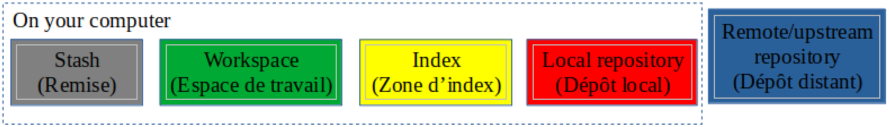
- **Stash** (**remise**): A place to hide modifications while you work on something else.

Git in detail



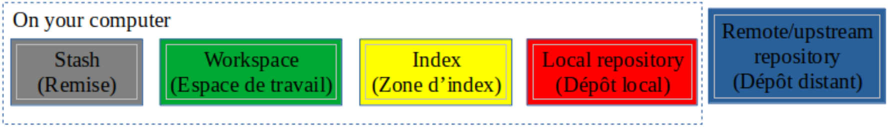
- **Stash** (**remise**): A place to hide modifications while you work on something else.
- **Workspace** (**espace de travail**): Local checkout.

Git in detail


On your computer
Stash (Remise)
Workspace (Espace de travail)
Index (Zone d'index)
Local repository (Dépôt local)
Remote/upstream repository (Dépôt distant)

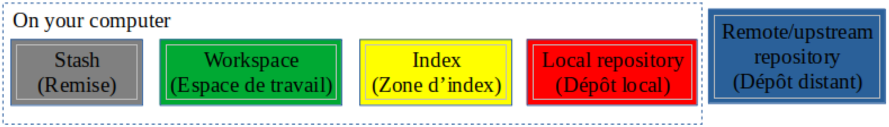
- **Stash** (**remise**): A place to hide modifications while you work on something else.
- **Workspace** (**espace de travail**): Local checkout.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache: Files you want to commit. Before you "commit" (checkin) files, you need to first add them to the index.

Git in detail


On your computer
Stash (Remise)
Workspace (Espace de travail)
Index (Zone d'index)
Local repository (Dépôt local)
Remote/upstream repository (Dépôt distant)

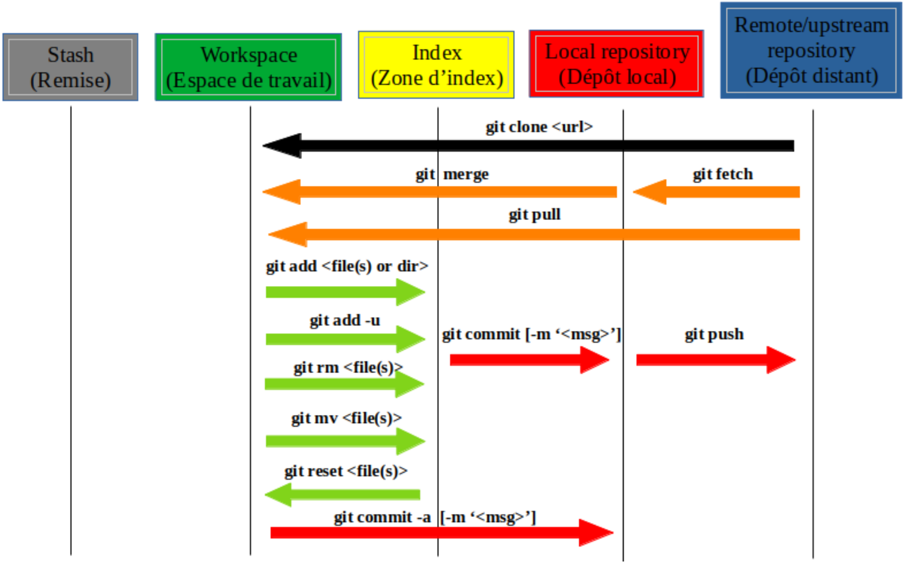
- **Stash** (**remise**): A place to hide modifications while you work on something else.
- **Workspace** (**espace de travail**): Local checkout.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache: Files you want to commit. Before you "commit" (checkin) files, you need to first add them to the index.
- **Local repository** (**dépôt local**): A subdirectory named `.git` that contains all of your necessary repository files - a Git repository skeleton.

Git in detail



- **Stash** (**remise**): A place to hide modifications while you work on something else.
- **Workspace** (**espace de travail**): Local checkout.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache: Files you want to commit. Before you "commit" (checkin) files, you need to first add them to the index.
- **Local repository** (**dépôt local**): A subdirectory named `.git` that contains all of your necessary repository files - a Git repository skeleton.
- **Remote/upstream repository** (**dépôt distant**): Versions of your project that are hosted on the Internet or network, ensuring all your changes are available for other developers.
  The default name is `origin`.

Sommaire


Stash
(Remise)
Workspace
(Espace de travail)
Index
(Zone d'index)
Local repository
(Dépôt local)
Remote/upstream repository
(Dépôt distant)
git clone <url>
git merge
git fetch
git pull
git add <file(s) or dir>
git add -u
git commit [-m '<msg>']
git push
git rm <file(s)>
git mv <file(s)>
git reset <file(s)>
git commit -a [-m '<msg>']

Synchronization with the remote repository

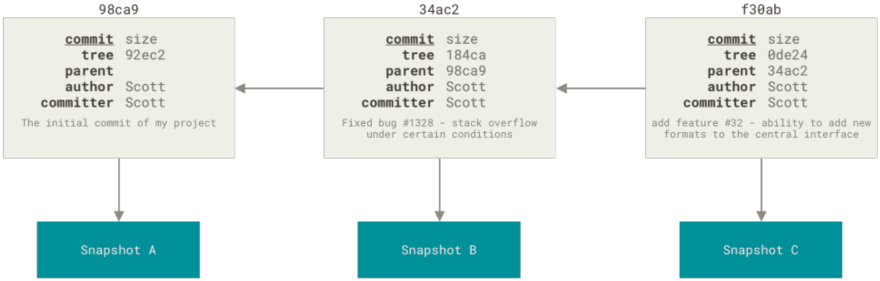
```
$ git clone <url>
```
Retrieve an entire repository from hosted location via URL.

```
$ git fetch <alias>
```
Fetch down all the branches from that Git remote.

```
$ git merge <alias/<branch>
```
Merge a remote branch into your current branch to bring it up to date.

```
$ git pull
```
Fetch and merge any commits from the tracking remote branch.

Commit


98ca9
commit size
tree 92ec2
parent
author Scott
committer Scott
The initial commit of my project
34ac2
commit size
tree 184ca
parent 98ca9
author Scott
committer Scott
Fixed bug #1328 - stack overflow under certain conditions
f30ab
commit size
tree 0de24
parent 34ac2
author Scott
committer Scott
add feature #32 - ability to add new formats to the central interface
Snapshot A
Snapshot B
Snapshot C

Make modifications

```
$ git add <file(s)>
```
Add a file (or several) as it looks now to your next commit (stage).

```
$ git rm <file(s)>
```
Delete the file (or several) from the project and stage the removal for commit.

```
$ git mv <old-name-file> <new-name-file>
```
Rename the file and stage the renaming.

```
$ git mv <existing-path> <new-path>
```
Change an existing file path and stage the move.

```
$ git reset <file(s)>
```
Unstage a file (or several) while retaining the changes in working directory.

```
$ git commit [-m "<descriptive message>]"
```
Commit your staged content as a new commit snapshot.

```
$ git push <alias> <branch>
```
Transmit local branch commits to the remote repository branch.

Exercise 1

1. Clone the repository from `https://github.com/amelieled/SE_GIT_MPRI.git`
2. Add at least 5 new items in the grocery list.
3. Fix the 5 errors.
4. Add a new section.

Informative commands

- Setup:
  Configuring user information used across all local repositories.
  - `$ git config --global user.name "[firstname lastname]"`
    Set a name that is identifiable for credit when review version history.
  - `$ git config --global user.email "[valid-email]"`
    Set a email address that will be associated with each history marker.

  Note : `export EDITOR=emacs` (or `vim`, etc.)
  To configure correctly your editor with Git.
- To collect information:
  - `$ git status`
    Show modified files in working directory, staged for your next commit.
  - `$ git diff`
    Diff of what is changed but not staged.
  - `$ git diff --staged`
    Diff of what is staged but not yet committed.

Sommaire

Branch

```
$ git branch
```
List your branches.
A star (*) will appear next to the currently active branch.

```
$ git branch <branch-name>
```
Create a new branch at the current commit.

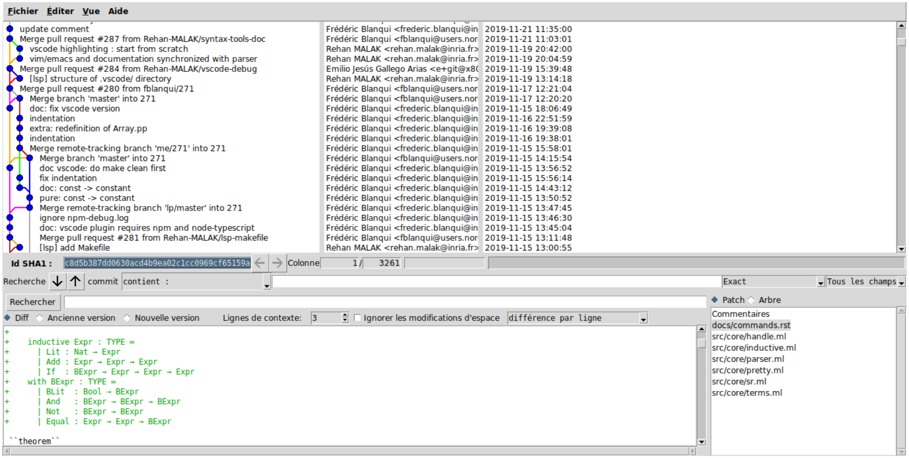
```
$ git branch -d <branch-name>
```
Delete the specified branch.

```
$ git checkout <branch-name>
```
Switch to another branch and check it out into your working directory.

```
$ git merge <branch-name>
```
Merge the specified branch's history into the current one.

```
$ git log
```
Show all commits in the current branch's history.

Gitk - Graphical interface



```
update comment                                               Frédéric Blanqui <frederic.blanqui@in  2019-11-21 11:35:00
Merge pull request #287 from Rehan-MALAK/syntax-tools-doc    Frédéric Blanqui <fblanqui@users.nor   2019-11-21 11:03:01
  vscode highlighting : start from scratch                   Rehan MALAK <rehan.malak@inria.fr>     2019-11-19 20:42:00
  vim/emacs and documentation synchronized with parser       Rehan MALAK <rehan.malak@inria.fr>     2019-11-19 20:04:59
Merge pull request #284 from Rehan-MALAK/vscode-debug        Emilio Jesús Gallego Arias <e+git@x8(  2019-11-19 15:39:48
  [lsp] structure of .vscode/ directory                      Rehan MALAK <rehan.malak@inria.fr>     2019-11-19 13:14:18
Merge pull request #280 from fblanqui/271                    Frédéric Blanqui <fblanqui@users.nor   2019-11-17 12:21:04
  Merge branch 'master' into 271                             Frédéric Blanqui <fblanqui@users.nor   2019-11-17 12:20:20
  doc: fix vscode version                                    Frédéric Blanqui <frederic.blanqui@in  2019-11-15 18:06:49
  indentation                                                Frédéric Blanqui <frederic.blanqui@in  2019-11-16 22:51:59
  extra: redefinition of Array.pp                            Frédéric Blanqui <frederic.blanqui@in  2019-11-16 19:39:08
  indentation                                                Frédéric Blanqui <frederic.blanqui@in  2019-11-16 19:38:01
  Merge remote-tracking branch 'me/271' into 271             Frédéric Blanqui <frederic.blanqui@in  2019-11-15 15:58:01
    Merge branch 'master' into 271                           Frédéric Blanqui <fblanqui@users.nor   2019-11-15 14:15:54
    doc vscode: do make clean first                          Frédéric Blanqui <frederic.blanqui@in  2019-11-15 13:56:52
    fix indentation                                          Frédéric Blanqui <frederic.blanqui@in  2019-11-15 15:56:14
    doc: const -> constant                                   Frédéric Blanqui <frederic.blanqui@in  2019-11-15 14:43:12
    pure: const -> constant                                  Frédéric Blanqui <frederic.blanqui@in  2019-11-15 13:50:52
    Merge remote-tracking branch 'lp/master' into 271        Frédéric Blanqui <frederic.blanqui@in  2019-11-15 13:47:45
    ignore npm-debug.log                                     Frédéric Blanqui <frederic.blanqui@in  2019-11-15 13:46:30
    doc: vscode plugin requires npm and node-typescript      Frédéric Blanqui <frederic.blanqui@in  2019-11-15 13:45:04
    Merge pull request #281 from Rehan-MALAK/lsp-makefile    Frédéric Blanqui <fblanqui@users.nor   2019-11-15 13:11:48
    [lsp] add Makefile                                       Rehan MALAK <rehan.malak@inria.fr>     2019-11-15 13:00:55
```

Id SHA1 : c8d5b387dd0630acd4b9ea02c1cc0969cf65159a Colonne 1 / 3261

Recherche commit contient : Exact Tous les champs

◆ Diff ◇ Ancienne version ◇ Nouvelle version Lignes de contexte: 3 ☐ Ignorer les modifications d'espace différence par ligne

```
+
+    inductive Expr : TYPE ≔
+      | Lit : Nat → Expr
+      | Add : Expr → Expr → Expr
+      | If  : BExpr → Expr → Expr → Expr
+    with BExpr : TYPE ≔
+      | BLit  : Bool → BExpr
+      | And   : BExpr → BExpr → BExpr
+      | Not   : BExpr → BExpr
+      | Equal : Expr → Expr → BExpr

 ``theorem``
```

Commentaires
docs/commands.rst
src/core/handle.ml
src/core/inductive.ml
src/core/parser.ml
src/core/pretty.ml
src/core/sr.ml
src/core/terms.ml

Or if you prefer: `git log --graph`.

Examining logs, diffs and object information

```
$ git log branchB..branchA
```
Show the commits on branchA that are not on branchB.

```
$ git log --follow <file>
```
Show the commits that changed file, even across renames.

```
$ git diff branchB...branchA
```
Show the diff of what is in branchA that is not in branchB.

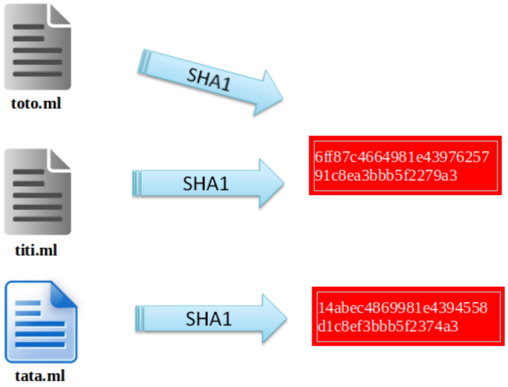
```
$ git log --stat -M
```
Show all commit logs with indication of any paths that moved.

```
$ git show <SHA>
```
Show any object in Git in human-readable format.

$\rightarrow$ Easier on Github (See later)

Sommaire

SHA1 is a hashing algorithm taking an input up to 2^{64} bits, and returns a unique sequence of 40 hexadecimal characters.



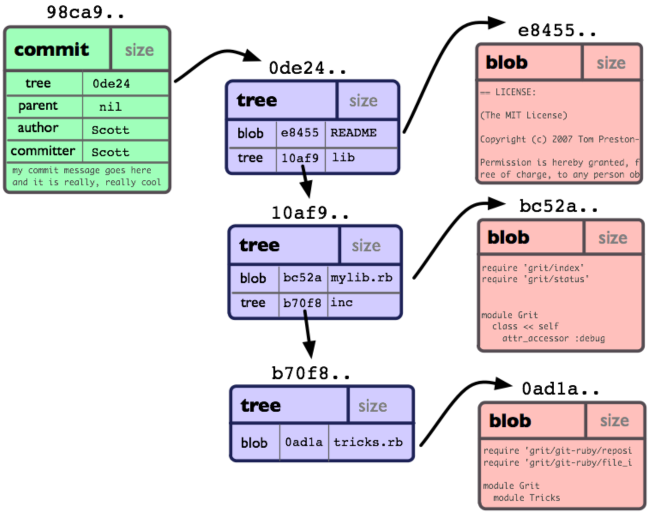
By hashing the contents of a file, Git obtains a series of unique digits symbolizing the file. Then, Git backs up only the files which are different hash (Git does not care about the names of the files, it only considers the content.).

Git objects

There are four Git objects:

- The **Blob** (*Binary Large Object*): It more commonly represents a **file**.
- The **Tree**: It more commonly represents a **directory** or **folder** of your application.
  Its content is the list of SHA1s of Blobs or other Trees that it can contain. What gives a tree structure of files.
- The **Commit**: This is the complete state of your project at a given moment, i.e. a **snapshot**.
  Its content is the SHA1 of the source Tree, and various information such as the name of the commit, the name of the author, the date, etc.
- The **Tag**: This is an object used to qualify a **particular commit** by giving it a comment.

An example



Each element has a unique SHA1.

How Git stores its information?

Only thanks to the directory `.git` at the root of your project.

- **config**: file relating to the configuration of the Git environment, such as information about the developer (name, email, etc.);
- **description**: contains information about your project;
- **objects/**: it is in this directory that all Git objects are stored (commits, tags, trees, blobs);
- **refs/*:** contains information on local branches of the repository;
- **logs/*:** contains log messages;
- **index**: file containing information about the status of the next commit.
- **HEAD**: pointer to current branch;
- **hooks/**: folder containing "hooks" or "triggers", i.e. actions/scripts that can be executed in pre or post condition.

Some questions

Some questions

- **How find a particular object?**

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob "4A558..."?**

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob "4A558..."?**
  See on /.git/objects/4A/558...

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob "4A558..."?**
  See on /.git/objects/4A/558...

- **This structure is too heavy?**

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob "4A558..."?**
  See on /.git/objects/4A/558...

- **This structure is too heavy?**
  The answer is no thanks to the "Zlib" compression system that Git uses to compress the data and maintain the correct weight.

Some questions

- **How find a particular object?**
  Thanks to SHA1, in particular: .git/objects/
  Example: **How find the blob "4A558..."?**
  See on /.git/objects/4A/558...

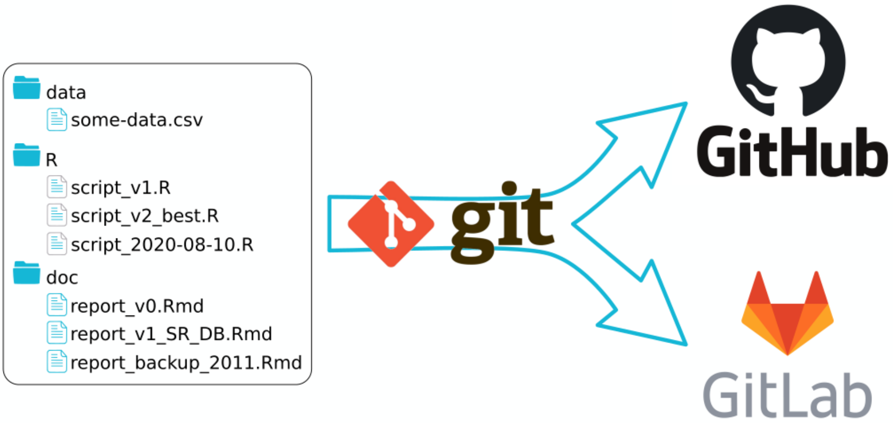
- **This structure is too heavy?**
  The answer is no thanks to the "Zlib" compression system that Git uses to compress the data and maintain the correct weight.

  Demonstration:

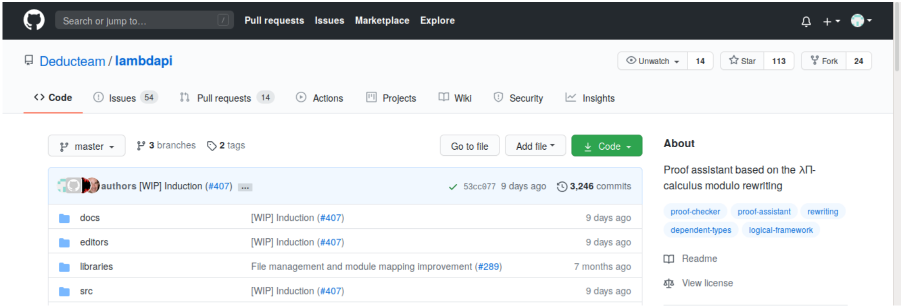
  ```
  $ sudo apt install qpdf
  $ zlib-flate -uncompress < FILE
  ```

Sommaire

Git and GitHub/GitLab



Graphical interface


Deducteam / lambdapi

authors [WIP] Induction (#407) ... 53cc077 9 days ago 3,246 commits

docs	[WIP] Induction (#407)	9 days ago
editors	[WIP] Induction (#407)	9 days ago
libraries	File management and module mapping improvement (#289)	7 months ago
src	[WIP] Induction (#407)	9 days ago

About

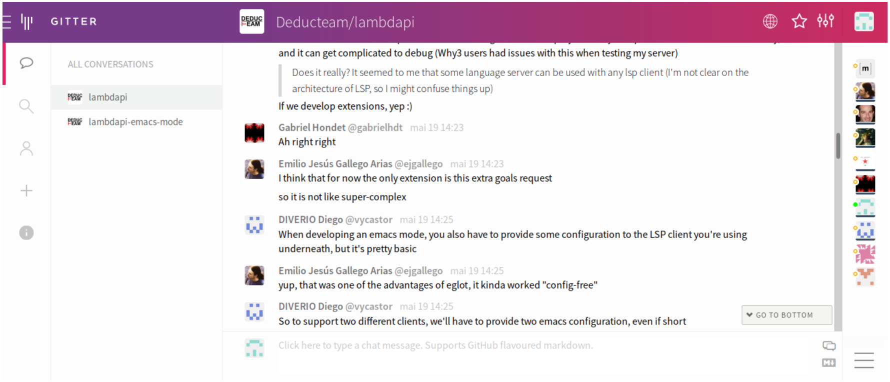
Proof assistant based on the λΠ-calculus modulo rewriting

- See current code
- See each commit
- See each issue
- Do integration continuous
- etc.

Continuous integration - Testing

Travis CI

Gitter



and it can get complicated to debug (Why3 users had issues with this when testing my server)

> Does it really? It seemed to me that some language server can be used with any lsp client (I'm not clear on the architecture of LSP, so I might confuse things up)

If we develop extensions, yep :)

Gabriel Hondet @gabrielhdt mai 19 14:23
Ah right right

Emilio Jesús Gallego Arias @ejgallego mai 19 14:23
I think that for now the only extension is this extra goals request

so it is not like super-complex

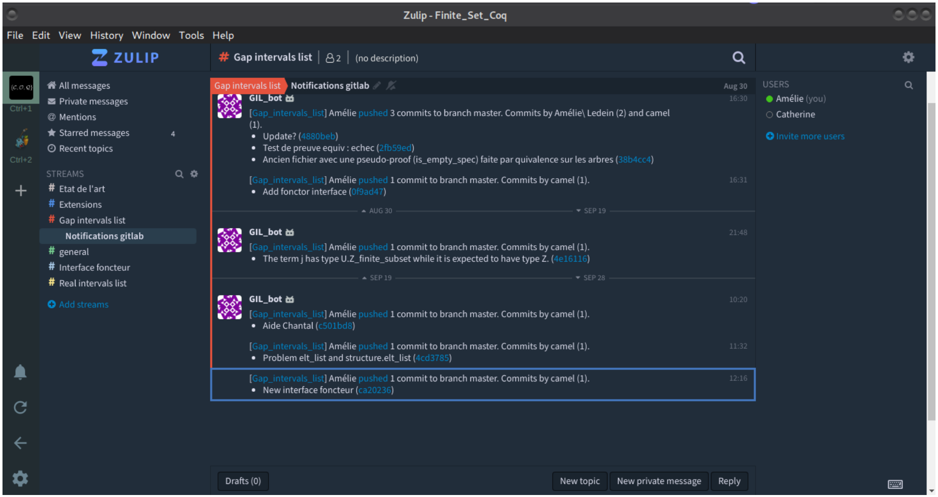
DIVERIO Diego @vycastor mai 19 14:25
When developing an emacs mode, you also have to provide some configuration to the LSP client you're using underneath, but it's pretty basic

Emilio Jesús Gallego Arias @ejgallego mai 19 14:25
yup, that was one of the advantages of eglot, it kinda worked "config-free"

DIVERIO Diego @vycastor mai 19 14:25
So to support two different clients, we'll have to provide two emacs configuration, even if short

Click here to type a chat message. Supports GitHub flavoured markdown.

Zulip



Gap intervals list Notifications gitlab Aug 30

GIL_bot 16:30
[Gap_intervals_list] Amélie pushed 3 commits to branch master. Commits by Amélie\ Ledein (2) and camel (1).
- Update? (4880beb)
- Test de preuve equiv : echec (2fb59ed)
- Ancien fichier avec une pseudo-proof (is_empty_spec) faite par quivalence sur les arbres (38b4cc4)

[Gap_intervals_list] Amélie pushed 1 commit to branch master. Commits by camel (1). 16:31
- Add fonctor interface (0f9ad47)

AUG 30 — SEP 19

GIL_bot 21:48
[Gap_intervals_list] Amélie pushed 1 commit to branch master. Commits by camel (1).
- The term j has type U.Z_finite_subset while it is expected to have type Z. (4e16116)

SEP 19 — SEP 28

GIL_bot 10:20
[Gap_intervals_list] Amélie pushed 1 commit to branch master. Commits by camel (1).
- Aide Chantal (c501bd8)

[Gap_intervals_list] Amélie pushed 1 commit to branch master. Commits by camel (1). 11:32
- Problem elt_list and structure.elt_list (4cd3785)

[Gap_intervals_list] Amélie pushed 1 commit to branch master. Commits by camel (1). 12:16
- New interface foncteur (ca20236)

References

- Interactive tutorial:
  - `learngitbranching.js.org`
- Cheat sheet:
  - `https://education.github.com/git-cheat-sheet-education.pdf` (English version)
  - `https://training.github.com/downloads/fr/github-git-cheat-sheet.pdf` (French version)
  - `https://ndpsoftware.com/git-cheatsheet.html` (Interactive one - English, French, Chinese, Spanish, German, Korean)
- Reference book : `http://git-scm.com/book` (`https://git-scm.com/book/fr/v2/` in French)