Introduction à Git

Pierre Lefebvre

Université de Lorraine

September 30, 2025

Sommaire

À quoi ça sert ?

Système de gestion de versions de fichiers (source ou autre)

Permet de :

- gérer les fichiers lorsque qu'on travaille à plusieurs sur un projet,
- avoir un historique des modifications,
- partager les fichiers entre plusieurs personnes ou équipes,
- avoir plusieurs versions en parallèle.

Création

Créé en 2005 par Linus Torvalds.

Succède à *BitKeeper*

Utilisé pour gérer les sources du noyau Linux (et plein d'autres choses aujourd'hui)

Utilisations possibles

En **ligne de commande**

Syntaxe générale

```
git <commande> [options]
```

Exemple

```
$> git commit -m "Message"
```

Avec une **interface graphique**

Logiciels

GitHub Desktop, Git Extensions, GitKraken...

Attention

Peut être obscur au début

Les plateformes

Compatible Windows / Linux / MacOS

Conseil sous Windows

Pour la ligne de commande, utiliser Gitbash

Fonctionnement

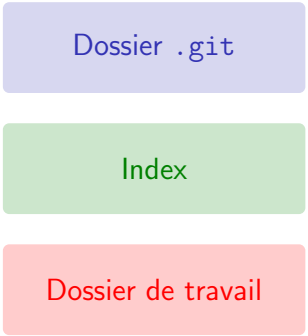

Figure: Les zones où peut se trouver un fichier

Les objets manipulés par Git

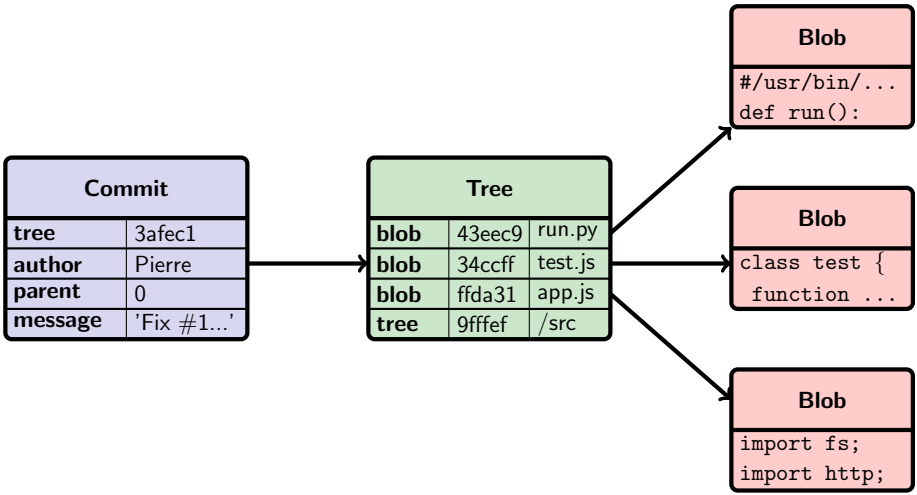

Figure: Structure interne de git

Anatomie d'un *commit*

Lorsque qu'un contenu est versionné, git crée une entrée (paire clé/valeur) permettant de retrouver ce contenu. C'est le *commit*. Un *commit* est identifié avec les informations suivantes:

- auteur du code,
- date de modification par l'auteur,
- arbre auquel il se référe,
- parent,
- message de commit.

Ces informations sont hashés (SHA-1), le *hash* servira d'identifiant pour le *commit.*

À propos des collisions...

Si l'ensemble des humains validaient une version d'un contenu équivalent au noyau linux toutes les secondes, il faudrait 4 ans pour que peut-être 2 commits aient le même hash.

L'historique

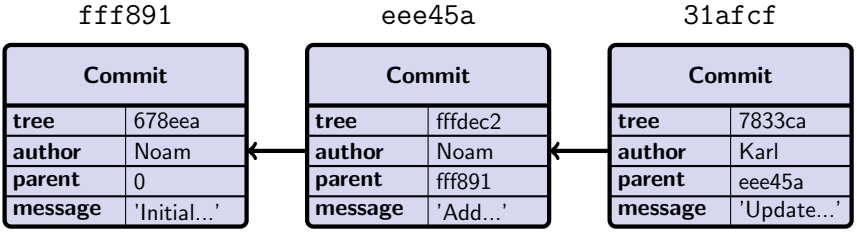

Figure: Enchaînement de *commits*

Notions de branche

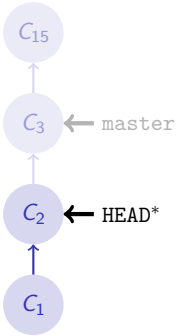

Figure: Branche

En pratique

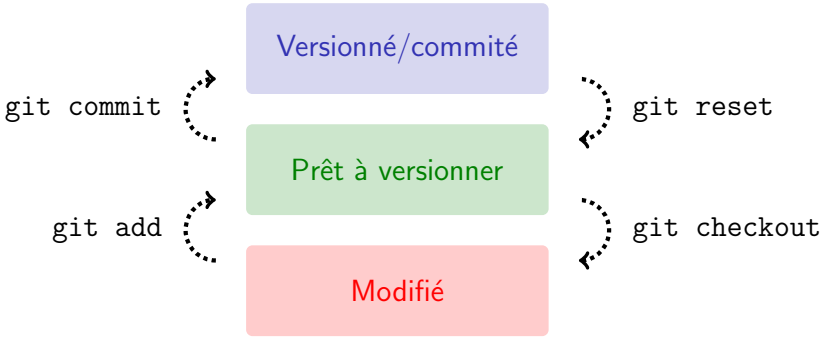

Figure: Les états d'un fichier dans un dépôt

Créer un nouveau dépôt: `git init`

Créé un dépôt en initialisant un dossier `.git`.

Syntaxe de `git init`

```
git init <target>
```

Clôner un dépôt existant: `git clone`

Télécharge un dépôt distant pour y travailler.

Syntaxe de `git clone`

```
git clone <remote>
```

Exemple : `git clone git@github.com:emacs-lsp/lsp-mode.git`

```
Clonage dans 'lsp-mode'...
remote: Enumerating objects: 108302, done.
remote: Counting objects: 100% (9409/9409), done.
remote: Compressing objects: 100% (634/634), done.
remote: Total 108302 (delta 4693), reused 9127 (delta 4455), pack-reused 98893
Réception d'objets: 100% (108302/108302), 72.91 Mio | 1.98 Mio/s, fait.
Résolution des deltas: 100% (56007/56007), fait.
```

Informations sur le dépôt: `git status`

Permet d'obtenir des informations sur:

- la branche courante et sa synchronisation avec la branche distante,
- les fichiers modifiés,
- les fichiers présents dans l'index,
- les fichiers non suivis

Exemple : `git status`

```
Sur la branche master
Votre branche est à jour avec 'origin/master'.

Modifications qui ne seront pas validées :
  (utilisez "git add/rm <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git restore <fichier>..." pour annuler les modifications dans le répertoire de travail)
  (valider ou annuler le contenu non suivi ou modifié dans les sous-modules)
 modifié : index.js

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
 test/test-read.js

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```

Voir les modifications apportées : `git diff`

Permet de voir les différences entre l'état du dépôt avant les modifications et aprés les modifications. Il peut s'utiliser avec:

- seul,
- un nom de fichier,
- un nom de répertoire,
- des expressions génériques (Shell).

Syntaxe de `git diff`

```
git diff [node]
```

Exemple : `git diff src/index.js`

```
diff --git a/src/express.js b/index.js
index f313ff3..10d7dcd 100644
--- a/src/index.js
+++ b/src/index.js
@@ -140,6 +140,7 @@ export default async (app) => {
   app.get(`${api}/client/:id/info`, crtl.info);
-  app.get(`${api}/client/:id/status`, crtl.status);
+  app.get(`${api}/client/:id/profile`, crtl.profile);
   app.get(`${api}/client/:id/login`, crtl.login);
```

Ajouter à l'index: `git add`

Permet d'ajouter un ensemble de fichiers ou dossiers à l'index. Il peut s'utiliser avec:

- un nom de fichier,
- un nom de répertoire,
- des expressions générique (Shell).

Syntaxe de `git add`

```
git add <node>
```

Exemple : syntaxe valide pour `git add`

```
git add test.py
git add ./src
git add test-*.cpp
git add .
```

Valider des modifications: `git commit`

Permet de valider un ensemble de modifications préalablement ajoutées à l'index. Un nouvel objet est créé dans l'historique.

Syntaxe de `git commit`

```
git commit [options]
```

Exemple : `git commit -m "Add unit test"`

```
[master d39b5fd] Add unit test
 1 file changed, 189 insertions(+)
 create mode 100644 test/test-nlp-prosody.js
```

Message de commit

Utiliser l'option `-m <message>` pour ajouter un message décrivant les modifications apportées par le commit

Mettre à jour la branche distante: `git push`

Met à jour une branche distante par rapport aux modifications effectuées en locales (partage les modifications).

Exemple : `git push`

```
Counting objects: 3, done.
Delta compression using to 4 threads.
Compressing obhects: 100% (2/2) done.
Writing objects: 100% (2/2), 2.3 MiB | 1.10 MiB/s, done.
Total 2 (delta 3), reused 1 (delta 1)
Everything up-to-date
```

Irréversible

Un historique pushé ne peut plus être modifiés (via `--amend` ou `rebase` par exemple. Utiliser cette commande le moins souvent possible.

Voir l'historique: `git log`

Permet de visualiser l'historique git d'un dépôt local.

Syntaxe à préférer pour `git log`

```
git log --decorate --graph --all --oneline
```

Exemple : `git log --decorate...`

```
* 8c11d07 (HEAD -> master, origin/master) Update submodules + dockerfiles
* 2e700d9 Remove UFO
*   e742631 Merge branch 'ci-cd' into 'master'
|\
| * 95cc845 CI
|/
* f897a7a Update submodule
| * 384cbd4 (origin/ci-cd) Add auto-doc
| * 3eecd54 Fix .gitlab-ci.yml
|/
* c4e05b1 Update .gitlab-ci.yml
* 65f9bea Add .gitlab-ci.yml
```

Résumé pour une utilisation basique

Aprés avoir clôné un dépôt, le schéma d'utilisation basique est le suivant:

1. effectuer des modifications sur les sources,
2. ajouter les modifications à l'index avec `git add`
3. valider les modifications avec `git commit`
4. partager les mises à jour avec `git push`

Vérifications intermédiaires

Les commandes `git status` et `git log` sont à utiliser sans modération et à chaque étape pour vérifier le bon déroulement

Autres ressources

Liens:

- Documentation officielle
- Outils interactif
- Hitler uses Git