



ÉCOLE CENTRALE LYON

Utilisation git



Auteur : Ugo Insalaco (Chest)
17 août 2020

Table des matières

1	Introduction	2
2	Fonctionnement général	2
3	Installation et mise en route	3
3.1	On veut juste utiliser git en local	3
3.2	On veut utiliser git avec un serveur	3
4	faire des modifications sur son projet et gérer les versions	5
4.1	Fonctionnement	5
4.2	Utilisation	5
4.3	Récupérer les dernières modifications du serveur	6
4.4	Envoyer les modifications au serveur	7
4.5	Gérer ses versions	7
5	Les branches	8
5.1	Création de branche en local	8
5.2	Récupérer une branche du serveur	9
5.3	Merge des branches	10
5.4	Supprimer des branches	11
6	Résumé des commandes et autres commandes utiles	11

1 Introduction

Git est un des outils les plus utilisés pour gérer le développement d'application et la gestion de versions plus généralement. Il répond à plusieurs problèmes simples lors de la création de projets : comment contribuer efficacement à plusieurs et à distance sur un même code ? Comment revenir simplement à une version antérieure de son code sans avoir à enlever chaque modification "à la main" ? ou encore comment développer son code tout en s'assurant d'avoir une version fonctionnelle à chaque instant ?

2 Fonctionnement général

Le principe de git est le suivant : le code source de l'application en cours de développement se trouve sur un serveur en ligne (généralement dans un repository github ou gitlab) et chaque contributeur peut "cloner" une version de ce code sur son ordinateur afin d'effectuer ses propres modifications. Une fois ses modifications terminées, le contributeur envoie ses modifications sur le serveur et la version du code est alors mise à jour.

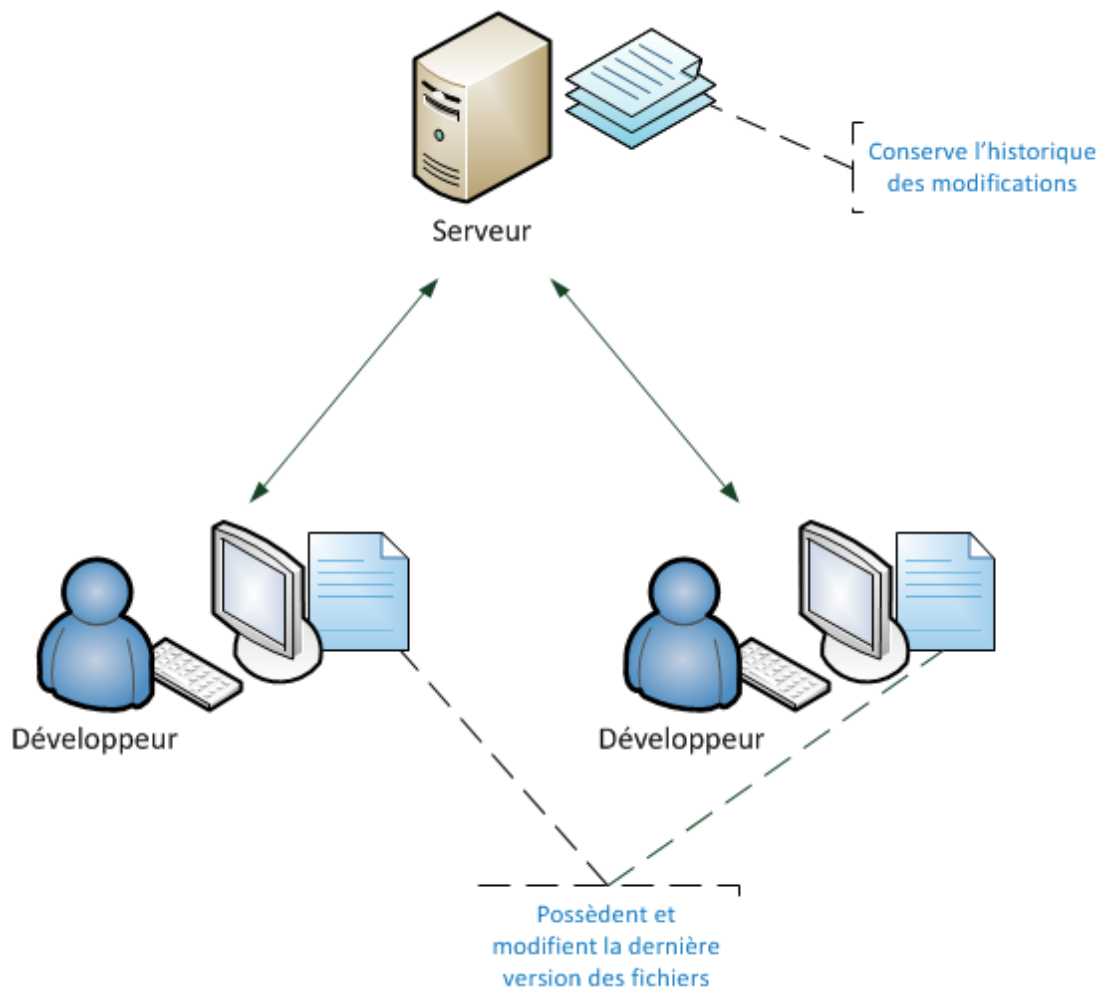


FIGURE 1 – Principe de fonctionnement de git

L'utilisation de git n'est pas nécessairement liée à celle d'un serveur. Dans le cas d'un développeur simple, l'installation de git sur son ordinateur est suffisante puisque chaque

version du code est aussi présente en local. Cependant l'utilisation d'un serveur peut toujours être utile puisqu'elle permet de faire des sauvegardes régulières de son code en ligne et ainsi de récupérer une version en cas de perte.

3 Installation et mise en route

Avant toute chose, il faut installer git (disponible [ici](#), sur linux git est installé de base). ensuite il faut lancer une invite de commande (en tapant cmd dans la barre de recherche sur windows) et configurer son nom et son email en tapant les commandes :

```
git config --global user.name <name>
```

```
git config --global user.email <email>
```

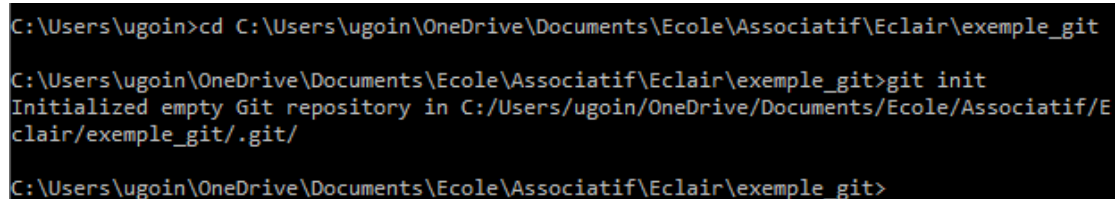
Là, on se déplace vers le dossier de notre projet que l'on veut gérer avec git puis plusieurs choix se présentent à nous.

3.1 On veut juste utiliser git en local

Dans ce cas, il suffit de lancer la commande :

```
git init
```

Cela va créer un dossier caché .git qui va contenir toutes les informations de la gestion des versions.



```
C:\Users\ugoin>cd C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_git  
  
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_git>git init  
Initialized empty Git repository in C:/Users/ugoin/OneDrive/Documents/Ecole/Associatif/Eclair/exemple_git/.git/  
  
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_git>
```

FIGURE 2 – Initialisation en local

3.2 On veut utiliser git avec un serveur

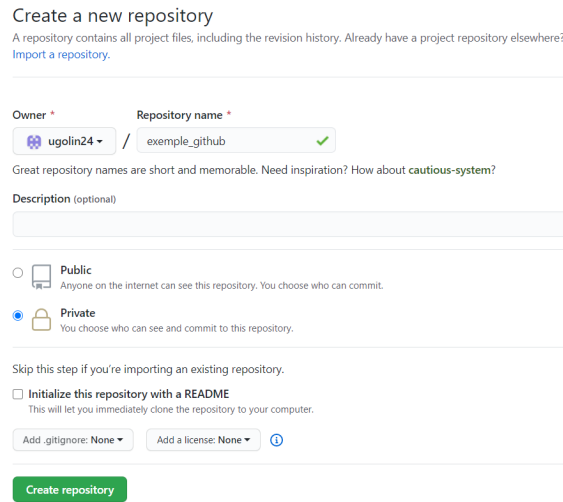
Si l'on veut utiliser un serveur (ce que l'on fait le plus souvent) on commence par créer un repository sur une plateforme en ligne. On va le faire sur github.

Pour l'exemple, on crée un fichier txt 'projet_de_fou.txt' dans le repository. Sur la page du projet on trouve un lien .git de la forme :

https://github.com/ugolin24/exemple_github.git qu'on peut copier.

De retour en local sur notre invite de commande, on lance la commande :

```
git clone https://github.com/ugolin24/exemple_github.git
```



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * Repository name *

ugolin24 / exemple_github ✓

Great repository names are short and memorable. Need inspiration? How about [cautious-system](#)?

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None Add a license: None ⓘ

Create repository

FIGURE 3 – Création de repository sur github

```
Microsoft Windows [version 10.0.18362.959]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\ugoin>cd C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair>git clone https://github.com/ugolin24/exemple_github.git
Cloning into 'exemple_github'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

FIGURE 4 – Clone de repository

Cela va récupérer le dossier présent sur le serveur et le copier ainsi que toutes les versions antérieurs sur mon ordi.

Une fois le dossier caché .git créé, l'initialisation est terminée, on peut lancer une des commandes les plus importantes de git :

git status

qui permet d'avoir toutes les informations nécessaires sur l'état actuel du projet git.

4 faire des modifications sur son projet et gérer les versions

4.1 Fonctionnement

Chaque fichier du projet peut être dans quatre états différents selon git :

- 'untracked' : le fichier n'est pas pris en charge par git. c'est le cas lorsqu'un fichier vient d'être créé.
- 'modified' : le fichier est pris en charge par git et a été modifié depuis le dernier changement de version
- 'staged' : le fichier est pris en charge par git et prêt à être enregistré
- 'commit' : le fichier est à jour depuis le dernier changement de version

Dès qu'un fichier est modifié il passe dans l'état modified. pour passer un fichier de l'état 'untracked' ou 'modified' à 'staged', on utilise la commande :

git add <filename>
ou
git add .

la première commande permettant d'ajouter les fichiers un à un et la seconde permettant de les ajouter tous d'un coup.

une fois que les fichiers sont 'staged', on applique un 'commit' qui permet d'enregistrer les modifications en local. Cela se fait étonnamment avec la commande :

git commit -m <message>

où l'on remplace le message par une description des changements appliqués

4.2 Utilisation

Dans le dossier du projet, on va d'abord créer un nouveau fichier ou faire des modifications dans l'ancien fichier et on fait un git status pour connaître l'état des fichiers :

```
Microsoft Windows [version 10.0.18362.959]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\ugoin>cd C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Projet_de_fou.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Projet_de_fou_la_suite.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

FIGURE 5 – Modification locale de fichiers

on remarque plusieurs choses importantes :

- "Your branch is up to date with 'origin/master'" : le dossier local est à jour avec la version principale en ligne (on verra le principe des branches après)

- "Changes not staged for commit ..." Les fichiers n'ont pas été sélectionnés pour être commit
 - "Untracked files ..." Les fichiers ne sont pas pris en charge par git
- on applique donc un "git add" pour passer les fichiers en staged :

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git add .

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Projet_de_fou.txt
        new file:   Projet_de_fou_la_suite.txt
```

FIGURE 6 – Passage des fichiers en staged

puis un fait un commit pour appliquer les modifications

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git commit -m
"ajout du fichier "projet_de_ouf_la_suite"
[master 11bb52c] ajout du fichier projet_de_ouf_la_suite
 2 files changed, 4 insertions(+)
 create mode 100644 Projet_de_fou_la_suite.txt

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

FIGURE 7 – Commit des modifications

On a ainsi fait une modification locale du projet et l'on a enregistré les modifications

Il est possible de dire à git que l'on ne veut pas que certains fichiers soient pris en compte lors des add et des commit. Pour cela, on crée un fichier '.gitignore' et on écrit ligne par ligne le nom de chaque fichier que l'on veut que git ignore.

4.3 Récupérer les dernières modifications du serveur

Afin de récupérer la dernière version du serveur, on peut lancer la commande

git pull

qui va récupérer toutes les modifications qui ont été faites sur le projet. l'algorithme de git va chercher à récupérer chaque nouveaux fichiers et fusionner les lignes de codes lorsque le même fichier a été modifié depuis le serveur et depuis la version locale (il faut quand même éviter de travailler à plusieurs sur le même fichier, ça évite tous les problèmes) et si l'algorithme n'arrive pas à fusionner les fichiers, il demandera ligne par ligne à l'utilisateur.

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 703 bytes | 175.00 KiB/s, done.
From https://github.com/ugolin24/exemple_github
   0e21d91..733bd3c  master    -> origin/master
Merge made by the 'recursive' strategy.
 fichier_pas_ouf.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 fichier_pas_ouf.txt
```

FIGURE 8 – Récupération du fichier 'fichier_pas_ouf.txt' depuis le serveur

4.4 Envoyer les modifications au serveur

Nos modifications ont été faites en local, mais les autres personnes n'ont pas accès à ces modifications puisqu'elles sont juste sur notre propre ordi et pas sur le serveur.

Pour envoyer les modifications au serveur il faut commencer par se mettre à jour avec la version du serveur car d'autres personnes ont pu faire des modifications eux même depuis la dernière version récupérée. on fait donc avant d'envoyer ses modifications un "git pull" comme précédemment.

Maintenant, le dossier local est à jour avec le serveur et l'on peut envoyer les changements au serveur en faisant :

git push

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git push
Enumerating objects: 10, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 675 bytes | 675.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/ugolin24/exemple_github.git
   733bd3c..63d6e0b  master -> master

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

FIGURE 9 – Envoi des modifications au serveur

On peut vérifier directement sur le repository github que le fichier ajouté en local est bien arrivé.

4.5 Gérer ses versions

on peut voir un historique des versions (ou commits) avec la commande

git log


```

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git log
commit 63d6e0b394d8e3b07d37d5e56188902b2b865eb2 (HEAD -> master, origin/master, origin/HEAD)
Merge: 11bb52c 733bd3c
Author: Ugo <ugo.insalaco@hotmail.fr>
Date: Mon Aug 3 15:01:14 2020 +0200

    Merge branch 'master' of https://github.com/ugolin24/exemple_github

commit 733bd3c126bfe4a82bb0968c70fd48a26eaea
Author: ugo lin24 <ugo.insalaco@hotmail.fr>
Date: Mon Aug 3 15:01:00 2020 +0200

    Create fichier_pas_ouf.txt

commit 11bb52c9e5c9f81b09f74cf8b0b35c5f6ccdf6ff
Author: Ugo <ugo.insalaco@hotmail.fr>
Date: Mon Aug 3 14:44:52 2020 +0200

    ajout du fichier projet_de_ouf_la_suite

commit 0e21d918c20db434531caf93c5f756401613c551
Author: ugo lin24 <ugo.insalaco@hotmail.fr>
Date: Mon Aug 3 14:02:58 2020 +0200

    Create Projet_de_fou.txt

```

FIGURE 10 – Historique des commits

chaque commit possède son propre identifiant. On peut revenir à une version antérieure en faisant

git checkout <idversion>

où l'on peut remplacer l'idversion par seulement les premiers caractères de la version. Cela va remettre les fichiers du projet comme ils étaient au moment de cette version.

5 Les branches

Imaginons que l'on ait une idée de fou pour développer notre appli mais que l'on soit pas trop sûr de comment l'implémenter, que ça prenne du temps et que si ça se trouve l'idée s'avère être nulle, là intervient le principe des branches. Le principe est de pouvoir créer une branche secondaire que l'on puisse développer en parallèle de la version originale, pour plus tard (si l'on décide de garder l'idée) fusionner la version développée avec l'origine ou alors supprimer totalement la branche en ayant quand même pu continuer à développer la branche principale (c'est là l'atout le plus puissant de git et qu'on peut retrouver dans son logo :)).

5.1 Création de branche en local

Pour créer une branche en local, rien de plus simple, on fait un

git branch <nombranche>

et on peut afficher toutes les branches avec

git branch

On voit qu'après 'git branch' la branche sélectionnée est l'originale (master). Pour se déplacer sur la nouvelle branche, la branche actuelle doit être à jour avec le serveur, puis on fait un

git checkout 'idee_de_genie'

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch idee_de_genie

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
  idee_de_genie
* master
```

FIGURE 11 – Création de la branche 'idee_de_genie'

```
C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git checkout idee_de_genie
Switched to branch 'idee_de_genie'

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
* idee_de_genie
  master
```

FIGURE 12 – Déplacement de branche en branche

la nouvelle branche sélectionnée est bien la nouvelle. Ce n'est pas un hasard si la fonction 'checkout' est aussi utilisée pour revenir à une version antérieure puisqu'en faisant cela, git va créer une branche temporaire correspondant à la version antérieure qui sera supprimée si l'on revient à la branche 'master'.

On peut maintenant faire nos modifications sur la nouvelle branche en toute tranquillité... sauf qu'il faut pouvoir les push sur le serveur, or ce dernier ne connaît pas notre nouvelle branche puisqu'elle n'existe qu'en local. Pour pallier à ce problème, on effectue lors du premier push la commande :

git push - -set-upstream origin idee_de_genie

qui va permettre de créer la branche idee_de_genie sur le serveur (on peut ici remplacer le idee_de_genie par n'importe quel nom car c'est celui qui apparaîtra sur le serveur et qui peut être différent que celui que l'on a en local). On peut regarder directement sur github et voir que la nouvelle branche y a été créé. Les prochaines fois on pourra push notre code normalement.

5.2 Récupérer une branche du serveur

En temps normal un 'git clone' va copier la branche principale présente dans le repository, ce qui n'est pas forcément ce que l'on souhaite. Ou alors, possédant déjà la branche principale, on veut récupérer une autre branche. Tout ceci peut être fait après avoir cloné le repository en faisant la commande :

git branch - -track <branchelocale> origin/<brancheserveur>

où "branchelocale est le nom qui apparaîtra en local et brancheserveur le nom de la branche que l'on souhaite copier depuis le serveur.

```

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
  idee_de_genie
* master

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch --track idee_pas_ouf origin/idee_pas_ouf
Branch 'idee_pas_ouf' set up to track remote branch 'idee_pas_ouf' from 'origin'.

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
  idee_de_genie
  idee_pas_ouf
* master

```

FIGURE 13 – copie de la branch 'idee_pas_ouf' déjà présente sur le serveur

5.3 Merge des branches

Une fois que l'on a écrit un code qui nous convient sur une branche et que l'on souhaite garder l'idée pour le projet final, la branche ne nous sert plus et l'on voudrait pouvoir la réunir avec la branche principale pour pouvoir continuer à faire d'autres branches et continuer à développer à partir du nouveau code.

Pour fusionner les deux branches, il faut commencer par se placer dans la branche où l'on veut mettre à jour le code (généralement la branche master mais cela peut aussi être une autre branche, en faisant 'git checkout master') puis on appelle la commande

git merge <brancheàmerge>

La branche donnée va venir se fusionner dans la branche actuelle (de la même façon que lorsqu'on pull des données depuis le serveur)(il faut s'assurer que les deux branches sont bien à jour avec le serveur)

Dans l'exemple suivant, on va merge la branche 'idee_de_genie' (qui possède le fichier 'code_revolutionnaire' en plus par rapport à la branche 'master') dans la branche master.

```

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
  idee_de_genie
  idee_pas_ouf
* master

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git merge idee_de_genie
Updating 63d6e0b..c540fea
Fast-forward
 code_revolutionnaire.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 code_revolutionnaire.txt

C:\Users\ugoin\OneDrive\Documents\Ecole\Associatif\Eclair\exemple_github>git branch
  idee_de_genie
  idee_pas_ouf
* master

```

FIGURE 14 – Merge de la branche 'idee_de_genie' dans la branche 'master'

5.4 Supprimer des branches

Si l'on décide d'abandonner une idée présente sur une branche ou si l'on en a fini avec une branche que l'on vient de merge, on peut vouloir supprimer des branches. Plusieurs cas se présentent selon ce qu'on veut faire :

On veut supprimer la branche du serveur :

```
git push origin :heads/<brancheàsupprimer>
```

on veut seulement supprimer la branche en local mais la garder sur le serveur alors que celle-ci a déjà été merge :

```
git branch -d <brancheàsupprimer>
```

on veut supprimer la branche en local alors qu'elle n'a pas été merge et donc perdre le travail de la branche :

```
git branch -D <brancheàsupprimer>
```

6 Résumé des commandes et autres commandes utiles

Configuration

- `git config -global user.name <name>` : configure le nom de l'utilisateur
- `git config -global user.email <email>` : configure l'email de l'utilisateur

Initialisation

- `git init` : initialise un répertoire git en local
- `git clone <urldépôt>` : clone un repository git en local
- `git status` : affiche l'état des fichiers du dossier selon git

Modification de fichiers

- `git diff <fichier>` : voit précisément les modifications faites sur un fichier
- `git add <fichier>` : ajoute le fichier à la liste des fichiers prêts à être commit
- `git add .` : passe tous les fichiers en mode staged
- `git commit -m <message>` : commit les modifications avec un message
- `git commit -a -m <message>` : commit tous les fichiers modifiés sans passer par `git add`
- `git commit --amend -m <message>` : modifie le message du dernier commit
- `git log` : affiche les derniers commits
- `git log -p` : affiche les détails des derniers commits
- `git log --stat` : affiche un résumé des derniers commits
- `git checkout <idcommit>` : permet de revenir à un commit passé
- `git checkout <fichier>` : restaure un fichier comme il était au dernier commit
- `git reset HEAD - <fichier_a_supprimer>` : retire un fichier du add
- `git pull` : récupère les commits du serveur
- `git push` : envoie les commits sur le serveur

Branches

- `git branch` : affiche toutes les branches
- `git branch <branche>` : crée une nouvelle branche en locale
- `git checkout <branche>` : permet de se déplacer sur une autre branche
- `git branch -r` : affiche les branches que le serveur connaît
- `git push -set-upstream origin <branche>` : copie la branche locale sur le serveur avec le nom défini par le paramètre
- `git push origin origin :refs/heads/<branche>` : crée une branche sur le serveur
- `git branch -track <branchelocale> origin/<brancheserveur>` : crée une branche local avec le nom `branchelocale` et copie la branche `brancheserveur` du serveur dans la branche locale
- `git push origin :heads/<brancheàsupprimer>` : supprimer une branche sur le serveur
- `git merge <branche>` : merge la branche dans la branche actuelle
- `git branch -d <branch>` : supprime une branche locale qui a été merge
- `git branch -D <branch>` : supprime une branche locale qui n'a pas été merge