

# Git/Github documentation

Adrien JUPITER & Soukaina EL GHALDY & Siva CHANEMOUGAM

14 novembre 2020

## Contents

<b>Git ou Github ?</b>	<b>2</b>
<b>Installation de git</b>	<b>2</b>
Linux . . . . .	2
MacOS . . . . .	2
Windows . . . . .	2
<b>Comprendre le principe de Git</b>	<b>3</b>
Les dépôts git . . . . .	3
Les commits . . . . .	3
Les Branches . . . . .	4
La fusion de branches . . . . .	5
<b>Les bases de git</b>	<b>8</b>
Démarrer un dépôt Git . . . . .	8
Enregistrer des modifications dans le dépôt . . . . .	8
Placer de nouveaux fichiers sous suivi de version . . . . .	9
Valider vos modifications . . . . .	9
Effacer des fichiers . . . . .	9
Déplacer des fichiers . . . . .	10
Visualiser l'historique des validations . . . . .	10
Envoyer vos modifications . . . . .	10
<b>GitHub : Comment ça marche ?</b>	<b>11</b>
La Création d'un dépôt distant . . . . .	11
La Création d'une branche . . . . .	11
La Validation des Commits . . . . .	13
L'Ouverture d'une Pull request . . . . .	15
La Fusion de la Pull request . . . . .	16
<b>Conclusion</b>	<b>17</b>
<b>Bibliographie</b>	<b>17</b>

## Git ou Github ?

Git est un **SCV** “Système de contrôle de version”, ce qui implique : création de fichier, modification du fichier, sauvegarde. Le système s’appuie sur la gestion de version **décentralisé**. Il permet donc à chaque développeur d’avoir en local sa copie du projet ainsi que son propre logiciel de gestion de version.

GitHub facilite la collaboration en utilisant git. C’est une plateforme qui peut contenir des dépôts de code dans un stockage dans le cloud afin que plusieurs développeurs puissent travailler sur un même projet et voir les modifications des autres en temps réel.

## Installation de git

### Linux

Un exemple d’installation sur Ubuntu qui est une distribution de Linux orienté vers le grand public et basé sur Debian.

Ouvrir un shell Unix et executer l’une des commandes suivantes:

- apt install git (en **root**)
- sudo apt install git (avec **sudo**)

Git est maintenant installé sur votre système, **BRAVISSIMO** !

### MacOS

Il existe plusieurs manière d’installer git sur MacOS, il est possible de l’installer par Homebrew en saisissant la commande suivante dans le shell, **brew install git** ou bien en installant Xcode qui integre par default git. Ou bien taper simplement **git --version** dans un terminal shell, et le système verifera si git est installé, dans le cas contraire le systeme vous proposera de l’installer.

### Windows

Si par malheur vous n’avez ni Linux ou ni MacOS, il existe des solutions alternatives tel que Windows.

Pour installer git sur Windows, rendez-vous à la page suivante <https://git-scm.com/download/win> et choisir la version adapté à votre configuration système.

Ouvrir l’exécutable, laisser les options par défaut et lancez l’installation.

Ecrivez ensuite “select default shell” et selectionner git bash.

*Une fois installé sur l’un des trois système, il est nécessaire de lancer ces deux lignes de commandes afin d’apporter une configuration minmale à git:*

- **git config --global user.name “votre\_username”**
- **git config --global user.email “votre\_mail”**

# Comprendre le principe de Git

## Les dépôts git

Pour pouvoir utiliser Git, il faut dans un premier temps créer un répertoire Git (un dépôt Git). Il est possible de créer un dépôt vide ou alors de convertir un projet existant, sans version en un dépôt Git. Concernant la conversion d'un projet existant en dépôt Git, il vous suffira de vous placer dans le répertoire du projet : `Cd /chemin/projet` et exécuter la commande depuis cet endroit. Dans un nouveau projet, `git init` sera la première commande que vous exécuterez car une grande majorité des autres commandes ne sont pas disponibles en dehors d'un dépôt git initialiser. En exécutant `git init`, nous créons un sous-répertoire, `.git` qui contiendra toutes les métadonnées qui sont nécessaires pour le dépôt. Vous pourrez ainsi créer vos différents fichiers de travail, les modifier et les enregistrer à travers des « snapshot », plus communément appelé *commit*.

## Les commits

L'élément essentiel à comprendre est que Git fait des instantanés (snapshot) d'un projet afin de les versionner. A chaque snapshot nous avons une nouvelle version du projet qui est enregistré. Il existe sur Git trois zones bien distinctes, où notre projet va évoluer :

- La zone de travail: zone où nous allons travailler sur nos fichiers
- La zone de transit: Zone dans laquelle nous allons placer les fichiers à versionner (ex : mettre des objets dans un carton qui sera prêt à être envoyé au dépôt).
- Le dépôt : Zone où les fameux « snapshot » sont versionnés et stockés. Une référence à cette version sera également créée afin de pouvoir la consulter.

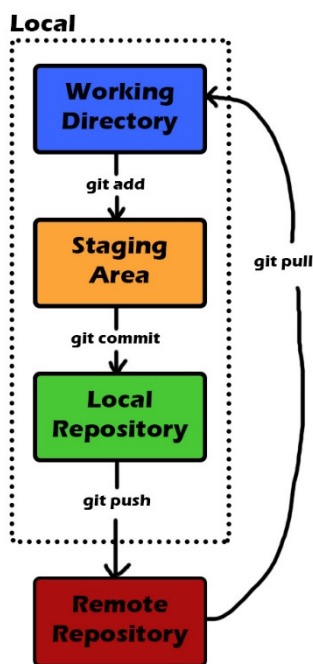


Figure 1: Mécanisme de Git

Chaque snapshot représente une version de votre projet. Sur Git ceci est appelée *un commit*. A chaque fois qu'un fichier passe par les trois zones, il y a un nouveau commit. Les commits sont stockés par ordre chronologique.

Exemple:

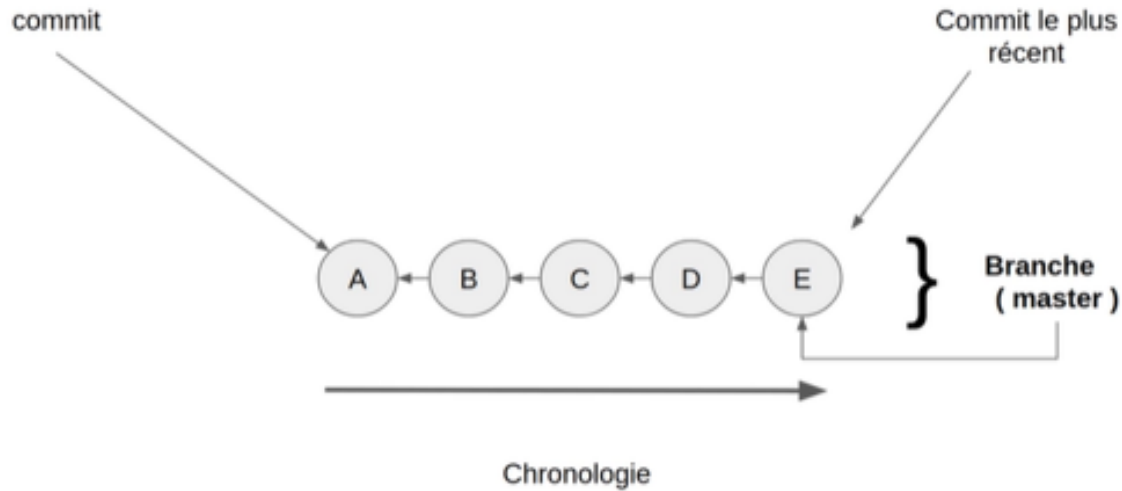


Figure 2: Le Commit

Le commit E est la version la plus récente du projet. L'ensemble de ces commits forment une branche.

## Les Branches

Vous pouvez voir une branche comme une « timeline » de votre projet contenant les différentes versions de votre projet et qui vous sont accessibles. Par défaut, la branche « Master » est créée par Git et l'ensemble de votre projet évolue sur cette branche. Très souvent pour des raisons pratiques, logistiques, nous devons créer plusieurs branches. Par exemple : Nous avons développé une application, nous en sommes à la 5<sup>ème</sup> version. Nous souhaitons développer une nouvelle « feature » pour l'appli, mais nous ne souhaitons pas impacter le fonctionnement de l'application actuelle. Nous allons donc créer une branche secondaire sur laquelle nous développerons cette « feature ». Une fois la « feature » prête nous pourrions l'intégrer à la branche principale. Très souvent les développeurs utilisent des branches secondaires pour réaliser des tests ou développer de nouvelles fonctionnalités.

Exemple de projets à deux branches:

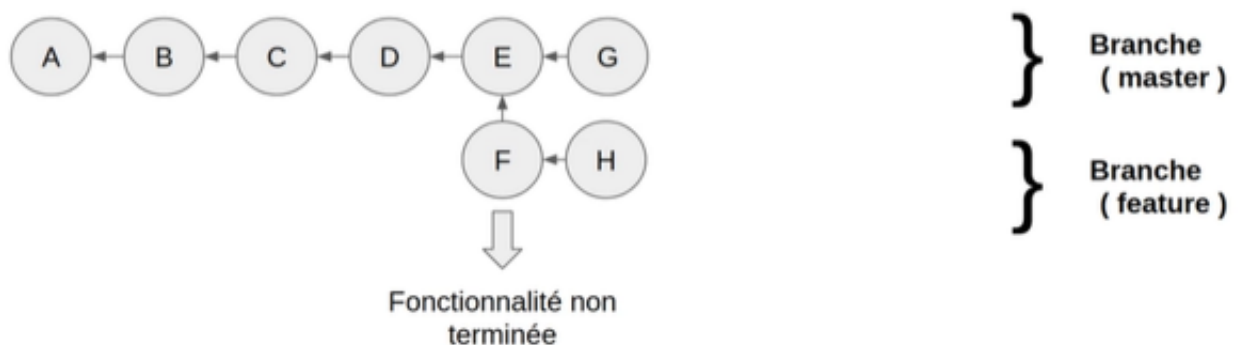


Figure 3: Système de branches

Il est très facile de naviguer entre les branches. Il existe un élément appelé « HEAD » qui pointe par défaut sur la branche Master. Mais vous pouvez le manipuler très facilement et le faire pointer sur la branche de votre choix.

## La fusion de branches

Comme vous l'avez compris, il est possible de fusionner des branches. C'est la raison pour laquelle dans un projet git, les « features » et tests sont réalisés dans des branches parallèles (secondaire). Une fois les tests réalisés ou « features » terminé, nous pouvons les fusionner avec notre branche principale afin d'y ajouter les fonctionnalités des branches secondaires. Il existe deux types de fusion : Merge fast forward : La fusion la plus simple, elle s'applique dans le cas où il n'y a pas eu de nouvelles versions (commit) dans la branche principale (master), après la création de la branche secondaire.

Exemple:

- Branche principale: Master
- Branche secondaire: foo

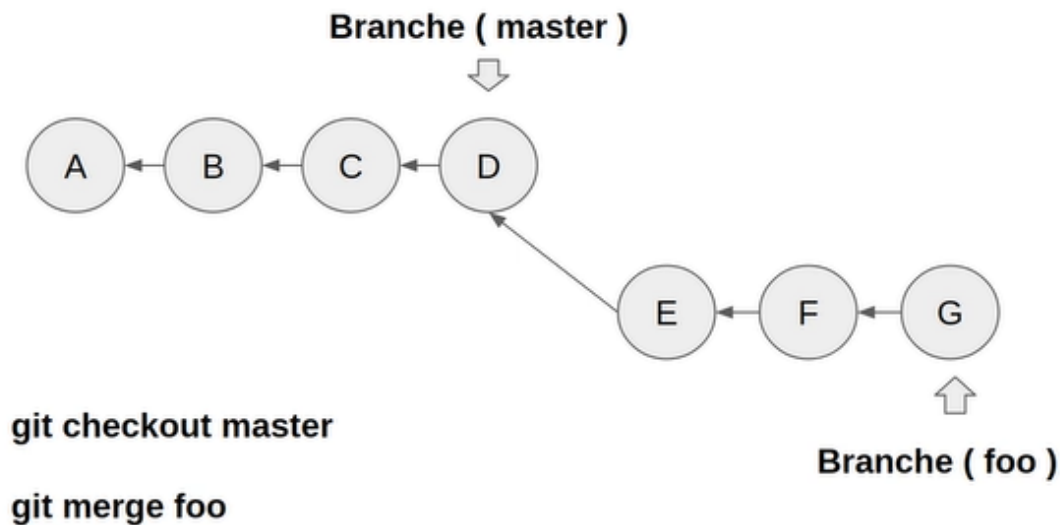


Figure 4: Merge fast forward

Dans cet exemple, une branche secondaire a été créée au niveau du commit D et on remarque qu'il n'y a pas eu de nouvelle version après la D sur la branche master. Les commits suivants (E,F,G) sont sur la branche secondaire (Foo). A la suite de la fusion, les commits E, F et G viendront s'ajouter sur la branche Master à la suite du commit D. La branche Foo sera alors supprimée car il n'est plus utile.

Merge three way: Un peu plus complexe, cette fusion s'effectue lorsqu'il y a déjà eu des commits réalisés sur la branche principale après la création de la branche secondaire. Dans ce cas, Git va prendre en considération trois éléments: le dernier commit sur la branche principale (Master), le dernier commit sur la branche secondaire (Foo) et le 1er ancêtre commun (ici F). A partir de là Git va créer un commit de merge.

## Merge three-way

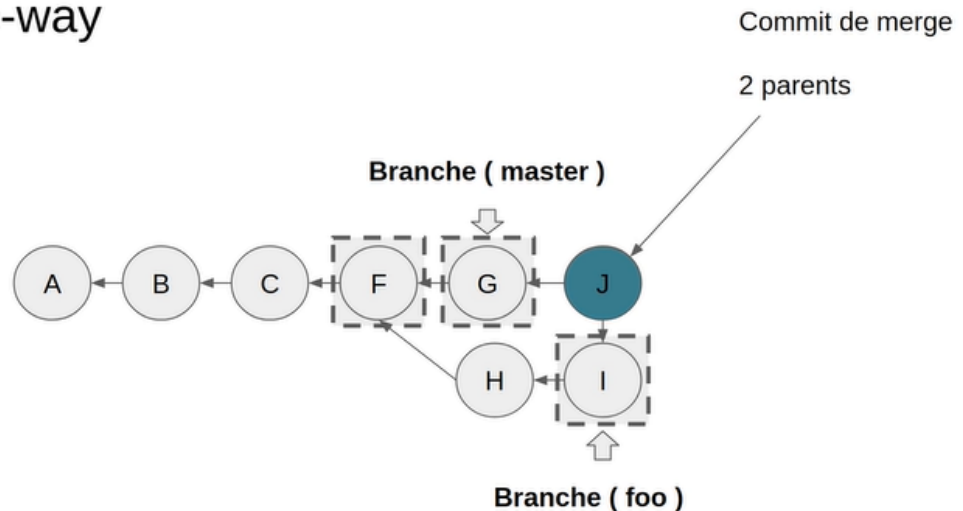


Figure 5: Merge three way

Il peut y avoir des conflits surtout lorsqu'un même fichier a été modifié au même endroit (ex: même lignes) dans les versions «G» et «I». Il faudra alors indiquer à Git quelle modification il devra privilégier lors de la fusion.

Grâce à ce système de commit, de branche et de fusion, voici à quoi peut ressembler un projet git un peu plus complexe:

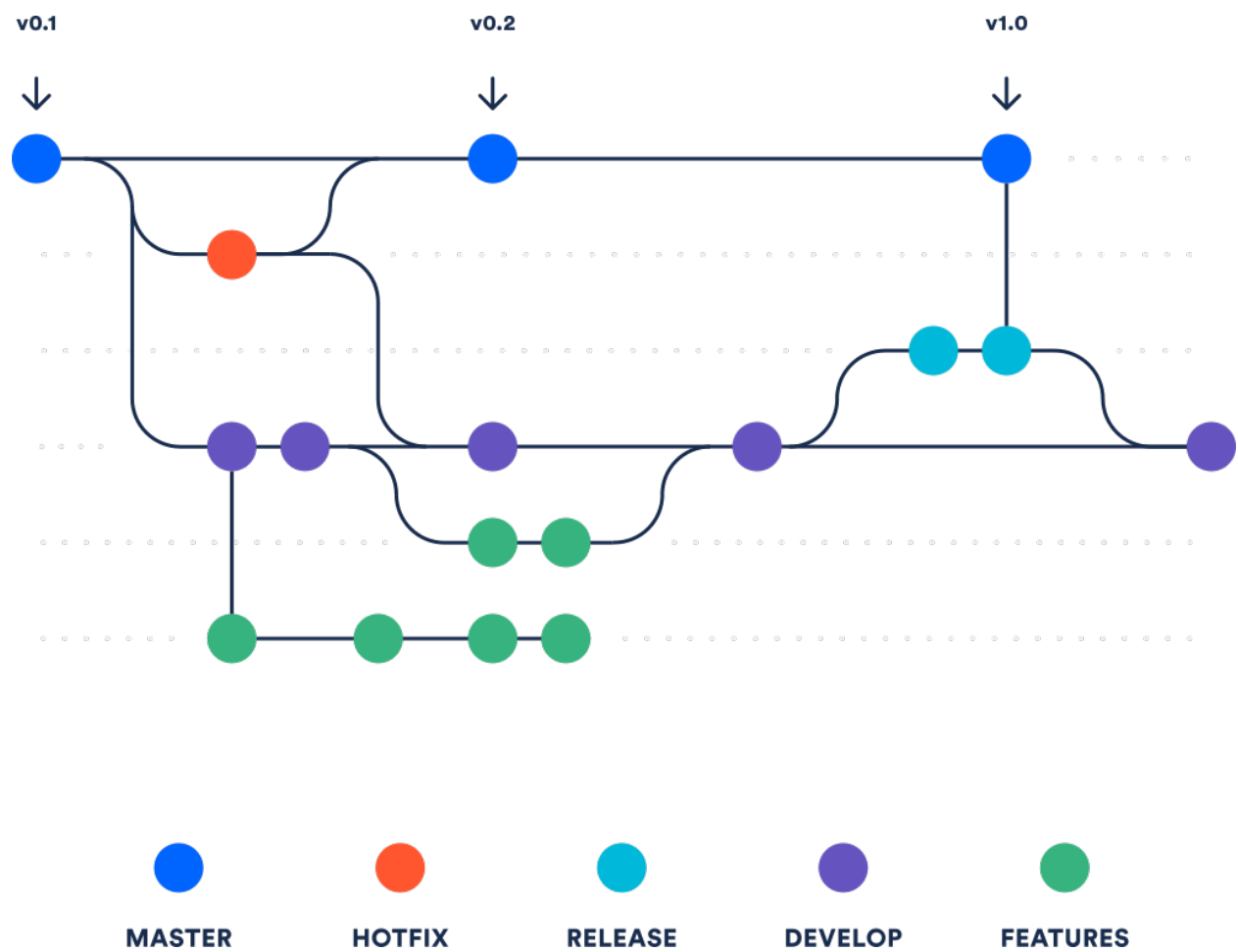


Figure 6: système de commit

## Les bases de git

Dans cette partie nous n'expliqueront pas toutes les fonctionnalités de git mais uniquement les plus utiles pour bien commencer.

### Démarrer un dépôt Git

#### Initialisation d'un dépôt Git dans un répertoire existant

La commande ***git init*** permet de créer un nouveau sous-répertoire nommé `.git` qui contient tous les fichiers nécessaires au dépôt.

#### Cloner un dépôt existant

Le clone d'un dépôt git distant se fait avec la commande ***git clone***.

Il existe deux manières courantes de cloner un dépôt git:

- soit par https
- soit par ssh

Par exemple pour cloner un répertoire git en utilisant https, on tape ***git clone [url\_https]*** dans votre répertoire de travail.

### Enregistrer des modifications dans le dépôt

#### Vérifier l'état des fichiers

L'outil principal pour déterminer quels fichiers sont dans quel état est la commande ***git status***. Si vous lancez cette commande juste après un clonage, vous devriez voir ce qui suit :

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
rien à valider, la copie de travail est propre
```

Figure 7: git status

Ce message signifie que votre copie de travail est propre, en d'autres termes, aucun fichier suivi n'a été modifié.



Si vous ajoutez un nouveau fichier se nommant **LISEZMOI** dans le dépôt git et que vous lancer un git status, alors vous verrez votre fichier non suivi comme ci-dessous:

```
$ echo 'Mon Projet' > LISEZMOI
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

LISEZMOI

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents
(utilisez "git add" pour les suivre)
```

Figure 8: git status

## Placer de nouveaux fichiers sous suivi de version

Pour commencer à suivre un nouveau fichier, vous utilisez la commande git add. Pour commencer à suivre le fichier LISEZMOI, on entre ceci:

- ***git add LISEZMOI***

Si vous lancez à nouveau la commande git status, vous pouvez constater que votre fichier LISEZMOI est maintenant suivi et indexé :

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)

nouveau fichier : LISEZMOI
```

Figure 9: git add

## Valider vos modifications

Pour valider une modification, on utilise la commande ***git commit***.

## Effacer des fichiers

Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version puis valider. La commande ***git rm*** réalise cette action mais efface aussi ce fichier de votre copie de travail.

## Déplacer des fichiers

Si vous souhaitez renommer/déplacer un fichier dans Git, vous pouvez lancer la commande suivante:

- *git mv nom\_origine nom\_cible*

## Visualiser l'historique des validations

Après avoir créé plusieurs commits ou si vous avez cloné un dépôt ayant un historique de commits, vous souhaitez probablement revoir le fil des événements. Pour ce faire, la commande `git log` est l'outil le plus basique et le plus puissant. Les exemples qui suivent utilisent un projet très simple nommé `simple git` utilisé pour les démonstrations. Pour récupérer le projet, lancez :

- *git clone https://github.com/schacon/simplegit-progit*

Lorsque vous lancez `git log` dans le répertoire de ce projet, vous devriez obtenir un résultat qui ressemble à ceci :

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

Figure 10: git log

## Envoyer vos modifications

Afin d'envoyer vos modifications vers le dépôt git, il faut procéder en trois étapes en lançant par ordre les commandes suivantes:

- 1) *git add "vos\_fichiers"*
- 2) *git commit -m "mettre un nom pour le commit ici"*
- 3) *git push*

## GitHub : Comment ça marche ?

GitHub est une plateforme d'hébergement de code pour le contrôle de version et la collaboration. Il vous permet, à vous et aux autres, de travailler ensemble sur des projets où que vous soyez. Cette partie est un guide qui vous apprendra les bases de GitHub, telles que les dépôts, les branches, les commits et les pull requests.

Pour compléter ce tutoriel, vous aurez uniquement besoin d'un compte sur GitHub.com et d'un accès Internet.

### La Création d'un dépôt distant

Un dépôt ou un "Repository" en Anglais est un référentiel, habituellement utilisé pour organiser un seul projet, qui peut contenir des dossiers, des fichiers, des images, des vidéos, des feuilles de calcul et des ensembles de données – tout ce dont votre projet a besoin.

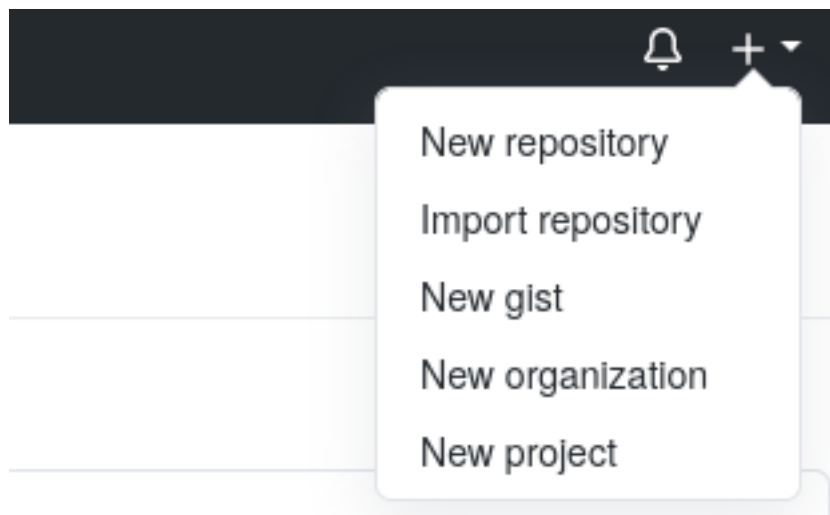
Il est fortement conseillé d'ajouter un fichier README ou un fichier contenant des informations sur votre projet lors de la création de votre nouvel "Repository". GitHub vous permet d'ajouter un README automatiquement et offre également d'autres options courantes comme l'ajout d'un fichier pour une licence.

Votre dépôt peut être un endroit où vous stockez des idées, des ressources, ou même partager et discuter des choses avec d'autres.

#### Créer un nouveau dépôt

Dans le coin supérieur droit, à côté de votre avatar ou identicon, cliquez sur + et sélectionnez New repository.

- Nommez votre nouveau dépôt **Test**.
- Écrivez une courte description de son contenu.
- Sélectionnez Initialiser ce dépôt avec un README.
- Cliquez ensuite sur Create repository.



*Pour l'instant, votre nouveau repository ne contient qu'un seul et unique fichier qui est le **README**. Ce dernier est automatiquement en format markdown.*

### La Création d'une branche

Le système de branche permet de travailler sur différentes versions d'un dépôt. Par défaut, votre dépôt a une branche nommée **main** qui est considérée comme la branche principale ("*master*"). Nous utilisons les branches pour expérimenter et faire des modifications avant de les enregistrer ("*Commit*") et les fusionner ("*Merge*") à la branche principale.

Create a new repository

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

Owner <sup>\*</sup>  / Repository name <sup>\*</sup>

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

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.

Initialize this repository with:

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

☒ Add a README file  
This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore  
Choose which files not to track from a list of templates. [Learn more.](#)

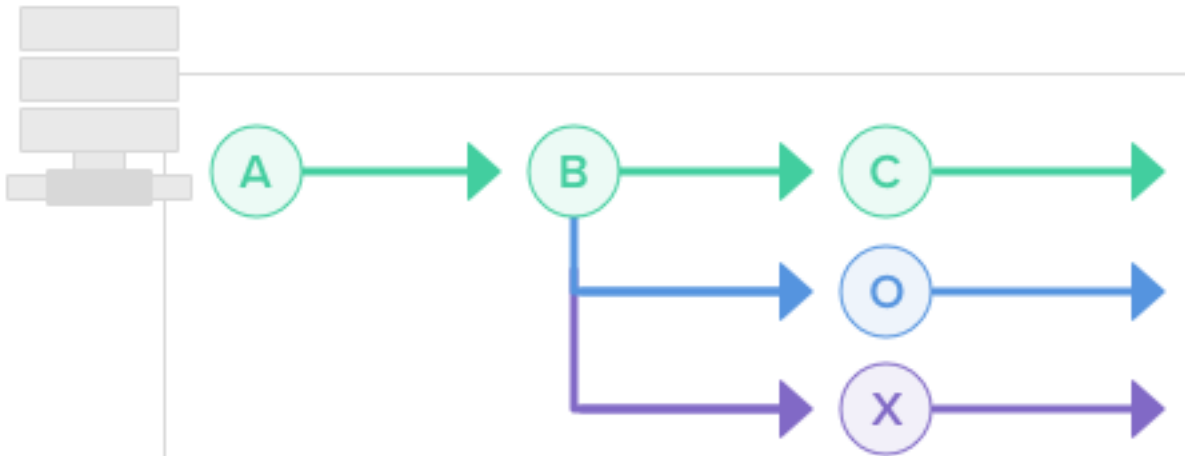
☐ Choose a license  
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

Create repository

Figure 11: Votre repository “Test”

Lorsque vous créez une branche à partir de la branche principale, vous faites une copie, ou un instantané, de “*main*” comme elle l’était à ce moment-là. Si quelqu’un d’autre a apporté des changements à la branche principale pendant que vous travailliez sur votre branche, vous pouvez alors “pull” ces mises à jour à tout moment.



Ce diagramme montre :

- La branche principale *main* en vert.
- Une seconde branche en bleu qui est une instantanée à partir de la version B de la branche principale.
- Une troisième branche en violet qui est une instantanée à partir de la version B de la branche principale.
- L’existence d’une seconde et d’une troisième branche montre qu’il y’a deux changements simultanés de la version B. Ceci peut-être dû, par exemple, à l’existence de 2 collaborateurs.

Avez-vous déjà enregistré différentes versions d’un fichier ? Les noms de fichiers ci-dessous vous rappelle-t-il quelque chose ?

```
monRappportdeStage.txt
monRappportdeStage_versionEdité.txt
monRappportdeStage_versionOfficiel.txt
```

Les branches dans les dépôts GitHub atteignent des objectifs similaires.

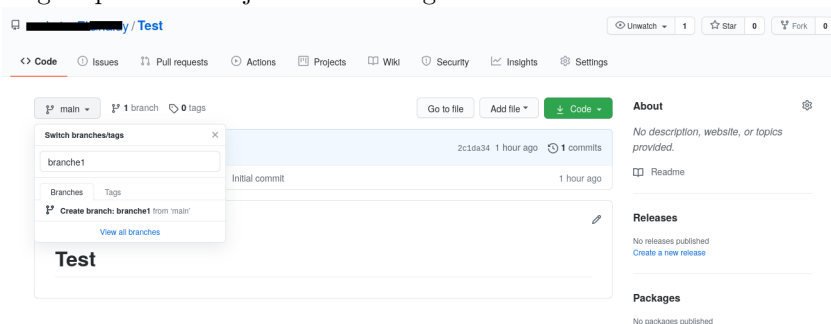
Sur GitHub, les développeurs, rédacteurs et designers utilisent des branches pour garder les corrections de bugs et le travail de fonctionnalité séparés de la branche principale. Quand un changement est prêt, ils fusionnent leur branche avec **main**.

## Créer une nouvelle branche

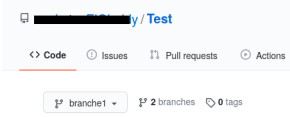
- Allez à votre nouveau dépôt Test.
- Cliquez sur le menu déroulant en haut de la liste des fichiers qui indique branche : **main**.
- Saisissez un nom de branche, “**branche1**”, dans la nouvelle zone de texte de branche.
- Sélectionnez la boîte bleue Create branch ou appuyez sur Entrée sur votre clavier.

*Et voilà! vous venez de créer votre première branche !!*

Maintenant, vous avez deux branches, **main** et “**branche1**”. Ils se ressemblent exactement, mais pas pour longtemps ! Allons ajouter nos changements à la nouvelle branche.



Vous pouvez remarquer que vous avez 2 branches.



Et que vous vous situez dans la branche1.

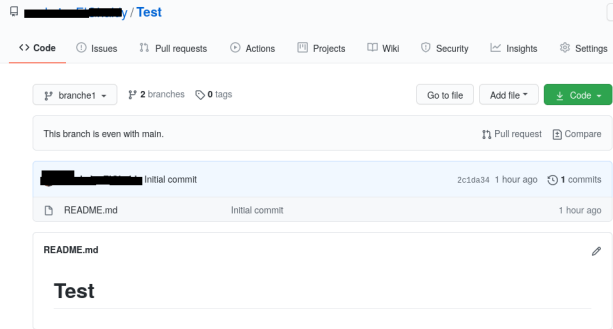
## La Validation des Commits

Dans cette partie, nous apprendrons ensemble à valider des enregistrements ou ce qu'on appelle un “Commit”. Faire un “Commit” est le fait de valider une version modifiée de votre travail et permet d'enregistrer vos modifications dans un instant T (souvent pour ne pas perdre son code si nous souhaitons tester de nouvelles combines par exemple).

### Faites votre premier Commit

Sur GitHub, les modifications enregistrées sont appelées commits. Chaque commit a un message de validation associé, qui est une description expliquant pourquoi un changement particulier a été effectué. Les messages de validation saisissent l'historique de vos modifications afin que les autres contributeurs puissent comprendre ce que vous avez fait et pourquoi.

- Cliquez sur le fichier README.md.



- Cliquez sur l'icône du crayon dans le coin droit du fichier à modifier.

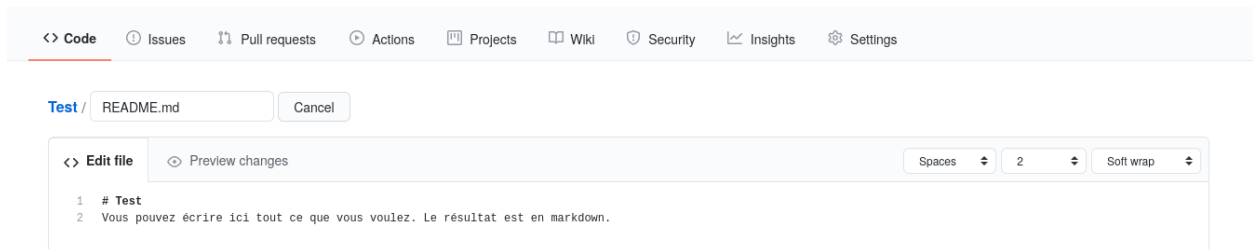
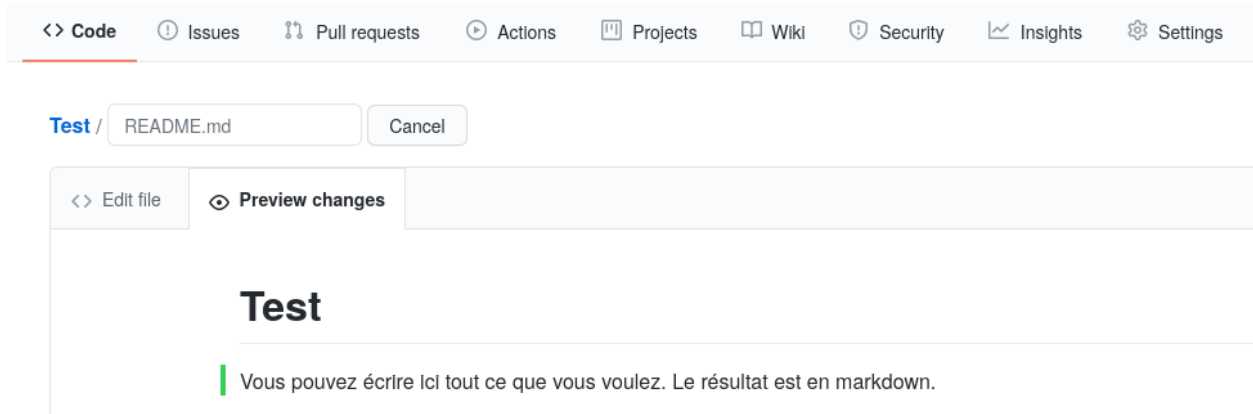
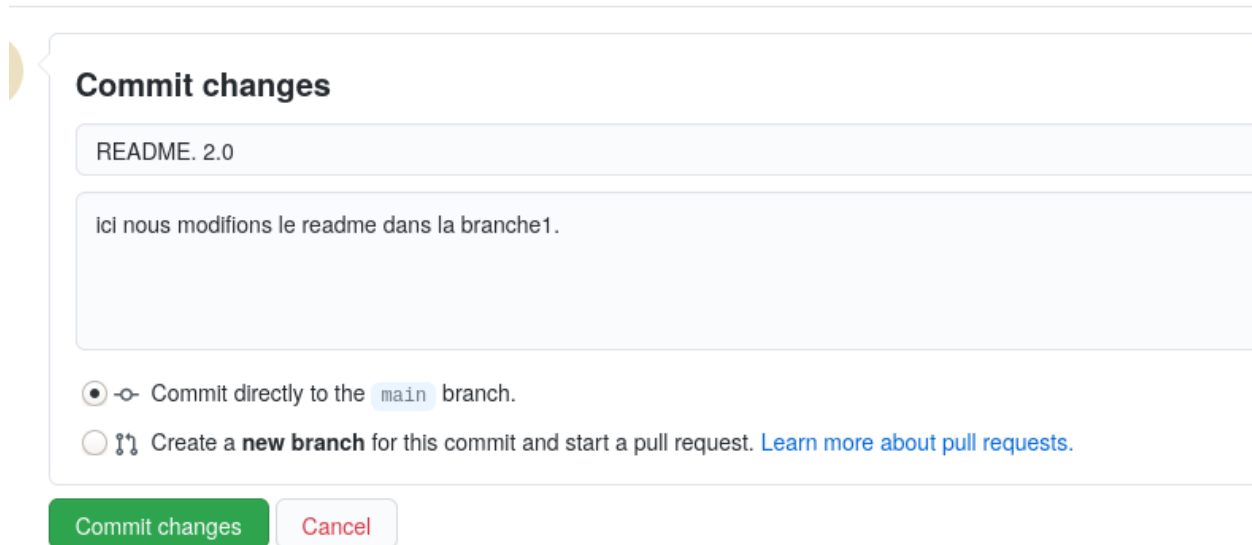


Figure 12: Modifiez le README

- Dans l'éditeur, écrivez quelque chose.
- Ecrivez un message de validation décrivant vos modifications.





- Cliquez sur le bouton Commit changes en bas de page.

Ces modifications seront apportées au fichier README de votre branche **branche1**, donc maintenant cette branche contient du contenu différent de main.

## L'Ouverture d'une Pull request

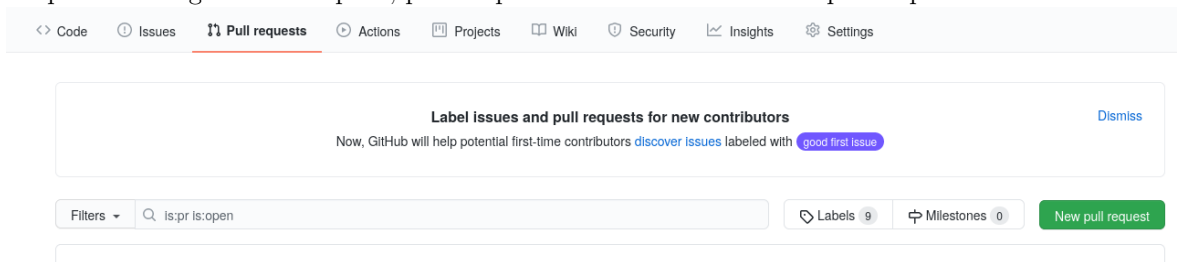
Maintenant que vous avez des changements dans une branche autre que main, vous pouvez ouvrir une demande d'extraction ou une "Pull request" en anglais.

Les "Pull Requests" sont au cœur du processus de collaboration sur GitHub. Lorsque vous ouvrez une demande d'extraction, vous proposez vos changements et demandez à quelqu'un d'examiner votre contribution et de la fusionner avec sa branche. Les Pull requests montrent les différences entre le contenu des deux branches. Les changements, ajouts et soustractions sont affichés en vert et en rouge.

Dès que vous faites un commit, vous pouvez ouvrir "une Pull request" et lancer une discussion, avant même que le code ne soit terminé. En utilisant le système @mention de GitHub dans votre message de demande d'extraction, vous pouvez demander des commentaires à des personnes ou à des équipes particulières, qu'elles se trouvent au bout du couloir ou à 10 fuseaux horaires. Vous pouvez même ouvrir des requêtes d'extraction dans votre propre dépôt et les fusionner vous-même. C'est une excellente façon d'apprendre le flux GitHub avant de travailler sur des projets plus importants.

## Faites votre première Pull request


- Cliquez sur l'onglet Pull Request, puis cliquez sur le bouton vert New pull request.



- Dans la boîte Example Comparisons, sélectionnez la branche que vous avez créée, **branche1**, pour la comparer avec **main**.

## Compare changes

Compare changes across branches, commits, tags, and more below. If you need to, you can also [compare across forks](#).


 base: main < compare: main

Choose different branches or forks above to discuss and review changes. [Learn about pull requests](#) [Create pull request](#)



### Compare and review just about anything

Branches, tags, commit ranges, and time ranges. In the same repository and across forks.

Example comparisons	
 <a href="#">branche1</a>	4 hours ago
 <a href="#">main@{1day}...main</a>	24 hours ago

- Examinez vos changements dans les différences sur la page de comparaison, assurez-vous qu'ils sont ce que vous voulez soumettre. Un code couleur est établi pour vous aider à faire la différence entre la README de la branche principale et la README de la branche1.
- Lorsque vous êtes convaincu que ce sont les changements que vous voulez soumettre, cliquez sur le grand bouton vert "Create pull request".
- Donnez un titre à votre demande d'extraction et écrivez une brève description de vos changements.


*Astuce : Vous pouvez utiliser l'emoji (pour le pull) et glisser et déposer des images et gifs sur les commentaires et les demandes de pull.*


## La Fusion de la Pull request


Afin de soumettre cet enregistrement à la branche principale et le valider autant que version officiel de notre rendu final, nous allons devoir le fusionner avec la branche principale. Dans cette dernière étape, il est temps de regrouper vos changements – en fusionnant votre branche **branche1** avec la branche principale.


### Faites votre premier "Merge"

- Cliquez sur le bouton vert Merge pull request pour fusionner les modifications avec la branche principale.
- Cliquez sur Confirm merge afin de confirmer la fusion.
- Allez-y et supprimez la branche, puisque ses modifications ont été intégrées, avec le bouton Supprimer la branche dans la boîte mauve.




 **This branch has no conflicts with the base branch**  
Merging can be performed automatically.

 [Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



**Pull request successfully merged and closed**  
You're all set—the `readme-edits` branch can be safely deleted.

 [Delete branch](#)

*Et voilà ! Le tour est joué.*



## Conclusion

Afin d'améliorer l'efficacité et la productivité, le travail d'équipe doit utiliser des outils qui favorisent la communication et le partage d'idées. De plus, étant donné que les membres d'une équipe peuvent ne pas toujours travailler sur le projet d'équipe en même temps, ou même que leur progression de travail peut ne pas être nécessairement la même, grâce à l'outil git ils peuvent partager et consulter des idées à tout moment. En fait, il permet à chaque membre de l'équipe de participer et de commenter le travail effectué par ses coéquipiers. En résumé, Git est idéal pour gérer votre projet et éviter les erreurs qui peuvent coûter très cher. Il faut que cela devienne un automatisme dans votre manière de travailler. Vous y gagnerez à la fois en temps et en qualité.

## Bibliographie

- <https://git-scm.com/downloads>
- <https://git-scm.com/doc>
- <https://git-scm.com/book/en/v2>
- <https://guides.github.com/activities/hello-world/>
- <https://openclassrooms.com/fr/courses/5641721-utilisez-git-et-github-pour-vos-projets-de-developpement/6112481-corrigez-vos-erreurs-sur-votre-depot-local>
- <http://larmarange.github.io/analyse-R/rmarkdown-les-rapports-automatises.html>
- [https://git-scm.com/docs/git#\\_git\\_commands](https://git-scm.com/docs/git#_git_commands)