



# git

## Qu'est-ce que GIT ?

Git est un outil de gestion de projets et de versions collaboratif.

**Git** est un [logiciel de gestion de versions décentralisé](#). C'est un [logiciel libre](#) créé par [Linus Torvalds](#), auteur du [noyau Linux](#), et distribué selon les termes de la [licence publique générale GNU](#) version 2. En 2016, il s'agit du [logiciel de gestion de versions](#) le plus populaire qui est utilisé par plus de douze millions de personnes.

**GIT download :**

<https://git-scm.com/downloads>

**Installation GIT :**

Après l'installation

1 Lancer Git Bash

2 Enregistrer le login :

```
git config --global user.name "nom utilisateur"
```

3 Enregistrer son adresse mail :

```
git config --global user.email mail utilisateur
```

4 Pour vérifier que sa configuration est correcte :

```
git config --list
```

5 Activer la coloration :

```
$ git config --global color.diff auto  
$ git config --global color.status auto  
$ git config --global color.branch auto
```

6 Configurer son éditeur de texte par défaut et Vimdiff comme outil de merge :

```
$ git config --global core.editor notepad++      pour Notepad++  
$ git config --global core.editor code           pour Visual studio code  
$ git config --global merge.tool vimdiff         outils de merge
```

7 Accéder à votre répertoire (dépôt) :

Taper cd puis l'url de votre répertoire par ex :

```
$ cd C:/Users/nom_utilisateur/Desktop/git
```

8 Initialiser le répertoire :

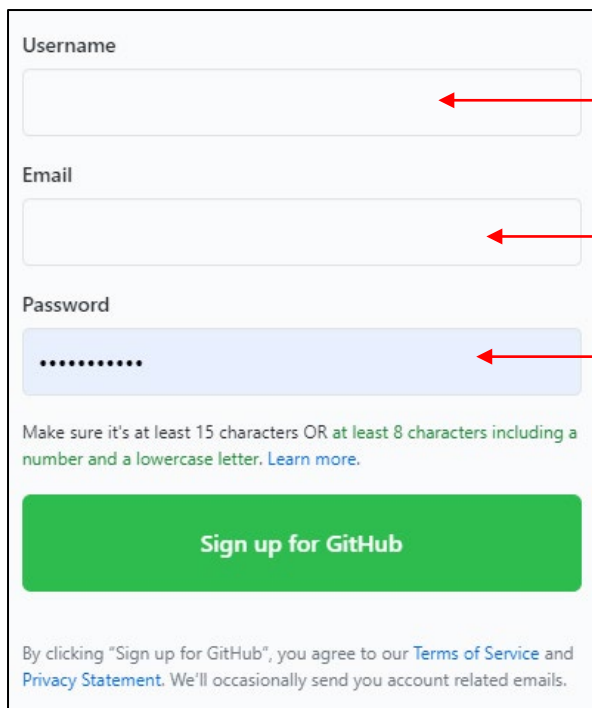
```
$ git init
```

## Créer un compte GitHub :

1 Se connecter sur github.com :

<https://github.com>

2 Cliquer sur Sign up en haut à droite :



The image shows the GitHub sign-up form. It has three input fields: 'Username', 'Email', and 'Password'. The 'Password' field is highlighted in blue and contains a series of dots. Below the fields is a green button labeled 'Sign up for GitHub'. At the bottom, there is a line of text: 'By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.'

Username

Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

Sign up for GitHub

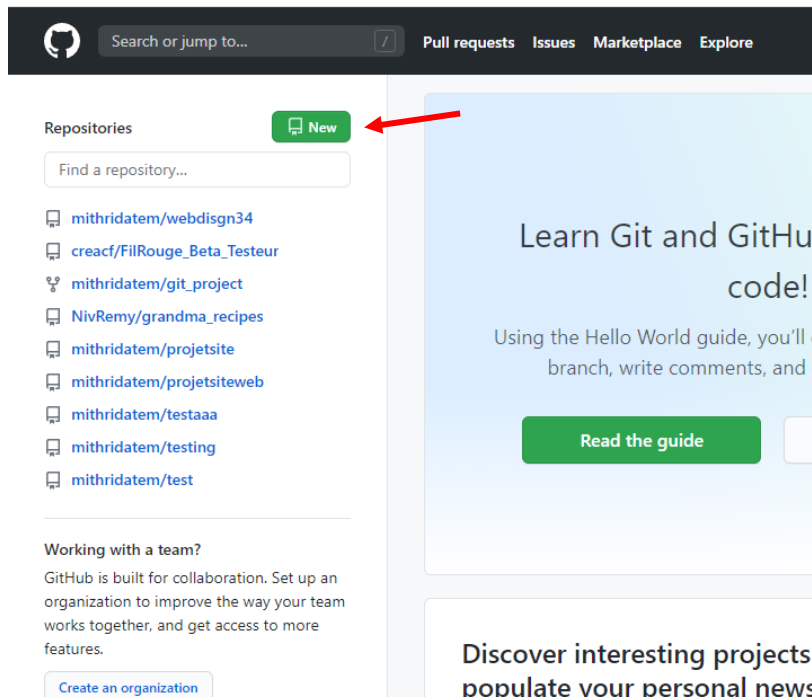
By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

Renseigner un Username (*nom utilisateur*)

Renseigner un Email (*adresse mail*)

Renseigner un Password (*mot de passe*)

## Créer un repository sur GitHub :



1 Cliquer sur new :

2 Choisir un nom pour votre dossier :

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 \*

Great repository names are short and memorable. Need inspiration? How about `musical-engine`?

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.](#)

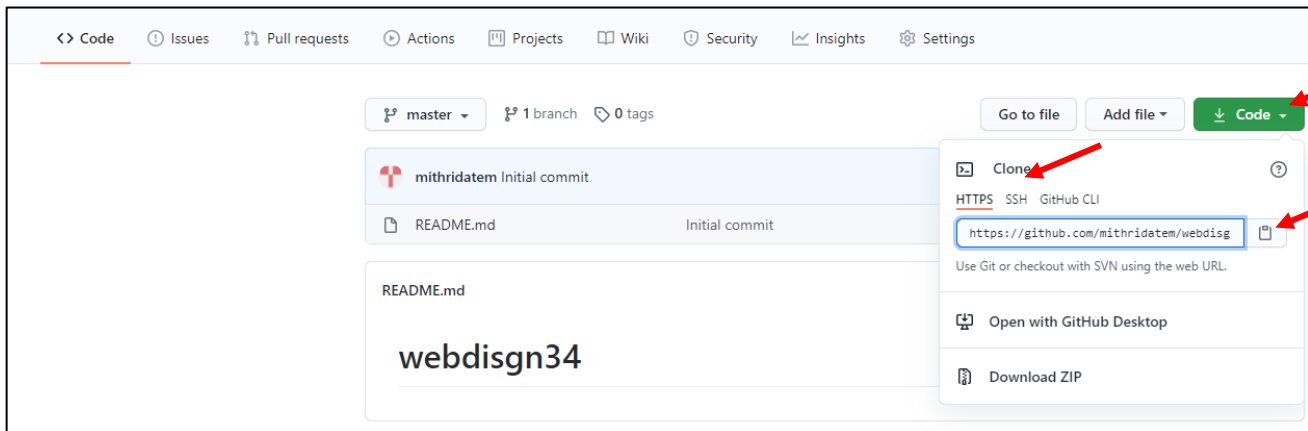
☐ Choose a license  
A license tells others what they can and can't do with your code. [Learn more.](#)

3 Cliquer sur create repository en bas de page (bouton)

## Cloner votre dépôt Github :

1 Cliquer sur code

2 Récupérer l'url de votre dépôt (ssh)



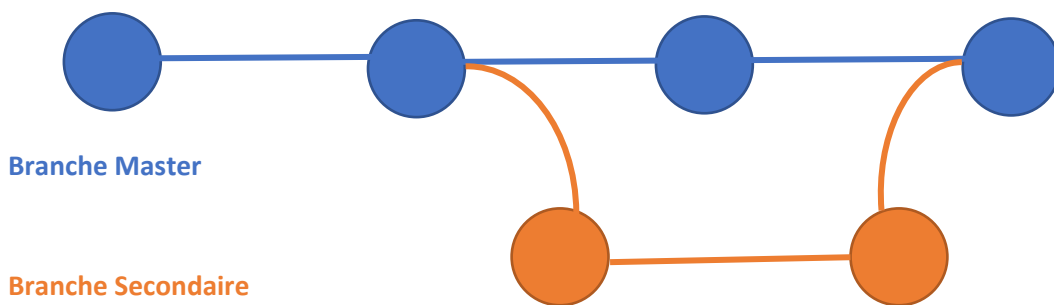
3 Taper la ligne de commande :

```
git remote add clone https://github.com/votre_url.git
```

## Système de branches :

Le principal atout de Git est son système de branches

La branche principale est appelée la branche **master**. C'est celle-ci, où au final, vous aurez à la fin toutes vos modifications. Le but est de ne surtout pas réaliser les modifications directement sur cette branche, mais de réaliser les modifications sur d'autres branches, et après tests, les intégrer sur la branche **master**.



### 1 afficher les branches :

```
git branch
```

Cela va s'afficher comme ci-dessous : (*git branch*)

```
git branch
*master
```

### 2 Créer une branche :

```
git branch nom de la branche (secondaire)
```

### 3 pour basculer sur la branche secondaire :

```
git checkout nom de la branche
```

Cette commande va afficher la liste des branches avec une astérisque sur la branche sélectionnée.

La branche va fonctionner comme un dossier virtuel. Avec la commande **Git checkout**, on va être téléporté dans le dossier virtuel **nom de la branche** « (secondaire) ».

### 4 sauvegarder les modifications :

Vous avez réalisé des évolutions sur la branche **nom de la branche** et il va falloir maintenant demander à Git de les enregistrer.

Pour cela on va utiliser la commande commit.

```
git commit -m "description de la modification"
```

## Ajouter un ou plusieurs fichiers à une branche et modifications :

On utilise la commande git add pour ajouter un fichier. (*Le fichier doit être déjà créé dans le répertoire*)

```
git add PremierFichier.txt
```

On utilise la commande git commit pour sauvegarder notre fichier.

```
git commit -m «nom du commit»
```

```
git commit
```

On vous demande alors d'indiquer le message du commit puis de valider. Pour valider le message, une fois que vous l'avez écrit, appuyez sur Echap (votre curseur va basculer sur la dernière ligne) et tapez **:x!**.

Cette commande va sauvegarder et quitter l'éditeur des messages de commit.

## Ajouter plusieurs fichiers à une branche :

On utilise la commande git commit pour sauvegarder tous nos fichiers.

```
git commit -am «nom du commit»
```

## Création d'une remise :

### 1 Voir l'état des fichiers :

```
git status
```

### 2 Créer une remise :

Une remise permet de mettre de côté des modifications pour pouvoir les copier sur une autre branche.

Modifier vos fichiers puis taper la commande.

```
git stash
```

### 3 Créer une nouvelle branche :

```
git branch brancheCommit
```

 (la nouvelle branche s'appelle brancheCommit)

### 4 Basculer sur la nouvelle branche (brancheCommit) :

```
git checkout brancheCommit
```

### 5 Application de la remise :

Cette commande va appliquer toutes les modifications précédentes (Créer une remise) sur la branche sélectionnée (brancheCommit)

```
git stash apply
```

### 6 voir la liste des remises :

```
git stash list
```

Cela permet de voir la liste des remises avec un identifiant associé à chacune

### 7 Application d'une remise avec son identifiant :

```
git stash apply stash@{0}
```

Cette commande applique sur la branche sélectionnée la remise « **stash@{0}** » (la première dans ce cas)



## 8 Restaurer la branche master :

Si vous avez effectué un commit (update des fichiers) non désiré on peut restaurer nos fichiers à un état antérieur.

Pour se faire on va lancer la commande suivante pour récupérer l'id du commit :

- Se positionner sur la branche master.  
`git checkout brancheCommit`
- Afficher le log des commits pour récupérer l'identifiant.  
`git log`

```
$ git log
commit ca83a6dff817ec66f443420071545390a954664949 (identifiant)
Author: nom utilisateur <mail utilisateur>
```

- `Date: Mon Mar 19 21:52:11 2019 -0700`
- Récupérer l'identifiant ci- dessus
- Assurez vous que vous êtes bien sur la branche master
- Taper la commande ci-dessous :  
`git reset --hard HEAD^`
- Basculer sur la branche ou vous voulez copier les modifications :
- `git checkout brancheCommit`
- Taper la commande ci-dessous : (on n'a besoin que des 8 premiers caractères de l'identifiant)
- `git reset --hard ca83a6df`
- Cela va appliquer un commit sur la branche sélectionnée (brancheCommit)

## 9 Modifier le message du dernier commit :

```
git commit --amend -m "Votre nouveau message de commit"
```

## 10 Ajouter un fichier oublié dans un nouveau commit :

Vous avez fait votre commit mais vous réalisiez que vous avez oublié un fichier. Ce n'est pas bien grave ! Nous allons réutiliser la commande `git --amend`, mais d'une autre manière. La fonction `git --amend`, si vous avez bien compris, permet de modifier le dernier commit.

Nous allons donc réutiliser cette fonction, mais sans le `-m` qui permettait de modifier son message.

Nous allons dans un premier temps ajouter notre fichier, et dans un deuxième temps réaliser le `git --amend`.

```
git add nom du fichier
git commit --amend --no-edit
```

Votre fichier a été ajouté à votre commit et grâce à la commande `--no-edit` que nous avons ajoutée, nous n'avons pas modifié le message du commit.

Pour résumer, `git commit --amend` vous permet de sélectionner le dernier commit afin d'y ajouter de nouveaux changements en attente. Vous pouvez ajouter ou supprimer des changements afin de les appliquer avec

`commit --amend`.

Si aucun changement n'est en attente, `--amend` vous permet de `git commit --amend -` le dernier message de log du commit avec `-m`. exemple ci-dessous :

```
git commit --amend -m « mon message » (voir section 9 Modifier le message du dernier commit )
```

## Sauvegarder vos fichiers sur un dépôt distant (ex Github) :

Pour sauvegarder vers un dépôt distant :

1 Récupérer l'url du dépôt (voir cloner votre dépôt sur Github page 5) :

2 Saisir la commande suivante :

```
git push url de votre dépôt github
```

Cela va ouvrir une fenêtre qui va vous demander de saisir vos identifiants login ou mail et mot de passe.

Si vous trompez de mot de passe ou d'identifiant relancer la commande.

NB : *vos identifiants et mot de passe seront sauvegardés.*

## Corriger un mauvais commit à distance :

1 Corriger un mauvais commit (le dernier) envoyé avec un push sur votre Github saisir la commande ci-dessous :

```
git revert HEAD^
```

## Réparer l'accès à distance :

Git base toute sa gestion d'authentification sur le mécanisme des clés SSH. Ce système est d'ailleurs immensément utile de façon générale sous Linux, Unix et OSX, dès qu'il s'agit de s'authentifier sur une machine tierce.

1 Génération des clés SSH :

```
ssh-keygen -t rsa -b 4096 -C "adresse_mail"
```

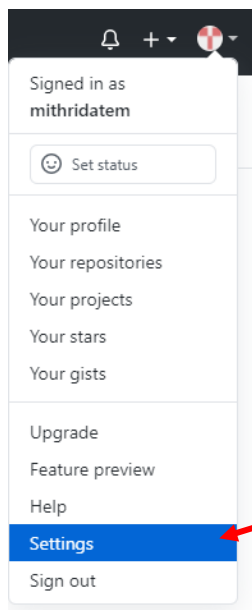
 saisir l'adresse mail créé au début.

Vous pouvez la retrouver à cette adresse :

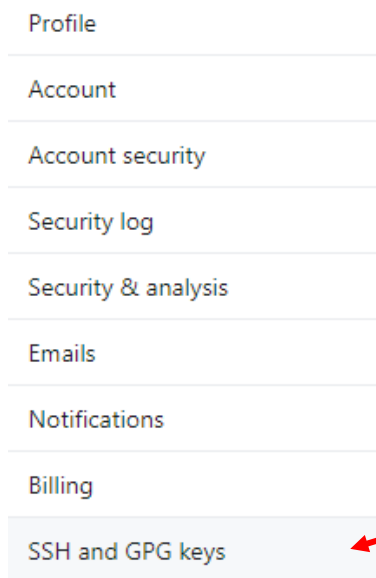
**C:\Users\VotreNomD'Utilisateur\.ssh**, et d'afficher les dossiers masqués.

2 Ajouter vos clés SSH à Github :

-Se connecter à github.com puis allez sur votre icone de profil (en haut à droite) et cliquez sur settings :



-Dans la fenêtre de profil cliquez sur SSH and GPG keys dans le menu à gauche :



-Cliquez sur new SSH Key :



There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

Copier-coller le contenu du fichier situé à cette adresse **C:\Users\VotreNomD'Utilisateur\.ssh**

Et donner un titre :

[SSH keys](#) / Add new

Title

Key

Begin with 'ssh-rsa', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

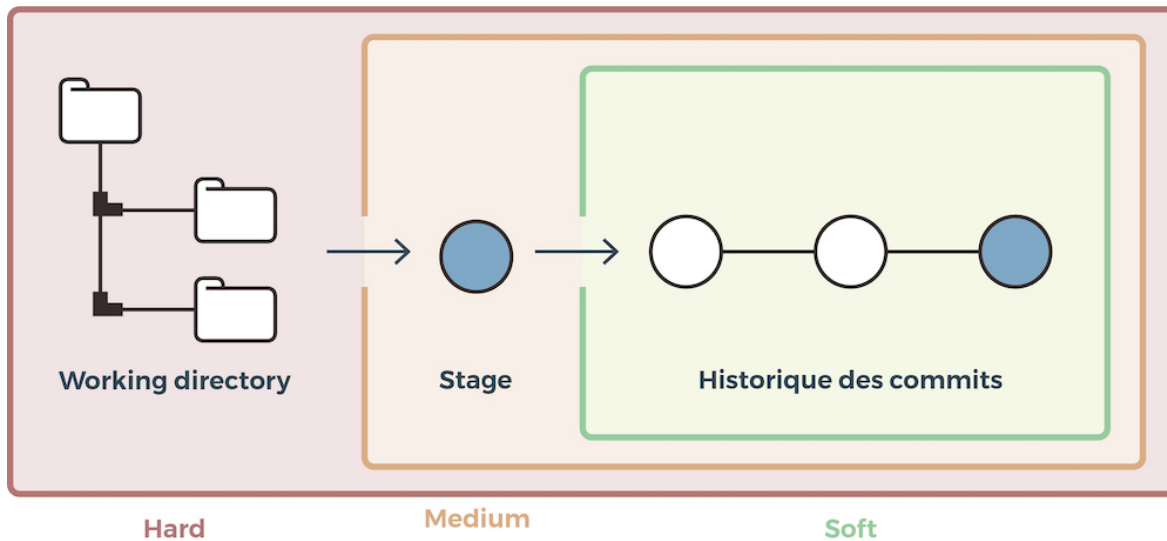
Add SSH key

Cliquer sur Add SSH key.

Vous devrez ensuite confirmer votre mot de passe, et votre clé SSH sera alors ajoutée à votre compte GitHub.

### Les 3 types de réinitialisations de GIT :

La commande `git reset` est un outil complexe et polyvalent pour **annuler les changements**. Elle peut être appelée de trois façons différentes, qui correspondent aux arguments de ligne de commande **--soft**, **--mixed** et **--hard**.



#### 1 Commande Reset --hard :

Cette commande permet de revenir à n'importe quel commit mais il faut faire très attention car l'on **perd tous les commits suivants !!!**

Pour se faire il nous faut l'identifiant du commit que l'on obtient avec la commande :

```
$ git log
```

Puis l'on saisit la commande suivante :

```
git reset identifiant_commit --hard
```

Cette commande est à utiliser en dernier recours car l'on **perd tous les commits suivants !!!!**

#### 2 Commande Reset --mixed :

Le `git reset --mixed` va permettre de revenir juste après votre dernier commit ou le commit spécifier, sans supprimer vos modifications en cours. Il va par contre créer un HEAD détaché. Il permet aussi, dans le cas de fichiers indexés mais pas encore commités, de désindexer les fichiers.

Si rien n'est spécifié après `git reset`, par défaut il exécutera un `git reset --mixed HEAD~`

#### 3 Commande Reset --soft :

Le `git reset --Soft` permet juste de se placer sur un commit spécifique afin de voir le code à un instant donné ou créer une branche partant d'un ancien commit. Il ne supprime aucun fichier, aucun commit, et ne crée pas de HEAD détaché.

## Corriger un dépôt local :

Nous allons voir dans cette partie comment corriger un dépôt local.

On va créer un nouveau dossier dans notre ordinateur qui va nous servir de bac à sable pour tester les corrections.

### 1 Créer un nouveau dossier sur votre ordinateur :

On va l'appeler test.

### 2 dans l'interface de git on va initialiser ce nouveau dossier :

Taper cd puis l'url de votre répertoire test :

```
$ cd C:/Users/nom_utilisateur/Desktop/test
```

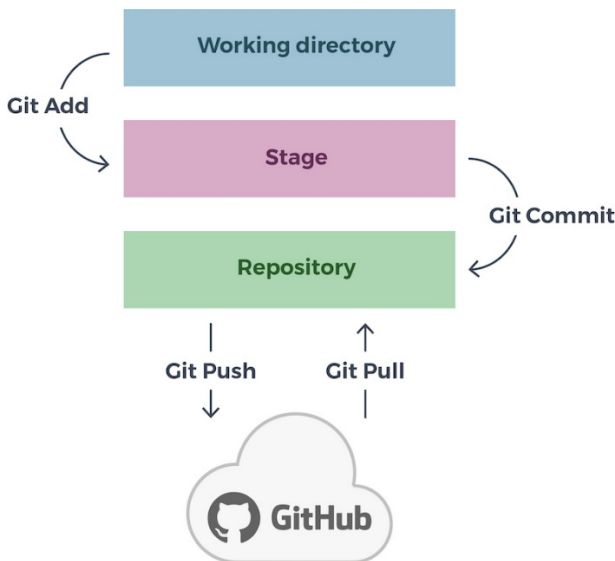
Il va nous falloir initialiser notre répertoire avec la commande ci-dessous :

```
$ git init
```

Votre dépôt est maintenant initialisé, et si vous faites apparaître les dossiers masqués, vous pouvez voir le dossier .git.

Git gère les versions de vos travaux locaux à travers 3 zones locales majeures :

- **Le répertoire de travail** (working directory/WD) ;
- **L'index, ou stage** (nous préférons le second terme) ;
- **Le dépôt local** (Git directory/repository).



L'index, ou stage, désigne tous les fichiers modifiés que vous souhaitez voir apparaître dans votre prochain commit. C'est avec la fonction **git add** que l'on ajoute un fichier au stage.

Le dépôt local est l'historique de l'ensemble de vos actions (commits, configurations...). L'archivage se fait principalement avec la commande **git commit**. Il est possible d'accéder à cet historique en faisant un **git reflog** qui affichera toutes vos actions et leurs SHA. Le SHA, c'est ce grand code qui vous permettra de revenir à un commit exact. C'est l'identifiant de votre action !

Passons à nos erreurs !

## 1° cas J'ai créé une branche que je n'aurais pas dû créer :

Avant de créer une branche, vous devez créer votre branche principale (master). Pour créer la branche master, vous devez simplement ajouter un fichier et le commiter.

1 Créez un fichier "PremierFichier.txt" dans votre répertoire Test, et ajoutez-le avec la commande :

```
git add PremierFichier.txt
```

```
git commit
```

On vous demande alors d'indiquer le message du commit puis de valider. Pour valider le message, une fois que vous l'avez écrit, appuyez sur Echap (votre curseur va basculer sur la dernière ligne) et tapez `:x` ou `:x!`

Cette commande va sauvegarder et quitter l'éditeur des messages de commit.

2 Nous allons maintenant créer une branche brancheTest avec la commande ci-dessous :

```
git branch brancheTest
```

Nous pouvons le vérifier avec la commande ci-dessous :

```
git branch
```

```
$ git branch
brancheTest
* master
```

Youppiiii !

En fait, non, nous voulions ajouter nos fichiers avant de la créer et nous sommes maintenant bloqués avec cette branche que nous ne voulions pas tout de suite. Heureusement, il est très simple sous Git de supprimer une branche que nous venons de créer.

3 Pour cela, il suffit d'exécuter la commande :

```
git branch -d brancheTest
```

Attention, si toutefois vous avez déjà fait des modifications dans la branche que vous souhaitez supprimer, il faudra soit faire un commit de vos modifications, soit mettre vos modifications de côté, et ça, je vous l'expliquerai un peu plus tard ;

4 Forcer la suppression en faisant :

```
git branch -D brancheTest
```

**Attention** : Forcer la suppression de cette manière entraînera la suppression de tous les fichiers et modifications que nous n'aurons pas commités sur cette branche.

## 2 ° cas J'ai modifié la branche principale :

Nous allons dans un premier temps voir ensemble le cas où vous avez modifié votre branche master mais que vous n'avez pas encore fait le commit, et nous verrons dans un second temps le cas où vous avez commité.

Vous avez modifié votre branche master avant de créer votre branche et vous n'avez pas fait le commit. Ce cas est un peu plus simple. Nous allons faire ce qu'on appelle une remise. La remise va permettre de mettre vos modifications de côté, le temps de créer votre nouvelle branche et ensuite appliquer cette remise sur la nouvelle branche.

Afin de voir comment cela fonctionne, allez sur votre branche master, modifiez des fichiers. Vous pouvez à tout moment voir à quel état sont vos fichiers en faisant :

```
Git status
```

#### 1 créer une remise :

```
git stash
```

Vous pouvez maintenant vous assurer que votre branche master est de nouveau propre, en faisant un nouveau `git status`

Vous devriez avoir :

```
$ git status
# On branch master
nothing to commit, working directory clean
```

#### 2 créer notre branche brancheCommit :

```
git branch brancheCommit
```

#### 3 basculer sur la nouvelle branche :

```
git checkout brancheCommit
```

Et finalement, nous allons pouvoir appliquer la remise, afin de récupérer nos modifications sur notre nouvelle branche.

#### 4 réaliser la remise :

```
git stash
```

Cette commande va appliquer la dernière remise qui a été faite. Si pour une raison ou une autre, vous avez créé plusieurs remises, et que la dernière n'est pas celle que vous souhaitiez appliquer, pas de panique, il est possible d'appliquer une autre remise. Nous allons d'abord regarder la liste de nos remises.

#### 5 afficher la liste des remises :

```
git stash list
```

Cette commande va nous retourner un "tableau" des remises avec des identifiants du style :

```
$ git stash list
stash@{0}: WIP on master: f337838 création de la branche master
```

Maintenant, admettons que vous ayez réalisé vos modifications et qu'en plus vous ayez fait le commit. Le cas est plus complexe, puisque vous avez enregistré vos modifications sur la branche master, alors que vous ne deviez pas.

#### 6 modifiez des fichiers, et réalisez-le commit :

Nous allons devoir aller analyser vos derniers commits avec la fonction **git log**, afin de pouvoir récupérer l'identifiant du commit que l'on appelle couramment le *hash*. Par défaut, **git log** va vous lister par ordre chronologique inversé tous vos commits réalisés.

```
$ git log
```

```
commit ca83a6dff817ec66f443420071545390a954664949
```

```
Author: Mathieu <mathieu.mith@laposte.com>
```

```
Date: Mon Mar 19 21:52:11 2019 -0700
```

Maintenant que vous disposez de votre identifiant, gardez-le bien de côté. Vérifiez bien que vous êtes sur votre branche master

```
git checkout master
```

7 réalisez la commande suivante :

```
git reset --hard HEAD^
```

Cette ligne de commande va permettre de supprimer de la branche master votre dernier commit. Le Head^ indique que c'est bien le dernier commit que nous voulons supprimer.

8 nous allons maintenant créer notre nouvelle branche :

```
git branch branchecommit
```

9 basculer sur cette branche :

```
git checkout brancheCommit
```

Maintenant que nous sommes sur la bonne branche, nous allons de nouveau faire un **git reset**, mais celui-ci va permettre d'appliquer ce commit sur notre nouvelle branche ! Il n'est pas nécessaire d'écrire l'identifiant en entier. Seuls les 8 premiers caractères sont nécessaires.

```
git reset --hard identifiant
```

Notre cas est résolu.

### 3 ° cas je souhaite changer le message de mon commit :

Lorsque l'on travaille sur un projet avec Git, il est très important, lorsque l'on propage les modifications, de bien marquer dans le message descriptif les modifications que l'on a effectuées. Si jamais ne vous faites une erreur dans l'un de vos messages de commit, il est tout à fait possible de changer le message après coup.

**Attention** cette commande va fonctionner sur votre dernier commit.

Imaginons que vous veniez de faire un commit et que vous ayez fait une erreur dans votre message. L'exécution de cette commande, lorsqu'aucun élément n'est encore modifié, vous permet de modifier le message du commit précédent sans modifier son instantané. L'option **-m** permet de transmettre le nouveau message.

```
git commit --amend -m "Votre nouveau message de commit"
```

On peut vérifier avec la commande **git log**.

### 4 ° cas J'ai oublié un fichier dans mon dernier commit :

Imaginons maintenant que vous ayez fait votre commit mais que vous réalisiez que vous avez oublié un fichier. Nous allons réutiliser la commande **git --amend**, mais d'une autre manière. La fonction **git --amend**, si vous avez bien compris, permet de modifier le dernier commit.

Nous allons donc réutiliser cette fonction, mais sans le **-m** qui permettait de modifier son message.

1 ajouter votre fichier, et dans un deuxième temps réaliser le **git --amend** :

```
git add nom du fichier.extension
```

```
git --amend --no-edit
```

Votre fichier a été ajouté à votre commit et grâce à la commande **--no-edit** que nous avons ajoutée, nous n'avons pas modifié le message du commit.

Pour résumer, **git commit --amend** vous permet de sélectionner le dernier commit afin d'y ajouter de nouveaux changements en attente. Vous pouvez ajouter ou supprimer des changements afin de les appliquer avec **commit --amend**. Si aucun changement n'est en attente, **--amend** vous permet de modifier le dernier message de log du commit avec **-m**.



## Corrigez vos erreurs en local et à distance :

Vous avez par mégarde push des fichiers erronés. Le problème, c'est que maintenant ce n'est plus que sur votre dépôt local, mais à disposition de tout le monde.

Il est possible d'annuler son commit public avec la commande **Git revert**. L'opération Revert annule un commit en créant un nouveau **commit**. C'est une méthode sûre pour **annuler des changements**, car elle ne risque pas de **réécrire l'historique du commit**.

1 utiliser la commande :

```
git revert HEAD^
```

Nous avons maintenant revert notre dernier commit public et cela a créé un nouveau commit d'annulation. Cette commande n'a donc **aucun impact sur l'historique** ! Par conséquent, il vaut mieux utiliser `git revert` pour annuler des changements apportés à une branche publique, et `git reset` pour faire de même, mais sur une branche privée.

## Modifiez vos informations d'identification et supprimez la clé :

1 exécuter la commande :

Cela va générer vos clé ssh.

```
ssh-keygen -t rsa -b 4096 -C "mail@utilisateur"
```

Pour la trouver, il suffit d'aller à l'adresse : **C:\Users\VotreNomD'Utilisateur\**, et d'afficher les dossiers masqués.

Dans ce dossier, vous avez donc deux fichiers, votre clé publique et votre clé privée.

La **clé id\_rsa.txt** est votre **clé privée** alors que la **clé id\_rsa.pub** est votre **clé publique**. Vous pouvez copier votre clé en l'ouvrant dans un bloc-notes.

Ajoutons maintenant notre clé à notre compte GitHub.

1 Connectez-vous à votre espace GitHub puis allez dans l'angle droit de votre compte et cliquez sur Settings.

2 Cliquez sur SSH and GPG keys :

3 Puis sur New SSH Key :

4 Choisissez un titre et collez votre clé SSH :

5 Cliquez sur le bouton add ssh key :

Vous devrez ensuite confirmer votre mot de passe, et votre clé SSH sera alors ajoutée à votre compte GitHub.

## Comment fonctionne la fusion sous Git ?

Il est très courant sous Git de vouloir fusionner le travail fait sur différentes branches. Pour cela, nous avons la fonction **Merge**. Un `git merge` ne devrait être utilisé que pour la récupération fonctionnelle, intégrale et finale d'une branche dans une autre, afin de préserver un graphe d'historique sémantiquement cohérent et utile, lequel représente une véritable valeur ajoutée. Comme son nom l'indique, `merge` réalise une **fusion**. `git merge` va combiner plusieurs séquences de commits en un historique unifié. Le plus souvent, `git merge` est utilisé pour combiner deux branches. `git merge` va créer un nouveau commit de merge.

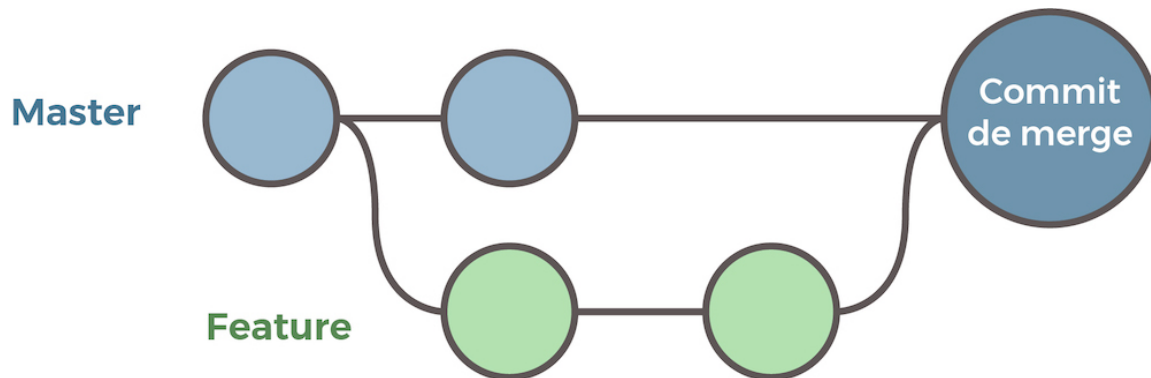
Imaginons que vous ayez votre branche master et une branche "Nouvelle fonctionnalité". Nous souhaitons maintenant faire un merge de cette branche de fonctionnalité dans la branche master. Appeler cette commande permettra de merger la fonctionnalité de branche spécifiée dans la branche courante, disons master.

**Attention** Il faut toujours préparer le terrain avant de réaliser un merge !

Vous devez toujours vous assurer d'être sur **la bonne branche**. Pour cela, vous pouvez réaliser un `git status`. Si vous n'êtes pas sur la bonne, réalisez un `git checkout`, pour changer de branche. Maintenant que le terrain est prêt, vous pouvez réaliser votre merge.

```
git merge Nouvelle Fonctionnalité
```

Votre branche Nouvelle fonctionnalité va être fusionnée sur la branche master en créant un nouveau commit.



Si les deux branches que vous essayez de fusionner modifient toutes les deux la même partie du même fichier, Git ne peut pas déterminer la version à utiliser. Lorsqu'une telle situation se produit, Git s'arrête avant le commit de merge, afin que vous puissiez résoudre manuellement les conflits.

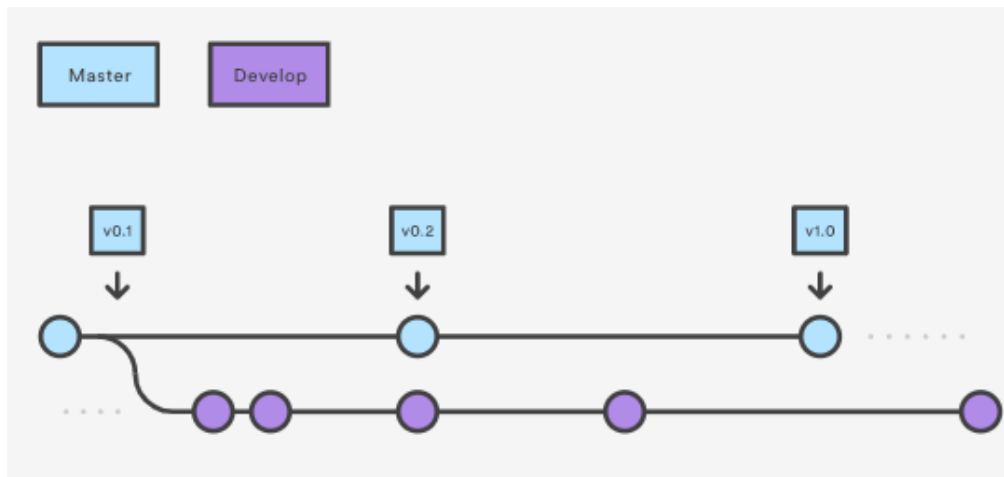
## GitFlow :

Le workflow Gitflow définit un modèle de création de branche strict conçu autour de la livraison de projet. Cela fournit un framework solide pour la gestion de projets plus importants.

En réalité, Gitflow n'est qu'une abstraction d'un workflow Git. Cela signifie qu'il dicte quel type de branches doit être configuré et comment les merger.

Sous Windows, vous devrez [télécharger et installer git-flow](#). Une fois **git-flow** installé, vous pouvez l'utiliser dans votre projet en exécutant `git flow init`. Git-flow est un wrapper Git. La commande **git flow init** est une extension de la commande **git init** par défaut. Elle ne change rien dans votre dépôt et ne fait que créer des branches pour vous.

Fonctionnement :



Ce workflow utilise deux branches au lieu d'une seule branche principale (master) pour sauvegarder l'historique du projet. La branche principale (master) stocke l'historique officiel des versions, et la branche de développement (develop) sert de branche d'intégration pour les fonctionnalités. Il peut également être utile de taguer tous les commits de la branche principale (master) avec un numéro de version.

La première étape consiste à compléter la branche principale (master) par défaut avec une branche de développement (develop). Une solution très simple consiste à créer une branche de développement (develop) vide en local et d'en faire un push vers le serveur :

```
git branch develop git push -u origin develop
```

Cette branche contiendra l'historique complet du projet, tandis que la branche principale (master) en contiendra une version abrégée. À ce stade, les autres développeurs devraient cloner le dépôt centralisé et créer une branche de développement (develop) de suivi.

**Sources :**

<https://fr.wikipedia.org/wiki/Git>

<https://git-scm.com/>

<https://openclassrooms.com/fr/courses/5641721-utilisez-git-et-github-pour-vos-projets-de-developpement/5641728-decouvrez-la-magie-du-controle-de-versions>

<https://www.atlassian.com/fr/git/tutorials/comparing-workflows/gitflow-workflow>