

---

# Cours GitHub

Les bases de GitHub



---

# Pourquoi Git ?

Git est un outil de versionning, c'est-à-dire qu'il permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Dans le monde du développement il est fortement utilisé lors des projets d'équipe. En effet celui-ci permet une gestion des avancés des projets et des suivis de développement très visuel et pratique et permet au développeur de ne plus avoir à donner son code dans une clé USB à ses collègues grace à l'utilisation de « forges » ( Sites d'hébergement ) !

Il permet aussi aux développeurs plus indépendant de partager son code avec le grand publique et permettre au développeur du monde entier de travailler sur le même projet, c'est l'open source !

---

# Les différentes étapes de ce module sur GitHub

- Mise en place de l'environnement Git
- Prise en main d'un terminal
- Création d'un repos sur GitHub
- Étude des différentes possibilités de gestion de projet avec Git (SSH et HTTPS)
- Mise en place d'une clé RSA pour utiliser Git en SSH
- L'ensemble des commandes de base
- La gestion des branches et résolution des conflits

---

## Comment utiliser Git


- On utilise des « forges », ou plus simplement hébergeur de repositories Git. Ce sont des interfaces web permettant de commander Git en ligne en utilisant moins les lignes de commande en terminal.
- C'est à dire, qu'il se charge d'héberger les versions de notre application et ainsi on peut y accéder en ligne et y autoriser des collaborateurs pour la reprise du code. C'est un peu comme Dropbox.
- Il existe plusieurs hébergeurs (gitbucket, gitlab etc ..), mais nous utiliserons pour notre cours GITHUB, le plus connu et le plus utilisé.

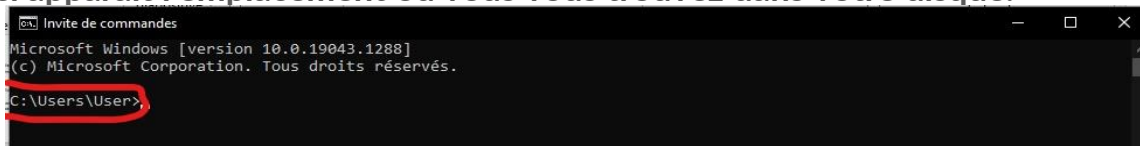
---

## Quelques mots à connaître

- Clés SSH = SSH utilise la cryptographie à clé publique (CP) ou *cryptographie asymétrique* pour sécuriser les connexions. La clé publique permet ainsi de valider l'identité d'un utilisateur et est propre à une machine.
- Repository = dossier .git qui contient toutes les infos de l'application
- Merger = fusionner une branche sur laquelle on travaille avec une autre branche . Je merge ma branche "panier" sur la branche « master » ou « main » (Branche principale).
- Add = ajouter les fichiers à ensuite Commiter.
- Commiter = créer une version.
- Push = envoyer les fichiers sur le serveur en ligne.
- Pull = Récupérer les fichiers de la dernière version en ligne.

## Quelques bases d'utilisation d'un terminal (Windows 1/2)

-  suivi de « cmd » puis la touche entrée vous permettra d'accéder à votre terminal de Windows à la racine de votre session d'utilisateur.
- Ici apparait l'emplacement où vous vous trouvez dans votre disque.



- La commande « dir » vous permettra d'avoir la liste des dossiers et fichiers présents là où vous vous trouvez .
- La commande « cd » suivie de « nomDeDossier » vous permettra de vous déplacer dans ce dossier, si vous utilisez la commande « cd » suivie de « ../ » la console sortira du dossier ou vous vous étiez déplacé. Si vous utilisez la commande « cd » suivie du début du nom de dossier et de « Tab », il autocomplétera le nom du dossier.

---

## Quelques bases d'utilisation d'un terminal (Windows 2/2)

- Si vous saisissez un nom de fichier ou d'exécutable présent dans le dossier ou vous vous trouvez avec son extension bien entendu celui-ci sera ouvert.
- Le chemin spécifié pour ouvrir un fichier ou se déplacer dans un dossier peut être écrit en chemin absolue (ex: cd « C:\Users\User\Pictures », ou «C:\Users\User\Pictures\logo.png » )
- La commande « mkdir » suivie de « nomDuDossier » créera un dossier de ce nom.
- La commande « copy NUL » suivie de « nomDuFichier.ext » créera un fichier de ce nom vide
- La commande « move » suivie de « nomDeFichier » ou « nomDeDossier » suivie de « cheminAbsolue » ou « cheminRelatif » déplacera celui-ci à l'endroit spécifié .
- Enfin la commande « del » suivie de « nomDeFichier » ou « nomDeDossier » supprime celui-ci

---

## Quelques bases d'utilisation d'un terminal (Mac 1/2)

- Ouvrez le dossier Applications puis le sous dossier Utilitaires et ouvrez Terminal.
- Toute commande précédée de « sudo » permet d'exécuter celle-ci en tant que superAdmin
- La commande « pwd », vous permet de savoir où vous vous trouvez dans le disque
- La commande « ls » vous permettra d'avoir la liste des dossiers et fichiers présents là où vous vous trouvez , « ls -lh » vous donnera la même liste avec beaucoup plus d'information, entre autre les droits de lecture et d'écriture.
- La commande « cd » suivie de « nomDeDossier » vous permettra de vous déplacer dans ce dossier, si vous utilisez la commande « cd » suivie de « ../ » la console sortira du dossier où vous vous étiez déplacé. Si vous utilisez la commande « cd » suivie du début du nom de dossier et de « Tab », il autocomplétera le nom du dossier.



---

## Quelques bases d'utilisation d'un terminal (Mac 2/2)

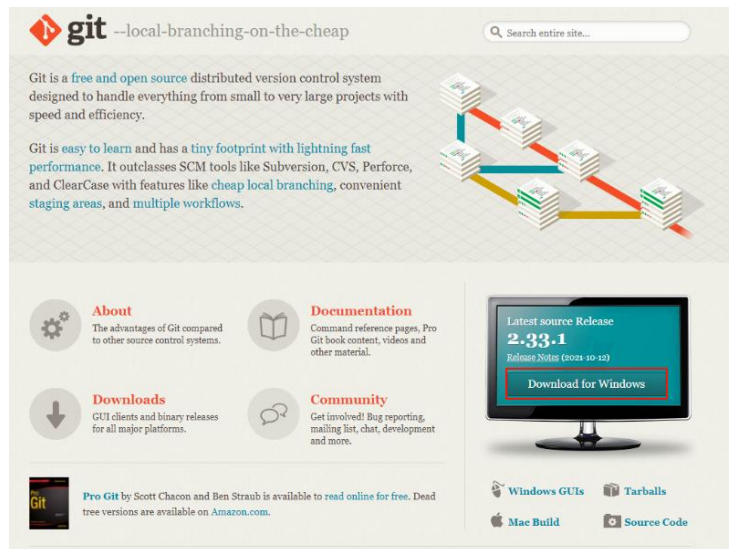
- Si vous saisissez un nom de fichier ou d'exécutable présent dans le dossier ou vous vous trouvez avec son extension bien entendu celui-ci sera ouvert.
- Le chemin spécifié pour ouvrir un fichier ou se déplacer dans un dossier peut être écrit en chemin absolue (ex: cd « C:\Users\User\Pictures », ou « C:\Users\User\Pictures\logo.png » )
- La commande « mkdir » suivie de « nomDuDossier » créera un dossier de ce nom.
- La commande « touch » suivie de « nomDuFichier.ext » créera un fichier de ce nom vide
- La commande « mv » suivie de « nomDeFichier » ou « nomDeDossier » suivie de « cheminAbsolue » ou « cheminRelatif » déplacera celui-ci à l'endroit spécifié .
- Enfin la commande « rm » suivie de « nomDeFichier » ou « nomDeDossier » supprime celui-ci

# Mise en place de Git

GitBash est le CLI (command line interface, correspond à la console) de Git qui nous permettra de communiquer avec nos repositories.

Pour ça rendez vous sur le lien suivant :

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



The screenshot shows the Git website homepage. At the top, there's a navigation bar with the Git logo and the tagline "--local-branching-on-the-cheap". A search bar is on the right. The main content area features a description of Git as a free and open source distributed version control system, highlighting its speed, efficiency, ease of learning, and tiny footprint. A diagram on the right illustrates branching and merging with stacks of papers. Below this, there are four sections: "About" (advantages compared to other systems), "Documentation" (reference pages, book content, etc.), "Downloads" (GUI clients, binary releases), and "Community" (bug reporting, mailing list, etc.). On the right side, there's a section for the "Latest source Release 2.33.1" with a "Download for Windows" button. At the bottom, there's a footer with links to "Pro Git" (a book by Scott Chacon and Ben Straub) and various download options like "Windows GUIs", "Mac Build", "Tarballs", and "Source Code".

**git** --local-branching-on-the-cheap

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

**Latest source Release**  
**2.33.1**  
Release Notes (2023-10-12)  
[Download for Windows](#)

**Pro Git** by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

[Windows GUIs](#) [Tarballs](#)  
[Mac Build](#) [Source Code](#)

---

# Commande de base de GitHub - git config

```
git config --global user.name "Votre Nom D'utilisateur GitHub"
```

Cette commande va nous permettre de définir le nom d'utilisateur utilisé lors des tentatives de push sur vos repos git

```
Utilisateur@DESKTOP-64F6T8L MINGW64 ~  
$ git config --global user.name "test"
```

```
git config --global user.email votremailgithub@test.fr
```

Cette commande va nous permettre de définir le mail utilisé lors des tentatives de push sur vos repos git

```
Utilisateur@DESKTOP-64F6T8L MINGW64 ~  
$ git config --global user.email test@test.fr
```

# Mise en place d'une paire clés SSH (windows)

## Générer une paire de clés SSH

Vous pouvez utiliser la commande `ssh-keygen` pour générer une paire de clés SSH (ou clés rsa), comme dans l'exemple ci-dessous. Deux fichiers seront créés avec cette commande et, par défaut, seront nommés `id_rsa` et `id_rsa.pub` et placés dans un répertoire `~/ssh/`.

Les deux fichiers sont de simples fichiers texte contenant vos clés privées (`id_rsa`) et publiques (`id_rsa.pub`). Vous pourrez télécharger le contenu du fichier de la clé publique dans votre instance pour pouvoir vous authentifier via SSH.

```
$ ssh-keygen -t rsa -b 2048

Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa
Your public key has been saved in /home/user/.ssh/id_rsa.pub
The key fingerprint is:
eb:79:fb:3f:15:ff:3a:04:dd:46:74:3b:da:a2:2b:9b user@hostname
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .o               |
|      +                |
|      . =              |
|      . +. +           |
|      S   + oo         |
|      . . o o          |
|      . . ...          |
|      . oo . o .       |
|      oE++..o+         |
+-----+

```

Une fois les fichiers créés, vous pouvez ouvrir le fichier de clé publique avec un éditeur de texte et copier son contenu dans votre presse-papiers.

---

## Mise en place d'une paire clés SSH (mac)

1. Ouvrir le terminal local sur votre léopard :

```
ssh-keygen -t rsa
```

2. Accepter le chemin par défaut qui vous est proposé par le prompt
3. Entrer 2 fois votre passphrase
4. Pour finir, copier votre clé publique sur le serveur en utilisant la commande suivante

```
cat ~/.ssh/id_rsa.pub | ssh login@monserver.com "cat - >> ~/.ssh/authorized_keys"
```

Veillez à modifier login@monserver.com avec vos réelles informations de connexion.

Pour faire de même avec tous vos serveurs, répéter l'étape 4

# Association clés publique avec GITHUB (windows)

On va pouvoir à présent copier/coller le contenu du fichier

Nom	Modifié le	Type	Taille
id_ed25519	28/10/2021 19:35	Fichier	1 Ko
id_ed25519	28/10/2021 19:35	Document Micros...	1 Ko
id_rsa	23/06/2021 20:30	Fichier	3 Ko
id_rsa	23/06/2021 20:30	Document Micros...	1 Ko
known_hosts	23/06/2021 20:57	Fichier	3 Ko

qui a été généré par l'étape précédente de notre génération de clés. Et la coller dans le champ "Key" sur la capture ci-contre.

SSH keys / Add new

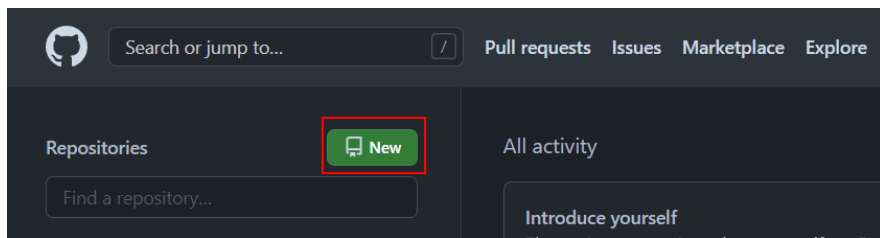
Title

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

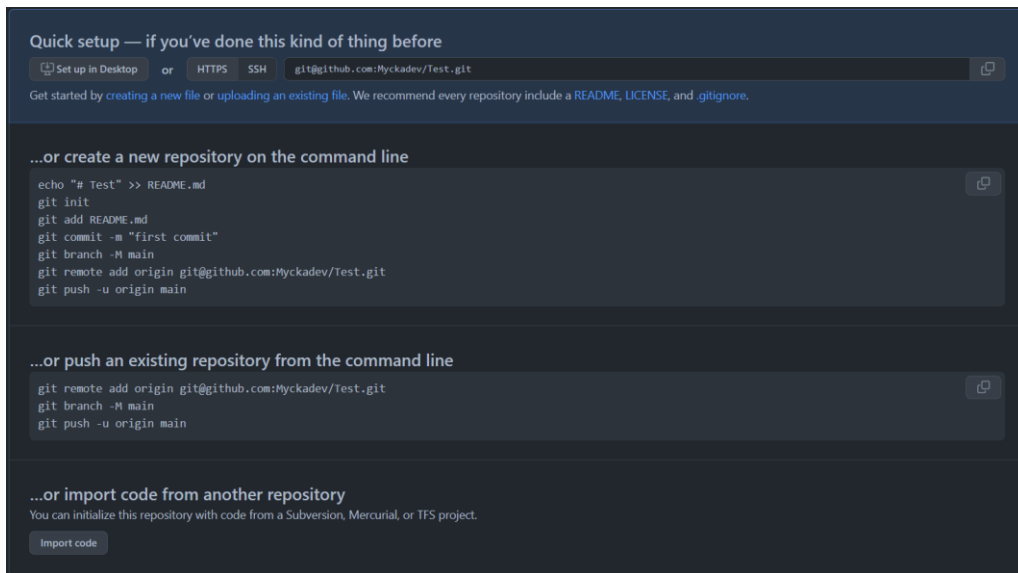
Add SSH key

## Mise en place de notre premier repos GitHub (1/2)



On va pouvoir créer un nouveau repos en cliquant sur le bouton encadré en rouge et en suivant les instructions

## Mise en place de notre premier repos GitHub (2/2)

A screenshot of the GitHub 'Quick setup' page. At the top, it says 'Quick setup — if you've done this kind of thing before'. Below this is a navigation bar with 'Set up in Desktop', 'or', 'HTTPS', 'SSH', and a text input field containing 'git@github.com:Myckadev/Test.git'. A note below the bar says 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.' The main content is divided into three sections: 1. '...or create a new repository on the command line' with a copy icon, containing a list of git commands: 'echo "# Test" >> README.md', 'git init', 'git add README.md', 'git commit -m "First commit"', 'git branch -M main', 'git remote add origin git@github.com:Myckadev/Test.git', and 'git push -u origin main'. 2. '...or push an existing repository from the command line' with a copy icon, containing the same git commands as the previous section. 3. '...or import code from another repository' with a note 'You can initialize this repository with code from a Subversion, Mercurial, or TFS project.' and an 'Import code' button.

```
Quick setup — if you've done this kind of thing before
```

Set up in Desktop or HTTPS SSH git@github.com:Myckadev/Test.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

### ...or create a new repository on the command line

```
echo "# Test" >> README.md
git init
git add README.md
git commit -m "First commit"
git branch -M main
git remote add origin git@github.com:Myckadev/Test.git
git push -u origin main
```

### ...or push an existing repository from the command line

```
git remote add origin git@github.com:Myckadev/Test.git
git branch -M main
git push -u origin main
```

### ...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Une fois le repos créé GitHub nous indique les étapes à suivre pour la mise en ligne de nos projets. C'est donc ce que nous allons faire, et nous allons en profiter pour voir un ensemble de commande de base de Git !



---

# Explication du process de GitHub

En effet avant de passer à la suite des commandes de base il est important de comprendre le process de GitHub. On a pu le voir dans un premier temps nous avons initialisé le projet. Nous avons ensuite définie un point d'entrée au projet qui sera sauvegardé sous le nom qu'on lui donne dans notre dossier .git situé à la racine du projet.

Maintenant nous allons définir un petit schéma du process à suivre pour prendre l'ensemble du projet, le mettre en mémoire tampon pour notre Git, lui indiquer un nom à l'ensemble des modifications qu'on lui envoie, et les envoyer.

```
git init -> git add -> git commit -> git push
```

NB : Nous allons voir chacune de ces commandes.

---

## Commande de base de GitHub - git init

Cette commande nous permettra d'initialisé un repos GitHub vide et générera un dossier .git avec l'ensemble des informations y compris l'entrée (remote) définie. Pour cela rendez-vous dans le répertoire courant de votre projet.

```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test  
$ git init  
Initialized empty Git repository in M:/test/.git/
```

NB : Toutes les commandes Git hormis celle vu précédemment se passeront **OBLIGATOIREMENT** dans le dossier du projet que vous voulez mettre sur GitHub.

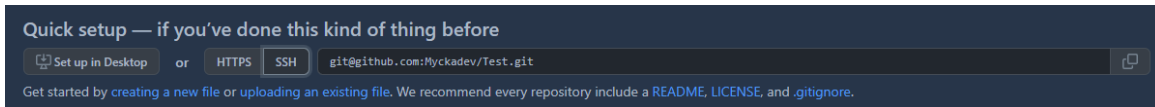
---

# Commande de base de GitHub - git remote

Maintenant que nous avons défini l'utilisateur qui allait utiliser Git grâce au git config nous allons devoir définir le point d'entrée de notre git nommé le remote.

```
git remote add origin <url>
```

Par défaut « origin » étant le point d'entrée du serveur et « url » sera par défaut lui défini par l'url de votre repos GitHub avec « : » remplaçant « / » après le « github.com » et l'extension « .git » ajouter à la fin



```
git remote remove|rm <nom>
```

Cette commande permet de supprimer une entrée et de la refaire ou tout simplement en supprimant le fichier caché .git à la racine de votre projet

---

# Commande de base de GitHub - git status

La commande git status va nous permettre de voir l'état des fichiers

Vérifier l'état des fichiers :

```
git status
```

```
utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
```

---

# Commande de base de GitHub - git add

Comme son nom l'indique cette commande permet d'indexer les changements des fichiers avant de soumettre les modifications au commit (commande que l'on verra par la suite)

```
git add . ou git add --all ou encore git add nomDefichier.ext
```

Ici on va pouvoir indexer l'ensemble des modifications de l'ensemble des fichiers du répertoire courant (Projet)

```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git add .

Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   test.txt
```

Ici on constate que lorsque l'on réutilise la commande

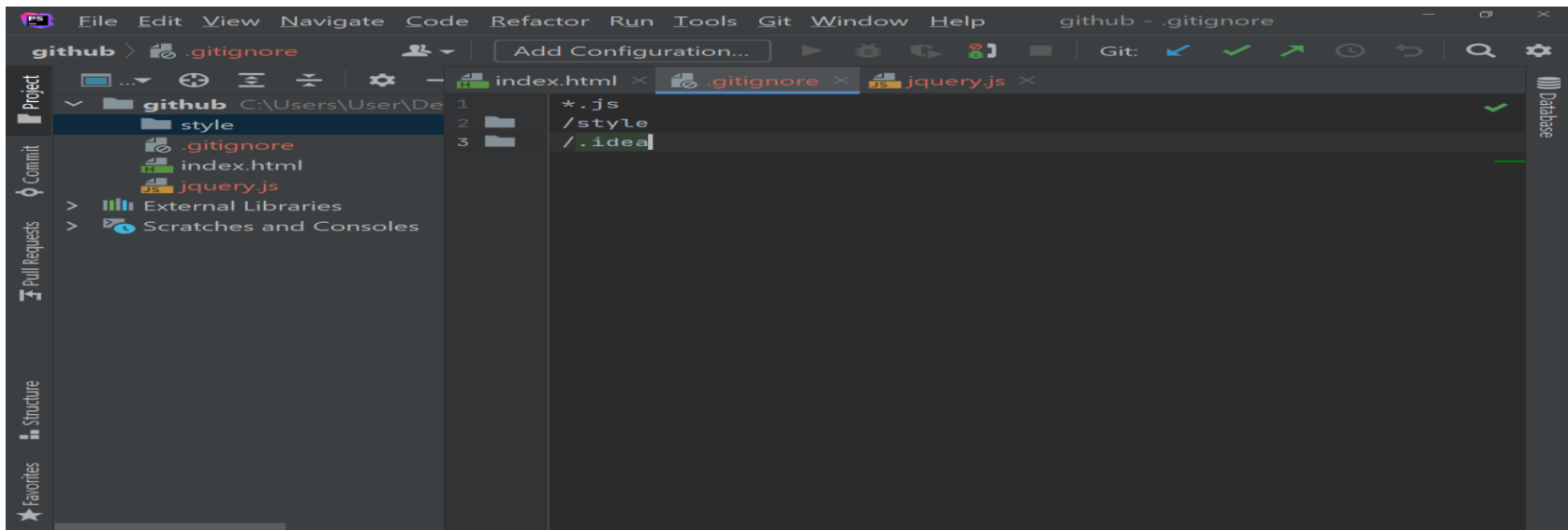
```
git status
```

On constate que notre fichier test est indexé à notre projet git

---

# Le .gitignore

- Alors on a vu que l'on pouvait pousser un fichier, plusieurs fichiers ou tous. Mais que faire si on ne veut jamais pousser certains fichiers, car trop volumineux ou ré installable par composer ou npm. Eh bien on crée à la racine de notre projet un fichier que l'on nomme « .gitignore »



- Ici j'ai indiqué dans mon `.gitignore` que je souhaite ne pas push le contenu du dossier `style` et le dossier en lui-même, pareille pour le `.idea` et tout les fichiers se terminant par `.js`
- Pour créer rapidement des `.gitignore` fonctionnel nous avons a disposition un outil : [gitignore.io \(https://toptal.com/developers/gitignore\)](https://toptal.com/developers/gitignore) où l'on pourra spécifier les types de fichier qu'on ne souhaite pas push et le site construira le `.gitignore` tout prêt à copier/coller.

# Commande de base de GitHub - git commit & git log

## Valider les modifications

```
git commit -m "Mon premier commit"
```

La commande «commit» est faite pour valider ce qui a été indexé avec «git add».

Après l'option -m (message) est suivi d'un commentaire de l'utilisateur

décrivant ce qui a été accompli et le fichier est ajouté au répertoire Git/dépôt (local) mais pas encore sur le dépôt

distant. (noté qu'il est important dans un process de teamworking d'avoir des

noms de commit les plus explicites possible afin d'assurer la compréhension

travail).

## Visualiser l'historique des validations

```
git log
```

Par défaut, git log énumère en ordre chronologique inversé les commits réalisés. Cela signifie que les commits les plus récents apparaissent en premier

```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git commit -m "mon premier commit"
[master (root-commit) 386fcb3] mon premier commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.txt
```

```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git log
commit 386fcb32ec6158f4070d9191f81e06dbb5412d5c (HEAD -> master)
Author: [redacted]
Date: Thu Nov 4 17:49:36 2021 +0100

    mon premier commit
```



---

# Commande de base de GitHub - git push

Pousser son travail sur un dépôt distant

```
git push origin master
```

(On se souvient qu'ici "origin" désigne le remote (point d'entrée) et "master" la branch

La commande «push» sert à envoyer tous les «commits» effectués se trouvant dans le répertoire Git/dépôt (HEAD) de la copie du dépôt local vers le dépôt distant.

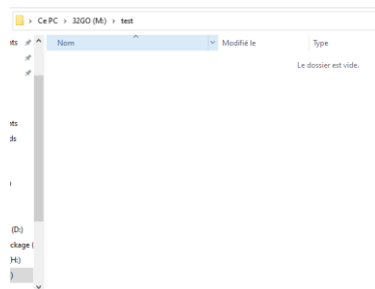
```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git push origin master
Enter passphrase for key '/c/Users/Utilisateur/.ssh/id_ed25519':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 266 bytes | 266.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Myckadev/Test.git
 386fcb3..2c01205 master -> master
```

# Commande de base de GitHub - git pull

Récupérer et tirer depuis des dépôts distants

git pull

La commande «pull» permet de mettre à jour votre dépôt local des dernières validations (modifications des fichiers). La commande est faite avant l'indexation des modifications.



```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/dzed (master)
$ git pull origin master
Enter passphrase for key '/c/Users/Utilisateur/.ssh/id_ed25519':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 433 bytes | 1024 bytes/s, done.
From github.com:Myckadev/Test
 * branch      master      -> FETCH_HEAD
 * [new branch] master      -> origin/master
```

Nom	Modifié le	Type	Taille
test	04/11/2021 18:08	Document texte	1 Ko

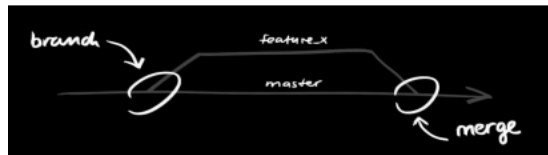
---

# Les branches avec Git

Faire une branche signifie diverger de la ligne temporelle principale de développement et continuer à travailler sans se préoccuper de cette ligne principale.

La branche par défaut dans Git quand vous créez un dépôt s'appelle main et elle pointe vers le dernier des commits réalisés.

Pourquoi des branches ?



Pouvoir se lancer dans des évolutions ambitieuses en ayant toujours la capacité de revenir à une version stable que l'on peut continuer à maintenir indépendamment.

Pouvoir tester différentes implémentations d'une même fonctionnalité de manière indépendante.

---

## Commande de base de GitHub - git branch & git checkout

Créer une nouvelle branche nommée « nomdelabranche »

```
$ git branch "nomdelabranche"
```

Basculer vers une branche existante

```
$ git checkout "nomdelabranche"
```

Cela déplace (HEAD) le pointeur vers la branche nomdelabranche. Tous les commits à ce moment sont fait sur la branche courante

Supprimer la branche

```
$ git branch -d "nomdelabranche"
```

Ici l'option -d désigne "delete" pour supprimer la branch

---

# Commande de base de GitHub - git merge

Incorporation des modifications d'une branche dans la branche courante (HEAD) par fusion (merge)

```
$ git merge "nomdelabranche"
```

Fusion d'une autre branche avec la branche active (par exemple master). Il est possible qu'il y ait des conflits à résoudre lors d'un merge.

---

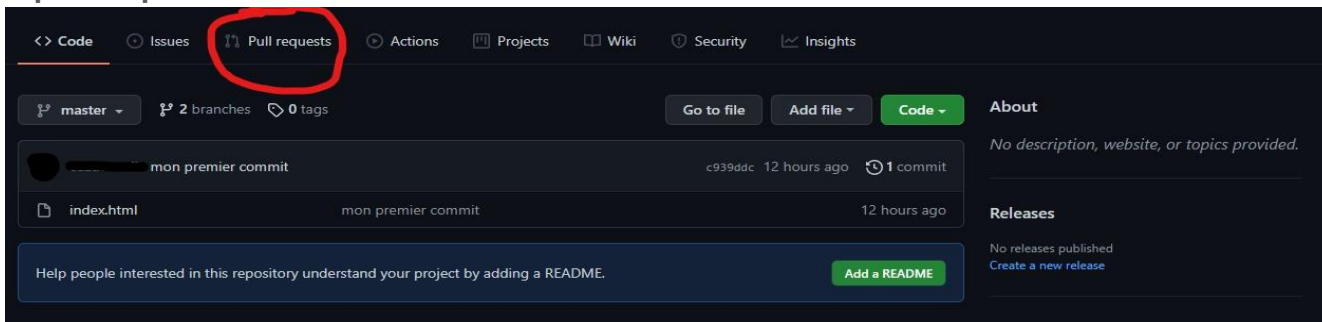
# Conflit de merge avec Git

Lorsque vous modifiez différemment la même partie du même fichier dans les deux branches que vous souhaitez fusionner, Git ne sera pas capable de réaliser proprement la fusion :

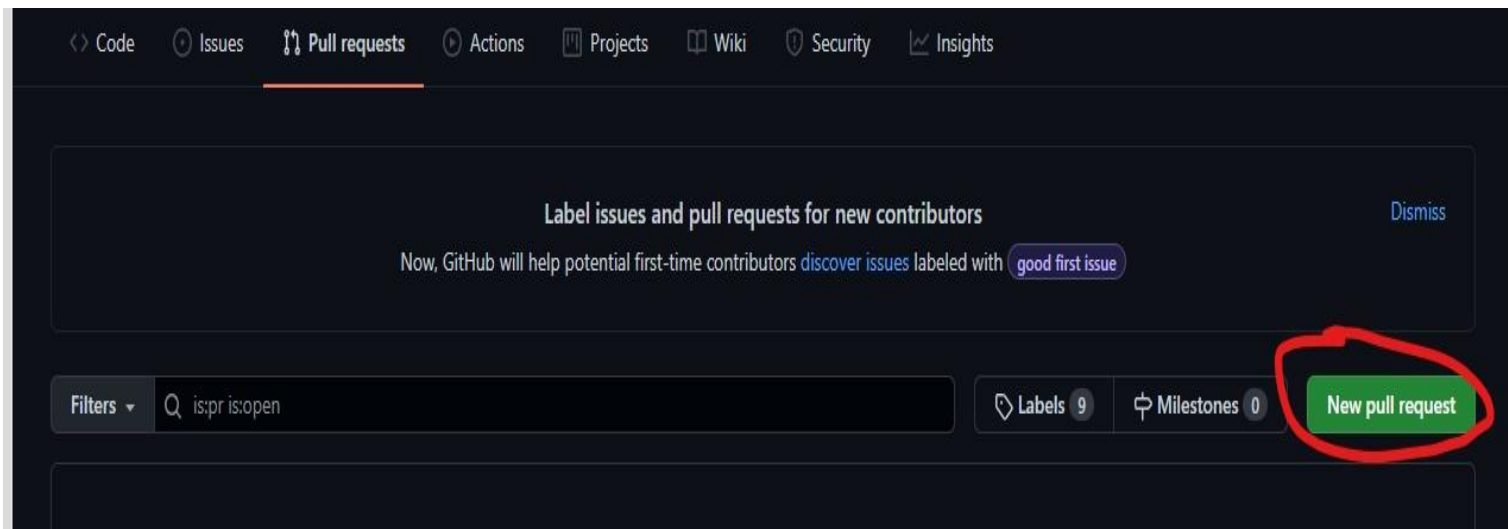
- Aucun “commit” de fusion n’est créé et le processus est mis en pause.
- Vous devez alors régler ces conflits manuellement en éditant les fichiers indiqués par git.
- Pour éviter cela plusieurs possibilités:
  - La première étant bien entendu de ne pas travailler sur le même code, un programmeur a un bout de code à réaliser sur lequel il est censé être le seul à travailler
  - Si ce n’est pas le cas nous allons voir le principe de « pull request » de GITHUB qui permet de transmettre une notification à la personne ayant produit le code présent sur la branche « master », cette notification lui indiquant de vérifier le code sur l’autre branche et de valider ou non les modifications apportées. Si tout est en ordre alors cette personne procèdera au merge de la branche sur la branche « master » et probablement à sa suppression

# Mise en situation

- Paul est devenu le collaborateur de Jean sur un projet. Il l'a cloné et commence à travailler sur sa nouvelle branche « whiteBackground » sur laquelle il effectue des changements sur le style de l'index.html, trouvant le background trop « flashy ». Il fait à présent son « git add index.html » puis son « git commit - m « change to white Background » » et enfin son « git push -u origin whiteBackground ». Jusque là aucun problème, à présent il va sur son GitHub et fait une demande de pull requests.

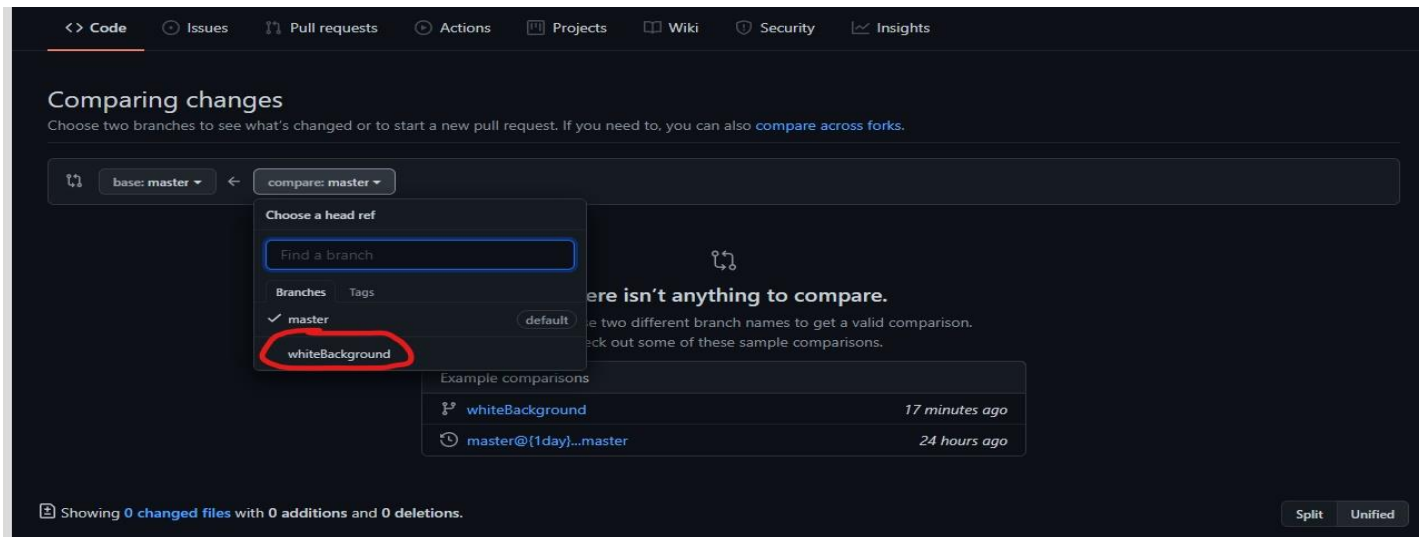


- Cette demande consiste à faire valider les changements effectués entre 2 branches.





- Il choisit donc entre quelle et quelle branche la comparaison doit être effectuée



- Ceci lui montre donc quels sont les changements entre les 2 branches, il n'a plus qu'à créer sa pull request

**Comparing changes**  
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base: master ← compare: whiteBackground ✓ **Able to merge.** These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) **Create pull request**

1 commit 1 file changed 1 contributor

Commits on Nov 10, 2021

**change to white background**  
committed 18 minutes ago

Showing 1 changed file with 1 addition and 1 deletion. Split Unified


@@ -10,7 +10,7 @@	
10 <body>	10 <body>
11 <style>	11 <style>
12 body{	12 body{
13 - background-color: green;	13 + background-color: white;
14 margin: 20% 40%;	14 margin: 20% 40%;
15 }	15 }
16	16

# Suite

- Il saisit un message avec @jean (pseudo de GitHub) qui lui transmet une notification

**Open a pull request**  
Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master ← compare: whiteBackground ✓ **Able to merge.** These branches can be automatically merged.



Write Preview H B I ≡ <> 🔗 ≡ ≡ ≡ @ 🗨️ ↶

dis moi si ca te convient @

Attach files by dragging & dropping, selecting or pasting them.

**Create pull request**

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

**Reviewers**  
No reviews

**Assignees**  
No one—assign yourself

**Labels**  
None yet

**Projects**  
None yet

**Milestone**  
No milestone

**Linked issues**  
Use **Closing keywords** in the description to automatically close issues

**Helpful resources**  
[GitHub Community Guidelines](#)

1 commit 1 file changed 1 contributor

- Jean répond donc à cette demande en annotant le fait que ce changement n'est pas possible vu que la couleur d'écriture est en blanc aussi et que de ce fait background blanc et color blanc donne tout blanc.

The screenshot shows a GitHub pull request titled "change to white background #1". The pull request is from the "whiteBackground" branch to the "master" branch. A comment from a collaborator, posted 19 minutes ago, asks "dis moi si ca te convient @...". A review from the owner, posted 1 minute ago, shows a code change in "index.html" where the background color is changed from green to white. The review comment, circled in red, states: "Non impossible, si tu souhaites faire ce changement change les couleurs d'écriture en autre que Blanc, sinon on ne verra rien". Below the review, a comment from the collaborator, posted 26 seconds ago, is also circled in red and says: "blanc sur blanc => tout blanc". The right sidebar shows the pull request details, including reviewers, assignees, labels, projects, milestones, linked issues, and notifications.

change to white background #1  
wants to merge 1 commit into master from whiteBackground

commented 19 minutes ago Collaborator

dis moi si ca te convient @...

change to white background ba0de1a

reviewed 1 minute ago View changes

```
index.html
+++ --- @@ -10,7 +10,7 @@
10 10 <body>
11 11 <style>
12 12 body{
13 - background-color: green;
13 + background-color: white;
```

reviewed 1 minute ago Owner

Non impossible, si tu souhaites faire ce changement change les couleurs d'écriture en autre que Blanc, sinon on ne verra rien

Reply...

Resolve conversation

reviewed 26 seconds ago View changes

left a comment Owner

blanc sur blanc => tout blanc

Reviewers

Still in progress? Convert to draft

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked issues

Successfully merging this pull request may close these issues.

None yet

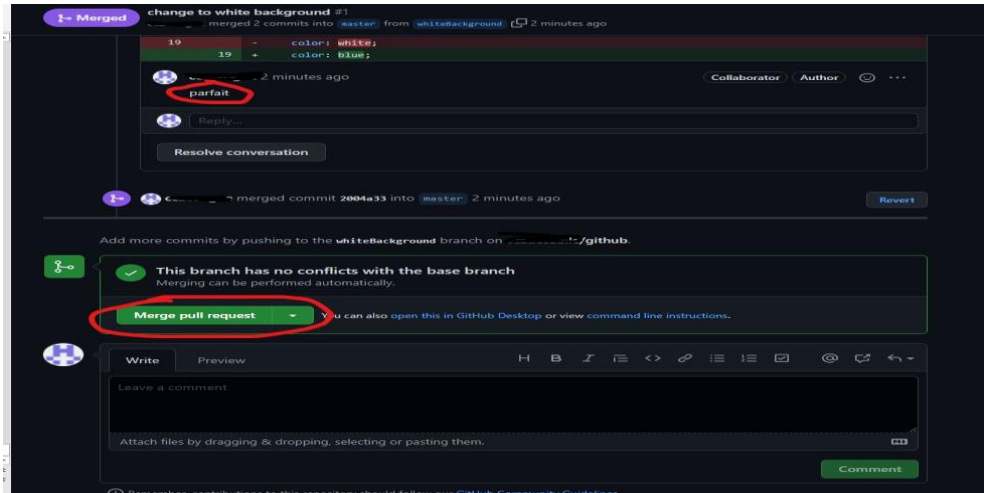
Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

2 participants

- Enfin Paul recevra une notification de ce commentaire et effectuera les modification sur la couleur de l'écriture avant de refaire un nouveau add->commit->push et de refaire une pull request.
- Jean validera ce changement et effectuera lui-même le merge sur la branche principale et ainsi le problème est réglé



## Les commandes de base : git diff

- On a vu juste au dessus la possibilité de voir les modification lors d'un merge directement sur GitHub. Mais il est possible de les voir via une commande directement sur le terminal avec la commande git diff.

```
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
$ git diff
diff --git a/text.txt b/text.txt
index 0c99137..832df97 100644
--- a/text.txt
+++ b/text.txt
@@ -1,6 @@
-Ceci est un test
+Bonjour je pense que ce texte est un Lorem ipsum dolor sit amet, consectetur ad
+ipiscing elit, sed do eiusmod
+tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
+quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
+consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
+cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
+proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
\ No newline at end of file

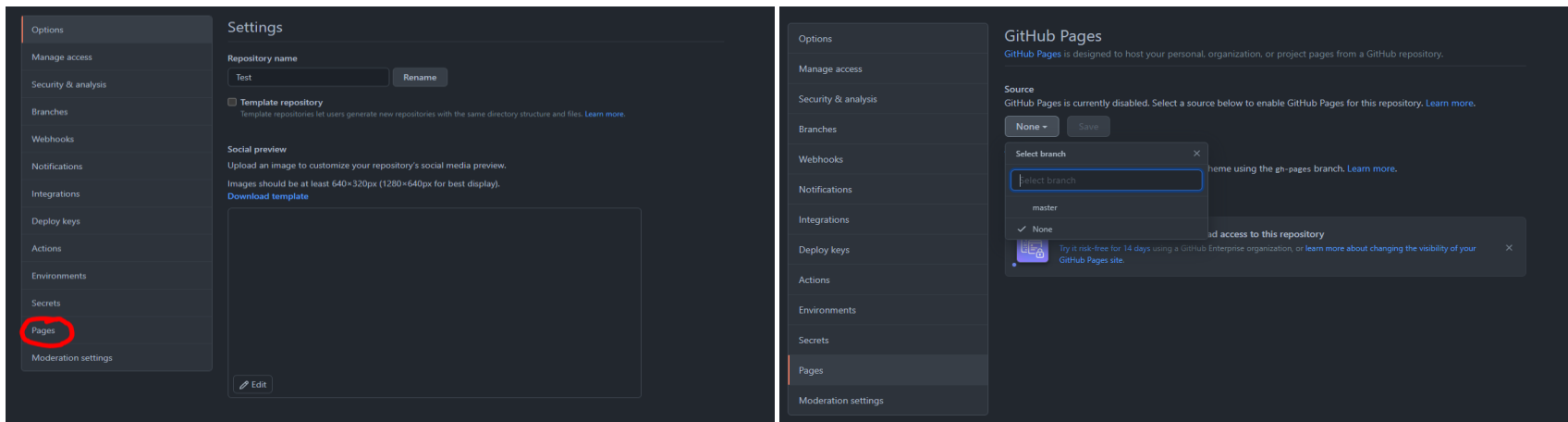
Utilisateur@DESKTOP-64F6T8L MINGW64 /m/test (master)
```

- Ici en rouge se trouve les lignes supprimé par rapport au précédent push, et en vert les lignes ajouté par rapport au précédent push

# Pour finir GitHub et Pages

- GitHub nous offres la possibilité de mettre en ligne nos repos gratuitement (uniquement pour les sites statiques).

Voici comment procéder et à quoi ça ressemble. (Attention, le repos doit être publique)



# Suite

**Options**

Manage access

Security & analysis

Branches

Webhooks

Notifications

Integrations

Deploy keys

Actions

Environments

Secrets

**Pages**

Moderation settings

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

**Source**

GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. [Learn more.](#)

Branch: master / (root) **Save**

**Theme Chooser**

Select a theme to publish your site with a Jekyll theme using the gh-pages branch. [Learn more.](#)

**Choose a theme**

**Publish privately to people with read access to this repository**

Try it risk-free for 14 days using a GitHub Enterprise organization, or [learn more about changing the visibility of your GitHub Pages site.](#)

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is published at <https://myckadev.github.io/Test/>

Source

- Choisissez la branche que vous souhaitez mettre en ligne et cliquez sur « save », le tour est joué
- N'oubliez pas d'avoir un index à la racine de votre projet (point d'entrée de votre site)
- Cliquez sur le lien et profiter de la vue de votre travail !