

Un peuple-Un but-Une foi



**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE
DIRECTION GENERALE DE L'ENSEIGNEMENT SUPERIEUR**

**GROUPE
ISI**

VIRTUALISATION – RESEAUX ET SERVICES

Présentation de Git & GitHub



Sommaire :

- I. Qu'est-ce que Git, GitHub ?
- II. Son rôle dans le développement de logiciels
 - a. Outils complémentaires
 - b. Avantage d'utilisation de Git
- III. Installation de git sur windows
- IV. Mise en place de l'environnement de travail et la découverte des commandes.

Conclusion

I. Qu'est-ce que Git ? GitHub ?

Git est un système de contrôle de version ou VCS (Version Control System) qui nous permet de gérer nos projets, il est libre et open source.

Un VCS est un logiciel dont le rôle est d'enregistrer l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler les versions antérieures de ce fichier à tout moment.

Les projets de développement de logiciel sont en générale dans des dossiers ou répertoires dans lesquels se trouve un ensemble de fichiers contenant des instructions transcrites selon la syntaxe de langage utilisée. Git est donc un outil qui nous permettra d'avoir une trace sur les différents changements ou mise à jour apportés au cours d'un projet spécifique et nous aide à les envoyer sur le web ou l'héberger sur GitHub.

GitHub est service d'hébergement open source, permettant aux programmeurs et développeurs d'héberger et de partager leur code informatique afin de travailler de façon collaborative.

II. Le rôle de git et github dans le développement de logiciel

Git gère les ajouts et changements apportés au code source de manière tracée.

Ainsi, si une erreur est commise, les développeurs peuvent revenir en arrière et comparer les versions antérieures du code, ce qui leur permet de corriger l'erreur tout en minimisant les perturbations pour tous les membres de l'équipe.

Git facilite donc le travail collaboratif en minimisant les risques de perte de travail et en permettant également aux développeurs de travailler chacun sur leurs branches et donc en autonomie sans empiéter sur le travail des autres.

En terme de sécurité, l'intégrité du code source géré par Git en fait un outil primordial. En effet, l'intégrité a été la priorité absolue lors de la conception de Git. Ainsi la traçabilité des changements ne peut jamais être remise en cause.

Les équipes de développement qui n'utilisent aucune forme de contrôle de version se heurtent souvent à des problèmes tels que le fait de ne pas savoir quels changements ont été mis à la disposition des utilisateurs ou l'apport de changements incompatibles entre deux éléments indépendants, qui doivent alors être minutieusement démêlés et retravaillés.

a. Outils complémentaires

On ne peut parler de Git sans parler également des systèmes d'hébergement et de gestion de code.

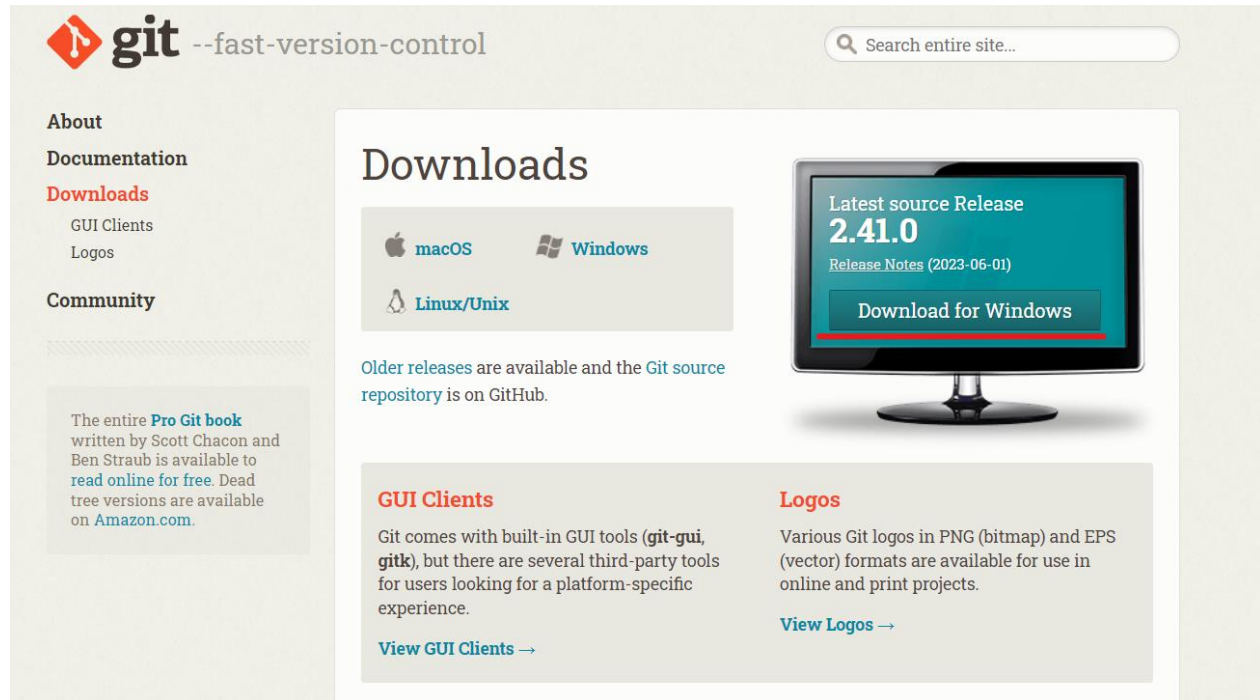
Les principaux sont **Github**, **Gitlab** ou bien **BitBucket**. Ces hébergeurs sont complémentaires à Git et viennent en extension de celui-ci.

Ils permettent notamment d'héberger le code sur internet et donc de le rendre accessible facilement aux équipes, tout en offrant une interface graphique simple et intuitive aux fonctionnalités prévues par Git.

b. Quelques avantages d'utilisation Git

- La collaboration : si plus d'une personne travaille sur le même projet ou sur un même code, git nous permet de fusionner proprement les différents changements de manière logique.
- Historique des différentes versions
- Développement en parallèle avec le système de branche.

III. Installation de Git



The screenshot shows the Git website homepage. At the top left is the Git logo with the tagline "--fast-version-control". A search bar is at the top right. On the left sidebar, there are links for "About", "Documentation", "Downloads" (highlighted), "GUI Clients", "Logos", and "Community". The main content area has a "Downloads" section with buttons for macOS, Windows, and Linux/Unix. To the right, a monitor displays the "Latest source Release 2.41.0" with a "Download for Windows" button. Below this, there are sections for "GUI Clients" and "Logos". A sidebar on the left mentions the "Pro Git book".

git --fast-version-control

Search entire site...

About
Documentation
Downloads
GUI Clients
Logos
Community

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

Downloads

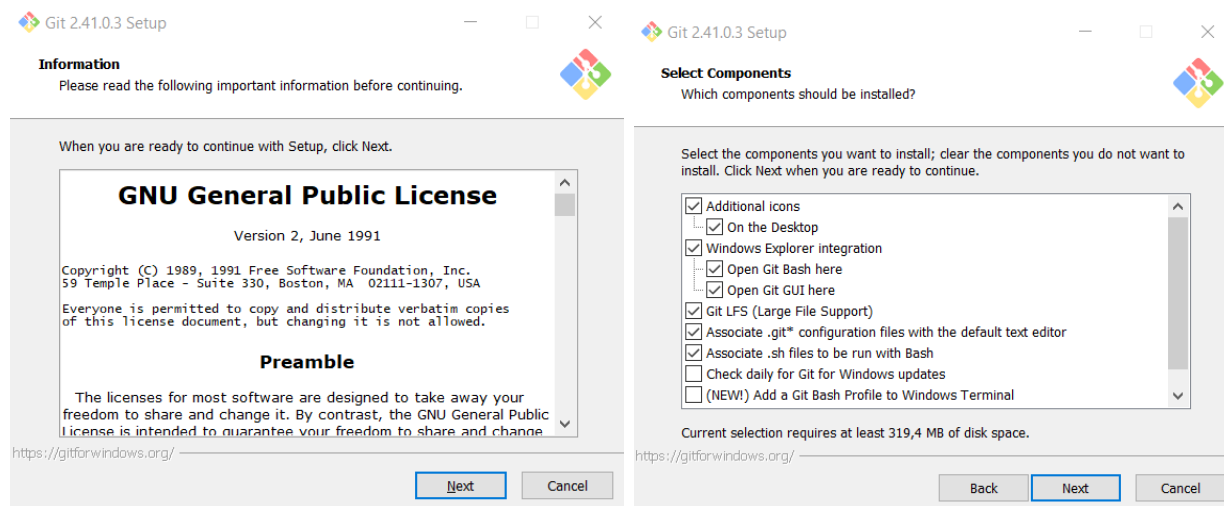
macOS Windows Linux/Unix

Older releases are available and the Git source repository is on GitHub.

GUI Clients
Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.
[View GUI Clients →](#)

Logos
Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.
[View Logos →](#)

Latest source Release
2.41.0
[Release Notes \(2023-06-01\)](#)
[Download for Windows](#)



The screenshot shows the Git 2.41.0.3 Setup window. It has two panes. The left pane, titled "Information", shows the "GNU General Public License" text. The right pane, titled "Select Components", shows a list of components to be installed. The "Next" button is highlighted in both panes.

Git 2.41.0.3 Setup

Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

<https://gitforwindows.org/>

Next Cancel

Git 2.41.0.3 Setup

Select Components
Which components should be installed?

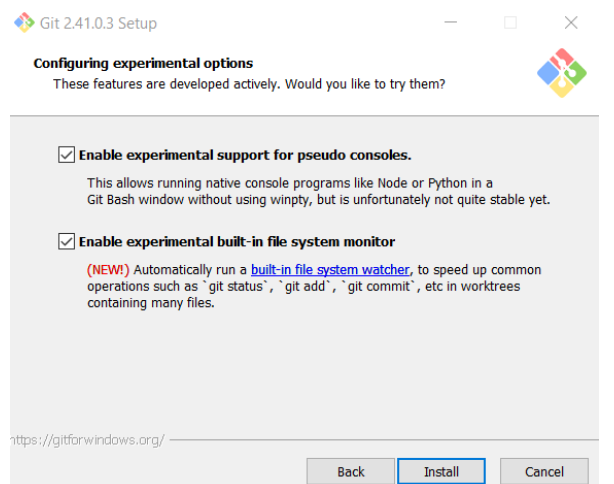
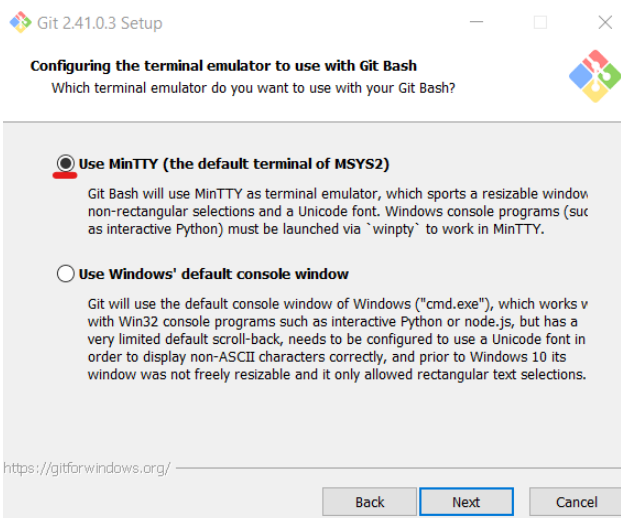
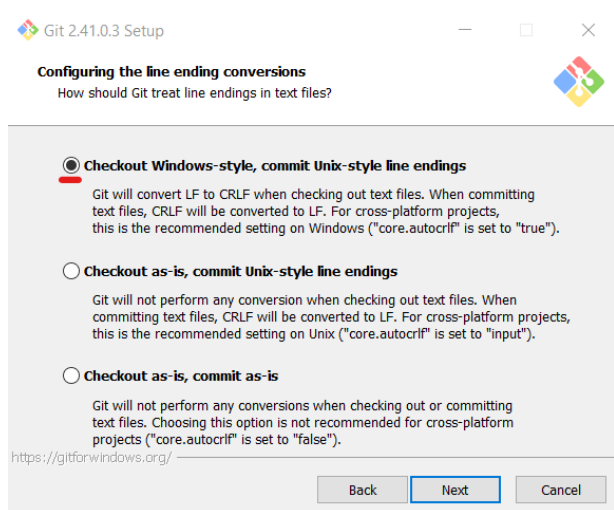
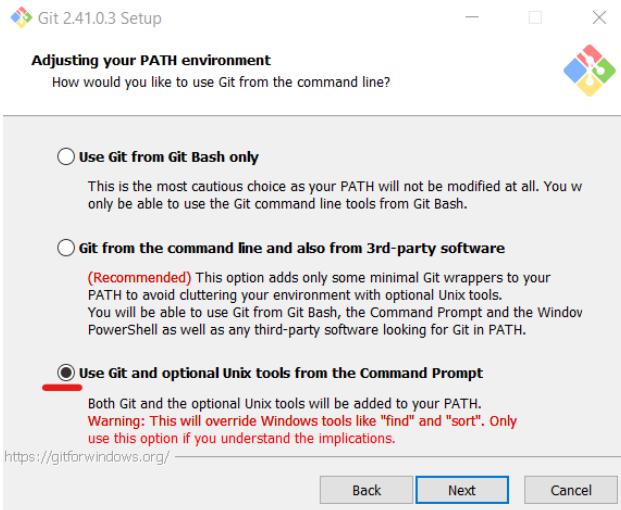
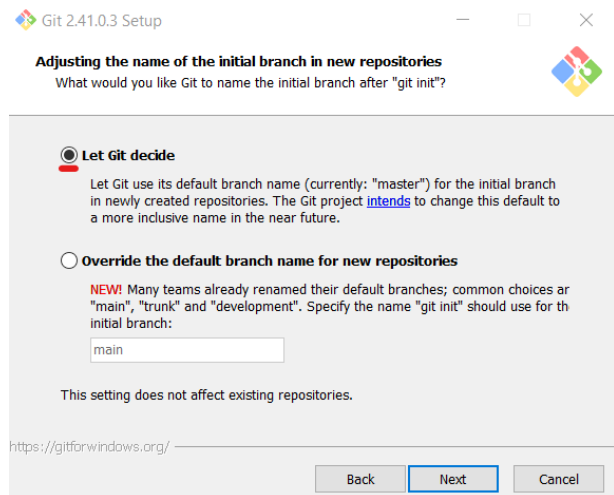
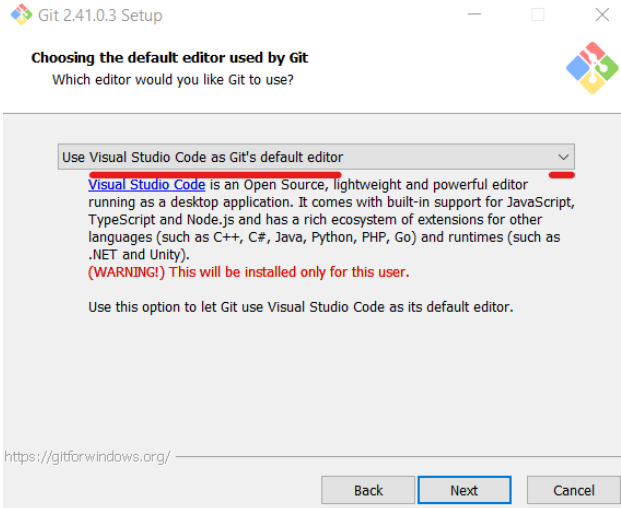
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☒ Additional icons
 - ☒ On the Desktop
- ☒ Windows Explorer integration
 - ☒ Open Git Bash here
 - ☒ Open Git GUI here
- ☒ Git LFS (Large File Support)
- ☒ Associate .git* configuration files with the default text editor
- ☒ Associate .sh files to be run with Bash
- ☐ Check daily for Git for Windows updates
- ☐ (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 319,4 MB of disk space.

<https://gitforwindows.org/>

Back Next Cancel



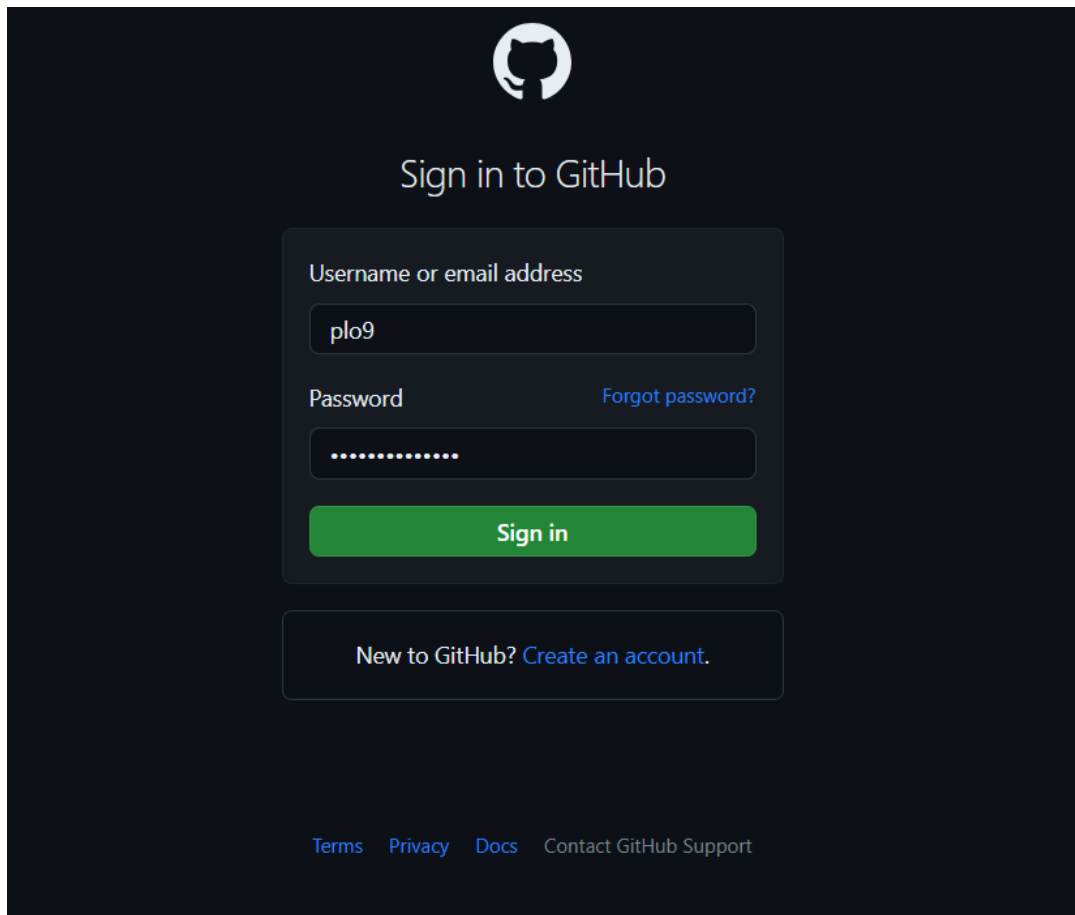
IV. Mise en place de l'environnement de travail et la découverte des commandes.

Scenario : Nous allons créer un dossier où sera contenu notre projet, y ajouter du code, faire des modifications avec différentes versions du projet et les envoyer en ligne sur GitHub.

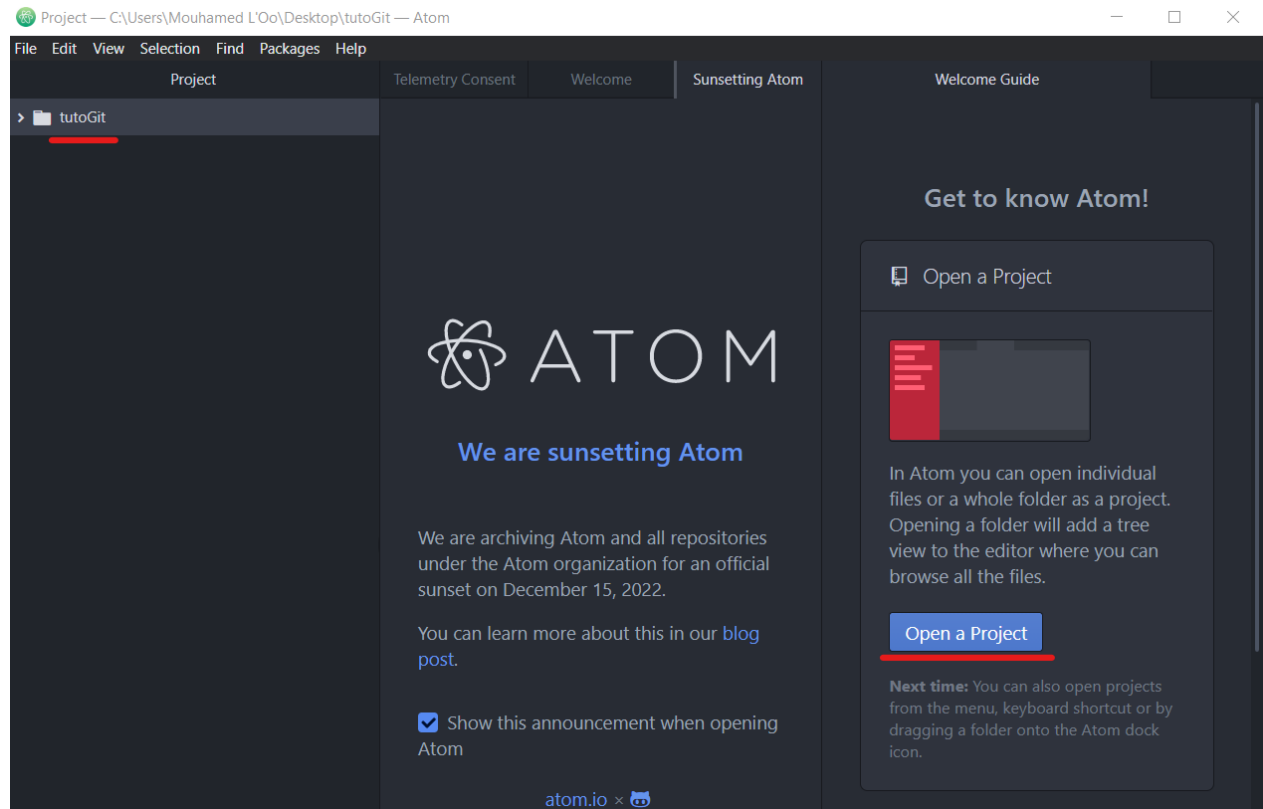
Nous apprendrons les commandes de bases de git qui nous permettront de mener à bien notre projet.

Si vous n'avez pas de compte GitHub vous pouvez en créer un rapidement sur la plateforme.

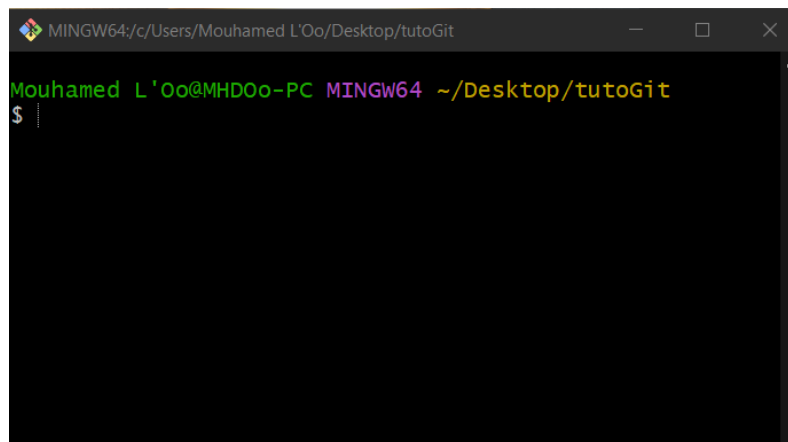
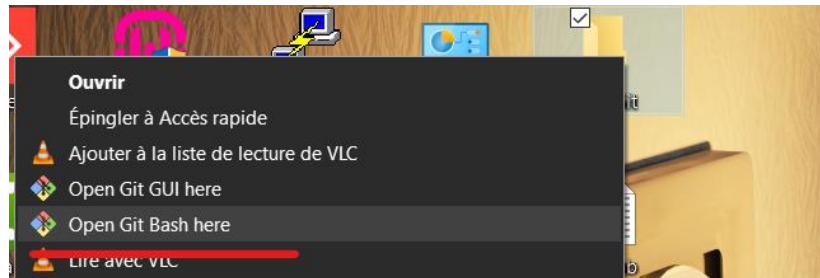
A notre niveau nous avons déjà un compte. Voir ci-dessous :



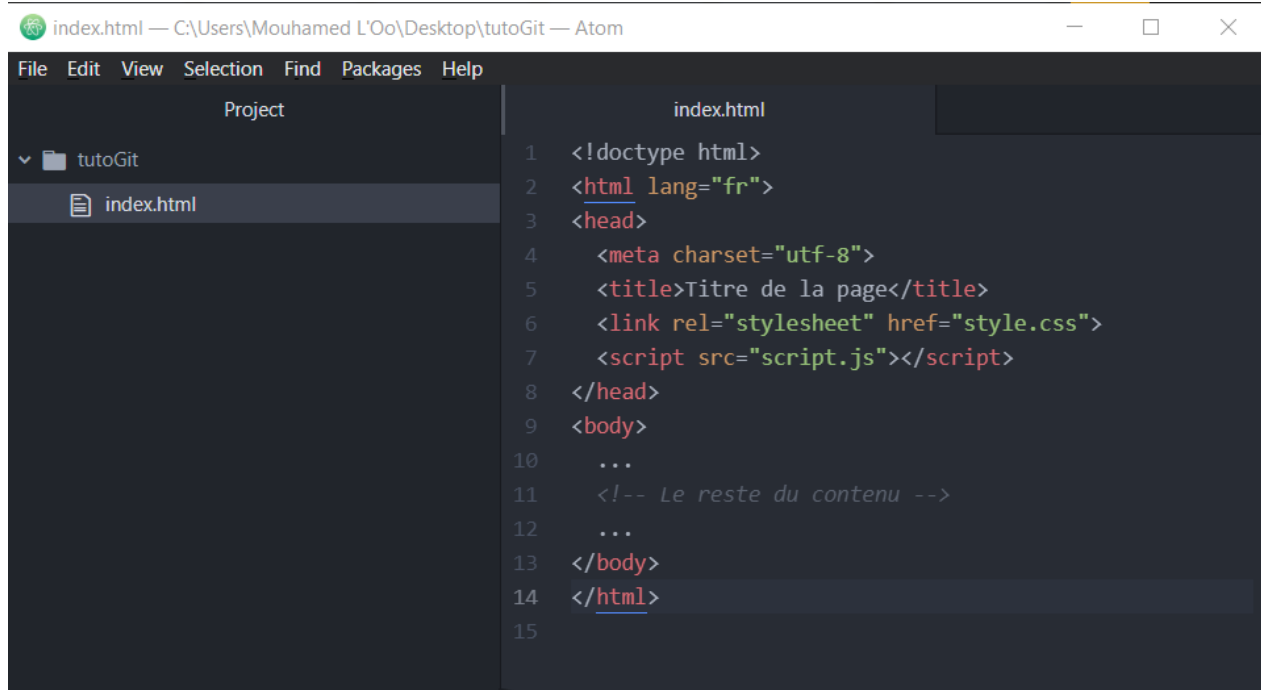
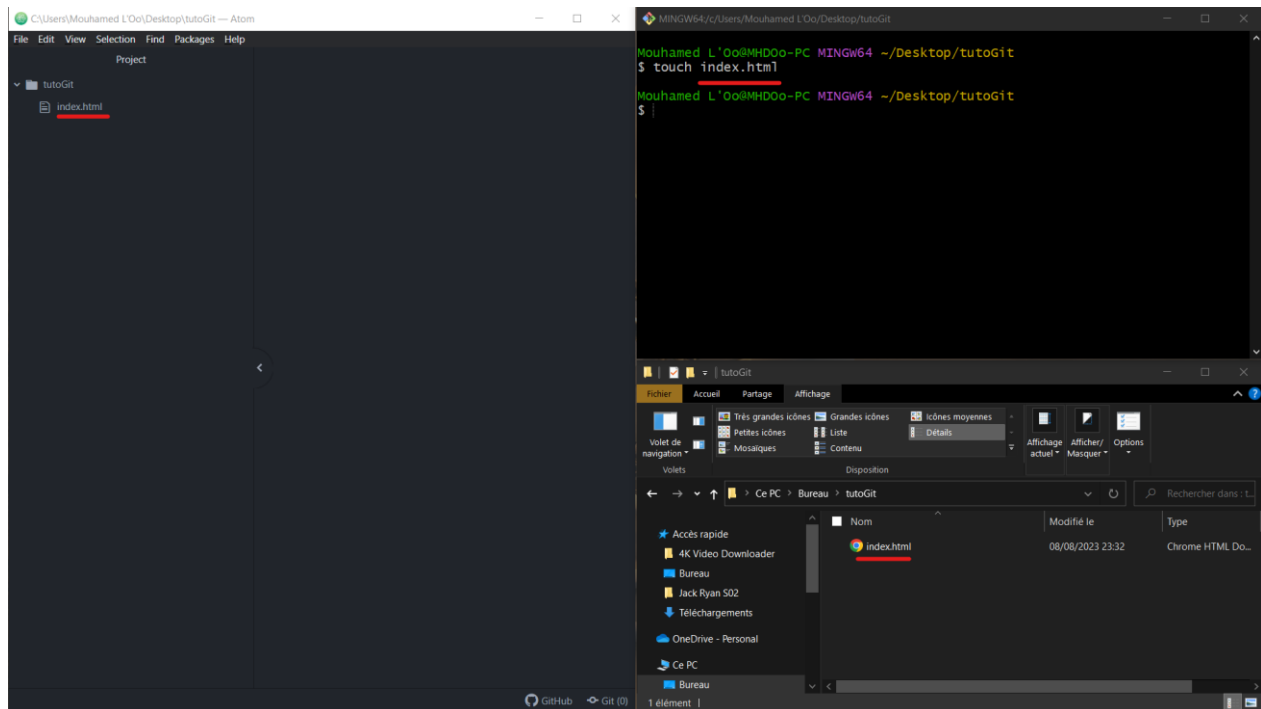
Nous allons créer notre répertoire de projet sur le bureau avec comme nom **tutoGit** puis nous allons l'ouvrir sur notre éditeur de texte **Atom**.



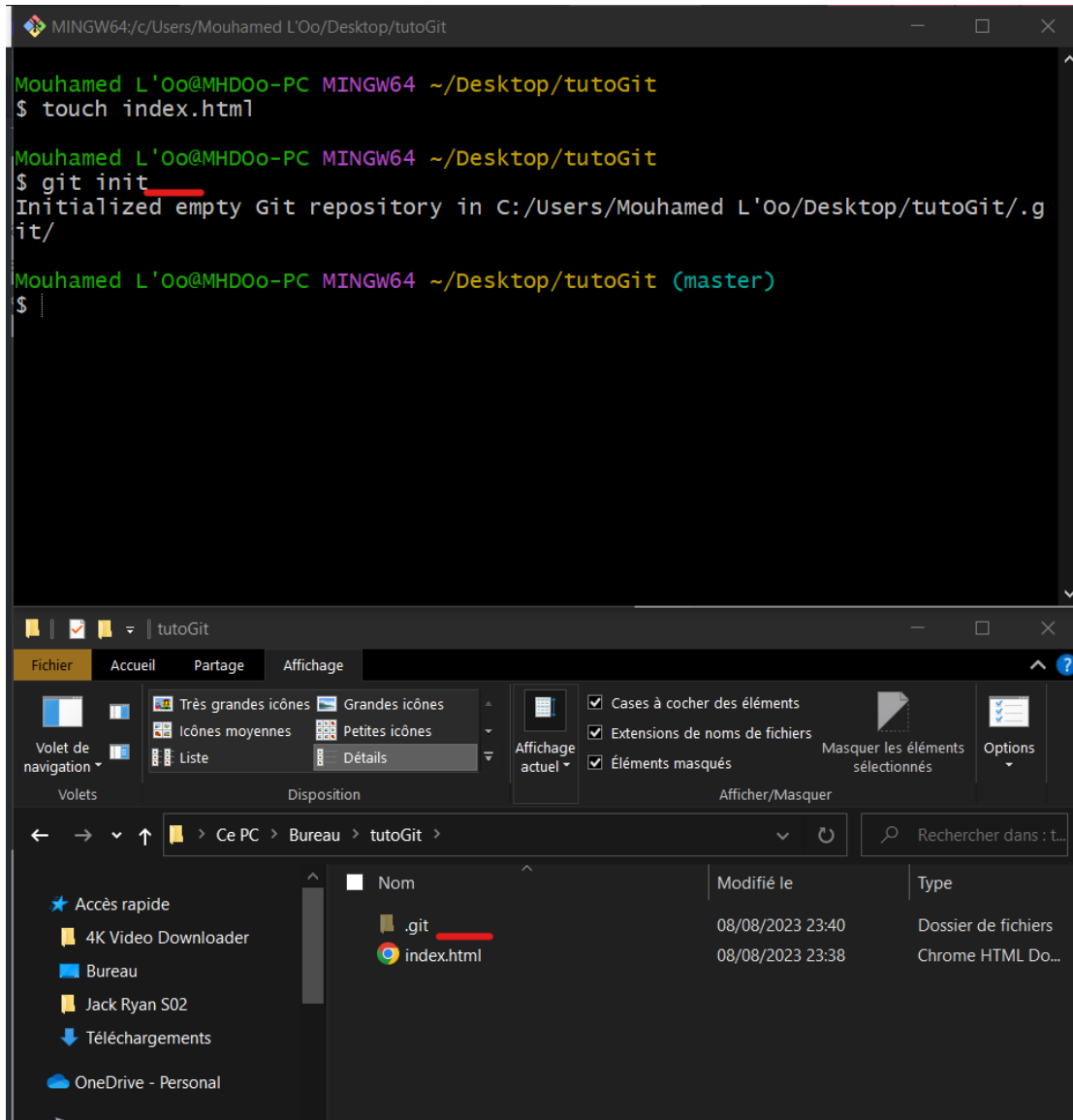
Ouvrons maintenant le dossier dans *Git bash*, là où se fera les commandes.



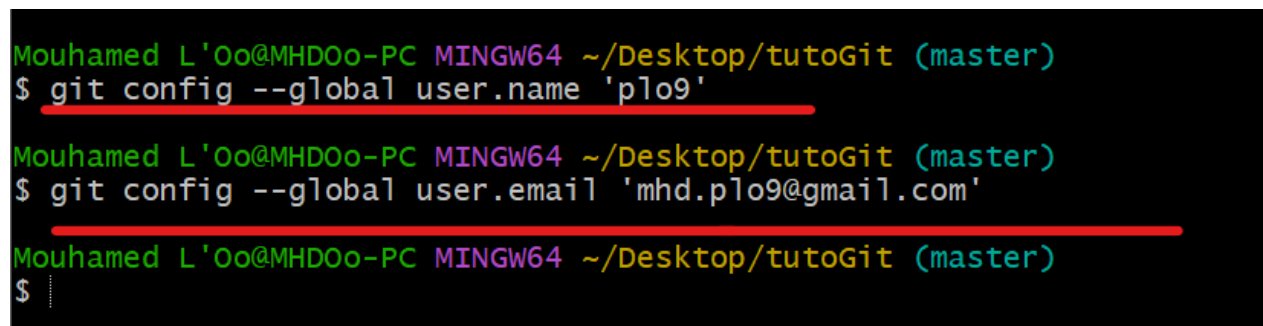
Nous allons créer un fichier `index.html` où on mettra un bout de code en html. Le fichier est créé sur git bash avec la commande ***touch*** et on peut remarquer que ça se crée automatiquement dans le dossier `tutoGit`.



Nous allons initialiser notre code sur git avec la commande `git init`



Nous allons à présent nous identifier sur git bash avec la commande `git config`. Ici j'ai utilisé les mêmes identifiants que sur GitHub.

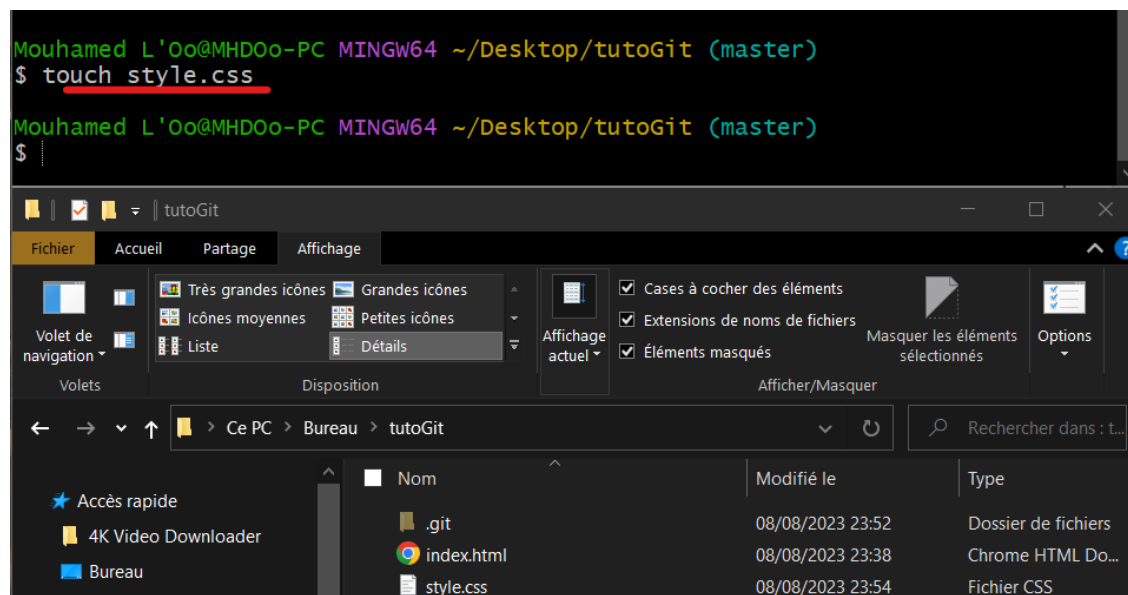


Maintenant que git bash nous a identifié nous pouvons utiliser toutes les commandes. On ajoute notre fichier *index.html* à la liste des fichiers à sauvegarder (ou au pré sauvegarde) avec la commande ***git add***.

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git add index.html

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Nous allons créer un nouveau fichier *style.css*



La commande ***git add .*** permet d'ajouter tous les fichiers présents dans un dossier

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git add .

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

On peut voir à l'aide de la commande `git status` la liste des fichiers disponible au pré sauvegarde. Nous allons maintenant les sauvegarder avec la commande ***git commit***.

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git status
On branch master

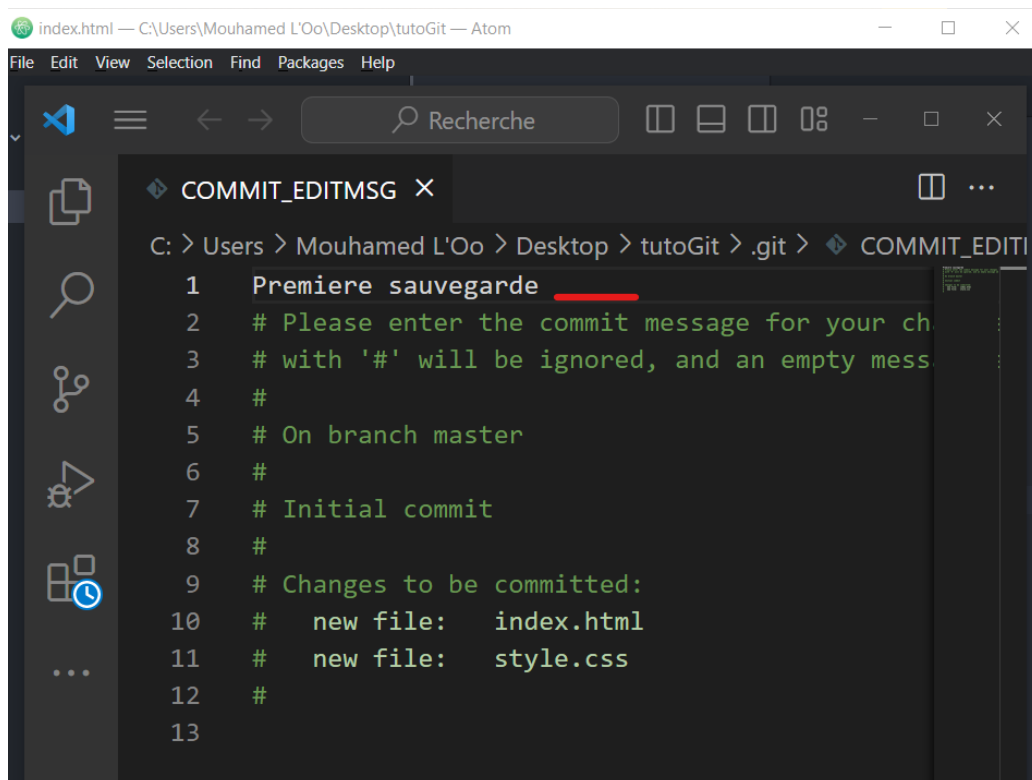
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html
    new file:   style.css

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Quand la commande **git commit** est exécutée, un éditeur de texte s'ouvre et nous demande de commenter la sauvegarde de façon à avoir une traçabilité. Il peut s'agir de l'éditeur vim ou n'importe quel autre éditeur défini lors de l'installation de git bash. Dans notre cas c'est *VS code*.

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git commit
hint: Waiting for your editor to close the file... |
```



```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (master)
$ git commit
[master (root-commit) a7d965a] Premiere sauvegarde
2 files changed, 14 insertions(+)
create mode 100644 index.html
create mode 100644 style.css

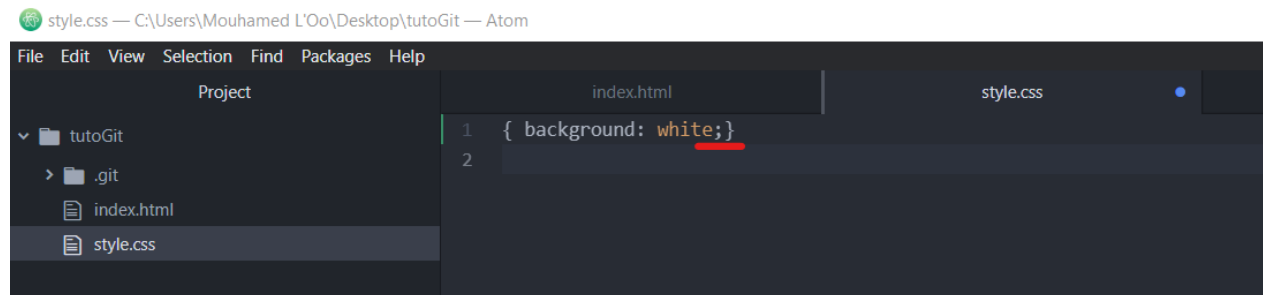
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Et si on refait un *git status* on verra que les fichiers ne sont plus à l'étape de pré sauvegarde mais plutôt sauvegardés.

```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (master)
$ git status
On branch master
nothing to commit, working tree clean

Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Là, nous allons modifier notre fichier css en y ajoutant une ligne de code et voir ce qui se passe.



On effectue un *git status* et on nous signale que le fichier style.css a été modifié et qu'il faut l'ajouter à nouveau.

On effectue à nouveau un *git add* et cette fois ci avec comme commande *git add *.css* pour n'ajouter que le fichier css.

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git status
On branch 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:   style.css
no changes added to commit (use "git add" and/or "git commit -a")

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Et maintenant nous devons faire un deuxième commit mais cette fois ci nous voulons sauter l'étape où nous allons écrire notre commentaire sur l'éditeur de texte. On mettra donc le commentaire au niveau de la commande : voir capture ci-dessous

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git add *.css

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git commit -m 'changement fichier css'
[master c772275] changement fichier css
1 file changed, 1 insertion(+)

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Comme expliquer plus haut il est possible de faire du développement en parallèle. Cela se fait avec les branches.

Git status nous montre que nous sommes sur la branche **master** et nous voulons créer un fichier JavaScript sur une branche à part puis lier le fichier html et le fichier JavaScript

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git branch FichierJS

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$
```

Et pour changer de branche on utilise la commande *git checkout*

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (master)
$ git checkout FichierJS
Switched to branch 'FichierJS'

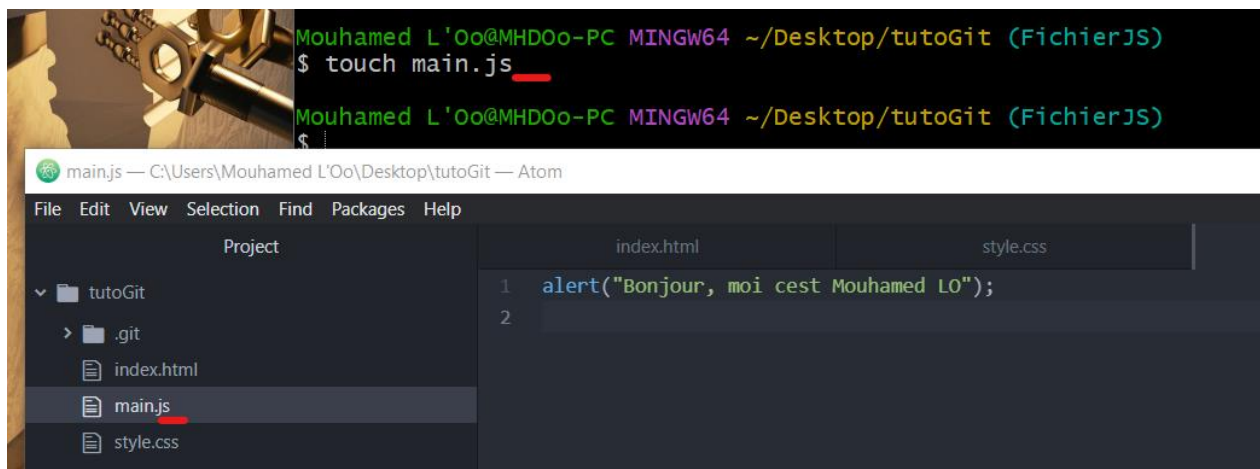
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$
```

La commande *git branch --list* nous permet de voir la liste des branches existantes.

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$ git branch --list
* FichierJS
  master

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$
```

On crée ensuite un fichier *main.js* et on y insère une ligne de code en JavaScript.



Nous allons ensuite le référencer dans notre fichier html à l'aide des balises scripts.

```
index.html  style.css
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="utf-8">
5    <title>Titre de la page</title>
6    <link rel="stylesheet" href="style.css">
7  </head>
8  <body>
9    <script src="main.js"></script>
10 </body>
11 </html>
12
```

On ajoute à nouveau tout notre dossier avec **git add .** puis on fait un commit avec comme commentaire : Ajout du fichier javascript

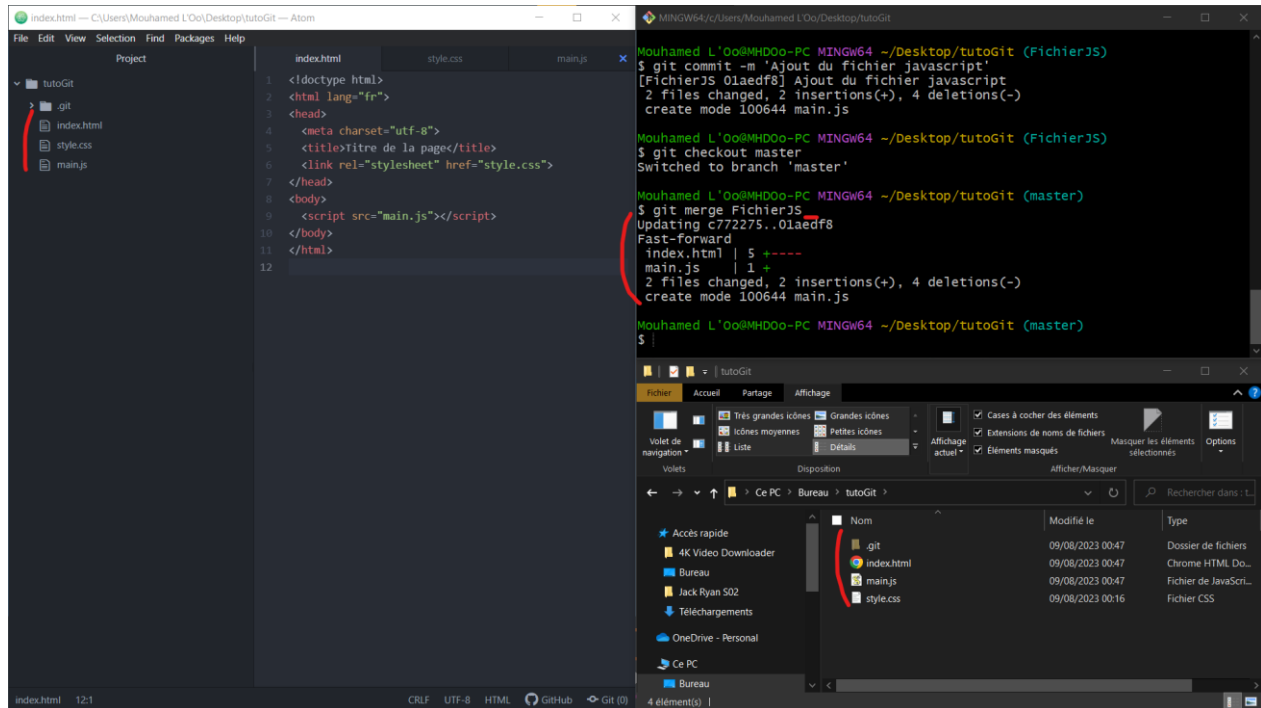
```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$ git add .

Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$ git commit -m 'Ajout du fichier javascript'
[FichierJS 01aedf8] Ajout du fichier javascript
 2 files changed, 2 insertions(+), 4 deletions(-)
 create mode 100644 main.js

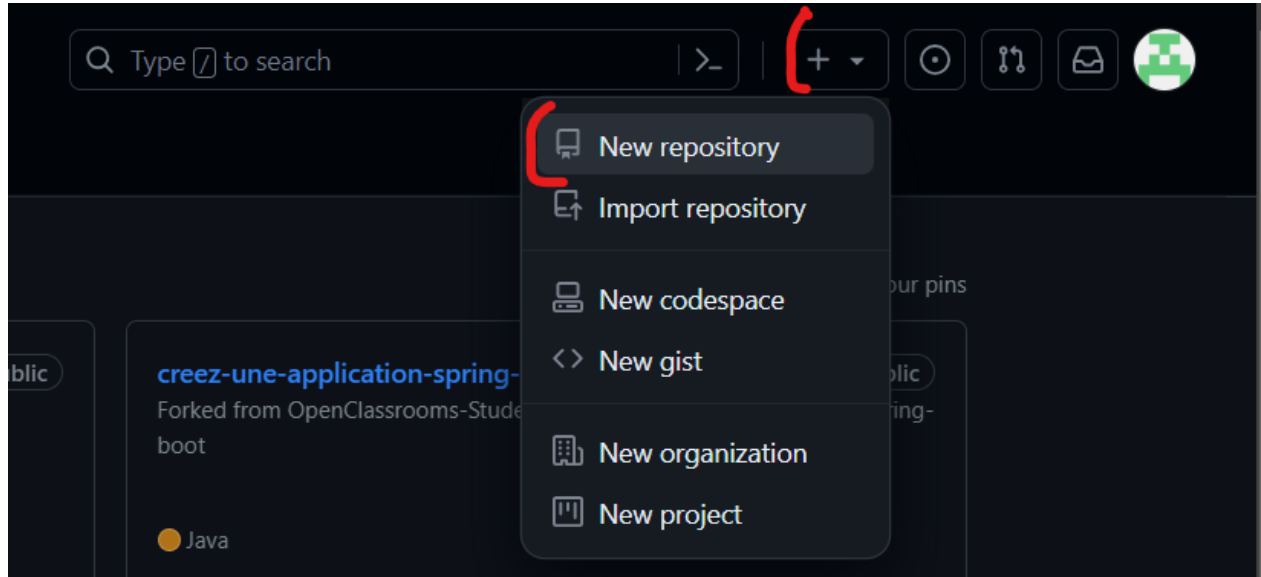
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (FichierJS)
$
```

Nous allons maintenant revenir sur la branche master et on remarquera que les modifications fait au niveau de la branche FichierJS sont disparus, on parle alors de développement en parallèle.

Maintenant que nous avons fini de faire nos changements on fusionne nos 2 branches avec la commande **git merge**



Nous allons maintenant sur GitHub et créer un repository ou répertoire qui contiendra notre projet.

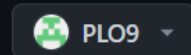


Create a new repository

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

Required fields are marked with an asterisk ().*

Owner *



Repository name *

tutoGit

✔ tuto is available.

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

Description (optional)

Pour la présentation de git et github



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:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template:None ▾

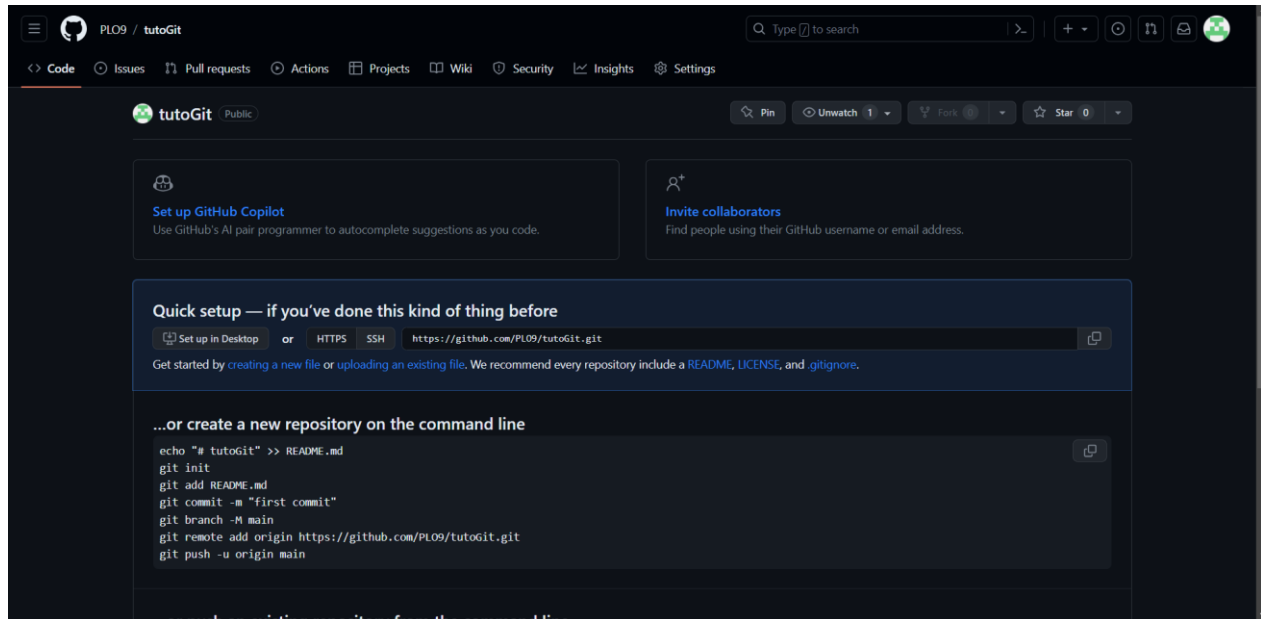
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License:None ▾

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

On donne un nom au répertoire et une description



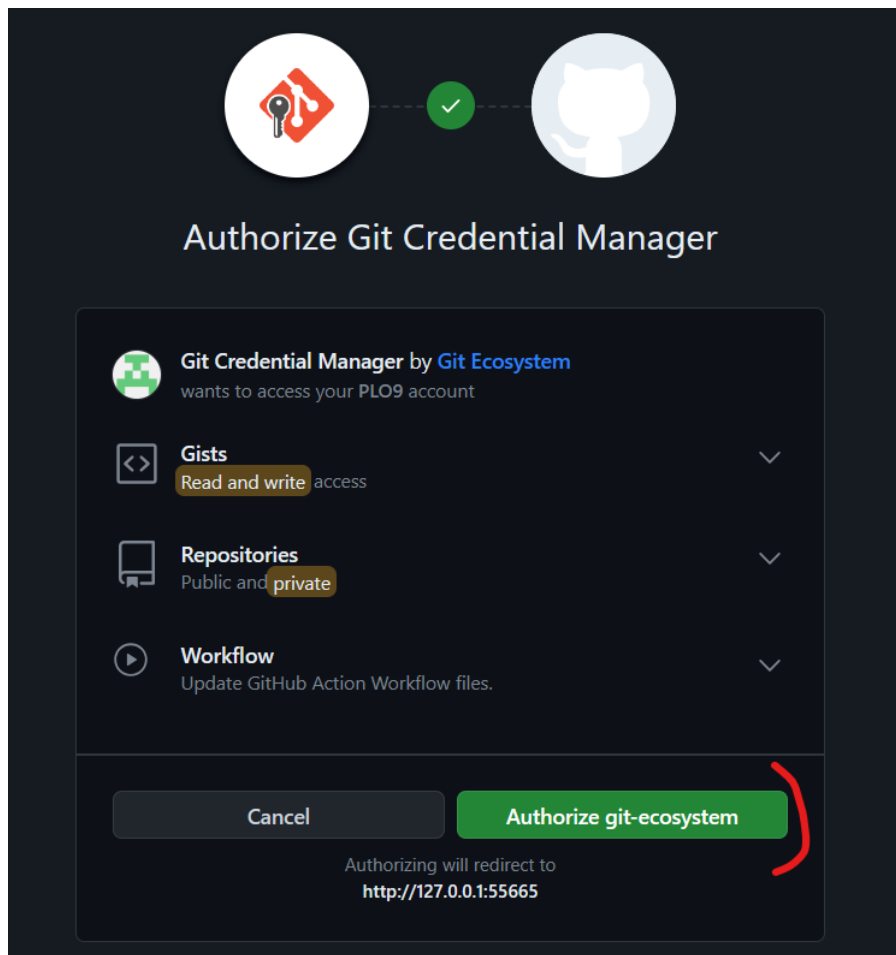
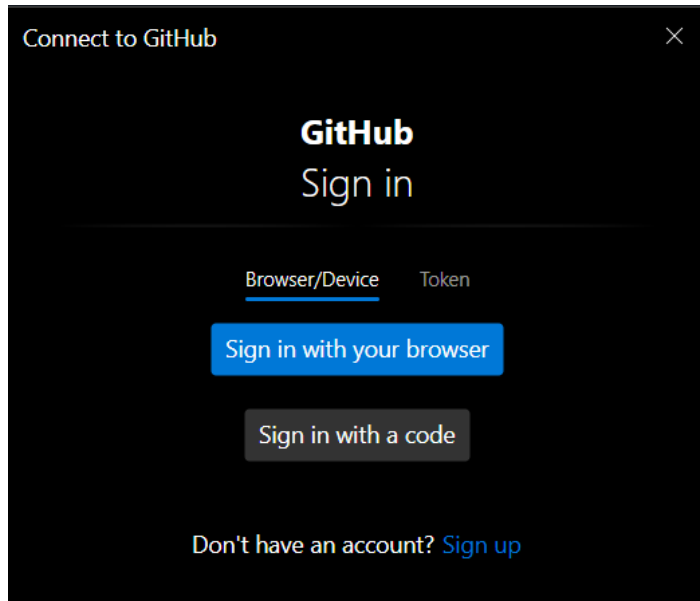
A ce niveau git nous explique comment ajouter notre premier fichier à notre repo ce qui n'est pas notre cas car on a déjà un projet existant donc on choisit la deuxième option qui nous explique comment push notre dossier existant



On copie les trois lignes et on colle sur notre git bash.



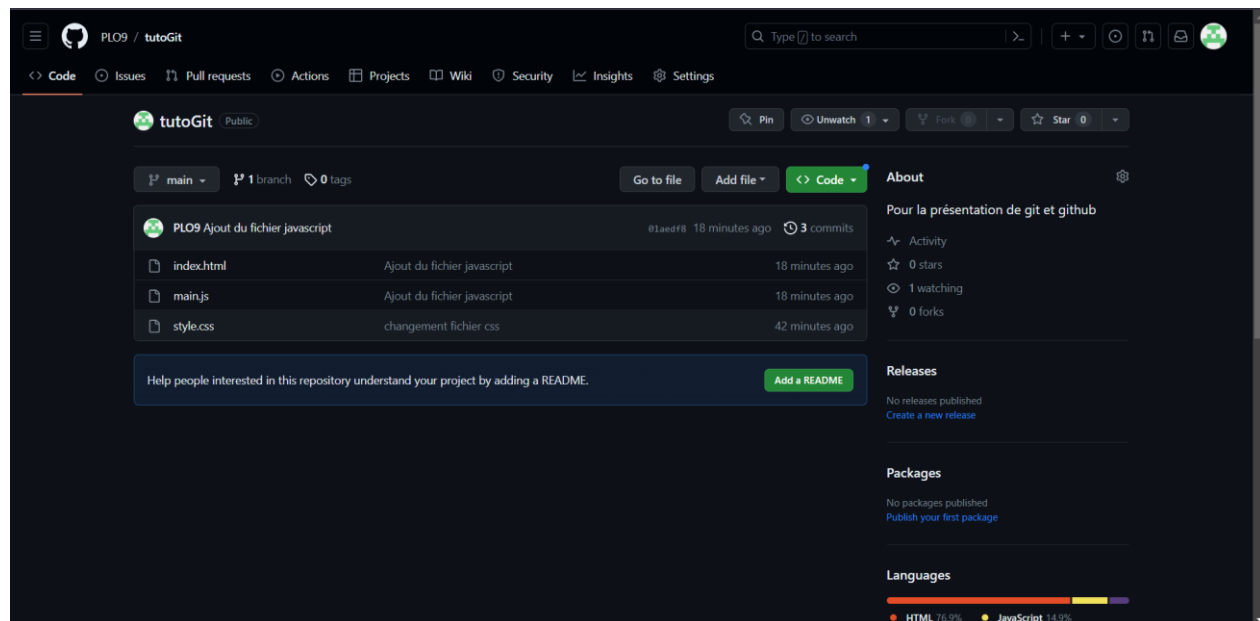
Si c'est notre première connexion git nous demandera de mettre nos identifiants.



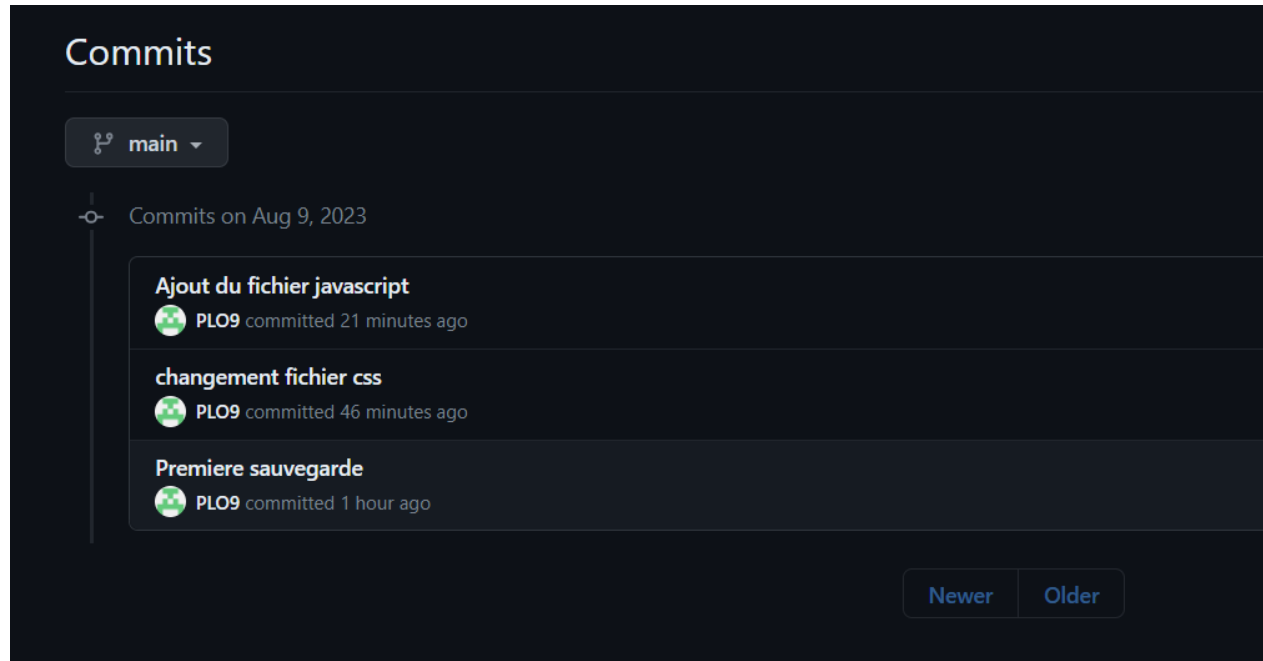
```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (main)
$ git push -u origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (11/11), 1.06 KiB | 542.00 KiB/s, done.
Total 11 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/PL09/tutoGit.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (main)
$
```

On revient sur GitHub, rafraichissons la page et nous verrons que notre projet a été bien importé.



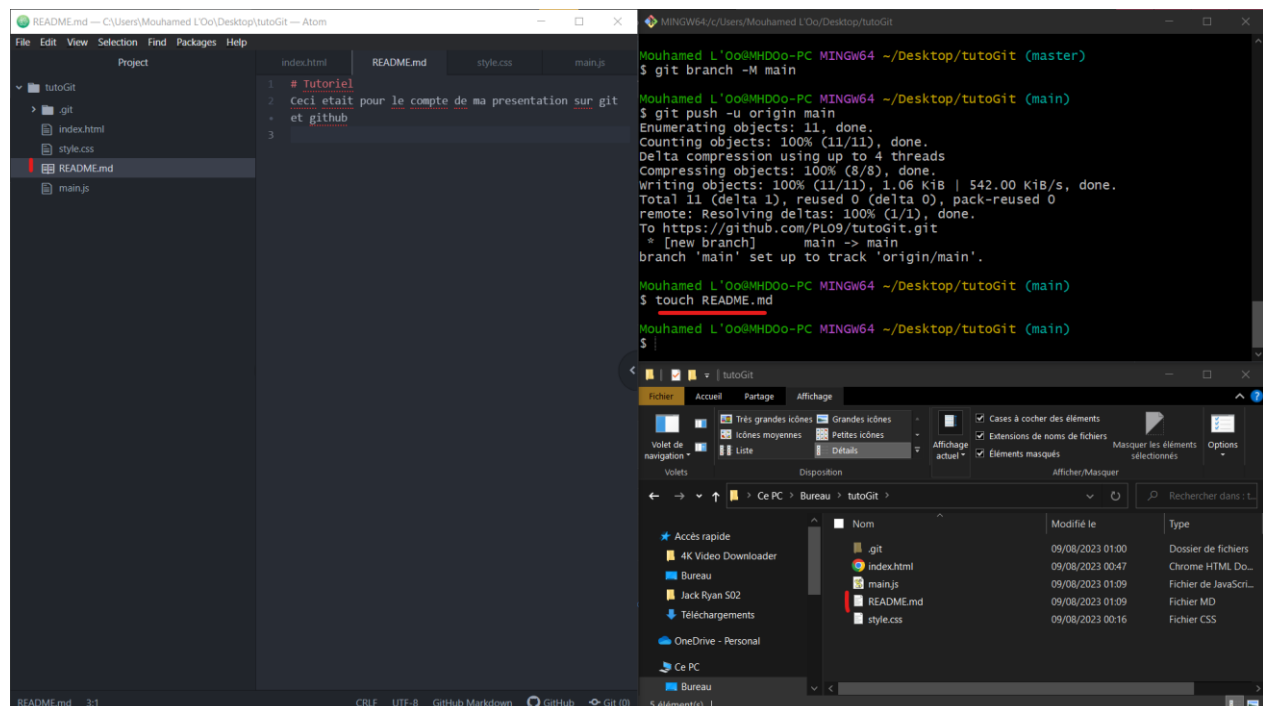
Sur commit on peut récupérer le code à n'importe quelle version.



Sur github on voit souvent que certains repository ont une description en dessous. Cela est possible avec le fichier README.

On crée le fichier README avec la commande *touch* et avec l'extension *.md* sinon cela s'affichera pas correctement sur github.

Ajoutons ensuite la phrase de description avec notre éditeur de texte.



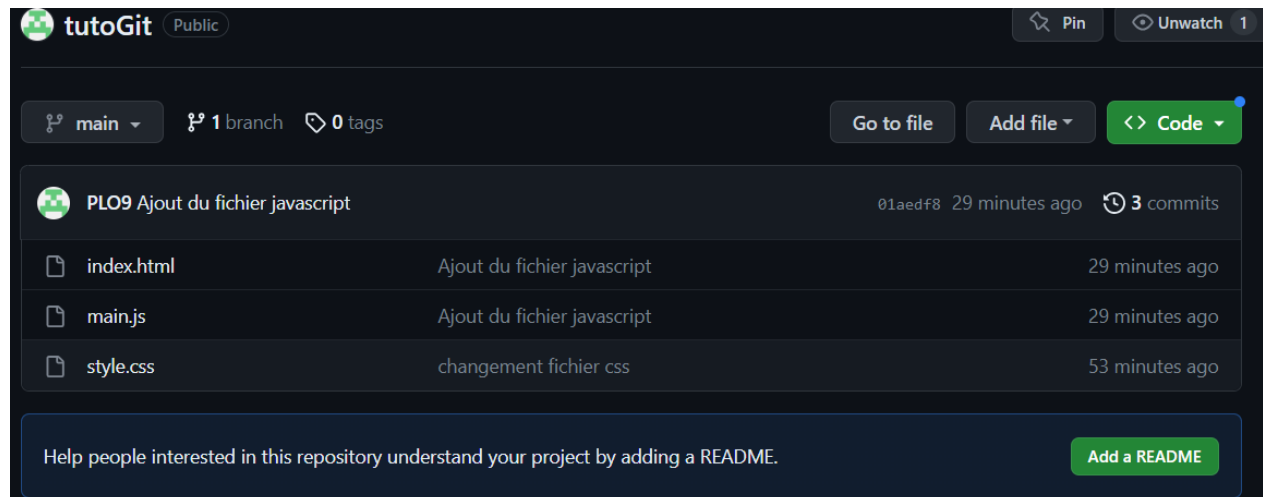
On ajoute ensuite notre README avec un git add puis git commit

```
Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (main)
$ git add .

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (main)
$ git commit -m 'Ajout du fichier readme'
[main 87d66e6] Ajout du fichier readme
1 file changed, 2 insertions(+)
create mode 100644 README.md

Mouhamed L'Oo@MHDOo-PC MINGW64 ~/Desktop/tutoGit (main)
$
```

En rafraichissant la page on voit rien, ce qui est normal.



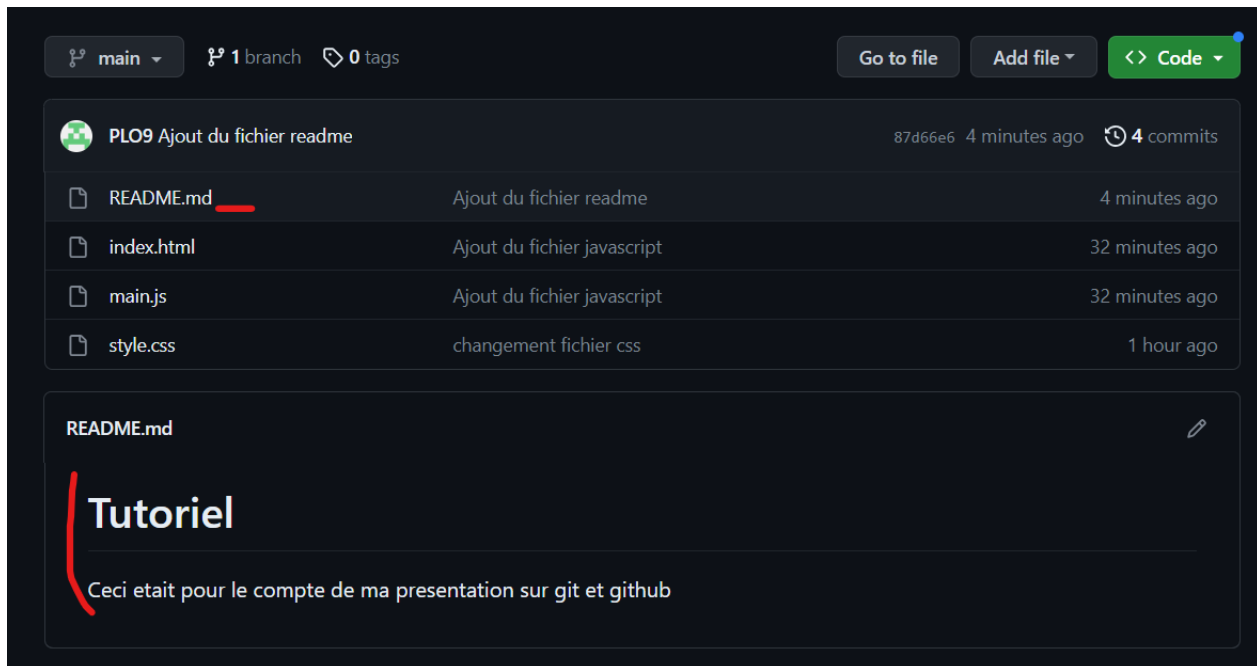
Il faut le pousser avec git push

Cela nous permet de mettre tous les changements sur notre repository de github.

```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 408 bytes | 408.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/PL09/tutoGit.git
01aedef8..87d66e6  main -> main

Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (main)
$
```

Après rafraichissement on peut maintenant voir



The screenshot shows the GitHub interface for a repository named 'PL09 Ajout du fichier readme'. At the top, there are buttons for 'Go to file', 'Add file', and 'Code'. Below this, a table lists the files in the repository:

File	Commit Message	Time
README.md	Ajout du fichier readme	4 minutes ago
index.html	Ajout du fichier javascript	32 minutes ago
main.js	Ajout du fichier javascript	32 minutes ago
style.css	changement fichier css	1 hour ago

Below the file list, the content of the selected 'README.md' file is shown. It features a red bracket on the left side of the text:

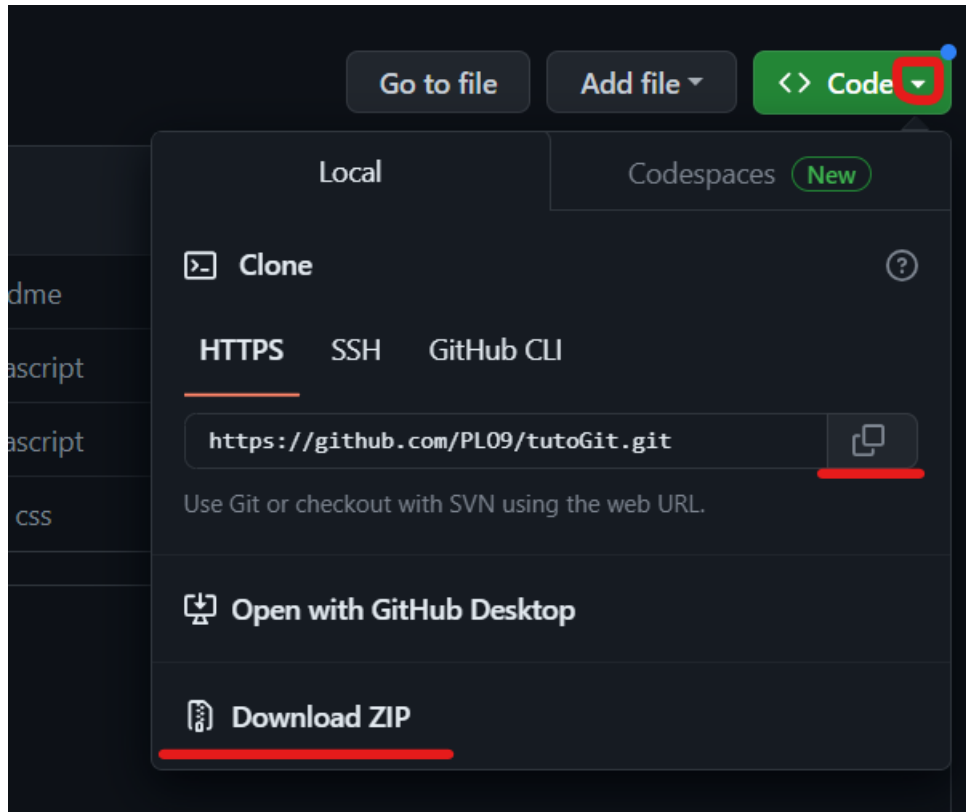
```
Tutoriel
Ceci etait pour le compte de ma presentation sur git et github
```

En cas de changement par l'un des membres de l'équipe.

```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (main)
$ git pull
Already up to date.

Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (main)
$
```


Un dernier cas de figure c'est qu'en cas de perte ou de suppression du dossier du projet on pourra le récupérer à travers GitHub sur un fichier Zippé



```
Mouhamed L'Oo@MHD0o-PC MINGW64 ~/Desktop/tutoGit (main)
$ git clone https://github.com/PL09/tutoGit.git
```

Conclusion

GitHub se révèle être bien plus qu'une simple plateforme de gestion de versions. C'est un écosystème dynamique qui favorise la collaboration, l'innovation et la création collective. Grâce à ses fonctionnalités de suivi des modifications, de gestion des problèmes et de partage de code, GitHub facilite le travail d'équipes dispersées géographiquement et permet aux développeurs du monde entier de collaborer efficacement sur des projets communs. Que vous soyez un débutant désireux d'apprendre ou un professionnel chevronné cherchant à accélérer le développement, GitHub offre les outils nécessaires pour transformer des lignes de code en réalisations exceptionnelles.