# SOMMAIRE

[SOMMAIRE 1](#_Toc155460663)

[INTRODUCTION GENERALE 2](#_Toc155460664)

[I. Historique 3](#_Toc155460665)

[II. Définition et rôles 4](#_Toc155460666)

[III. Fonctionnement 6](#_Toc155460667)

[IV. Typologie 10](#_Toc155460668)

[V. Sécurité 14](#_Toc155460669)

[VI. Avantages et inconvénients 16](#_Toc155460670)

[VII. Domaines d’application 20](#_Toc155460671)

[VIII. Implémentation 22](#_Toc155460672)

[Conclusion GENeRALE 35](#_Toc155460673)

[Glossaire 36](#_Toc155460674)

[TABLE DES MATIERES 38](#_Toc155460675)

# INTRODUCTION GENERALE

Le développement de logiciels est devenu une activité complexe et collaborative. La gestion des versions, le suivi des modifications et la collaboration entre développeurs sont devenus essentiels pour garantir l'efficacité et la qualité des projets. C'est dans ce contexte que **Git** et **GitHub**, deux outils de gestion de code source, de gestion de version très populaires qui jouent un rôle clé dans la collaboration, le suivi des modifications et la sauvegarde du code. Cependant, dans cet exposé, pour comprendre la quintessence de ces outils, nous allons examiner en détail **Git** et **GitHub**, en expliquant leur définition, leurs rôles, leur fonctionnement, leur sécurité, leurs avantages et inconvénients, ainsi que leurs domaines d'application.

# Historique

L'histoire de Git remonte à 2005, lorsque **Linus Torvalds**, le créateur du noyau Linux, a ressenti le besoin d'un meilleur système de contrôle de version pour gérer le développement du noyau Linux. À l'époque, les systèmes de contrôle de version centralisés étaient couramment utilisés, mais ils présentaient des limitations en termes de performances et de fonctionnalités pour un projet de la taille et de la complexité du noyau Linux. Pour résoudre ces problèmes, **Linus Torvalds** a développé Git en s'inspirant de certains aspects du système de contrôle de version BitKeeper, qu'il utilisait auparavant. Git a été conçu avec une approche distribuée, permettant à chaque développeur d'avoir une copie complète de l'historique du code source sur sa machine locale. La première version publique de Git a été publiée en avril 2005. Rapidement, Git est devenu un outil populaire au sein de la communauté du développement logiciel, en particulier pour les projets open source. Sa flexibilité, sa rapidité et sa capacité à gérer des projets de grande envergure ont été largement appréciées par les développeurs.

En 2008, GitHub a été créé par **Tom Preston-Werner**, **Chris Wanstrath** et **PJ Hyett**. GitHub est une plateforme web basée sur Git qui offre un espace de stockage et de partage pour les dépôts Git. Elle a été conçue pour faciliter la collaboration entre les développeurs, en offrant des fonctionnalités telles que le suivi des problèmes, la gestion de projet et la revue de code. GitHub a rapidement gagné en popularité et est devenu un point central de collaboration pour les projets open source. De nombreux projets et communautés open source ont choisi GitHub comme plateforme pour héberger leurs dépôts Git et faciliter la contribution des développeurs du monde entier.



Figure 1 : Logo de GitHub

En 2018, Microsoft a annoncé l'acquisition de GitHub, suscitant des inquiétudes et des interrogations au sein de la communauté open source. Cependant, **Microsoft** a maintenu GitHub en tant qu'entité indépendante et a continué à investir dans son développement et son amélioration.

Depuis lors, Git et GitHub ont continué à évoluer et à s'améliorer. Des fonctionnalités supplémentaires ont été ajoutées pour faciliter la collaboration entre les développeurs, la gestion des projets et l'intégration avec d'autres outils et services de développement logiciel.

Aujourd'hui, Git est devenu un outil de contrôle de version incontournable, utilisé par des millions de développeurs à travers le monde. GitHub est devenu la plateforme de référence pour l'hébergement de dépôts Git, qu'il s'agisse de projets open source, de projets personnels ou de projets d'entreprise. L'histoire de Git et GitHub témoigne du besoin croissant d'outils de gestion de code source performants et de plateformes de collaboration pour soutenir le développement logiciel moderne.

# Définition et rôles



Figure 2 : Logos respectifs de Git et de GitHub

**Git** est un système de contrôle de version distribué, conçu pour gérer efficacement le développement logiciel. Il se distingue des systèmes de contrôle de version centralisés traditionnels en permettant à chaque développeur d'avoir une copie complète de l'historique du code source sur sa propre machine. Cela signifie que les développeurs peuvent travailler de manière indépendante, même sans connexion réseau, et fusionner leurs modifications ultérieurement. Le rôle principal de Git est de suivre les modifications apportées au code source au fil du temps. Chaque modification est enregistrée sous forme de "**commit**" dans l'historique de Git. Ces commits enregistrent les modifications spécifiques apportées aux fichiers, tels que l'ajout de nouvelles fonctionnalités, la correction de bugs ou la suppression de code obsolète. En conservant un historique complet des commits, Git permet de revenir en arrière et de restaurer des versions antérieures du code si nécessaire. En plus de suivre les modifications, Git facilite également la collaboration entre les développeurs. Il permet à plusieurs développeurs de travailler simultanément sur différentes branches du code, ce qui est particulièrement utile lorsque des fonctionnalités distinctes sont développées en parallèle. Une fois que les travaux sont terminés, Git offre des outils pour fusionner les modifications des différentes branches, permettant ainsi de consolider le travail de l'équipe.

**GitHub**, quant à lui, est une plateforme basée sur Git qui étend ses fonctionnalités en ajoutant des outils de collaboration supplémentaires. Il fournit un espace de stockage et de partage pour les dépôts Git sur des serveurs distants, ce qui facilite la collaboration entre les développeurs travaillant sur un même projet. GitHub offre également des fonctionnalités de gestion de projet, de suivi des problèmes et de revue de code, ce qui en fait une plateforme complète pour la gestion du développement logiciel. Le rôle de GitHub est de faciliter la collaboration et la communication entre les membres de l'équipe. Les développeurs peuvent cloner les dépôts sur leur machine locale, effectuer des modifications, puis pousser ces modifications vers le dépôt distant. Ils peuvent également soumettre des demandes de pull pour proposer leurs modifications aux mainteneurs du projet, qui peuvent ensuite les examiner, les discuter et les fusionner si elles sont jugées pertinentes.

Git joue un rôle essentiel dans le suivi des modifications et la gestion des versions du code source, tandis que GitHub étend les fonctionnalités de Git en offrant des outils de collaboration, de gestion de projet et de revue de code. Ensemble, Git et GitHub permettent aux développeurs de travailler de manière efficace, de partager leur code, de contribuer à des projets open source et de collaborer de manière transparente avec d'autres membres de l'équipe.

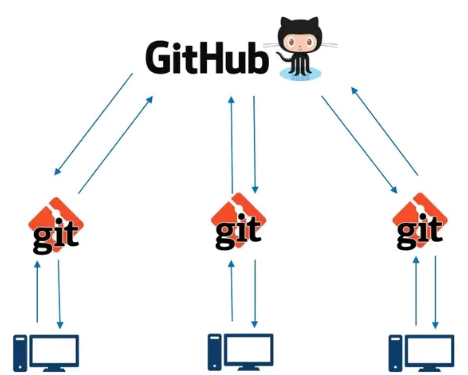


Figure 3 : Relation entre Git et GitHub

# Fonctionnement

Le fonctionnement de Git repose sur plusieurs concepts clés qui permettent de suivre les modifications apportées au code source et de faciliter la collaboration entre les développeurs. Les principaux éléments du fonctionnement de Git :

1. **Dépôt (repository)**

Un dépôt Git est un espace de stockage qui contient l'ensemble des fichiers et de l'historique des modifications d'un projet. Il peut être local (sur la machine d'un développeur) ou distant (hébergé sur un serveur). Chaque développeur peut cloner un dépôt pour obtenir une copie complète de l'historique sur sa machine locale.

1. **Commit**

Un commit représente une modification spécifique apportée à un ou plusieurs fichiers du projet. Chaque commit enregistre un instantané (snapshot) de l'état du code à un moment donné, ainsi qu'un message décrivant les modifications effectuées. Les commits forment l'historique du projet et permettent de revenir en arrière ou de récupérer des versions antérieures du code.

1. **Branches**

Les branches sont des lignes de développement indépendantes dans un dépôt Git. Elles permettent aux développeurs de travailler sur différentes fonctionnalités ou corrections de bugs en parallèle, sans perturber le code principal. Les branches peuvent être créées à partir d'une branche existante (généralement la branche principale appelée "master" ou "main"), et les modifications apportées sur une branche peuvent être fusionnées ultérieurement dans une autre branche.

1. **Fusion (merge)**

La fusion est le processus de combinaison des modifications provenant de différentes branches dans une branche commune. Lorsqu'une fonctionnalité ou une correction de bug est terminée sur une branche, les développeurs peuvent fusionner ces changements dans une autre branche, généralement la branche principale. Git analyse les modifications apportées et tente d'incorporer les modifications de manière automatique, mais peut demander une intervention manuelle en cas de conflits.

1. **Pousser (push) et tirer (pull)**

Lorsqu'un développeur a effectué des commits sur sa branche locale, il peut pousser ces changements vers le dépôt distant pour les partager avec les autres développeurs. Le push met à jour l'historique du dépôt distant avec les nouveaux commits. Inversement, lorsqu'un autre développeur a effectué des modifications sur le dépôt distant, les développeurs peuvent tirer (pull) ces modifications pour les intégrer à leur branche locale.

1. **Demandes de pull (pull requests)**

Les demandes de pull sont un mécanisme de collaboration dans lequel un développeur propose ses modifications à d'autres membres de l'équipe pour examen et fusion. Un développeur crée une demande de pull pour une branche spécifique, puis les autres développeurs peuvent examiner les modifications, commenter le code et demander des modifications supplémentaires si nécessaire. Une fois que les modifications sont approuvées, elles peuvent être fusionnées dans la branche cible.

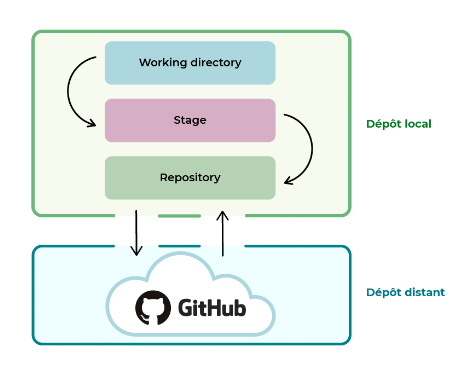


Figure 4 : Fonctionnement de GitHub

***Quelques commandes git :***

Voici quelques commandes Git couramment utilisées avec une brève explication de leur fonction :

1. **`git init`**

Cette commande initialise un nouveau dépôt Git dans le répertoire de travail actuel. Elle crée un répertoire `. git` qui contient tous les fichiers nécessaires au fonctionnement de Git.

1. **`git clone <URL>`**

Cette commande permet de cloner un dépôt Git existant à partir d'une URL. Elle crée une copie locale complète du dépôt, y compris tout l'historique des commits.

1. **`git add <fichier>`**

Cette commande ajoute un fichier spécifique à l'index de Git, préparant ainsi les modifications pour le prochain commit. Si vous souhaitez ajouter tous les fichiers modifiés, vous pouvez utiliser `*git add* .` ou `*git add --all*`.

1. **`git commit -m "message de commit"`**

Cette commande crée un nouveau commit à partir des modifications ajoutées à l'index. Le message de commit fournit une description concise des modifications apportées.

1. **`git push`**

Cette commande envoie les commits locaux vers un dépôt distant. Elle met à jour l'historique du dépôt distant avec les derniers commits.

1. **`git pull`**

Cette commande récupère les commits du dépôt distant et les fusionne automatiquement avec la branche locale en cours. Elle est utilisée pour mettre à jour votre copie locale avec les dernières modifications du dépôt distant.

1. **`git branch`**

Cette commande affiche la liste des branches présentes dans le dépôt. L'option `-a` affiche également les branches distantes.

1. **`git checkout <branche>`**

Cette commande permet de passer à une autre branche. Elle met à jour l'espace de travail avec le contenu de la branche spécifiée.

1. **`git merge <branche>`**

Cette commande fusionne les modifications de la branche spécifiée dans la branche actuelle. Elle combine les modifications apportées sur les deux branches et crée un nouveau commit de fusion.

1. **`git status`**

Cette commande affiche l'état actuel du dépôt Git. Elle indique les fichiers modifiés, les fichiers en attente d'ajout (staging) et les conflits éventuels.

1. **`git log`**

Cette commande affiche l'historique des commits du dépôt. Elle montre les informations sur chaque commit, telles que l'auteur, la date, le message et l'identifiant.

1. **`git remote add <nom> <URL>`**

Cette commande ajoute un dépôt distant avec un nom spécifié. Cela permet de référencer plus facilement le dépôt distant lors des opérations de push et de pull ultérieures.

Ces commandes représentent seulement une petite partie des fonctionnalités de Git. Il existe de nombreuses autres commandes et options pour gérer l'historique du code, résoudre les conflits, créer des branches, etc. Vous pouvez consulter la documentation Git officielle pour en savoir plus sur l'utilisation de ces commandes et découvrir d'autres fonctionnalités avancées.

Les concepts fondamentaux de Git permettent aux développeurs de travailler efficacement, de suivre les modifications apportées au code source et de collaborer de manière transparente sur un même projet. L'utilisation de ces concepts nécessite une bonne compréhension de Git et de ses commandes associées, qui peuvent être exécutées en ligne de commande ou via des interfaces graphiques fournies par des outils comme GitHub, GitLab ou Bitbucket.

# Typologie

Dans le cadre de Git et GitHub, on peut identifier plusieurs typologies qui peuvent aider à classer les concepts et les fonctionnalités associées. Voici quelques typologies importantes :

## Typologie des commandes Git

* **Commandes de base**

Cela inclut des commandes telles que `git init`, `git clone`, `git add`, `git commit`, `git push`, `git pull`, `git branch`, `git checkout`, qui sont utilisées pour gérer les dépôts Git locaux et distants, effectuer des opérations de commit, de fusion, de clonage, etc.

* **Commandes de gestion des branches**

Cela inclut des commandes comme `git branch`, `git checkout`, `git merge`, `git rebase`, qui permettent de créer, de basculer entre les branches, de fusionner les branches et de réorganiser l'historique des commits.

* **Commandes de gestion des remotes**

Cela inclut des commandes telles que `git remote`, `git fetch`, `git pull`, `git push`, qui sont utilisées pour gérer les dépôts distants, récupérer les nouvelles modifications, envoyer les modifications locales vers le dépôt distant, etc.

* **Commandes de gestion des conflits**

Cela inclut des commandes telles que `git diff`, `git mergetool`, `git status`, qui aident à résoudre les conflits qui surviennent lors de la fusion des branches ou de la récupération de nouvelles modifications.

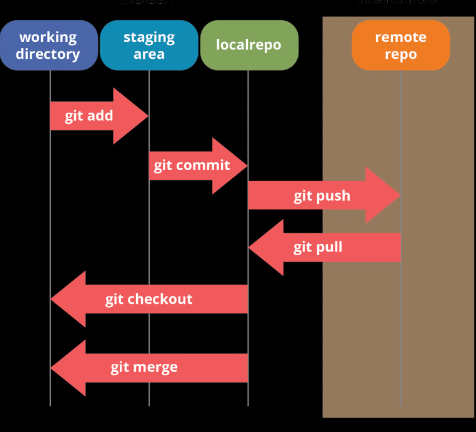


Figure 5 : Quelques commandes Git

## Typologie des fonctionnalités de GitHub

* + **Gestion des dépôts**

GitHub permet de créer, de cloner, de supprimer et de gérer des dépôts Git à la fois locaux et distants.

* + **Collaborations**

GitHub offre des fonctionnalités pour faciliter la collaboration entre les membres d’une équipe de développement, notamment la possibilité de créer des demandes de pull (pull requests), de gérer les commentaires, de réviser le code, d’approuver les modifications et de fusionner les branches.

* + **Suivi des problèmes (issue tracking)**

GitHub fournit un système intégré de suivi des problèmes qui permet de signaler, de suivre et de résoudre les problèmes, les bogues ou les demandes de fonctionnalités.

* + **Intégration continue**

GitHub offre des fonctionnalités d’intégration continue (CI) qui permettent d’automatiser les processus de construction, de test et de déploiement à chaque nouvelle modification du code source.

* + **Pages GitHub**

Les Pages GitHub permettent de publier et d’héberger des sites web statiques directement à partir d’un dépôt GitHub.

## Typologie des workflows de développement

* + **Workflow centralisé**

Dans ce workflow, un référentiel central est utilisé comme source de vérité pour le code. Les développeurs clonent le référentiel, effectuent leurs modifications localement, puis soumettent des demandes de fusion (pull requests) pour proposer leurs changements. Un administrateur ou un responsable de projet est chargé de fusionner les demandes de fusion dans le référentiel central.

* + **Workflow de flux de travail basé sur les branches**

Ce workflow encourage l'utilisation de branches pour travailler sur des fonctionnalités, des correctifs de bugs ou des tâches spécifiques. Chaque développeur crée sa propre branche pour effectuer des modifications, puis propose une demande de fusion pour intégrer ses changements dans la branche principale (habituellement `master` ou `main`). Cette approche permet une meilleure séparation des tâches et facilite la collaboration parallèle.

## Typologie des stratégies de fusion (branch merging)

* + **Fusion normale (fast-forward)**

Lorsque les modifications apportées sur une branche sont en avance par rapport à la branche cible, une fusion normale peut être effectuée en utilisant une fusion "fast-forward" qui déplace simplement le pointeur de la branche cible vers la dernière validation de la branche source.

* + **Fusion récursive (recursive)**

Lorsque les modifications apportées sur une branche sont en conflit avec la branche cible, une fusion récursive est utilisée pour combiner les modifications. Git tente automatiquement de fusionner les modifications de manière automatisée, mais si des conflits surviennent, ils doivent être résolus manuellement.

Cette typologie n'est pas exhaustive, mais elle donne un aperçu des différentes fonctionnalités et concepts que l'on peut rencontrer lors de l'utilisation de Git et GitHub.

# Sécurité

La sécurité est un aspect crucial dans le cadre de Git et GitHub, car ces outils sont utilisés pour gérer et partager du code source. Voici quelques points importants liés à la sécurité dans le contexte de Git et GitHub :

## Contrôle d'accès

* + ***Authentification***

GitHub propose des options d'authentification sécurisées telles que l'authentification à deux facteurs (2FA) pour protéger les comptes des utilisateurs. Il est recommandé d'activer cette fonctionnalité pour renforcer la sécurité.

* + ***Autorisations d'accès***

GitHub permet de configurer des autorisations d'accès granulaires pour les dépôts, les branches et les fonctionnalités spécifiques. Les administrateurs peuvent définir qui peut lire, écrire ou fusionner des modifications dans un dépôt donné.

## Gestion des secrets

* + ***Variables d'environnement sécurisées***

GitHub permet de stocker des variables d'environnement sécurisées dans les dépôts. Cela permet de gérer de manière sécurisée les informations sensibles tels que les clés d'API, les identifiants de connexion, etc.

* + ***GitHub Actions et workflows***

Lors de l'utilisation de GitHub Actions pour l'automatisation des workflows, il est important de prendre des mesures de sécurité appropriées pour protéger les secrets et les informations sensibles utilisées dans les workflows.

## Audit et suivi

* + ***Activité des dépôts***

Git et GitHub conservent un historique complet des activités de modification du code, y compris les auteurs, les dates et les messages de commit. Cela permet de suivre les modifications, d'identifier les problèmes de sécurité potentiels et de retracer les responsabilités.

* + ***Journalisation et surveillance***

Il est recommandé d'activer la journalisation et la surveillance des activités sur les dépôts et les comptes GitHub afin de détecter toute activité suspecte ou non autorisée.

## Gestion des vulnérabilités

* + ***Dépendances et mises à jour***

Il est important de maintenir à jour les dépendances du projet en utilisant des versions sécurisées et de suivre les annonces de sécurité pour les bibliothèques et les outils utilisés.

* + ***Analyse de sécurité***

GitHub propose des fonctionnalités intégrées d'analyse de sécurité qui permettent de détecter les vulnérabilités connues dans le code source. Il est recommandé d'utiliser ces outils pour identifier les problèmes potentiels.

## Bonnes pratiques de développement sécurisé

* + ***Validation et sanitisation des données d'entrée***

Il est essentiel de valider et de filtrer les données d'entrée pour prévenir les attaques d'injection de code.

* + ***Gestion des secrets***

Les identifiants, les clés d'API et les informations sensibles doivent être stockés en toute sécurité et ne pas être inclus dans le code source.

* + ***Revue de code***

La revue de code par les pairs est une pratique importante pour détecter les erreurs de sécurité et les vulnérabilités dans le code source.

Il est important de souligner que la sécurité est un sujet complexe et en constante évolution. Il est recommandé de suivre les meilleures pratiques de sécurité, de rester informé des dernières mises à jour et de consulter la documentation officielle de Git et GitHub pour obtenir des informations plus détaillées sur les mesures de sécurité spécifiques à ces outils.

# Avantages et inconvénients

## Avantages

Git et GitHub offrent de nombreux avantages dans le développement logiciel collaboratif. Voici quelques-uns des avantages clés:

### Gestion efficace des versions

* + **Suivi des modifications**

Git permet de suivre précisément les modifications apportées au code source, y compris les ajouts, les suppressions et les modifications de fichiers. Cela permet de retracer l'évolution du code au fil du temps et de comprendre les raisons des changements.

* + **Branches**

Git facilite la création de branches, ce qui permet aux développeurs de travailler sur des fonctionnalités ou des correctifs de bugs isolément, sans perturber la branche principale. Cela favorise le développement parallèle et la collaboration efficace.

* + **Fusion facile**

Git propose des outils de fusion avancés pour combiner les modifications provenant de différentes branches. Cela permet d'intégrer facilement les fonctionnalités développées séparément et de résoudre les conflits éventuels.

### Collaboration et gestion des projets

* + **Plateforme centralisée**

GitHub fournit une plateforme centralisée pour héberger des dépôts Git, ce qui facilite le partage, la collaboration et la contribution au code source. Les développeurs peuvent cloner, modifier et soumettre des demandes de fusion (pull requests) pour proposer leurs modifications.

* + **Suivi des problèmes**

GitHub offre un système intégré de suivi des problèmes (issue tracking) qui permet de signaler, de suivre et de résoudre les problèmes, les bogues et les demandes de fonctionnalités. Cela facilite la communication entre les membres de l'équipe et la gestion des tâches.

* + **Révision de code**

GitHub facilite la révision de code grâce à des fonctionnalités telles que les commentaires sur les demandes de fusion et les revues de code par les pairs. Cela permet d'améliorer la qualité du code, d'identifier les erreurs et de partager des connaissances au sein de l'équipe.

### Sauvegarde et restauration

* + **Repositories distants**

Git permet de cloner des dépôts locaux à partir de dépôts distants, ce qui facilite la sauvegarde du code source et la collaboration avec d'autres développeurs. Les dépôts distants agissent comme des sauvegardes et permettent de restaurer le code en cas de perte ou de corruption des dépôts locaux.

* + **Historique complet**

Git conserve un historique complet des modifications apportées au code, ce qui permet de revenir facilement à des versions antérieures en cas de besoin. Cela offre une protection contre les erreurs et permet de récupérer rapidement en cas de problème.

### Intégration continue et déploiement

* + **Intégration avec des outils CI/CD**

GitHub s'intègre facilement avec des outils d'intégration continue (CI) et de déploiement continu (CD) tels que Travis CI, Jenkins, CircleCI, etc. Cela permet d'automatiser les processus de construction, de test et de déploiement du code.

* + **GitHub Actions**

GitHub Actions est un système d'automatisation intégré qui permet d'exécuter des workflows personnalisés en réponse à des événements spécifiques dans le dépôt. Cela facilite l'automatisation des tâches récurrentes et le déploiement de logiciels.

## Inconvénients

Bien que Git et GitHub offrent de nombreux avantages, il y a également quelques inconvénients à prendre en compte :

### Courbe d'apprentissage initiale

* + **Git**

Git a une courbe d'apprentissage initiale assez raide, en particulier pour les développeurs novices qui n'ont pas encore utilisé un système de contrôle de version. Comprendre les concepts clés de Git, tels que les commits, les branches et les fusions, peut nécessiter un certain temps et une formation adéquate.

* + **GitHub**

Bien que l'interface utilisateur de GitHub soit conviviale, il peut y avoir une certaine complexité lors de la configuration des autorisations d'accès, de la gestion des problèmes ou de l'intégration avec d'autres outils.

### Dépendance à l'égard des services hébergés

* **Git**

Git lui-même n'a pas de dépendance à l'égard d'un service hébergé spécifique. Cependant, de nombreux développeurs utilisent des services Git hébergés tels que GitHub, GitLab ou Bitbucket pour faciliter la collaboration et la gestion des projets. Cela signifie que si ces services rencontrent des problèmes de disponibilité ou sont interrompus, cela peut affecter l'accès au code source et aux fonctionnalités supplémentaires fournies par ces plateformes.

### Complexité des fusions et des conflits :

* **Git**

Bien que Git propose des outils puissants pour fusionner les modifications de différentes branches, les fusions peuvent parfois être complexes, en particulier lorsqu'il y a des conflits entre les modifications apportées sur des parties similaires du code. La résolution de conflits peut nécessiter des compétences et des efforts supplémentaires.

* **GitHub**

Bien que GitHub facilite la résolution des conflits avec des fonctionnalités telles que les demandes de fusion et les commentaires, il peut toujours y avoir des scénarios où la résolution des conflits nécessite une attention particulière et une communication étroite entre les contributeurs.

### Dépendance à l'égard de la connectivité Internet :

* **Git**

Git est un système de contrôle de version décentralisé, ce qui signifie que les développeurs peuvent travailler localement sans connexion Internet. Cependant, pour certaines fonctionnalités de synchronisation, de collaboration et de partage, comme la synchronisation avec des dépôts distants, il est nécessaire d'avoir une connectivité Internet stable.

* **GitHub**

L'utilisation de GitHub nécessite une connectivité Internet pour cloner, pousser ou tirer des modifications vers et depuis les dépôts distants. L'indisponibilité ou la lenteur de la connexion Internet peut affecter la productivité et la collaboration.

Il est important de noter que la plupart des inconvénients mentionnés ci-dessus sont liés à des aspects spécifiques de l'utilisation de Git et GitHub, et peuvent être atténués avec une connaissance approfondie des outils, une formation adéquate et une bonne gestion des workflows de développement.

# Domaines d’application

Git et GitHub sont utilisés dans de nombreux domaines d'application, notamment :

## Développement logiciel

* **Collaboration d'équipes** : Git et GitHub permettent aux équipes de développeurs de travailler ensemble sur un même projet, de partager leur code source, de gérer les modifications et de fusionner les contributions.
* **Contrôle de version** : Git offre un contrôle de version puissant, permettant de gérer l'historique des modifications, de revenir à des versions antérieures du code et de suivre l'évolution du projet au fil du temps.
* **Intégration continue (CI) et déploiement continu (CD)** : Git et GitHub s'intègrent facilement à des outils d'automatisation tels que Jenkins, Travis CI ou CircleCI pour automatiser les processus de construction, de test et de déploiement du code.

## Open source et développement communautaire

* **Hébergement de projets open source** : GitHub est largement utilisé pour héberger et partager des projets open source. Les développeurs peuvent contribuer aux projets existants, signaler des problèmes et soumettre des demandes de fusion pour proposer des améliorations.
* **Collaboration communautaire** : GitHub fournit des outils pour faciliter la collaboration entre les développeurs open source, notamment le suivi des problèmes, les revues de code et les discussions sur les demandes de fusion.

## Gestion de projet

* **Suivi des problèmes** : GitHub propose un système intégré de suivi des problèmes, qui permet de signaler, de suivre et de résoudre les problèmes, les bogues et les demandes de fonctionnalités.
* **Tableaux de projet** : GitHub fournit des fonctionnalités de gestion de projet basées sur des tableaux Kanban, qui permettent de suivre l'état des tâches et d'organiser le travail de manière visuelle.

## Documentation technique

* **Pages GitHub** : GitHub permet de créer et de publier facilement des sites web statiques à partir des dépôts. Cela peut être utilisé pour héberger de la documentation technique, des guides de démarrage rapide, des tutoriels, etc.
* **Wiki** : GitHub propose également une fonctionnalité de wiki intégrée, permettant de créer et de maintenir une documentation collaborative directement associée à un dépôt.

## Gestion de contenu

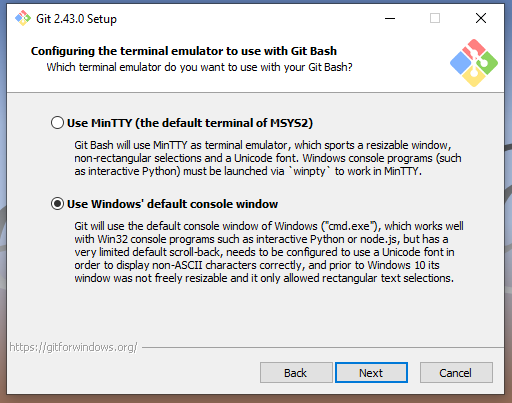
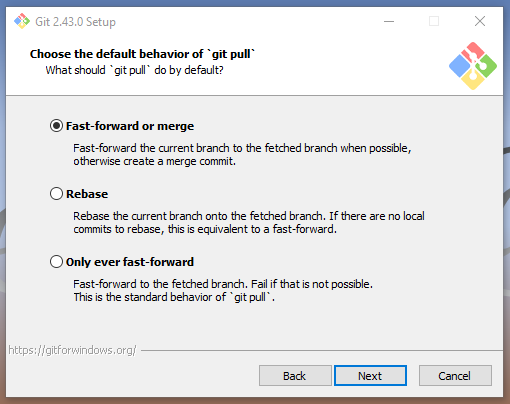
* **Gestion de fichiers** : Git permet de gérer les modifications apportées à des fichiers de toutes sortes, pas seulement du code source. Il peut être utilisé pour suivre les modifications apportées à des fichiers de configuration, des documents texte, des images, etc.
* **Contrôle de version pour les médias** : Git LFS (Large File Storage) est une extension de Git qui permet de gérer les fichiers binaires volumineux tels que les images, les vidéos ou les fichiers audio dans un dépôt Git.

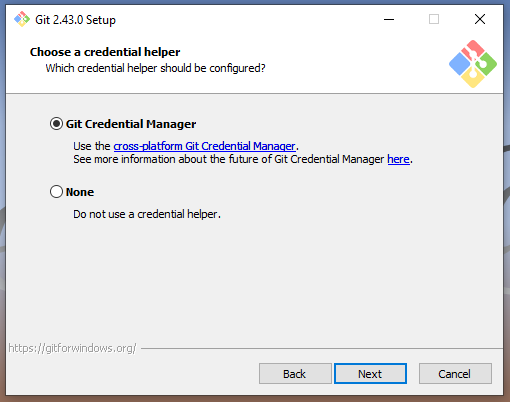
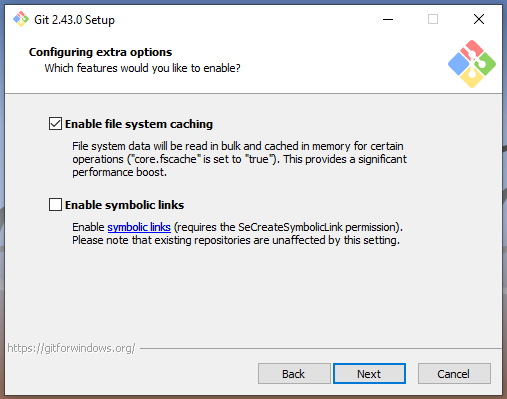
Il convient de noter que ces domaines d'application ne sont pas exhaustifs et que Git et GitHub peuvent être utilisés de différentes manières en fonction des besoins spécifiques de chaque projet et organisation.

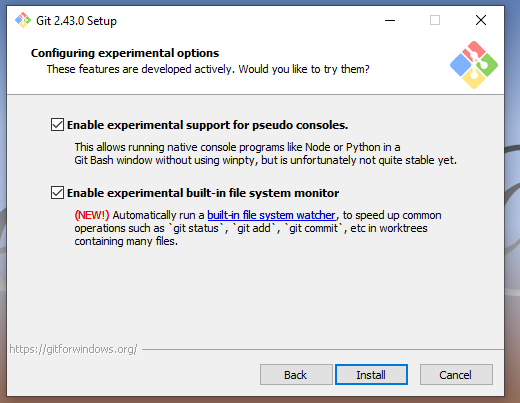
# Implémentation

Installation de Git depuis le **cmd.exe :**

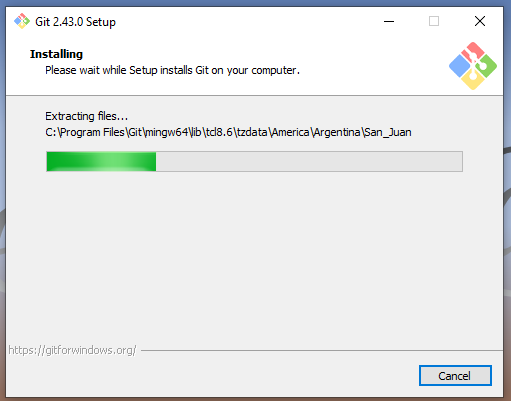
**winget install --id Git.Git -e --source winget**

** **

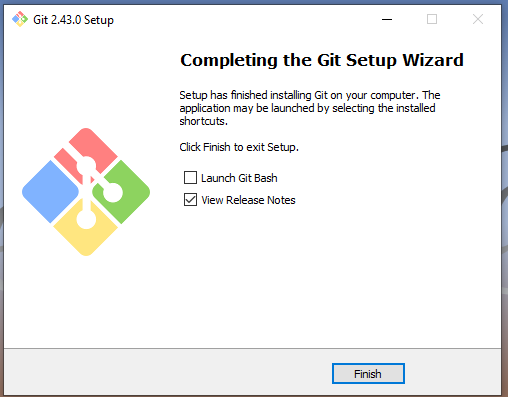
** **

****

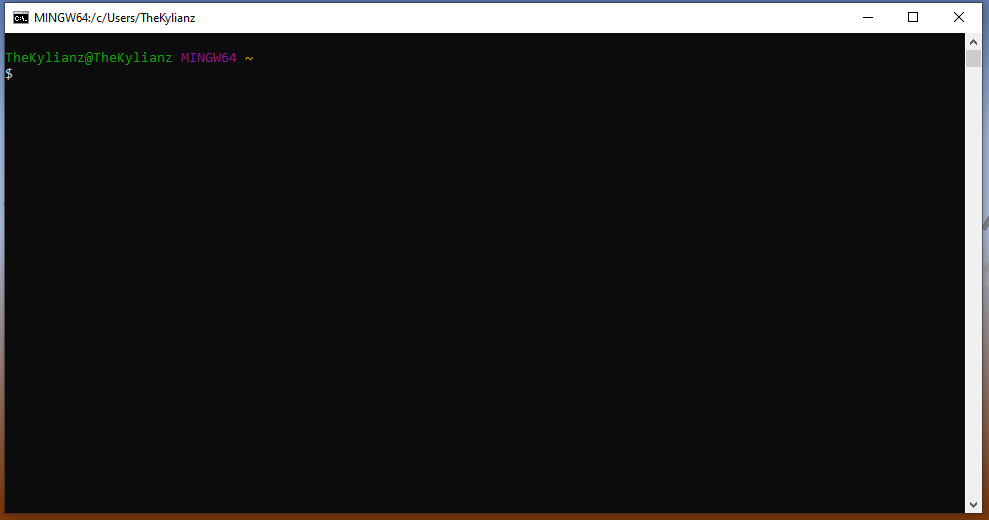
Pour toutes ces captures, du début jusqu’à arriver à install, on clique sur Next puis sur Install pour installer le logiciel. On peut également télécharger Git depuis le site officiel de Git via le lien git.com.

****

L’installation a commencé, cela prendra une certaine minute donc attendre la fin.

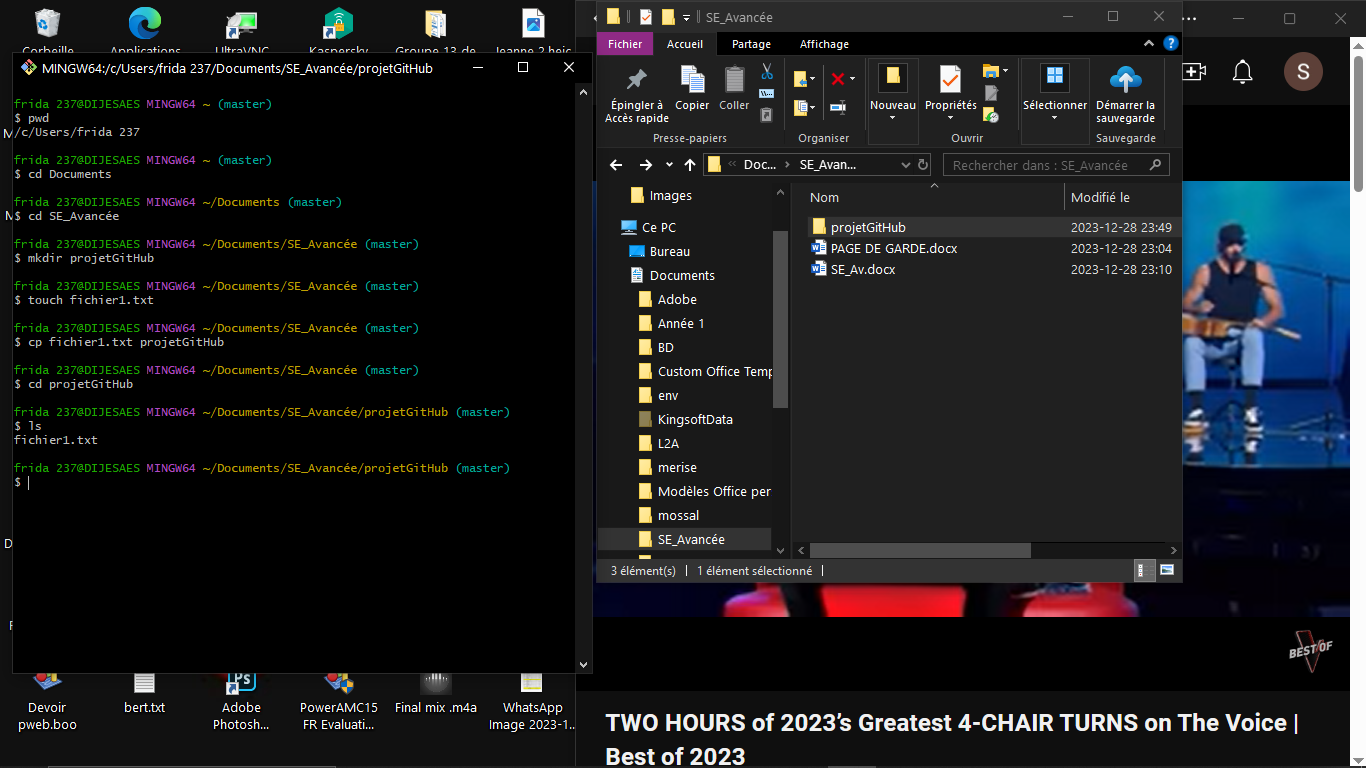
****

Une fois l’installation terminée, cocher la case Launch Git Bash pour lancer le terminal de Git, le résultat est ci-dessous. Et si on veut ou pas voir les notes sur la version installée, on peut cocher ou décocher selon ce qu’on veut.

****

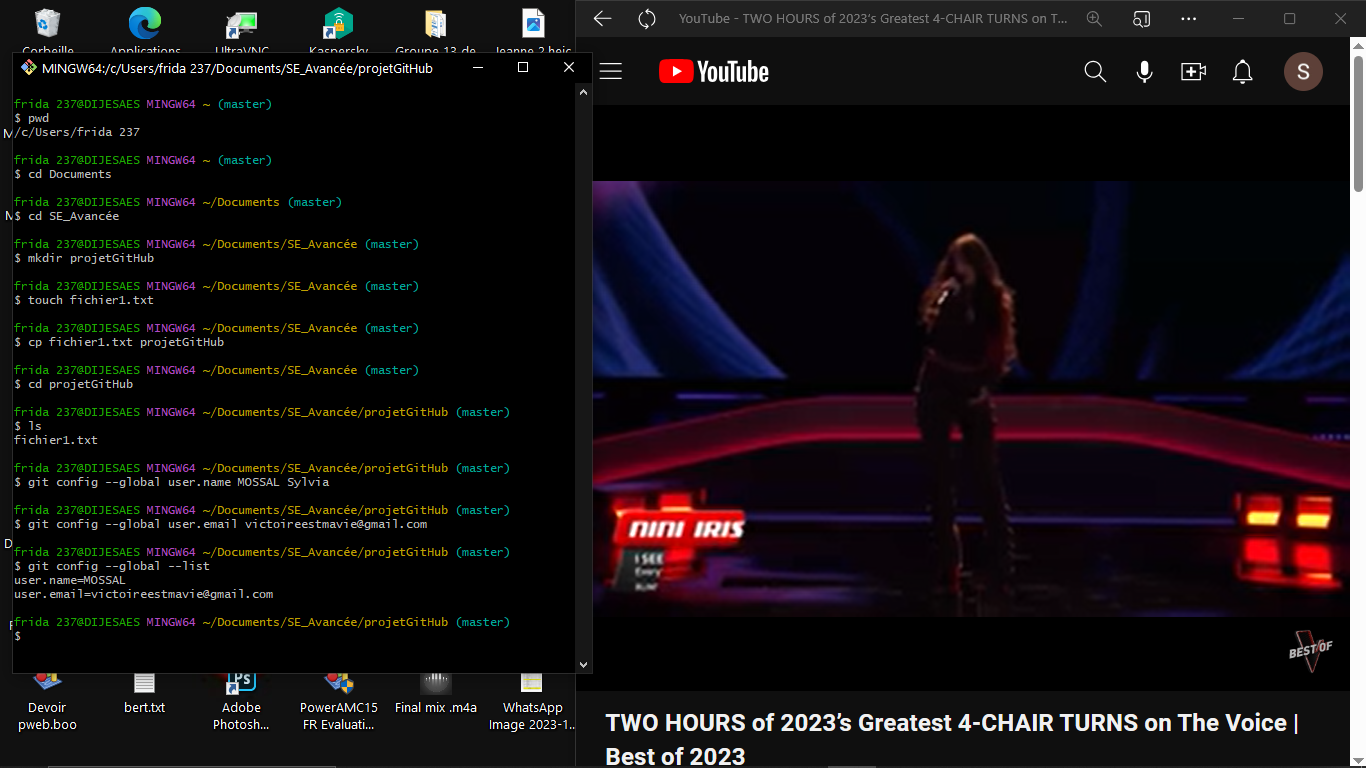
Le terminal de Git est lancé. Ci-dessus, on peut voir le nom d’utilisateur qui a lancé le terminal et le nom de la machine. On peut faire beaucoup de manipulation sur cette interface en utilisant les commandes correspondant à ce qu’on voudra faire. Notamment :

**Naviguer dans Git Bash :**

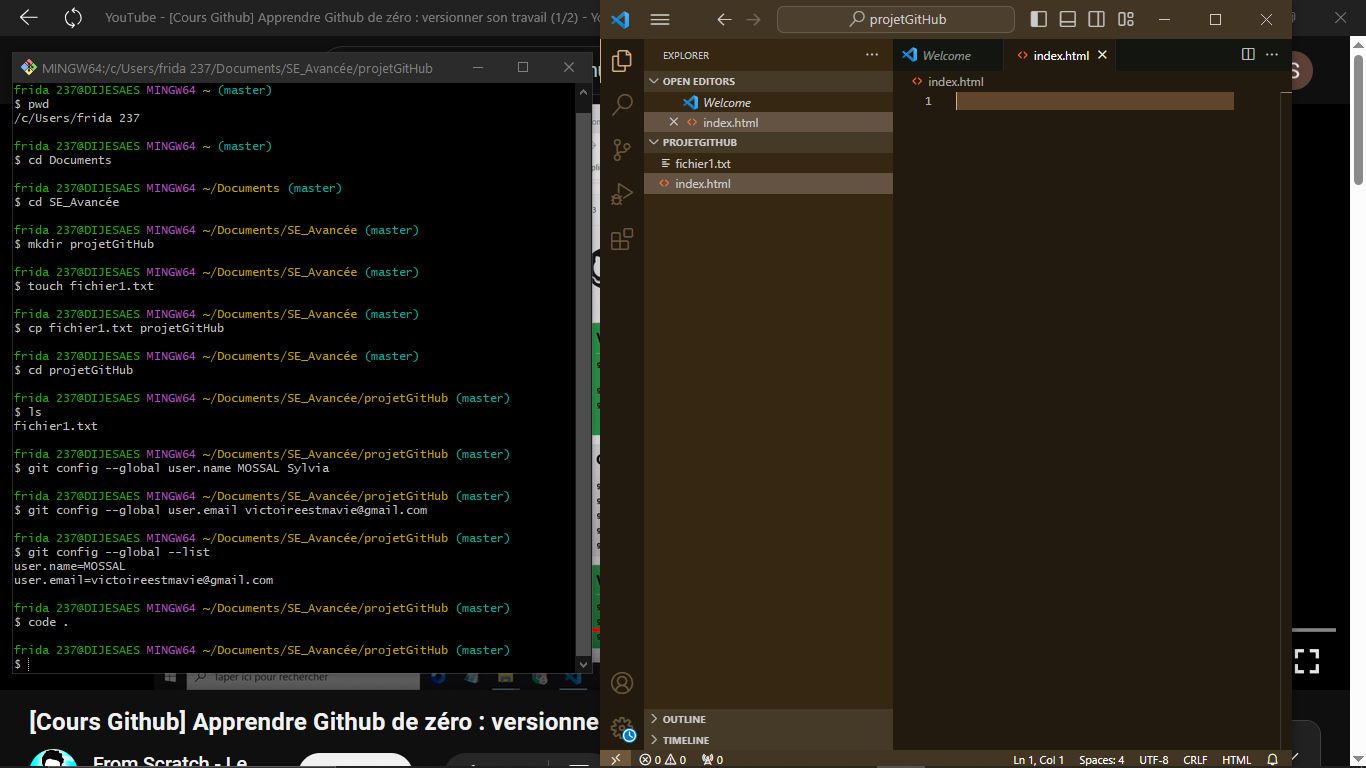


Sur cette commande, la première commande « **pwd** » permet de voir dans quel répertoire l’on se trouve, ici nous sommes dans le répertoire **/c/Users/frida 237.** La deuxième **« cd »** pour se déplacer d’un répertoire à un autre, ici **cd Documents** nous amène dans Documents ensuite **cd SE\_Avancée**. La commande **« mkdir projetGithub »** crée le dossier **projetGithub** dans lequel on crée le fichier **fichier1.txt** puis en listant ce dossier en faisant **ls,** on peut bel et bien que le fichier y est.

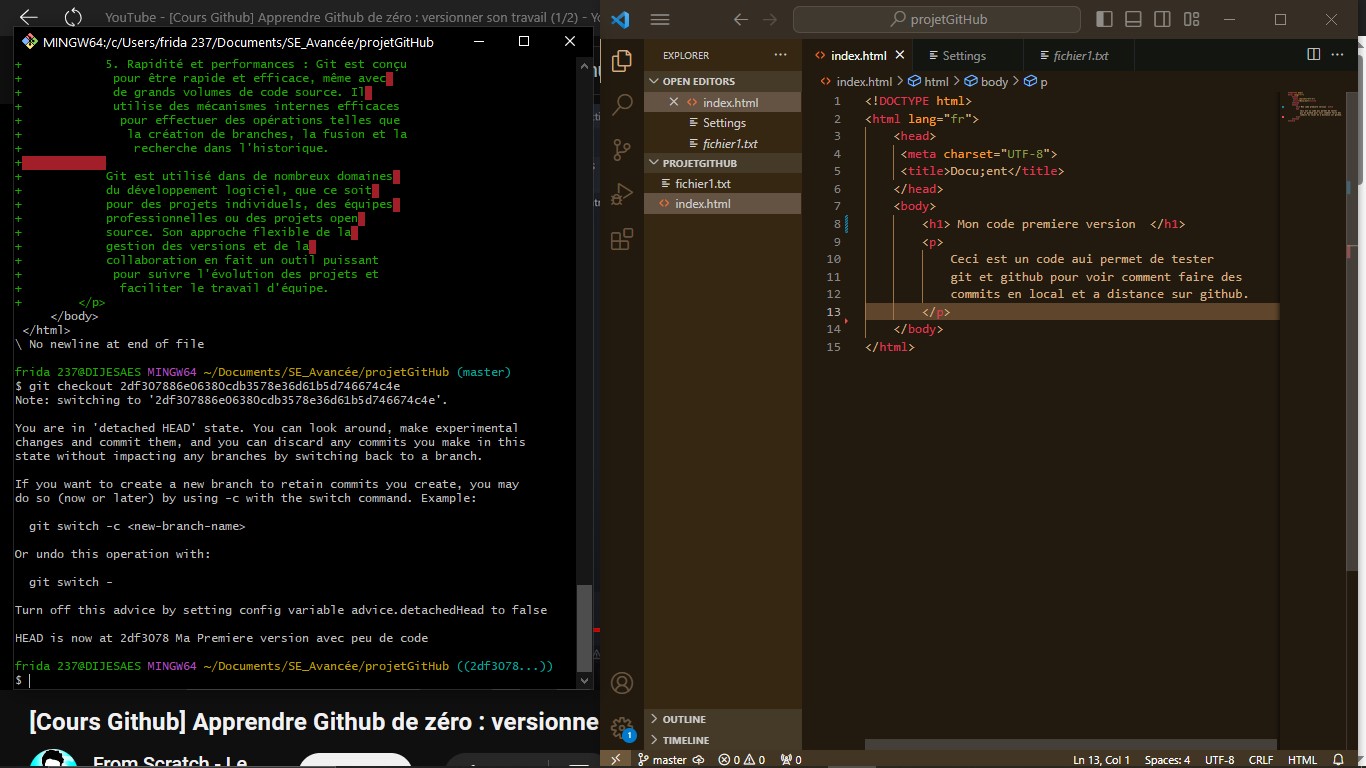
**Configuration de Git :**



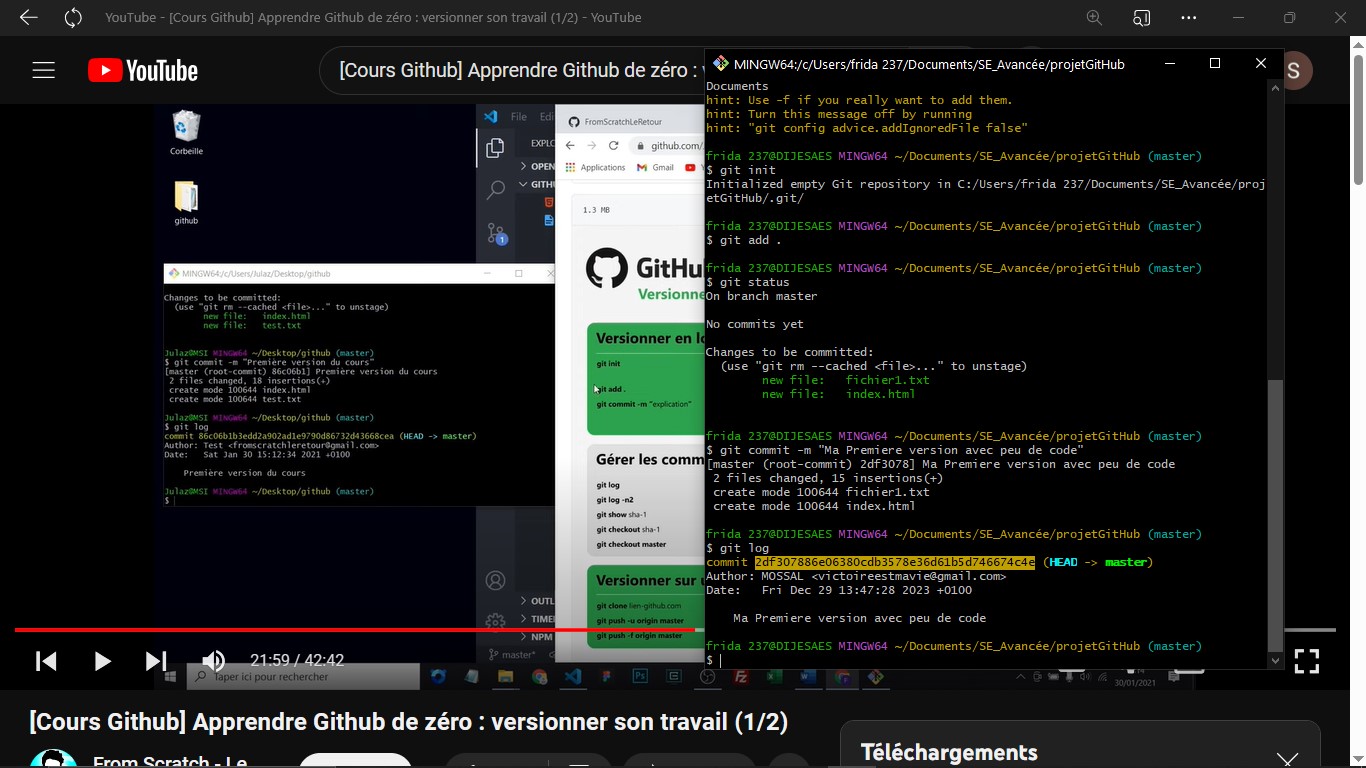
La commande « **git config --global user.name MOSSAL Sylvia** » configure Git de telle manière de reconnaître MOSSAL Sylvia comme nom d’utilisateur, « **git config --global user.email victoireestmavie@gmail.com** » configure l’adresse mail de l’utilisateur puis « **git config --global --list**» affiche donc le nom d’utilisateur et son e-mail.



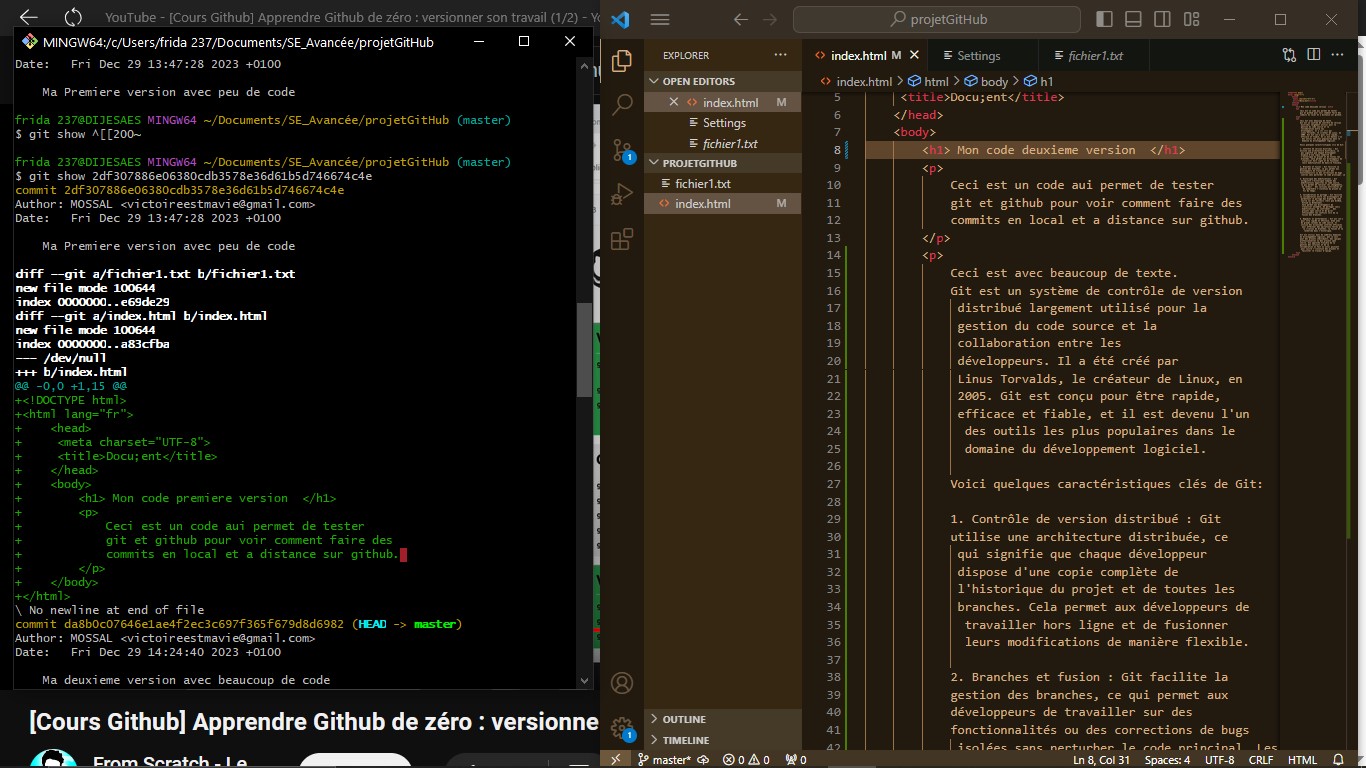
Ici, la commande « **code . »** lance VS Code, l’éditeur de code que mon système possède. Une fois dans VS Code, on crée le fichier **index.html** dans laquelle on écrit la première version de notre code que l’on peut voir dans la capture suivante :



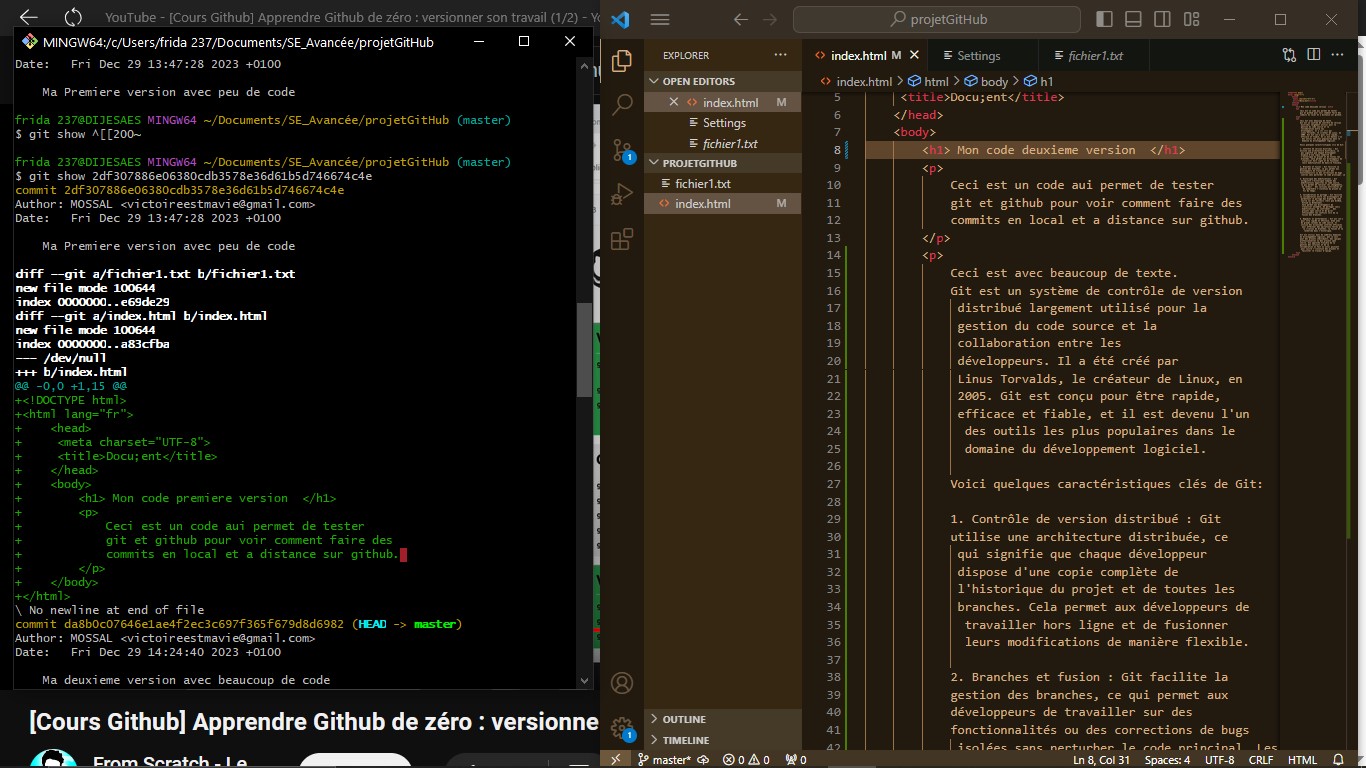
Une fois le code écrit, on peut le versionner en local avec Git. Pour cela, on se rend dans Git Bash. La capture suivante expliquera les actions à poser sur ce fichier.



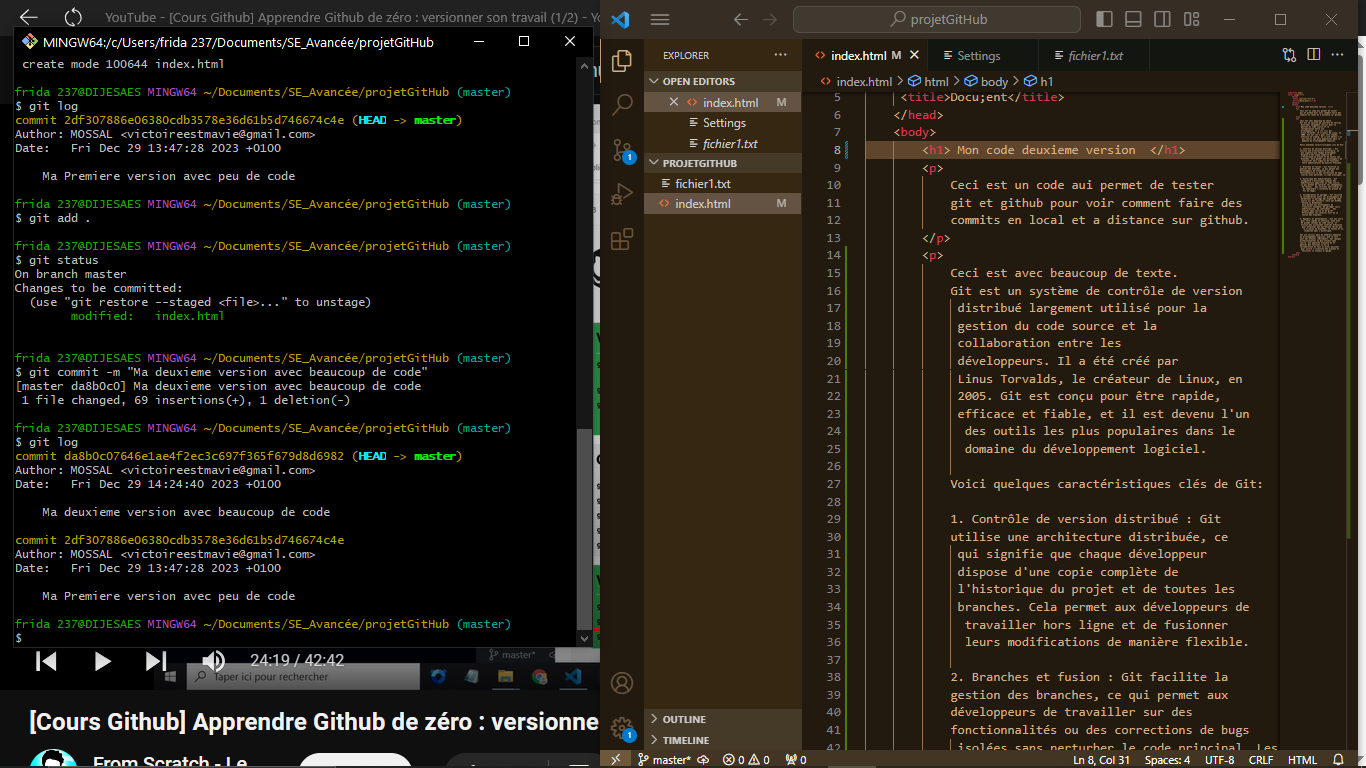
Avec **“git init”,** on initialise le dépôt en le mettant dans le bon dossier, ce qui crée le fichier **.git** qui contiendra toutes les informations sur le dépôt. On met les fichiers du dossier projetGithub dans une zone d’index avec la commande **« git add . »** où ils recevront toutes les modifications à chaque évolution. En faisant donc « git status », nous pouvons voir l’état des différents fichiers du dossier, ici, nous sommes sur la branche master et le dossier contient deux fichiers dont **index.html** et **fichier1.txt** et la commande **« git commit -m ‘Ma premiere version avec peu de code’ »** crée un nouveau commit dans lequel les sauvegardes réelles seront faites. Puis enfin, la commande **« git log »**  permet de lister les commits avec leurs informations relatives telle que sha-1 qui est ici la suite de caractère surlignée en jaune. Ainsi donc, notre premier commit en local est fait et on peut le consulter avec la commande **« git show sha-1 »** suivant :

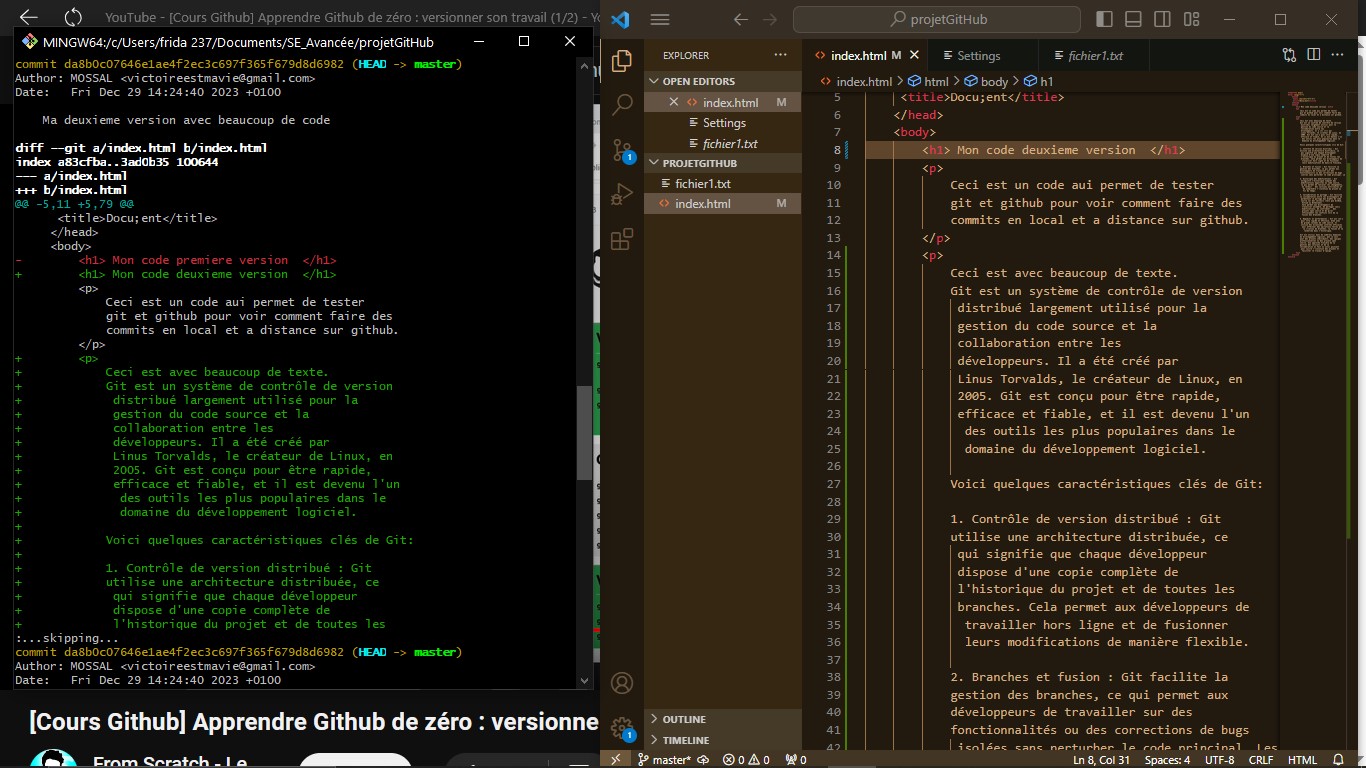


On se rend ensuite dans VS Code pour coder la deuxième version de notre projet. Le résultat est le suivant :



On peut donc le commit en local. Les captures d’écran suivantes l’illustreront mieux avec l’entrée de la commande **« git show sha-1 »** qui est précédée par **git add** ., **git commit -m ‘Ma deuxieme version de code’, git log** permettant de connaître son sha-1.

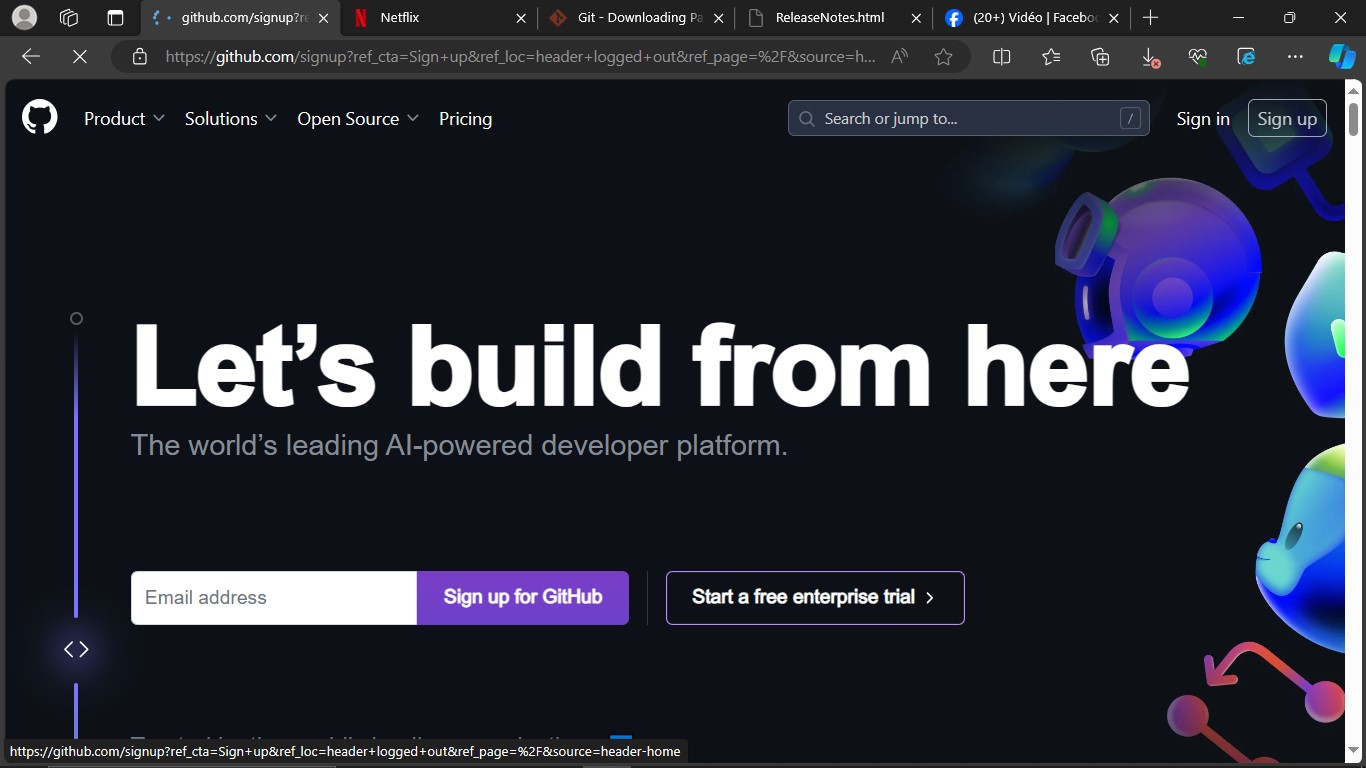




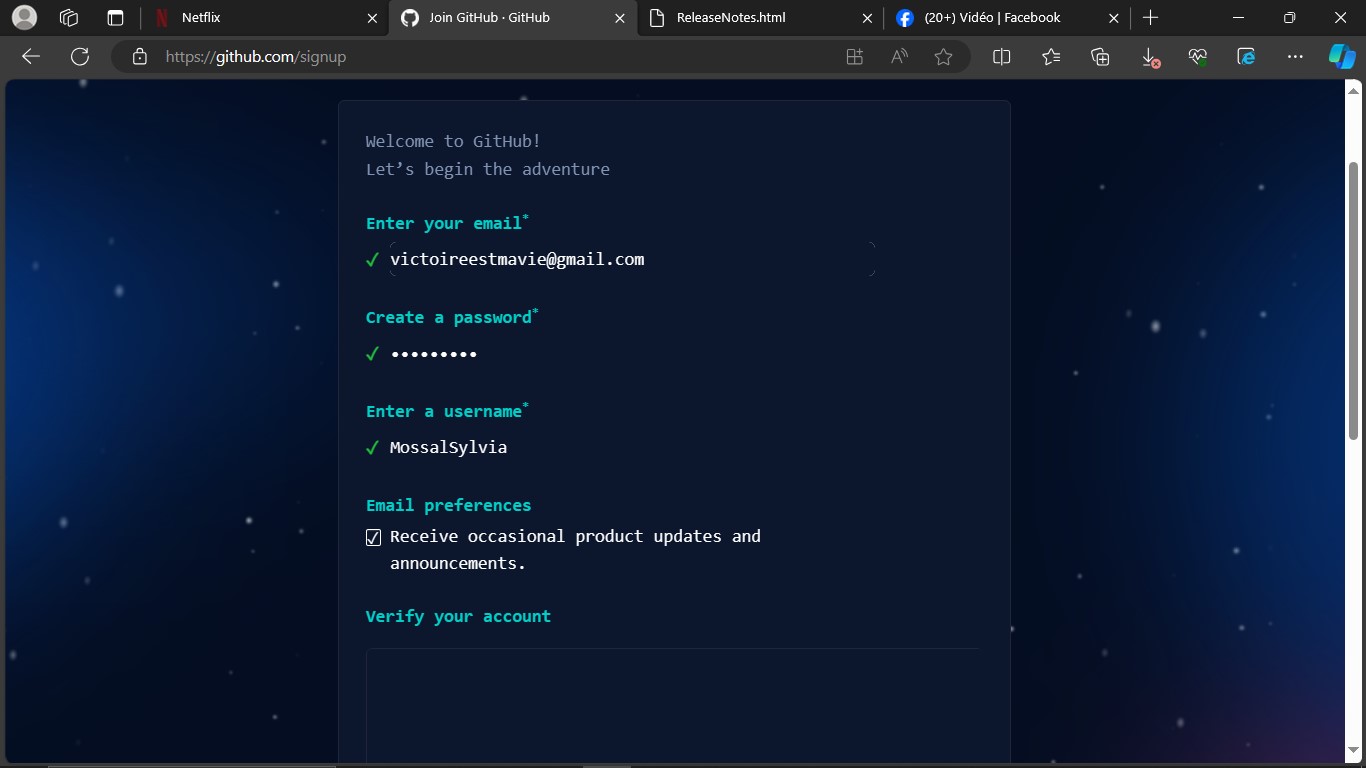
Le versioning en local de notre code est fait maintenant mais le code n’est toujours en lieux sûr car si un problème arrive sur la machine locale, notre code pourrait disparaître. Pour remédier à cela, nous allons créer un compte GitHub pour le téléverser.

Création d’un compte GitHub

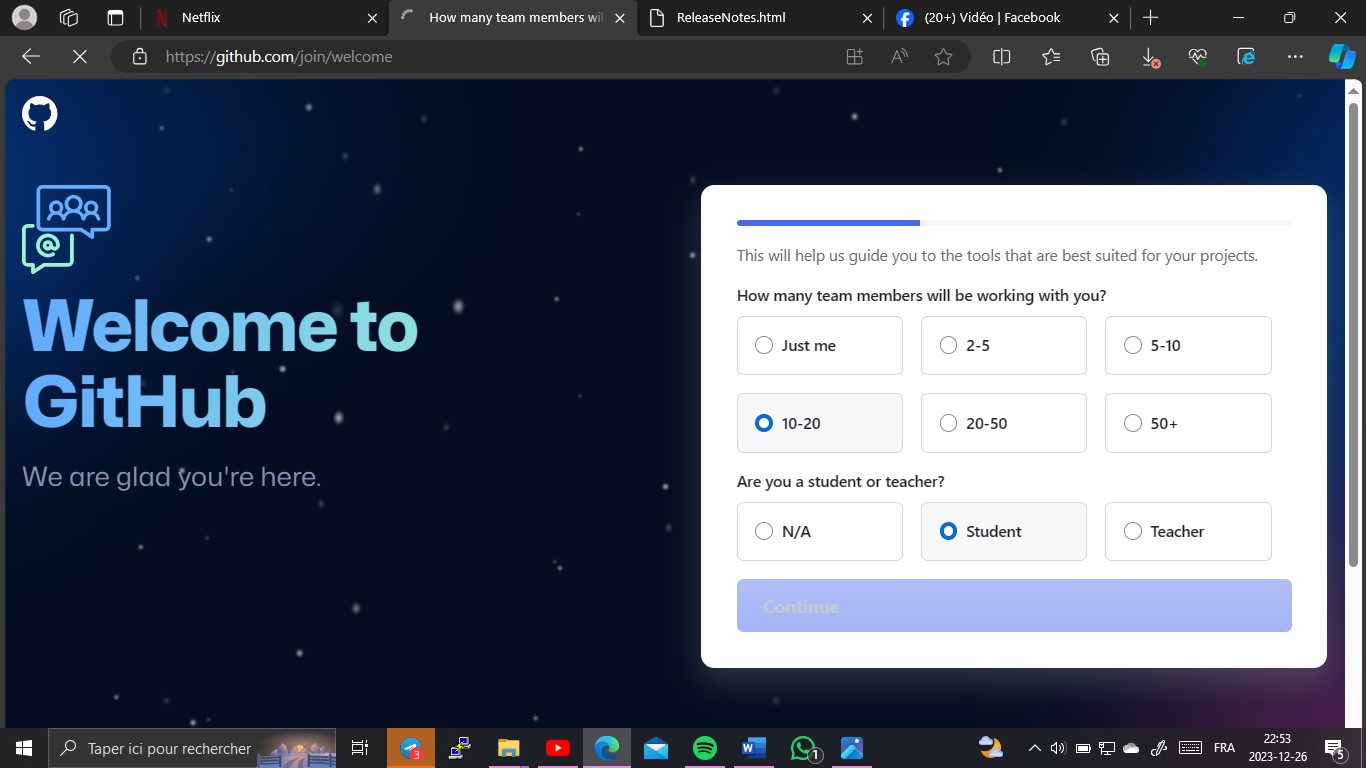
On se rend sur le site officiel de GitHub via le lien <https://github.com>



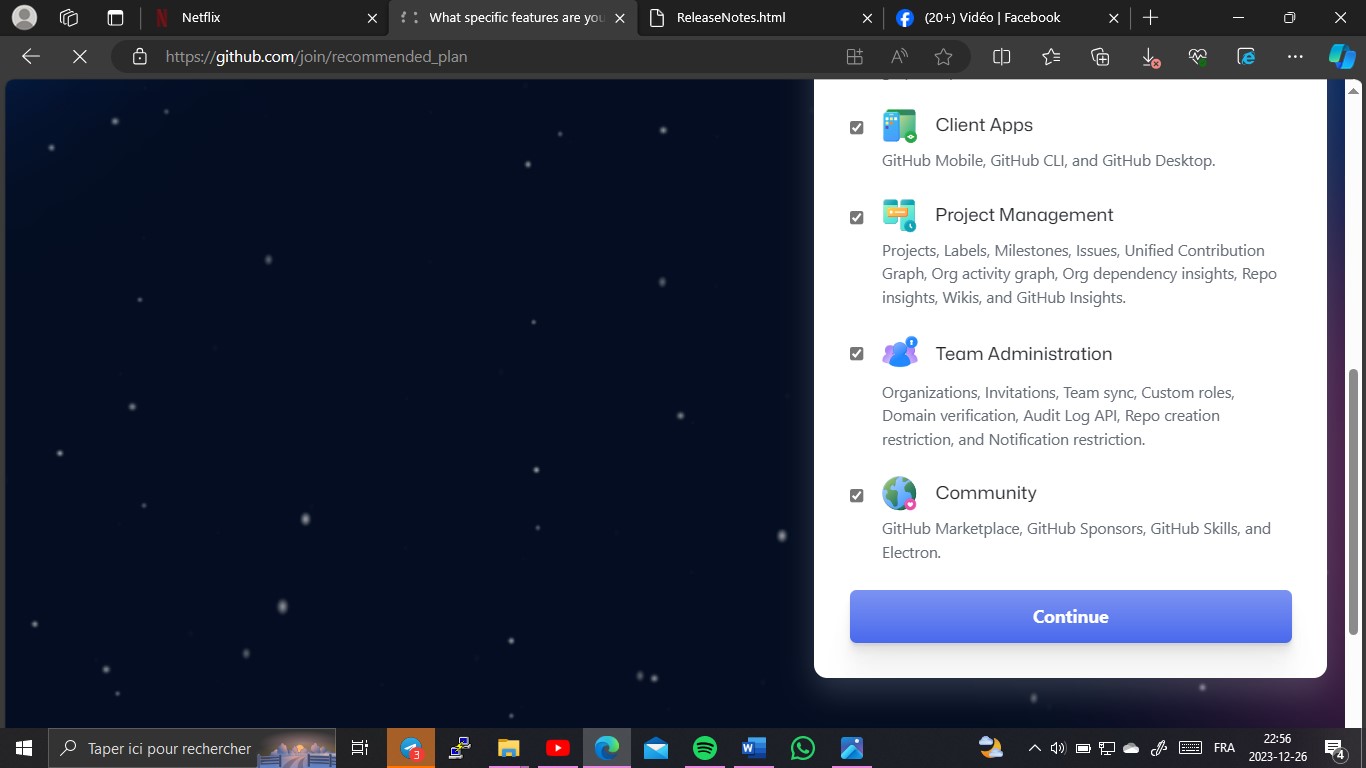
On clique sur l’onglet **Sign Up** pour créer un compte, ce qui nous amène à cette page suivante. Il faut remplir les différents champs selon les différentes informations demandées dont l’email, le mot de passe, le nom d’utilisateur, pour arriver à chaque champ, il faut cliquer sur **continue** pour aller à une autre page. Pour vérifier son compte, il faut cliquer sur le bon dessin pour vérifier que l’utilisateur est un humain.



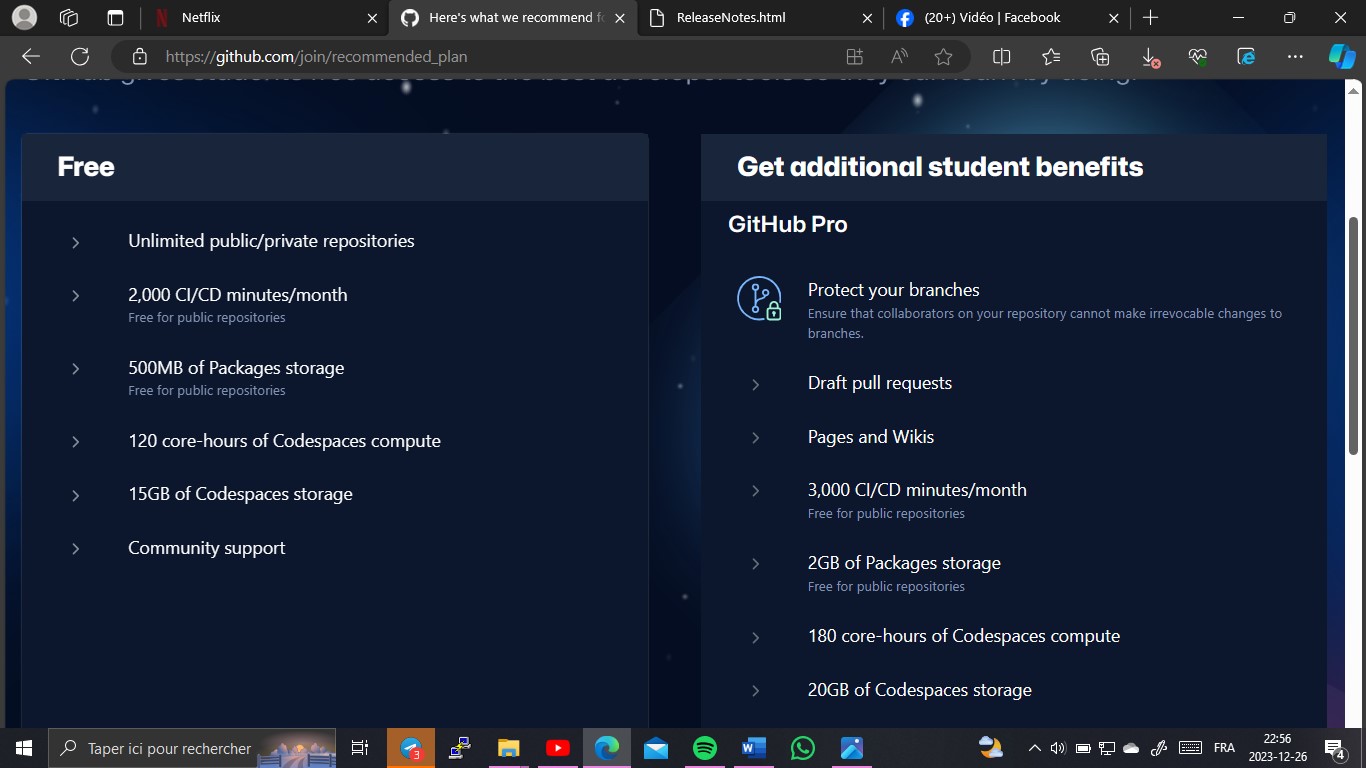
Une fois cela fait, il faut cliquer sur **continue** pour aller à la plage suivante où il faudra choisir sa date de naissance puis sur **continue.**

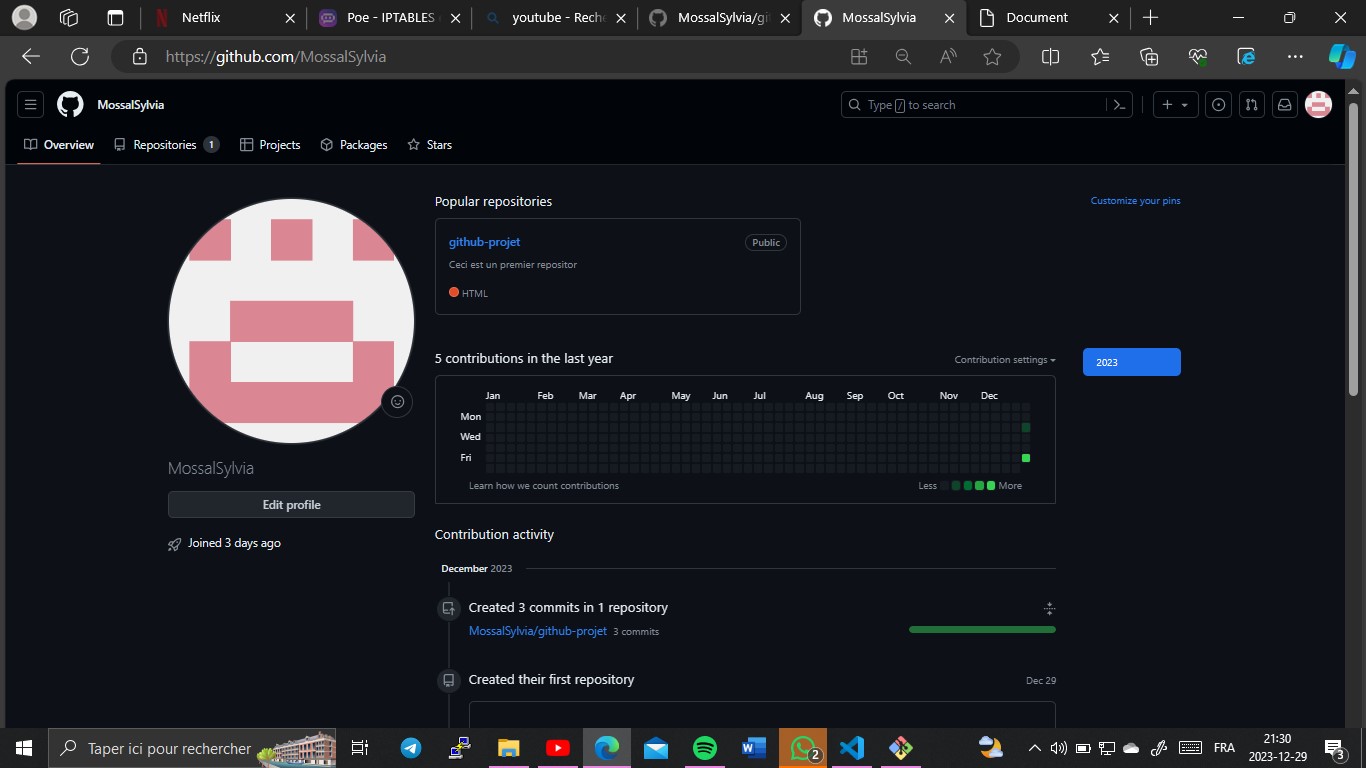


Ici, l’âge de l’utilisateur est entre **10-20** et a pour fonction **Student.**



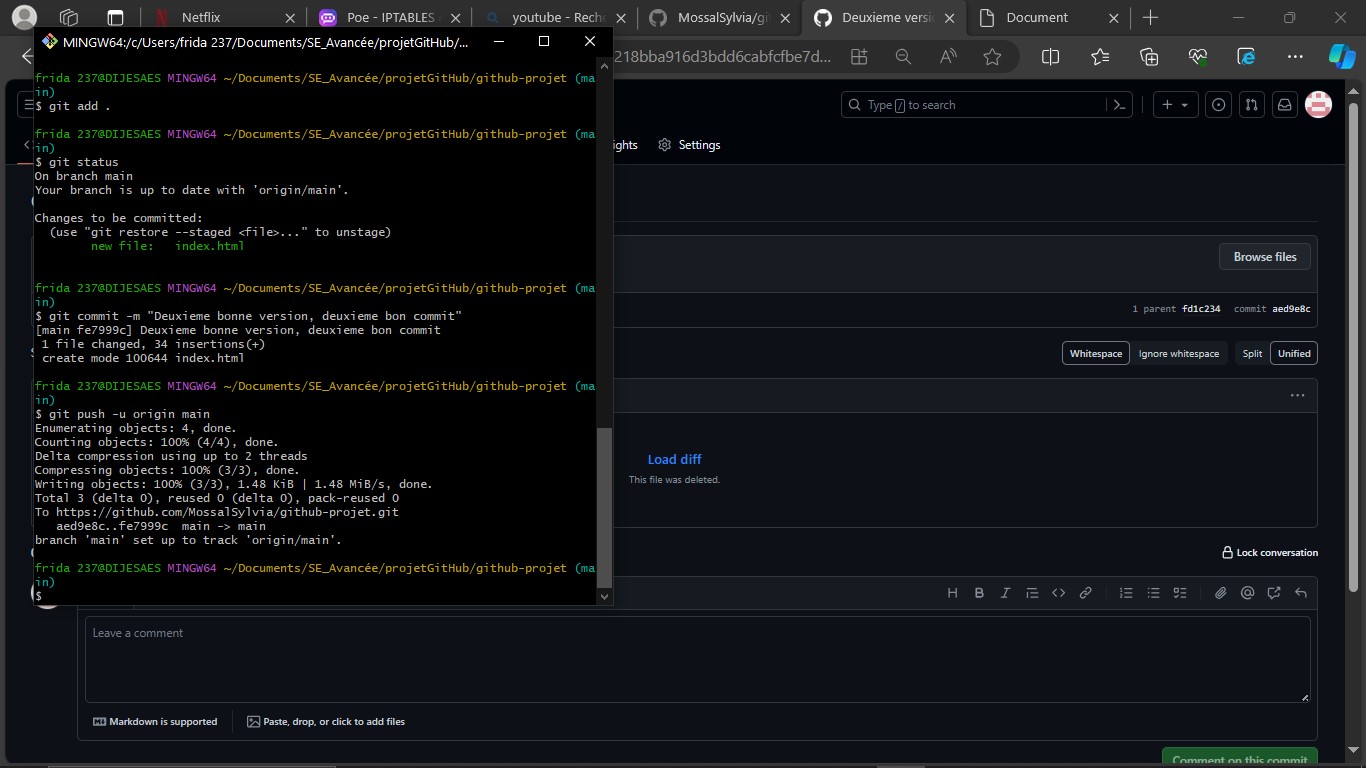
Ici, il faut cocher sur toutes les cases dont on veut selon notre convenance puis cliquer sur **continue** pour aller à la page suivante où il faudra cliquer sur **continue for free** tout en bas pour utiliser la plateforme gratuitement.



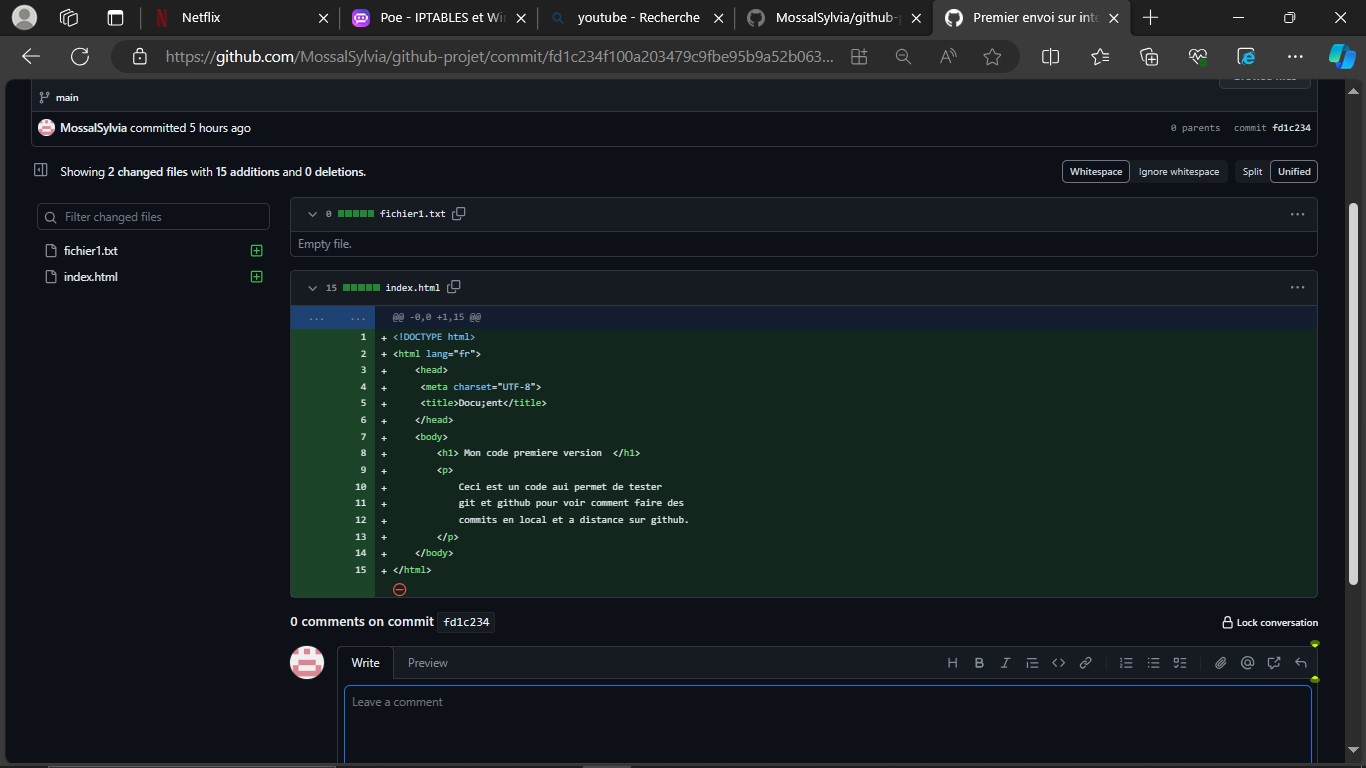
Une fois cela fait, nous sommes rédigés à la page de connexion où il faudra entrer son e-mail ou son nom d’utilisateur et le mot de passe puis cliquer sur **Sign in** pour se connecter à son compte puis on accède à l’interface de GitHub. 

Ainsi donc notre compte GitHub est créé. Il faut maintenant aller dans Git Bash pour versionner notre code sur GitHub.

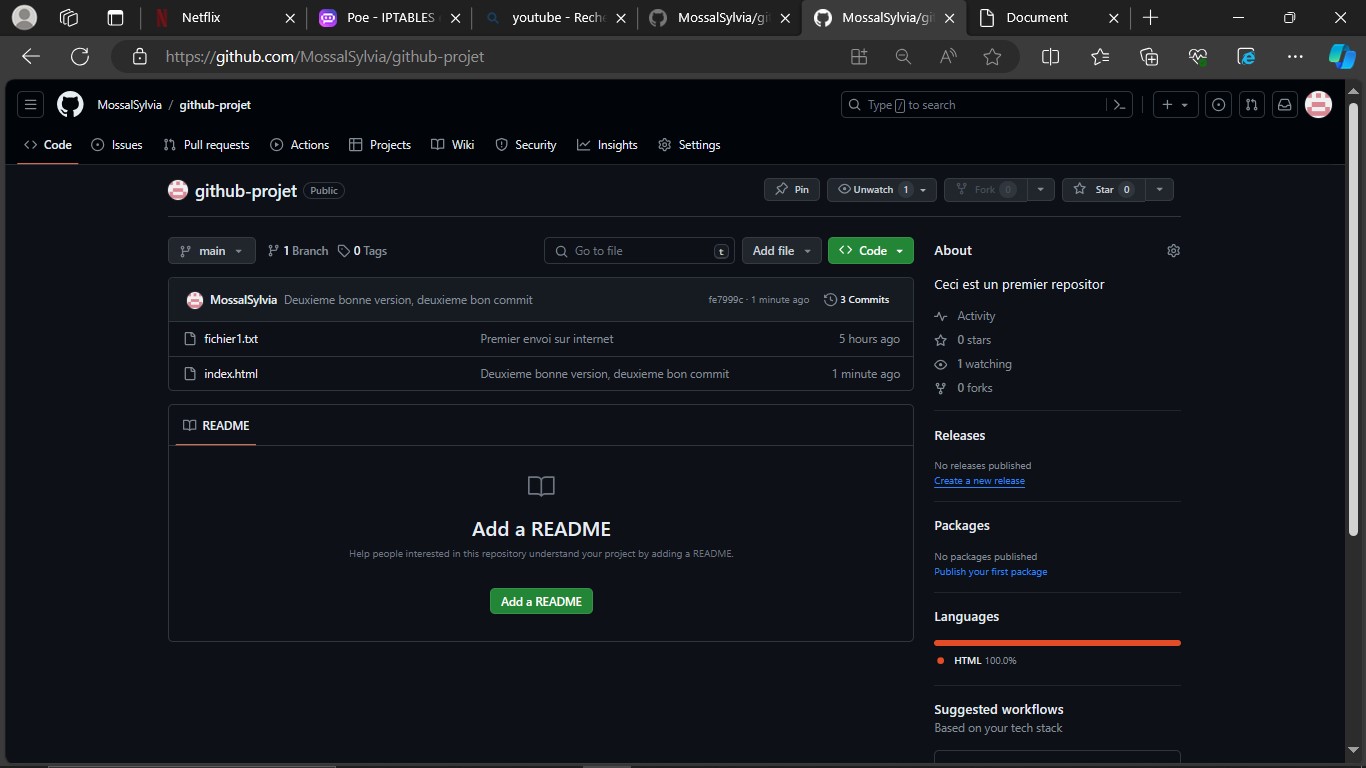
On téléverse notre code sur GitHub dans la branche principale ou main avec la commande « **git push -u origin main »**  précédée par **git add ., git commit -m 'Ma deuxième vrai version, deuxième vrai commit’.** La capture suivante l’illustre mieux :



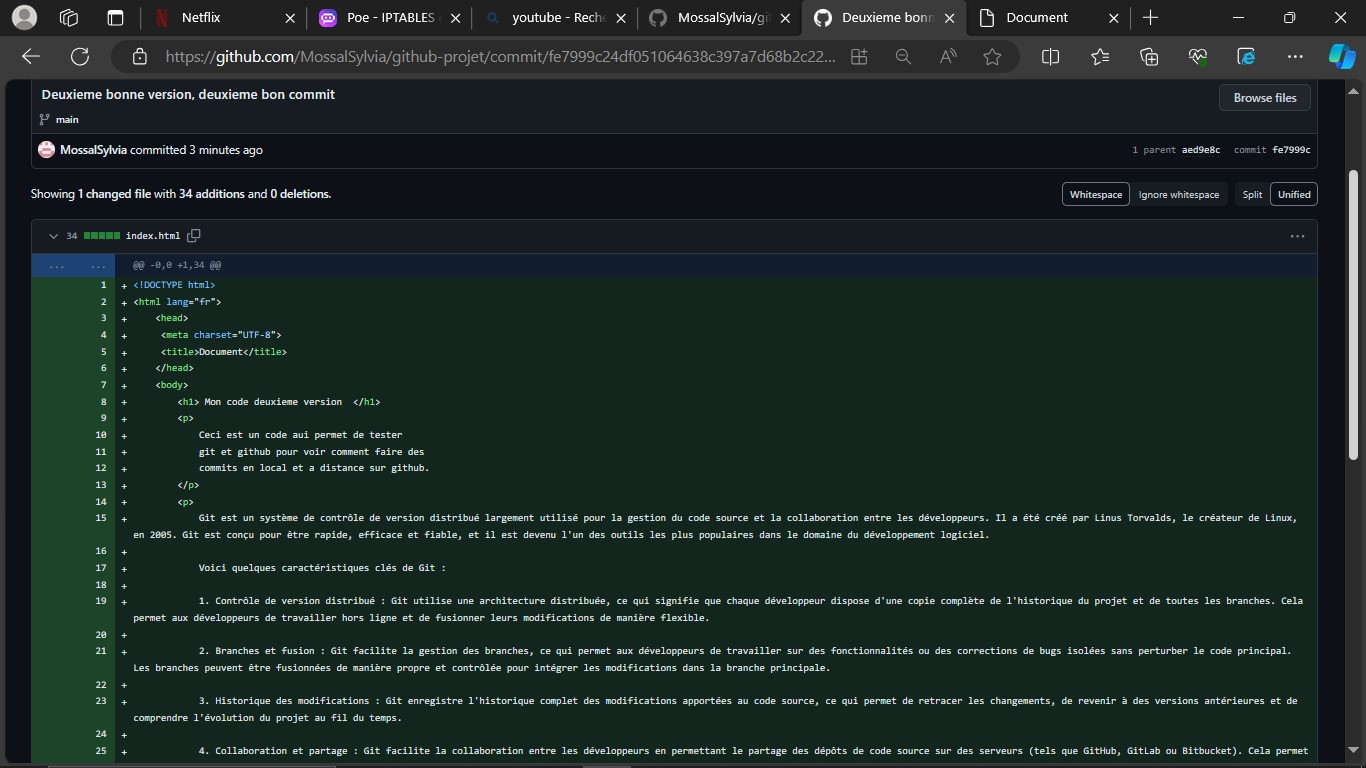
Vérifions nos dépôts sur la plateforme :



Ici, l’utilisateur MossalSylvia est connecté et a commit deux fichiers dont index.html et fichier1.txt avec la première version du code dont on peut consulter.

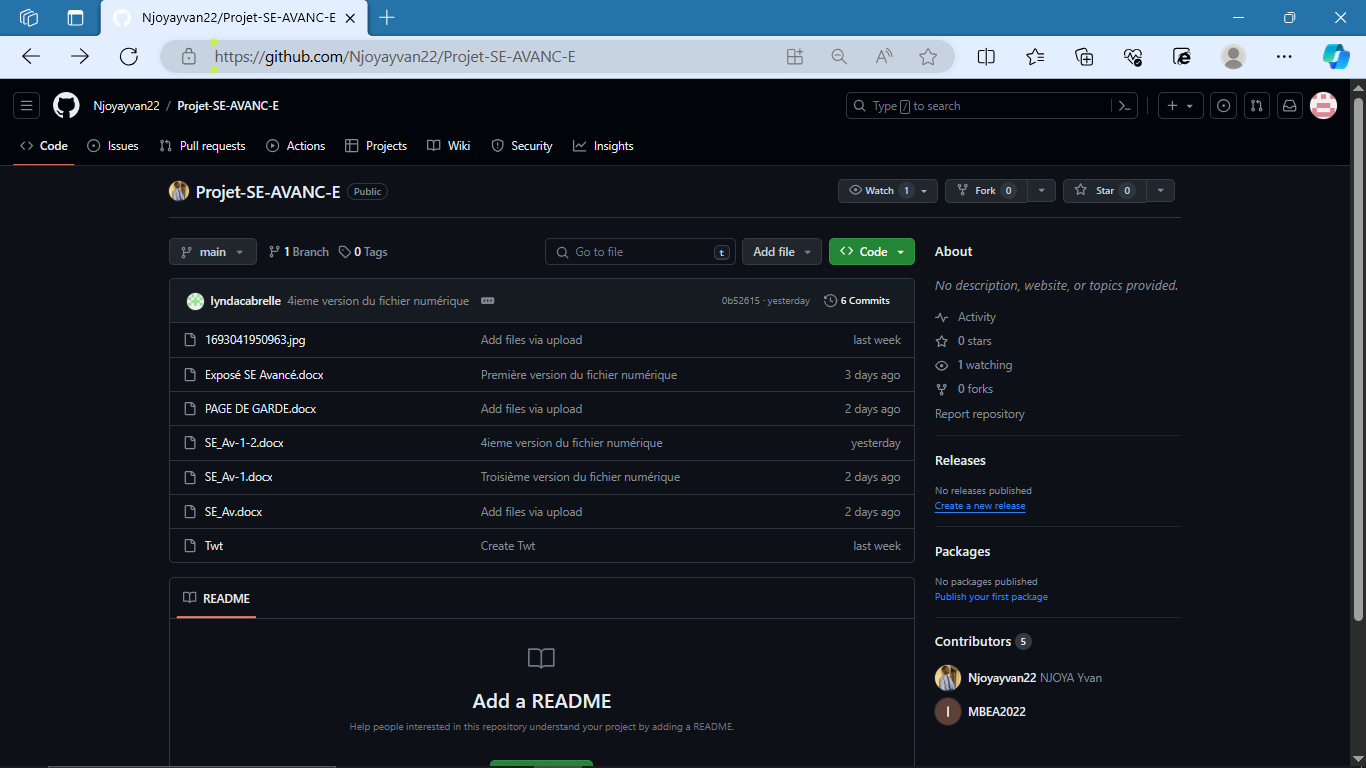


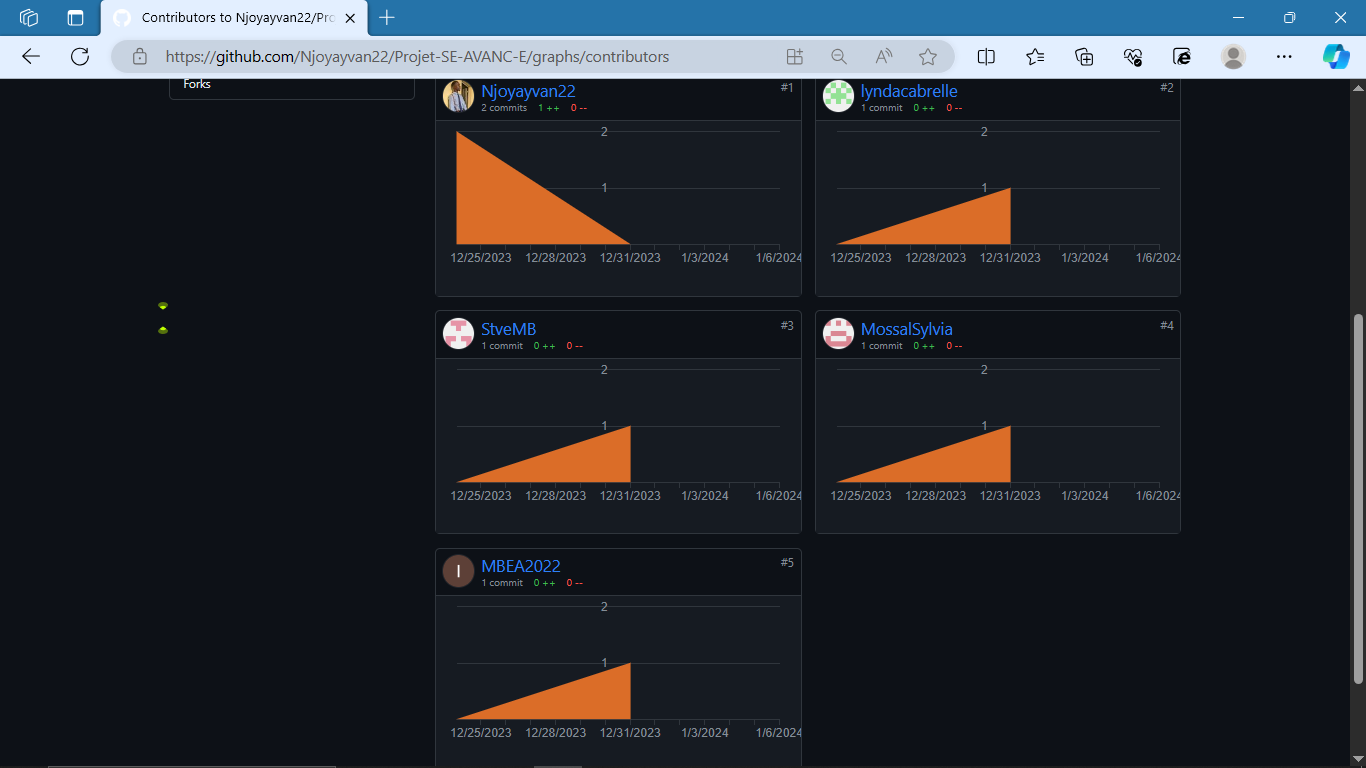
La deuxième version est commit et on peut le vérifier en cliquant sur le fichier en question dont index.html. Le résultat est le suivant :



On peut également travailler en collaboration sur GitHub, dans la capture qui suit, on peut y voir les différents commits qui ont été faits et en cliquant sur chacun, on pourra donc voir qui a fait quoi.

Sur cette capture suivante, le groupe Projet-SE-AVANC-E est créé et a en son sein les différents membres ou utilisateurs, chacun a commit les fichier selon ce qu’il a eu à améliorer.



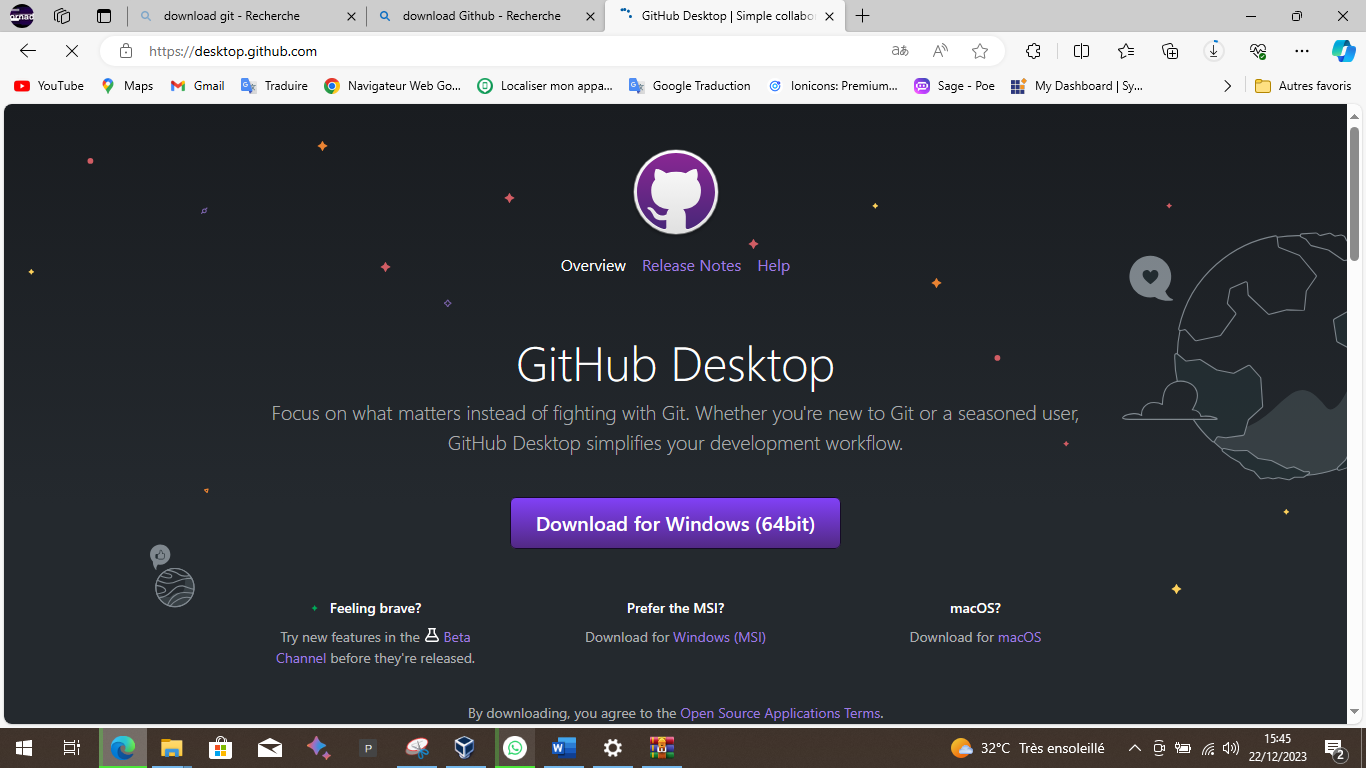


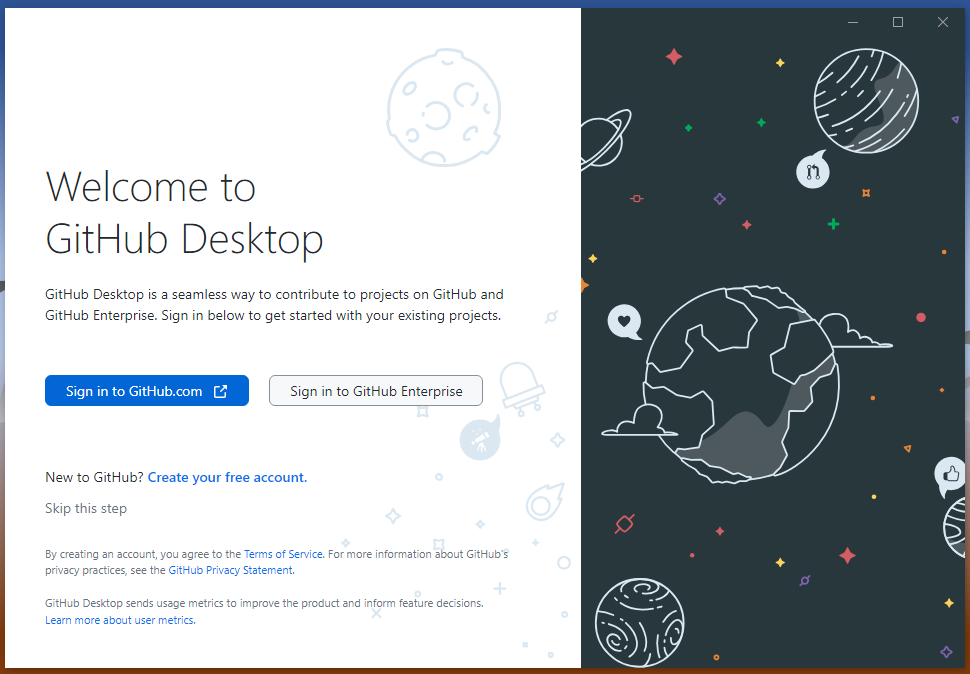
Sur cette capture, nous pouvons voir le commit de chaque collaborateur.

Il est également possible d’utiliser les commandes dans Git Bash pour téléverser le travail sur GitHub ou le faire directement sur la plateforme pour ceux qui ne maîtrise pas les commandes.

**Installation de GitHub Desktop :**

Il est possible d’installer GitHub en tant que logiciel dans son poste de travail puis créer son compte et suivre les différentes étapes citées plus haut pour créer son compte et exploiter github.com.





Cliquer sur Sign in to GitHub.com pour accéder à la plate-forme.

# Conclusion GENeRALE

En définitive, Git et GitHub offrent de nombreux avantages qui en font des outils essentiels pour le développement logiciel collaboratif. Git permet une gestion efficace des versions, favorise la collaboration et simplifie la fusion des modifications. GitHub, quant à lui, offre une plateforme centralisée pour héberger des dépôts Git, facilite la collaboration entre les développeurs, propose des fonctionnalités de suivi des problèmes et de révision de code, et s'intègre à des outils d'intégration continue et de déploiement. Cependant, il est important de noter certains inconvénients potentiels, tels que la courbe d'apprentissage initiale, la dépendance à l'égard des services hébergés, la complexité des fusions et des conflits, ainsi que la dépendance à l'égard de la connectivité Internet. Ces inconvénients peuvent être atténués grâce à une formation adéquate, une gestion appropriée des workflows et une compréhension approfondie des outils. Dans l'ensemble, Git et GitHub trouvent leur place dans de nombreux domaines d'application, tels que le développement logiciel, l'open source, la gestion de projet, la documentation technique et la gestion de contenu. Leur utilisation permet d'améliorer la collaboration, de faciliter le suivi des modifications, d'automatiser les processus et de favoriser la transparence et la participation communautaire. En tant qu'outils largement adoptés par la communauté des développeurs, Git et GitHub sont des ressources précieuses pour les équipes de développement souhaitant travailler de manière collaborative, efficace et transparente.

# Glossaire

**Archiver (archive)** : Archiver un dépôt Git signifie créer une copie compressée (par exemple, un fichier .zip) de tout l'historique et des fichiers du dépôt à un moment donné

**Branche (branch)** : Une branche est une version distincte d'un dépôt Git qui peut être créée pour travailler sur des fonctionnalités ou des correctifs spécifiques sans affecter le code principal

**Clone** : Cloner un dépôt signifie créer une copie locale complète d'un dépôt distant.

**Collaboration** : La collaboration dans le contexte de Git et GitHub fait référence au travail d'équipe sur un projet.

**Commit** : Un commit est une opération dans Git qui enregistre les modifications apportées aux fichiers d'un dépôt.

**Conflit (conflict)** : Un conflit survient lorsqu'il y a des modifications contradictoires dans un fichier lors de la fusion de branches ou de l'application d'un pull request.

**Dépôt (repository)** : Un dépôt est une structure de stockage utilisée par Git pour enregistrer les fichiers, les dossiers et l'historique des modifications d'un projet.

**Fork** : Fork est une fonctionnalité de GitHub qui permet à un utilisateur de créer une copie indépendante d'un dépôt existant.

**Fusion (merge)** : La fusion est le processus de combinaison des modifications provenant d'une branche vers une autre branche.

**Git** : Git est un système de contrôle de version distribué.

**GitHub** : GitHub est une plateforme en ligne qui utilise Git pour l'hébergement de projets et la gestion du développement collaboratif.

**Gitignore** : Un fichier .gitignore est utilisé pour spécifier les fichiers et les dossiers qui doivent être ignorés par Git.

**Histoire (history)** : L'histoire d'un dépôt Git fait référence à l'ensemble des commits et des modifications qui ont été enregistrés au fil du temps.

**Organisation (organization)** : Une organisation est une entité sur GitHub qui regroupe des dépôts et des membres dans un espace commun.

**Origine (origin)** : L'origine est le nom généralement donné au dépôt distant par défaut lors de la configuration d'un dépôt Git local.

**Poussoir (push)** : Pousser est l'action d'envoyer les modifications locales d'un dépôt Git vers un dépôt distant

**Pull request** : Un pull request (demande de tirage) est une fonctionnalité de GitHub qui permet aux développeurs de proposer des modifications à un dépôt.

**Rébase (rebase)** : La rébase est une opération qui permet de déplacer, de fusionner ou de réorganiser les commits d'une branche pour les appliquer sur une autre branche

**Référentiel distant (remote repository)** : Un référentiel distant est un dépôt Git hébergé sur un serveur distant, tel que GitHub.

**Révision (revision)** : Une révision fait référence à une version spécifique d'un fichier ou d'un ensemble de fichiers dans un dépôt Git.

**Submodule** : Un sous-module est un référentiel Git incorporé à l'intérieur d'un autre référentiel Git.

**Suivi des problèmes (issue tracking)** : Le suivi des problèmes est une fonctionnalité de GitHub qui permet aux utilisateurs de signaler, de suivre et de résoudre les problèmes, les bugs ou les demandes de fonctionnalités dans un projet.

**Tag** : Un tag est une référence spécifique à un commit dans Git.

**Tirer (pull)** : Tirer est l'action de récupérer les modifications d'un dépôt distant vers un dépôt local.

# **TABLE DES MATIERES**

[SOMMAIRE 1](#_Toc155476198)

[INTRODUCTION GENERALE 2](#_Toc155476199)

[I. Historique 3](#_Toc155476200)

[II. Définition et rôles 4](#_Toc155476201)

[III. Fonctionnement 6](#_Toc155476202)

[IV. Typologie 10](#_Toc155476203)

[IV.1 Typologie des commandes Git 10](#_Toc155476204)

[IV.2 Typologie des fonctionnalités de GitHub 12](#_Toc155476205)

[IV.3 Typologie des workflows de développement 12](#_Toc155476206)

[IV.4 Typologie des stratégies de fusion (branch merging) 13](#_Toc155476207)

[V. Sécurité 14](#_Toc155476208)

[V.1 Contrôle d'accès 14](#_Toc155476209)

[V.2 Gestion des secrets 14](#_Toc155476210)

[V.3 Audit et suivi 15](#_Toc155476211)

[V.4 Gestion des vulnérabilités 15](#_Toc155476212)

[V.5 Bonnes pratiques de développement sécurisé 15](#_Toc155476213)

[VI. Avantages et inconvénients 16](#_Toc155476214)

[VI.1 Avantages 16](#_Toc155476215)

[VI.1.1 Gestion efficace des versions 16](#_Toc155476216)

[VI.1.2 Collaboration et gestion des projets 17](#_Toc155476217)

[VI.1.3 Sauvegarde et restauration 17](#_Toc155476218)

[VI.1.4 Intégration continue et déploiement 18](#_Toc155476219)

[VI.2 Inconvénients 18](#_Toc155476220)

[VI.2.1 Courbe d'apprentissage initiale 18](#_Toc155476221)

[VI.2.2 Dépendance à l'égard des services hébergés 18](#_Toc155476222)

[VI.2.3 Complexité des fusions et des conflits : 19](#_Toc155476223)

[VI.2.4 Dépendance à l'égard de la connectivité Internet : 19](#_Toc155476224)

[VII. Domaines d’application 20](#_Toc155476225)

[VII.1 Développement logiciel 20](#_Toc155476226)

[VII.2 Open source et développement communautaire 20](#_Toc155476227)

[VII.3 Gestion de projet 21](#_Toc155476228)

[VII.4 Documentation technique 21](#_Toc155476229)

[VII.5 Gestion de contenu 21](#_Toc155476230)

[VIII. Implémentation 22](#_Toc155476231)

[Conclusion GENeRALE 40](#_Toc155476232)

[Glossaire 41](#_Toc155476233)

[TABLE DES MATIERES 43](#_Toc155476234)