Gérez du code avec Git et GitHub

Git permet de suivre les modifications de votre code et d'organiser vos projets de développement. C'est un outil essentiel, que vous travailliez **seul**, en **équipe**, ou même sur **un projet en open source** !

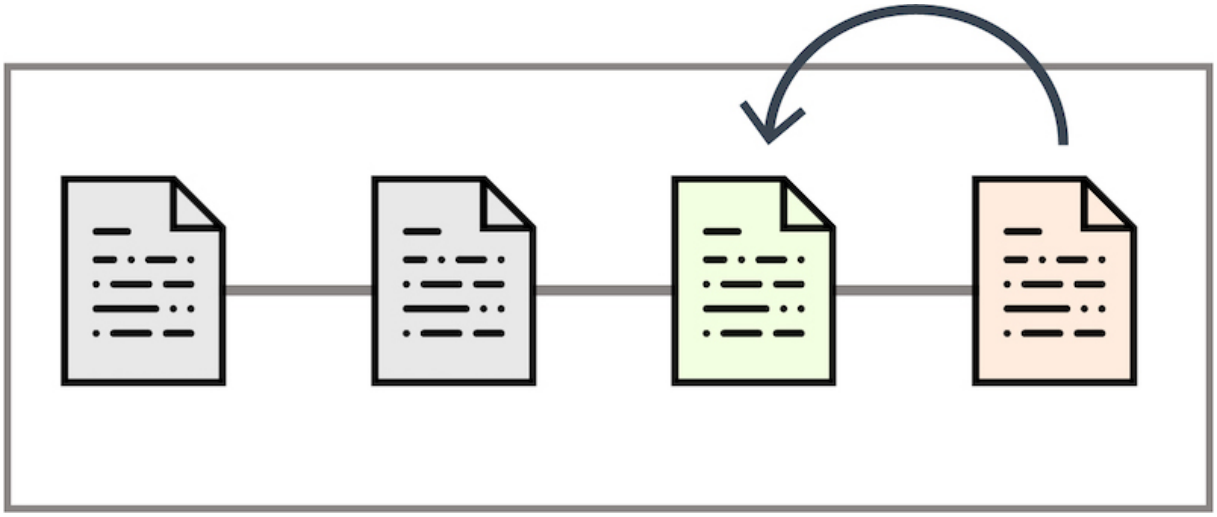
Dans ce cours, vous installerez et configurerez **Git** et son ami **GitHub**. Vous découvrirez les **commandes de base** avant d'apprendre à corriger vos erreurs simplement et efficacement.

Objectifs pédagogiques

- Installer et configurer Git et GitHub
- Utiliser les commandes de base de Git
- Corriger les erreurs courantes sur GitHub

Dans ce cours, je vais vous présenter l'outil indispensable à tout bon développeur : **le gestionnaire de versions**.

Imaginez, vous avez passé une semaine à écrire votre code, et donc produit votre première version, la V1. Votre collègue vous a fait des suggestions pour l'améliorer. Vous allez donc créer une deuxième version de votre code, la V2. Pour garder un historique de votre travail et éviter les mauvaises surprises (bugs, perte de votre travail), vous aurez besoin d'un **gestionnaire de versions**.



Le contrôle de versions

Qu'est-ce qu'un gestionnaire de versions ?

Alors, qu'est-ce qu'un gestionnaire de versions ?

Un gestionnaire de versions est un programme qui permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers.

L'action de contrôler les versions est aussi appelée "versioning" en anglais, vous pourrez entendre les deux termes.

Le gestionnaire de versions permet de garder en mémoire :

- chaque modification de chaque fichier ;
- pourquoi elle a eu lieu ;
- et par qui !

Si vous travaillez seul, vous pourrez garder l'historique de vos modifications ou revenir à une version précédente facilement.

Si vous travaillez en équipe, plus besoin de mener votre enquête ! Le gestionnaire de versions fusionne les modifications des personnes qui travaillent simultanément sur un même fichier. Grâce à ça, vous ne risquez plus de voir votre travail supprimé par erreur !

Cet outil a donc trois grandes fonctionnalités :

1. Revenir à une version précédente de votre code en cas de problème.
2. Suivre l'évolution de votre code étape par étape.
3. Travailler à plusieurs sans risquer de supprimer les modifications des autres collaborateurs.

Dans ce cours, vous allez apprendre à utiliser le gestionnaire de versions Git.

Git est de loin le système de contrôle de versions le plus largement utilisé aujourd'hui.

C'est un programme qui a une structure **décentralisée**.

Qu'est-ce que cela veut dire ?

Avec Git, l'historique complet du code n'est pas conservé dans un unique emplacement. Chaque copie du code effectué correspond à un nouveau dépôt dans lequel est conservé l'historique des modifications.

La maîtrise de Git est très souvent demandée lors d'un recrutement, c'est pourquoi il est essentiel de le maîtriser. 

Pourquoi est-ce utile dans le travail d'équipe ?

Prenons un exemple concret ! Alice et Bob travaillent sur le même projet depuis un mois. Tout se déroulait à merveille jusqu'à hier : le client leur a demandé de livrer leur projet en urgence. Alice a réalisé en vitesse les dernières modifications, enregistré les fichiers et envoyé le tout au client.

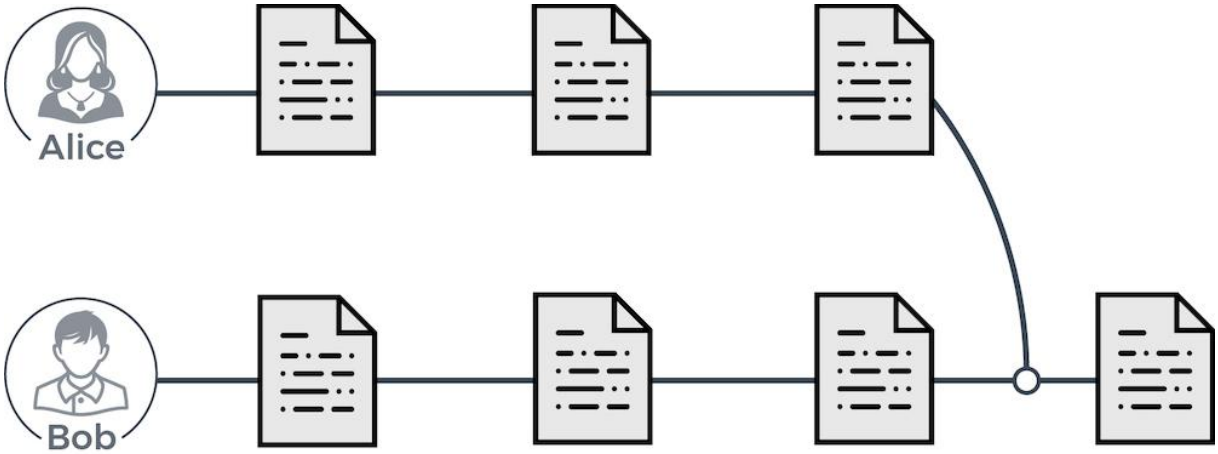
Le lendemain, le client appelle, très énervé : rien ne fonctionne. Alice et Bob ne comprennent pas. Ils s'étaient pourtant bien réparti les tâches et tous les deux avaient fait correctement leur travail.

Le problème ? Sans s'en rendre compte, Alice a écrasé le code de Bob en effectuant ses modifications. Bob n'a pas copié son travail en local, il a donc codé pendant un mois pour rien car il lui est impossible de récupérer son travail.

"En local" signifie sur votre machine, par opposition à "en ligne".

Tout cela aurait pu être évité avec le gestionnaire de versions !

Si Alice et Bob avaient initialisé Git pour leur projet, ils auraient pu modifier leurs fichiers, envoyer et recevoir les mises à jour à tout moment, sans risque d'écraser les modifications de l'autre. Les modifications de dernière minute n'auraient donc pas eu d'impact sur la production finale !



Alice et Bob ont initialisé Git. Ils travaillent chacun sur leur partie du code. Quand ils les regroupent, leurs versions précédentes sont archivées.

Git ou GitHub ? Quelle est la différence ?

Git et GitHub sont deux choses différentes.

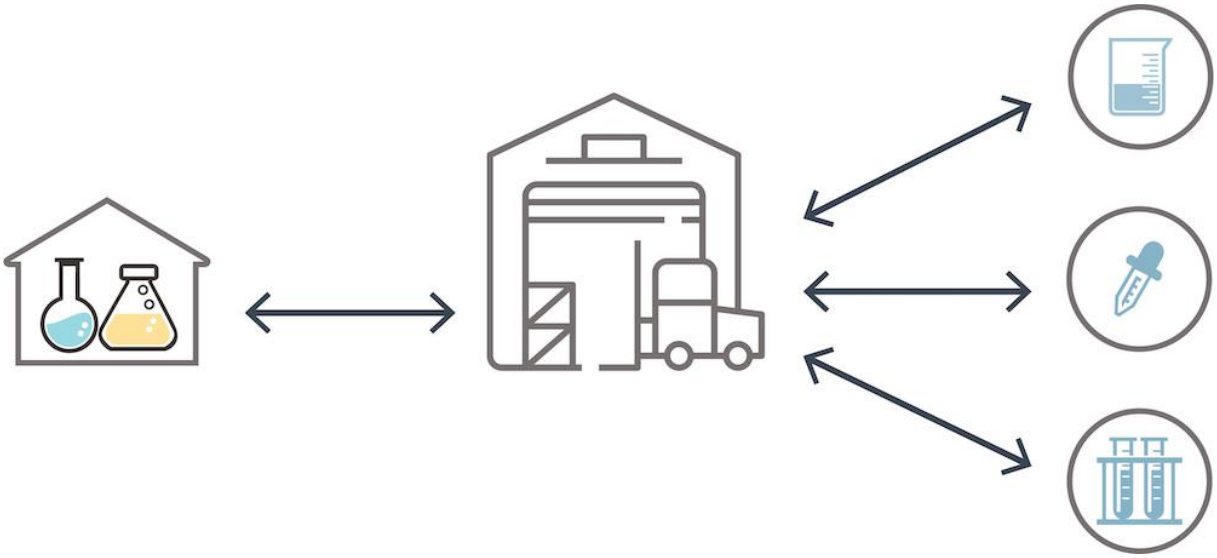
Git est un gestionnaire de versions. Vous l'utiliserez pour créer un dépôt local et gérer les versions de vos fichiers.

GitHub est un service en ligne qui va héberger votre dépôt. Dans ce cas, on parle de **dépôt distant** puisqu'il n'est pas stocké sur votre machine.

C'est un peu confus ? Pas de panique, prenons un exemple pour illustrer cela.

Imaginez, vous participez à la préparation d'un parfum. Chez vous, vous créez la base du parfum en mélangeant différents ingrédients. Puis vous envoyez cette base dans un entrepôt où il sera stocké. Cette base de parfum pourra être distribuée telle quelle ou être modifiée en y ajoutant d'autres ingrédients.

Eh bien, c'est la même chose pour Git et GitHub. Git est la préparation que vous avez réalisée chez vous, et GitHub est l'entrepôt où elle peut être modifiée par les autres ou distribuée.



Avec Git, vous préparez votre code. Avec GitHub, vous stockez votre code.

En résumé

- Un gestionnaire de versions permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers.
- Git est un gestionnaire de versions tandis que GitHub est un service en ligne qui héberge les dépôts Git. On parle alors de *dépôt distant*.

Saisissez l'utilité des dépôts distants sur GitHub

Faites la différence entre dépôt local et dépôt distant

Avant tout, avez-vous bien compris ce qu'était un dépôt ?

Un dépôt est comme un dossier qui conserve un historique des versions et des modifications d'un projet. Il peut être local ou distant.

Dans la documentation en ligne ou en milieu professionnel, on parle souvent de repository, qui est la traduction anglaise du terme "dépôt".

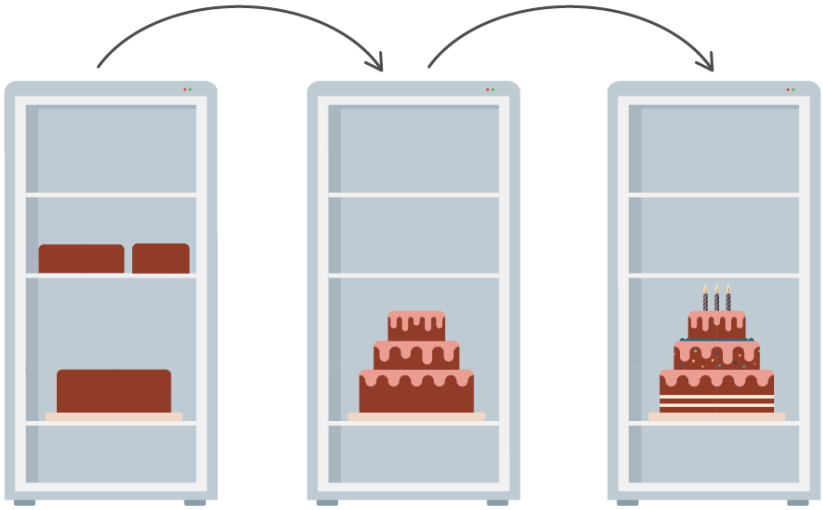
Le dépôt local

Un **dépôt local** est un entrepôt virtuel de votre projet. Il vous permet d'enregistrer les versions de votre code et d'y accéder au besoin.

Pour illustrer cette idée, prenons l'image de la réalisation d'un gâteau. Pour faire un gâteau, vous allez réaliser les étapes suivantes :

- préparer la pâte du gâteau ;
- stocker cette pâte au réfrigérateur ;
- réaliser la crème et en garnir la pâte ;
- stocker le gâteau assemblé au réfrigérateur ;
- décorer votre gâteau ;
- remettre le gâteau au réfrigérateur.

Dans cet exemple, le réfrigérateur est comme un dépôt local : c'est l'endroit où vous stockez vos préparations au fur et à mesure.



Utilisez un dépôt local comme un réfrigérateur et préparez votre gâteau !

Un **dépôt local** est utilisé de la même manière ! On réalise une version, que l'on va petit à petit améliorer. Ces versions sont stockées au fur et à mesure dans le dépôt local.

Le dépôt distant

Le **dépôt distant** est un peu différent. Il permet de stocker les différentes versions de votre code afin de garder un **historique délocalisé**, c'est-à-dire un historique hébergé sur Internet ou sur un réseau. Vous pouvez avoir plusieurs dépôts distants avec des droits différents (lecture seule, écriture, etc.).

Ben oui, imaginez que votre PC rende l'âme demain, vous aurez toujours vos super programmes sur un dépôt distant ! C'est pourquoi, je vous conseille de toujours commencer par copier vos sources sur un dépôt distant lorsque vous commencez un

nouveau projet, avec GitHub par exemple ! Vous pourrez aussi les rendre publics, et chacun pourra y ajouter ses évolutions.

Sur un dépôt public, les personnes pourront collaborer à votre projet alors que sur un dépôt privé, vous seul aurez accès à votre travail et déciderez des personnes qui pourront ou non avoir accès à votre repository !

Le dépôt distant est donc un type de dépôt indispensable si vous travaillez à plusieurs sur le même projet, puisqu'il permet de centraliser le travail de chaque développeur. C'est aussi sur le dépôt distant que toutes les modifications de tous les collaborateurs seront fusionnées.

Alors, pourquoi créer une copie locale ?

Tout simplement car votre dépôt local est un clone de votre dépôt distant. C'est sur votre dépôt local que vous ferez toutes vos modifications de code.

Ainsi, les dépôts sont utiles si :

- vous souhaitez conserver un historique de votre projet ;
- vous travaillez à plusieurs ;
- vous souhaitez collaborer à des projets open source ;
- vous devez retrouver par qui a été faite chaque modification ;
- vous voulez savoir pourquoi chaque modification a eu lieu.

Un projet open source est un projet dont le contenu – ici le code – est conçu pour être accessible au public : n'importe qui peut voir, modifier et utiliser le code à sa convenance.

Quelle plateforme utiliser pour héberger votre code ?

Il existe plusieurs outils intéressants (GitHub, GitLab, Bitbucket), et nous allons donc voir les principaux avantages et inconvénients de chacun.

GitHub

C'est mon préféré, mais chuttt !! GitHub est un outil de communication et de collaboration entre plusieurs développeurs (ou toute autre personne qui écrit du texte). C'est une interface web créée pour faciliter l'interaction avec Git.

L'avantage de GitHub, c'est que depuis quelques années, il est devenu le **book/portfolio des développeurs** ! Dans beaucoup de processus de recrutement, on vous demandera maintenant votre lien GitHub ! Si ça, c'est pas un argument de taille ! Il permet de mettre en avant la qualité de son code, et ainsi montrer ses capacités et sa plus-value lorsque l'on recherche un emploi. GitHub est considéré comme un véritable

réseau social, et permet de contribuer à des projets open source. Il fonctionne par abonnement, mais pas de panique, il y a un abonnement gratuit qui est déjà très bien.

GitLab

GitLab est la principale alternative à GitHub depuis le rachat de GitHub par Microsoft ! Les anti-Microsoft ont même lancé le hashtag *#MovingToGitLab* ! GitLab propose une version gratuite hebergée par ses soins ou sur vos propres serveurs. Il existe aussi des versions payantes avec plus d'options.

Bitbucket

Bitbucket est la version de Atlassian. Elle plaira aux habitués de la gestion de projet sous Atlassian. Bitbucket conviendra aussi bien aux étudiants ou petites équipes qu'aux grands groupes. Une version gratuite est disponible.

Vous avez fait votre choix ? Nous étudierons dans ce cours la solution GitHub, qui est la plus plébiscitée par les développeurs.

En résumé

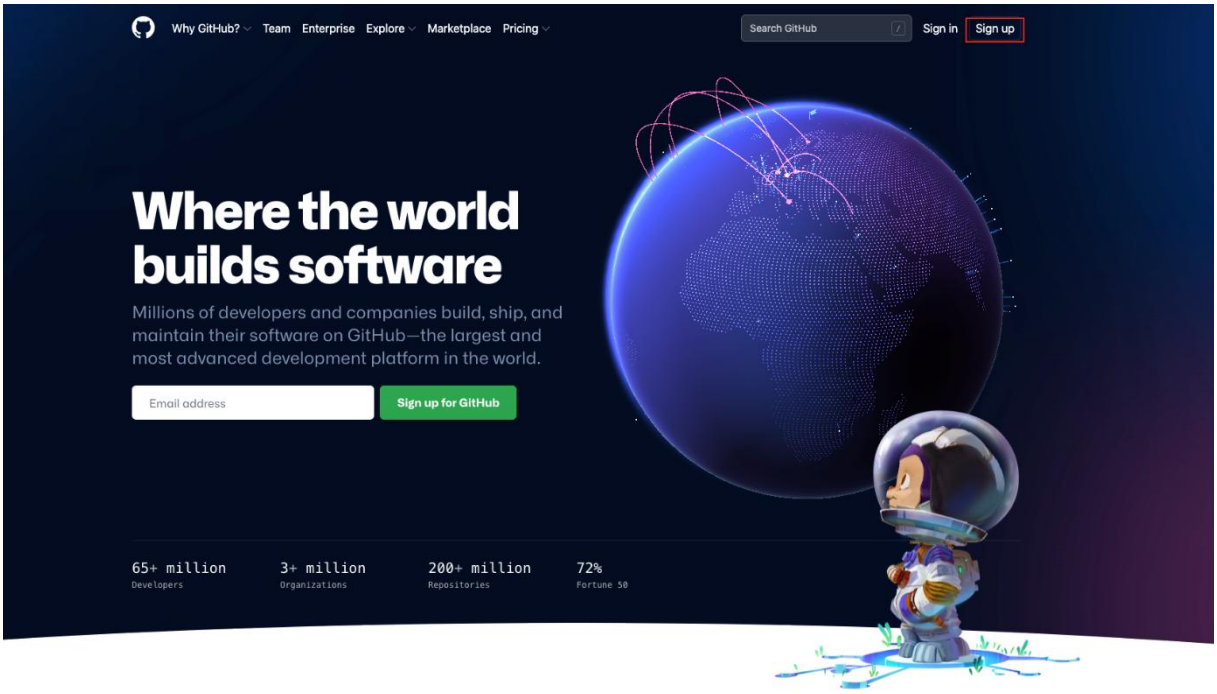
- Un dépôt est comme un dossier qui conserve un historique des versions et des modifications d'un projet. Il est essentiel pour travailler en équipe ou collaborer à un projet open source.
- Un dépôt local est l'endroit où l'on stocke, sur sa machine, une copie d'un projet, ses différentes versions et l'historique des modifications.
- Un dépôt distant est une version dématérialisée du dépôt local, que ce soit sur Internet ou sur un réseau. Il permet de centraliser le travail des développeurs dans un projet collectif.
- Il existe plusieurs services en ligne pour héberger un dépôt distant, GitHub étant l'un des plus populaires.

Démarrez votre projet avec GitHub

Créez un compte GitHub

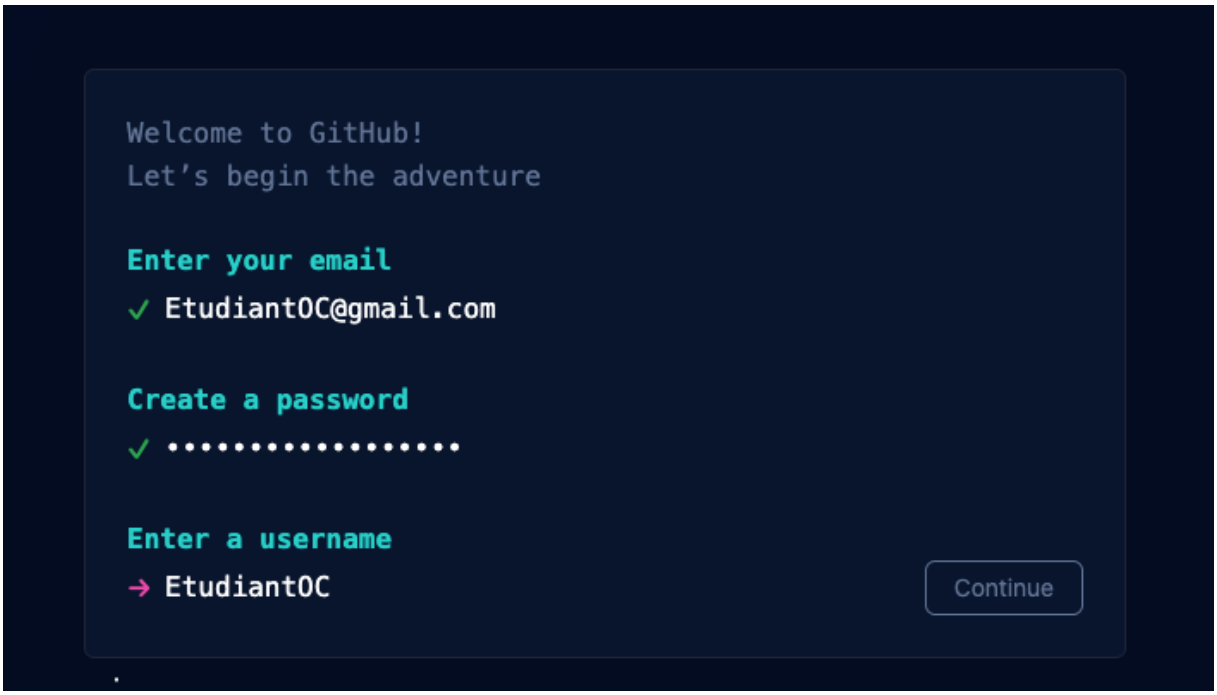
Comme nous l'avons vu dans le chapitre précédent, GitHub est un service en ligne qui va vous permettre d'héberger vos dépôts distants.

Pour créer votre compte GitHub, rendez-vous sur la page d'accueil et cliquez sur Sign up.



Accédez à la page d'accueil de GitHub et cliquez sur Sign up en haut à droite de l'écran

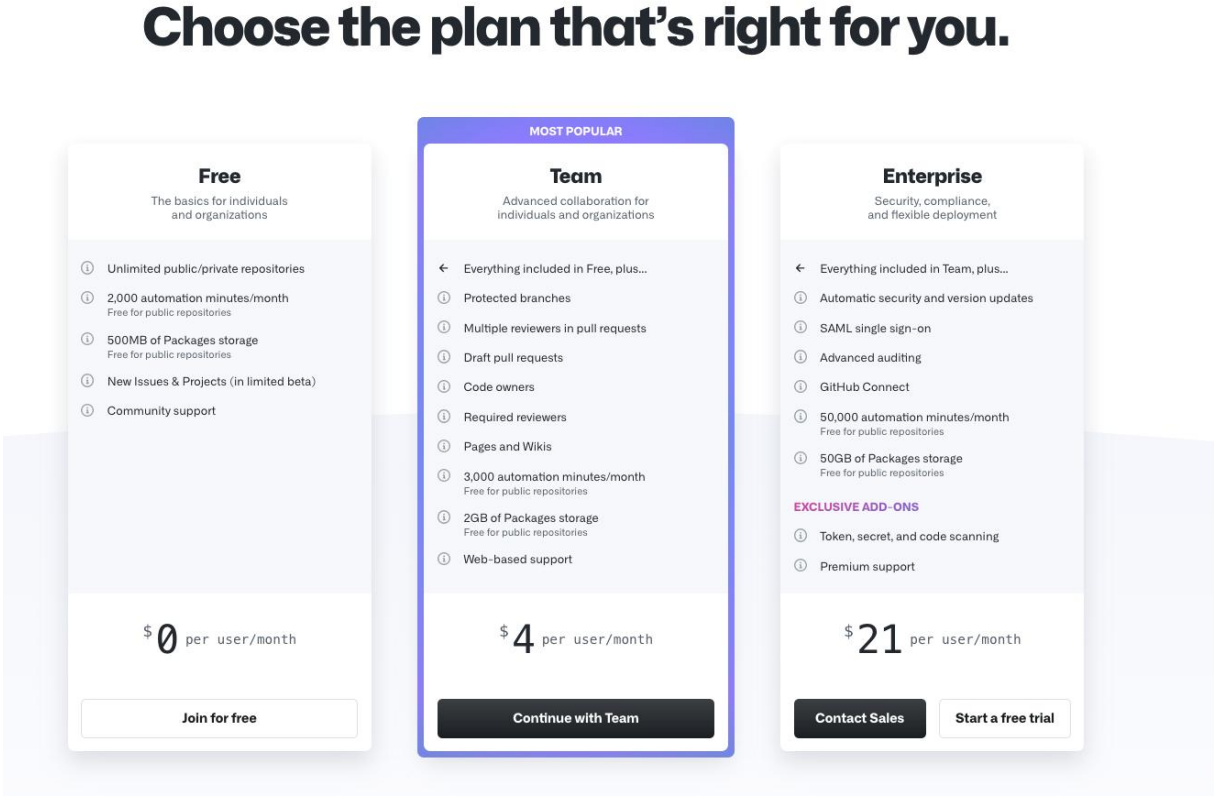
On vous demandera alors de renseigner un e-mail, un mot de passe et un nom d'utilisateur.



Renseignez un e-mail, un mot de passe et un nom d'utilisateur pour vous inscrire

Un code de vérification vous sera envoyé sur votre adresse e-mail afin de confirmer votre identité.

Et voilà, vous êtes à présent inscrit sur GitHub ! Par défaut, GitHub est gratuit. Mais sachez qu'il existe également des offres payantes si vous décidez de passer à la vitesse supérieure.



Découvrez les différents abonnements GitHub

[À vous de créer votre compte GitHub!]

Faites un petit tour de GitHub

GitHub est assez facile à prendre en main et simple d'utilisation.

Votre tableau de bord

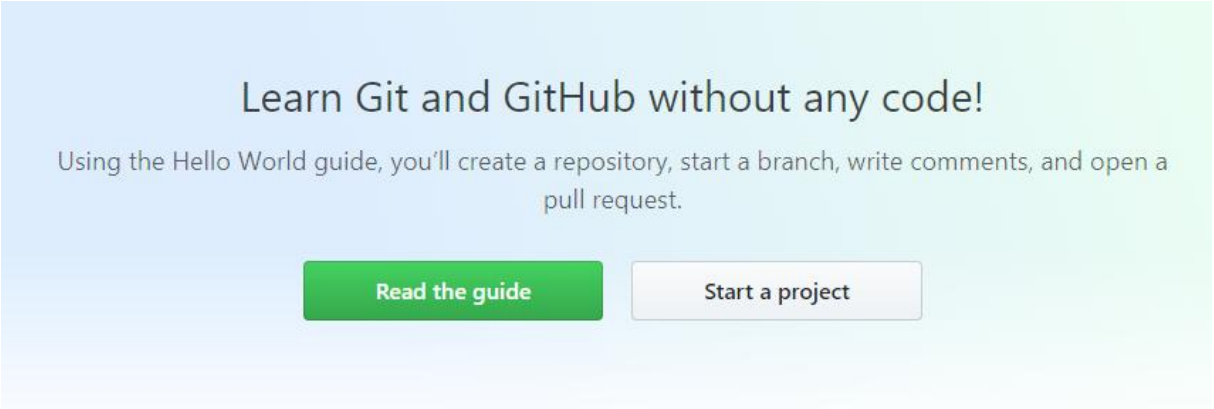
Vous pouvez consulter votre **tableau de bord** personnel pour :

- suivre les problèmes et extraire les demandes sur lesquelles vous travaillez ou que vous suivez ;
- accéder à vos principaux repositories et pages d'équipe ;
- rester à jour sur les activités récentes des organisations et des repositories auxquels vous êtes abonné.

L'interface repository

L'interface Repositories est l'emplacement où vous pourrez créer et retrouver vos dépôts existants.

Pour créer un projet, il suffit de cliquer sur "Start a project".

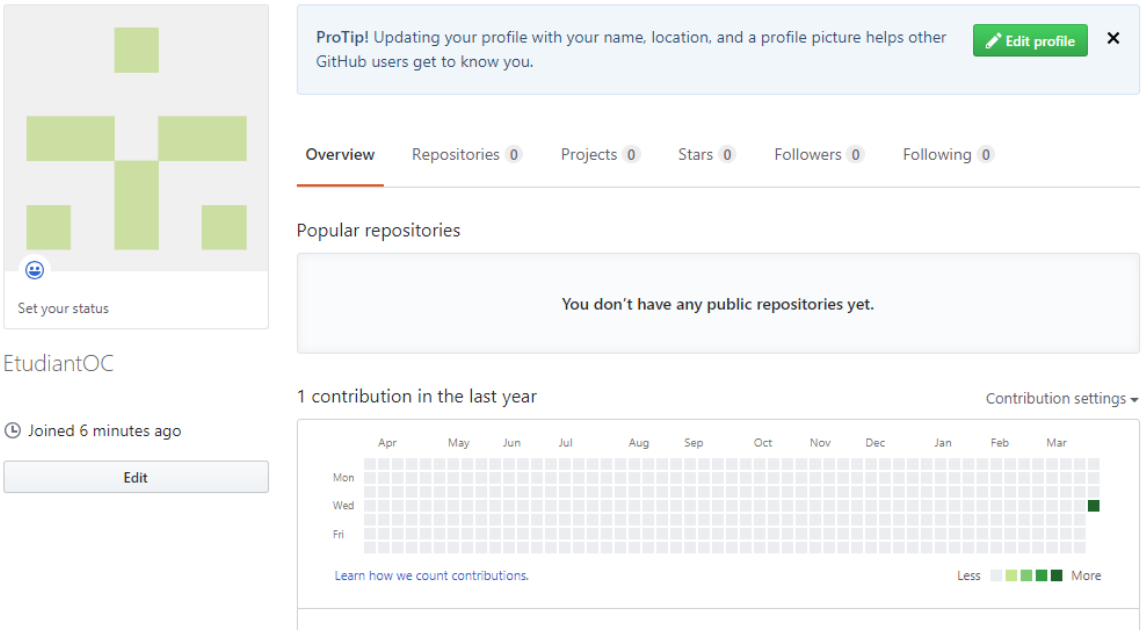


Cliquez sur "start a project"

Votre profil

Sur votre profil, vous pourrez éditer vos informations, mais aussi voir le total de vos contributions sur les différents projets.

Les *contributions* sont toutes les actions sur des repositories que vous allez effectuer. Que ce soient vos repositories, ceux d'autres personnes ou des repositories publics.



Consultez votre profil

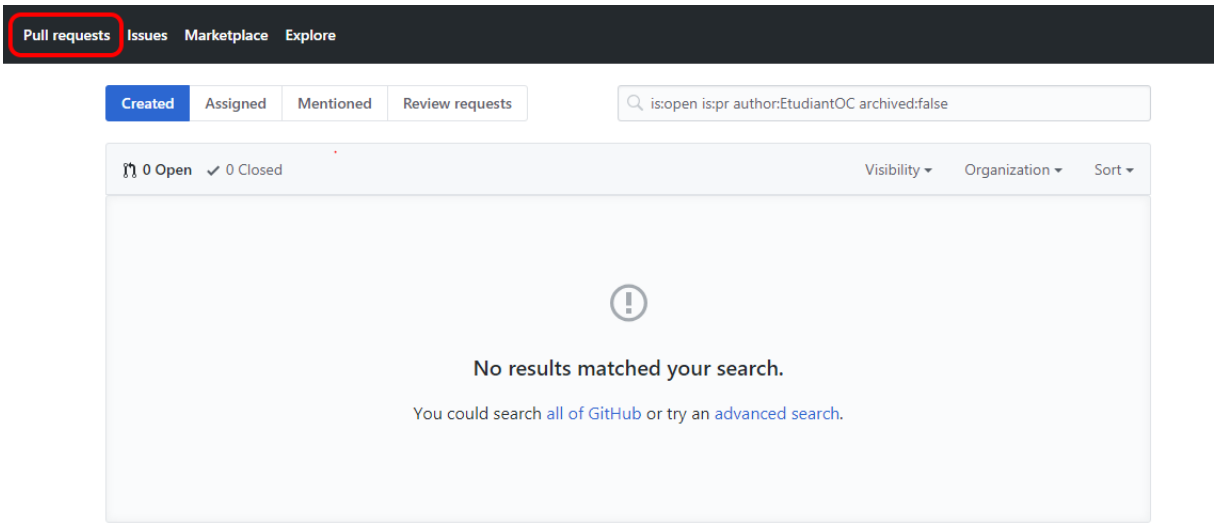
L'onglet Pull requests

L'onglet Pull requests, quant à lui, permet de faire des demandes de modifications réalisées sur le code.

Ah bon, comment ça marche ?

Les pull requests (ou *demandes de pull*), vous permettent d'informer les autres utilisateurs des modifications que vous avez appliquées à une branche d'un repository sur GitHub, et que vous voulez fusionner avec le code principal.

Pas de panique, nous reviendrons sur toutes ces notions plus tard. Pour l'instant il s'agit plutôt de vous situer sur l'interface de GitHub.

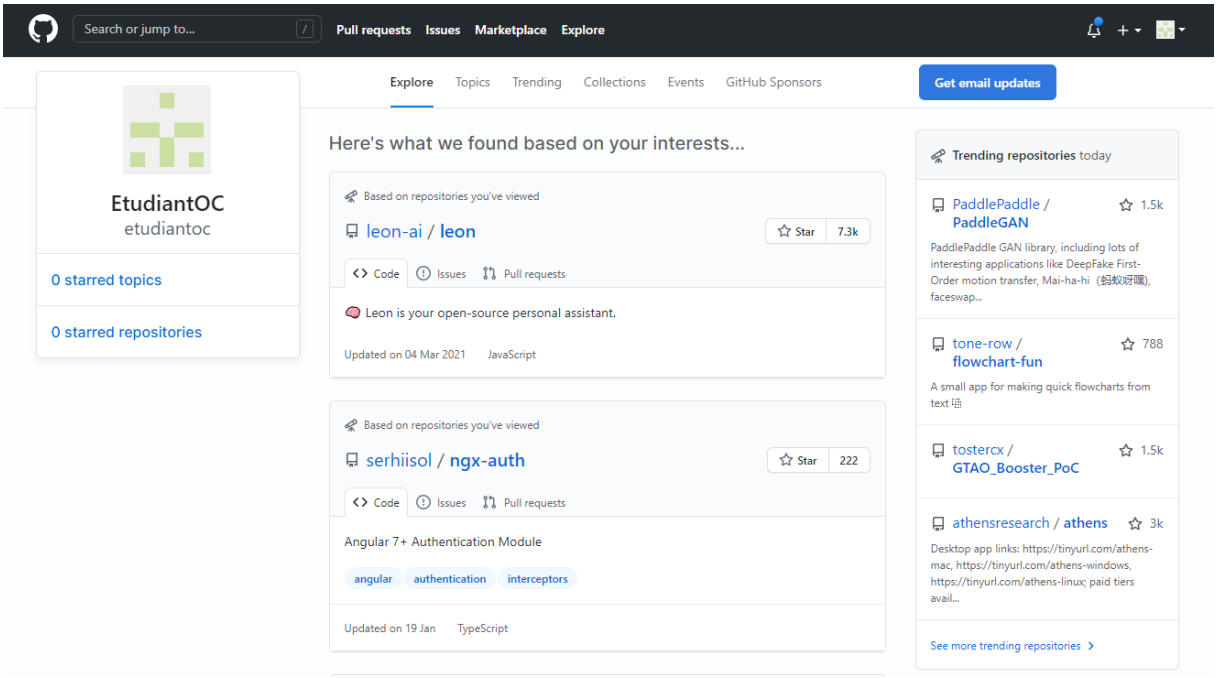


L'onglet Pull requests

La fonctionnalité Explore

Un des derniers points importants sur GitHub est la fonctionnalité **Explore**.

Via Explore, vous pourrez trouver de nouveaux projets open source sur lesquels travailler, en parcourant les projets recommandés, en vous connectant à la communauté GitHub et en recherchant des repositories par sujet ou par libellé.



La fonctionnalité Explore

Créez votre propre dépôt

Pour mettre votre projet sur GitHub, vous devez créer un **repository** (ou dépôt en français) dans lequel il pourra être installé.

Mais avant, j'aimerais aborder un sujet important : la sécurité.

Les repository contiendront votre code, mais aussi toute autre information que vous souhaitez stocker : certains les utilisent pour du texte, des images ou des listes (de sites, livres, films, etc.). Tout est possible !

Pour les repository publics, soyez prudents : toute information stockée est **accessible à tous et donc publique**.

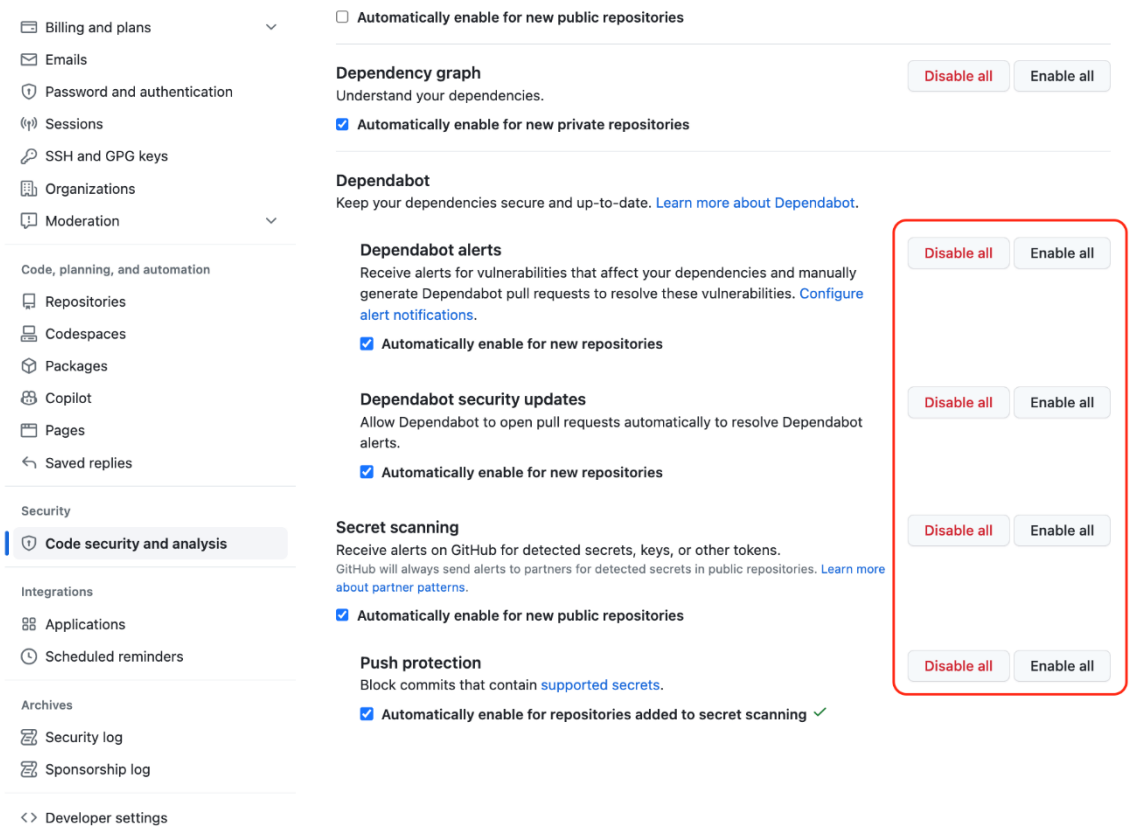
Il arrive souvent que par étourderie ou erreur certains utilisateurs ajoutent dans le repository des informations privées, comme des mots de passe, des identifiants, des informations de carte bancaire. Avant de créer votre premier repository, nous allons activer une fonctionnalité de GitHub qui se nomme : *Push Protection*.

Cela signifie que lorsque vous enverrez votre code sur GitHub, celui-ci scannera votre code afin de vérifier qu'il ne contient aucune information sensible comme des clés ou secrets. S'il y a des informations privées présentes dans votre code, Push Protection vous alertera avant de stocker votre code en ligne.

En l'activant dès maintenant, vous serez assuré qu'elle sera activée pour chaque nouveau repository créé, aussi bien public que privé. Si besoin, vous pourrez désactiver l'option dans les paramètres généraux ou au cas par cas dans les paramètres de votre repository. Si vous voulez en savoir plus, je vous invite à lire la documentation sur GitHub.

Cliquez sur votre photo, puis sur *Settings*.

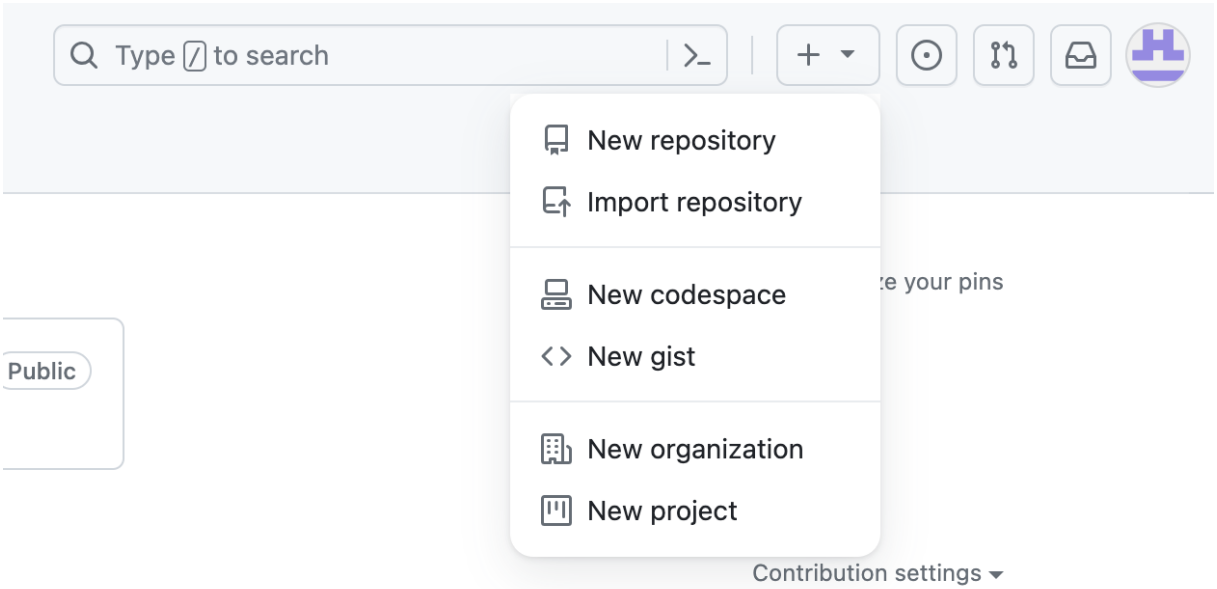
Puis cliquez sur *Code security and analysis*. Vous devriez avoir cette page :



Activez les fonctionnalités de sécurité pour vos repository: dependabot et secret scanning.

Cliquez sur *Enable All* ou cochez chaque option comme sur ce screenshot. L'option sera automatiquement activée pour tout nouveau repository. Parfait ! Il est temps de créer votre premier repository !

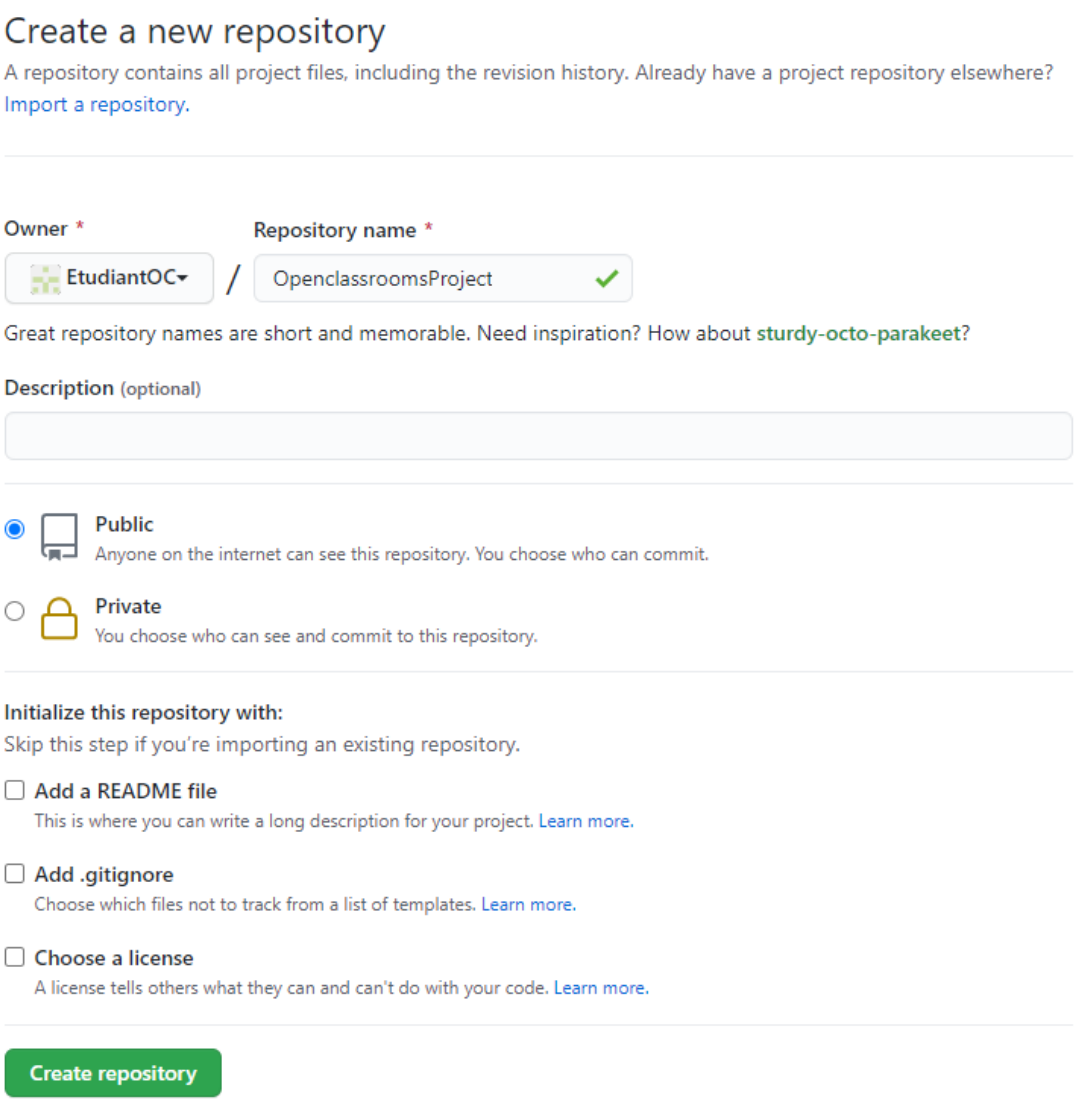
Cliquez sur le + dans le coin supérieur droit, pour faire apparaître l'option *New repository*.



Ajoutez un repository

Choisissez un nom simple pour votre dépôt. Dans ce cours, nous utiliserons “OpenclassroomsProject”.

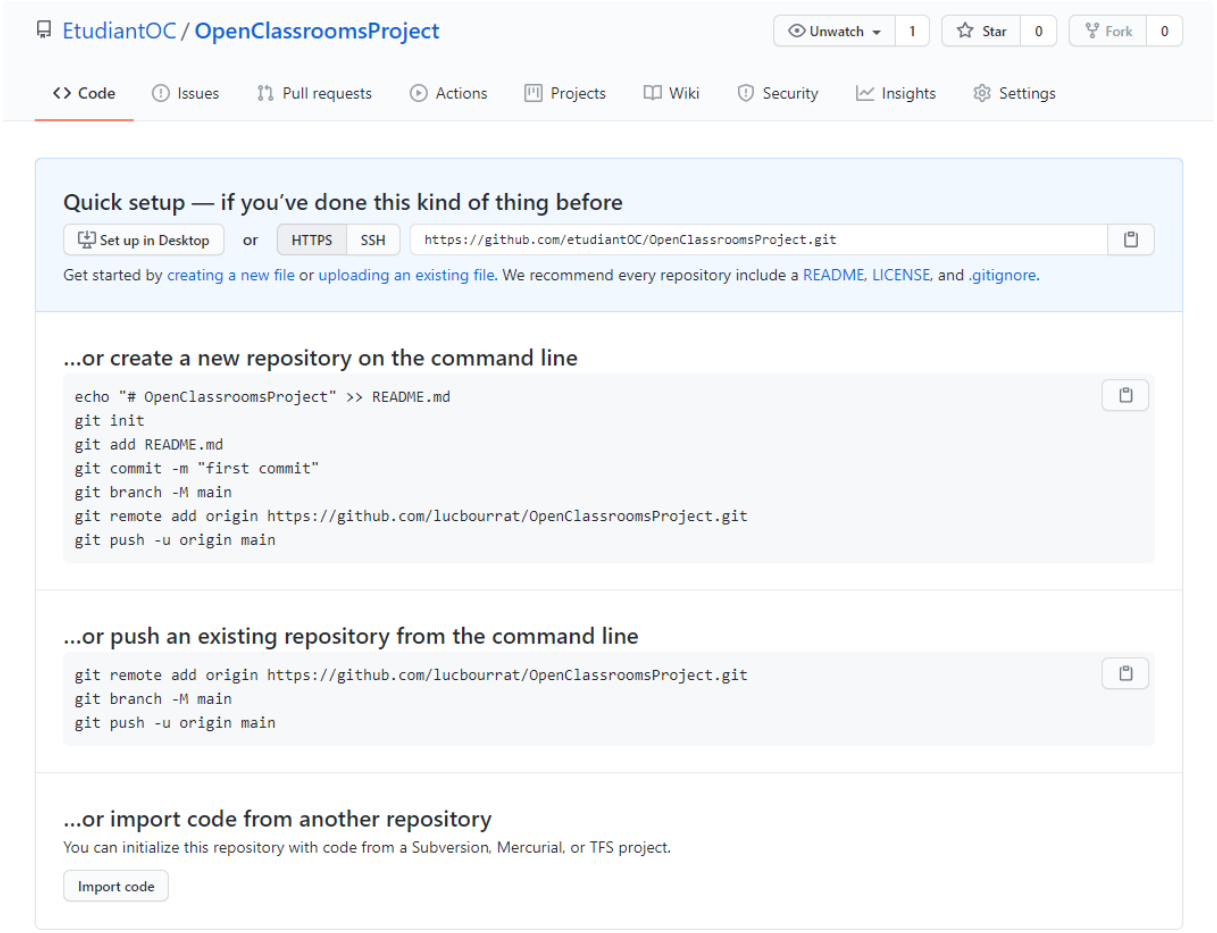
Puis, choisissez si vous souhaitez créer un dépôt public ou privé.



Créez un dépôt

README est un fichier qui indique les informations clés de votre projet : description, environnement à utiliser, dépendances possibles et droits d'auteurs. C'est un peu comme le mode d'emploi de votre projet.

gitignore est un fichier qui permet d'ignorer certains fichiers de votre projet Git. Nous reviendrons là-dessus plus tard.



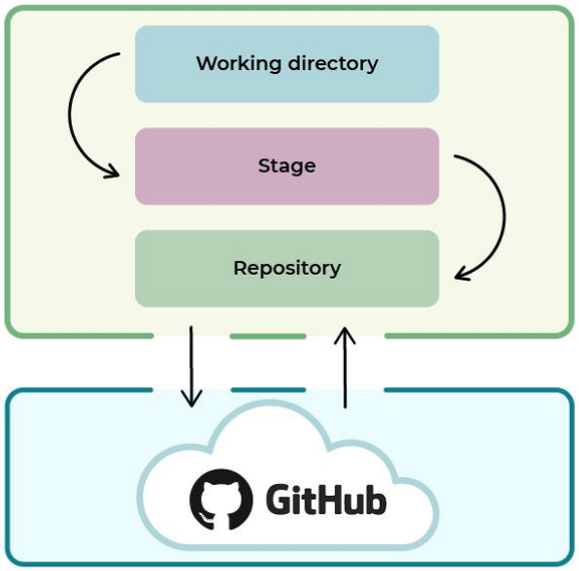
Cliquez ensuite sur “Create repository” pour créer un dépôt.

En résumé

- Pour démarrer un projet, vous devez créer votre compte GitHub.
- Vous pouvez suivre vos différents projets facilement grâce au tableau de bord.
- Pour mettre votre projet sur GitHub, vous devez créer un repository.

Fiche récap : Gérez du code avec Git et Github

Développement



Dépôt local

Dépôt distant

Configuration et initialisation 💻

`$ cd Documents/PremierProjet`
Pour vous positionner dans le dossier PremierProjet

`$ git init`
Pour initialiser un nouveau dépôt Git

`git clone URL_DU_REPO`
Pour cloner un dépôt existant à partir de l'URL fournie

Travailler avec des repos distant 🛰️

`git push`
Pour envoyer la nouvelle version sur le dépôt distant

`git pull`
Pour récupérer les dernières modifications du dépôt distant

Gestion des fichiers et des commits 📝

`git status`
Pour montrer l'état des fichiers

`$ git add fichier.html`
Pour ajouter des fichiers à l'index pour le prochain commit

`git commit -m "Message de commit"`
Pour créer un nouveau commit avec les fichiers ajoutés à l'index

Gestion des branches 🌱

`git branch`
Pour lister toutes les branches

`git branch NOM_DE_LA_BRANCHE`
Pour créer une nouvelle branche

`git checkout NOM_DE_LA_BRANCHE`
Pour changer de branche

`git merge NOM_DE_LA_BRANCHE`
Pour fusionner la branche spécifiée dans la branche actuelle

Historique et inspection 🔎

`git log`
Pour voir l'historique des commits

`git stash`
Pour enregistrer temporairement des modifications non indexées

`git stash apply`
Pour appliquer les modifications enregistrées avec stash