**Utilisez Git et GitHub pour vos projets de développement**

Vous êtes un développeur débutant et vous souhaitez vous lancer sur **des projets plus ambitieux**?

Vous êtes au bon endroit !

Les projets informatiques sont parfois chaotiques, alors comment garder une trace de tout son travail ?  
Comment revenir à une version précédente ?  
Et comment réparer ses erreurs ?

Dans ce cours, je vous donnerai toutes les réponses pour gérer et déployer tous vos projets informatiques avec **Git**. :D  
  
Git permet de suivre les modifications et organiser votre projet. 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. Vous découvrirez la **structure de l'arbre Git** et comment garder vos **branches** propres. Enfin, vous apprendrez à utiliser **GitFlow** et à gérer les demandes de **pull**.

À la fin de ce cours, vous serez capable de :

* utiliser les commandes de base de Git ;
* corriger les erreurs courantes sur GitHub ;
* gérer plusieurs versions sur GitHub ;
* collaborer grâce à GitHub en utilisant les workflows.

Prérequis : pour suivre ce cours, vous devez savoir écrire une ligne de commande.

## Découvrez la magie du contrôle de versions

Dans ce cours, je vais vous présenter la clé à molette de tout bon développeur : **le gestionnaire de code source**.

Quel développeur n’a jamais été confronté à un de ces **problèmes** :

Ma modif n’a pas marché et j’ai oublié de sauvegarder une copie de mon code avant de le bidouiller...

Qui a touché à mon fichier ? Il présente un bug maintenant !

À quoi servent ces nouveaux fichiers ?

Ne touche surtout pas à ce fichier, je suis en train de le modifier.

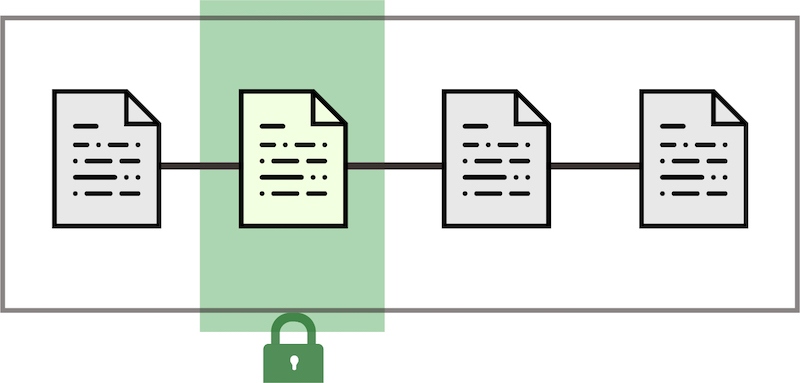
Et bien d’autres problèmes, que tout développeur sera susceptible de rencontrer sur son parcours. Si maintenant, je vous disais que vous ne rencontrerez plus jamais ces problèmes grâce au contrôle de versions…. C’est parti ! :soleil:

### Qu'est-ce que le contrôle de versions ?

Le nom n’a rien de magique, je vous l’accorde, et pourtant ce petit outil vous deviendra vite **indispensable** ! Si, si !

Mais déjà, qu’est-ce qu’un contrôleur de versions ?

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

Contrôleur de versions

**Si vous êtes seul** à travailler sur votre projet,  le contrôle de versions vous sera d’une grande aide. Il vous permettra de garder l’historique des modifications de tous vos fichiers. Je vous conseille donc fortement de l’utiliser ! Le contrôle de versions permet de garder en mémoire chaque modification de chaque fichier qui a eu lieu, pourquoi elle a eu lieu et par qui ! Plus besoin de mener son enquête pour connaître les dernières modifications. Le contrôle de versions permet d’assembler les modifications de deux personnes travaillant simultanément sur un même fichier, afin d’éviter d’écraser le travail des autres.

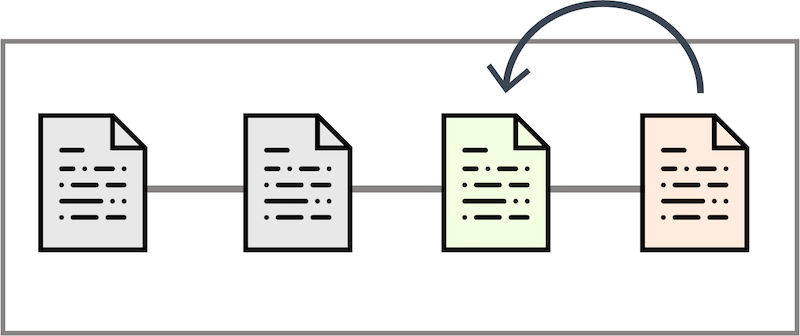
 Ces outils ont donc trois grandes fonctionnalités :

* travailler à plusieurs sans risquer de supprimer les modifications des autres collaborateurs ;
* revenir en arrière en cas de problème ;
* suivre l’évolution étape par étape d’un code source pour retenir les modifications effectuées sur chaque fichier.

Prenons un exemple simple, imaginez que vous remportiez un projet de création d’un formulaire de recrutement. Youpiii ! :D

Vous avez réussi à créer le formulaire et votre client est super content ! Voyant que son projet a bien avancé, il a l’idée d’ajouter une fonctionnalité d’envoi de mails. Il vous demande donc de le faire au plus vite. Ni une ni deux, vous vous lancez dans le code.

Malheureusement, la modification que vous avez faite a entraîné des **régressions** et le formulaire ne fonctionne plus. Votre client n’est pas content et souhaite que vous fassiez un retour arrière sur votre code pour que le formulaire fonctionne de nouveau. Avec un contrôle de versions, il n’y a aucun problème. :) Vous remettez la version 1 en production et votre client est satisfait. ;)

Retour à une ancienne version

L’intérêt de ce type d'outil est donc de pouvoir revenir sur n’importe quelle version en cas de bug dans l’application.

L’utilisation de ces logiciels est donc indispensable pour un développeur digne de ce nom !

Le contrôle des versions est un outil extrêmement utile dans le cadre **d'un développement personnel** comme dans le cadre **d’un projet mutualisé**.

Admettons que Henry modifie le fichier X, mais que vous aussi ayez modifié ce fichier ; pas de panique, avec le contrôle de versions, vous pourrez assembler vos modifications. C’est pas magique ?

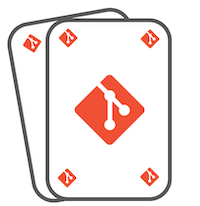
Dans ce cours, nous allons apprendre à utiliser le contrôleur de versions Git.

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

Par sa structure **décentralisée**, Git illustre parfaitement ce qu'est un système de contrôle de versions décentralisé. Plutôt que de consacrer un seul emplacement pour l'historique complet des versions du logiciel, dans Git, chaque copie de travail du code est également un dépôt qui contient l'historique complet de tous les changements.

La maîtrise de Git est très souvent demandée lors d’un recrutement, c’est pourquoi il est essentiel de le maîtriser, mais pas de panique ! :)

À la fin de ce cours, vous serez des as de Git (ahahaha vous avez compris ? As de pique/As de Git... OK, je l’admets, ce n’était pas drôle... ). :honte:



As de Git

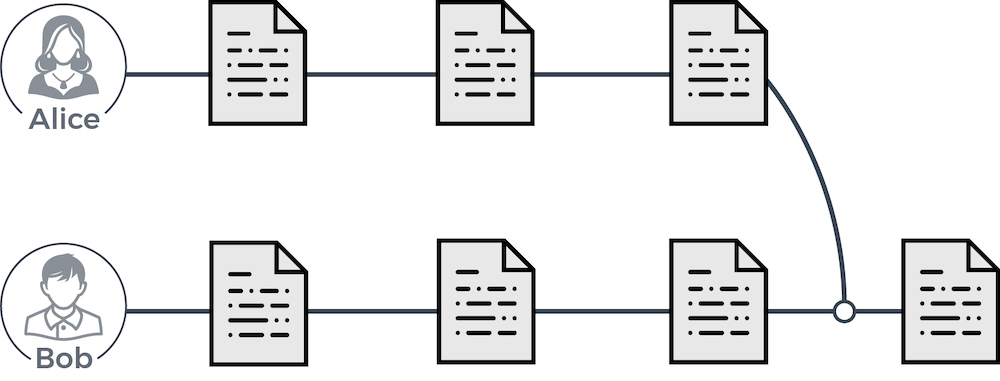
### Pourquoi est-ce utile dans le travail d’équipe ?

Prenons un exemple concret !  Alice et Bob travaillent sur un même projet depuis un mois et jusque là tout se passait bien. Hier, leur client leur a demandé de livrer en production leur travail en urgence. Alice a réalisé au plus vite les dernières modifications, a enregistré les fichiers et a envoyé le tout au client.

Le lendemain, le client les appelle, très énervé : rien ne fonctionne comme prévu. Alice et Bob ne comprennent pas, ils avaient séparé les tâches et tous les deux avaient fait correctement le travail. Oui mais Alice, sans le savoir, a écrasé le code qu’avait réalisé Bob lorsqu’elle a fait ses modifications de dernière minute ; et en enregistrant, elle a perdu le travail de Bob. Bob n’ayant pas de copie en local, il a travaillé 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".

Cela aurait pu être évité avec le gestionnaire de codes sources ! Maintenant, reprenons cet exemple concret avec un gestionnaire de codes sources (Git). Alice et Bob travaillent sur un même projet et ont **initialisé Git** pour leur projet. Grâce à Git, chacun modifie ses fichiers, et chacun peut envoyer et recevoir les mises à jour des fichiers à n’importe quel moment, et cela sans écraser les modifications de l’autre. Des modifications même en urgence n’auront aucun impact sur le travail de l’autre !

Travail mutualisé d'Alice et Bob

### Comment l’utiliser dans des projets open source ?

Un des aspects passionnants lorsque vous faites du développement, c'est que vous pouvez apporter votre pierre à plein d'édifices en contribuant à des projets open source.

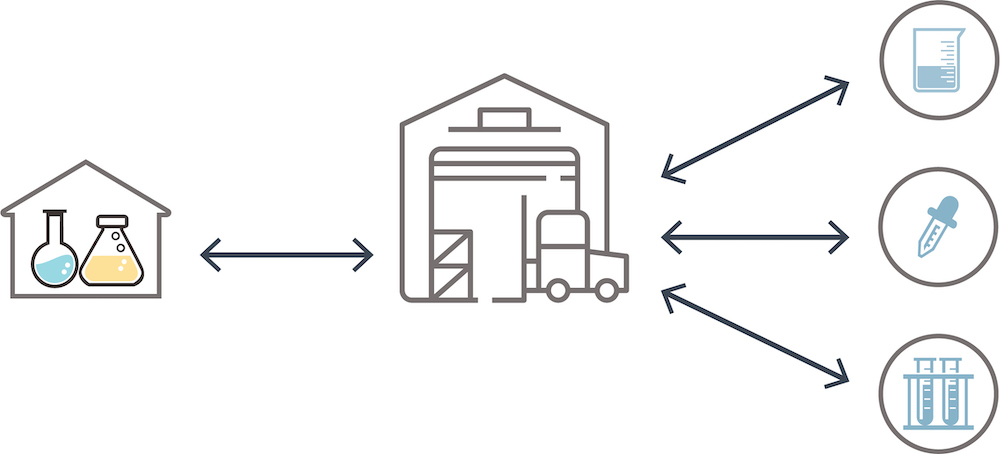
Open source signifie que le code source d'un logiciel est public et accessible. Le logiciel en question peut alors être modifié et diffusé par n'importe quel individu.

Travailler sur un projet open source est passionnant et permet de développer rapidement ses compétences. Que ce soient le codage, la conception de l’interface utilisateur, la conception graphique, la rédaction ou l’organisation, si vous cherchez de la pratique, il y a une tâche pour vous sur un projet open source.

Git ou GitHub ? Quelle est la différence ?

Git est l'outil qui nous permet de créer un dépôt local et de gérer les versions de nos fichiers, alors que GitHub est un service en ligne qui va héberger notre dépôt, qui sera du coup distant (puisqu'il ne sera pas sur notre machine).

Prenons un petit exemple. Admettons que nous fassions une préparation pour du parfum. Chez nous, nous allons créer la base du parfum en mélangeant divers ingrédients. Ensuite, nous allons envoyer notre base de parfum à un entrepôt où ils pourront soit le distribuer tel quel, soit le modifier en y ajoutant des ingrédients, des arômes, etc. Eh bien, c'est la même chose que Git et GitHub. Git est la préparation que vous avez réalisée chez vous, et GitHub est l'entrepôt où il peut être modifié par les autres ou distribué. :)

Git et GitHub

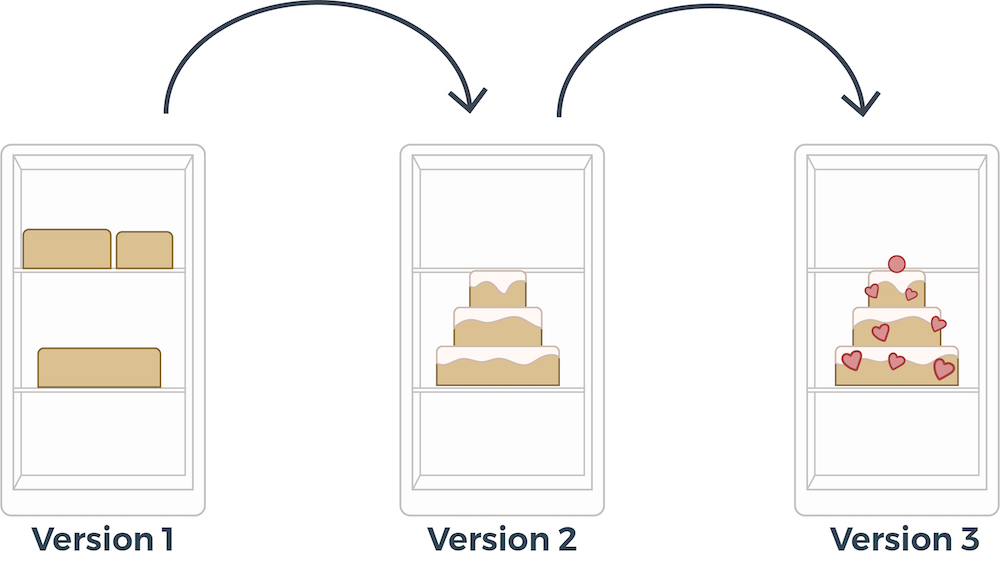
Prenons, par exemple, un projet réalisé sous GitHub. La plupart du temps, il faut suivre les étapes suivantes pour collaborer sur un projet open source :

* le premier réflexe est de regarder la documentation. Il y a souvent tout un tas d’informations sur la manière de collaborer au mieux au projet ;
* dans un second temps, vous devrez rapatrier le dépôt distant sur votre dépôt local. Sur votre dépôt local, vous pourrez réaliser vos modifications ;
* une fois toutes les modifications réalisées, vous pourrez envoyer vos modifications en ajoutant des messages de description. Il faut que la personne gérant le dépôt distant comprenne les modifications que vous avez faites. En allant de nouveau sur le dépôt distant, vous pourrez maintenant soumettre vos modifications.

## aisissez l'utilité des dépôts distants sur GitHub

### Faites la différence entre dépôt local et dépôt distant

Prenons l'image de la réalisation d'un gâteau. On va dire que le réfrigérateur est le dépôt local. C'est l'endroit où l'on va stocker nos préparations au fur et à mesure. Dans un premier temps, on réalise la pâte, on la stocke au réfrigérateur, puis on réalise la crème, on l'assemble avec la pâte et on stocke l'ensemble au réfrigérateur. Finalement, on réalise la décoration du gâteau, on finalise notre gâteau en y ajoutant les décorations  et on le remet au réfrigérateur.

Fonctionnement du dépôt local : rangez votre frigo !

Voilà le fonctionnement de notre **dépôt local** ! On réalise une version, que l'on va petit à petit venir améliorer en stockant toutes ces versions.

On peut aussi prendre l'image d'un livre d'école ! Vous avez sûrement déjà remarqué que sur vos livres, il y a souvent écrit première édition ou cinquième édition. L'éditeur a réalisé une version et petit à petit, il a corrigé l'orthographe, ou modifié le contenu. Eh bien, les versions dans Git fonctionnent de la même manière. On va réaliser une première version, que l'on va venir améliorer au fil du temps.

Le **dépôt distant**est un peu différent. Il permet de stocker certaines versions qu'on lui aura envoyées, afin de garder un historique délocalisé. Ben oui, imaginez que votre PC rende l’âme demain, vous aurez toujours vos super programmes sur GitHub. Mais en plus de les stocker, vous pouvez aussi les rendre publics, et chacun pourra alors venir y ajouter ses évolutions.

Afin que vous puissiez collaborer sur des projets, il est nécessaire de disposer de dépôts distants. Le dépôt distant est un **historique** de votre projet hébergé sur Internet ou sur un réseau. Vous pouvez avoir plusieurs dépôts distants avec des droits différents (lecture seule, écriture, etc.).

Mais avant tout, avez-vous bien compris ce qu’était réellement un dépôt ?

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

C’est bon, ça devient un peu plus clair ? Super !

Le dépôt distant est un type de dépôt qui devient réellement important (voire **indispensable**) lorsque l’on travaille à plusieurs sur le même projet, puisqu’il permet de centraliser le travail de chaque développeur. Je vous recommande donc fortement d’utiliser GitHub pour vos dépôts distants. :)Plus aucun souci de **pertes de données** ou de travail. Sur GitHub, vous pouvez bien entendu créer des dépôts distants publics, mais aussi privés.

Sur un dépôt public, les personnes pourront collaborer à votre projet alors que sur un dépôt privé, vous seul aurez accès à votre travail !

Je vous rappelle que l’intérêt de Git est le **suivi des modifications**, mais aussi la sauvegarde de vos projets. C’est pourquoi je vous conseille de toujours commencer par copier vos sources sur un dépôt distant, si possible situé à l’extérieur de vos locaux ! Oui oui, exit la paranoïa des vilains méchants du web qui pirateraient votre dépôt, vous avez bien plus de chances de perdre vos données chez vous que sur GitHub. C’est aussi sur le dépôt distant que toutes les modifications de tous les collaborateurs seront fusionnées. Vous allez me dire, mais alors pourquoi des dépôts distants, avec GitHub on a tout ce dont on a besoin, on va pas s'embêter à créer une copie locale. Eh bien si ! La majeure partie de votre travail se fera sur votre dépôt local qui est un clone de votre dépôt distant. C’est sur votre dépôt local que vous ferez toutes vos modifications de codes, vos créations de branches (il y en a aussi sur le dépôt distant), et vos **commits** ; et seulement quand vos modifications seront prêtes à être partagées à l’équipe, vous les pousserez sur le dépôt distant.

### Pourquoi utiliser un dépôt ?

Vous êtes convaincu de la nécessité d’utilisation de dépôts ?

Pas encore ?

Avec les dépôts, vous vous éviterez de longues heures de perte de travail, de recherche des dernières modifications et de galère de travail d’équipe !

Les dépôts sont utiles si :

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

Je vous l’ai dit, qu’il deviendrait vite votre meilleur ami. :-°

### Quels sont les outils existants ?

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

Commençons par [GitHub](https://github.com/) (mon préféré, mais chuttt !! ). **GitHub** est un outil de communication et de collaboration entre plusieurs développeurs (ou tout autre personne qui écrit du texte). C’est une interface web créée pour faciliter l’interaction avec Git.

Bon, c’est vrai, tous ces outils font cela ! :euh:

L’avantage de GitHub, c’est que depuis quelques années GitHub 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. GitHub fonctionne par abonnement, mais pas de panique, il y a un abonnement gratuit qui est déjà très bien. :)

Concernant [GitLab](https://about.gitlab.com/), il est la principale alternative à GitHub depuis le rachat de GitHub par Microsoft ! Les anti-Microsoft ont même lancé le hashtag #MovingToGitLab ! GitLab fonctionne avec une version gratuite à installer sur son propre serveur ou une version cloud payante.

[BitBucket](https://fr.atlassian.com/software/bitbucket) est la version de Atlassian. Payante, elle plaira néanmoins aux habitués de la gestion de projet sous Atlassian. BitBucket conviendra aussi bien aux étudiants ou petites teams qu’aux grands groupes.

Parlons enfin de SourceForge, le petit dinosaure dans le domaine. SourceForge a été créé 10 ans avant les autres, afin de gérer à la base des projets open source. SourceForge intègre un outil de suivi des bugs et un répertoire de code intégré. Il n’est plus très populaire depuis ces dernières années.

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

Maintenant, mettons les pieds dans le plat ! :soleil:

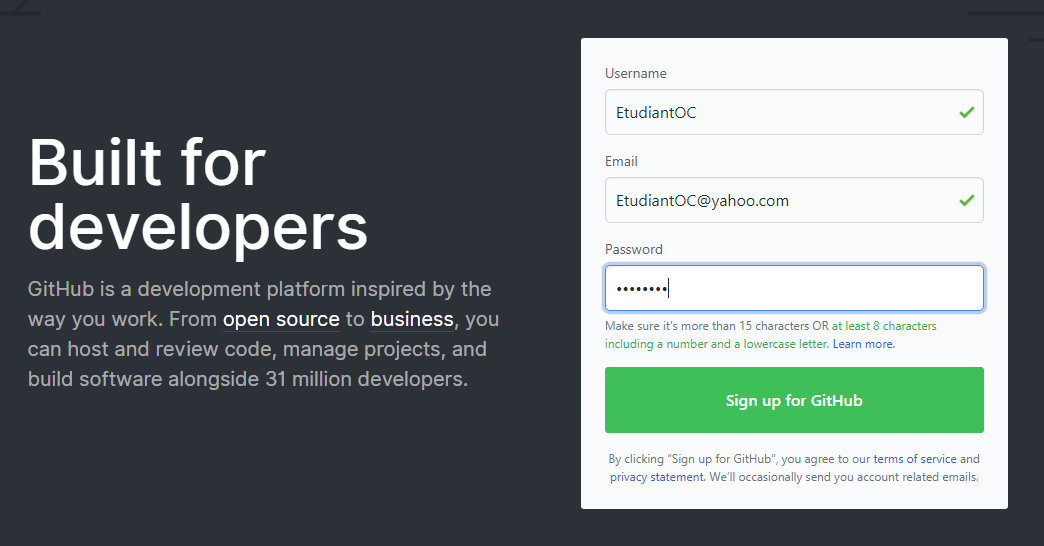


## Démarrez votre projet avec GitHub

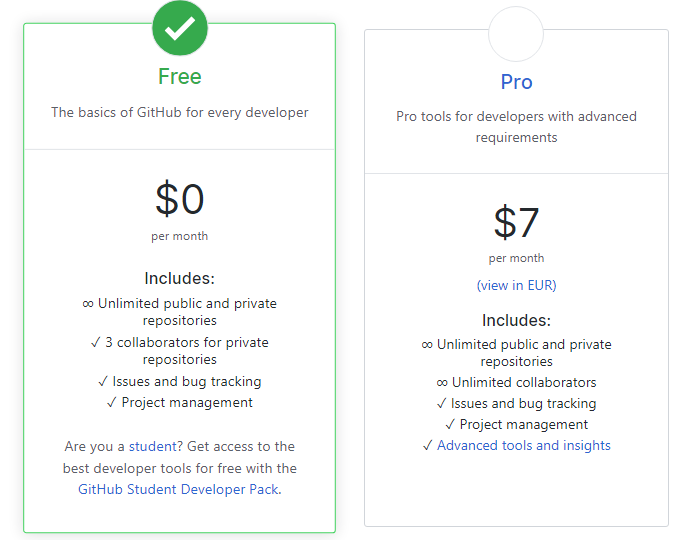
### Créez un compte GitHub

Comme nous l’avons vu dans le chapitre précédent, GitHub est un service en ligne permettant d’héberger ses dépôts distants.

Pour créer votre compte GitHub, rendez-vous sur la page d’accueil, cliquez sur Sign up. On vous demandera alors de renseigner un nom d’utilisateur, un email et un mot de passe.

Inscription GitHub

Une fois ces informations remplies, vous devrez choisir votre **abonnement** (gratuit ou pro). La principale différence entre les deux offres est que la première est destinée aux particuliers, ou aux équipes de moins de 3 collaborateurs, alors que la seconde offre est destinée aux plus grandes équipes. Sachez néanmoins que si vous faites des projets open source, il n’y a aucune limitation sur le nombre de collaborateurs. ;)

Plan GitHub

[À vous de créer votre compte GitHub !](https://github.com/)

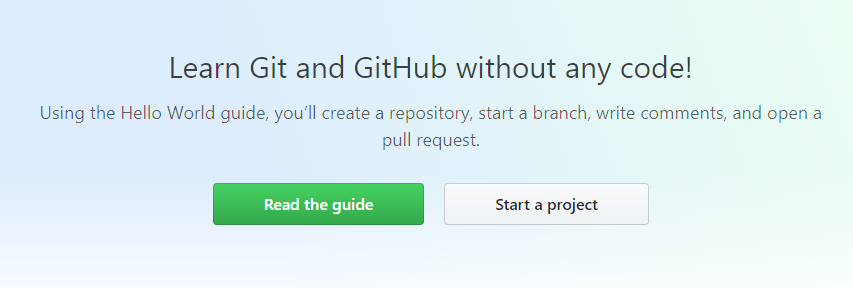
### Faites un petit tour de GitHub

GitHub est assez facile à prendre en main et simple d’utilisation. :-°

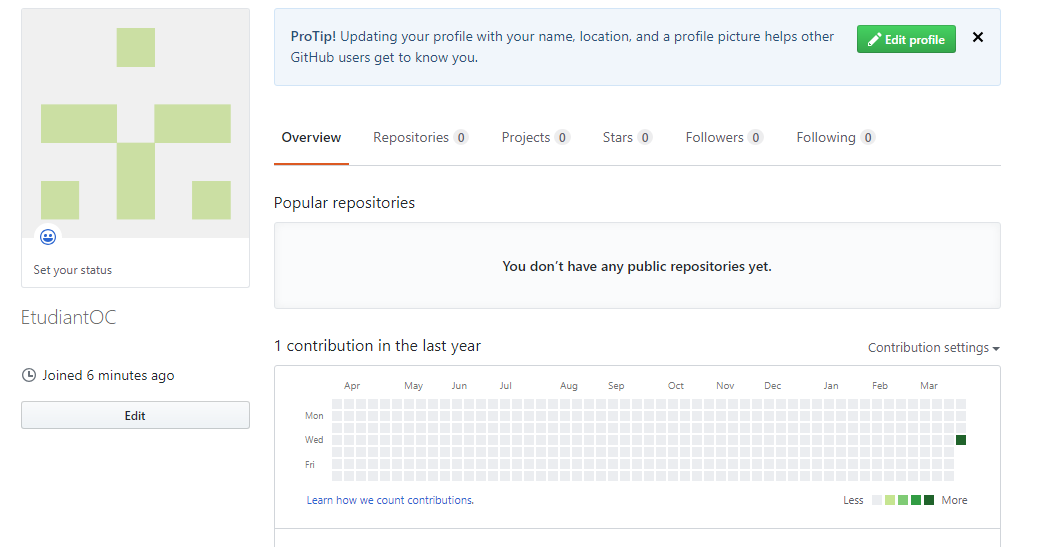
Vous pouvez consulter votre **tableau de bord** personnel pour suivre les problèmes et extraire les demandes sur lesquelles vous travaillez ou que vous suivez, accéder à vos principaux référentiels et pages d'équipe, rester à jour sur les activités récentes des organisations et des référentiels auxquels vous êtes abonné et explorer les référentiels recommandés.

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

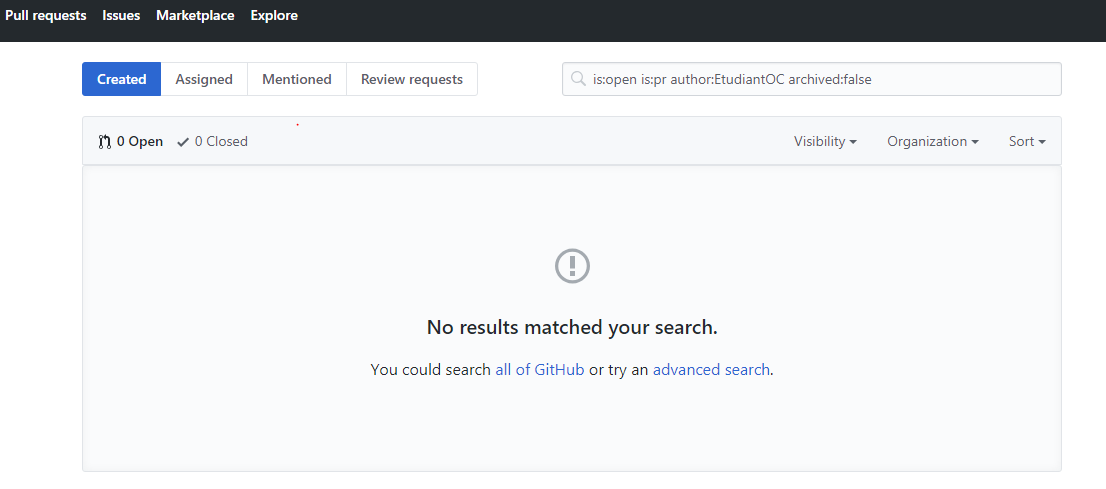
Pour créer un projet, il suffit de cliquer sur “Start a project”.

Cliquez sur "Start a project"

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

Profil

**L’onglet Pull requests**, quant à lui, permet de réaliser des demandes de pull. Les demandes de pull (extractions) vous permettent d'informer les autres sur les modifications que vous avez appliquées à une branche d'un référentiel sur GitHub. Une fois qu'une demande d'extraction est ouverte, vous pouvez discuter et examiner les modifications éventuelles avec les collaborateurs, et ajouter des validations de suivi avant que vos modifications ne soient fusionnées dans la branche de base.

Onglet "Pull requests"

Dans la section "**Activité récente**" de votre fil d’actualité, vous pouvez rapidement rechercher et suivre les problèmes récemment mis à jour, et extraire les demandes sur lesquelles vous travaillez. Sous "Activité récente", vous pouvez prévisualiser jusqu'à 12 mises à jour récentes effectuées au cours des deux dernières semaines.

Une activité est récente lorsque :

* vous avez ouvert un problème ou une demande d'extraction ;
* quelqu'un a commenté un problème ou tiré une demande que vous avez ouverte ;
* votre problème ou demande d'extraction a été rouvert ;
* votre avis a été demandé sur une demande de tirage ;
* vous avez été affecté à un problème ou à une demande d'extraction ;
* vous avez référencé un problème ou une requête d'extraction via un commit ;
* vous avez commenté un problème ou une demande d'extraction.

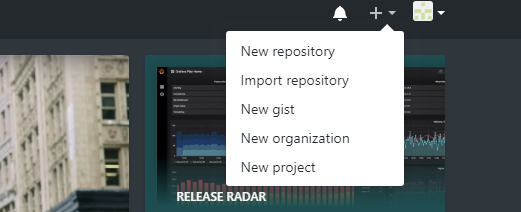
Un des derniers points importants sur GitHub est la fonctionnalité **Explore**.

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

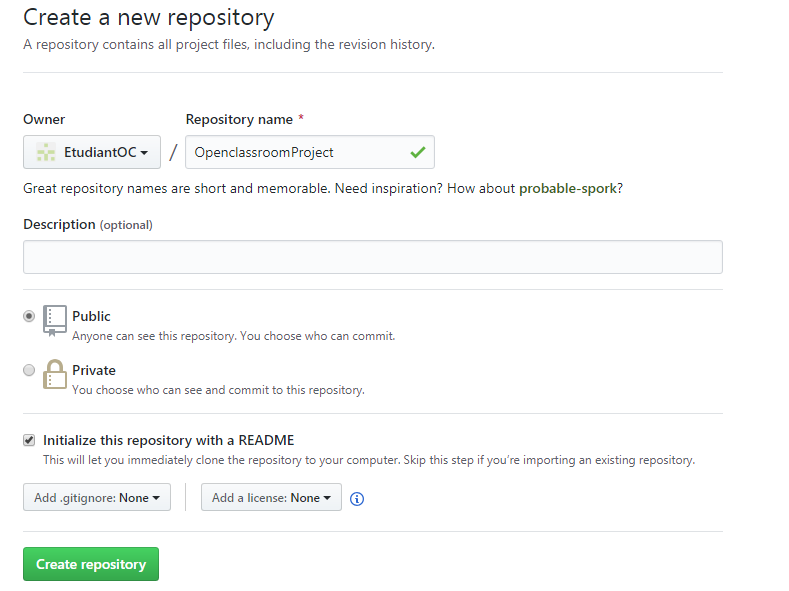
### Créez votre propre dépôt

Pour mettre votre projet sur GitHub, vous devez créer un référentiel dans lequel il pourra être installé.

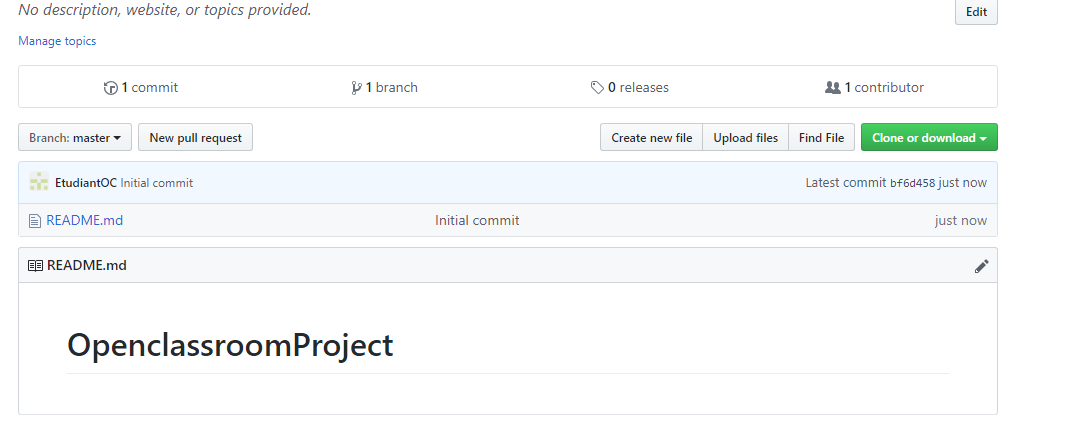
Cliquez sur le "+" dans le coin supérieur droit, pour faire apparaître l’option New repository.

Ajoutez un référentiel

Choisissez un nom simple pour votre dépôt, choisissez si vous souhaitez créer un dépôt public ou privé, initialiser un readme et un gitignore.

Créez un dépôt

Cliquez ensuite sur “Créer un dépôt”.

Votre premier dépôt Git

## Installez Git sur votre ordinateur

Tadammm ! L’heure est venue d’installer **votre futur meilleur ami**. :)

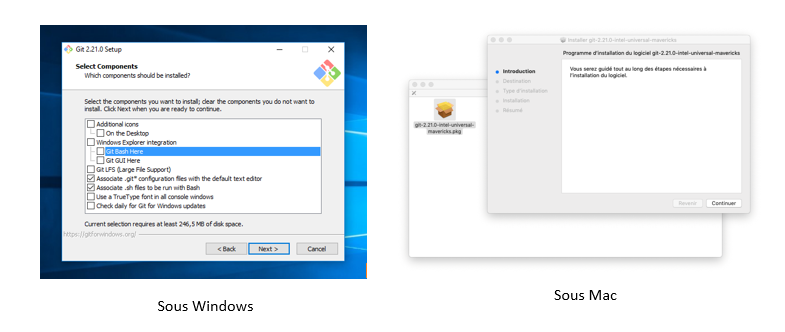
Je vous invite à suivre ce screencast pour installer Git sur votre ordinateur. Les étapes sont également détaillées sous la vidéo.

Afin d'installer Git, il faut dans un premier temps le **télécharger**. Vous pourrez trouver [le lien de téléchargement ici](https://git-scm.com/downloads). Choisissez la **version** qui correspond à votre installation et téléchargez-la.

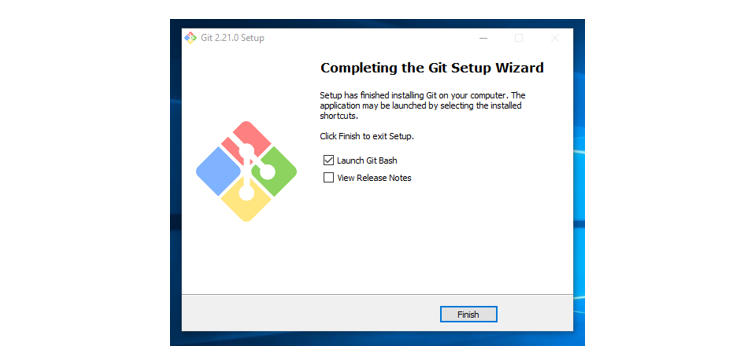
Exécutez le fichier que vous venez de télécharger :

Exécutez le fichier

Appuyez sur Suivant à chaque fenêtre puis sur **Installer**. Lors de l’installation, laissez toutes les options par défaut, elles conviennent bien.

Lancez l'installation

Cochez ensuite Launch Git Bash.

Lancez Git Bash

Git Bash se lance.

Fenêtre Git Bash

### Initialisez Git

La première chose à faire est de configurer son identité. Pour cela, nous allons entrer dans le monde des lignes de commande !

Nous allons commencer par renseigner votre nom et votre adresse e-mail. C'est une information importante car toutes les validations dans Git utilisent cette information et elle est indélébile dans toutes les validations que vous pourrez réaliser :

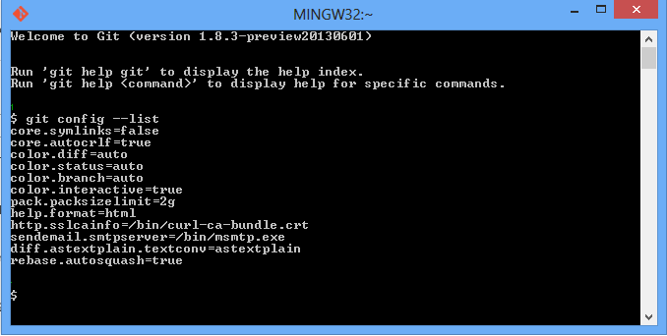
$ git config --global user.name "Rahal Mohamed"

$ git config --global user.email rahalfat10@yahoo.fr

Grâce à l’option --global, vous n’aurez besoin de le faire qu'une fois.

Si vous souhaitez par contre, pour **un projet spécifique**, changer votre nom d’utilisateur, vous devrez repasser cette ligne mais sans le --global.

Afin de vérifier que vos paramètres aient bien été pris en compte, et vérifier les autres paramètres, il suffit de passer la commande  git config --list



Il est recommandé d’activer les couleurs afin d’améliorer la lisibilité des différentes branches. Pour cela, passez ces trois lignes dans Git Bash :

$ git config --global color.diff auto

$ git config --global color.status auto

$ git config --global color.branch auto

Par défaut, Git utilisera Vim comme éditeur et Vimdiff comme outil de merge. Vous pouvez les modifier en utilisant :

$ git config --global core.editor notepad++

$ git config --global merge.tool vimdiff

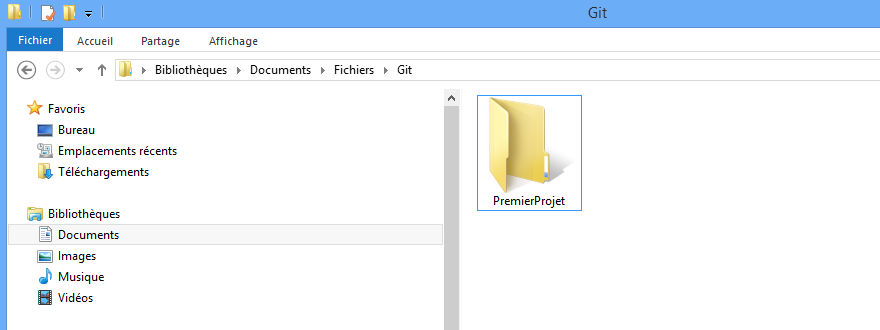
Maintenant que nous avons paramétré les paramètres de base, nous allons **créer notre fameux dépôt local**. Pour ce faire, deux solutions possibles :

* créer un dépôt local vide pour accueillir un nouveau projet ;
* cloner un dépôt distant, c’est-à-dire rapatrier tout l’historique d’un dépôt distant en local, afin de pouvoir travailler par dessus.

Dans un logiciel de gestion de versions comme Git, un dépôt représente une copie du projet.

Chaque ordinateur d’un développeur qui travaille sur le projet **possède donc une copie du dépôt**.  
Dans chaque dépôt, on trouve les fichiers du projet ainsi que leur historique.

Nous allons maintenant voir comment créer un dépôt vide. On va créer, dans un premier temps, un dossier sur notre disque (avec le nom de notre projet, c’est mieux ;)).

Créez un dossier

Accédez à votre dossier, et lancez la ligne ci-dessous dans Git Bash en ciblant ce dossier :

johndoe ~

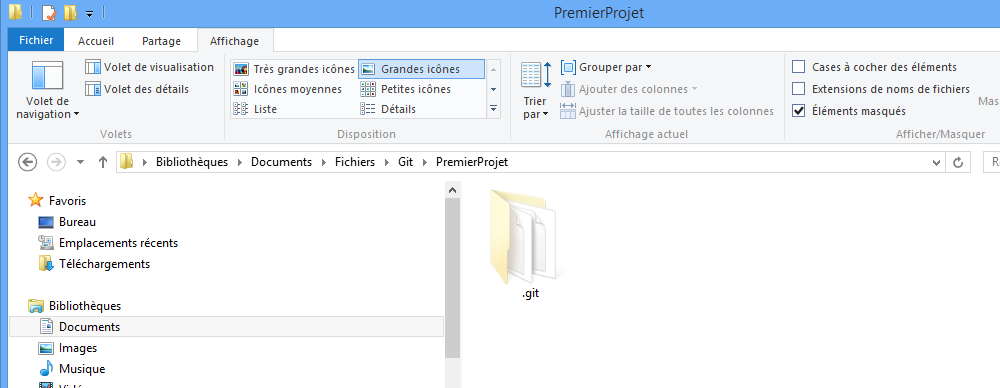
$ cd Documents/Fichiers/Git/PremierProjet

johndoe ~/Documents/Fichiers/Git/PremierProjet

$ git init

Initialized empty Git repository in c:/users/JohnDoe/Documents/Fichiers/Git/PremierProjet/

Votre dossier n’a rien de nouveau, mais c’est normal. :) Vous avez pourtant bien initialisé votre dépôt Git. **Un dossier caché .git a été créé** ! Vous pouvez l'afficher en allant dans Affichage => Éléments masqués.

Affichez le dossier Git caché

Nous verrons plus tard comment l’utiliser. Dans le prochain chapitre, vous découvrirez comment contribuer à la communauté grâce aux projets open source !