



Installer et utiliser GIT et GITHUB

-Niveau débutant & intermédiaire-

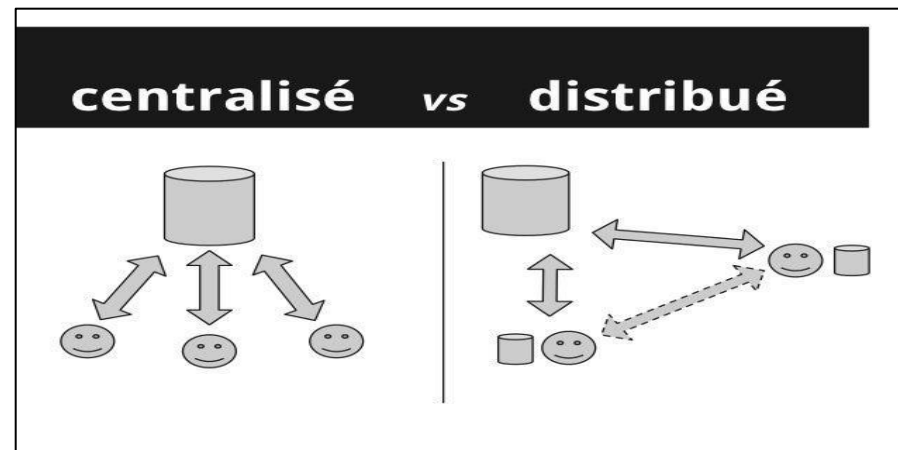
Un logiciel de **versionning**, ou logiciel de **gestion de version**, **est** un logiciel qui permet de conserver un historique des modifications effectuées sur un projet afin de pouvoir rapidement identifier les changements effectués et de revenir à une ancienne version en cas de problème.

Les **logiciels de gestion de versions** sont quasiment **incontournables** aujourd'hui car ils **facilitent** grandement la **gestion de projets** et ils permettent de travailler en équipe de manière beaucoup plus efficace.

Parmi les logiciels de gestion de versions, **Git** est le leader incontesté et il est donc indispensable pour tout développeur de savoir utiliser **Git**.

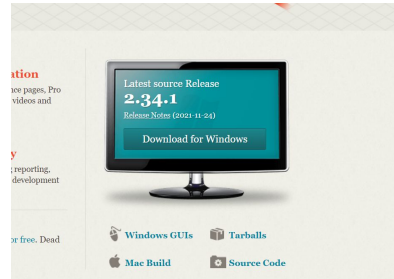
Il existe deux type d'outils de **versionning**:

- **Les outils de versionning centralisés** utilisent un serveur qui héberge le dépôt de référence, et chaque utilisateur lit et écrit sur celui-ci.
- **Les outils de versionning décentralisés** utilisent un serveur qui héberge le dépôt de référence central, mais chaque utilisateur héberge son propre dépôt de référence local. Cela permet d'éviter les dégâts en cas de perte du serveur, puisqu'il peut être recréé via le dépôt d'un utilisateur, et plus encore, cela permet de travailler en local tout en conservant la possibilité de versionner ses modifications, créer des branches, etc...

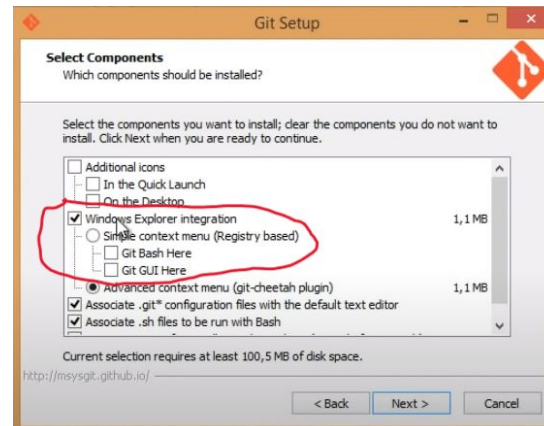


Sous Windows:

- 1) Rendez vous à l'url suivante : <https://git-scm.com/> et téléchargez la dernière version de git correspondant à votre version de Windows (normalement 64 bits).
- 2) Exécutez le fichier obtenu. Vous tomberez automatiquement sur l'installation de git.

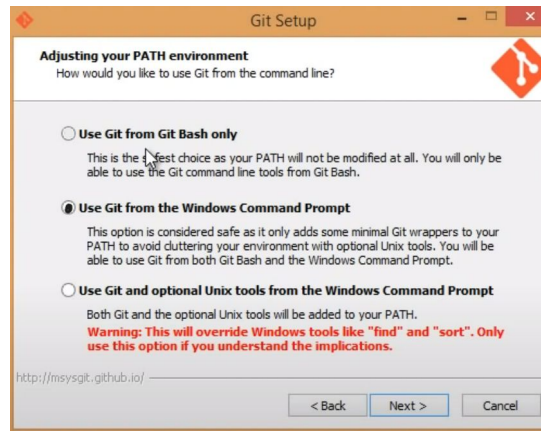


Dans cette fenêtre, vous avez la possibilité d'intégrer les commandes git directement dans Windows explorer. Faites comme bon vous semble, mais sachez que dans ce cours, nous ne l'utiliserons pas.



Sur cet écran, nous avons trois choix.

- Le premier vous force à utiliser un terminal spécial pour git (git bash).
- Le second vous donne la possibilité d'utiliser git avec le terminal de Windows.
- Le troisième vous offre la possibilité d'utiliser les commandes unix (cd, ls...), quand vous utiliserez git, cochez le second choix et continuez l'installation.



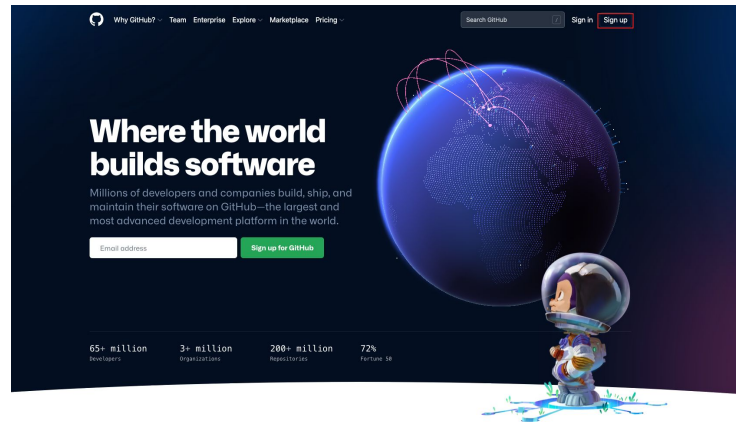
L'installation étant terminée, entrez dans votre terminal et faites : `git help`

Cette commande vous affichera la documentation de git et vous permettra de vérifier si git a été correctement installé sur votre système.

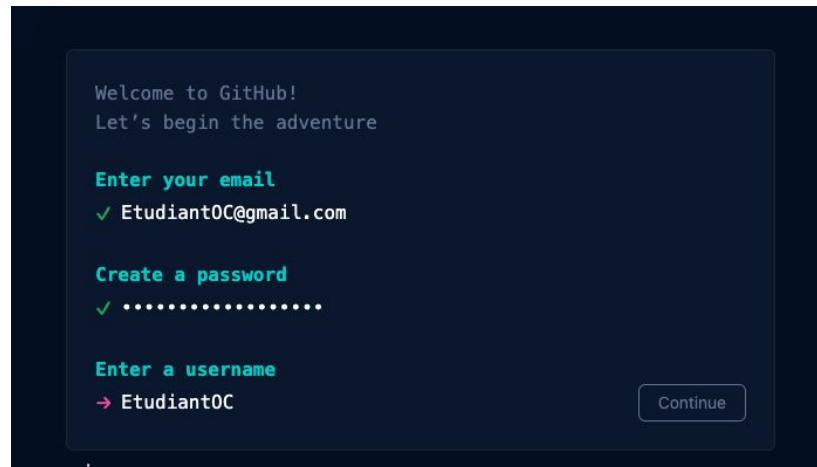
Pour installer git sur votre **système linux**, dans le terminal faites simplement : `sudo apt-get install git`

Pour vérifier l'installation, faites `git-help` qui vous affichera la documentation de git.

GitHub est un service en ligne qui va vous permettre d'héberger vos dépôts distants. Pour créer votre **compte GitHub**, rendez-vous sur <https://github.com/> et cliquez sur « Sign up »

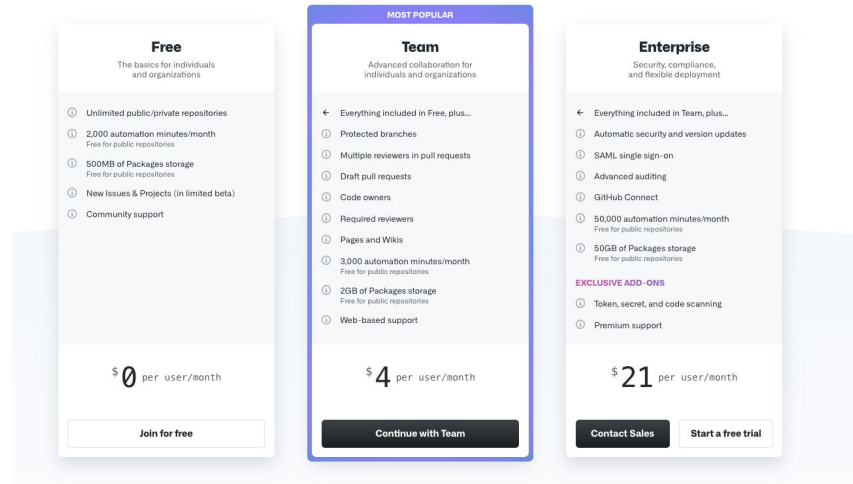


Vous devez renseigner un e-mail, un mot de passe et un nom d'utilisateur.



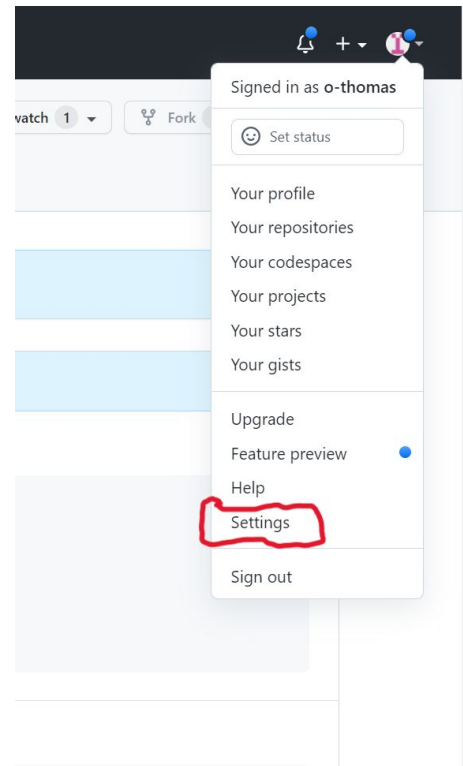
Un code de vérification vous sera envoyé sur votre adresse mail afin de confirmer votre identité. Et voilà, vous êtes à présent inscrit sur GitHub ! Par défaut, GitHub est gratuit. Mais sachez qu'il existe également des offres payantes.

Choose the plan that's right for you.

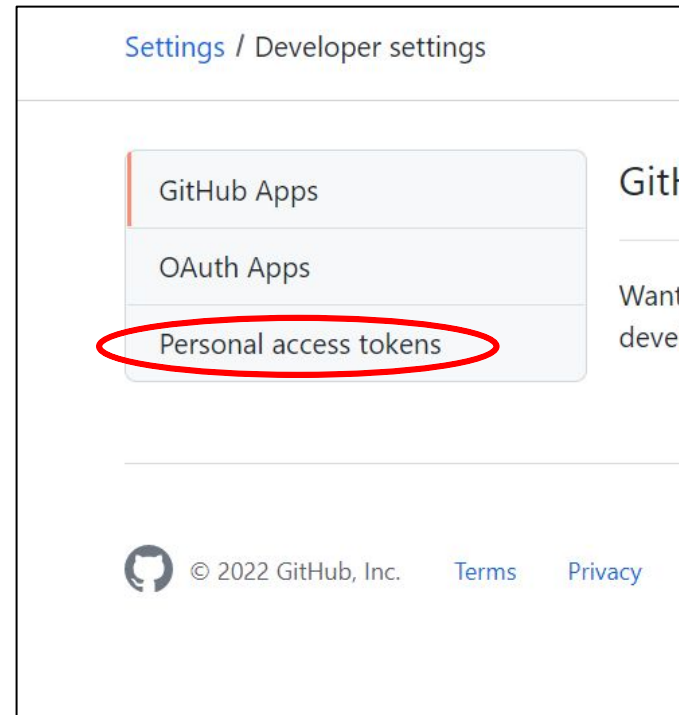
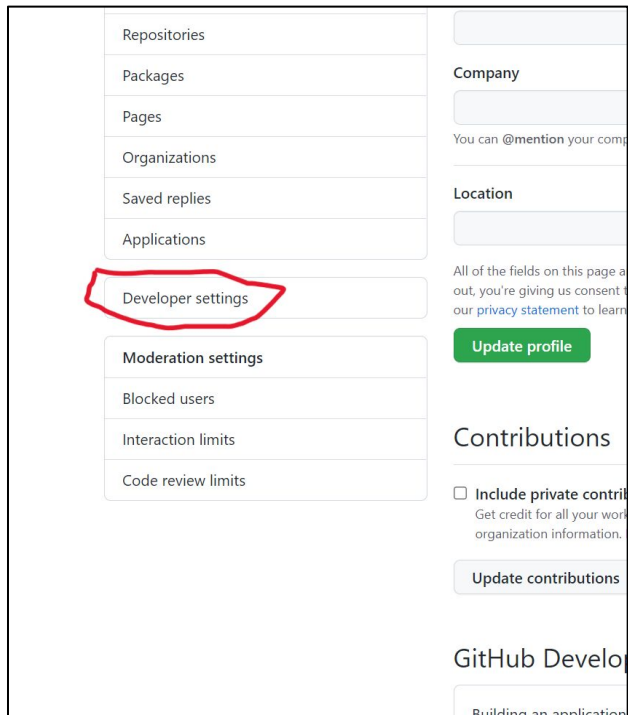


Free	Team	Enterprise
The basics for individuals and organizations	Advanced collaboration for individuals and organizations	Security, compliance, and flexible deployment
<ul style="list-style-type: none">Unlimited public/private repositories2,000 automation minutes/month (Free for public repositories)500MB of Packages storage (Free for public repositories)New Issues & Projects (in limited beta)Community support	<ul style="list-style-type: none">Everything included in Free, plus...Protected branchesMultiple reviewers in pull requestsDraft pull requestsCode ownersRequired reviewersPages and Wikis3,000 automation minutes/month (Free for public repositories)2GB of Packages storage (Free for public repositories)Web-based support	<ul style="list-style-type: none">Everything included in Team, plus...Automatic security and version updatesSAML single sign-onAdvanced auditingGitHub Connect50,000 automation minutes/month (Free for public repositories)50GB of Packages storage (Free for public repositories)EXCLUSIVE ADD-ONSToken, secret, and code scanningPremium support
\$0 per user/month	\$4 per user/month	\$21 per user/month
Join for free	Continue with Team	Contact Sales Start a free trial

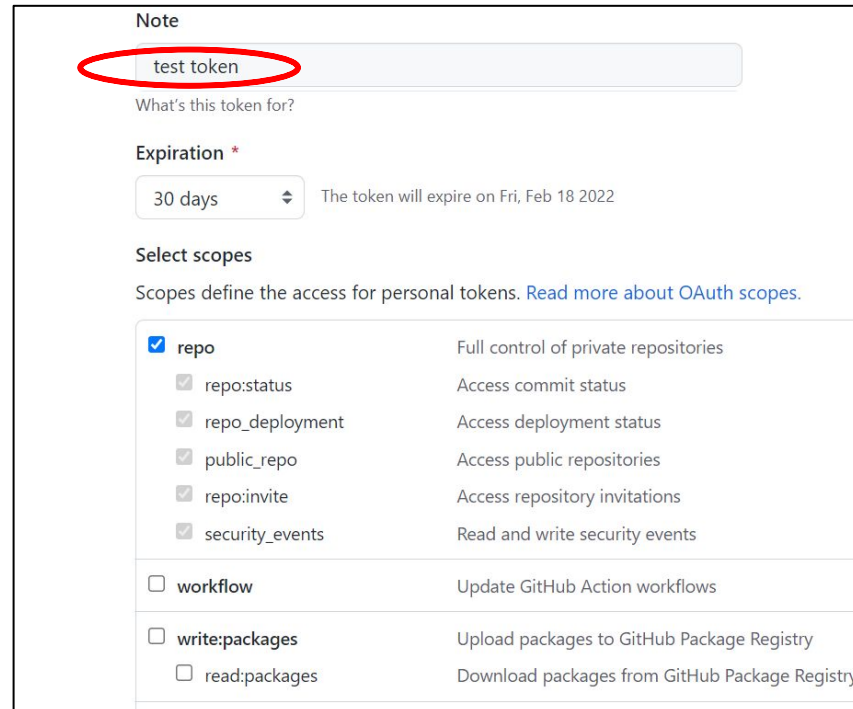
Nous allons maintenant procéder à la génération d'un token de sécurité. Il s'agit d'une chaîne de caractère qui vous permettra d'accéder à vos différents repositories entre autres. Pour cela, cliquez sur l'icône de votre profil et dans le menu déroulant, sur settings.



Ensuite, cliquez sur "Developer settings ». Sur la page suivante, cliquez sur "Personal access tokens » puis cliquez sur le bouton "generate new token ».



Mettez maintenant une note à votre token, une date d'expiration et définissez les accès de votre token. (cochez uniquement la case repo). Nous n'aurons pas besoin du reste dans ce cours.



Note

test token

What's this token for?

Expiration *

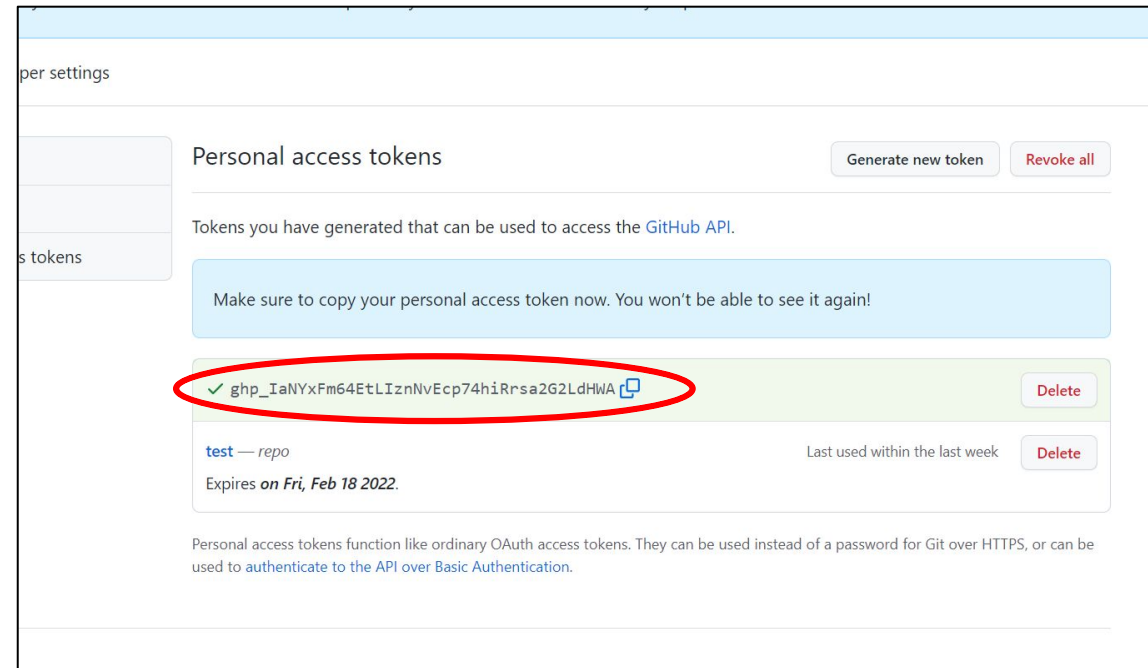
30 days The token will expire on Fri, Feb 18 2022

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry

Copiez la clé ainsi créée et collez-la dans un document texte, nous en aurons besoin plus tard.



Un **repository** (ou dépôt en français) est tout simplement l'endroit où vous stockerez votre projet. Pour le créer, il faut cliquer sur l'icone « + » à côté de votre profil. Cliquez ensuite sur "New repository"




Entrez ensuite les informations de votre projet (nom, description) et cliquez sur « Private ».

De ce fait, le repository sera accessible à vous et à toutes les personnes que vous avez sélectionnées. Si le repository reste en « public », tout le monde aura accès à votre code. Cela peut être utile si vous souhaitez présenter un projet à un futur employeur ou mettre un projet en open source par exemple.

Create a new repository

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

Owner * Repository name *

 o-thomas /

Great repository names are: Your new repository will be created as Nom-du-projet. your expert-parakeet?

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

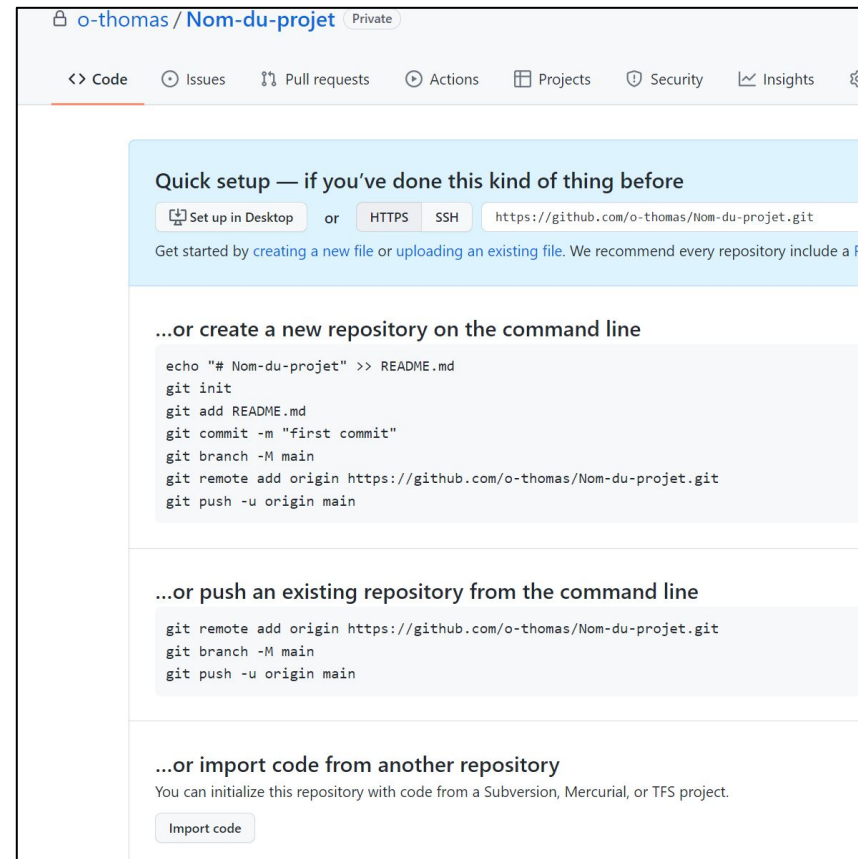
☐ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

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

[Create repository](#)

Votre **repository** est maintenant créé. Sur la page actuelle, il y a des informations pour commencer à travailler. Ignorons-le pour le moment, notre façon de procéder diffèrera sensiblement de la méthode présentée ici.



Nous allons maintenant configurer votre identité git sur votre ordinateur.

Rendez vous sur votre terminal et faites les commandes suivantes:

```
git config --global user.name "Votre username github"
```

```
git config --global user.email "Votre mail github"
```

Cela aura pour effet d'enregistrer vos identifiants github dans votre logiciel local.

Le mot clé `--global` permet d'enregistrer vos informations dans la configuration globale de git.

Si vous souhaitez, pour **un projet spécifique**, changer votre nom d'utilisateur, vous devrez repasser cette ligne mais sans le `--global`

Dans un premier temps, créez un dossier sur votre ordinateur avec le nom de votre projet. Avec le terminal, naviguez dans votre nouveau dossier et faites:

Cela aura pour effet d'initialiser le dépôt git. `git init`

Un dossier caché .git a été créé ! Vous pouvez l'afficher en allant dans Affichage => Éléments masqués.

Pour les plus curieux, sachez que ce dossier caché contient tous les éléments non visibles de Git : la configuration, les logs, les branches...

Dans votre dossier de projet, créez un fichier HTML et CSS avec le contenu de votre choix. Voici un exemple:

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>

<body>
  <h1>Mon projet git</h1>
</body>
</html>
```

```
h1 {
  color: red;
}
```

Vous avez créé 2 fichiers, index.html et styles.css. Ces fichiers sont situés dans le dépôt Git. Pour créer une nouvelle version de votre projet, vous allez maintenant indexer les fichiers.

Pour cela, retournez dans le terminal.

Vous pouvez maintenant faire passer les fichiers index.html et styles.css vers l'index en utilisant la commande "git add" suivie du nom du fichier:

```
git add index.html styles.css
```

Notez que si vous souhaitez ajouter tous vos fichiers en même temps, faites simplement la commande "git add" suivi d'un "."

```
git add .
```

Voilà, vous avez indexé vos deux fichiers !

Maintenant que vos fichiers modifiés sont indexés, vous pouvez créer une version, c'est-à-dire archiver le projet en l'état. Pour cela, utilisez la commande "git commit" :

```
git commit -m "Ajout des fichiers html et css de base"
```

L'argument « -m » permet de définir un message particulier rattaché au commit effectué. Si vous n'utilisez pas cet argument, la commande "git commit" ouvrira un éditeur de texte dans lequel vous pourrez saisir le message de commit. Vos modifications sont maintenant enregistrées avec la description "Ajout des fichiers html et css de base".

Nous allons maintenant lier votre dépôt local et distant. Pour cela, nous allons utiliser une simple commande:

```
git remote add origin https://NOM_UTILISATEUR:TOKEN_GITHUB@github.com/NOM_UTILISATEUR/NOM_DU_DEPOT.git
```

Nous allons maintenant renommer notre branche principale (master). Par convention, git propose le nom "main ». Nous n'allons pas les contrarier. Faites donc la commande:

```
git branch -M main
```

Enfin, vous avez relié le dépôt local au dépôt distant. Vous pouvez donc envoyer des commits du repository vers le dépôt distant GitHub en utilisant la commande suivante :

```
git push -u origin main
```

Vous allez maintenant créer un dépôt pour chaque projet que vous avez sur votre ordinateur et "pusher" ces projets sur vos dépôts.

C'est un exercice, mais dans un autre temps, si vous souhaitez reprendre l'un de ces projets plus tard, ou les présenter à un employeur et que vous changez de PC, vous pourrez les récupérer facilement.

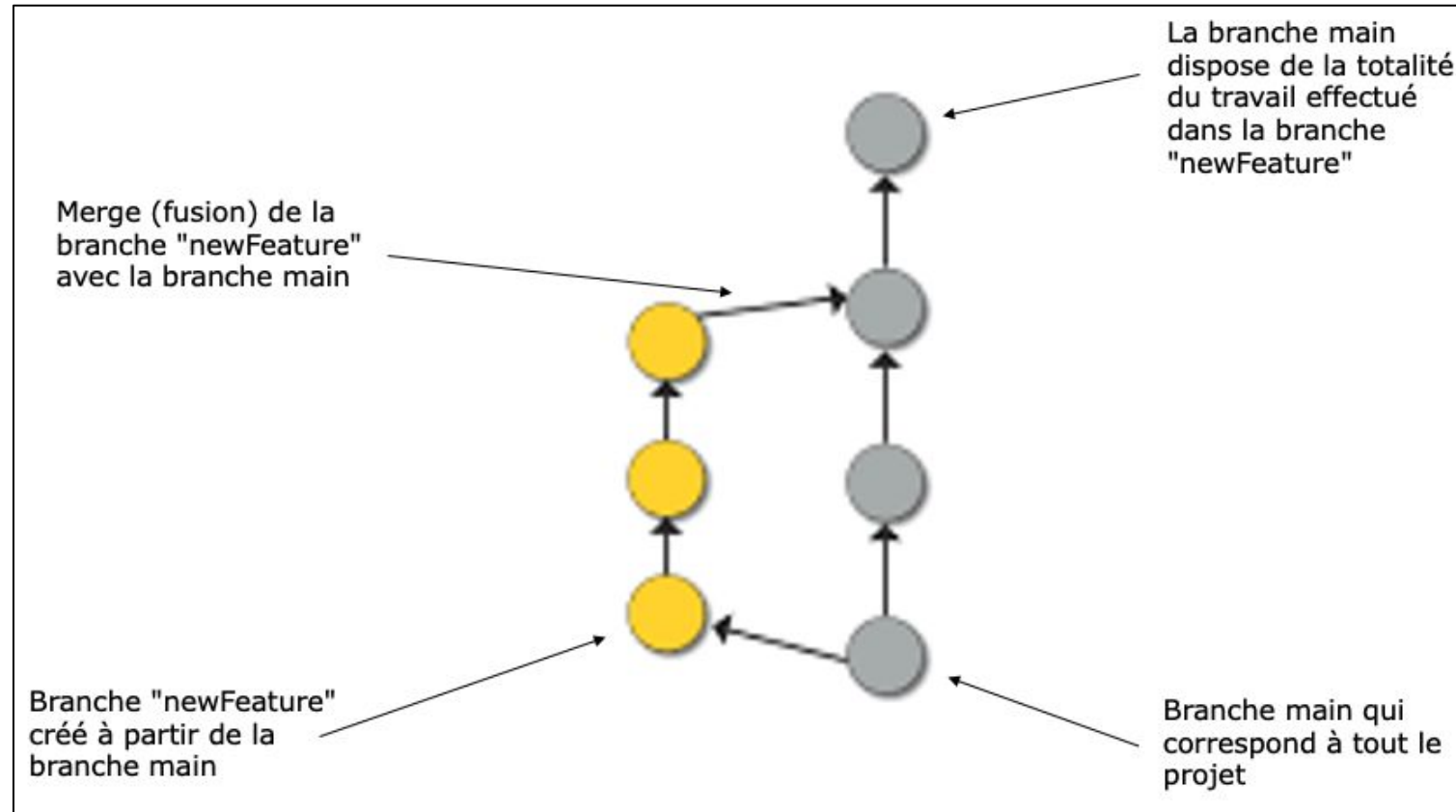
Le principal atout de Git est **son système de branches**. C'est sur ces branches que repose toute la magie de Git ! Les différentes branches correspondent à des copies de votre code principal à un instant T, où vous pourrez tester toutes vos idées sans que cela impacte votre code principal.

La branche principale (main) portera l'intégralité des modifications effectuées. Le but n'est donc pas de réaliser les modifications directement sur cette branche, mais de les réaliser sur d'autres branches, et après divers tests, de les intégrer sur la branche principale.

Imaginez que vous ayez réalisé une application pour un client, et que ce dernier ait une superbe idée de feature à ajouter à son application.

Votre application fonctionne parfaitement, elle est en production, et y toucher serait prendre le risque de tout faire planter. Avec Git et ses fameuses branches, pas de soucis. Vous pouvez créer une branche correspondant à l'évolution et cela **sans toucher à votre application en cours de production**.

Une fois que toutes vos modifications auront été testées, vous pourrez les envoyer en production sans crainte en les intégrant à la branche **main** (et dans le pire des cas, revenir en arrière simplement) !



Pour connaître les branches présentes dans notre projet, il faut taper la ligne de commande suivante:

```
git branch
```

Dans un premier temps, vous n'aurez que :

```
git branch  
* main
```

Ceci est normal. L'étoile signifie que c'est la branche sur laquelle vous vous situez et que c'est sur celle-ci qu'actuellement vous réalisez vos modifications.

Il est souvent préférable de créer une branche pour une modification. La création prend 30 secondes, vous fait économiser du temps et éviter de faire des bêtises sur votre branche principale.

Mais revenons au projet de M. Robert !

Nous avons donc notre branche **main**, et nous souhaitons maintenant réaliser la fonctionnalité Cagnotte. Pour cela, il faut taper `git branch` suivi du nom de la branche.

Il est plus pratique d'utiliser un nom qui a un rapport avec la modification que vous souhaitez apporter. Imaginons que vous souhaitez ajouter à votre application une interface de connexion:

```
git branch connexion
```

Votre branche a bien été ajoutée, mais pour travailler, il faut basculer dessus. Pour se faire, il faut utiliser la commande `git checkout` suivi du nom de la branche. Par exemple, pour basculer sur notre branche connexion, nous écrivons:

```
git checkout connexion
```

Vous pouvez désormais réaliser votre évolution sans toucher à la branche **main** qui abrite votre code principal fonctionnel. Vous pouvez re-basculer si besoin à tout moment sur la branche **main**, sans impacter les modifications de la branche connexion.

Vous avez réalisé des évolutions sur la branche « *connexion* », il faut maintenant demander à Git de les enregistrer et de créer une nouvelle version du projet comprenant les évolutions réalisées sur la branche « Cagnotte ».

Vous devez créer un "commit" grâce à la commande:

```
git commit -m "Ajout de l'interface de connexion"
```

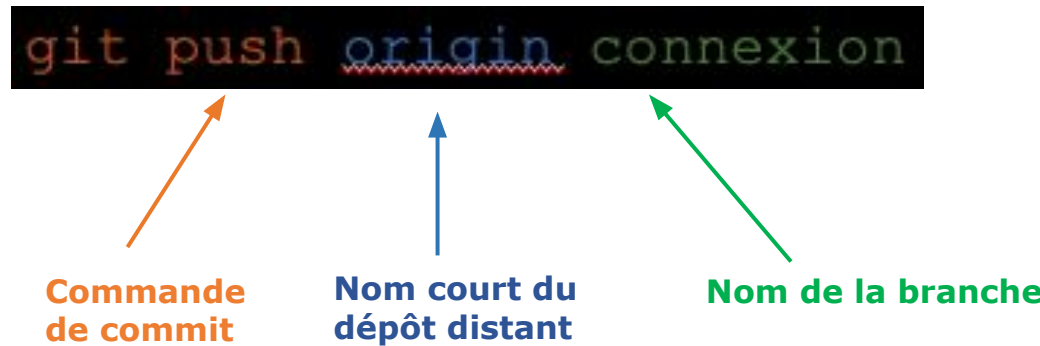
Commande
de commit

Argument
message

Message de
commit

Vos modifications sont maintenant enregistrées avec la description "Ajout de l'interface de connexion".

Avec la commande « git commit », nous avons enregistré des modifications en local, uniquement. Il ne vous reste plus qu'à envoyer les modifications réalisées sur le dépôt distant grâce à la commande « git push ».



```
git push origin connexion
```

The diagram shows the command `git push origin connexion` with three colored arrows pointing to its parts: an orange arrow from "Commande de commit" to `git push`, a blue arrow from "Nom court du dépôt distant" to `origin`, and a green arrow from "Nom de la branche" to `connexion`.

Voilà plusieurs jours que vous travaillez sur l'évolution "connexion" de l'application. Après de nombreux *commits* et *pushs*, le travail semble terminé. Vous effectuez donc une série de tests pour vérifier le bon fonctionnement de l'évolution "connexion" et tout est fonctionnel !

À présent, il faut intégrer l'évolution réalisée dans la branche "connexion" à la branche principale "main". Pour cela, vous devez utiliser la commande "git merge".

Cette commande doit s'utiliser à partir de la branche dans laquelle nous voulons apporter les évolutions. Dans notre cas, la commande s'effectuera donc dans la branche **main**. Pour y retourner, utilisez les commandes :

```
git checkout main
```

Permet de basculer sur la branche main

```
git merge connexion
```

Permet de fusionner la branche connexion à la branche main

Rendez-vous sur cet URL: https://learngitbranching.js.org/?locale=fr_FR

Il s'agit d'un petit jeu qui vous permettra de mieux appréhender **git** et ses commandes parfois obscures.