Git

Maintenant que vous savez utiliser le terminal, nous allons pouvoir gérer des projets avec git.

CONFIGURATION DE GIT

Git est déjà installé sur votre ordinateur. La première chose à faire, c'est de le configurer.

Tout d'abord, il faut lui indiquer qui vous êtes : à chaque modification que vous ferez, votre nom et adresse mail apparaîtront pour toutes les personnes qui font partie du projet. C'est ce qui permettra de savoir qui a fait quoi.

Pour configurer votre identité, allez dans le terminal entrez les commandes :

```
git config --global user.name "Votre nom"

git config --global user.email adresse@email.com
```

Il est conseillé de mettre une adresse mail, même fictive, sinon vous risquez d'avoir des messages intempestifs pour renseigner celle-ci.

La commande `config` est un outil qui va vous permettre de configurer Git en local. Le paramètre `--global` signifie que ces paramètres seront automatiquement utilisés sur tous les projets que vous aurez sur votre machine.

À noter : *Toutes les commandes git commencent par **`git`**.*

La deuxième chose à configurer sont les couleurs dans le terminal Git. Vous le verrez très vite, les différentes couleurs sont très importantes.

Dans le terminal, il suffit de taper :

```
git config --global color.ui true
```

Git est maintenant configuré.

METTRE UN PROJET SOUS GIT

À noter : Quand on parle d'un projet sur Git, on va généralement parler d'un repo Git. Cela vient du terme anglais repository qui signifie « dépôt ».

On va donc **créer un repo git**.

Pour mettre un répertoire sous git, avec le terminal il faut se placer dans le répertoire du projet que vous souhaitez mettre sous git, puis l'initialiser avec la commande :

```
git init
```

Votre repo Git est initialisé, prêt à suivre l'avancement de votre projet.

ENREGISTRER DES MODIFICATIONS

Le statut des fichiers

Dans git les fichiers vont passer par différents statuts, ceux-ci vont nous permettre de suivre les modifications de nos fichiers.

Les différents statuts dans git :

- **Non suivi** : le fichier existe en local, mais n'est pas suivi par Git. En cas de modification de ce fichier, Git n'en tiendra pas compte et fera comme s'il n'existait pas.
- **Non modifié** : le fichier est suivi par Git, et n'a pas été modifié en local. Il ne se passe rien.
- **Modifié** : le fichier a été modifié en local. Git est capable de dire qu'il y a eu des modifications, mais il ne les a pas stockées pour la prochaine écriture de l'historique.
- **Stocké** : le fichier est "stocké" à part et ses modifications seront incluses dans la prochaine écriture de l'historique.

Pour vérifier le statut des fichiers de votre projet, depuis votre terminal naviguer jusqu'à votre répertoire et utiliser la commande :

```
git status
```

Le terminal va alors vous détailler le statut des fichiers, par exemple :

```
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        images/
        index.html
        style.css
```

Ajouter des fichiers au suivi Git

Par défaut, Git ne suit pas l'état d'avancement des fichiers. Il faut lui indiquer quels fichiers il doit suivre, pour qu'il puisse tenir compte de leurs modifications.

Pour ajouter un document ou un fichier au suivi de Git, il suffit d'utiliser la commande `git add`. Par exemple :

```
git add index.html
```

Git indique qu'il a repéré le nouveau fichier « index.html », et qu'il va le suivre :

```
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier : index.html

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        fonts/
        images/
        style.css
```

Quand on demande à Git de suivre un fichier, il le passe directement de statut **non suivi** à **stocké**. Il est donc prêt à être inscrit dans l'historique.

Concrètement, si vous passez un projet sur un serveur via Git, les personnes qui récupéreraient le projet auront uniquement les fichiers qui sont suivis.

Dans notre exemple, à ce stade, si vous passez ce projet sur un serveur via Git, les personnes qui récupéreraient le projet auront uniquement le fichier `index.html`*.*

Pour ajouter plusieurs éléments au suivi de Git, il suffit de mettre des espaces entre chaque.

Par exemple :

`git add style.css images/`

Ici on va donc ajouter un fichier « style.css » et un répertoire « images » au suivi.

Faire un point de sauvegarde

Pour écrire dans l'historique de notre projet git, on va faire ce que l'on appelle un « commit ». Cela nous sert de point de sauvegarde de notre version du projet.

On va alors utiliser la commande :

`git commit -m "Mon premier commit"`

Ici « mon premier commit » est le message donné à notre « commit ».

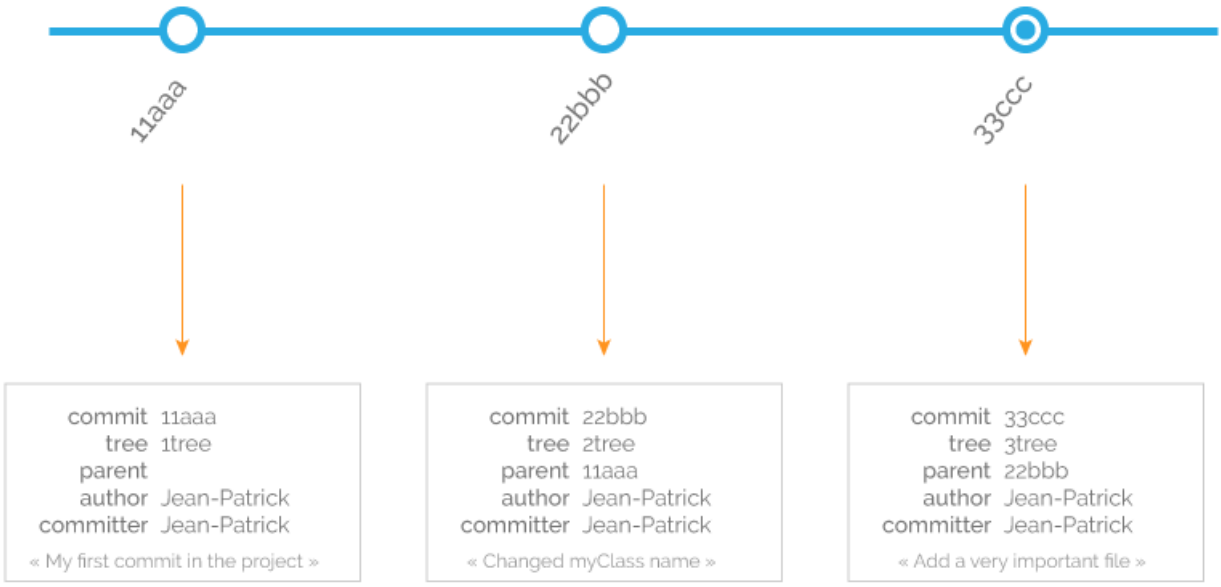
À noter : Le but du message est de comprendre ce que l'on a fait. Pour pouvoir s'y retrouver dans l'historique, dans un projet personnel comme lorsque l'on travaille à plusieurs..

Par exemple :

`ajout(header):logo`

À ce stade si vous faites un commit vos fichier auront donc le statut **non modifié**.

Visuellement, on peut représenter un projet git comme ceci :



Chaque point bleu est un commit, une version de votre projet, le dernier point est donc la dernière version du projet, votre dernier commit. L'ensemble des cases qui détaillent chaque commit est l'historique.

Vous pouvez vérifier l'historique de votre projet avec la commande :

`git log`

Pour quitter l'historique il suffit d'utiliser la commande **`q`**

Suivre des modifications

Lorsque vous faites une modification sur un fichier qui est suivi, si vous vérifiez le statut de votre projet le fichier en question passera alors en **modifié**.

Par exemple sur le fichier *`index.html`* :

```
Sur la branche master

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

        modifié :         index.html

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```

Vous pouvez donc maintenant le **stocker** lors du prochain commit. Pour cela vous pouvez utiliser la même commande que le suivi **`git add`**. Par exemple :

`git add index.html`

Vous n'avez plus qu'à faire un commit et vérifier qu'il apparaît dans les logs.

Si vous faites de nouvelles modifications avant de faire le commit, celles-ci ne seront pas stockées. Il vous suffit alors de refaire **`git add`** pour pouvoir les ajouter au commit.

Pour vérifier les modifications avant de les stocker ou de commit, utilisez la commande **`git diff`**. Par exemple :

```
git diff index.html
```

EN CAS D'ERREUR

Se tromper, cela arrive tout le temps et à tout le monde, ça fait même partie de la culture du développeur : se tromper, réparer, recommencer. Généralement, git vous indique les méthodes pour revenir en arrière quand vous vérifiez le statut **`git status`**.

- Vous avez demandé à Git de suivre un fichier que vous ne souhaitiez pas suivre
- Vous avez ajouté au stockage une modification que vous ne souhaitiez pas ajouter

```
git reset HEAD monfichier.html
```

- Vous avez modifié un fichier par erreur

```
git checkout monfichier.html
```

- Vous avez fait un commit avec une faute de frappe
- Vous avez oublié des modifications que vous aimeriez ajouter à ce commit

```
git commit –amend
```

- Effacer plusieurs commits

ATTENTION : Cette opération est irréversible ! Cela réécrit l'historique en effaçant vos derniers commits, il sera impossible de les récupérer.

```
git reset HEAD~1
```

Le 1 signifie que vous allez revenir 1 commit en arrière. Vous pouvez mettre 2,3,4... pour revenir 2,3,4... commits en arrière.

POUR RÉSUMER
Git a un historique de modification des fichiers du projet. Si l'historique et le fichier actuel sont identiques, Git ne fait rien.

Si vous modifiez un ou plusieurs fichiers :

- Git va vous résumer quels fichiers ont été modifiés, par rapport à l'historique.
- Vous allez lui indiquer, parmi tous ces fichiers, lesquels vous souhaitez inscrire dans l'historique.
- Git va les stocker séparément.
- Quand vous êtes prêts, vous pouvez lui dire d'écrire l'historique avec les fichiers que vous lui avez fait stocker.
- Git va écrire l'historique.

À noter :

`git help` vous donne la liste des commandes disponibles pour Git.
Si vous avez un doute sur une commande, vous pouvez faire, à tout moment :

`git help <command>`

ou

`git <command> -h`