GITHUB

VERSIONNER SON CODE

> versionner son code c'est garder les traces de toutes les modifications.

Regrouper les modifications entre elles sous forme de **commit**.

> **Commit :** série de modifications sur un groupe de fichiers associés à un message.

```
git status
# Permet de connaître dans quel état se trouve la branch courante.
```

Github

C'est un service en ligne qui permet d'héberger ses repositories. C'est actuellement le plus populaire et est utilisé pour des projets personnelle mais aussi par des entreprises et des startups. On peut s'inscrire gratuitement.

Tout le monde à accès aux repositories public, il est possible d'apporter des modifications au projets opensources :

```
git clone www.url_du_repositories.github.com
```

Github permet d'avoir un aperçu visuel et intuitif de ses repositories. Il est possible d'interragir directement avec depuis cette interface graphique.

Initialisation et commandes de base

> **Un dépôt ou repositories** en anglais est l'endroit où le code source et les modifications apportées aux fichiers sont stockés (source : wikipedia).

Initialiser le répertoire de travail git :

```
git init
```

On peux maintenant commencer à travailler et à apporter des modifications à notre repository.

Pour enregistrer les modifications, il faut les ajouter à l'index :

```
# Ajouter le fichier file_name.md
git add file_nam

# Ajouter toutes les modifications
git add --all
```

L'indexe est a liste des fichiers qui sont gardés en mémoire par git.

Ensuite les changements doivent être *commited* :

```
git commit -m "ajout du fichier file_name.md"
```

Envoyer son travail sur github

On vérifie bien le **status**, pour bien être au courant des modifications qui vont être envoyé sur github. Les modifications qui sont ajoités à l'indexe seront envoyés :

```
# Sans arguments envoie la branche courante
git push
```

Récupérer des modifications

Github est un outil idéale pour les travaux colaboratifs. Plusieurs personnes peuvent travailler sur un même repositories. La commande **pull** permet de récupérer les modifications qui ont été envoyé sur github par vos différentes machines ou par les différents colaborateurs.

```
git pull
```

Historique des commits

Afficher l'historique des commits :

```
git log
```

Et voilà à quoi ressemble un commit :

```
commit 35fb29f0b3c51930608ad1f7931b2dee63b822ec
Author: Mon_pseudo <mon_mail@hotmail.fr>
Date:   Tue May 22 20:16:00 2018 +0200

    Ajout du fichier file_name.md
```

Chaque commit possède un identifiant unique, ici :

35fb29f0b3c51930608ad1f7931b2dee63b822ec

On peut revenir en arrière dans l'historique des commits avec l'identifiant du commit et la commande **checkout** :

```
git checkout 35fb29f0b3c51930608ad1f7931b2dee63b822ec
```

Les branches

Créer des branches

Il est intéressant de travailler avec différentes branches pour se détacher un peu du projet principal. Celà permet par exemple de faire des tests sans avoir à modifier le code principal.

```
# Lister les branches
git branch

# Créer un branch test
git branch test

# Changer de branch pour aller dans lka branch test
git checkout test
```

Le code principal se trouve par défaut dans la branch **master**.

> Les modifications faites sur une branch n'affectent pas les autres.

Il est possible d'apporter des modifications à une branch et d'avoir envie de changer de branch sans commit les modification de la branch en cours :

```
# Mettre de côté les modifications de la branch courantes
git stash
# Il est maintenant possible de changer de branch

# Reprendre ce que l'on a mis de côté.
git stash pop
```

La commande **stash** permet de garder un hisorique plus clair.

Fusionner les branches

Fusionner des branchs permet d'apporter les modifications d'une branch vers une autre.

```
# ajoute les modifications de ma_branch vers la branch courante.
git merge ma_branch
```

Les conflits

Résoudre les conflits

Dans la vie de tous les jours, travailler à plusieurs sur un même projet peut amener des conflits si des modifications sont apportées au même endroit et/ou

par des développeurs différents.

Les conflits peuvent être résolu simplement avec l'éditeur de texte. Mais il existe aussi des interfaces graphiques et des manières visuelle de résoudre les conflits.

Les modifications des utilisateurs

```
#affiche l'historique des modifications apportés par les utilisateurs au fichier file_name.
git blame file_name.md

# montre de manière visuelle les modifications qui ont été apporté lors d'un commit précis
git show id_du_commit
```

Il est aussi possible d'utiliser la commande **log** mais il est possible que ce soit plus difficile car trop de bruit présent.

Ignorer des modifications

Dans certains cas on a des fichiers que l'on ne souhaite pas ajouter à git.

On crée donc un fichier **.gitignnore** qui est fichier texte dans lequel on liste les fichiers que l'on souhaite ignorer.

les astuces

```
# Créer la branch nouvelle branch et se positionner dessus (= git branch + git checkout)
git checkout -b nouvelle_branch
```

Si un commit sur un fichier *file_name.md* a déjà été *add*, il a été ajouté dans l'indexe. Un nouveau commit concernant de nouvelles modifications sur le fichier file_name.md avec l'argument **-a** permet d'ajouter les modifications des fichiers qui sont dans l'indexe sans avoir à relancer la commande *git add*.

```
# add + commit
git commit -am "nouvelle modification du fichier file_name.md"
```

Les bonnes pratiques

- commit réguliairement les modifications apportés
- penser à pull les modifications de ses collègues.
- travailler avec des branches
- éviter de travailler à plusieurs sur les mêmes fichiers
- rester en contact avec ses colaborateurs pour résoudre les conflits