Git

> Git est un système de contrôle de version qui a été inventé et développé par Linus Torvalds, également connu pour l'invention du noyau Linux, en 2005. Il s'agit d'un outil de développement qui aide une équipe de développeurs à gérer les changements apportés au code source au fil du temps.

Par **Rachid EDJEKOUANE** ⭐

Sommaire

---

1. Configuration de Git

On initialise un nom + email, qu'on peut modifier à tout moment avec :

```
git config --global user.name "John doe"
git config --global user.email "johndoe@gmail.com"
```

Active la colorisation de la sortie en ligne de commande :

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

Ouvre le fichier de configuration globale dans un éditeur de texte pour une édition manuelle :

```
git config --global --edit
```

Définit l'éditeur de texte utilisé par les commandes pour tous les utilisateurs de la machine. (editor) doit être la commande qui lance l'éditeur souhaité :

```
git config --global core.editor [editor]
```

---

2. Créer des dépôts

Crée un dépôt local à partir du nom spécifié :

```
git init [nom-du-projet]
```

Crée un dépôt local à partir du nom spécifié :

```
git clone [url]
```

Crée une connexion avec un dépôt distant :

```
git remote add [name] [url]
```

Pour envoyer le contenu du dépôt local vers le dépôt distant, tout en liant les branches main à main avec -u(upstream) :

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

---

3. Commande de base

Informe du statut de notre dépôt, liste tous les nouveaux fichiers et les fichiers modifiés à commiter :

```
git status
```

Ajoute un instantané du fichier en zone de transit (staging area), en préparation pour le suivi de version :

```
git add [fichier]
```

Enregistre des instantanés de fichiers de façon permanente dans l'historique des versions :

```
git commit -m "Message descriptif du commit"
```

Enleve le fichier de l'index, mais conserve son contenu :

```
git reset [fichier]
```

Montre l'historique des versions pour la branche courante :

```
git log
```

Pour afficher l'historique de commits sur une ligne :

```
git log --oneline
```

Pour rattraper un commit dans lequel on aurait oublié d'intégrer un fichier, on ajoute d'abord le ou les fichiers oubliés :

```
git commit --amend
```

Afin de visualiser les changements qui ont eu lieu dans les fichiers modifiés, on peut éxécuter :

```
git diff
```

---

4. Branch

Liste toutes les branches locales dans le dépôt courant :

```
git branch
```

Crée une nouvelle branche :

```
git branch [nom-de-branche]
```

Bascule sur la branche spécifiée :

```
git switch [nom-de-branche]
```

Crée une branche et de se déplace dessus intantanément :

```
git switch -c [nom-de-branche]
```

Change le nom d'une branche, on navigue dessus, puis on lance :

```
git branch -m [nouveau-nom-de-branche]
```

Fusionne une branche, on se déplace d'abord sur la branche qu'on veut fusionner avec une autre puis :

```
git merge [nom-de-branche]
```

Afin de supprimer une branche, on doit se situer sur une branche différente de celle qu'on veut supprimer, puis :

```
git branch -d [nom-de-branche]
```

De base, vous pouvez tomber sur trois types de fusion :

Fusion simple "Fast Forward", les historiques de commits se lient sans conflit, car il y avait des changements seulement sur une branche.

Fusion par récursion, lorsque il y a des commits différents sur plusieurs branches qu'on "merge", mais que ces commits ne rentrent pas en conflit.

Fusion avec conflit, lorsqu'il y a des commits différents sur plusieurs branches que l'on "merge" et que ces commits modifient des choses similaires(ajout de code à la même ligne, changement du code initial, supression de fichiers) .

---

5. Checkout

Déplace parmi les commits avec checkout :

```
git checkout [id-du-commit]
```

Pour revenir en haut de la liste de commits, nous pouvons faire au choix :

```
git switch [nom-de-branche]

git checkout [nom-de-branche]
```

---

6. Git tag

On se déplace d'abord sur le commit que l'on veut "taguer", puis :

```
git tag [nom-du-tag]
```

Nous pouvons désormais nous déplacer vers le commit du tag correspondant :

```
git checkout [nom-du-tag]
```

Nous pouvons visualiser la liste de tags :

```
git tag
```

Nous pouvons supprimer un tag :

```
git tag -d [nom-du-tag]
```

---

7. Reset et Restore

Si on a effectué des changements et qu'ils ne nous plaisent plus :

```
git restore [nom-du-fichier]
```

Cela permet de rapidement remettre à zéro un fichier sur lequel on travaillait, ça ne veut pas dire que le fichier sera forcement vide, il reaffiche simplement le contenu du commit sur lequel on se trouve.

Nous pouvons également supprimé les fichiers en zone de transit avec :

```
git restore --staged index.html
```

Pour supprimer des commits jusqu'à un commit passé :

```
git reset idCommit
```

Pour supprimer des commits jusqu'à un commit passé tout en supprimant les changements qu'il peuvent avoir été éffectués :

```
git reset --hard idCommit
```

Pour sauter un ou des commits et atterir dans un nouveau commit contenant le contenu d'un ancien commit :

```
git revert idDuCommitÀRetrouver
```

---

8. Cherry-pick

Pour intégrer seulement un commit ou des commits qui nous intéressent mais pas une branche entière :

```
git cherry-pick idDuCommit
```

⬆️ retour au sommaire