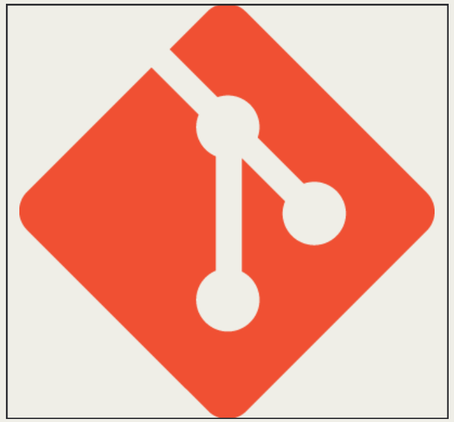


GIT

(LE TRUC QUI PERMET LA GESTION DE VERSION)

UVSQ université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

INTRODUCTION À

GIT

SAE 5.1 - BUT3FA

ALEXIS CHOISY
ELIAS MOUSSAMIH
THOMAS PAVLETIC
DAVID GRONDIN
SIDY SOUMARE
FLORIAN DE SOUSA

SOMMAIRE

C'EST QUOI GIT ?

Les fondamentaux

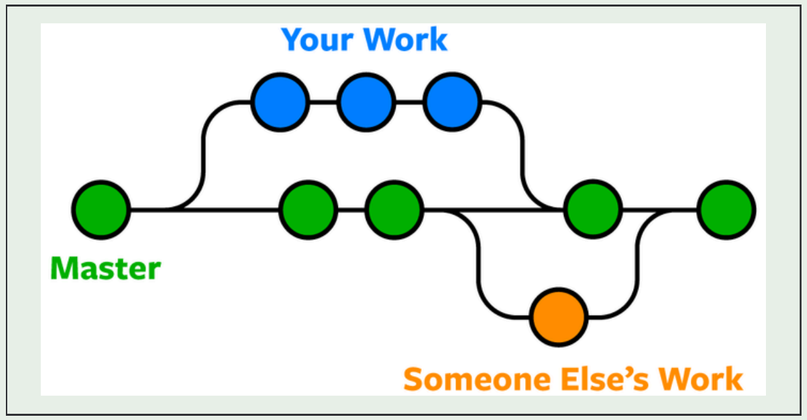
Git est un système de **gestion de version** décentralisé qui permet à plusieurs personnes de **travailler simultanément sur un projet** (et d'éviter l'image sur la gauche).

Il faut voir Git comme un outil avec des **conventions flexibles** plutôt qu'une **technologie rigide**. Vous conservez un libre arbitre sur ce que vous faites. L'essentiel est que le versionnage du projet reste **facile à comprendre**, que ce soit **pour un nouvel arrivant** ou **pour reprendre du code**.

(À noter que Git ne s'utilise pas exclusivement pour du code : vous pouvez très bien créer un projet Git pour stocker vos photos de famille)

C'EST QUOI GIT ?

Les branches



Toute branche découle de la branche principale (appelée **master** ou **main**) et a pour but de développer **une fonctionnalité spécifique** d'un projet. En général, on protège la branche principale car elle doit rester fonctionnelle. Et on travaille seul sur sa propre branche pour **éviter d'écraser le travail des autres**. C'est pourquoi il est utile de créer des branches dérivées, qui seront **fusionnées** une fois la fonctionnalité développée.

C'EST QUOI GIT ?

Les plateformes

LES PLUS CONNUES :

GitHub

GitLab

Bitbucket

CELLE QUE VOUS ALLEZ UTILISER POUR LES TPS :

Gitea

(Tout ce que vous ferez dans le cadre de vos TPs est reproductible sur toutes les plateformes utilisant Git)

INSTALLATION DE GIT

SUR WINDOWS

> Installer le .exe sur le site de Git

ou

> winget install --id Git.Git -e --source winget

SUR LINUX

> apt-get install git / yum install git / dnf install git / pacman -S git / ... (Selon votre gestionnaire de paquets)

UVSQ université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

C'EST QUOI UN COMMIT ?

Un commit permet de **sauvegarder l'état actuel d'une branche**. Il faut y **joindre un message** qui décrit les **changements effectués** afin de **pouvoir suivre l'historique du projet**. Chaque commit possède un **ID unique**, qui permet de revenir à une version précédente si nécessaire.

SAVOIR QUELS FICHIERS ONT SUBI UNE MODIFICATION DEPUIS LE DERNIER COMMIT

> git status (La commande t'indiqueras d'autres commande pour réaliser certaines actions tel qu'ajouter un fichier dans le commit)

AJOUTER/SUPPRIMER DES FICHIERS POUR LE COMMIT QUI VA SUIVRE

> git add <nom-du-fichier> / git restore --staged <nom-du-fichier> (C'est contre-intuitif mais il faut aussi **git add** un **fichier qu'on souhaite supprimer dans le commit**)

CRÉER UN COMMIT EN SPÉCIFIANT CE QUI A ÉTÉ RÉALISÉ

> git commit -m "MessageDeCommit" | Ex : > git commit -m "Ajout de la méthode manger dans la classe Humain"

VISUELLEMENT:

ACTIONS SUR LES BRANCHES

INITIALISER UN DÉPÔT GIT LOCAL (**Nécessaire** avant de se connecter à un dépôt distant)

> git init

AJOUTER UN DÉPÔT DISTANT (**Nécessaire** pour récupérer les branches distantes)

> git remote add origin <url-du-depot>

RÉCUPÉRER LOCALEMENT UNE BRANCHE DISTANTE

> git pull origin <nom-de-la-branche>

POUSSER UNE BRANCHE LOCALE SUR LE DÉPÔT DISTANT

> git push -u origin <nom-de-la-branche>

ACTIONS SUR LES BRANCHES

LISTER LES BRANCHES DISTANTES

> git branch -r

LISTER LES BRANCHES LOCALES / CRÉER UNE NOUVELLE BRANCHE

> git branch
> git branch <nom-de-la-branche>

CHANGER DE BRANCHE / COMMIT

> git checkout <nom-de-la-branche **ou** id-du-commit>

RENOMMER UNE BRANCHE LOCALE

> git branch -m <nouveau-nom-de-la-branche>

ACTIONS SUR LES BRANCHES

VOIR L'HISTORIQUE D'UNE BRANCHE

> git log <nom-de-la-branche>

> git log --stat --oneline <nom-de-la-branche> (affiche l'historique avec **plus de détails**)

ÉCRASER LE DERNIER COMMIT PAR UN AUTRE

> git commit --amend (gardera le message du commit actuel)
> git commit --amend -m "Nouveau message de commit"

FAIRE LA DIFFÉRENCE ENTRE DEUX COMMITS/BRANCHES

> git diff <id-commit ou id-branche> <id-commit ou id-branche>

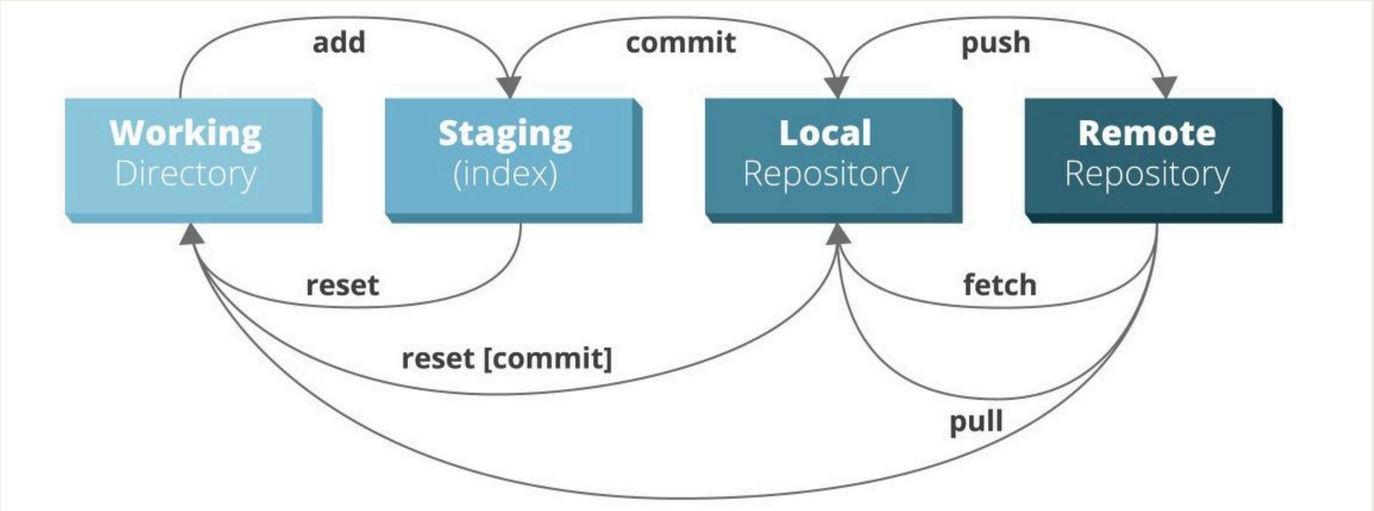
SCHÉMA EXPLICATIF

WORKING DIRECTORY

C'est le **répertoire actuel** dans lequel tu travaille. **Il ne tient pas compte des branches et toutes les notions liées a Git.**

STAGING

Zone intermédiaire où tu **prépares les fichiers modifiés** avant de les valider (commit).



LOCAL REPOSITORY

C'est la **copie du dépôt distant en local**. Il contient les **branche locales**. Avant de pusher, il est préférable qu'elle soit à jour.

REMOTE REPOSITORY

C'est **répertoire distant** hébergé sur Gitea. Il contient les **branches distantes.**

UVSQ université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

FUSIONS DE BRANCHES

EXEMPLE DE MERGE CONFLICTUEL

BRANCHE1 :

main.py :

```
def chat():
        return "miaou"
```

Nous nous situons sur la **branche1** et souhaitons y fusionner la **branche2**.

BRANCHE2 :

main.py :

```
def chien():
        return "wouf"
```

Il y aura un **conflit** car le fichier main.py est **différent d'une branche à l'autre** (ligne 1 et 2 sont différentes).

> git merge branche2

Si on exécute la commande pour merge il faudra résoudre le problème suivant (Il est possible d'abandonner le merge en faisant la commande suivante : git merge **--abort**)

"**<<<<<<< HEAD**" : indique que les lignes suivantes sont issues de la branche actuelle
"**=======**" : sépare les deux versions du code en conflit
"**>>>>>>> branche2**" : indique que les lignes précédentes sont issues de la branche "branche2"

main.py :

```
<<<<<<< HEAD
def chat():
        return "miaou"
=======
def chien():
        return "wouf"
>>>>>>> branche2
```

FUSIONS DE BRANCHES

EXEMPLE DE MERGE CONFLICTUEL

Pour finir la fusion de ces deux branches, il faut adapter le code aux besoins, ici on souhaite garder les deux méthodes. Donc, il faudra juste supprimer le formatage issu du merge. Ainsi on obtiendra un fichier main.py qui ressemble à cela :

main.py :

```
<<<<<<< HEAD
def chat():
        return "miaou"
=======
def chien():
        return "wouf"
>>>>>>> branche2
```

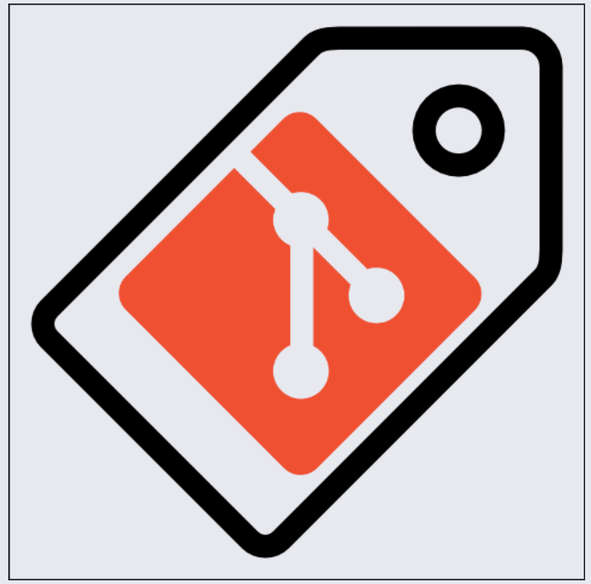
main.py :

```
def chat():
        return "miaou"
def chien():
        return "wouf"
```

Une fois qu'il n'y a plus de **conflits** dans le fichier, n'oubliez pas de **commit** afin de **sauvegarder les changements**. C'est d'ailleurs l'une des rare fois où il n'est pas nécessaire de spécifier un message lors du commit, il y en aura un automatique.

LES ÉTIQUETTES (TAGS)

Les étiquettes (tags) sont des marqueurs spécifiques qu'on attache à un commit afin de spécifier une version clé du projet qui sera par ailleurs visible dans l'historique du dépôt. Ils sont généralement utiles pour déclarer une version stable du projet dans le cadre d'une release. On retrouve deux types de tags :



TAG LÉGER

Il s'agit d'un pointeur vers un commit qui ne contient **aucune information supplémentaire** (tel qu'un message ou l'auteur du tag). Il contient uniquement le **nom du tag**.

TAG ANNOTÉ

Il s'agit d'un pointeur qui est **plus riche** que le tag léger en information. Il peut contenir des informations supplémentaires telles qu'un message, l'auteur du tag, la date ainsi que d'autres données qui permettent **d'enrichir le tag**.

UVSQ université PARIS-SACLAY
IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

LES ÉTIQUETTES (TAGS)

CRÉER UN TAG LÉGER

> git tag <nom-du-tag>

CRÉÉR UN TAG ANNOTÉ

> git tag -a <nom-du-tag> -m “mon message de commit”

AFFICHER LES DÉTAILS D’UN TAG ANNOTÉ

> git show <nom-du-tag>

LISTER LES DIFFÉRENTS TAGS

> git tag

COMPARER DEUX TAGS

> git diff <nom-du-premier-tag> <nom-du-deuxieme-tag>

SUPPRIMER UN TAG LOCAL

> git tag -d <nom-du-tag>

SUPPRIMER UN TAG DISTANT

> git push origin --delete <nom-du-tag>

À VOUS DE JOUER SUR LES

TPS