**Branches Git**

Les branches sous Git sont incroyablement légères. Elles sont simplement des références sur un commit spécifique -- rien de plus. C'est pourquoi beaucoup d'enthousiastes répètent en cœur :

des branches le plus tôt possible, et des branches souvent

Parce qu'il n'y a pas de surcoût (stockage/mémoire) associé aux branches, il est facile de diviser son travail en de nombreuses branches plutôt que d'avoir quelques grosses branches.

Nous verrons comment les branches et les commits interagissent quand nous les utiliserons ensemble. Pour l'instant, souvenez-vous qu'une branche est un moyen d'exprimer "Je veux inclure le contenu de ce commit et de tous les commits parents."

## Branches et Merges

Super ! Nous savons désormais comment faire des commits et des branches. Maintenant nous devons apprendre comment combiner ensemble les contenus de deux branches différentes. Ceci nous permettra de créer une nouvelle branche, développer une nouvelle fonctionnalité sur cette dernière, puis intégrer cette fonctionnalité en combinant le contenu de cette branche de développement à la branche d'origine (main par exemple).

La première méthode que nous allons voir pour combiner le contenu de deux branches est git merge. Faire un 'merge' avec Git crée un commit spécial qui a deux parents. Un commit avec deux parents indique en susbtance "Je veux inclure le contenu de ce parent et le contenu de cet autre parent, et l'ensemble de leurs parents."

C'est plus facile en visualisant, regardons dans la vue suivante.

## Git Rebase

La seconde façon de combiner les contenus de deux branches est rebase. Rebase prend un ensemble de commits, les "recopie", et les ajoute en bout de chaîne à un autre endroit.

Bien que cela puisse sembler compliqué, l'avantage de rebase est de permettre d'obtenir une simple séquence linéaire de commits. Les logs/l'historique du dépôt seront bien plus propres si seul rebase est autorisé (plutôt que merge).

Voyons rebase en action…

## Se déplacer dans Git

Avant que nous découvrions quelques-unes des fonctionnalités les plus avancées de Git, il est important de comprendre les différents manières de se déplacer dans l'arbre des commits qui représente votre projet.

Une fois que ces déplacements seront aisés, votre puissance avec les autres commandes de Git sera amplifiée !

## HEAD

Premièrement nous avons parlé de "HEAD". HEAD est le nom symbolique pour le commit sur lequel nous nous situons actuellement -- plus simplement c'est le commit sur lequel nous travaillons.

HEAD pointe toujours sur le commit le plus récent dans l'arbre des commits. La plupart des commandes Git qui modifient l'arbre des commits vont commencer par modifier HEAD.

Normalement HEAD pointe sur le nom d'une branche (comme bugFix). Quand vous effectuez un commit, le statut de bugFix est modifié et ce changement est visible par le biais de HEAD.

## Annuler des changements avec Git

Il y a de nombreuses façons d'annuler des changements avec Git. De même que pour les commits, annuler des changements avec Git est à la fois un aspect bas-niveau (gestion des fichiers et morceaux de fichiers) et un aspect de plus haut niveau (comment les changements sont effectivement annulés). Nous allons nous intéresser à ce dernier point.

Il y a principalement deux façons d'annuler des changements avec Git : l'une est git reset et l'autre est git revert. Nous allons maintenant voir chacune de ces façons.

GIT - Cours ENI

# Présentation

Git : gestion de versions dans les projets

Objectif : pouvoir restaurer une version antérieure d’un fichier soit pour un retour arrière (annuler) soit pour faire des comparaisons de l’état des fichiers, soit voir qui a modifie quoi, soit déterminé quel changement a causé des problèmes.

VCS : plusieurs types : locaux, centralisés (code source sur un serveur, copie pour chaque personne puis envoie modif), distribués (git, on ne dépend plus de serveur central, tout devient un référentiel, on n’a pas un seul endroit où se trouve le code, on a un ensemble de nœuds, cluster. On a plusieurs serveurs. Chaque serveur est un nœud. L’avantage est que l’on a plus un point unique de panne. On a tt l’historique du projet. Si le serveur est perdu, n’importe quel dépôt client peut servir de serveur.

Git : système décentralisé (DCVS) Chaque développeur a son dépôt.

Les fichiers sont identifiés par un hash sha-1 : on prend le fichier et on en extrait une signature unique. Les fichiers sont dupliqués de version en version.

# Mécanisme de Git

Génération d’un instantané a chaque changement dans le projet.

Référence vers la version antérieure si un fichier n’est pas modifié.

Les fichiers vont avoir différents états :

* Sans état : fichier non connu du référentiel
* Validé : fichier versionné et en sécurité. L’état des fichiers quand on récupère un projet par exemple.
* Modifié : A partir du moment où on touche à un fichier, il va être à l’état validé. Git n’a pas duplique le fichier, il attend que l’utilisateur lui dise de le dupliquer
* Indexé : Le fait de passer de l’état de modifie à indexé va permettre d’enregistrer les changements, de signer le fichier pour pouvoir le versionner et en enregistrer une version. Les changements sont enregistrés, le fichier peut être validé.

Dépôt distant : Chaque poste de dvp a une copie du dépôt. On va devoir synchroniser le dépôt local avec le dépôt distant pour échanger des infos : récupérer le travail des autres développeurs ou donner mon travail. Le rôle du dépôt distant est de centraliser les sources et permettre la collaboration. Ceux sont les développeurs qui choisissent quand ils vont synchroniser leur code avec les dépôts distants.

Une image contenant texte, horloge

Description générée automatiquement

Nb : Il est également possible de prendre les modifications et les appliquer directement au dépôt d’un autre développeur. Un autre développeur pourrait venir chercher les modif depuis mon dépôt. Il n’y a pas de serveur central ! Dans la réalité, on a un serveur central.

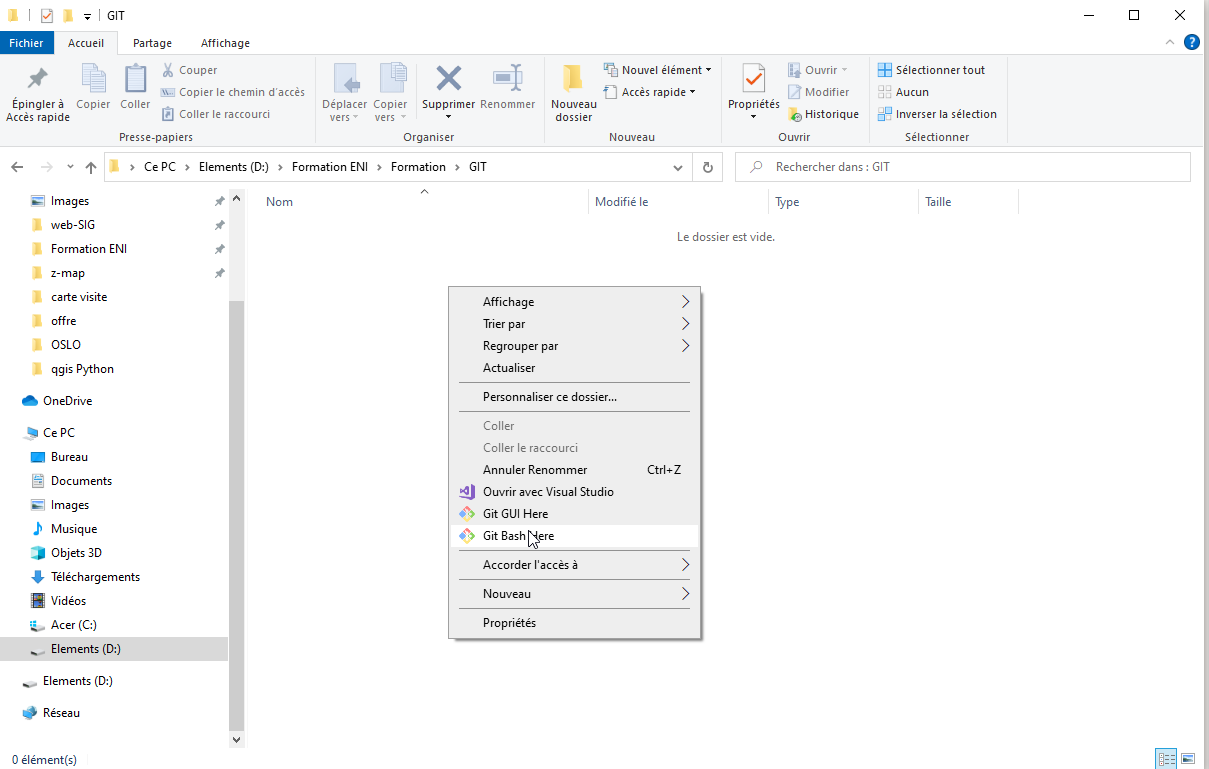
**Manipulation de Git**

1. **Initialiser un projet**

Installer Git sur git-scm.com



**Configurer GIT**



Taper la commande : git

Si on ressort une documentation, c ok

Une image contenant texte, plaque, capture d’écran

Description générée automatiquement

git --version : version utilisée

git --help : renvoie a la documentation de base et indique les commandes

Une image contenant texte

Description générée automatiquement

Configurer:

git config -h : documentation succinct avec -h (valable pour toutes les commandes)

Une image contenant texte

Description générée automatiquement

git config --global ou git config --system : comportement de git pour tous les utilisateurs, pour tout mon poste de travail

git config - - local : comportement uniquement pour le dépôt dans lequel je suis

Configurer le nom et l’email (on peut me contacter) :

git config - -global user.name « Tara HH »

git config - -global user.mail « hachem.huet.tara@gmail.com »

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Il y a également la possibilité de faire :

git config - - global - - edit : va nous ouvrir vi et explique les clés positionnées pour paramétrer git.



Une image contenant texte

Description générée automatiquement

Ou on peut préciser l’éditeur que l’on souhaite utiliser par defaut. Ex : vim avec 

Nb : pour sortir, taper : Echap ZZ

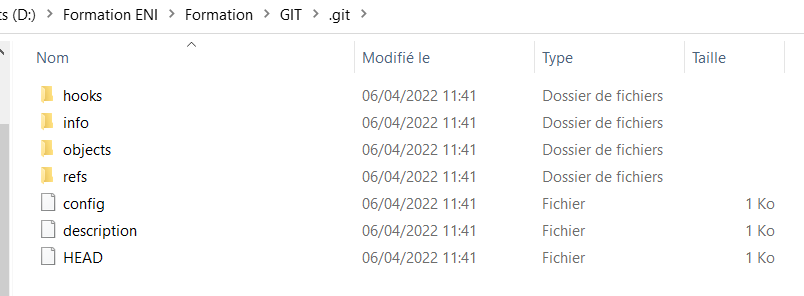
**Initialiser le projet**

git init

Une image contenant texte

Description générée automatiquement

Git initialise un dépôt vide dans le répertoire. On obtient un dossier caché qui contient les infos uniques a la mécanique de git : les hooks (détermine des comportements supplément au commit, fetch…), info sur le projet lui-même, les nœuds (fichiers copies de version en version), refs et autre



1. Indexer les changements

On voit désormais entre parenthèses « master » le nom de la branche sur laquelle on est en train de travailler.

Avec vi, on va créer un fichier source.js qui contient un code source quelconque :



Une image contenant texte

Description générée automatiquement

git status : renseigne sur le statut du dépôt actuel. Ici, git dit que l’on se trouve sur la branche master et on a des fichiers untracked cad qui ne sont pas référencés par git. Le fichier source.js n’est pas référencé par git et git ne prendra pas l’initiative de référencer ce fichier. C a nous de le faire. On doit le versionner. Pour se faire on utilise la commande git add nom du fichier

Une image contenant texte

Description générée automatiquement

git add source.js : on ajoute le fichier source.js a notre référentiel.

Une image contenant texte

Description générée automatiquement

Avec git status, on voit que le fichier est prêt a être comité. On voit que dans le commit il y aura ce fichier qui est prêt a être commité

Une image contenant texte

Description générée automatiquement