# Initialisation de Git

Renseigner nom + mail

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

Lister la config

git config –list

# Dépôt distant

Créer une clé SSH

ssh-keygen -t rsa -b 4096 -C "[your github's email]"

Permettre à Git de pointer vers un dépôt distant

git remote add nom\_local\_depot <https://github.com/OpenClassrooms-Student-Center/ProjetOpenSource.git>

Cloner le depot local

git clone https://github.com/OpenClassrooms-Student-Center/ProjetOpenSource.git

# Organisation d’un dépot

Lister les branches

git branch

Créer une branche

git branch nom\_branche

Aller sur une branche

git checkout nom\_branche

Supprimer une branche

git branch -d brancheTest

Réaliser un commit

git commit

Réaliser un commit avec un commentaire

git commit -m “J’écris mon commentaire”

Ajouter un fichier

git add PremierFichier.txt

git commit

Voir le statut des fichiers

git status

# Les remises

Créer une remise

git stash

La remise va permettre de mettre vos modifications de côté, le temps de créer votre nouvelle branche et ensuite appliquer cette remise sur la nouvelle branche.

appliquer la remise (quand on est sur la bonne branche

git stash apply

lister les remises

git stash list

appliquer une remise

git stash apply stash@identifiant\_de\_la\_remise

supprimer une remise

git stash drop

supprimer toutes les remises

git stash clear

# Les modifications

Afficher la liste des commit

git log

Supprimer de la branche master le dernier commit

git reset --hard HEAD^

HEAD^ indique le dernier commit

Le commit peut être appliqué à une nouvelle branche avec

git reset --hard id\_commit

id\_commit : 8 premier caractère SHA

Changer le message du dernier commit

git commit --amend -m "Votre nouveau message de commit"

Rajouter un fichier à mon dernier commit

git add FichierOublie.txt

git commit --amend --no-edit

Annuler un commit public en créant un nouveau commit

git revert HEAD^

Annuler un commit sur une branche privée

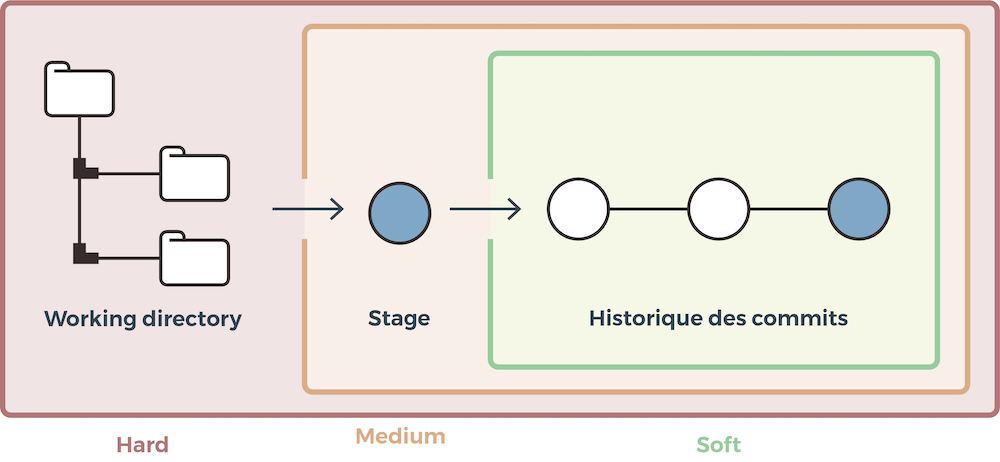
git reset

Générer un duo de clé SSH

ssh-keygen -t rsa -b 4096 -C [johndoe@example.com](mailto:johndoe@example.com)

Retrouver la clé dans C:\Users\VotreNomD'Utilisateur\

Trois type de réinitialisation avec git reset



**--soft :** permet de se placer sur un commit spécifique afin de voir le code à un instant donné ou créer une branche partant d'un ancien commit.

**--mixed : (option par défaut) revenir au commit cible en créant un nouveau commit. les modifications peuvent donc être retrouvées**

**--hard : On supprime tout ce qui a été fait depuis le commit cible**

revenir à n’importe quel commit en supprimant tout ce qui s’est passé après

git reset notreCommitCible --hard

afficher le log des commits, mais aussi toutes les autres actions que vous avez pu faire en local

git reflog

Revenir à une action donnée

git checkout e789e7c

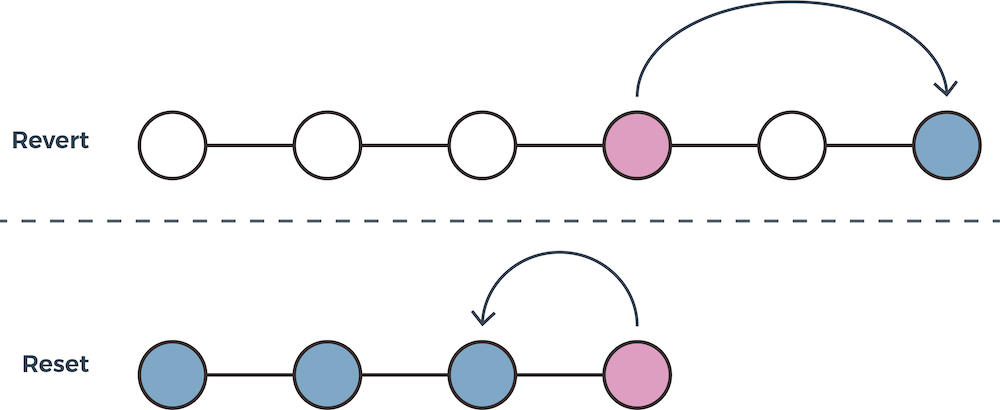
Examiner **le contenu d’un fichier ligne par ligne** et de déterminer la date à laquelle chaque ligne a été modifiée, et le nom de l’auteur des modifications

git blame monFichier.txt

fusionner les commit cible d’autres branches à la branche master

git cherry-pick d356940 de966d4

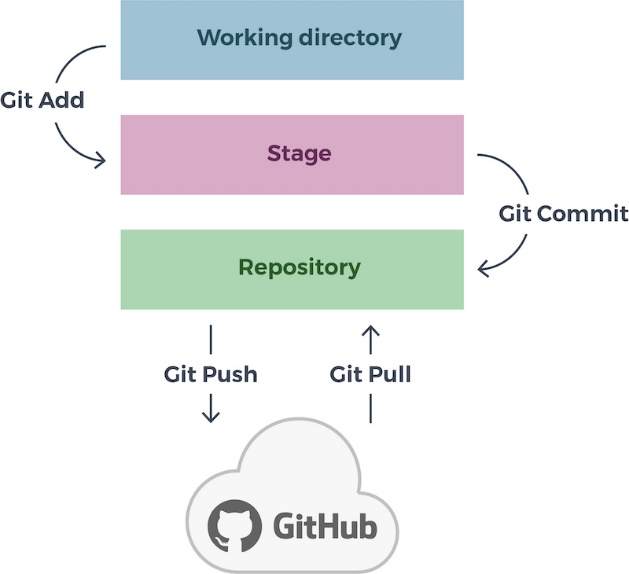
Différence entre reset et revert



Corrigez les erreurs sur le dépôt local

Git gère les versions de vos travaux locaux à travers 3 zones locales majeures :

* le répertoire de travail (working directory/WD) ;
* l’index, ou stage (nous préférerons le second terme) ;
* le dépôt local (Git directory/repository).



# Arbre Git et sa structure

trois principaux types d'objets sont :

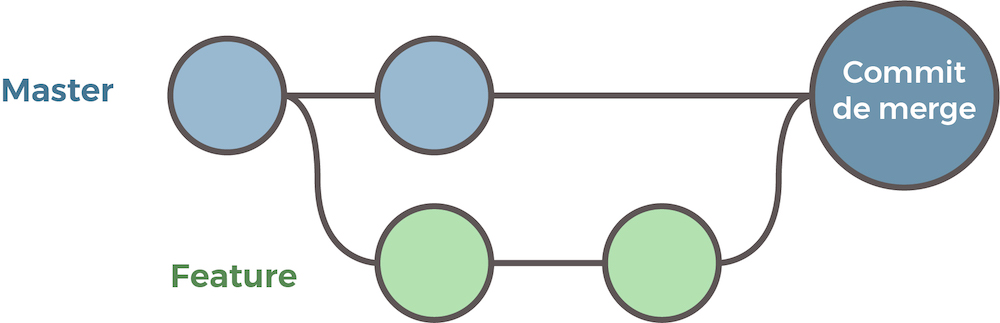
* le "tree" ou l'arbre Git qui est une forme de répertoire. Il va référencer une liste de trees et de blobs (sous-répertoires et fichiers) ;
* le "commit" qui va pointer vers un arbre spécifique et le marquer, afin de représenter son état à un instant donné ;
* Le "blob" qui représente en général un fichier (Binary Large Object)

fusionner le travail fait sur différentes branches

git merge

Dans le détail

git checkout fctMerge  
git status  
git checkout master  
git merge fctMerge



Charger une branche du depot local vers le hub en ligne

git push -u origin nom\_branche

récupérer un fichier du hub vers le local

git pull

Git fetch, contrairement à Git pull, va aller **chercher les modifications** sur le dépôt distant mais **ne va pas les fusionner avec nos modifications locales**.

git fetch

# Rebase

effectuer un rebase : reorganization d’une branche au sein de la branche master

git rebase

rebase interactif, on choisit dans quel ordre on place les commit

git rebase -i

git rebase -i HEAD~3

drop 58gfbg56 commit3

pick 14hg58g1 commit2

pick 25frgf83 commit1

drop pour supprimer un commit

ensuite on peux ordonner les commit comme on veut avec la commande pick devant

## Modifier un message de validation

git rebase -i HEAD^

edit 54dfiosd

git commit --amend

git rebase --continue

## Faire un Squash

C’est un regroupement de plusieurs commit

git rebase -i HEAD~3

pick 57dcsd58 Création du formulaire

squash 58gkbg56 Design Formulaire

squash 899hbg78 Correction du formulaire

## Bisect :

git bisect start [bad] [good]

Si le commit ne présente pas le bug :

git bisect good

Si le commit présente le bug :

git bisect bad

## Sous arborescence et sous module

Ajouter un sous module

git submodule add https://github.com/etudiantOC/ProjetSubModule dossier/destination

Ajouter une sous arborescence

git subtree push -P monRépertoire git@mon-serveur-git:group/projet.git master

# Initialiser GitFlow

## Organisation des branches du gitflow

* **master** : versions officielles publiées pour les utilisateurs
* **develop** : versions en attente de mise en production pour tester les nouvelles fonctionnalités
* **feature** : ces branches permettent de commencer à travailler sur de nouvelles fonctionnalités. elle est tiré de la branche develop et est comité à la fin à develop
* **hotfix** : permet de corriger les beug en production. elle est créée à partir de master. une fois la branche terminée, elle est mergée dans les branches develop et master.
* **release** : créée à partir de develop en cas de livraison imminente.dans un gros projet, une fois que toutes les fonctionnalités d’une version ont été créées, on crée la branche release. Elle permet ensuite de réaliser les tests. à la fin, elle doit être mergée sur les branche develop et master

créer un dossier gitflow

lancer

git flow init

git flow feature start main

# Le marketplace GitHub

## Sécurité

**WhiteSource Bolt** for GitHub est une application gratuite, qui analyse en permanence tous vos dépôts, détecte les vulnérabilités des composants open source et apporte des correctifs. Il prend en charge les référentiels privés et publics

## Gestion de projets

**ZenHub** est le seul outil de gestion de projet qui s'intègre de manière native dans l'interface utilisateur de GitHub. ZenHub est un outil de gestion de projet agile fonctionnant par sprint et générant des rapports assez poussés.

## Intégration **continue**

**Travis CI** permet à votre équipe de tester et déployer vos applications en toute confiance. Très polyvalent, il s'adapte aux petits comme aux grands projets

## Comparer 2 fichiers avec interface graphique

**WinMerge** et **Meld**.