**GIT BASH**

**Initialisez Git**

* Pour renseigner son nom et son adresse e-mail :

git config --global user.name "Loetitia Rabier"

git config --global user.email [loe.rabier@gmail.com](mailto:loe.rabier@gmail.com)

Si vous souhaitez par contre, pour **un projet spécifique**, changer votre nom d’utilisateur, vous devrez repasser cette ligne mais sans le --global.

* Pour vérifier que les paramètres sont bien pris en compte :

git config –list

* Pour activer les couleurs afin d’améliorer la lisibilité des différentes branches :

git config --global color.diff auto

git config --global color.status auto

git config --global color.branch auto

* Par défaut, Git utilisera Vim comme éditeur et Vimdiff comme outil de merge. Vous pouvez les modifier en utilisant :

git config --global core.editor notepad++

git config --global merge.tool vimdiff

* créer un dépôt local vide pour accueillir un nouveau projet

se placer dans le répertoire (cd …) puis commande git init

cd Documents/git/PremierProjet

git init

🡺 ajout répertoire masqué .git

### Accédez à un dépôt distant OC

* Allez sur GitHub à l’adresse du dépôt distant, cliquer sur code puis copier l’adresse de la partie clone

Ex : <https://github.com/OpenClassrooms-Student-Center/ProjetOpenSource.git>

* Permettre de dire au dépôt que l’on pointe vers le dépôt distant :

git remote add OC <https://github.com/OpenClassrooms-Student-Center/ProjetOpenSource.git>

### Clonez le dépôt en local

* Allez dans git bash et cloner :

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

Cloning into 'ProjetOpenSource'...

remote: Enumerating objects: 7, done.

remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 7

Unpacking objects: 100% (7/7), 2.33 KiB | 170.00 KiB/s, done.

* Répertoire créer avec le code source

### Appréhendez le système de branches

* Pour connaître les branches présentes dans un projet :

Se placer dans le répertoire du dépôt distant copié sur notre pc et faire :

git branch

\* master

* Pour créer une branche :

git branch cagnotte

vérification :

git branch

cagnotte

\* master

Pour basculer sur la banche (\* master 🡺 dans la branche master) :

git checkout cagnotte

vérification :

git branch

\* cagnotte 🡺 bien sur la branche cagnotte

master

### Réalisez un commit

Ajouter le fichier modifié

git add login.css

Et commiter :

git commit -m "Ajout cagontte"

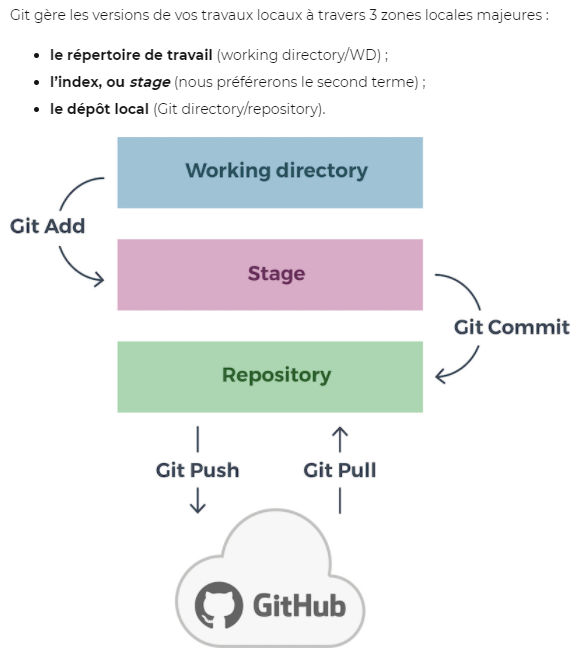
### Réalisez votre premier push

Pour envoyer les modifications que l'on a réalisées en local sur le dépôt à distance :

git push

Pour récupérer en local le projet distant :

git pull



* Pour créer une branche principale master : il faut au moins un commit

Ajouter un fichier dans le répertoire puis :

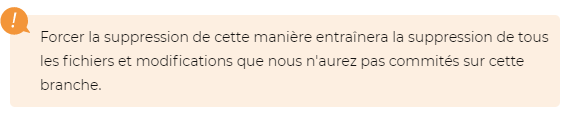
git add test1.txt

Commiter

git commit

puis ajouter le texte de commit et fermer le fichier de notepad++

* supprimer une branche que nous venons de créer



ajouter une branche : git branch brancheTest

puis la supprimer : git branch -d brancheTest

### ****J’ai modifié la branche principale master par erreur sans avoir commiter****

Se placer dans la branche master (git checkout master).

Ajouter un fichier dans le répertoire puis git add test1.txt

Pour voir ce qu’il faut commiter : git status

Commiter : git commit -m ’oulala’

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 :

git status

git stash

pour vérifier que la branche master est propre :

git status

On branch master

nothing to commit, working tree clean

Les commit sont mis de côté on peut les commiter sur une nouvelle branche

git branch brancheTest

git checkout brancheTest

git stash apply

Si plusieurs remises ont été faites :

* voir les remises : git stash list

stash@{0}: WIP on master: dfddb60 toto

stash@{1}: WIP on master: dfddb60 toto

* appliquer la bonne remise grâce à l’identifiant : git stash apply stash@{0}

### ****J’ai modifié la branche principale master par erreur et commité****

On récupère le commit de la branche master :

git log

commit 257746dad8f72a47c0d4a4127c070fa82c7fbc65 (HEAD -> master)

Author: loedata <loe.rabier@gmail.com>

Date: Mon Oct 26 09:29:11 2020 +0100

aie

commit dfddb60af34bc6aa5be2c53ded252dbe313d2ff4 (brancheTest)

Author: loedata <loe.rabier@gmail.com>

Date: Mon Oct 26 09:03:13 2020 +0100

toto

On a l’identifiant du commit : 257746dad8f72a47c0d4a4127c070fa82c7fbc65

Sur la branche master :

git reset --hard HEAD^

HEAD is now at dfddb60 toto

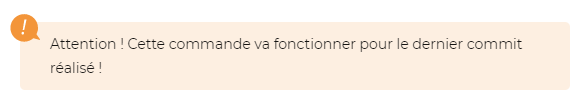
Puis on créé une nouvelle branche, on bascule sur la branche et on applique le commit sur cette branche :

git branch brancheCommit

git checkout brancheCommit

git reset –hard 257746dad8f72a47c0d4a4127c070fa82c7fbc65

### Je souhaite changer le message de mon commit



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

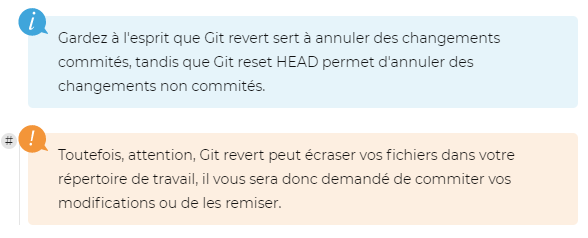
et on vérifie : git log

### J’ai oublié un fichier dans mon dernier commit

git add test2.txt –fichier oublié

git commit --amend --no-edit

### Corrigez vos erreurs en local et à distance



Il est possible d'annuler son commit public avec la commande **Git revert**. L'opération Revert annule un commit en créant un nouveau commit. C'est une méthode sûre pour annuler des changements, car elle ne risque pas de réécrire l'historique du commit.

Git revert HEAD^

### L'accès à distance ne fonctionne pas

Git base toute sa gestion d’authentification sur le mécanisme **des clés SSH**.

Génération des clés privées et publiques SSH

ssh-keygen -t rsa -b 4096 -C [loe.rabier@gmail.com](mailto:loe.rabier@gmail.com)

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/PC Maison/.ssh/id\_rsa):

Created directory '/c/Users/PC Maison/.ssh'. 🡺 entr

Enter passphrase (empty for no passphrase): 🡺 loe21

Enter same passphrase again:

Your identification has been saved in /c/Users/PC Maison/.ssh/id\_rsa

Your public key has been saved in /c/Users/PC Maison/.ssh/id\_rsa.pub

The key fingerprint is:

SHA256:sydBFTkucAscS2VTv1cnCVZpkxndJQkVv9hPBey7Ccs loe.rabier@gmail.com

The key's randomart image is:

+---[RSA 4096]----+

| .oo+.+o+=BO+|

| .+o.ooo .O=o|

| .+.o ..oo.=|

| .o . .+o+|

| S. ...+.|

| + .....|

| o .. o o.|

| o E o |

| |

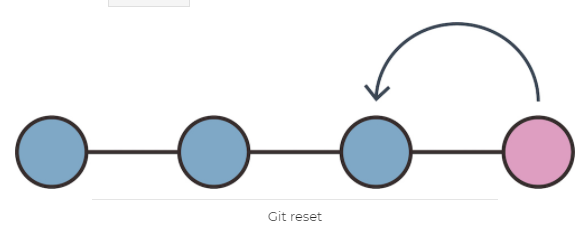
+----[SHA256]-----+

Pour la trouver, il suffit d'aller à l'adresse : C:\Users\VotreNomD'Utilisateur\, et d'afficher les dossiers masqués, dans le répertoire ssh.

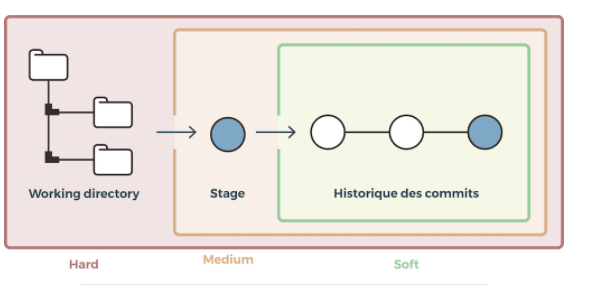
La clé id\_rsa.txt est votre **clé privée** alors que la clé id\_rsa.pub est votre**clé publique**.

Vous pouvez copier votre clé en l'ouvrant dans un bloc-notes.

### Utilisez Git reset



a commande  git reset  est un outil complexe et polyvalent pour **annuler les changements**. Elle peut être appelée de trois façons différentes, qui correspondent aux arguments de ligne de commande**--soft, --mixed et --hard**



Git reset notrecommitcible –hard

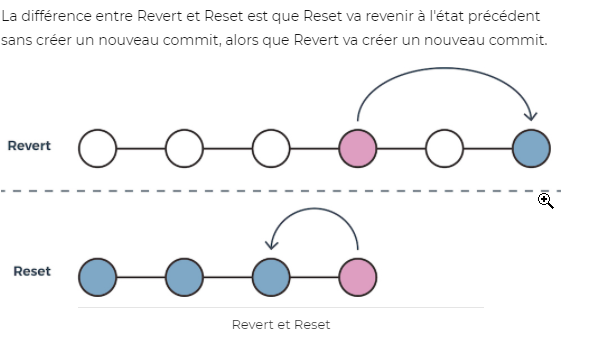
Le  git reset --mixed  va permettre de revenir juste après votre dernier commit ou le commit spécifié, sans supprimer vos modifications en cours. Il va par contre créer un HEAD détaché. Il permet aussi, dans le cas de fichiers indexés mais pas encore commités, de désindexer les fichiers.

Nous avons enfin le  git reset --Soft. Le **git reset --Soft** permet juste 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. Il ne supprime aucun fichier, aucun commit, et ne crée pas de HEAD détaché

### Oups, j'ai des conflits !

Ourvrir le fichier et résoudre les commits

### La machine à remonter le temps ! J’ai ajouté le mauvais fichier au commit



### Corriger un commit râté

Utiliser la journalisation.

**consulter l'historique de votre projet** et **énumère en ordre chronologique inversé**les commits réalisés **:**

git log

Git reflog va enregistrer vos commits, vos modifications de messages, vos merges, vos resets, enfin tout, quoi, va loguer les commits, mais aussi toutes les autres actions que vous avez pu faire en local.

Git reflog va afficher un identifiant SHA-1 pour chaque action. Il est donc très facile de revenir à une action donnée **grâce au SHA.**

Git reflog

Puis : git checkout avec l’identifiant SHA-1.

### Qui s’est amusé dans mon dépôt ? Git blame

La commande  git blame  permet **d’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.

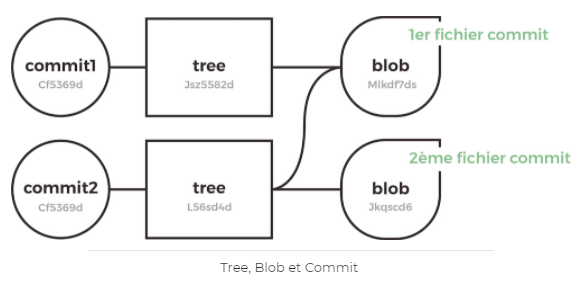
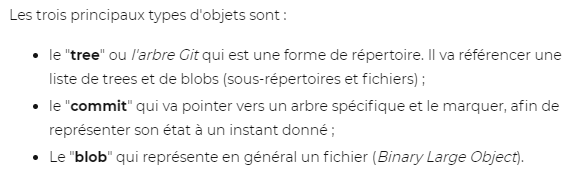
git blame monfichier.php

### Il me faut ce commit ! Vite Git cherry-pick

Admettons que vous travailliez sur votre branche "Mes évolutions", et que vous ayez déjà réalisé plusieurs commits. Votre collègue a besoin de l'une de ces évolutions pour la livrer au client, mais pas des autres. C'est dans ce cas bien précis que nous allons faire appel à  git cherry-pick  ! Cette commande va permettre de sélectionner un ou plusieurs commits grâce à leurs SHA (décidément ils sont partout) et de les migrer sur la branche master, sans pour autant fusionner toute la branche "Mes évolutions".

Git cherry-pick d356910 de91111 (8 premiers caractères de l’identifiant SHA-1)

### Découvrez l’arbre Git et sa structure



### Représentation cryptographique d'un commit

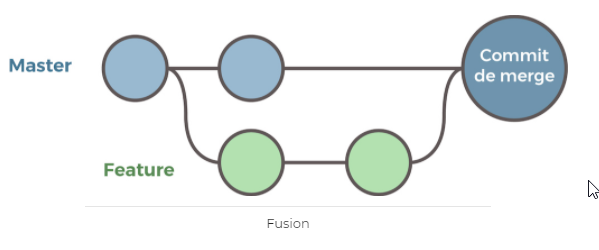
Pour chacun des objets dans Git, vous trouverez cette chaîne de 40 caractères que nous appelons le **hash SHA-1.**

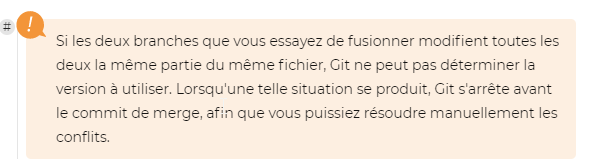
**Le commit étant un objet**, lui aussi a son empreinte SHA-1.

### Comment fonctionne la fusion sous Git ?

Git merge : permet de fusionner 2 branches

git merge nomBranche



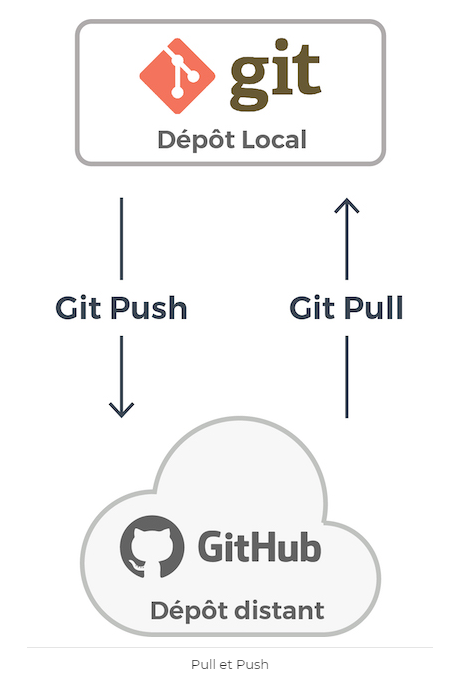


### Les options Git pull/Git push

La commande Git pull permet de télécharger les modifications qui ont eu lieu sur le dépôt distant, dans le but de les rapatrier sur le dépôt local.

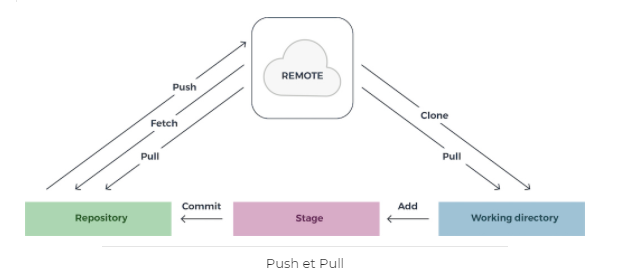
Git pull = git fetch puis git merge

la commande Git push permet d'envoyer des modifications que l'on a réalisées en local sur le dépôt à distance.



### À quoi sert Git fetch ?

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 isole le contenu récupéré en tant que contenu local existant, cela n'a absolument aucun effet sur votre travail de développement local. La commande Git fetch va récupérer toutes les données des commits effectués sur la branche courante qui n'existent pas encore dans votre version en local. **Ces données seront stockées dans le répertoire de travail local**, mais ne seront pas fusionnées avec votre branche locale.



### Comment fonctionne Git rebase ?

Au niveau du contenu, le rebase **consiste à changer la base de votre branche d'un commit vers un autre**, donnant l'illusion que vous avez créé votre branche à partir d'un commit différent. Git va prendre vos modifications d'une branche et les transposer sur une autre branche.

### Réécrivez l’historique avec l’interactif Rebase de Git

Exécuter  git rebase  avec l'option -i démarre **une session de rebasage interactive**. Cette fonctionnalité permet de déplacer les commits un à un en ayant la possibilité de les modifier. Vous avez donc la possibilité de supprimer certains commits ou de les modifier. Cette action ouvre un éditeur dans lequel vous pouvez entrer des commandes  pour chaque commit à rebaser.

### Modifiez l'ordre des commits

git rebase -i HEAD^

edit 54dfiosd

$ git rebase -i HEAD^

Stopped at 54dfiosd... updated the gemspec to hopefully work better

You can amend the commit now, with

git commit --amend

Once you’re satisfied with your changes, run

git rebase –continue

git commit –amend

$ git rebase –continue

## Utilisez des techniques de nettoyage de branche

### Comment utiliser Rebase pour nettoyer votre branche ?

# Commandes :

# p, pick = utilisez le commit

# r, reword = utilisez le commit, mais éditez le message de commit

# e, edit = utilisez le commit, mais arrêtez-vous pour apporter des changements

# s, squash = utilisez le commit, mais intégrez-le au commit précédent

# f, fixup = commande similaire à "squash", mais qui permet d'annuler le message de log de ce commit

# x, exec = exécutez la commande (le reste de la ligne) à l'aide de Shell

# d, drop = supprimez le commit

git rebase -i HEAD~2

drop 58gkbg56 commit52

drop 899hbg78 commit53

### Supprimez les branches non suivies

git branch -d brancheTest

### Squash ? Débarrassez-vous des commits à tout va !

Mais alors, comment l'utiliser ? il s'utilise comme le reste des commandes du rebase interactif. On va aller sélectionner tous les commits nous intéressant, et on va leur appliquer la commande Squash !

git rebase -i HEAD~3

On leur applique la commande Squash :

pick 57dcsd58 Création du formulaire

squash 58gkbg56 Design Formulaire

squash 899hbg78 Correction du formulaire

Concrètement, on dit à GIT de se baser sur le premier commit et on lui applique tous les suivants pour n’en faire qu’un seul.

Lorsque l’on valide le squash, Git va réappliquer les commits dans le même ordre dans lequel ils ont été configurés juste avant. On met alors un message de commit qui concerne le regroupement de nos commits.

### Trouvez l’origine d’un bug avec Git bisect

Le principe est plutôt simple. Vous indiquez à Git que **vous cherchez un bug**, **il se déplace de commit en commit,** vous testez la version et vous lui dites si le bug est présent dans le commit courant ou pas.

**Le but est de retrouver le premier commit où le bug est apparu.**

On commence par la commande start :

git bisect start [bad] [good]

Au lieu de *bad*, vous devrez mettre le hash d'un commit où le bug est présent. À la place de *good*, vous devrez mettre le hash d'un commit où le bug n'était pas présent !

Git va alors naviguer dans chacun des commits entre les deux, dans le but de trouver le premier commit où le bug est apparu.

Git va se déplacer sur chaque commit et vous allez devoir, pour chacun de ces commits, lui indiquer si le commit est **good** ou **bad**.  :)

Si le commit ne présente pas le bug :

git bisect good

Si le commit présente le bug :

git bisect bad

Une fois chaque commit vérifié, Git va vous indiquer le commit qui a provoqué le bug. Il va l'indiquer de cette manière :

fvsd54g5s5d4g5f34g5dfg47df578q9qdff6 is first bad commit

commit fvsd54g5s5d4g5f34g5dfg47df578q9qdff6

Author: Moi <Moi@example.com>

Date: Tue mar 27 16:28:38 2019 -0800

Add fonctionnality AB