GIT

INITIALISATION DE GIT

1. « **git init** » : dans le dossier.
2. « **git config --global user.name «name»** » : définit l'identité de l'utilisateur.
3. « **git config --global user.email «email»** » : définit l'adresse email de l'utilisateur.
4. « **git config --global branch.autosetuprebase always** » : pour les projets à plusieurs personnes dessus, cela permet d'éviter de devoir à faire un rebase à chaque pull avant de push.
5. « **ungit** » : pour lancer ungit.

COMMIT

1. « **git status** » : pour obtenir l'état du staging.
2. « **git add <nomDuFichier>** » : pour ajouter <nomDuFichier> au staging.
3. « **git add <*.extension>** » : pour ajouter tous les fichiers pourtant les extensions au staging.
4. « **git add .** » : pour ajouter toutes les modifications au staging sauf les suppressions.
5. « **git add --all** » : pour ajouter toutes les modifications au staging (même les suppressions).

MANIPULATION DE L'HISTORIQUE DES COMMITS

1. « **git commit --amend** » : intègre les nouvelles modifications au commit précédent (ouvre un éditeur, avec la possibilité de modifier le message du commit, quittable avec :wq).
2. « **git checkout** » : pour revenir en arrière en tant que spectateur.
3. « **git revert <commit>** » : pour défaire un commit avec un nouveau commit.
4. « **git reset <nomDuFichier>** » : à faire après un staging pour le défaire si fait par erreur.
5. « **git reset** » : pour défaire tout le staging.
6. « **git reset <commit>** » : permet de revenir en arrière au commit sélectionné, nettoie l'historique des commits mais laisse les modifs en staging.
7. « **git reset <commit> --hard** » : revient au commit sélectionné en supprimant l'historique mais aussi les modifs (DANGEREUX).
8. « **git rebase -i HEAD~<nombre>** : permet de manipuler/modifier les derniers commits sur une longueur de <nombre>.

BRANCHES

1. « **git branch** » : donne la liste des branches.
2. « **git branch -a** » : donne la liste de TOUTES les branches.
3. « **git branch <nomBranche>** » : créé une branche avec le nom nomBranche.
4. « **git checkout <nomBranche>** » : saute dans la branche nomBranche.
5. « **git branch <nomBranche>** » : supprime la branche nomBranche.
6. « **git merge <nomBranche>** » : fusionne la branche nomBranche avec la branche sur laquelle on est actuellement (il faut se placer sur master au préalable si on veut que la branche nomBranche soit fusionnée avec la branche Master et cela est faisable si un fast-forwards est possible. Si la branche Master a avancé en même temps, on aura un auto-merging et un nouveau commit de fusion sera créé lors de la fusion).
7. « **git merge --no-ff <nomBranche>** » : permet la fusion sans fast-forwards (en cas de conflit pendant la fusion, il faut consulter le fichier concerné, supprimer les commentaires, garder ce qu'on veut conserver et re-commit).
   - ⇨ Faire un « **git rebase -i master** » en se plaçant sur la branche qu'on veut déplacer au préalable permet de lancer une fusion intéractive pour modifier les commits (« **squash** » permet de mettre le commit dans le précédent).
   - ⇨ Ensuite, il faut refaire un **« git checkout <nomBranche>** » pour préparer le merge classique.
   - ⇨ Enfin, il faut faire un « **git merge <nomBranche>** » classique.
8. « **git branche -d <nomBranche>** » : supprime la branche nomBranche (a faire après fusion).

REMISAGE

1. « **git stash** » : pour tout stocker en mémoire (mettre en attente sans staging).
2. « **git stash -u** » : pour tout stocker en mémoire (même les fichiers untracked).
3. « **git stash save <description>** » : pour faire un stash avec un nom custom.
4. « **git stask list** » permet d'établir la liste des stash en mémoire.
5. « **git stash show stash@{<numero>}** » : permet de visualiser un stash en particulier.
6. « **git stash apply** » : permet d'appliquer les modifications mises en mémoire.
7. « **git stash apply stash@{<numero>}** » : applique un stash en particulier.
8. « **git stash drop** » : permet de supprimer les stashs en mémoire.
9. « **git stash pop** » : applique les stash et les efface de la mémoire.
10. « **git stash branch <nomBranche>** » : applique les stashs a la branche nomBranche nouvellement créée en même temps.

REMOTE

1. « **git remote -v** » : pour obtenir la liste des différents remotes.
2. « **git remote remove origjn** » : supprime du projet le dépôt distant nommé origin.
3. « **git remote add origin <adresse>** » : ajouter un dépôt au projet.
4. « **git remote rename origin <nouveauNom>** » : renomme le dépôt origin en nouveauNom.
5. « **git branch -r** » : affiche la liste des branches du dépôt.
6. « **git push origin master** » : publie la branche master sur le dépôt nommé origin.
7. « **git push origin --delete <nomBranche>** : supprime du dépôt nommé origin la branche nomBranche.
8. « **git pull origin master** » : récupère la branche master du dépôt origin dans le dossier ciblé.
9. « **git clone <chemin> <nomDossier>** » : pour une personne tierce : récupère et clone le contenu du chemin dans le nouveau dossier créé pour l'occasion qui s'appelle nomDossier.
10. « **git pull --rebase origin master** » : récupère le contenu du dépôt avant d'envoyer son propre contenu afin que tout soit fusionné proprement (il suffira d'un classique « **git push origin master** » ensuite pour que tous les commits soient ajoutés dans l'ordre et proprement).

FORK ET PULL REQUEST

1. « **ssh-keygen -t rsa -C «email»** » : pour générer une clé ssh (il faudra valider le dossier dans lequel sauvegarder la clé).
   - ⇨ **Id_rsa** sera la clé privée (à ne jamais partager).
   - ⇨ **Id_rsa.pub** sera la clé publique.
   - ⇨ Rentrer la clé publique dans les options de Github.

Pour faire un fork (clone d'un dépôt qui ne nous appartient pas) :

1. 1. Cliquer sur « fork » en haut du dépôt Github qui nous intéresse puis récupérer le lien en bas à droite.
2. « **git clone <adresseDepot>** » : clone le dépôt à l'adresseDepot sur notre installation locale.

POUR METTRE A JOUR UN FORK APRES UNE PERIODE D'ABSENCE (si l'auteur original avance sans nous) :

1. Se rendre sur le dépôt de l'auteur original (pas le fork).
2. Copier l'url de l'auteur.
3. « **git remote add upstream <url>** » :
4. Vérifier qu'il a bien été ajouté avec un « **git remote -v** ».
5. « **git fetch upstream** ».
6. « **git merge upstream/master** » : pour récupérer les modifs.

Pour faire une modification sur un fork :

1. Créer une branche avec « **git branch <nouvelleBranche-fix>.**
2. Se rendre sur la branche avec un « **git checkout <nouvelleBranche-fix>** ».
3. Faire la modification.
4. Commit la modification.
5. Envoyer la modification avec « **git push origin <nouvelleBranche-fix>** ».

Pour faire un pull request :

1. Sur son propre fork, cliquer sur le bouton vert « pull request ».
2. Tout remplir sur la modification du fork (faite avant) et envoyer à l'auteur.