**Commandes en Git**

Voc :

repository = dépôt

commit = version

stage = index répertoriant tous les éléments modifiés à envoyer lors de la prochaine sauvegarde

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Utilisation | |
| Commandes générales Linux |  |  | |  | |
| pwd | indique le répertoire courant | pwd | |  |  |
| ls | donne la liste des fichiers et dossiers dans le répertoire courant  sous forme de liste  en liste avec plus de détails | ls  ls -l  ls -l -a | |  |  |
| cat | affiche le contenu d'un fichier | cat monFichier.txt | |  |  |
| mkdir | créer un dossier | mkdir monDossier | | *créer un nouveau dossier et se placer à l'intérieur*  mkdir monDossier && cd $\_ |  |
| touch | créer un fichier | touch monFicier.txt | |  |  |
| cd | se placer dans le dossier indiqué  remonte l'arborescence de 2 dossiers  revient au répertoire principal | cd monDossier\monSousDossier  cd ..\..  cd ~ | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Initialisation de Git | **après avoir installé Git sur la machine, il faut configurer les éléments suivants :** | | |  |  |
| config --global | définir le nom ou pseudo et adresse mail qui serviront de ref dans les commits et à lier le compte GitHub | git config --global user.name 'Mon Nom'  git config --global user.email 'monmail@sfr.fr' | |  |  |
| --list | voir la liste des paramétres de configuration | git config --list | |  |  |
| --global branch.autosetuprebase | utiliser rebase par defaut lors de la recupération des branches  (git pull = git pull --rebase) | git config --global branch.autosetuprebase always | |  |  |
|  |  |  | |  |  |
| Commandes Git | **ATTENTION : utilisation de GitBash pour pouvoir utiliser les commandes version Linux** | | | |  |
| help | obtenir l'aide liée à la commande donnée en paramètre | git help maCommande | |  |  |
| init | initialise le repository dans le dossier dans lequel on est positionné | git init | |  |  |
| ls -l -a .git | passe en revue les éléments du dossier .git | ls -l -a .git | |  |  |
| cat | accède au contenu du fichier | cat monFichier.txt | |  |  |
|  |  |  | |  |  |
| Commit |  |  | |  |  |
| status | permet de connaitre l'état du repository (fichiers untracked, unmodified, modified, staged, …) |  | |  |  |
| diff | affiche les modifications ayant été faites dans chaque fichier (ou dans le fichier spécifié) depuis le dernier commit | git diff  git diff monFichier.txt | |  |  |
| add | ajoute le fichier ou le dossier au stage, cad aux éléments qui seront sauvegardés lors du prochain commit | git add monFichier.txt | |  |  |
| . | ajoute tous les fichiers modifiés et déjà suivis au stage  ATTENTION : ignore les nouveaux fichiers jamais commités | git add . | |  |  |
| \*.type | ajoute au stage tous les fichiers (modifiés ET nouveaux) du type indiqué | git add \*.html | |  |  |
| --all | ajoute tous les éléments au stage (fichiers modifiés ET nouveaux fichiers jamais commit)  ATTENTION : à utiliser avec précaution ! |  | |  |  |
| rm--catched | retire le fichier/dossier du stage | git rm --cached monFichier | |  |  |
|  |  |  | |  |  |
| .gitignore | fichier texte à créer à la racine du dossier de travail et à remplir avec les titres des fichiers à ignorer | mkdir .gitignore  *remplissage manuel du fichier*  git add .gitignore | *Contenu possible d'un fichier* **.gitignore** *:*  # un commentaire  *ignore*  \*.a  *fichiers de type a*  !lib.a *sauf pour le fichier lib.a*  /TODO  *le fichier TODO à la racine du projet*  build/  *tout le répert. build*  doc/\*.txt fichiers .txt directement dans le  dossier doc  doc/\*\*/\*.txt tous les .txt dans le dossier et sous-  dossier de doc | | ATTENTION : .gitignore doit faire partie des fichiers suivit dans le stage et commité afin de transmettre les infos qu'il contient ! |
|  |  |  |  | |  |
| reset | vide le Stage (mais conserve les modifications sans modifier les fichiers) | git reset | |  |  |
| HEAD  -- | supprime un fichier en particulier du satge | git reset HEAD monFichier.txt  git reset -- monFichier.txt | |  |  |
| --hard | vide le Stage ET supprime aussi toutes les modifications depuis le dernier commit sans possibilité de les rétablir | git reset --hard | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| commit -m | crée la version, cad sauvegarde toutes les fichiers contenus dans le stage et ayant étés modifiés | git commit -m ’Nom du commit' | |  |  |
| commit -a -m | crée le commit directement sans avoir à passer par le stage (cad git add …)  ATTENTION : inclus dans le commit uniquement les fichiers déjà suivis dans les précédents commits (pas les nouveaux) | git commit -a -m’Nom du commit' | |  |  |
|  |  |  | |  |  |
| log | affiche l'historique des commits, du plus récent au plus ancien | git log | |  |  |
| --oneline | affiche l'historique des commits plus lisiblement avec une seule ligne par commit | git log --oneline | |  |  |
| -p | montre les différences introduites entre chaque commit | git log -p | |  |  |
| -2 | limite le nombre de résultats au nombre indiqué en paramètre | git log -10 | |  |  |
|  |  |  | |  |  |
| show | accède aux détails et changements dans un commit à partir du début de sa hash | git show 2221cd93 | |  |  |
| cat-file -p | accède au contenu d'un fichier référencé dans le commit avec sa hash de 40 ca. | git cat-file -p 25adf65… | |  |  |
| blame | accède aux auteurs qui ont modifié chaque ligne d'un fichier particulier et au début de la Hash du commit qui a validé la ligne | git blame monFichier.txt | |  |  |
|  |  |  | |  |  |
| tag | affiche la liste des tags | git tag | |  |  |
| tag -a -m | place un tag (étiquette) sur le commit en cours | git tag -a MonTag -m ’message associé au tag' | |  |  |
|  |  |  | |  |  |
| commit -a --amend -m | modifie le dernier commit | git commit -a --amend -m’Nouveau nom du commit' | |  | ATTENTION : à éviter car il y a modification de l'historique, le commit précédent étant détruit pour être remplacé par le nouveau ! |
|  | change l'auteur du commit | git commit --amend --no-edit --reset-author | |  |  |
| checkout | restaure uniquement le fichier concerné tel qu'il est dans le commit indiqué | git checkout e32bb29 monFichier.txt | |  | Rq: le contenu du fichier est alors automatiquement chargé dans le Stage comme une simple modification actuelle et sera sauvegardé lors du prochain commit |
| revert e32bb29 | créé un nouveau commit qui annule les modifications du commit indiqué | git revert e32bb29 | |  | ATTENTION : ne revient pas en arrière en rétablissant tout le projet au commit indiqué, annule seulement les modifications prises en compte dans le commit indiqué ! |
| reset | supprime tous les commits depuis le plus récent jusqu'à celui indiqué, mais conserve les modifications sans modifier les fichiers (pouvant alors être ajouté au Stage) | git reset e32bb29  git reset e32bb29 --mixed | |  | ATTENTION : utiliser avec précaution car modification de l'historique ! |
| --mixed | en conservant les modifications sans modifier les fichiers (pouvant alors être ajouté au Stage) |  | |  |  |
| --soft | idem et en laissant les modifs dans le Stage | git reset e32bb29 --soft | |  |  |
| --hard | en supprimant également toutes modifications dans les fichiers (revient exactement à l'état du commit indiqué sans possibilité de retour en arrière !) | git reset e32bb29 --hard | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Stash | zone mémoire permettant de sauvegarder temporairement les modifications en cours. Utile pour :  - retrouver l'état du dernier commit et faire une petite modification urgente avant de reprendre le travail en cours  - pouvoir changer de branche et faire une modification urgente sans perdre le travail en cours  - si on s'est trompé de branche : sauvegarder le travail, changer de branche puis appliquer les modifs à le bonne branche  - ... | | | | |
| stash | ajoute les modifications en cours dans le Stash | git stash  git stash save 'Mon commentaire' | |  | ATTENTION : exclu les nouveaux fichiers |
| -u | ajoute les modifications en cours dans le Stash en incluant les nouveaux fichiers (normalement exclus par défaut) |  | |  |  |
| stash list | affiche la liste de tous les stash en mémoire | git stash list | |  |  |
| stash show stash{2} | affiche les modifications sauvegardées dans le Stash numéro 3 (index = 2) | git stash show stash{2} | |  |  |
| -p | idem avec détail du code modifié | git stash show stash{2} -p | |  |  |
| stash apply | restaure toutes les modifications sauvegardées (ou seulement celle tu stash indiqué) | git stash apply  git stash apply stash{2} | |  |  |
| stash drop | supprime les Stash de la sauvegarde (ou seulement celui indiqué) | git stash drop  git stash drop stash{2} | |  |  |
| stash pop | = apply + drop : restaure les modifications et vide la sauvegarde en même temps | git stash pop  git stash pop stash{2} | |  |  |
| stash branch | créé une nouvelle branche et y applique les modifications sauvegardées | git stash branch maNouvelleBranche | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Branches |  |  | |  |  |
| checkout | changer de commit ou de branche (se place sur le dernier commit de la branche) | git checkout 25dhfz652dfkhdf…  git checkout maBranche | | git checkout master  git chechout HEAD^^  git checkout HEAD~2 | => retourne sur le dernier commit de master  => remonte de deux commits parents  => remonte de deux commits parents |
| branch | afficher toutes les branches du repository | git branch | |  |  |
| branch | créer une nouvelle branche | git branch maBranche | |  |  |
| checkout -b | créer une nouvelle branche et se positionner dessus en une seule instruction | git checkout -b maNouvelleBranche | |  |  |
| branch -d | supprimer la branche | git branch -d maBrancheASupprimer | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Fusionner les branches |  |  | |  |  |
| merge | fusionner la branche indiquée en paramètre sur la branche actuelle | git merge maBrancheAFusionner | |  |  |
|  | en cas de conflit, les résoudre manuellement dans les fichiers, puis : | git add monFichierConflictuel  git commit -a | | *Sinon, pour annuler la fusion :*  git merge --abort |  |
|  | ou utiliser une interface graphique pour résoudre le conflit avec | git mergetool monInterfaceGraphique | |  |  |
| rebase | déplace tous les commits de la branche actuelle sur le dernier commit de la branche indiquée en paramètre | git rebase master | |  |  |
|  | en cas de conflit, les résoudre manuellement un par un, puis : | git add monFichierConflictuel  git rebase --continue | | *Sinon, pour annuler la fusion :*  git rebase --abort |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Clé SSH |  |  | |  |  |
| ssh-keygen | génère une clé SSH à utiliser pour l'identification entre la machine et le compte GitHub | *Dans de dossier* Utilisateurs/Moi/.ssh  $ ssh-keygen -t rsa -b 4096 -C 'monadresse@sfr.fr' | |  | une fois générée, copier le contenu de id\_rsa.pub sur GitHub > settings > SSH  rien d'autre à faire sur la machine locale |
|  |  |  | |  |  |
| Remote repository | dépos à distance (via Github par exemple) |  | |  |  |
| clone | télécharge le repository distant complet sur la machine | git clone adresse/duRepository.git | |  |  |
| --depth | permet de définir le nombre de commit de l'historique à récupérer | git clone adresseDuRepo nonDuDossierCible --depth 1 | |  | Rq: inutile de récupérer la totalité de l'historique depuis le début du projet, seul le dernier commit est nécessaire |
| remote | affiche la liste des serveurs à distance | git remote | |  |  |
| remote add | sélectionne le repository distant via son adresse | git remote add origin git@github.com:nom/adresse.git | | git remote add monRemoteRepository git@github.com:nom/adresse.git | Rq: origin est le nom conventionnellement donné pour le repository remote principal. |
| remote show | récupère les données du serveur distant indiqué en paramètre | git remote show origin | |  |  |
| push -u origin | pousse le contenu validé (le/les commit(s) validés sur la machines mais absents sur le serveur) vers le repository distant  pour la branche en question  toutes les branches suivies (cad ayant été déclarées avec l'option -u) | git push -u origin maBranche  git push | |  | Rq: -u est un raccourcit de --set-upstream qui permet de mettre en place le suivit entre le repository local et distant pour la branche en question. |
| fetch | affiche les différences entre les commits sur le serveur et ceux sur la machine | git fetch origin | |  |  |
| git pull | télécharge sur la machine les commits récents de  la branche en question  toutes les branches suivies (cad ayant été déclarées avec l'option -u) | git pull origin master  git pull | |  |  |
| --rebase | récupère les derniers commits de la branche origine en faisant un rebase (et non pas un merge par defaut)  => historique des commits plus lisible | git pull --rebase origin master | |  | Rq: pour définir un pull en mode rebase par défaut sans avoir à préciser --rebase à chaque fois, voir config --global branch.autosetuprebase |
| checkout -b pull-sample | télécharge une branche sur le serveur qui n'existe pas encore en local | git checkout -b pull-sample origin/maBrancheRemote | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |
|  |  |  | |  |  |

PULL-REQUEST

Pour contribuer sur un projet open source :

1. Trouver le projet sur GitHub
2. Lire le fichier README pour connaitre les modalités de contributions à la section "Contributing" ou Pull-Request"
3. Fork le projet sur notre propre compte GitHub
4. Cloner le fork à partir de notre compte sur notre machine (git clone …)
5. (Il est possible de synchroniser la branche master de notre repository local avec, non pas le fork, mais la version officielle du repo pour prendre en compte toutes les récentes modifs

pour cela

* 1. on définit un nouveau remote : git remote add upstream https://github.com/...
  2. on se place sur la branche master
  3. on la synchronise : git branch --set-upstream-to=upstream/master )

1. Créer une nouvelle branche et travailler sur le code
2. Push la nouvelle branche sur notre fork
3. Créer une Pull-Request à partir de GitHub pour soumettre les modifications au projet officiel