GIT - LES LIGNES DE COMMANDES PRINCIPALES

`configuration`

RAPPEL

l'aide en ligne de commande.

```
git help config
git help push
git help pull
git help branch
```

CONFIGURATION

```
# Identity Name
git config --global user.name "john"

# Identity
Email git config --global user.email "john@exemple.fr"

# Editor Tool
git config --global core.editor subl

# Diff Tool
git config --global merge.tool filemerge
```

Liste des globales

```
git config --list
```

`Principales commandes`

STATUT DES FICHIERS

```
git status
```

LISTER LES BRANCHES

```
git branch -a
```

`*` sur la branche courante

CREATION D'UNE BRANCHE

```
# Deux lignes: créer et basculer sur la nouvelle branche (branch)
git branch nom_de_ma_branche_nouvelle
git checkout nom_de_ma_branche_nouvelle

# Une seule ligne: créer et basculer
git checkout -b nom_de_ma_branche_nouvelle
```

CHANGEMENT DE BRANCHE

```
git checkout nom_de_ma_branche

# GIT --version 2.23
git switch nom_de_ma_branche
```

SUPPRESSION D'UNE BRANCHE

```
# Si la branche est local et n'est pas créée sur le repo distant
git branch -d nom_de_ma_branch_local

# Si la branche est présente sur le repo distant
git push origin --delete nom_de_ma_branch_distante
```

PREMIER COMMIT

```
git add .
git commit -m "initial commit"
```

COMMIT SUIVANT

```
git add chemin_vers_mon_fichier
git commit -m "message du commit"
```

ANNULATION DU DERNIER COMMIT ET MODIFICATIONS

```
git reset --hard md5_commit
git push --force
```

ANTIDATÉ UN COMMIT

```
git add .
GIT_AUTHOR_DATE="2021-10-12 08:32 +100" git commit -m "Commit antidaté"
```

MIS À JOUR DU DÉPÔT LOCAL

```
git pull
```

MIS À JOUR DU DÉPÔT LOCAL D'UNE BRANCHE SPÉCIFIQUE

```
git pull origin MA_BRANCHE
```

ENVOYER SES COMMITS VERS LE DÉPOT DISTANT

```
git push
```

ENVOYER SES COMMITS VERS LE DÉPOT DISTANT SUR UNE BRANCHE SPÉCIFIQUE

```
git push origin MA_BRANCHE
```

SUPPRESSION D'UN FICHIER DU RÉPERTOIRE DE TRAVAIL ET DE L'INDEX

```
git rm nom_du_fichier
```

SUPPRESSION D'UN FICHIER DE L'INDEX

```
git rmg --cached nom_du_fichier
```

`Annuler commit`

ANNULER COMMITS (SOFT)

Seul le commit est retiré de Git ; vos fichiers, eux, restent modifiés. Vous pouvez alors à nouveau changer vos fichiers si besoin est et refaire un commit.

ANNULER LE DERNIER COMMIT

```
git reset HEAD^
```

Pour indiquer à quel commit on souhaite revenir, il existe plusieurs notations :

HEAD	Dernier commit
HEAD^	Avant-dernier commit
HEAD^^	Avant-avant-dernier commit
HEAD~2	Avant-avant-dernier commit (notation équivalente)
d6d545153868578a7f38dea79833b56d0326fcba1	Indique un numéro de commit
d6d9892	Indique un numéro de commit version courte

ANNULER COMMITS (HARD)

Si vous voulez annuler votre dernier commit et les changements effectués dans les fichiers, il faut faire un reset hard. *Cela annulera sans confirmation tout votre travail !*

ANNULER LES COMMITS ET PERDRE TOUS LES CHANGEMENTS

```
git reset --hard HEAD^
```

ANNULER LES MODIFICATIONS D'UN FICHIER AVANT UN COMMIT

Si vous avez modifié plusieurs fichiers mais que vous n'avez pas encore envoyé le commit et que vous voulez restaurer un fichier tel qu'il était au dernier commit :

```
git checkout nom_du_fichier

# GIT --version 2.23
git restore nom_de_ma_branch
```

ANNULER/SUPPRIMER UN FICHIER AVANT UN COMMIT

Supposer que vous venez d'ajouter un fichier à Git avec `git add` et que vous vous apprêtez à le « commiter ». Cependant, vous vous rendez compte que ce fichier est une mauvaise idée et vous voulez annuler votre `git add`.

Il est possible de retirer un fichier qui avait été ajouté pour être « commité » en procédant comme suit :

```
git reset HEAD -- nom_du_fichier_a_supprimer
```

`Différence`

```
# Affiche la différence entre le contenu du dernier commit
# et celui du répertoire de travail.
# Cela correspond à ce qui serait commité par git commit -a.
git diff HEAD

# Affiche la différence entre le contenu pointé par A et celui pointé par B.
git diff A B

# Diff entre un dossier présent sur deux branches
git diff master..MA_BRANCH chemin/vers/mon_dossier
```

`Modifier l'historique`

MODIFIER LE MESSAGE DU DERNIER COMMIT

```
git commit --amend
```

La commande ci-dessus charge le dernier message dans l'éditeur,

- Modifier message ;
- Enregistrer et quitter l'éditeur afin de prendre les modifications en compte.

AJOUTER DES FICHIERS AU DERNIER COMMIT

```
git add mon_fichier
git commit --amend
```

ÉDITER L'HISTORIQUE AVEC REBASE

```
# Historique des trois derniers commits
git rebase -i HEAD~3
```

Ce qui va afficher ceci dans votre éditeur de texte.

```
pick 6cbdbe2 Message du commit 3
pick 0a75a2d Message du commit 2
pick da74a4e Message du commit 1

# Rebase b8d6fe1..da74a4e onto b8d6fe1 (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Chaque commit est précédé du mot `pick`, utiliser tous les mots-clés possibles.

- Pour éditer, un commit changer `pick` par `edit` ;
- Enregistrer et quitter l'éditeur afin de prendre les modifications en compte ;
- Et suivre les instructions.

Pour valider, les changements utiliser la commande :

```
git rebase NOM_DE_MA_BRANCHE
```

Dans le cas, ou les commits modifier sont déjà présent sur la branche distante, il faudra "forcer"

```
git push --force origin NOM_DE_MA_BRANCH
```

`git-logs - Afficher les journaux de validation`

```
# Classique
git log

# Affiche X derniers commits
git log -n X

# Affiche les commits concernant un dossier
git log --oneline -- chemin/vers/mon_dossier

# Affiche un ensemble de commits par date
git log --since=date --until=date

# Représentation de l'historique à partir de HEAD (commit / branch)
git log --oneline --graph --decorate

# Représentation de l'historique à partir d'un fichier (commit / branch)
git log --oneline --graph --decorate nom_du_fichier
```

`Créer une release`

1. Créer une nouvelle branche `release/aaaammyy_xx`

   ```
   git checkout -b release/20140707_01 dev
   ```

2. Modifier le numéro de version dans le fichier racine `READ.md`

   ```
   git commit -a -m "Bumped version 20140707_01"
   ```

3. Se mettre sur la branche `dev` et :

   ```
   git merge release/20140707_01
   ```

4. Pour finaliser la release, merger les corrections éventuelles sur la `master` et la `dev`

```
git checkout master && git merge --no-ff release/20140707_01 && git push origin
master

git checkout dev && git merge --no-ff release/20140707_01 && git branch -d
release/20140707_01 && git push origin dev
```

5. Marquage de la version sur la branche master

```
git tag -a 20140707_01 -m "New prod 20140707_01" master && git push origin master
--tags
```

`git-tag`

RETOURNE LA LISTE DES TAGS DISPONIBLE

```
git tag

# ou
git tag -l
git tag --list
```

CRÉER UN TAG

```
# `-m` permet d'ajouter un message associé au tag
git tag -a v1.0 -m "Mon commentaire pour la version 1.0"

# Ajouter un tag à partir d'un commit anterieur
git tag -a v1.0 -m "Mon commentaire pour la version 1.0" md5_commit
```

VÉRIFIER UN TAG EN AFFICHANT LES INFORMATIONS

```
git tag -v nom_de_l_étiquette
```

POUSSER LES TAGS SUR LA BRANCHE DISTANTE

```
git push origin nom_de_l_étiquette

# Pousser plusieurs tags sur la branch
git push origin --tags
```

Installation d'un projet

INITIALISER UN PROJET

```
cd chemin/vers/mon_dossier
echo "# MON_PROJET" >> README.md
git init
git add README.md
git commit -m "Initial commit"
git remote add origin ....git
git push -u origin master
```

RÉCUPÉRER LE REPO SUR GITHUB

```
git clone ....git chemin/vers/mon_dossier
```

Mon repo est composé d'au moins deux branches.

develop : dédié au développement et résolution de bug. master : reflète le code en production. Personne ne doit travailler directement sur cette branche.

Pour récupérer votre branche develop

```
git branch -a
git checkout origin/develop
git checkout -b develop origin/develop
git branch
```

DEVELOPPEMENT: Branche develop

PROD: Branche master

```
# On se met sur la branche master
git checkout master

# On merge la branche develop
git merge develop

# Pour pousser les changements
git push origin master

# Penser à revenir sur develop
git checkout develop
```