Git Cheatsheet

Prérequis

Pour comprendre comment marche git, comment l'installer sur vos différentes machines (Windows, MAC OS, GNU/Linux) et comment l'utiliser, je vous redirige vers l'excellent livre Pro Git [1] (*a minima* les 3 premiers chapitres), qui fait tout ça très bien. Ici, ce sera juste un rappel des principales commandes.

Configuration et recherche d'informations

git config

Gère la configuration des options du système et des répertoires. Le couteau suisse.

```
# Récupère la valeur de toutes les variables définies
git config --list

# Choisit le fichier de configuration à utiliser
# local : .git/config
# global : /.gitconfig
# system : /etc/gitconfig
# file : fichier à fournir
git config --[local|global|system|file <file>]

# Définir son identité
git config --global user.name "Linus Torvalds"
git config --global user.email "linux@torvalds.com"

# Récupère la valeur d'une clé
git config user.name

# Enlever une clé
git config --unset user.name

# Changement d'éditeur de texte par défaut pour les messages de commit
git config --global core.editor vim
git config --global core.editor nano
git config --global core.editor "'C:/Program
    ↪ Files/Notepad++/notepad++.exe' -multiInst -nosession"

# Définir des alias/sous-commandes
git config --global alias.co checkout
git co # Exécute 'git checkout'
git config --global alias.br branch

# Quelques alias un peu plus utiles
# Voir le dernier commit
git config --global alias.last 'log -1 HEAD'

# Voir un joli graphe de commits
git config --global alias.loggy "log --graph
    ↪ --pretty=format:'\%Cred\%h\%Creset -\%C(yellow)\%d\%Creset
    ↪ \%s \%Cgreen(\%cr) \%C(bold blue)<\%an>\%Creset'
    ↪ --abbrev-commit --all"

# Enlève un fichier de la zone de staging (ne modifie pas le fichier)
git config --global alias.unstage 'reset HEAD --'

# Annule le dernier commit (seulement en local!)
git config --global alias.undo 'reset --soft HEAD^'

# Reformule le dernier commit
git config --global alias.amend 'commit --amend'

# Activer le cache des identifiants HTTP et les garder 1h.
git config --global credential.helper 'store --file ~/.credentials
    ↪ --timeout 3600'
```

git diff

Affiche les changements entre deux index.

```
# Affiche les différences fichier par fichier entre les fichiers modifiés et ceux de la zone de
    staging.
git diff

# Affiche les différences entre la zone de staging et le dernier commit
git diff --staged

# Afficher uniquement les noms des fichiers modifiés (pratique pour les gros commits)
git diff --name-only

# Affiche le diff d'un fichier spécifique
git diff <file>

# Différences entre deux branches (pas forcément locale)
git diff master feature\ branch
git diff master origin/master

# Différences entre deux commit
git diff fe2b292 8257efb
git diff fe2b292 8257efb <file>
```

git status

Récupère des informations sur l'état du répertoire.

```
# Affiche tous les fichiers modifiés et les fichiers suivis du répertoire
git status

# Même affichage, mais simplifié
git status --short

# Affiche aussi les fichiers ignorés
git status --ignored
```

Jouer avec les branches

git branch

Manipule les différentes branches

```
# Affiche les branches connues, ainsi que la branche active
git branch --list
git branch # Équivalent

# Affiche le dernier commit sur chaque branche
git branch -v

# Affiche les branches déjà fusionnées avec la branche courante
git branch --merged

# Créé une nouvelle branche nommée 'feature'
git branch feature

# Supprime une branche qui a déjà été fusionnée
git branch -d merged

# Force la suppression d'une branche, même non fusionnée (ATTENTION : le travail est
    définitivement perdu)
git branch -D notmerged

# Renomme une branche
git branch -m feature_1 # Renomme la branche courante en 'feature_1'
git branch -m feture feature # Renomme la branche 'feture' en 'feature'

# Change la branche amont(upstream) de la branche courante
git branch -u origin/master
```

git checkout

Déplacement entre les branches.

```
# Bouge le pointeur de tête sur une autre branche. Si la branche n'existe pas, git essaye de
    récupérer origin/¡nom de la branche¿ et de la créer localement.
git checkout master

# On peut aussi aller voir l'état du répertoire à un commit ou un tag donné
git checkout v1.4
git checkout f29d518

# Créé une nouvelle branche et se déplace dessus
git checkout -b new_branch

# Se déplace sur la nouvelle branche ft, configurée pour suivre origin/feature
git checkout -b ft origin/feature

# Après modification d'un fichier, il est possible de le réinitialiser à son état initial (tel qu'il
    est dans la zone de staging, ou tel qu'il est sur le pointeur de tête)
git checkout README
git checkout -- README # Si il y a une branche nommée 'README'
```

git fetch

Télécharge le contenu depuis un répertoire distant (ne l'intègre pas à l'espace de travail).

```
# Récupère le contenu du répertoire distant donné (par défaut origin)
git fetch
git fetch origin
```

Travailler et le partager

git add

Permet d'ajouter des fichiers à suivre et de les inclure dans la zone de staging.

```
# - Ajoute un fichier
# - Suite à un conflit de fusion, git add permet de marquer le conflit comme résolu pour ce
    fichier
git add __init__.py

# Ajoute tous les fichiers qui finissent en .go
git add *\.go

# Ajoute tous les fichiers (supprimés, modifiés, non suivis)
git add -A

# Force l'ajout d'un fichier, quand il a été ignoré par exemple
git add -f <file>
```

La commande add fonctionne en prenant une image d'un fichier pour l'ajouter à la zone de staging. Si une modification ultérieure a lieu, il faut ré-ajouter le fichier, sans quoi les dernières modifications ne seront pas prises en compte.

```
# Permet de choisir quelles modifications seront ajoutées. Cette commande est
    particulièrement pratique lorsque beaucoup de modifications ont été faites sur le même
    fichier : il est toujours possible de les séparer.
git add -e <file>

# Effectue l'ajout en mode interactif pour choisir les fichiers à ajouter ou non
# Se référer au manuel pour l'usage complet
git add -i <file>
```

git commit

Un petit peu le cœur de git, le commit prend le contenu de la zone de staging et génère une nouvelle entrée dans l'historique -¿ le commit. Il met aussi à jour le pointeur HEAD.

```
# Commit basique, ouvre l'éditeur configuré pour entrer le message du commit
git commit

# Commit avec le message donné
git commit -m "message here"

# Commit tous les fichiers déjà suivis et modifiés sans se préocupper de la zone de staging.
git commit -a

# Ajoute le contenu de la zone de staging au dernier commit et permet de le reformuler
git commit --amend

# Signer cryptographiquement le commit
git commit -S
```

git merge

Fusionne deux branches ensemble

```
# Fusionne la branche feature avec la branche courante
git merge feature


# Réalise une fusion sans avance rapide -¿ créé un commit de fusion même si il pouvait être
    évité
git merge --no-ff feature


# Si la fusion se passe mal, permet de l'annuler
git merge --abort
```

git pull

Récupère le travail amont et tente de l'intégrer au répertoire courant.

```
# - Si la branche amont est en avance sur la votre, pas de problème.
# - Si la branche amont et la votre ont divergé, le pull va échouer.
git pull
# En vrai, c'est équivalent à ce qui va suivre :
git fetch
git merge

# Tente de récupérer le travail amont et de rejouer vos commits par dessus
git pull --rebase
# En vrai, c'est équivalent à ce qui va suivre
git fetch
git rebase
```

Bref git pull est bien utile lorsque l'historique est assez simple, que les conflits sont raisonnables.

git push

Met à jour les répertoires distants.

```
# Comportement par défaut, push les commits en avance sur la branche amont. Dans la
    configuration par défaut, il s'agit de origin/¡branche courante¿
git push

# Push sur le répertoire et la branche spécifiée
git push origin master
git push <repertoire> <branche>


# Lorsque la branche amont et la votre ont divergé, réécrit l'historique des commits distants
    avec le votre (ATTENTION : Ceci est un des seul moyens de perdre effectivement des
    modifications avec git. En cas de doute, ne pas utiliser)
git push --force

# Après avoir supprimé une branche localement, supprime la branche sur le répertoire distant
git push origin --delete deleted

# Après avoir créé un ou plusieurs tags, les push sur la branche amont.
git push --tags.
```

git rebase

Le rebasage rejoue un ensemble de commits à partir d'un autre point de départ. Se référer à la documentation pour des exemples complets.

```
# Rejoue les commits de la branche courante sur la branche master
git rebase master

# Rejoue les commits de la branche feature sur la branche master
git rebase master feature

# Rejoue les commits de la branche client depuis l'ancêtre commun avec la branche server sur
    master. Cf schéma.
git rebase --onto master server client
```

```
Exemple 1:

C1 -- C2 -- C3 <- master
 \
  C4 -- C5 -- C6 <- server
   \
    C7 -- C8 <- client

git rebase --onto master server client

          master
            |
            v
C1 -- C2 -- C3 -- C7' -- C8' <- client
 \
  C4 -- C5 -- C6 <- server


Exemple 2:

C1 -- C2 -- C3 <- master
              \
               C4 -- C5 -- C6 <- server
                             \
                              C7 -- C8 <- client

git rebase --onto master server client

C1 -- C2 -- C3 <- master
            | \
            |  C4 -- C5 -- C6 <- server
             \
              C7' -- C8' <- client
```

```
# Lorsqu'il y a un conflit de rebasage, possibilité de continuer, aborter ou sauter la patch
git rebase --continue
git rebase --abort
git rebase --skip

# Démarre le mode interactif. Permet de réecrire l'historique des commits pour le rendre plus
    propre (Ne change pas les fichiers).
```

```
git rebase -i HEAD~3 # Réécrit les trois derniers commits (HEAD, HEAD-1, HEAD-2)
git rebase -i master # Réécrit les commits jusqu'à l'ancêtre commun entre la branche courante et master.
```

Note : Un rebasage réécrit l'historique, alors qu'un merge ajoute un commit à l'historique. De manière générale, il faut éviter de rebaser des branches déjà publiques (d'autant que cela implique un force push).

git tag

Un tag est une référence vers un commit. Il est plus simple de se souvenir d'un tag que d'un hash de commit.

```
# Créé un nouveau tag sur le commit pointé par la tête
git tag v1.0

#Créé un nouveau tag sur un commit spécifique
git tag v1.0 7872c2e

# Liste tous les tags
git tag
git tag -l

# Créé un tag annoté. Le tag contiendra le nom de l'auteur, son adresse mail et un message
git tag -a v1.0

# Créé un tag annoté avec le message 'message'
git tag -a v1.0 -m 'message'

# Signe cryptographiquement un tag annoté
git tag -a -s v1.0
```

Références

[1] Scott Chacon. https ://git-scm.com/book/en/v2, November 2014.