Git

Ressources

- Documentation en français
- GIT Howto
- 19 astuces GIT
- Comment écrire de bon message de commit
- Doc Markdown GitLab
- Comment nommer ses branches

Ignorer tout le contenu d'un dossier

Créer un fichier *.gitignore* dans le dossier. Il devra contenir :

```
# Ignore everything in this directory
*
# Except this file
!.gitignore
```

Configurer Git globalement

Le fichier `~/.gitconfig` contient les paramétrages de tous les projets Git de l'utilisateur sur l'ordinateur. Pour le modifier :

- **git config –global user.name “Prénom NOM”** : Stocker son nom
- **git config –global user.email “moi@mon-domaine.org”** : Stocker son email
- **git config –global credential.helper cache** : Mettre en cache le login et mot de passe
- **git config –global credential.helper 'cache –timeout=86400'** : Stocker le login et mot de passe pour 24h
- **git config –global color.ui true** : activer la couleur.

Le fichier .gitconfig

Le fichier **.gitconfig** est présent dans son dossier /home/moi :

```
[user]
    name = Prénom NOM
    email = moi@mon-domain.org
[credential]
    helper = cache --timeout=86400
[color]
    ui = true
[push]
    default = simple
```

```
[alias]
    co = checkout
    ci = commit
    st = status
    br = branch
    lg = log --color --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-
commit --
    type = cat-file -t
    dump = cat-file -p
```

Allias pour .bashrc

```
alias gs='git status '
alias ga='git add '
alias gb='git branch '
alias gc='git commit'
alias gd='git diff'
alias go='git checkout '
alias gk='gitk --all&'
alias gx='gitx --all'
```

Envoyer un dépôt Git dans Github

- Créer un dépôt nommé “origin” pointant vers le dépôt GitHub : **git remote add origin https://github.com/<mon-compte>/<mon-projet>.git**
- Récupérer la branche distante pour synchroniser localement : **git pull origin master**
- Envoyer la branche master local vers le dépôt origin : **git push origin master**

Initialiser un nouveau dépôt

- Créer un dossier : `mkdir mon-projet`
- Aller dans le dossier du projet : `cd mon-projet`
- Initialiser le dépôt git du projet : `git init`

Récupérer en local un dépôt github distant

git clone https://github.com/<mon-compte>/<mon-projet>.git

Commandes de base : Workflow normal

git clone git://<depot> : récupérer en local un dépôt distant. Créer un dossier avec le nom présent avant le .git final

git pull : mettre à jour le dépôt local avec les modifications distantes

git status : savoir ou on en est

git diff –cached : vérifier ce qui va être validé.

git add fichier1 fichier2 … : ajouter des fichiers nouveaux ou nouvellement modifiés aux changements à prendre en compte

git commit -m “Commit message” : valider les changements à prendre en compte

git commit -a -m “Commit message” : ajouter et valider directement tous les fichiers (sauf les nouveaux)

git log : lister les commits effectués

git diff <commit1> <commit2> : voir les différences entre deux commits

git rm <nom_fichier> : supprimer un fichier de l'ordinateur et du dépôt git.

git mv <nom_fichier> <nouvelle_destination> : déplacer un fichier sur l'ordinateur et du dépôt git.

git push origin master : envoyer dans la branche “origin” (= le dépôt distant) les modfs de la branche “master” locale“.

Usage basique des branches et des merges

git branch <nom-branche> : créer une nouvelle branche ”<nom-branche>“

git branch : voir toutes les branches existantes (* = branche actuellement utilisée)

git checkout <nom-branche> : changer de branche. Ici bascule sur la branche ”<nom-branche>“

git merge <nom-branche> : fusionner les changements de la branche ”<nom-branche>“ sur la branche courante (ici “master”)

git diff : repérer les fichiers en conflit suite au merge. Éditer les fichiers, corriger puis utiliser la commande : *git commit -a*

gitk : visualiser graphiquement les branches

git branch -d <nom-branche> : supprimer la branche ”<nom-branche>“ une fois fusionné avec la branche courante (sinon erreur)

git branch -D <nom-branche> : supprimer SANS VÉRIFICATION de fusion la branche ”<nom-branche>“

git remote show origin : voir les branches distantes

git push origin <nom-branche> : pousser la branche locale vers le dépôt distant.

git push origin –delete <nom-branche> : supprimer la branche dans le dépôt.

Sauvegarde de modifications pour les "transporter"

Voir : http://sametmax.com/soyez-relax-faites-vous-un-petit-git-stash/

git stash : en se plaçant dans le dossier principal du projet

git stash apply : pour réappliquer les modifications présentes dans le stash

Gestion des tags

git tag -l : lister les tags

git tag <nom-du-tag> <id-du-commit> : permet de créer un tag léger <nom-du-tag> sur le commit possédant la clé <id-du-commit>.

git push origin <nom-du-tag> : pousser un tag sur le serveur distant. Il ne sera pas pousser avec la commande *git push* par défaut.

git checkout <nom-du-tag> : basculer sur le tag <nom-du-tag>.

git tag -d <nom-du-tag> : supprimer le tag <nom-du-tag> localement.

git push origin :refs/tags/<nom-du-tag> : supprimer le tag <nom-du-tag> distant après l'avoir supprimé localement.

Corrections, annulations...

git commit –amend : permet de modifier le commentaire du dernier commit dans un éditeur.

git commit –amend -m “New commit message” : modifier le message du dernier commit (sans passer par l'éditeur)

git checkout . : annule les modifications en cours depuis le dernier commit

git reset HEAD^ : annule le dernier commit non propagé et restaure les fichiers. Ceci remplace la copie de travail telle qu'elle était avant le commit.

git reset –soft HEAD^ : annule le dernier commit non propagé, et conserver les modifications. –soft permet de conserver les modifications

git reset –soft HEAD~2 : annule les 2 derniers commit non propagé et conserve les modifications. HEAD~2 correspond a 2éme parents de HEAD

git rm –cached <fichier> : permet de supprimer le fichier <fichier> distant sans le supprimer localement.

git rm –cached -r <dossier>/ : permet de supprimer le dossier <dossier> distant sans le supprimer localement.

Commandes d'annulation DANGEREUSE

git reset –hard HEAD : annuler les changements effectués depuis le dernier commit. Supprime les fichiers non validés DEFINITIVEMENTS !

git reset –hard HEAD^ : supprimer le dernier commit. Cette action peut être répétée autant de fois que vous le désirez. Supprime les fichiers non validés DEFINITIVEMENTS !

git revert <commit> : restaurer le dépôt tel qu'il l'était lors du commit spécifié. Pour fonctionner, toutes les modifications doivent être validées (ou annulées avec *git reset*)

Récupération des changements d'un collègue

git remote add <mon-collegue> git://github.com/<mon-collegue> : créer un alias qui fait pointer <mon-collegue> vers l'adresse du dépôt. Permet d'éviter d'avoir à taper l'adresse complète à chaque fois.

git fetch <mon-collegue> : récupérer les changements que <mon-collegue> a effectués.

git merge <mon-collegue>/master : fusionne les modifications de <mon-collegue> avec la branche master locale.

git pull <mon-collegue> : réaliser en une seule commande fetch puis merge.

Analyser l'historique — Git log

git log :

git log v2.5.. : commits depuis (non-visible depuis) v2.5

git log test..master : commits visibles depuis master mais pas test

git log master..test : commits visibles depuis test mais pas master

git log master...test : commits visibles pour test ou master, mais pas pour les 2

git log –since=“2 weeks ago” : commits des 2 dernières semaines

git log Makefile : commits modifiant le Makefile

git log fs/ : commits qui modifient les fichiers sous fs/

git log -S'foo()' : commits qui ajoutent ou effacent des données contenant la chaîne 'foo()'

git log –no-merges : ne pas montrer les commits de merge

Comparer les commits — Git diff

git diff master..test : afficher la différence entre le sommet de deux branches

git diff master...test : afficher la différence entre l'ancêtre commun de deux branches

git diff : afficher les changements dans le répertoire de travail qui ne sont pas encore assemblés pour le prochain commit.

git diff –cached : montrer la différence entre l'index et votre dernier commit. Ce que vous committerez si vous lancez « git commit » sans l'option « -a ».

git diff HEAD : afficher les changements de votre répertoire de travail depuis votre dernier commit. Ces changements seront committés si vous lancez git commit -a.

git diff experimental : montrer la différence entre votre répertoire de travail actuel et la capture de la branche « experimental ».

git diff HEAD – ./lib : montrer les différences entre votre répertoire de travail actuel et le dernier commit (ou plus précisément, le sommet de la branche actuelle), en limitant la comparaison aux fichiers dans le répertoire lib.

git diff –stat : ajouter l'option –stat, qui limitera la sortie aux noms de fichier qui ont changés, accompagné d'un petit graphe décrivant le nombre de lignes différentes dans chaque fichier.

Sous-modules

git submodule add git://github.com/demouser/myproject.git <mon-dossier> : ajouter un dossier *mon_dossier* liée au dépôt *git git://github.com/demo-user/demo.git*.

Pour clôner un projet contenant des sous-modules :

- **git clone git://github.com/demouser/myproject.git** : clône le dépôt.
- **git submodule init** : initialise votre fichier local de configuration.
- **git submodule update** : tire toutes les données de ce projet et récupére le commit approprié tel que listé dans le super-projet.

Notes sur les branches

La branche master est la branche par défaut

git branch –track <nom-branche> origin/<nom-branche> : pour récupérer automatiquement les modifs de la branche <nom-branche> sur le dépôt distant origin/<nom-branche>.

Note sur le message du « commit »

Bien que ce ne soit pas obligatoire, il est assez efficace de commencer le message du « commit » avec une courte ligne (moins de 50 caractères) qui résume le changement, suivi d'une ligne blanche, puis d'une description plus complète. Les outils qui transforment les commits en mail, par exemple, utilisent la première ligne du commit pour le sujet du mail et le reste pour le contenu.

Les règles à appliquer pour rédiger un message de commit :

1. Rédiger le commit en anglais
2. Limiter la ligne du sujet à 50 caractères.
3. Utiliser une lettre capitale seulement sur le 1er caractère
4. Ne pas mettre d'espace avant le caractère deux-points (":")
5. Ne pas mettre de point à la fin de la ligne du sujet
6. Ajouter une ligne blanche entre la ligne du sujet et le corps du texte
7. Limiter les lignes du corps du texte à 72 caractères.
8. Utiliser l'impératif.
9. Décrire ce qui a été fait et pourquoi et non comment.

Comment fermer automatiquement des bugs via un message de commit sur Github

Utiliser la syntaxe : Fix #35

Voir : https://help.github.com/articles/closing-issues-via-commit-messages

Comment supprimer un label d'issue d'un dépôt ?

Utiliser Curl avec l'API de Github : `curl -i -u "<mon-email>" -X DELETE https://api.github.com/repos/<mon-login>/<mon-projet>/labels/<mon-label>`