



Rapport individuel des briefs projets

Réalisé par : EL OUADI ABDELALI

Table des matières

Definition de git :	3
Fonctionnement de git	3
Git alias	5
Excluding file :	5
Conflit résolution	6
Git tags	8
Stashing	8
Voyage sur github :	9
Protocoles de Communicate avec le client et le server	10
Gestion de projet	13

Definition de git :

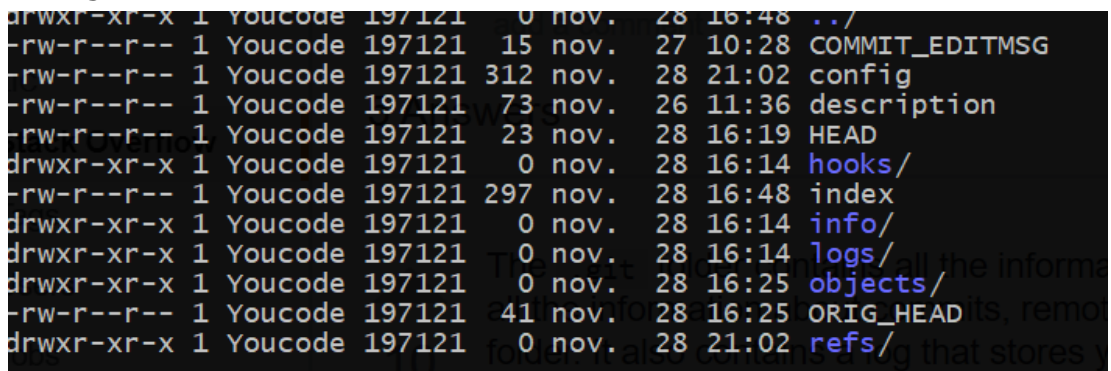
Git est un system de contrôle de version qui peut tracer tous les évolutions et les modifications qui on fait sur un ou plusieurs fichiers.

Fonctionnement de git

Pour commencer l'utilisation de git, il faut d'abord initialiser un dossier nommé. git par une commande « git commit »

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo
$ git init
Initialized empty Git repository in C:/Users/youcode/Desktop/Projects/Demo/.git/
```

Ce dossier contient toutes les informations qui permet git tracer l'évolutions des projets. La structure. git répertoire ressemble comme l'image suivante :



```
drwxr-xr-x 1 Youcode 197121  0 nov. 28 16:48 ../
-rw-r--r-- 1 Youcode 197121 15 nov. 27 10:28 COMMIT_EDITMSG
-rw-r--r-- 1 Youcode 197121 312 nov. 28 21:02 config
-rw-r--r-- 1 Youcode 197121  73 nov. 26 11:36 description
-rw-r--r-- 1 Youcode 197121  23 nov. 28 16:19 HEAD
drwxr-xr-x 1 Youcode 197121  0 nov. 28 16:14 hooks/
-rw-r--r-- 1 Youcode 197121 297 nov. 28 16:48 index
drwxr-xr-x 1 Youcode 197121  0 nov. 28 16:14 info/
drwxr-xr-x 1 Youcode 197121  0 nov. 28 16:14 logs/
drwxr-xr-x 1 Youcode 197121  0 nov. 28 16:25 objects/
-rw-r--r-- 1 Youcode 197121  41 nov. 28 16:25 ORIG_HEAD
drwxr-xr-x 1 Youcode 197121  0 nov. 28 21:02 refs/
```

- Une simple analyse de cette image
- Config :
 - Contient toutes les configurations qui on a créé pour chaque projet et on peut changer ces configurations par la commande suivante « git config -global ».
- COMMIT_EDITING :
 - This file contient la dernière « commit » .

- HEAD :
Est une référence à la branche actuelle qui active.
- Logs/ :
Ou' git stocker les noms de toutes les branches et les
« remotes ».
- Objects/ :
Contient tous SHA générer par git

Pour connait la différence entre notre zone de travail and la zone d'index on utilise :

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

→Ce dernier commentaire signifie qu'il y a aucun fichier au niveau de répertoire 'Demo'

Après création les fichiers de notre projet ou fait des changements au niveau des existants fichiers, il faut ajouter cette modification à la zone d'index pour préparer à la « commit » suivante.

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ touch Licence.md

youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git commit -m "Adding Licence.md file"
On branch master
Untracked files:
  Licence.md
```

```
nothing added to commit but untracked files present
```

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git add .
```

Pour valider ces modifications, il faut dire à git de stocker ces modifications au niveau .git dossier qui représente une base de données local, par la commande suivante pour les return au futur.

```
$ git commit -m "Adding Licence.md file"
[master 1284a5a] Adding Licence.md file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Licence.md
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
```

« -m » flag dans cet exemple nous donne l'option d'ajouter un message. Qui représente une explication de cette modification qu'on a fait.

Git alias

Les alias sont un moyen d'utiliser pour faciliter l'utilisation de git. Au lieu de créer une longue commande on peut remplacer par un alias par exemple :

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git config --global alias.historique "log --all --graph --decorate
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
```

Excluding file :

Parfois on peut que git ignore certains fichiers ou dossiers qu'on ne veut pas visionner dans l'historique de git. Dans ce cas on crée un fichier nommé

« .gitignore » et dans ce fichier il faut ajouter « path » ou l'extension de ces fichiers et dossiers.

Conflit résolution

Parfois qu'on a modifier le même fichier au sein de différente branche, ensuite nos peut fusionner ces deux branches, mais il apparait un conflit. A cause que git ne sait pas quelle modification il faut appliquer à ce fichier.

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git merge BAD
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

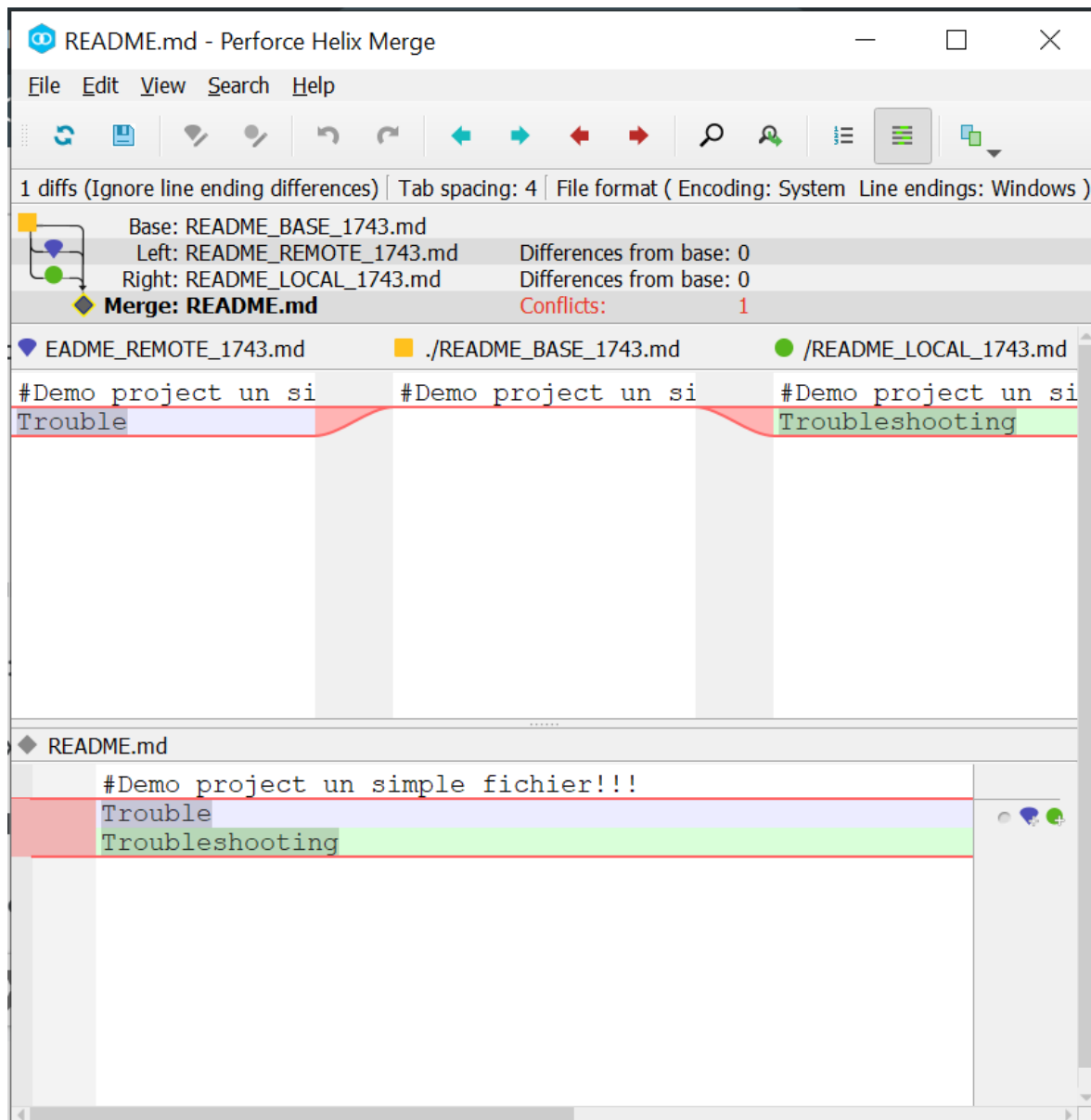
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master|MERGING)
```

Pour résoudre ce conflit. On peut d'installer le programme « p4merge » . Après l'installation, On a configuré git pour ajouter p4merge comme un « mergetool ». Ainsi on a édité la configuration de prompt au niveau de fichier. gitconfig :

```
MINGW64 ~/Desktop/Projects/Demo (master|MERGING)
$ git config --global merge.tool p4merge
```

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master|MERGING)
$ git config --global mergetool.p4merge.path 'C:\Program Files\Perforce\p4merge.exe'
```

Après la configuration de p4merge on peut utiliser la commande suivante « git mergetool » qui Affiche une interface qui montre les conflits des deux branches:



Nb! : Il y a trois méthode pour fusionner deux branches avec la commande « git merge »:

1 – fast forward : quand on crée une nouvelle branche à partir d’une autre branche et après on a modifié dans cette branche et pas dans l’autre,et après on fusionne dans ce cas merge est automatic.

2 – manuel : quand on crée une nouvelle branche à partir d’une autre branche et après on a modifié dans les deux branches, on rencontre, dans ce cas un conflit qui besoin d’une manière manuelle pour le résoudre, par p4merge par exemple .

3 – automatic .

Git tags

L'utilité des « tags » est marqué une point qui est important dans l'historique d'une répertoire, par exemple il est très utilisable pour marquer les versions d'une projet.

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
$ git tag -a V1.0 -m "RELEASE 1.0"

youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
$ git show V1.0
tag V1.0
Tagger: elouadi abdelali <elouadi.abdelali1997@gmail.com>
Date: Thu Nov 28 16:22:27 2019 +0100

RELEASE 1.0
```

Stashing

Git stash stocké tous les modifications au sein de la zone de travail et la zone d'index et rendre la zone de travail à l'état de dernier commit

```
Youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
$ git stash
Saved working directory and index state WIP on master: 9877283 fin
```

La commande « git stash list » nous donne une liste des stash et git status

```
Youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
$ git stash list
stash@{0}: WIP on master: 9877283 fin scenario 1

Youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
```

Et avec la commande “git stash pop” on peut retournera à l'état qu'on enregistré par stash, qui effacé ces modifications après l'utilisations de cette commande dans le stash zone.


```
Youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects2/demo (master)
$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direct
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (a285639bc85739bf298c12c2e2c84035ea496404)
```

Voyage sur github :

Github : c'est le plus grand web service pour l'hébergement de projets de Git.
Pour pousser un projet local vers un repo sur github.

Pour commencer l'utilisation de github, il faut ouvrir un compte sur GitHub,
on va créer un repo public et le cloner dans notre version locale.

```
Youcode@Desktop MINGW64 ~/Desktop/Projects2/demo (master)
$ git remote add origin https://github.com/abdelali/Projects-Github.git

Youcode@Desktop MINGW64 ~/Desktop/Projects2/demo (master)
$ git remote -v
origin https://github.com/abdelali/Projects-Github.git (fetch)
origin https://github.com/abdelali/Projects-Github.git (push)
```

Et pousser le tout vers le serveur Github :

```
Youcode@SpectRum MINGW64 ~/Desktop/Projects2/demo (master)
$ git push -u origin master --tags
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (25/25), 2.20 KiB | 322.00 KiB/s, done.
Total 25 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
To https://github.com/adamkhair/Projects-Github.git
 * [new branch]    master -> master
 * [new tag]       V1.0 -> V1.0
Branch 'master' set up to track remote branch 'master' from 'origi'
```

NB : pour pousser les tags aussi il ne suffit pas de faire le push de master mais aussi de ces derniers. On doit ajouter à la commande le nom du tag. En cas d'existence de plusieurs tags il faut simplement ajouter ' -tag'. Ainsi le -u « set-upstream » qu'on utilise dans la commande push a pour rôle d'ajouter une référence de suivi au serveur en amont sur lequel vous poussez.

- Github et les branches :

Quand on pousse les branches locale (excepte la branche principale) sur github, ce dernier nous donne une possibilité, avant le fusionnement de cette branche avec la branche principale, la création d'une « pull request ». Pour réviser ces changements par les autres développeurs de l'équipe et après on peut les fusionner.

- Repo distant et repo local :

Dans le cas d'une modification sur le repo distant (github). Qui n'est pas sur notre repo locale on peut ajouter ces modifications par « git pull » qui fusionner ces changements avec la branche master. C'est dans le cas on veut voir les changements avant le fusionnement on peut faire ça avec la commande « git fetch »

Protocoles de Communicate avec le client et le server

Dans le monde de world wide web les protocoles les plus utilisés pour faire une communication entre les servers et les clients (browser par exemple). En autre part il y a ssh (secure shell), pour utiliser ce dernier pour faire une communication entre notre repo local et repo github . on

doit obtenir une public clé et une clé privée . on obtient les clés par la commande suivante :

```
lenovo@DESKTOP-5RK8090 MINGW64 ~/Desktop/project/demo/.SSH (master)
$ ssh-keygen -t rsa -C "elouadi.abdelali1997@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/lenovo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/lenovo/.ssh/id_rsa.
Your public key has been saved in /c/Users/lenovo/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:4BZhaweodxovY26B4dCZGb5VXxgwhpawDgjrGgNBz9E elouadi.abdelali1997@gmail.com
The key's randomart image is:
+---[RSA 3072]-----+
|  o... ==*o          |
| ..*.*E=*+.o        |
| o+.B+O*+           |
| o..00+.++          |
| +. .o+S            |
| o. ..              |
| +                  |
| .o                 |
| o.                 |
+-----[SHA256]-----+
```

D'après l'obtention des clés il faut donner notre « github repository » la public

Personal settings
Profile
Account
Security
Emails
Notifications
Billing
SSH and GPG keys
Blocked users
Repositories
Organizations

SSH keys / Add new

Title

first communication using ssh

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGDYWAuKL7ENB1sbl2/b37MpD2byUTSRa0G5moi8Rw3xkEOXTGr6qNiAajyn
39ljdrI219O+i5H9pgr8yyyqzdsT0s9h5QRizHbBJMBk3Y89bb5R2j3wSSQBjjQ5T9mXe/csz/HIXYpgA35N+Q5NvtFrZG
tew1tIE+FoTMpL6+eq2nOq0rIvD5H7FPELcE02/uF5nKpCEl2UpJalwHosyHEY3XV67/2RBPmcZ3dFctwrK37ivyBEa1FT6r
We24E/VCIJ+GgzxqR2pQs7p5G75j4CkRxiXSP4RHtw45mNCY7YCIE5DuK+CefJW6a0UY7n1E2yiQpBYGSolDfpxmCRJe
wggZEfCwWd16Oqw/5wzuNPavpISO1BfkOhxOKh6+K9zeYGBNeY+jDYUugSjY4XM2rjulKgwgmKE3pRSn4anpAL876
+wwqIE2GkCp6Y/lgy6gQ++QPKMeyRImuvzBRMyW+5UYtpO/Cz1Gp3Ncd40N0Fsz9xmExwCbUtlAZHs+s=
elouadi.abdelali1997@gmail.com
```

Add SSH key

clé comme dessus

Et après on change le « remote » de https à SSH :

```
git remote set-url origin git@github.com:abdelalilwafi/ssh.git
```

Rebasing

Pour fusionner deux branches, il existe 2 commande faire la même chose sont :

« git merge » et « git rebase »,seule la manière de procéder diffère:

- 1 – git merge ajoute nouveau commit.
- 2 – rebase prend les différentes commit de 2 branches et fusionner dans la branche désirée.
- 3 – merge prend toutes les commits de la branche et ajouter dans la branche désirée.

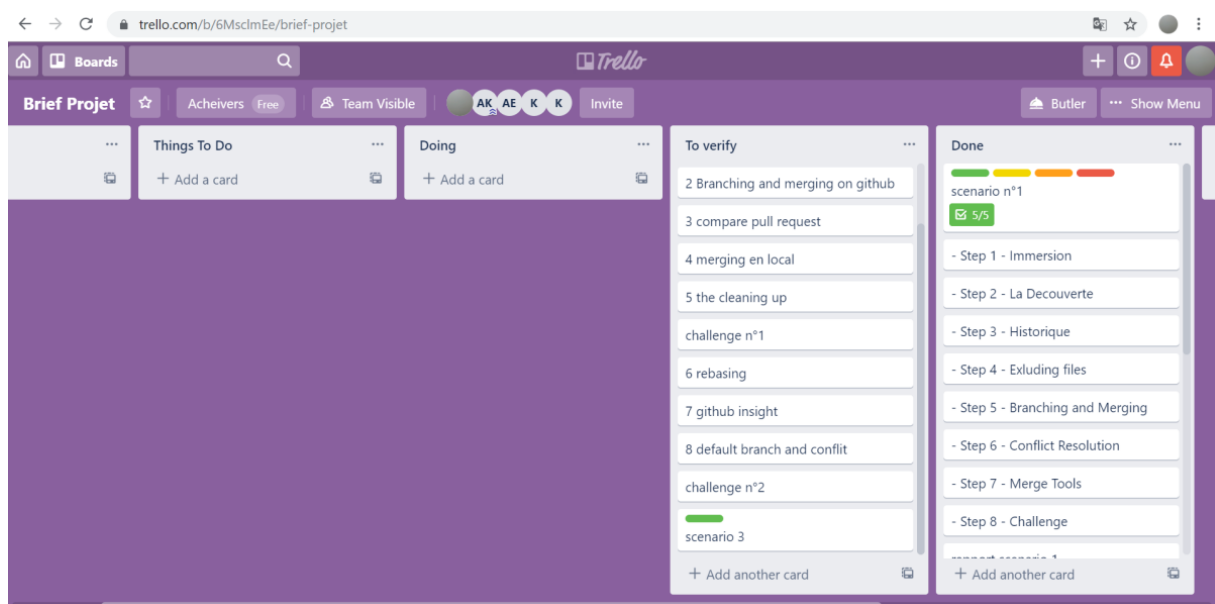
NB : « git rebase » n'est pas recommandé en cas nos travaille avec un groupe de développeur sur le même projet.

Parmi les autres utilité de « rebase », est modifier, supprimé, les commit avec « interactive rebase » 'git rebase -i HEAD~n' , dans ce cas , n représente le nombre des derniers commit.

Gestion de projet

Trello est un outil collaboratif gratuit, permettant de travailler en ligne et à distance sur des projets communs. Les projets sont gérés à travers de tableaux, où sont créées des listes et des cartes .

Dans ces briefs projets, où chaque scenario dépend de l'autre, donc c'est une bon chance pour appliquer la méthode « scrum », et trello nous aide et facilite l'application de cette méthode



Conclusion :

Dans ces briefs projet on a explorer le monde de git, leur utilité, et la manière d'utilisation et ces commandes les plus important. Avec git on a appliqué la méthode scrum par l'utilisation de l'application trello.