



RAPPORT INDIVIDUEL

EN CARDE PAR : HANANE JABANE

REALISE PAR :

MAJDA FANNAN

Table des matières :

| | |
|---|---|
| Introduction:..... | 2 |
| Scenario 1 : | 3 |
| Immersion et decouverte : | 3 |
| Historique : | 4 |
| Excluding files : | 4 |
| Branching and Merging, conflict resolution, P4merge:..... | 5 |
| Sceario 2 : | 6 |
| Tagging : | 6 |
| Stashing and Saving work in Progress: | 7 |
| Github, clone and push : | 7 |
| Fetch and pull : | 8 |
| Scenario 3 : | 8 |
| Compare and pull request:..... | 8 |
| Rebase:..... | 9 |

Introduction:

Dans tous les projets, on commence premièrement par la gestion des projets, La gestion de notre projet c'était sur la base de la méthode agile spécifiquement SCRUM.

Scrum est une méthode agile dédiée à la « **gestion de projet** ». Cette méthode de gestion, a pour objectif d'améliorer la productivité de notre équipe.

Repartitions des rôles dans scrum :

Master scrum:

Facilite la communication au sein de l'équipe.

Cherche à améliorer la productivité et le savoir-faire de son équipe.

Dans notre projet le scrum master c'est KHADIJA AMERDOULE.

L'équipe:

Tous les membres de l'équipe apportent leur savoir-faire pour accomplir les tâches.

Notre equipe: ADAM KHAIRI

MAJDA FANNAN

KHADIJA MELOUANE

MEROUANE

KHADIJA AMERDOULE

Le Product Owner :

Joue le role du client

Et deuxièmement par la gestion du temps et aussi la gestion et la planification des taches des projets et pour cela on a utilisé « Trello », c'est un outil de gestion de projet en ligne.

Et cet outil nous permet de suivre la progression des tâches de notre groupe et aussi de s'organiser facilement.

Scenario 1 :

Dans le premier scenario en travaille premièrement pour savoir git et leur environnement.

Immersion et decouverte :

Git est un Logiciel de contrôle de version qui permet de tracer l'évolution de projet et d'y apporter des modifications sereinement. Créé par Linus Torvalds. Il est utilisé pour gérer des codes sources.

Afin d'utiliser git la première étape est de configurer les config global de git:

```
git config --global user.name "Name"
```

```
git config --global user.email "email"
```

On a commencé par créer un repository et pour créer ce dossier, on doit utiliser la commande **mkdir « nom de dossier »**. Après on va se déplacer à ce dossier et taper la commande **« git init »** est une commande que vous utilisez durant la configuration initiale du nouveau dépôt. Cette commande créera un nouveau sous-répertoire `‘.git’` habituellement caché, il contient l'historique des commits du projet. Il garde en mémoire tous les changements qui ont été comités.

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop
$ cd projects1

youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/projects1
$ mkdir demo

youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/projects1
$ cd demo/

youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/projects1/demo
$ git init
Initialized empty Git repository in C:/Users/youcode/Desktop/projects1/demo/.git/
```

Les fichiers de configuration `‘.git’`

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo/.git (GIT_DIR!)
$ ls -al
total 13
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:40 ./
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:37 ../
-rw-r--r-- 1 youcode 197121 22 nov. 26 11:40 COMMIT_EDITMSG
-rw-r--r-- 1 youcode 197121 130 nov. 26 11:36 config
-rw-r--r-- 1 youcode 197121 73 nov. 26 11:36 description
-rw-r--r-- 1 youcode 197121 23 nov. 26 11:36 HEAD
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:36 hooks/
-rw-r--r-- 1 youcode 197121 137 nov. 26 11:40 index
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:36 info/
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:40 logs/
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:40 objects/
drwxr-xr-x 1 youcode 197121  0 nov. 26 11:36 refs/
```

On va expliquer les clauses de dossier. git :

Head : contient des informations sur la branche et les commit qu'il contienne etc.

Log : l'historique des commits

Branch : c'est un endroit contenant les branches effectués

Historique :

Après cette étape Nous allons voir comment nous pouvons adapter git à nos propres besoins avec Les ALIAS.

Tous d'abord un alias c'est un technique très utile, par exemple dans notre cas on utilise cette commande : `git config --global alias.historique 'log -1 HEAD'`

On remplacer un alias qui s'appel historique par la commande « `git log --graph --oneline` »
Après si on lance la commande « `git historique` » le même résultat va apparaître.

Excluding files :

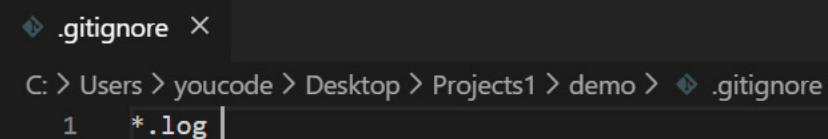
Ensuite on a créé un fichier '.gitignore'

Si vous créez un fichier dans votre référentiel nommé .gitignore, Git l'utilise pour déterminer les fichiers et les répertoires à ignorer avant d'effectuer une validation.

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

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

Ensuite on ajoute la ligne suivante « *.log » dans le fichier, cette ligne explique que tous les fichiers qui se terminant par .log doit exclure.



The screenshot shows a code editor window titled '.gitignore' with a close button. The file path is 'C:\> Users > youcode > Desktop > Projects1 > demo > .gitignore'. The content of the file is a single line: '1 *.log |'.

```
youcode@DESKTOP-OC51QH5 MINGW64 ~/Desktop/Projects/Demo (master)
$ git ls-files . --ignored --exclude-standard --others
application.log
```

Branching and Merging, conflict resolution, P4merge:

D'abord merging c'est le fait de vouloir ajouter dans une branche A les mises à jour que vous avez faites dans une autre branche B, dans notre cas la Branche A c'est la branche principale 'Master' et la Branche B c'est la branche 'BAD'.

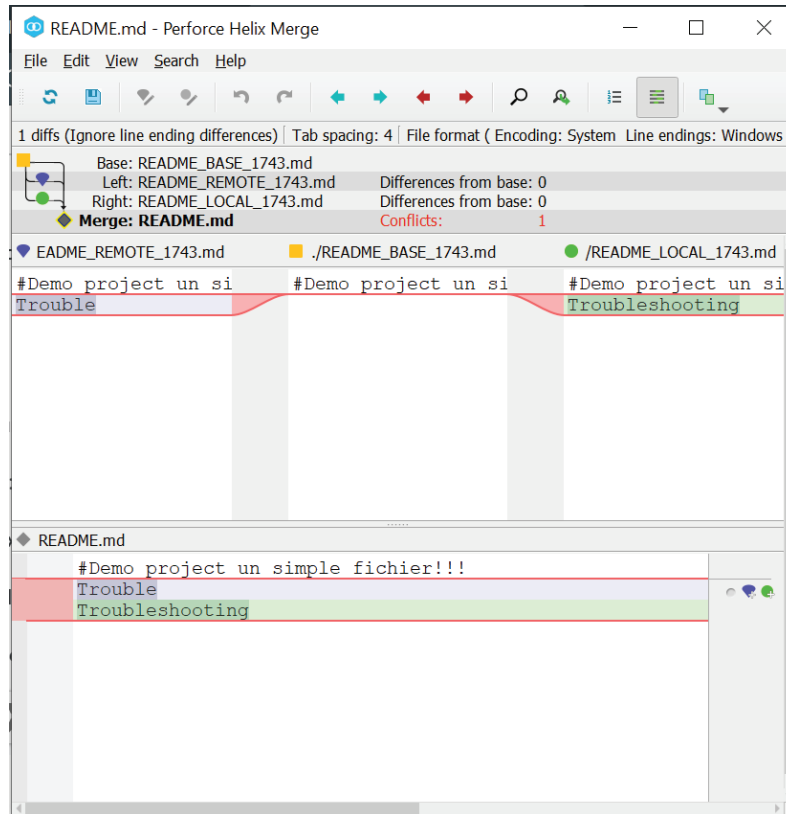
Dans cette étape on a faire une modification dans le fichier README.md, dans les deux branches. La branche master qui contient une ligne de texte 'troubleshooting', et la branche BAD qui contient une ligne de texte 'trouble'

Après lorsque on fait la commande « git merge Bad » il apparait un message de conflit entre les deux branches.

```
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.
```

Maintenant on doit résoudre le problème de choisir la modification qui nous doit garder en ouvrant directement le fichier posant problème dans la console.

On peut aussi travailler avec P4MERGE c'est un outil propose une interface graphique pour comparer les différences de versions d'un fichier.



Scenario 2 :

Dans le scenario 2 on travaille sur des fonctionnalités avancées Comme le tagging, le stashing et le push and clone et le fetsh and pull.

Tagging :

Le tag qui est pour rôle de faire une étiquette sur une version de projet, il y a deux type de tag « annotated tag » and « lightweight tag » et dans cette scenario j'apprends

Comment créer un tag par cette commande :

Git tag -a V1.0 -m "my version 1.0"

Comment afficher les informations des tags avec la commande :

Git show

Aussi comment lister les tags avec la commande :

git tag

Stashing and Saving work in Progress:

`git stash`

Git stash cette commande nous permet de revenir au dernier commit sans perdre le brouillon de notre projet.

`git stash list`

Cette commande nous donne la liste des stash utilise

`git stash pop`

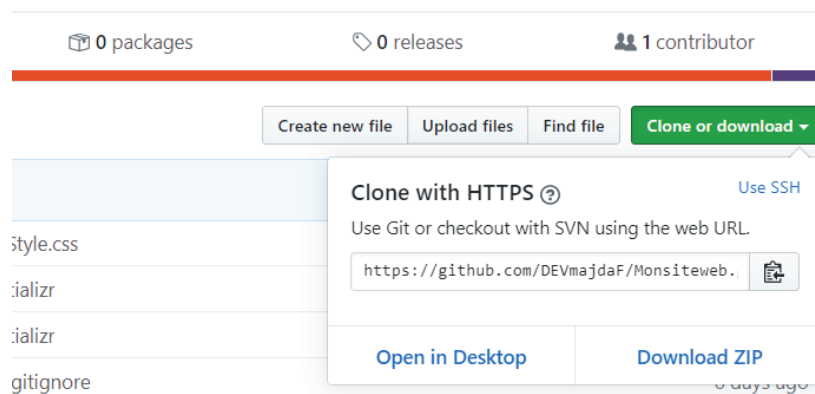
Va réintroduire vos modifications et effacer la zone de stashing.

Github, clone and push :

GitHub est une plateforme open source de gestion de versions et de collaboration destinée aux développeurs de logiciels, GitHub permet aux développeurs de modifier, d'adapter et d'améliorer le logiciel gratuitement à partir de référentiels publics. Elle repose sur Git, un système de gestion de code, dans le but d'accélérer le développement logiciel.

Cloning :

Lorsque vous créez un repository sur GitHub, il existe en tant que repository à distant. Vous pouvez cloner votre repo pour créer une copie locale sur votre ordinateur et effectuer une synchronisation entre les deux emplacements.



1. Copiez la commande clone.

2. Dans votre terminal git, accédez au répertoire local dans lequel vous souhaitez cloner votre référentiel.
3. Passez la commande que vous avez copiée depuis GITHUB sur le terminal git.

Pushing :

Après avoir fait vos modifications et votre commits, tu dois les transférer dans votre repo distant, afin que d'autre personne puissent les voir également.

Taper git push sur la ligne de commande pour envoyer vos commits de votre référentiel local vers GITHUB.

Fetch and pull :

La commande git fetch va récupérer toutes les données des commits effectués sur la branche courante qui n'existent pas encore dans votre version en local. Ces données seront stockées dans le répertoire de travail local mais ne seront pas fusionnées avec votre branche locale. Si vous souhaitez fusionner ces données pour que votre branche soit à jour, vous devez utiliser ensuite la commande git merge.

Git fetch

Mais pour La commande git pull c'est la commande qui regroupe les commandes git fetch suivie de git merge. Cette commande télécharge les données des commits qui n'ont pas encore été récupérées dans votre branche locale puis fusionne ensuite ces données.

Git pull

Scenario 3 :

Dans la troisième scenario on travailler sur deux principaux techniques « compare and pull request » et le « Rebase »

Compare and pull request:

Les demandes de pull affichent des différences pour comparer les modifications apportées dans votre branche à la branche principale dans laquelle vous souhaitez fusionner vos modifications.

Les pull requests sont une fonctionnalité facilitant la collaboration des développeurs avec github.

ajustement (less than a minute ago)

Compare & pull request

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾

DEVmajdaF Merge pull request #1 from DEVmajdaF/annulation

Latest commit 0fa8caa 10 minutes ago

| | | |
|------------|-------------------|------------|
| css | Create Style.css | 3 days ago |
| fonts | copy initializr | 5 days ago |
| js | copy initializr | 5 days ago |
| .gitignore | Create .gitignore | 5 days ago |
| README.md | editing Readme.md | 5 days ago |

Rebase:

Rebasing est un peu différent. Au lieu de fusionner l'historique de vos deux branches à partir d'un commit de base, vous donnez l'impression que vous avez créé votre branche à partir d'un commit différent de celui que vous aviez initialement. Cela vous permet de conserver un historique de projet linéaire. Cela rend beaucoup plus facile à l'avenir de revenir sur une modification si vous avez un bogue, car vous n'aurez pas à chercher où le commit a été effectué dans la timeline.

La git rebase c'est une commande qu'elle poursuit le même objectif que git merge. Ces deux commandes permettent d'intégrer des changements d'une branche dans une autre. Seule la manière de procéder diffère.