LE RAPPORT INDIVIDUEL DE LA BRIEF PROJET NUMÉRO 1

(LE GIT)

Réaliser par :

Khadija ZERZKHANE

La formatrice :

Hanane JABANE

Classe 2

2019/2020

Table des matières :

Introduction :

Dans le cadre de la validation des compétences de la période SAS ; le brief projet est un moyen utile pour que vous validiez les compétences dans leur niveau respectif (N1, N2, N3), ce projet se basé sur le git et quelques moyens de gestion du travail et de temps comme la méthode scrum le trello …

La méthode Scrum :

scrum est une méthode de développement agile orientée projet informatique dont les ressources sont régulièrement actualisées.

Learning by doing :

Méthode pédagogique active qui encourage le collaborateur à s'autoperfectionner. Il s'agit d'une approche ouverte sur l'environnement professionnel qui dépasse le simple entraînement répétitif, en favorisant l'initiative dans le développement permanent des compétences.

Le git :

Git a été créé par Linus Torvald début 2005 sous la licence GNU GPL version 2 et la première version stable est sortie en Décembre 2005, soit moins d'un an après le début du développement.

Une intention particulière a été placée sur l'optimisation de son fonctionnement sur un noyau Linux.

A bien des égards, Git peut être considéré comme un système de fichiers à part entière, possédant son propre système de versionnement. Avec le temps et la participation active de la communauté, Git se trouve doté de toutes les fonctionnalités d'un logiciel de gestion de versions.

Définition du trello :

Trello est un outil de gestion de projet en ligne, lancé en septembre 2011, et inspiré par la méthode Kanban de Toyota. Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.

La version de base est gratuite, tandis qu'une formule payante permet d'obtenir des services supplémentaires. Le service est disponible en plusieurs langues (23 en juin 2016).

Le brief projet

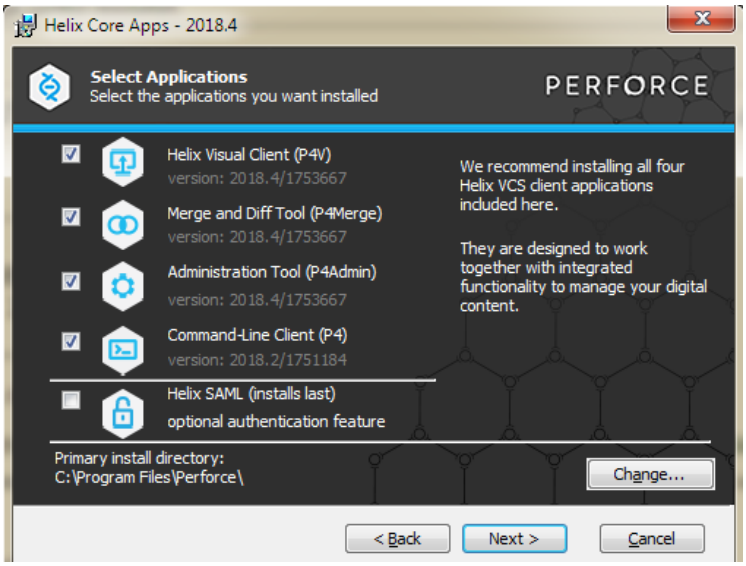
scénario 1:

Taper la commande « git mergetool » et conclusion:

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master|MERGING)
$ git mergetool
Merging:
README.md

Normal merge conflict for 'README.md':
  {local}: modified file
  {remote}: modified file
The merge tool p4merge is not available as 'C:\Program Files\Perforce\p4merge.ex
e'
```

On obtient un conflit de merge avec le message disant que l'outil p4merge n'est pas disponible donc on devrait l'installer pour l'utiliser pour le mege.

1) L'Installation de p4merge sur notre Pc :



1) Git config –global merge.tool p4merge

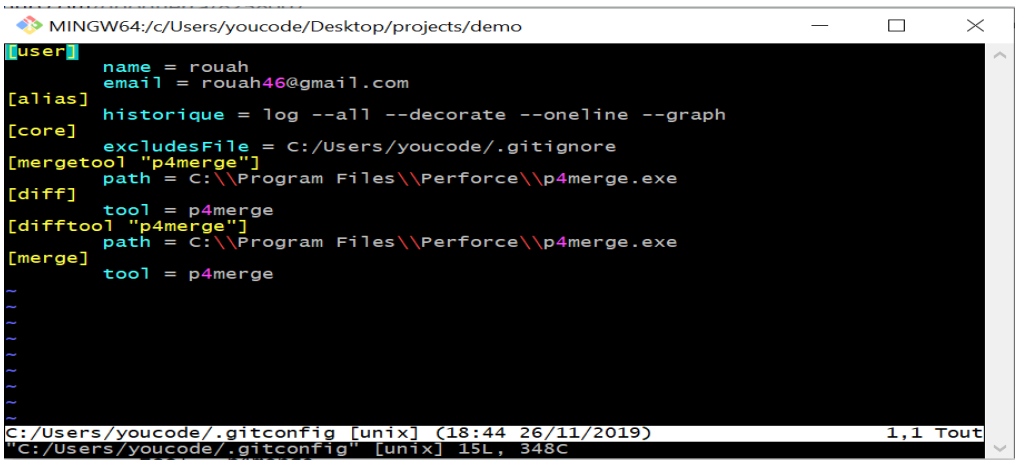
```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master|MERGING)
$ git config --global merge.tool p4merge
```

2) Git config –global mergetool.p4merge.path “lien d’installation” (.exe)

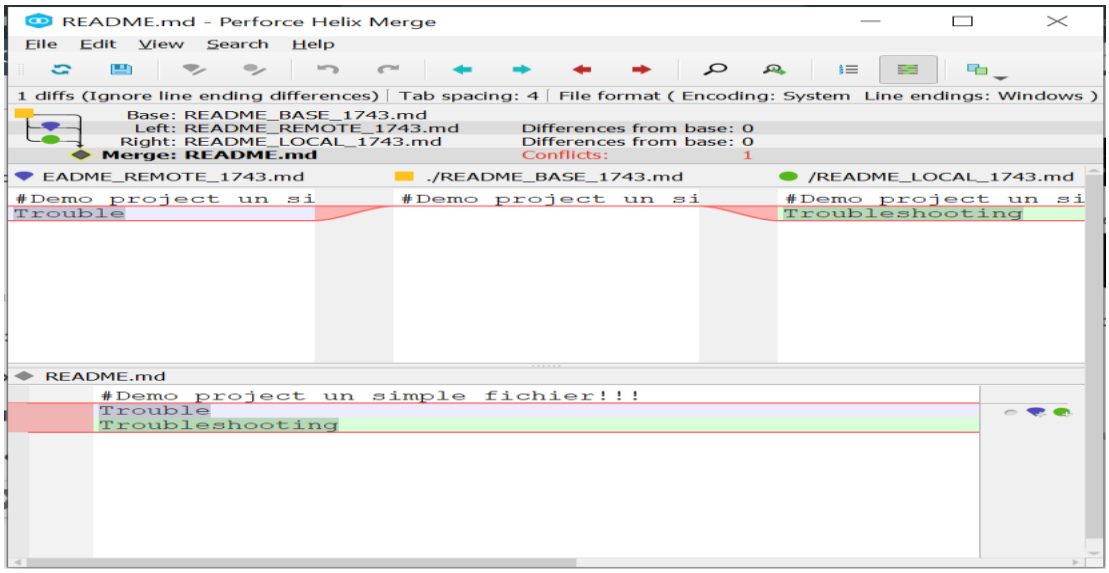
```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master|MERGING)
$ git config --global mergetool.p4merge.path "C:\Program Files\Perforce\p4merge.
exe"
```

3) Configuration du prompt :

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master|MERGING)
$ git config --global --edit
```



1) Taper la commande git mergetool :



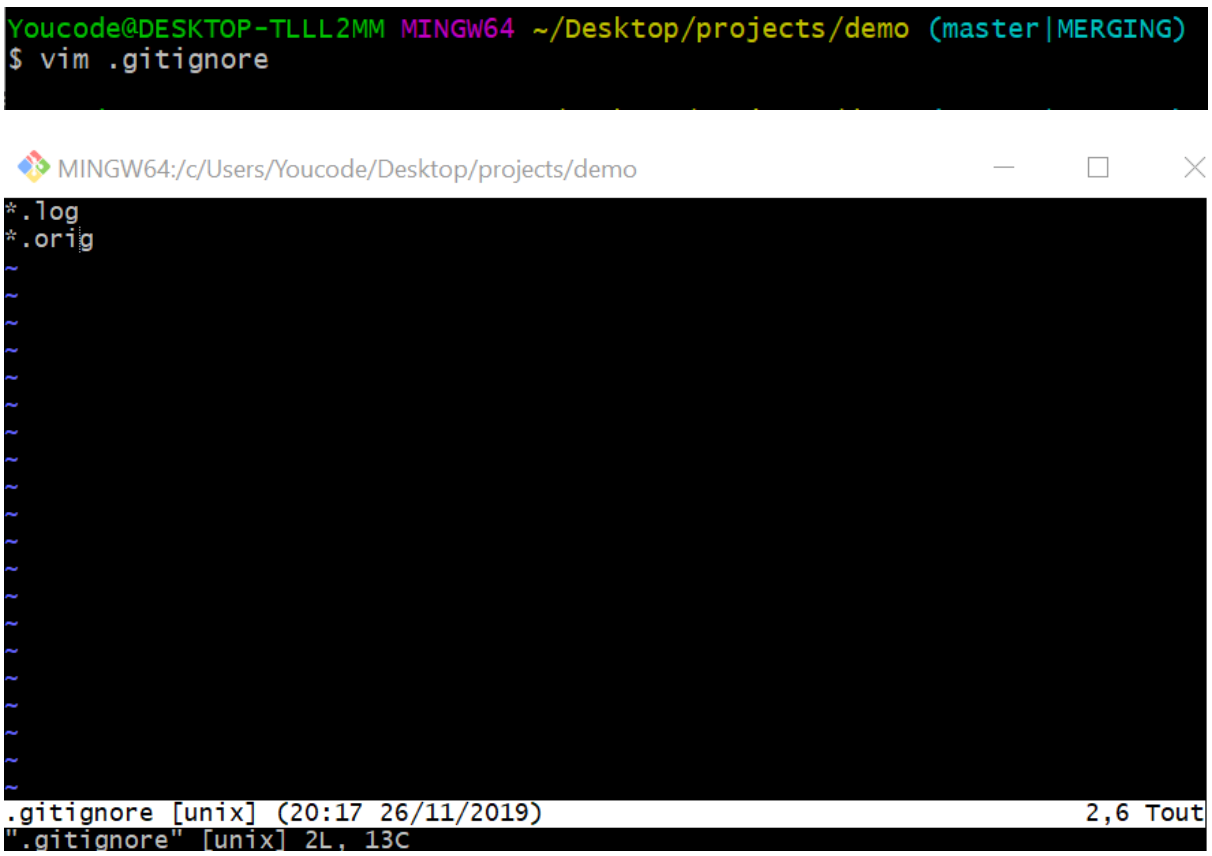
2) Analyser et expliquer la plateforme ouverte en temps réel :

On obtient la plateforme ci dessus qui affiche en haut à gauche la modification apportée à la branche master, la partie en haut à droite représente la branche BAD. Celle du milieu est celle où devrait se trouver le merge des deux modifications.
La partie du bas et celle qu'on peut modifier manuellement pour résoudre le conflit du merge.

#8: Challenge

1) Sur le fichier **.gitignore** ; écrire une clause pour rejeter les fichiers indésirables et redondants

On laisse que les fichiers : **licence.txt Readme.md .gitignore** (exemple *.log pour application.log)



Scénario 2 :

#1 :Tagging

1. Se déplacer sur la branche Principale:
2. Créer un TAG avec un nom V1.0 et un commentaire ‘ RELEASE 1.0 ‘

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git tag -a v1.0 -m 'REALEASE 1.0'
```

3. Afficher les informations sur le TAG:

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git show v1.0
tag v1.0
Tagger: rouah <rouah46@gmail.com>
Date:   Wed Nov 27 10:59:12 2019 +0100

REALEASE 1.0

commit 37de517c88345990c307a9e87f176a94b4257236 (HEAD -> master, tag: v1.0)
Merge: 13e1bff 4009b2f
Author: rouah <rouah46@gmail.com>
Date:   Wed Nov 27 09:44:59 2019 +0100

    4

diff --cc .gitignore
index 397b4a7,397b4a7..3598a4d
--- a/.gitignore
+++ b/.gitignore
@@@ -1,1 -1,1 +1,2 @@@
  *.log
++*.orig
```

4. Le rôle d’un TAG :

Les tags sont un aspect simple de Git, ils permettent d’identifier des versions spécifiques du code. Ils peuvent être considérés comme une étiquette, comme une branche qui ne change pas. Une fois créé, il perd la possibilité de changer l'historique des commits.

#2 : Stashing and Saving work in Progress

1) Modifier le fichier README.md et ajouter une ligne:

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
vim README.md
```

2) Tapez la commande git Stash ?

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/demo (master)
$ git stash
Saved working directory and index state WIP on master: dd35bd6 modif licence
```

3) Expliquer le fonctionnement de la commande git Stash

Git Stash permet d’ouvrir une nouvelle fenêtre de travail sans inclure

4) Taper la commande git stash list

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git stash list
stash@{0}: WIP on master: 37de517 4
```

5) Exécuter la commande git status

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

6) Conclusion:

Utiliser git stash c'est comme ouvrir une nouvelle page vide où on n'observe rien des modifications précédentes. Même si on a modifié avant le fichier README avant sans commiter, il ne le montre pas et nous dit qu'il y a rien à commiter.

7) Modifier le fichier « Licence.txt » et ajouter la ligne « APACHE 2.0 »

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ vim Licence.txt
```

8) Faire le staging et le commit en une seule ligne :

```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git commit -am 'modif licence'
warning: LF will be replaced by CRLF in Licence.txt.
The file will have its original line endings in your working directory
[master 7689cd3] modif licence
 1 file changed, 1 insertion(+)
```

9) Exécuter la commande « git stash pop »

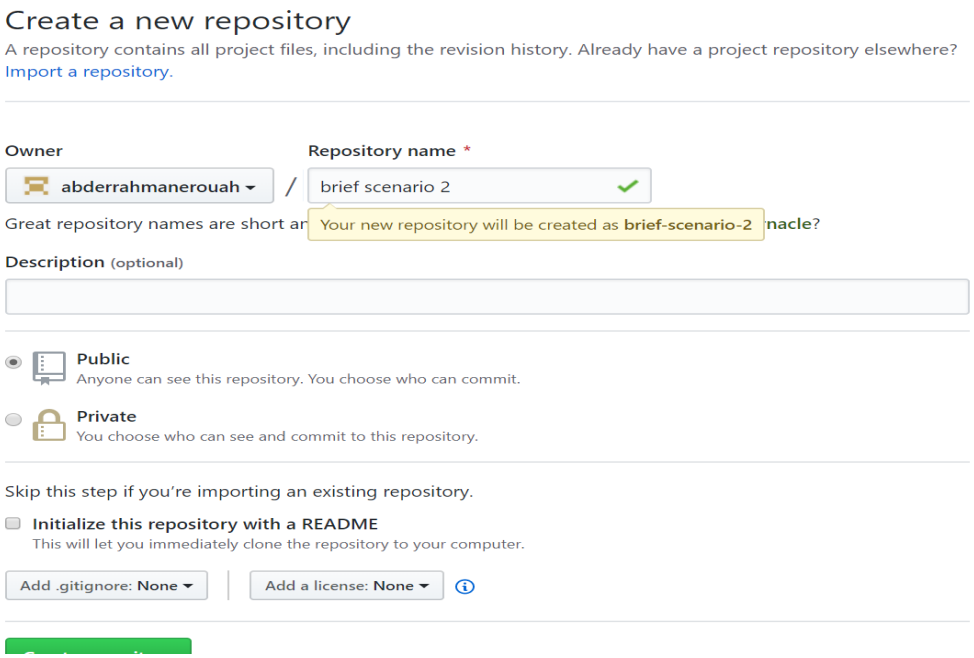
```
youcode@DESKTOP-KVTCEUL MINGW64 ~/Desktop/projects/demo (master)
$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (b5f16eef6b7539dd7086d360bbe8c0ecb64f0310)
```

10) Conclusion:

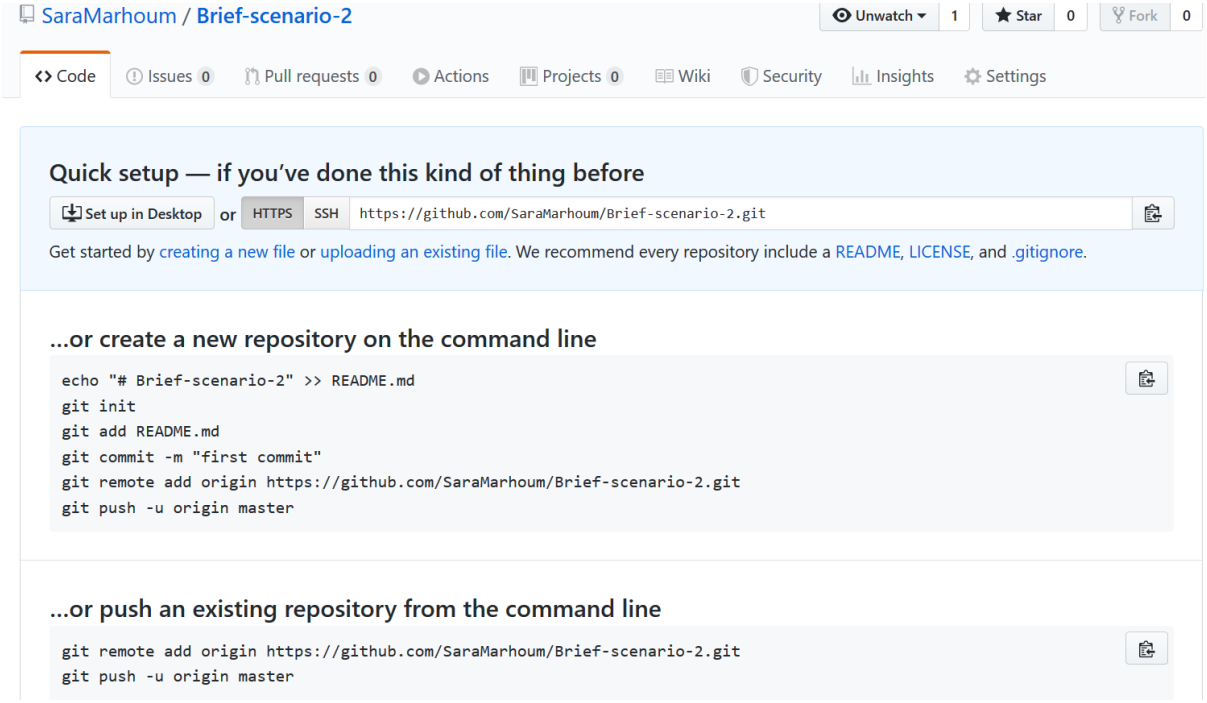
On constate que la dernière modification effectuée sur le fichier Licence.txt n'a pas été prise en compte comme si ça n'a jamais été fait et on ne vois que le document modifié avant et reste en attente d'être commit.

#3: Voyage sur Github, Local Repo to github Repo

1) Créer un repo github public sans ajouter le fichier README.md



2) Créer le remote en https :



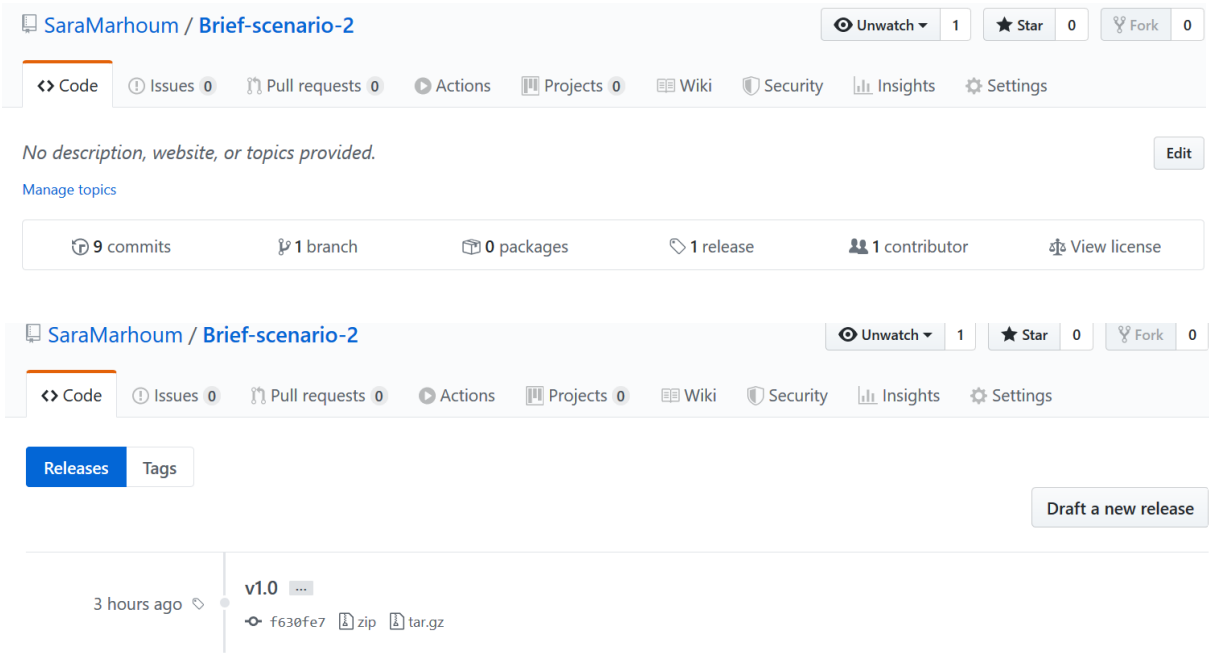
5) git remote add origin https://github.com/SaraMarhoum/Brief-scenario-2.git

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/demo (master)
$ git push -u origin master --tags
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 8 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (28/28), 2.43 KiB | 311.00 KiB/s, done.
Total 28 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
To https://github.com/SaraMarhoum/Brief-scenario-2.git
 * [new branch]      master -> master
 * [new tag]         v1.0 -> v1.0
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

3) Expliquer la commande :

La commande permet de transporter le répo créé sur Github vers le local Git et créer un lien entre les deux.

4) Sur Github Vérifier la liste des commits , les branches , les releases et les tags :



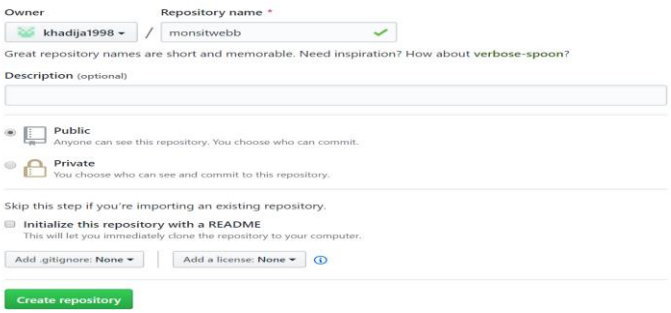
#4 : Mini challenge (optionnel)

En examinant les types d'authentification sur GITHUB, on tombe sur l'authentification HTTPS et SSH, certes HTTPS est beaucoup plus facile à manager tandis que le SSH est plus sécurisé tandis que fiable en ce qui concerne les transferts cryptés.

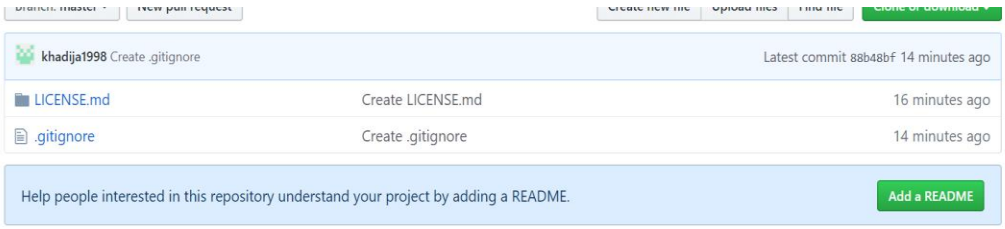
Le but de ce challenge est de créer une authentification SSH entre votre repo local et le repo GITHUB.

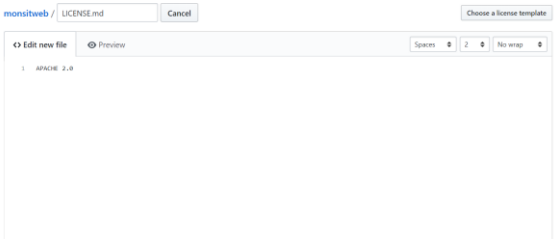
#5: Création d'une local copy :

1) Sur github créer un autre repo nommé (Monsiteweb)



2) Ajouter à l'arborescence toujours sur github le fichier **.gitignore** et un fichier **licence.txt 'APACHE 2.0 '**





3) Se déplacer dans le répertoire projets sous GIT:

```
Youcode@DESKTOP-4BCESEH MINGW64 ~/Desktop (master)
$ cd projects
```

4) Créer un clone github vers le local sous le nom (Monsiteweb-local)

```
Youcode@DESKTOP-4BCESEH MINGW64 ~/Desktop/projects (master)
$ git clone https://github.com/khadija1998/monsitweb.git Monsiteweb-local
Cloning into 'Monsiteweb-local'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
```

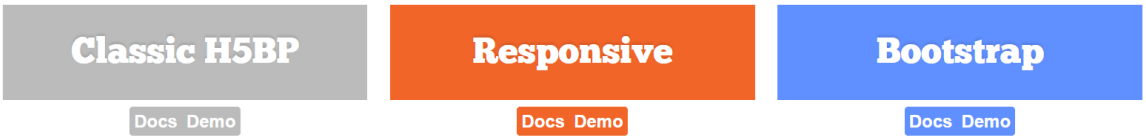
5) Vérifier si le clone est créé

```
Youcode@DESKTOP-4BCESEH MINGW64 ~/Desktop/projects (master)
$ ls
demo/  Monsiteweb-local/
```

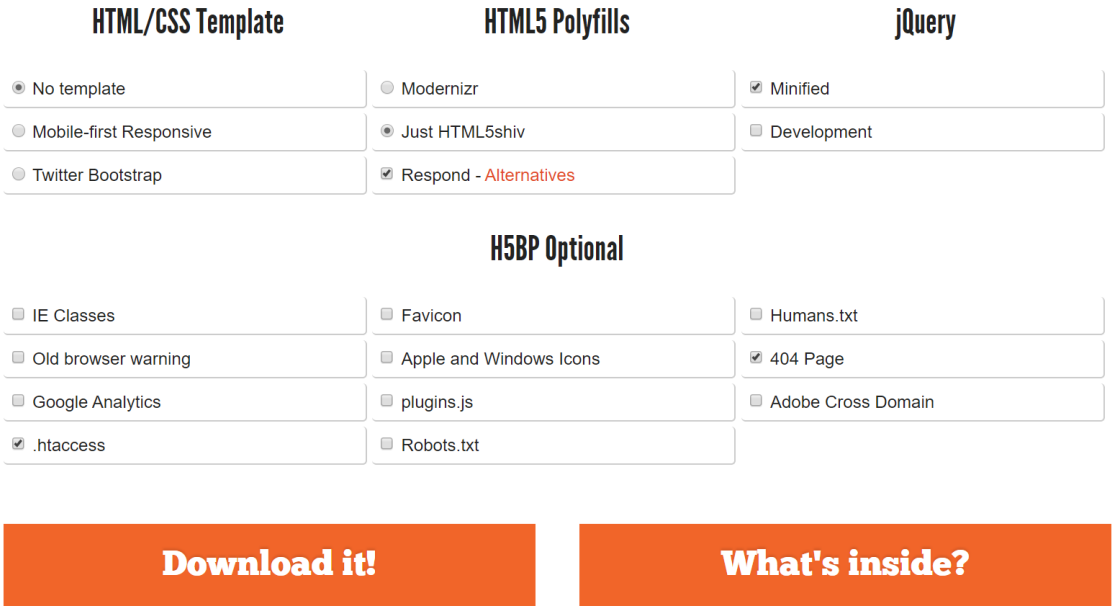
#6: Sending the website :

1) Télécharger le site web depuis le lien suivant : http://www.initializr.com/

1 - Pre-configuration



2) Sur le site telecharger un site bootstrap avec le fichier **.htaccess** et le fichier **404.html**



3) Analyser l'arborescence.

```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ cp -R  ~/DESKTOP/initializr/* .
```

6)Copier le site télécharger dans le repo local à travers une seule commande :

```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ cp -R  ~/DESKTOP/initializr/* .
```

Git status

7) Faites le stagging et le commit en une seule ligne

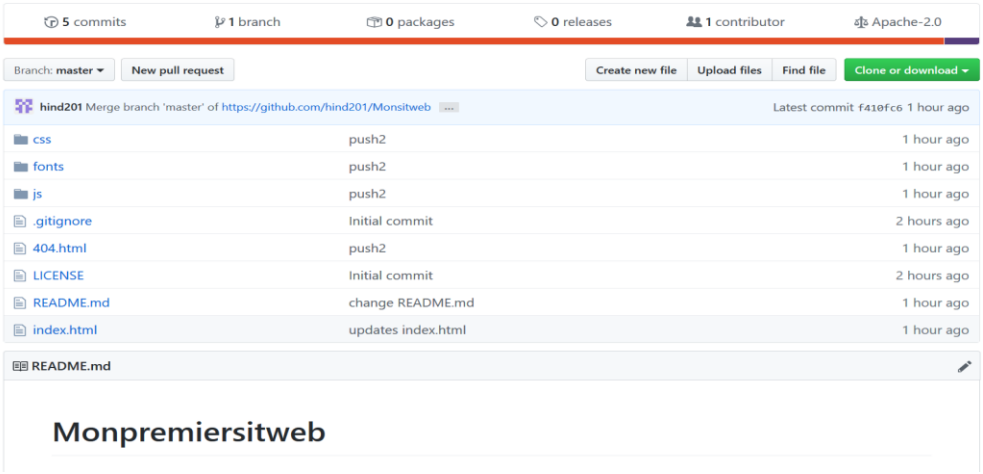
```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        404.html
        css/
        fonts/
        index.html
        js/

nothing added to commit but untracked files present (use "git add" to track)
```

```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git commit -am 'monsitweb'
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
        404.html
        css/
        fonts/
        index.html
        js/

nothing added to commit but untracked files present
```

8) Faite le push à github

```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git push origin master
Everything up-to-date
```

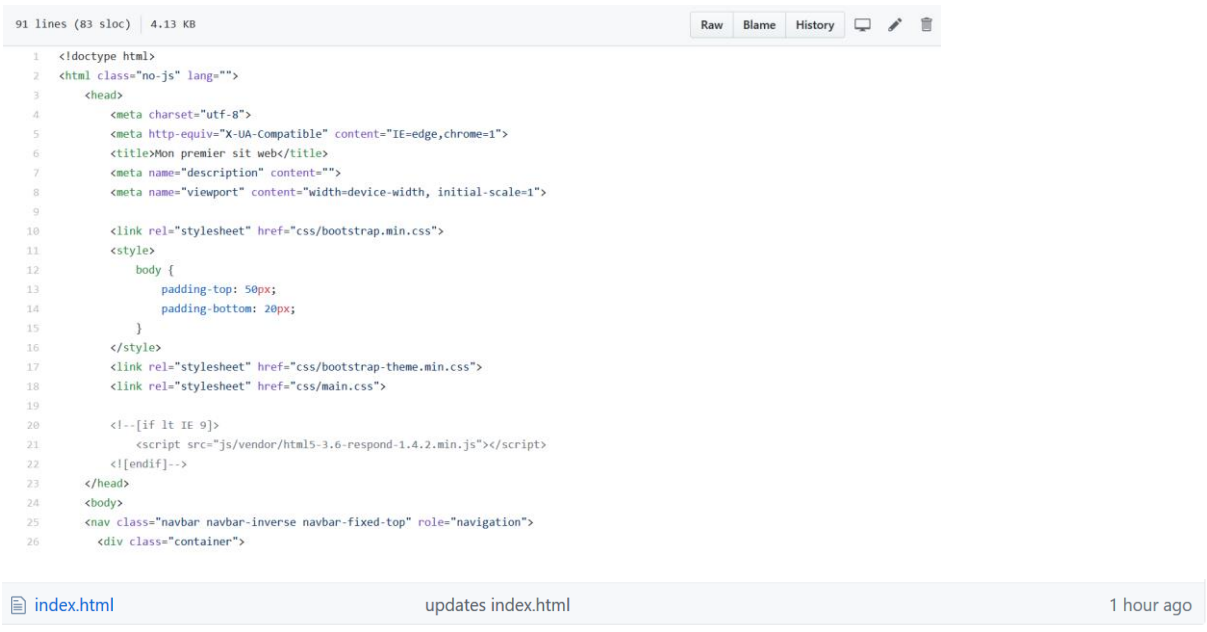
9) Vérifiez l'existence du site local sur votre repo github



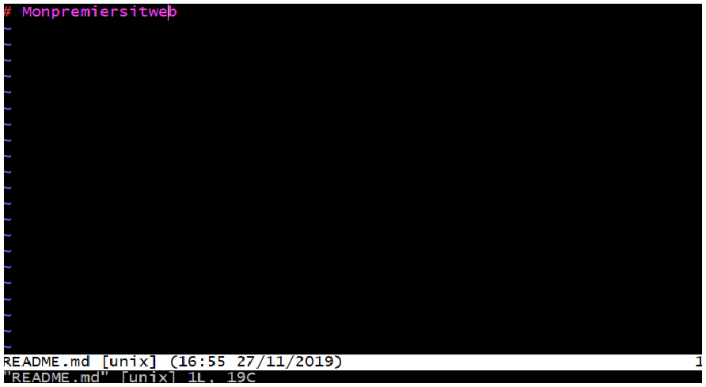
#7:and pull :

Sur github éditez le fichier **Index.html**, sur la balise <title> </title> ajoutez le titre , mon premier site web .

Faites le commit sur github



Sur git et sur le repo local , Editez le fichier README.md



Faites le stagging et le commit en une seule ligne

```
Youcode@DESKTOP-MJF171K MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git commit -am 'change README.md'
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory
[master 1039376] change README.md
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Exécutez la commande « git fetch »

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/SaraMarhoum/Monsiteweb
   05a1183..31981c0  master     -> origin/master
```

Git status :

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
```

Git pull pour puller et merger

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git pull
Merge made by the 'recursive' strategy.
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Git push

```
Youcode@DESKTOP-TLLL2MM MINGW64 ~/Desktop/projects/Monsiteweb-local (master)
$ git push origin master
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 590 bytes | 98.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/SaraMarhoum/Monsiteweb.git
   31981c0..ae35257  master -> master
```

Scénario 3 :

Dans le scénario 3 on a découvrir les deux commandes principales git rebase et git merge en effet :

Le git rebase:

Dans Git, la commande rebase intègre les changements d'une branche à une autre. C'est une alternative à la commande plus connue de "fusion".

De manière la plus visible, rebase diffère de la fusion en écrivant l'historique de validation afin de produire une succession linéaire et linéaire de validations.

Le git merge :

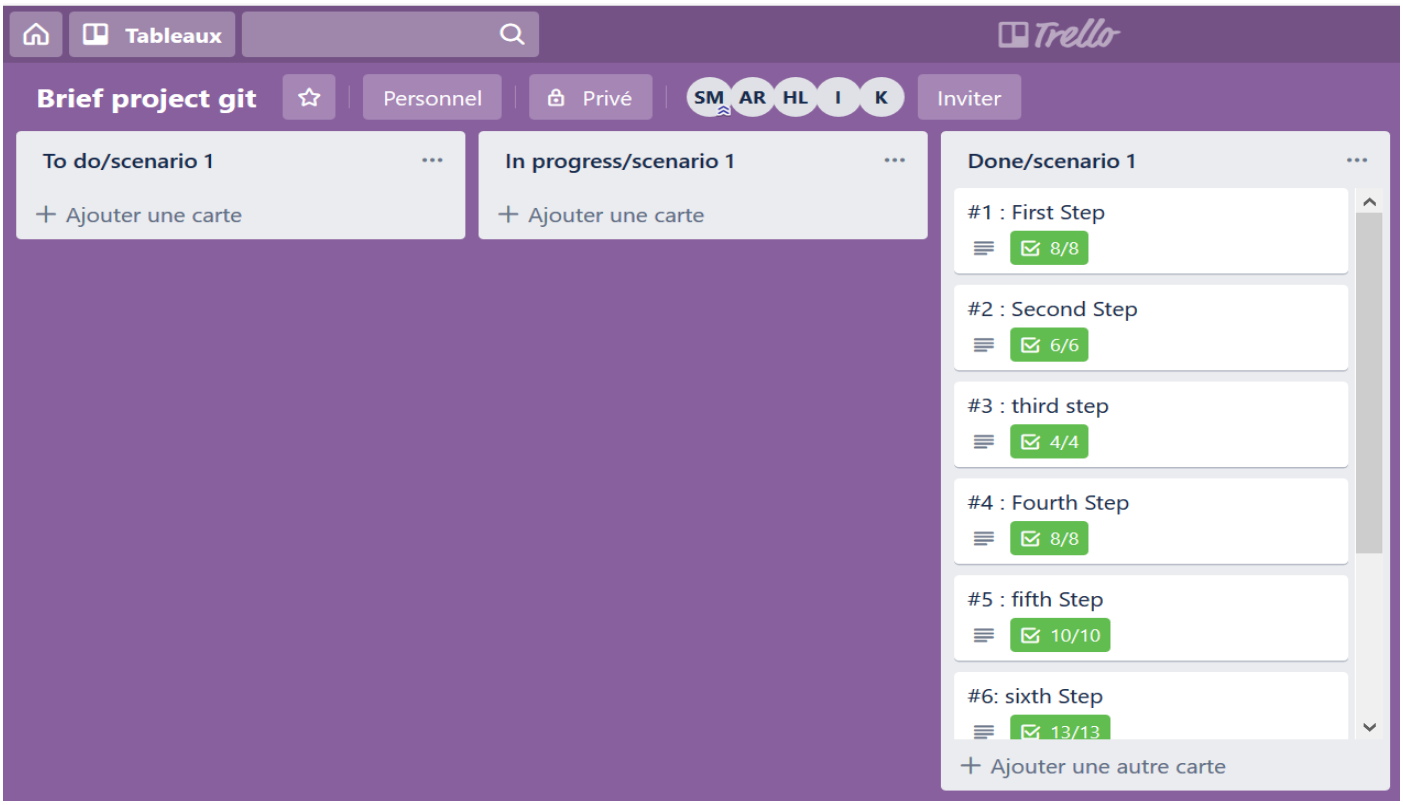
La commande "fusion" permet d'intégrer les modifications d'une autre branche.
La cible de cette intégration (c'est-à-dire la branche qui reçoit les modifications) est toujours la branche HEAD actuellement extraite.
Bien que Git puisse effectuer la plupart des intégrations automatiquement, certaines modifications entraîneront des conflits qui devront être résolus par l'utilisateur. En savoir plus sur la gestion des conflits de fusion dans notre livre en ligne.

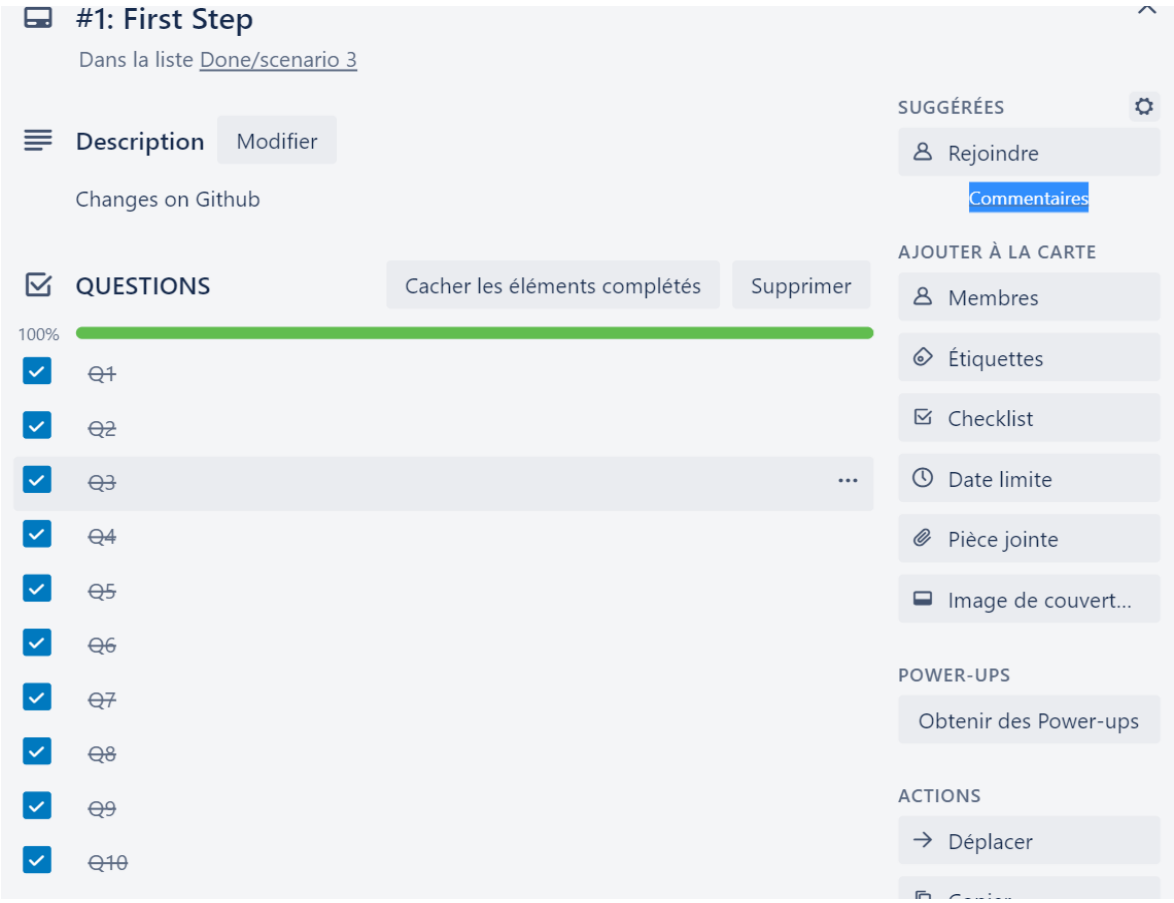
Le concepte de git rebase et git merge :

La première chose à savoir sur la commande git rebase est 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.

Songez à ce qu'il se produit si vous commencez à travailler sur une nouvelle fonctionnalité dans une branche dédiée, puis que l'un des membres de l'équipe met à jour la branche master avec de nouveaux commits. Résultat ? Vous obtenez un historique forké, un élément bien connu de tout développeur ayant déjà utilisé l'outil de collaboration Git.

Trello :


Tableaux
Trello
Brief project git
Personnel
Privé
SM AR HL I K
Inviter
To do/scenario 1
Ajouter une carte
In progress/scenario 1
Ajouter une carte
Done/scenario 1
#1 : First Step
8/8
#2 : Second Step
6/6
#3 : third step
4/4
#4 : Fourth Step
8/8
#5 : fifth Step
10/10
#6: sixth Step
13/13
Ajouter une autre carte


#1: First Step
Dans la liste Done/scenario 3
Description
Modifier
Changes on Github
QUESTIONS
Cacher les éléments complétés
Supprimer
100%
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
SUGGÉRÉES
Rejoindre
Commentaires
AJOUTER À LA CARTE
Membres
Étiquettes
Checklist
Date limite
Pièce jointe
Image de couvert...
POWER-UPS
Obtenir des Power-ups
ACTIONS
Déplacer

Conclusion :

Ce brief projet m’aura permis de découvrir tout les outils de git ainsi que dans cette expérience de develoope mon esprit d’équipe puis la gestion du temps et des conflits .