*RAPORT D’ACTIVITER DES SCENARIOS*

*GIT /GITHUB WORFLOW*

***(Réalise par othmane qorqach)***

*Scénario #1*

# ***Objectifs :***

* La formation d’un groupe de 5 apprenants
* Planification des taches selon une méthode de gestion de projet (SCRUM).
* La finalisation des Scénario dans les délais.
* La rédaction d’un rapport d’activités du groupe.
* La manipulation des commandes GIT.
* La découverte du monde GITHUB
* La gestion du temps, du stress et la finalisation des livrables.
* Les réunions de fin de Scénario
* La formalisation

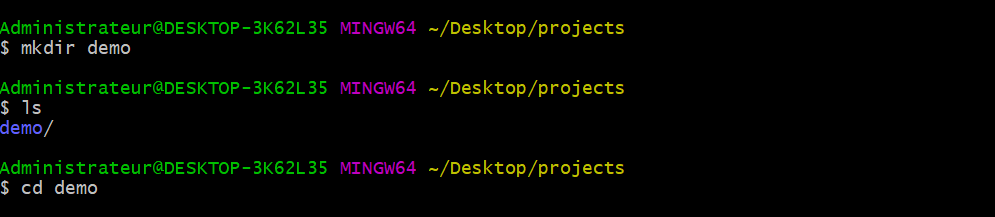
First Step : Immersion

**\_ Git :** Git est un système open source de gestion de versions distribué. Il a été conçu par Linux Torvald, le créateur du noyau Linux, pour gérer les sources de ce noyau que nous aimons tant. Git a été donc pensé dès le départ pour être efficace et rapide.

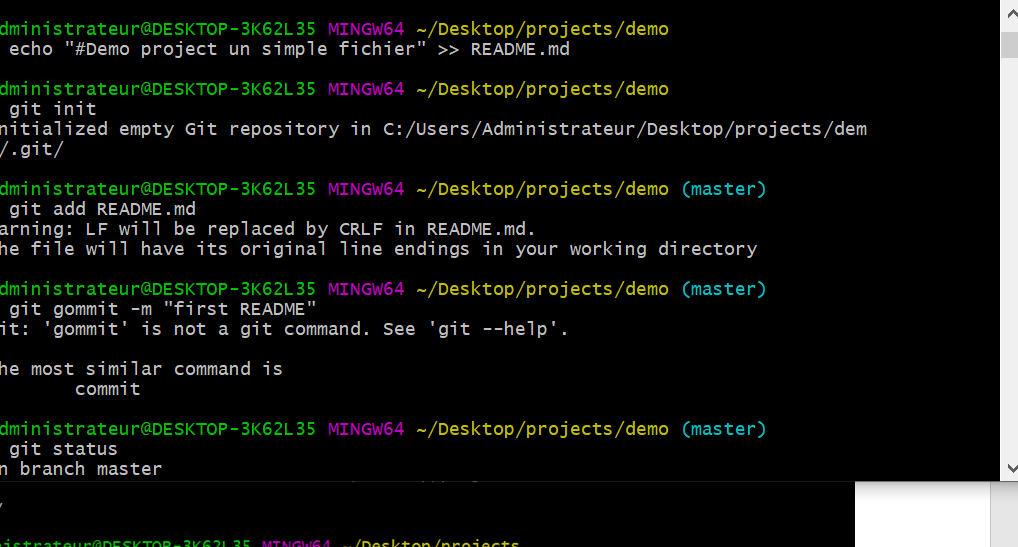
**\_** Le dossier **.git** contient toutes les informations nécessaires à votre projet dans le contrôle de version, ainsi que toutes les informations sur les validations, l'adresse du référentiel distant, etc. Toutes sont présentes dans ce dossier. Il contient également un journal qui stocke votre historique de validation afin que vous puissiez revenir à l'historique

**\_** créez un répertoire nommée « **/projects** » sur bureau

**\_** Créez un Repository Local

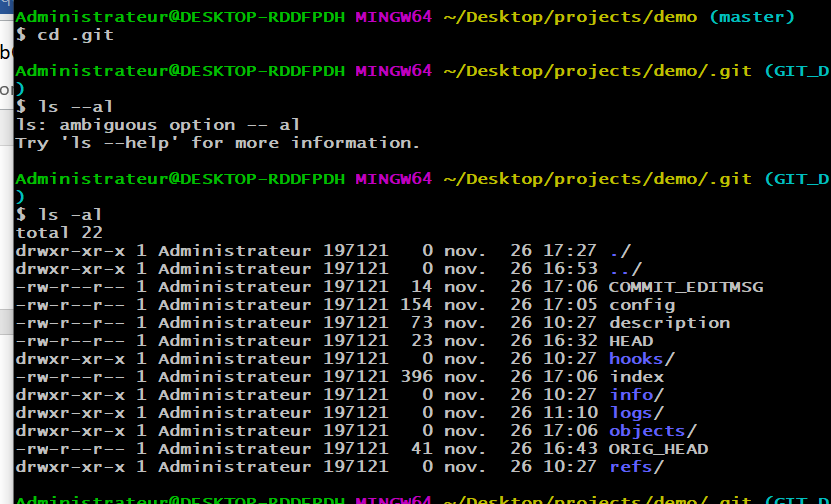


**\_** Créez le fichier README.md et ajouter la ligne suivante : ‘#Demo project un simple fichier’.Faites le stagging et le commiting avec un commentaire.

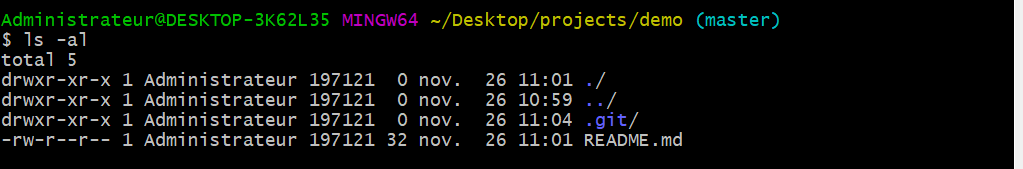


2 : Second Step : La découverte

\_ configuration « .**git** »



\_Tapez Ls –al

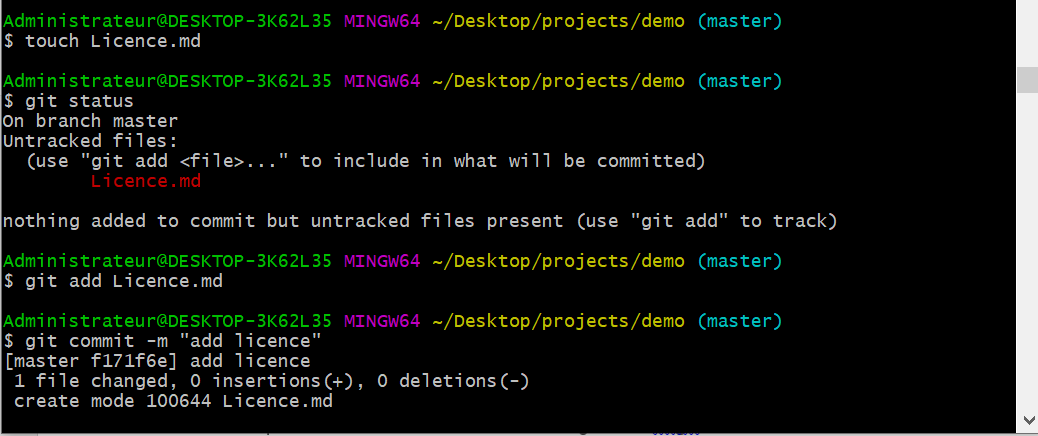


**Les branches** permettent d'avoir simultanément plusieurs versions de votre programme dans votre dépôt Git. C'est très utile, par exemple pour développer une nouvelle fonctionnalité, tout en gardant la branche principale intacte. Ainsi, vous pouvez toujours faire des changements dans la branche principale (corrections de bugs par exemple), tout en développant en parallèle une nouvelle fonctionnalité.

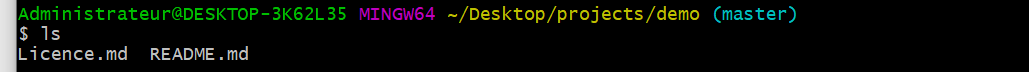
**HEAD** est une référence au dernier commit de la branche de check-out en cours

**Log** : git log ( affiche les commits log effectuées dans un projet)

\_ créez le fichier « **Licence.md »**. et Faites-le Commit

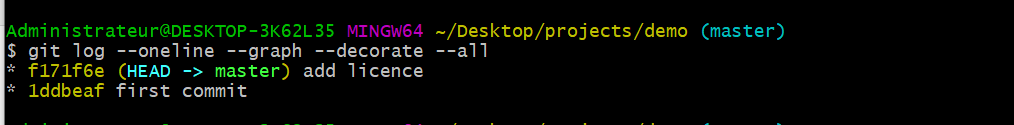


**\_** Affichez les fichiers traqués.

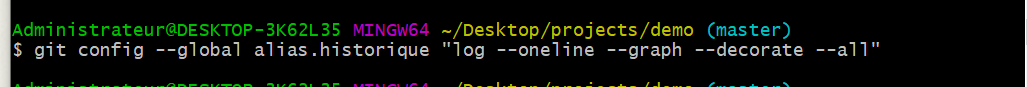


1. : third step : Historique

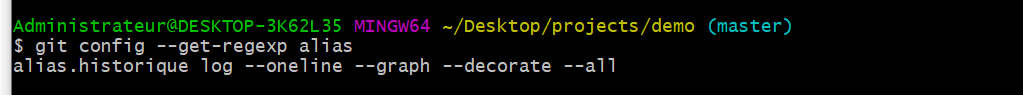
**\_** Affichez le dernier commit sur une ligne en ajoutant l’option d’affichage de la hiérarchie de la branche, avec les commit et leur branche aussi



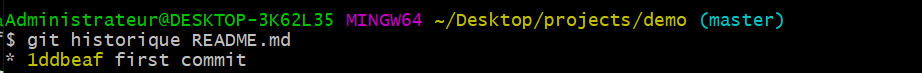
**\_** Créez un alias de la commande précédente le nom de l’alias est : historique



**\_** Affichez la liste des alias

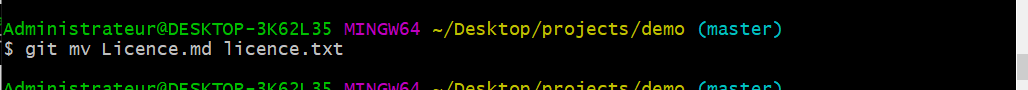


**\_** Affichez l’historique des commit du fichier README.md avec l’alias.

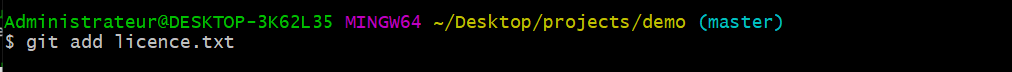


1. : Fourth Step : Excluding files

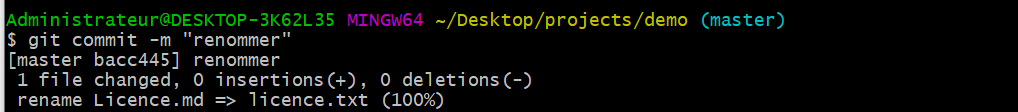
**\_**  Renommer le fichier Licence.md à Licence.txt



**\_** Faites le stagging avec mise à jour (ne pas faire « **git add .** »



**\_** Faites-le commit.



**\_**  Créez un fichier nommé application.log (touch application.log )

\_ Ne faites pas le stagging mais créez un fichier nommé « **.gitignore » (**touch .gitignore )

\_ Sur le fichier **.gitignore** Ajoutez la ligne suivante « \*.log » (code .)

\_ Faites le stagging et le commit

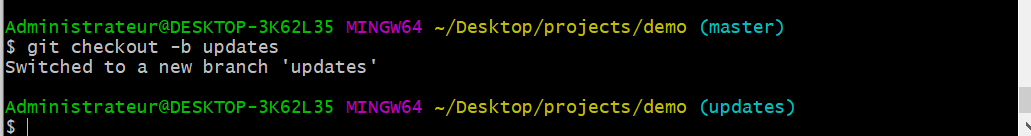
****

\_ **.gitignore**ignorer les fichiers \*.log

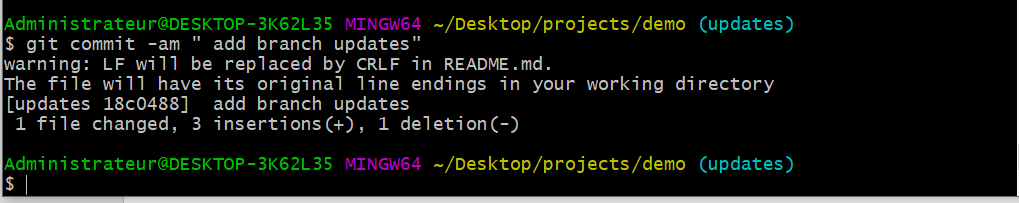
1. : fifth Step : Branching and Merging

\_Modifiez le fichier README.md ( code . )

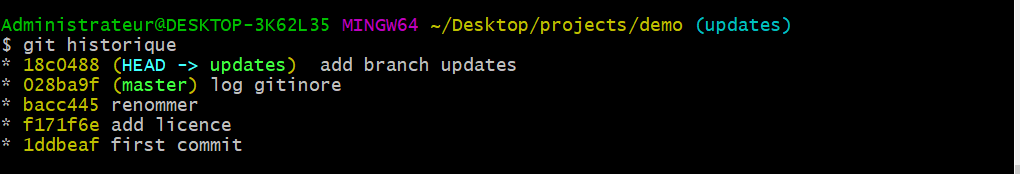
\_ Créez une branche pour la modification du fichier README.md du nom ‘updates’



\_ Faites le stagging et le commit en une seule ligne



\_ Affichez l’historique avec l’alias ( git historique ) .



**\_** les modifications ont été apportées au branche Updates et HEAD devenue en branche Updates.

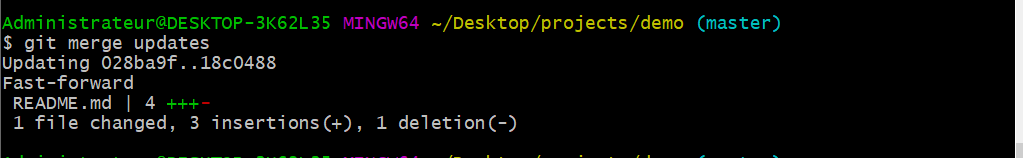
\_ Retournez vers la branche Master (git checkout Master )

\_ Affichez l’historique avec l’alias



\_ une fois retourné à un branche Master et tapé git historique nous trouvons HAED est devenue un branche Master, pas en branche Updates.

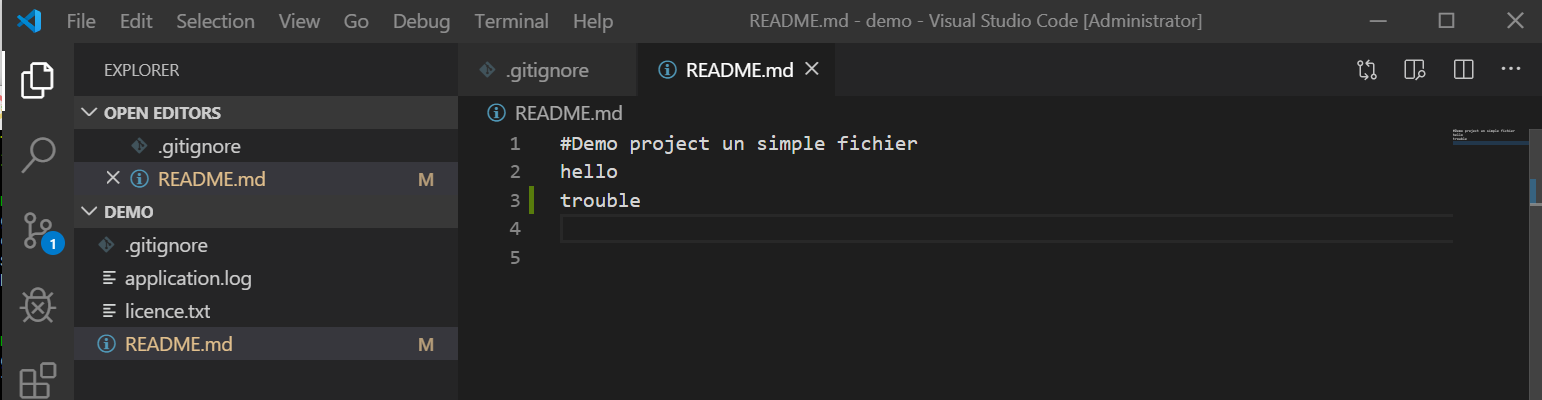
\_ faites le merge



6: sixth Step : Conflict Resolution

\_ Créez une branche avec le nom ‘BAD’ ( git checkout –b )

\_ Modifiez le fichier README.md et ajouter la ligne ‘Trouble’ ( code . )



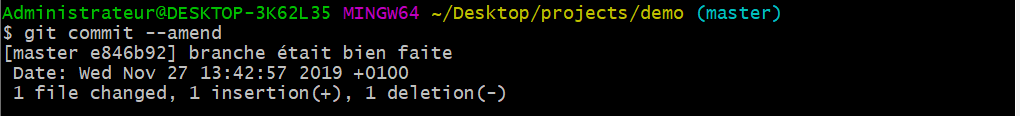
\_ Faites le stagging et le commit en une seule ligne ( git commit –am ‘’….’’ )

\_retournez la branche Master (git checkout master)

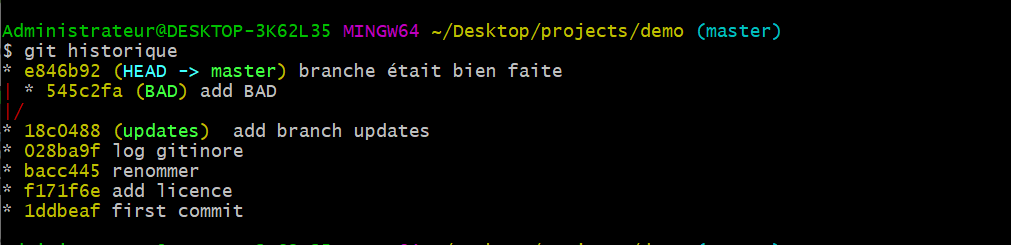
\_modifier le fichier README.md et ajouter la ligne ’Troubleshooting’ avec commande (code .)

\_ Stagging/commiting avec commentaire ‘branche bien faite’ ( git commit –am branche bien faite ’ )

\_ corriger le commentaire du dernier commit

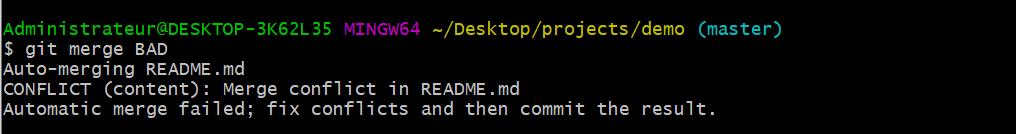


\_ affichez historique



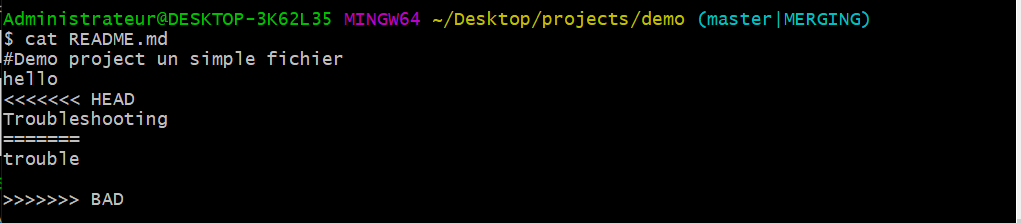
\_ nous avons trouvé un conflit, parce que n’est pas fusionner (merge)

\_ Faites le merge de la branche ‘BAD’ ( git merge BAD )



\_ conflit n’est pas résoudre car il existe deux modification au même fichier, il doit être discuter avec nos équipe d’accord à seul modification.

\_ la commande suivante : cat README.md



La commande cat affichée le contenu de fichier.

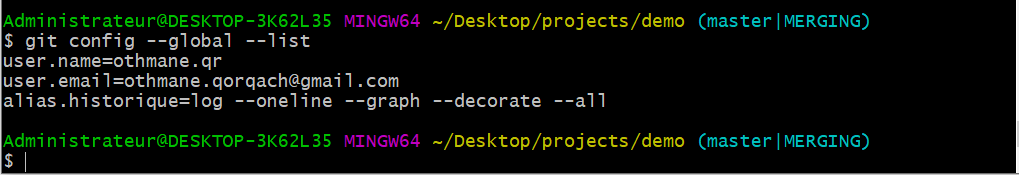
\_ la commande « git mergetool »



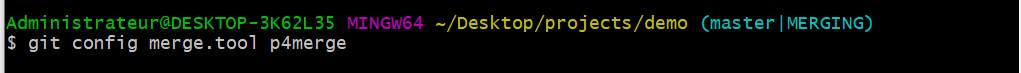
Il faut installer un logiciel p4merge pour le GIT accepter la commande git mergetool.

7: seventh Step: merge tools

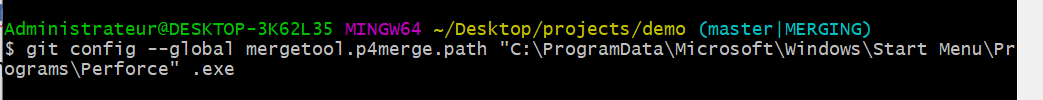
\_ Installation de p4merge sur notre pc

\_ tapez la commande git config – global --list

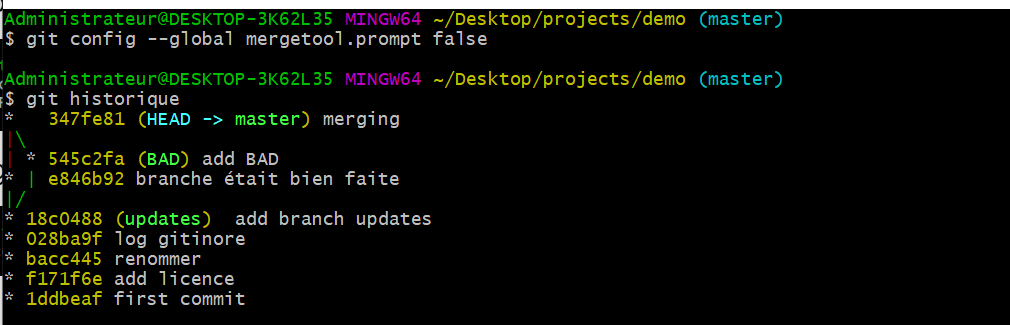
\_ Git config –global merge.tool p4merge



\_ Git config –global mergetool.p4merge.path “lien d’installation”

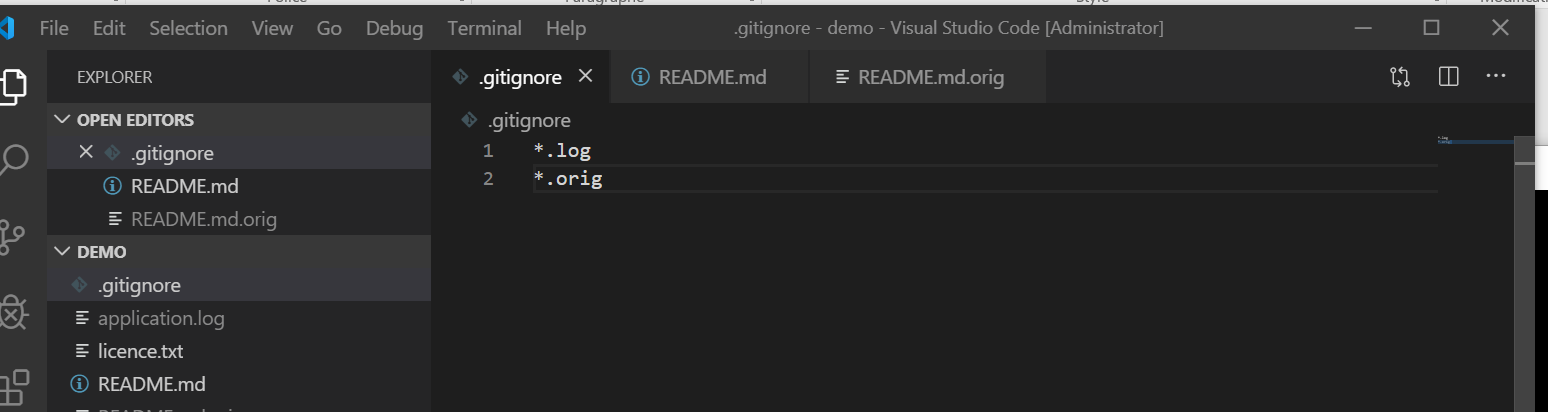


\_ configurer le prompt



8: eigth Step: Challenge

\_ Sur le fichier **.gitignore** ; écrivez une clause pour rejeter les fichiers indésirables et (‘



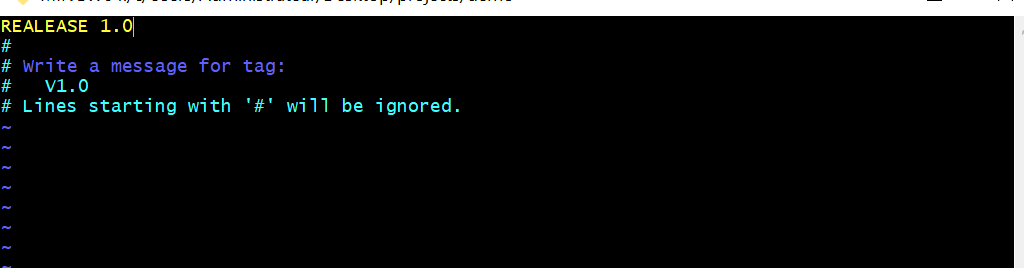
On laisse que les fichiers : **licence.txt Readme.md .gitignore**

*Scénario #2*

1 : First Step : Tagging

\_ Créez un TAG avec un nom V1.0 et un commentaire ‘ REALEASE 1.0

C:\Users\Administrateur\Downloads\-1.png

**

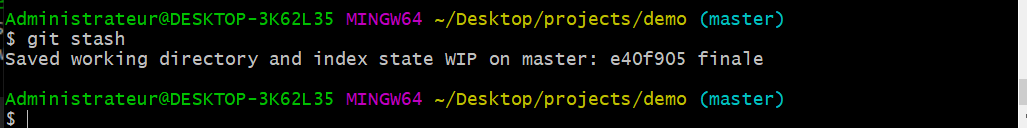
\_ Affichez les informations sur le TAG



1. : Second Step : Stashing and Saving work in Progress

\_ Modifiez le fichier README.md , ajoutez une ligne ( code .)

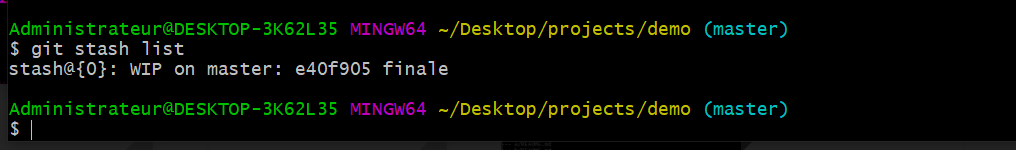
\_ la commande git Stash :

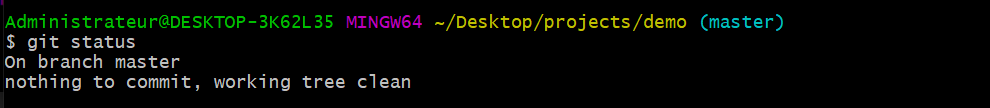


Git Stash:

Nous utilisons git stash pour réserver notre changement dans une autre directory

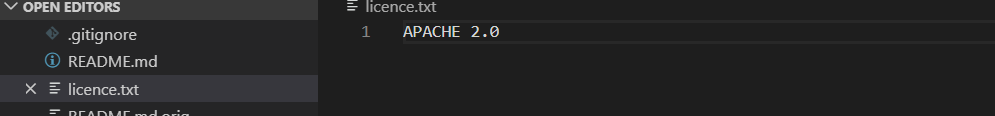
\_ Tapez la commande git stash list ( ls )



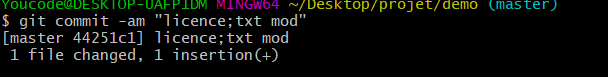
\_ Exécutez la commande git status 

\_ Nous constatons que la modification qu’on a fait est pas dans notre «working directory »

\_ Modifiez le fichier « Licence.txt », ajoutez la ligne « APACHE 2.0 » ( code .)



\_ Faites stagging et le commit en une seule ligne



\_ Exécutez la commande « git stash pop »



\_ Afficher les changements faits dans l’autre directory (git stash)

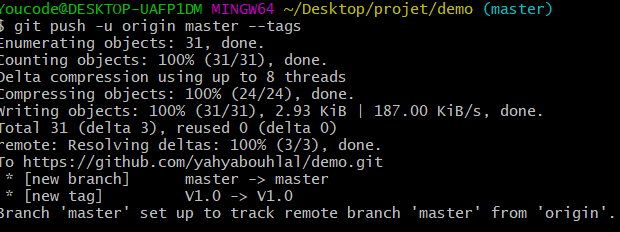
3: third step: Voyage sur Github, Local Repo to github Repo

\_ Créez un repo github public sans ajouter le fichier README.md (github.com )

\_ Examinez le remote

C:\Users\Administrateur\Downloads\-10.png

\_ Pushez le tous à travers la commande : git push –u origin master - -tags



\_ expliquez les options –u et –tag ?

**U :** cette option sert à configurer une relation de branche de suivi . pour chaque branche mise à jour ou poussée avec succès, ajoutez une référence en amont (suivi).

**--tag :** envoyer tous les tags que nous avons à github

4 : Fourth Step : Mini challenge ( optionnel)

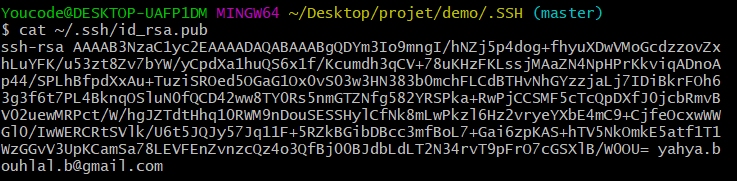
\_ Créez un fichier sous git nommé **.SSH ( mkdir .SSH )**

\_Déplacez-vous dans le fichier .SSH (cd . SSH)

**C:\Users\Administrateur\Downloads\-13 (1).png**

Créez une authentification sécurisé SSH entre votre repo local et votre repo distant

****

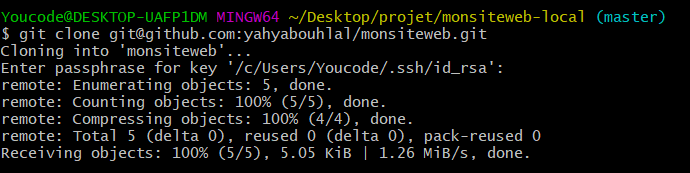
****

5: Fifth Step: Création d’une local copy

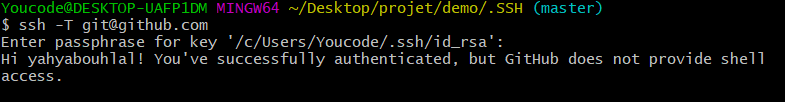
\_ Sur github créez un autre repo nommé ( Monsiteweb) (github.com)

\_ Ajoutez à l’arboresence toujours sur github le fichier **.gitignore** et un fichier **licence.txt ‘APACHE 2.0 ‘** sur (github.com)

\_ Créez un clone github vers le local sous le nom (Monsiteweb-local)



\_ Vérifiez si le clone est créé.

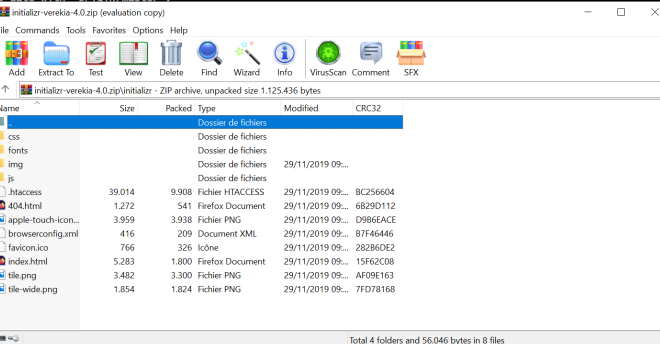


6: sixth step : Sending the website

Télécharger le site web depuis le lien suivant : <http://www.initializr.com/>

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

\_ Analysez l’arboresence.



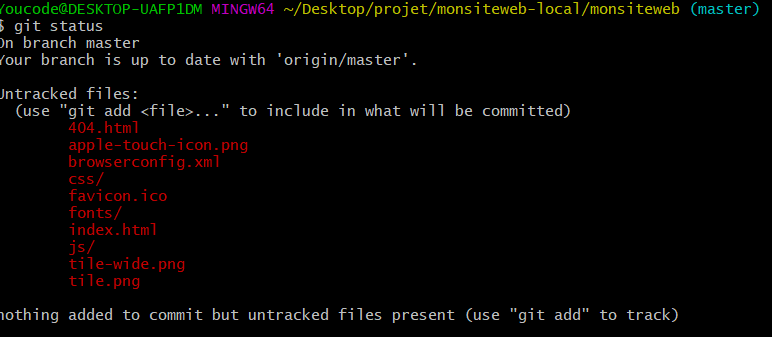
l’arborescence :

c’est le contenu du site web (css ,fonts,img,js…)

\_ Copiez le site télécharger dans votre repo local

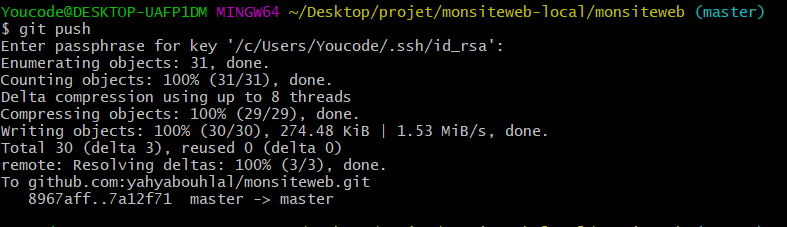
C:\Users\Administrateur\Downloads\-18.png

\_ Git status



\_ Faites le stagging et le commit en une seule ligne ( git add –a && git commit -am )

\_ Faite le push à github



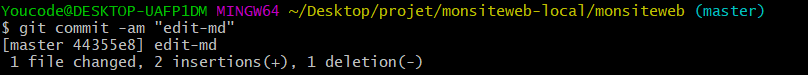
\_ Vérifiez l’existence du site local sur votre repo github

7: seventh step : Fetch and pull

\_ Sur github éditez le fichier **Index.html**, sur la balise <title> </title> ajoutez le titre , mon premier site web,et Faites le commit sur github (github.com)

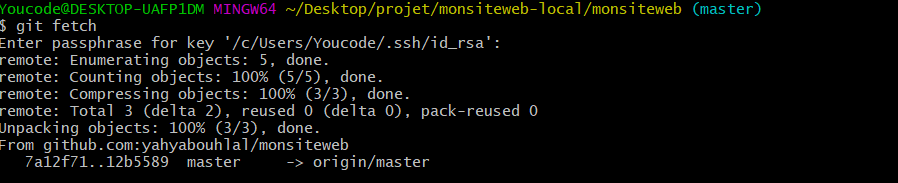
\_ Sur git et sur le repo local , Editez le fichier README.md (code .)

\_ Faites le stagging et le commit en une seule ligne

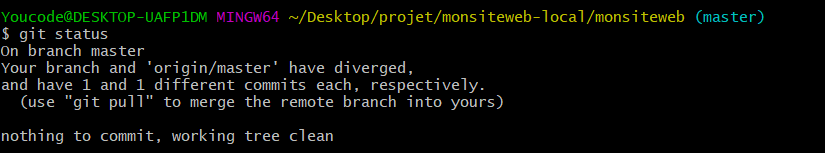


les modification en add et commit dans une seul ligne parceque on deja dans locale repot

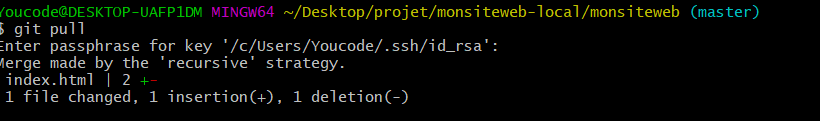
\_ Executez la commande « git fetch »

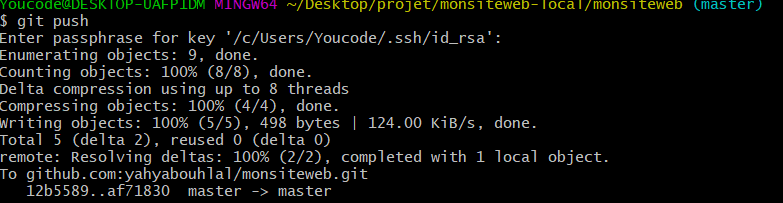


\_ Git status



\_ Git pull

\_ Git push

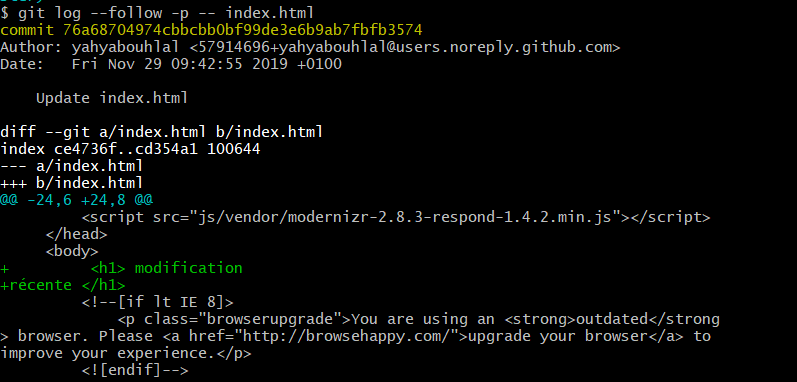


*Scénario #2*

Etape 1 : Changes on Github

* L’ajout d’un fichier nommé style.css (touch style.css)
* Modification du fichier index.html en ajoutent labalise :

(<h1>modification récente </h1>)

* Renommer le fichier 404.html en error404.html.
* La synchronisation avec le repo local
* Vérification de la liste du commit.
* Sur GITBASH, vérifiez avec « git status »
* Visualisez et télécharger les fichiers distants sur GITBASH.
* Faites le pull et fusionnez les changements distants avec le repo local. (git pull )(git merge).
* Affichez les informations du commit de l’ajout de la balise <h1> :

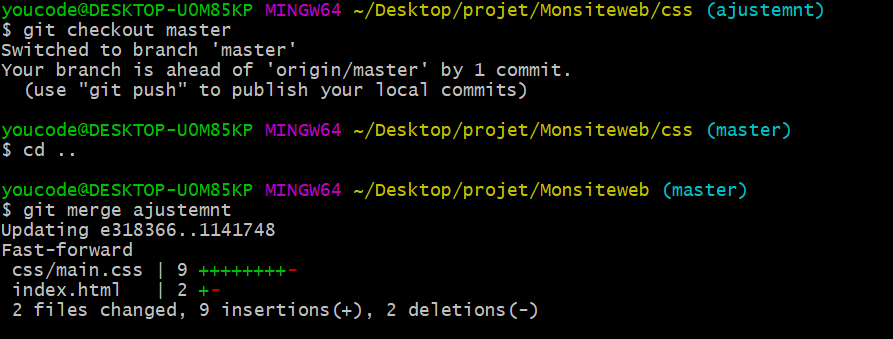
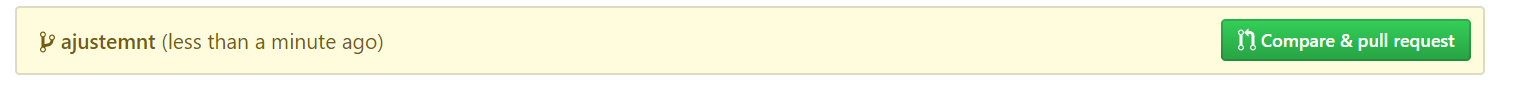
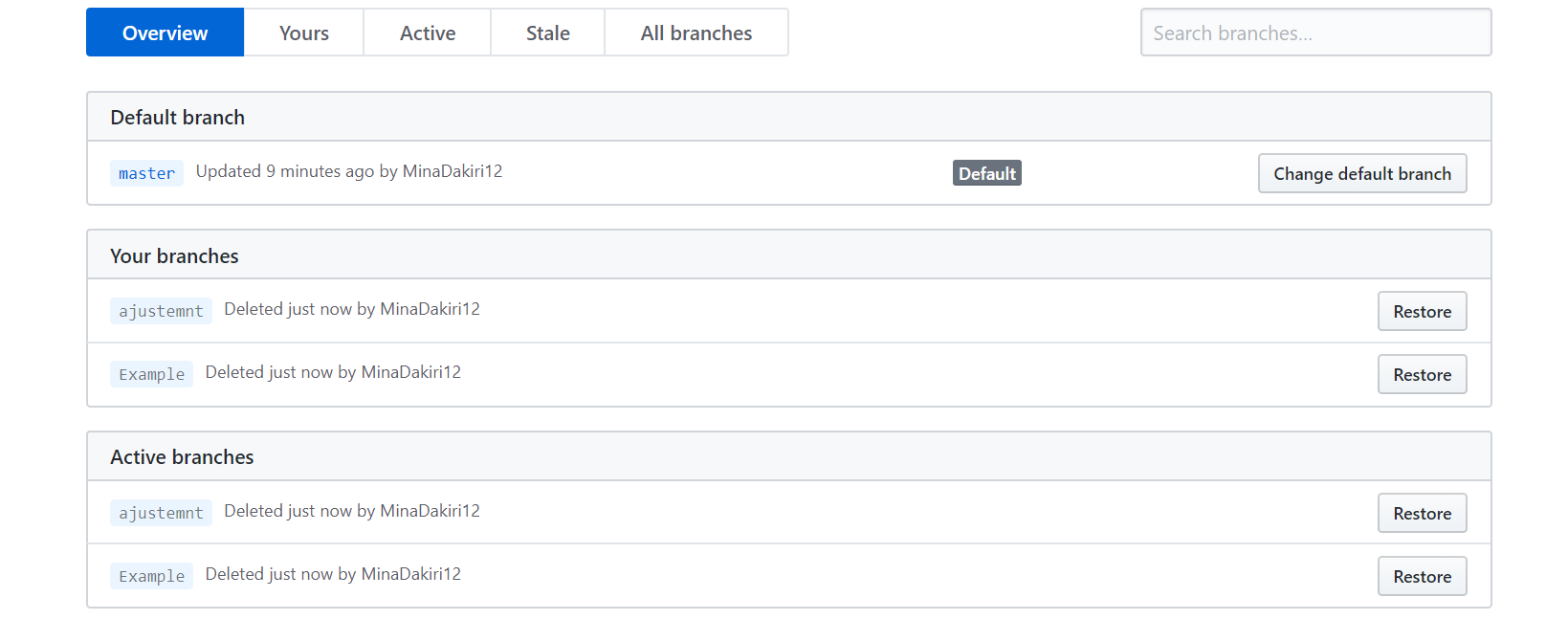
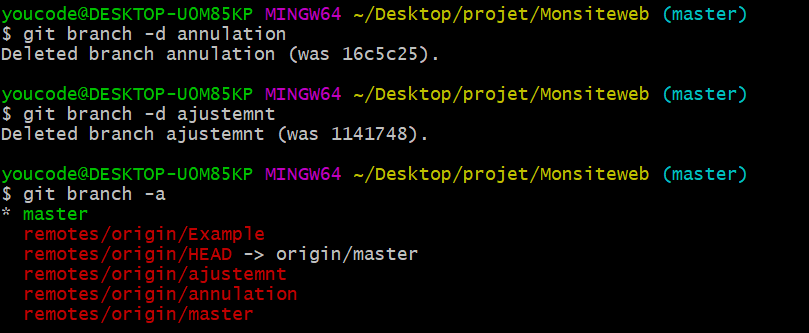
Etape 2 : Branching and merging sur GITHUB

* La création du branch ‘exemple’ sur github.
* La modification du fichier README.md sur github Tjr.
* Dans le menu branch sur github on a un message du nouvelle branch ‘exemple’ pour faire un pull requeste.
* La création du branch ‘annulation’ sur repo local.
* Suppression du balise H1 dans index.html tjr sir repo local.
* Le commit et le stagging en une seule ligne.
* « Git show » pour vérifier vos changements.
* Pushez les changements qu’on a sur la branch annulation vers le repo distant.
* Nous constatons un message indiquant qu’une branche a été ajoutez avec un bouton « compare and pull request ».

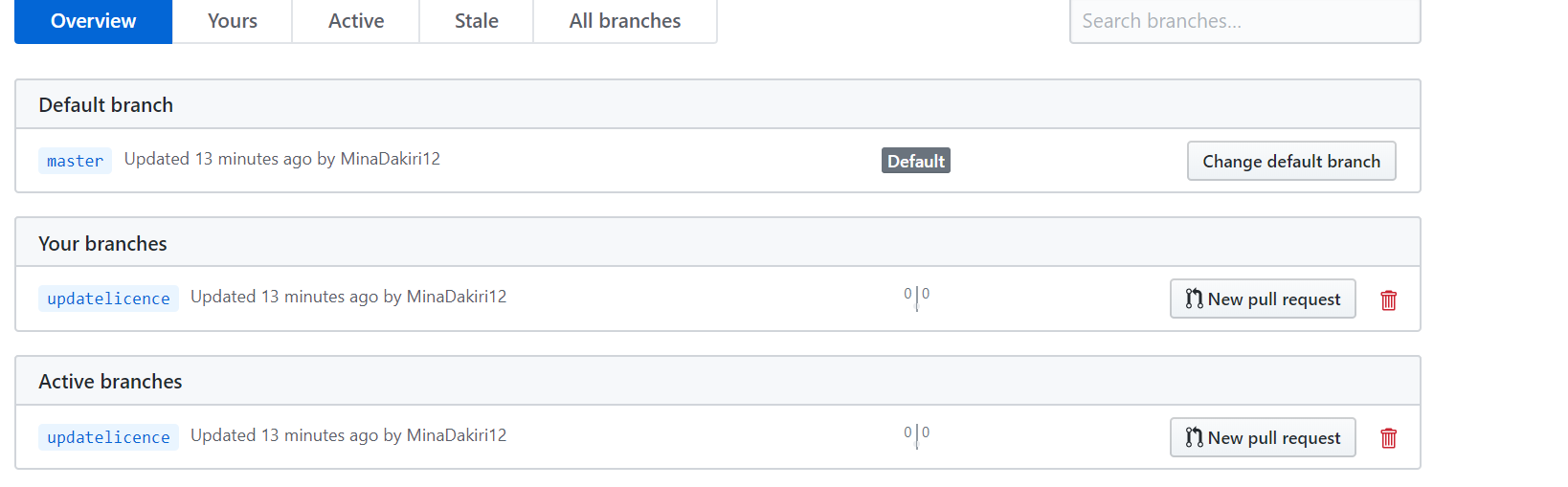
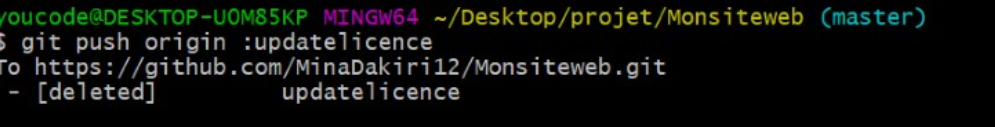
Etape 3 : compare pull Requests

* La création d’un pull request avec le commentaire ‘annulation à vérifier’.
* Une page s’ouvre indique que la branch able to merge.
* Effectuer le merge.
* La suppression du branch annulation .

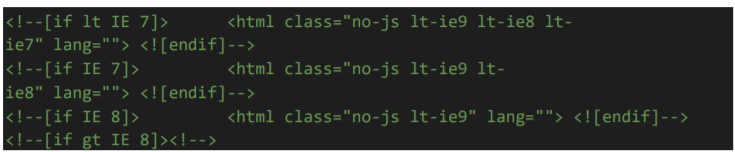
Etape 4 : merging en local

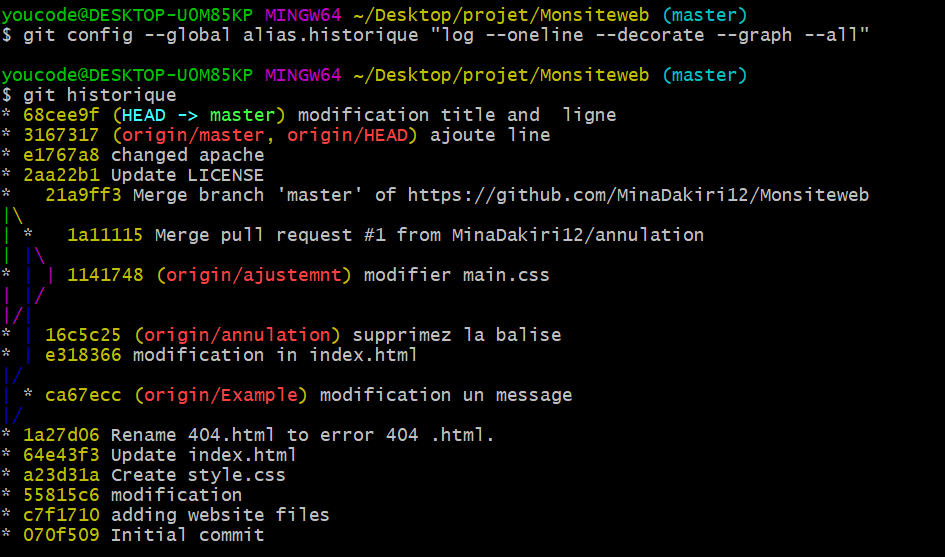
* La création du branch ajustement et lamodification du fichier main.css.
* Puchze les modifications sur Github.
* Revenez vers la branche master sur gitbash.
* Demande de pull (git pull).
* Type de merge on aura besoin pour faire un merge sans conflits : Fast-forward .
* Faites le merge à travers GITBASH :
* 
* Pushez les changements vers github.
* Explorez le pull pour visualiser les changements :
* 
* Suppression des branchs qu’on a créé sur github :
* 
* Apporter les changements qu’on a fait sur GITHUB.
* Tapez la commande git branch –a : les branchs qu’on a supprimé sur github est resté dans le remote :
* 
* La suppression des branchs qu’on a créé sur gitbash.
* Nous constatons que les branchs supprimer reste dans le remote.

Etape 5 : The Cleaning up

* Sur GITHUB Créez une branche nommée ‘updatelicence, éditez le fichier Licence et changer Apache 2.0 par Apache 3.0 :
* 
* Sur GITBASH faites un pull global : git pull –-all.
* Maintenant faites le merge de la branche ‘updatelicence’.
* Exécutez Git push :
* 
* Supprimer la branch : Git branch –d ‘updatelicence’.
* Exécutez la commande git branch –a : nous constatons que la branch qu’on a supprimé reste dans le remote.
* Trouvez la commande qui va nous permettre de supprimer la branche depuis le remote et ainsi de pushez les changements de la suppression simultanément : (git push origin –delete).

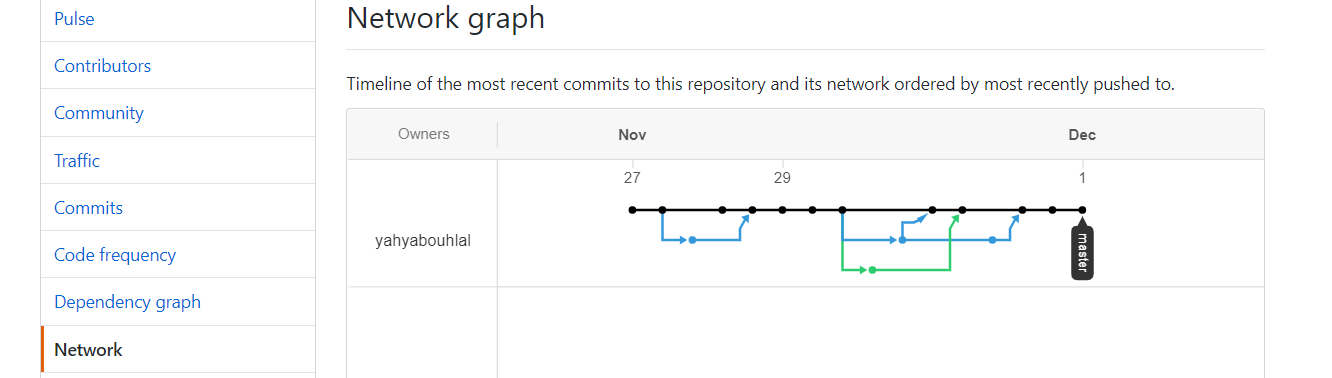
Etape 6 : Rebasing

* Editez le fichier README.md, ajoutez la ligne ‘updates’ et faite le commit.
* Sur GITBASH Editez le fichier index.html supprimez des lignes et modifier le titre du site. 
* Faites le stagging et le commit en une seule ligne.
* Maintenant apportez les changements faites sur le Repo distant : fetch.
* On a besoin de faire un rebase : git pull - - rebase.
*  Exécutez l’alias que vous avez créé dans le scénario #1 (historique) :

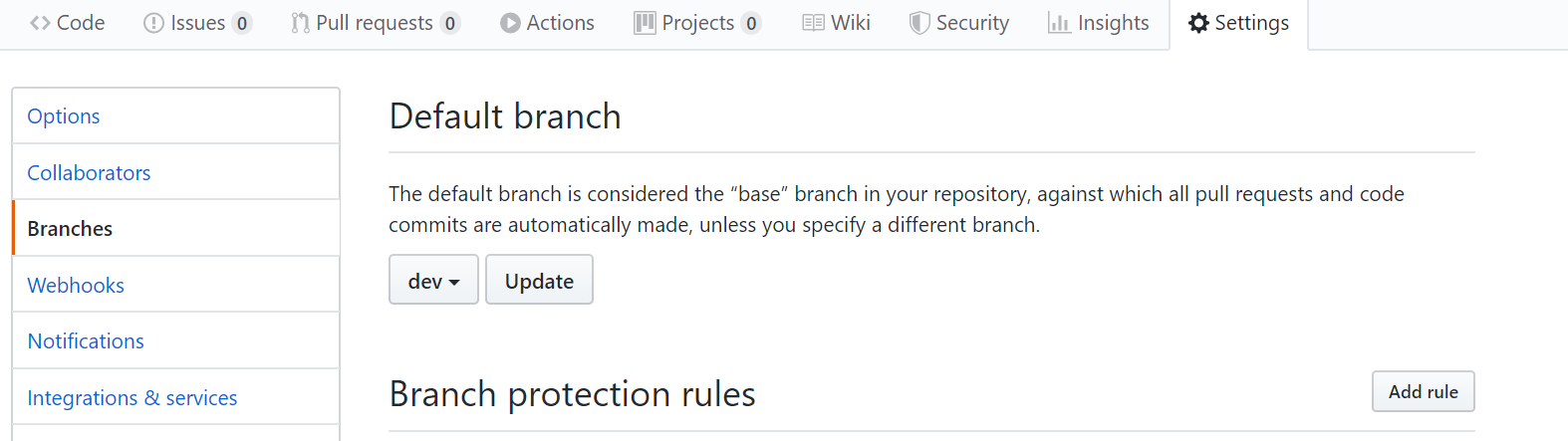
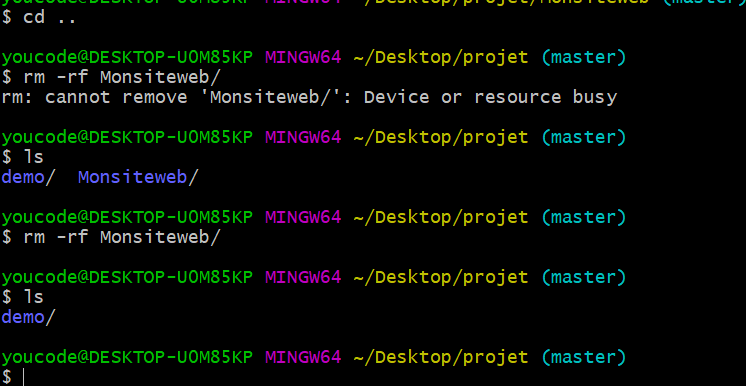
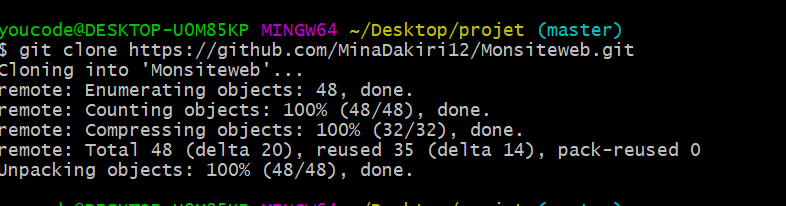
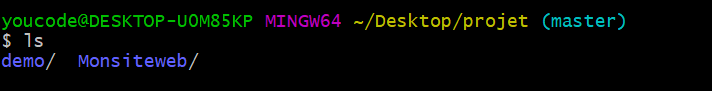


* Rebase permet de déplacer une branche et de changer son commit de départ (sa base).

Etape 7 : GitHub Insights

* Analysez le Network ou le timeline de votre workflow : 

Etape 8 : Default branch and conflicts

* Création d’une branch nommé ‘DIV’ sur github.
* La Branche Dev deviendra la branche par défaut ; sur le menu « settings » mettez la branche Dev en mode par défaut. 
* Créez une autre branche « demo », on va la considérer comme « feature ».
* Supprimez la branche feature depuis GITHUB.
* Sur GIT, supprimez récursivement et localement votre repo website : 
* Maintenant Clonez via https ou SSH
* Le contenu est récupéré : 
* Maintenant on a besoin de la branche master comme base : git checkout master :
* 