***Rapport***

***Brief Projet***



**Réalisé par**:

\*Elainaoui Abdelmounaim

**Scenario I**

Step 1: Immersion

Step 2: La découverte

Step 3:  Historique

Step 4:  Excluding files

Step 5: Branching and Merging

Step 6: Conflict Resolution

Step 7:  merge tools

Step 8:  Challenge

**Scenario II**

Step 1: Tagging

Step 2: Stashing and Saving work in Progress

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

Step 4:  Mini challenge

Step 5:  Creation d’une local copy

step 6:  Sending the website

step 7:  Fetch and pull

**Scenario III**

Step 1: Changes on Github

Step 2: Branching and merging sur GITHUB

Step 3: compare pull Requests

Step 4: merging en local

Step 5: The Cleaning up

Step 6: Rebasing:

Step 7: GitHub Insights:

Step 8: Default branch and conflicts

**Scenario I**

**Step 1: Immersion**

* **Définissez git.**

Git est logiciel de gestion des versions

* **Le fonctionnement du fichier caché « .git ».**

Le fichier caché «.git » indique que la répertoire créer est devenu liée avec le Git et c’est grâce à la commande « git init »

* **Sur votre bureau créez un répertoire nommée « /projects ».**
* **Créez un Repository Local sous le nom « /demo ».**
* **Vérifiez le statut du répertoire « /demo »**

Mkdir demo

Cd demo

Git init

Git status

* **Expliquez le commentaire.**

Le message affiché indique que ce repo local contient une branche master, et qu’il y’a pas des fichiers ou modification à enregistrer.

* **Créez le fichier README.md et ajouter la ligne suivante : ‘#Demo project un simple fichier’.**
* **Faites le staging et le committing avec un commentaire.**

Touch README.md

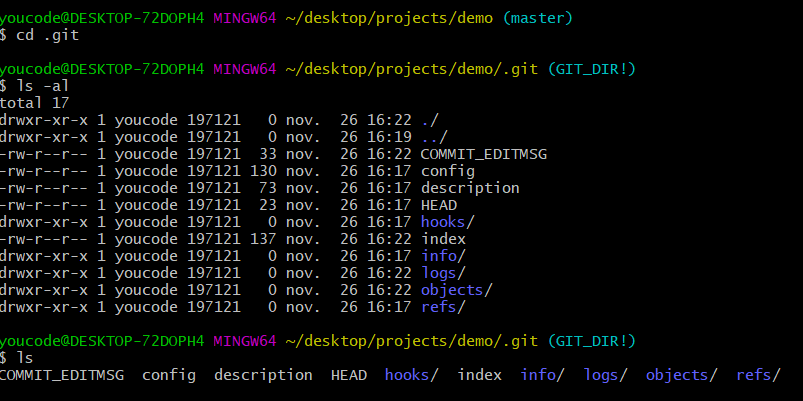
Code README.md

Git add .

Git commit –am”first commit create readme”

Git status

**Step 2: La découverte**



**● Expliquer les clauses suivantes : HEAD, LOGS, BRANCHES**

**Les branches permettent de créer une version en parallèle du code à un moment donné dans l'historique.**

**Head c’est une référence symbolique à la branche courante (pointeur).**

**Logs c’est un journal ou historique d’évènements effectués qui sont datés et classés par ordre chronologique.**

**● Retournez vers votre Repo et créez le fichier « Licence.md ».**

**● Faites-le Commit**

**● Affichez les fichiers traqués.**

**Cd ..**

**Touch Licence.md**

**Git commit –am”create file licence”**

**Git status**

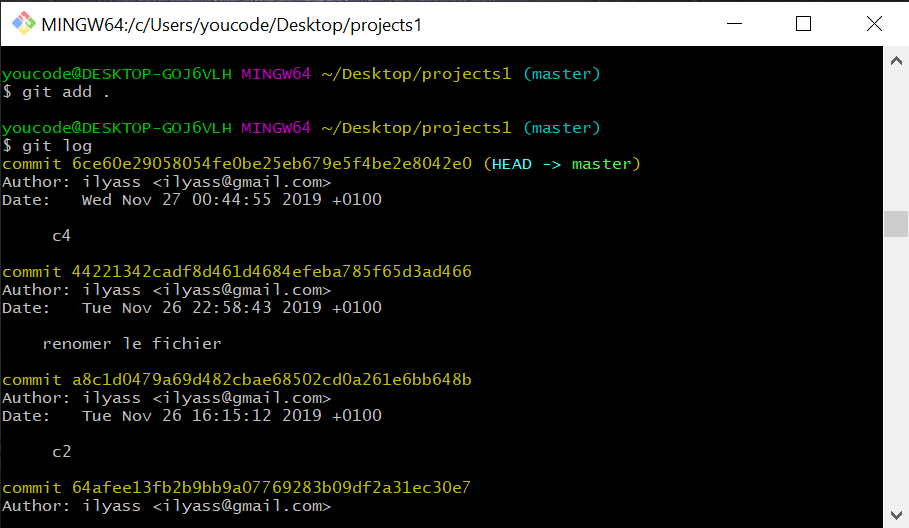
**Ls**

**Step 3: Historique**

Git log -1 –graph

Git config –global alias.historique ‘log –graph’

On constate que si un fichier mentionné sur le fichier ‘.gitignore’, il va pas être ajouté et enregistré dans une commit puis il n’appartient pas à l’arborescence.

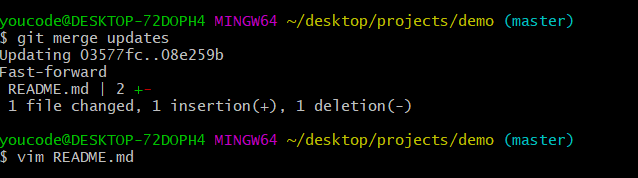


L’historique de la branche updates affiche la nouvelle commit plus les commits faites au niveau de branche master.

L’historique de la branche master n’affiche pas la commit faite sur la branche updates

**Step 5: Branching and Merging**

* **Maintenant faites le merge :**



**Step 6: Conflict Resolution :**

* **Modifiez le fichier README.md**
* **Ne faites pas le commit**
* **Créez une branche pour la modification du fichier README.md du nom ‘updates’**
* **Faites le staging et le commit en une seule ligne.**
* **Affichez l’historique avec l’alias**
* **Qu’est-ce que vous constatez ?**

L’historique de la branche updates affiche la nouvelle commit plus les commits faites au niveau de branche master.

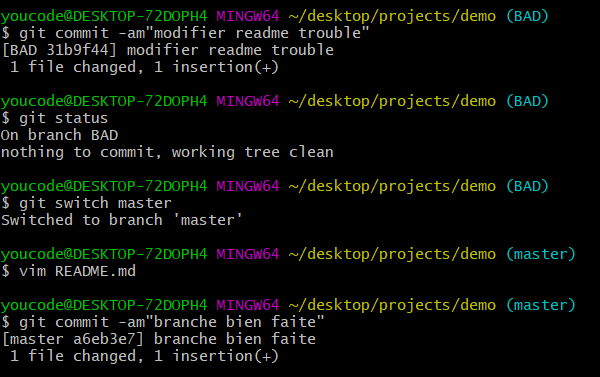
* **Retournez vers la branche Master.**
* **Affichez l’historique avec l’alias**
* **Qu’est-ce que vous constatez**

L’historique de la branche master n’affiche pas la commit faite sur la branche updates

* **Maintenant faites le merge.**

**Step 6: Conflict Resolution**

* **Créez une branche avec le nom ‘BAD’**
* **Modifiez le fichier README.md et ajouter la ligne ‘Trouble’**
* **Faites le staging et le commit en une seule ligne**
* **Switchez faire la branche principale ‘master’**
* **Maintenant modifiez le fichier README.md et ajoutez la ligne ‘Troubleshooting’**
* **Staging/committing avec commentaire ‘branche bien faite’**



* **Oups !! corriger le commentaire du dernier commit**

Git commit –amend –m’nouveau commentaire’

* **Affichez l’historique**
* **Qu’est-ce que vous constatez ? expliquez l’objectif de ce petit scénario ?**

On constate que les modifications faites sur une branche restant indépendantes par rapport master et l’historique de master n’affiche pas les commit de la branche.

* **Faites le merge de la branche ‘BAD’**

Git merge BAD

* **Expliquez le message ?**

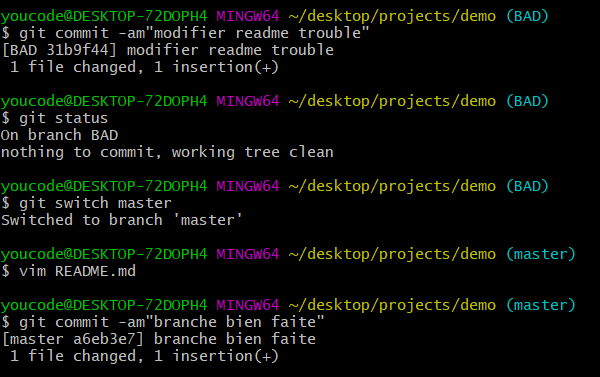
Le message indique qu’il y a un conflit grâce à des modifications sur même ligne.

* **Exécutez la commande suivante : cat README.md ? Expliquez ?**

Commande affiche deux modifications sur même ligne (trouble et troubleshooting) l’une au niveau de master et l’autre sur branch.

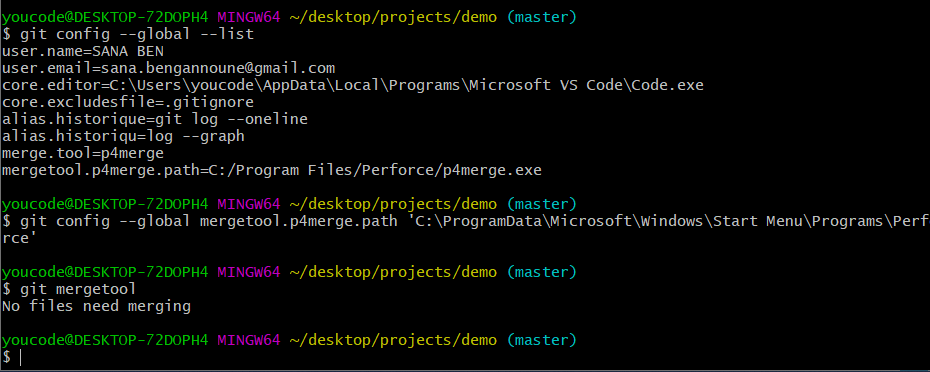
* **Tapez la commande « git mergetool » ? qu’est-ce que vous constatez ?**

On constate qu’il faut configurer et déclarer un gestionnaire de conflits avec le git.



**Step 7: Merge tools**

* **Installez p4merge sur votre PC**
* **Sur git bash tapez la commande : git config –global –list**
* **Git config –global merge.tool p4merge**
* **Git config –global mergetool.p4merge.path “lien d’installation” ( .exe)**

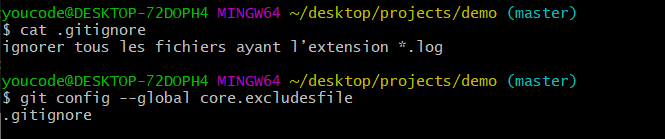


Puisque le logiciel p4merge est configuré sur le git, il va se lancer par défaut au moment de taper la commande mergetool pour afficher l’emplacement de conflits. A ce moment-là nous devons faire un choix entre les modifications comme une solution.

**Step 8 challenge :**

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

Git config –global core.excludesfile .gitignore

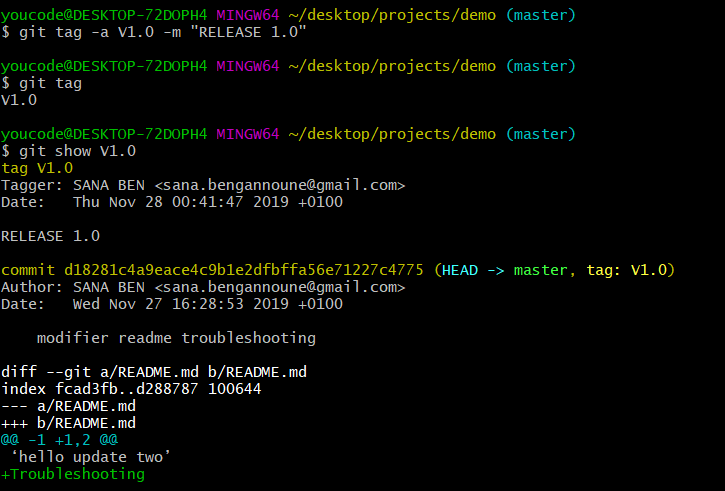
****

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

Tous les fichiers avec extension ‘.log’ sont exclues

**Scenario II**

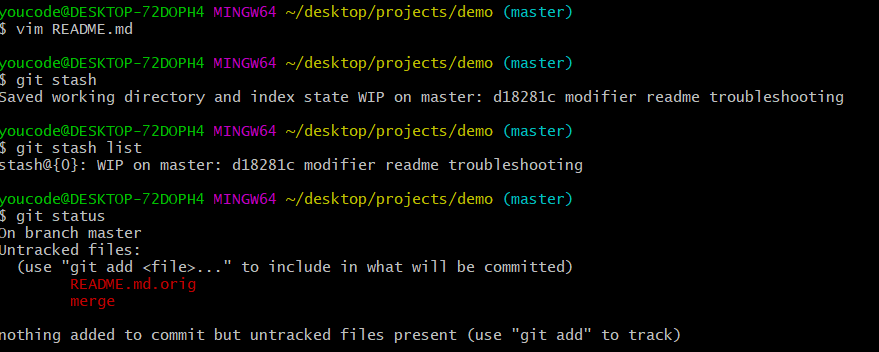
**Step 1: Tagging**



**Step 2: Stashing and Saving work in Progress**

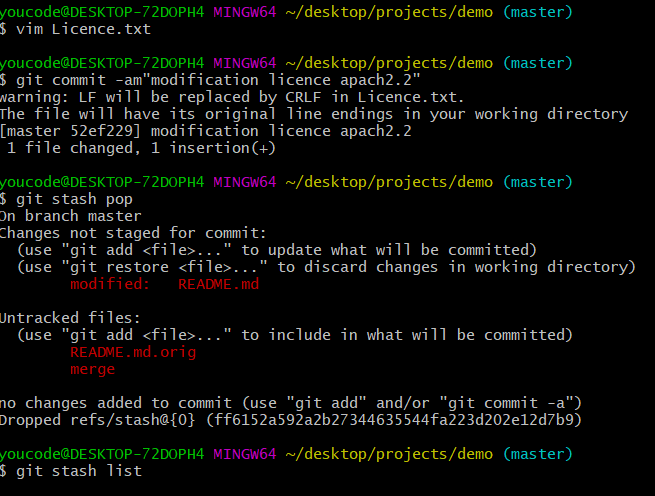
* **D’après vous un TAG il sert à faire quoi au juste ?**

Le rôle de tag est pour marquer une commit spécifique.



Git stash est une commande permettant de "mettre de côté" des modifications (commit temporaire).

On constate que les modifications mentionnés en liste stash ; ne sont pas pris comme non tracké.



**Git Stash Pop** : Les modifications en stash sont supprimées de la liste stash et ils sont disponible de les utilisés dans une commit.

**Step 3: Voyage sur Github, Local Repo to github Repo**

Git push –u origine master –tags

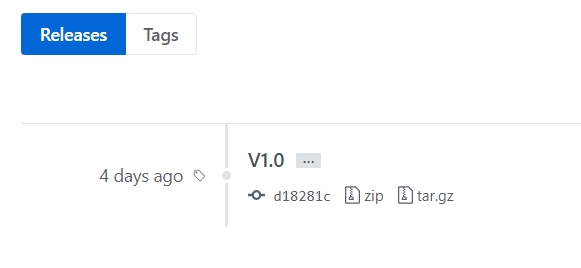
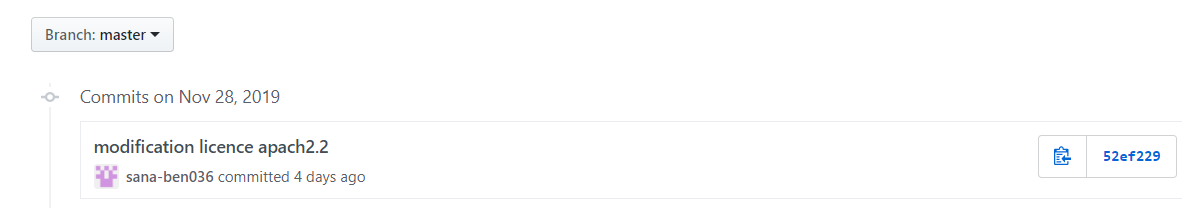
* **Expliquez la commande ?**

La commande utilisé pour transférer un dépôt local vers le github

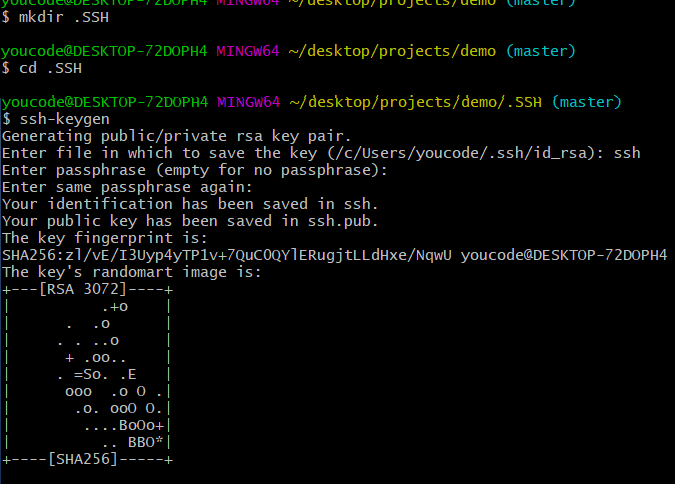
-**u** : signifie upstream c’est le dépôt Référentiel à partir duquel nous avons copié le projet.

**Tags** : ce sont des étiquètes pour marquer un certain état dans l’historique comme important

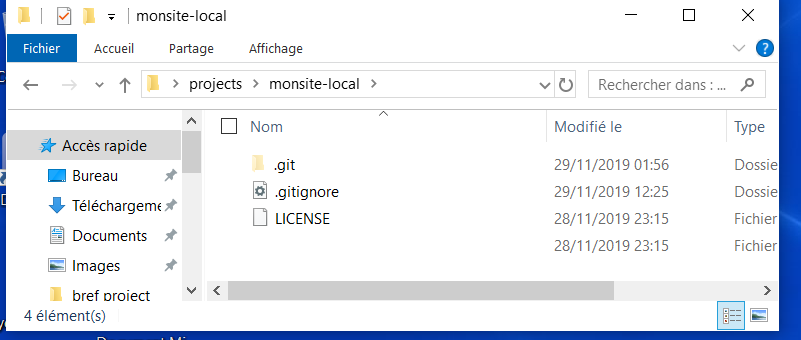
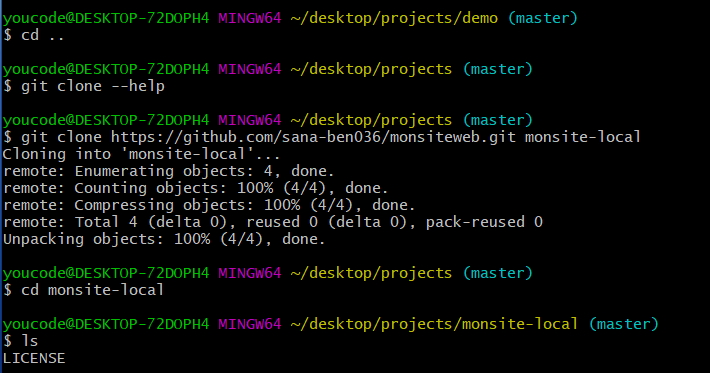
* **Sur Github Vérifiez la liste des commits, les branches, les releases et les tags**

****

**Step 4:  Mini challenge**

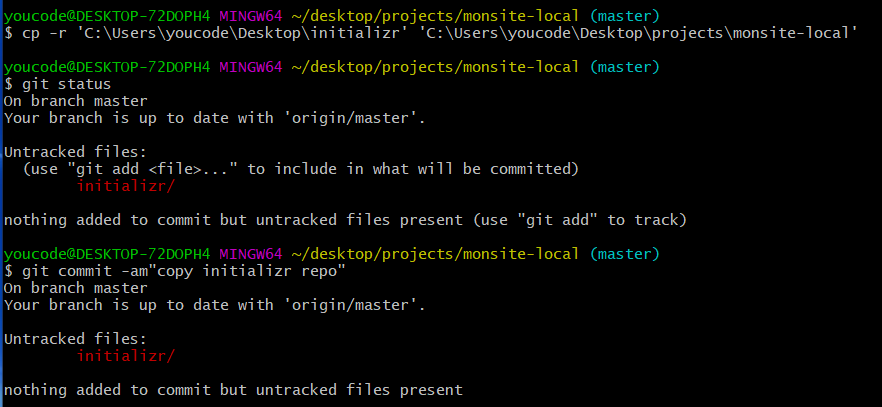
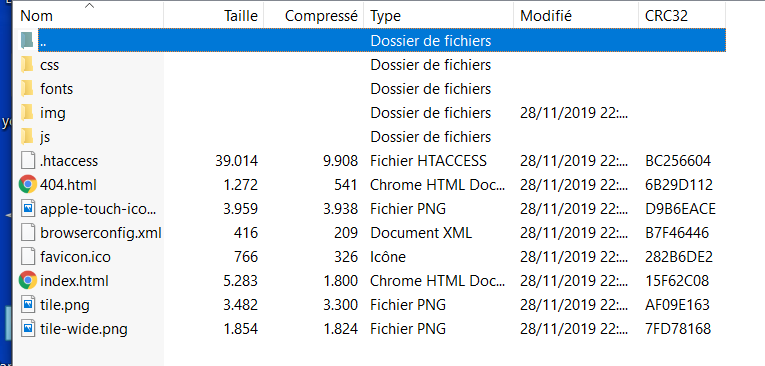


**Step 5:  Creation d’une local copy**

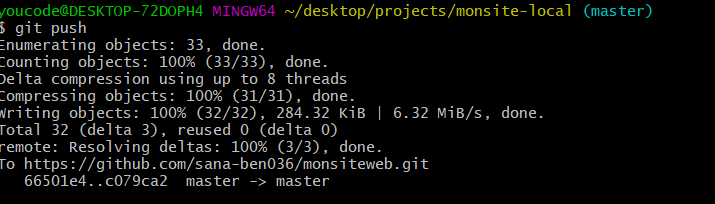
****

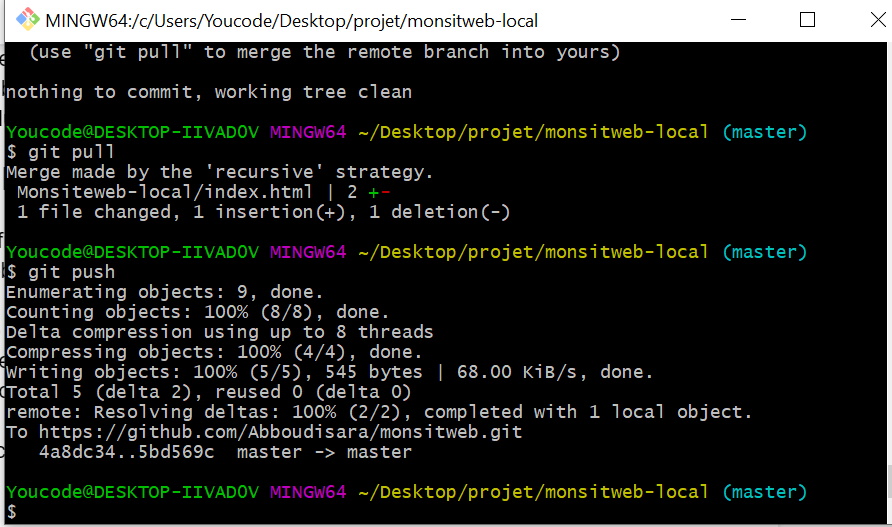
**step 6:  Sending the website**

* **Analysez l'arborescence.**
* **Expliquez?**

****

**step 7:  Fetch 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 staging et le commit en une seule ligne**
* **Qu’est-ce que vous constatez ?**

Les modifications faites sur le dépôt distant ne sont pas synchronisées avec le dépôt local et l’inverse aussi

* **Trouble shooting :**
  1. **Executez la commande « git fetch »**
  2. **Git status**
  3. **Expliquez ?**

Git fetch permet d’importer les modifications de dépôt distant sur le dépôt local ; mais ils nécessitent faire un merge.

* 1. **Git pull**
  2. **Git push**
  3. **Vérifiez les commit sur github**

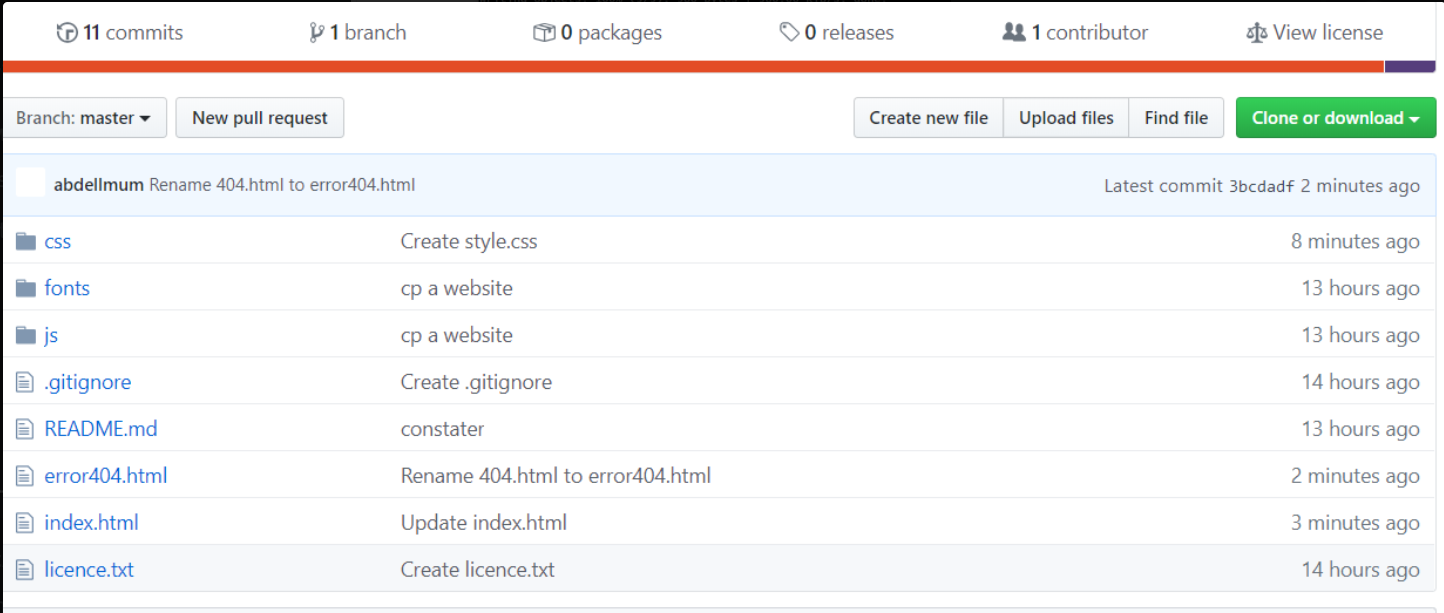
Senario III :

#1: First Step: Changes on Github

1. Sur le dossier CSS ajoutez un autre fichier nommée style.css

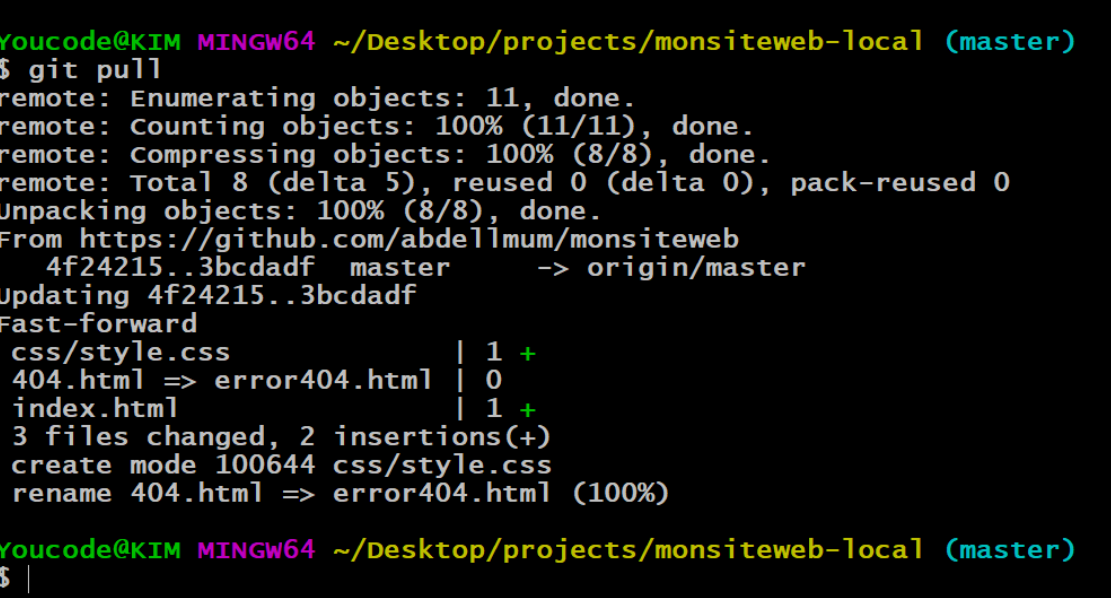
2. Sur index.html ajoutez juste après la balise <body> une balise <H1> : <h1> modification récente </h1>

3. Renommer le fichier 404.html en error404.html.

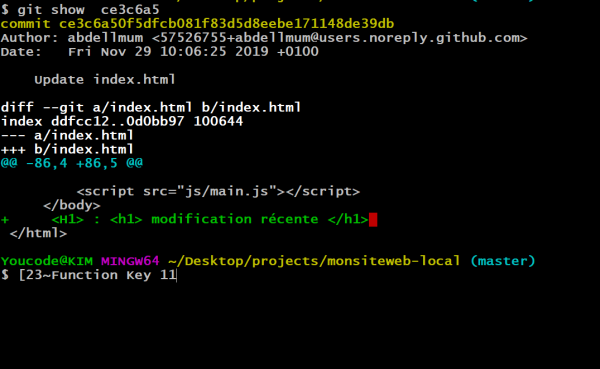


Maintenant faites le pull et fusionnez les changements distants avec le repo local

Visualisez et télécharger les fichiers distants sur GITBASH 4. Maintenant faites le pull et fusionnez les changements distants avec le repo local



Sur GITBASH affichez les informations du commit de l’ajout de la balise <h1></h1>



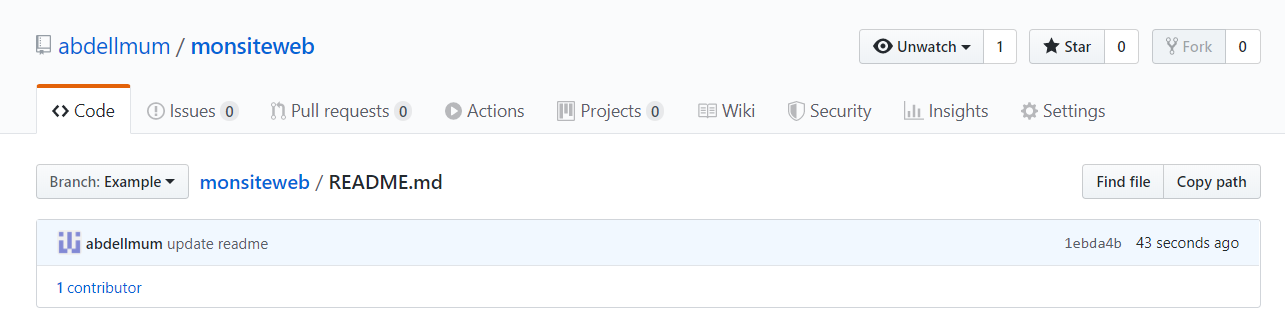
#2: Second Step: Branching and merging sur GITHUB

1. Repo distant :

⇒ Sur GitHUB créez une branche du nom ‘Example’

⇒ Sur la branche ‘Example’, modifiez le fichier READM.md et faites-le commit (toujours sur GITHUB)

⇒ Sur github et sur le menu ‘branch’ ; Expliquez ce que vous voyez comme message ?

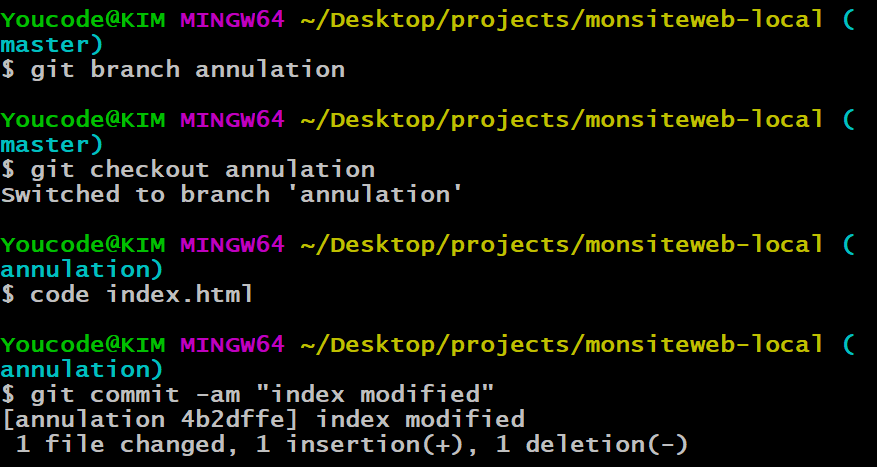


2. Repo local :

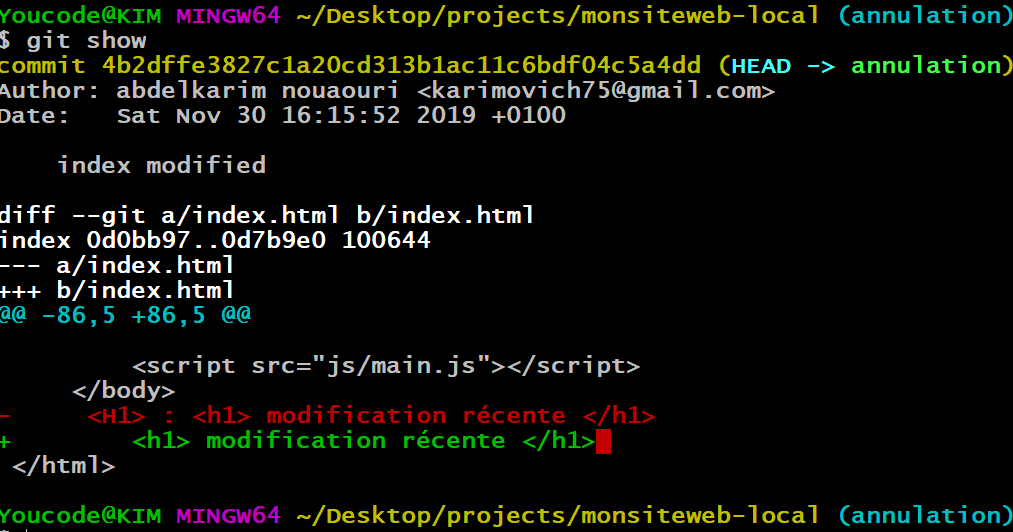
⇒ Sur votre repo local, créez une branche nommé ‘annulation’.

⇒ Sur votre repo local éditez le fichier « index.html » et supprimez la balise H1.

⇒ Faites-le commit et le stagging en une seule ligne



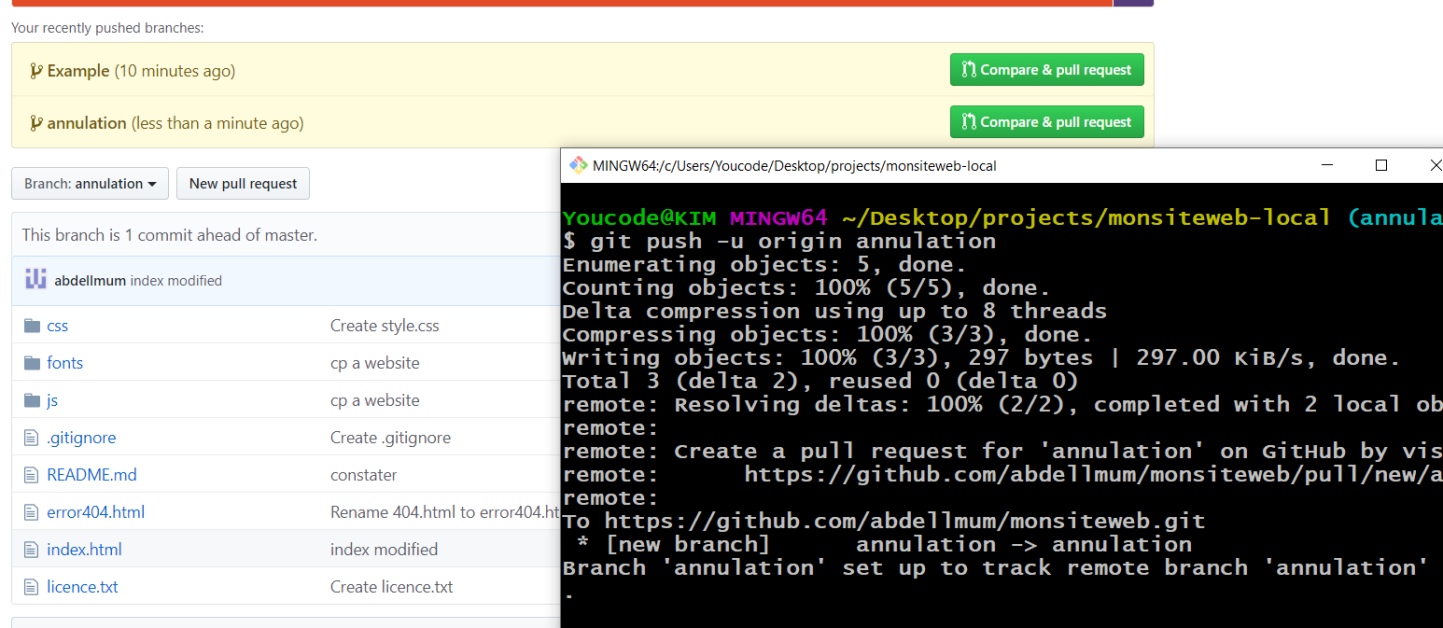
« Git show » pour vérifier vos changements



3. Vérification :

⇒ Sur GITBASH Pushez les changements que vous avez sur la branche ‘annulation’ à votre repo distant

⇒ Maintenant vérifiez votre repo distant.

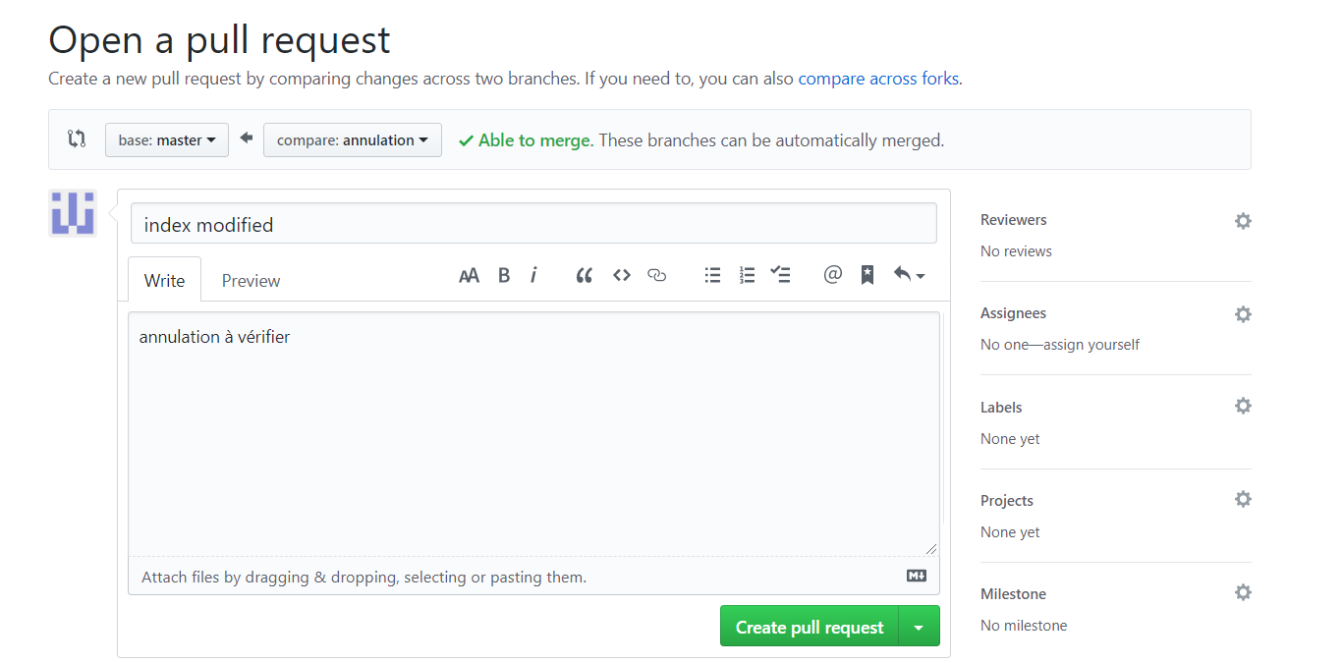


#3: Third Step: compare pull Requests

⇒ Sur votre repo Distant, un message indiquant qu’une branche a été ajoutez avec un bouton « compare and pull request »

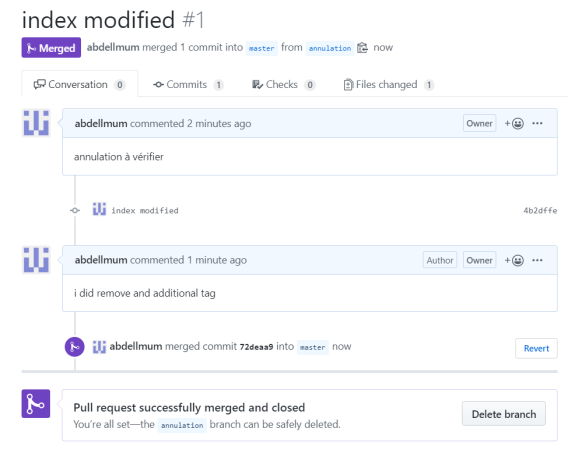
⇒ Cliquez sur le bouton ? analysez le rendu de la page ? qu’est-ce que vous constatez ?

⇒ Créez un pull request avec le commentaire ‘annulation à vérifier’



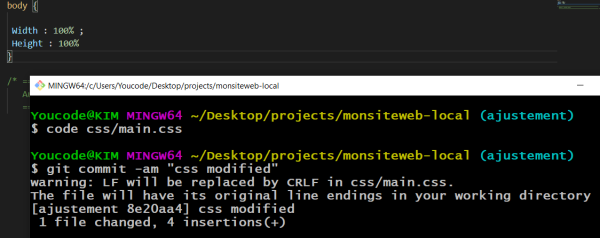
⇒ Maintenant si tout va bien ; aucun message d’erreur effectuez le merging ?

⇒ Supprimez la branche ‘annulation’ une fois tout est OK

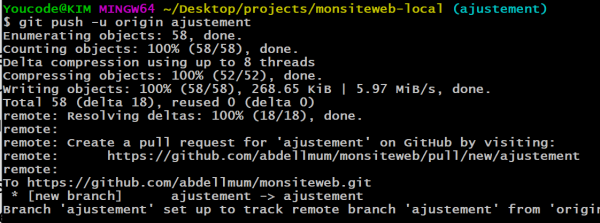


#4: fourth Step: merging en local

Pour finaliser l’objectif de cette étape, on va créer une branche nommé ‘ajustement’, ou on va modifier le fichier ‘main.css’ (css/main.css) en ajoutant la ligne suivante :



Faites maintenant le nécessaire pour ‘pushez’ les modifications sur GITHUB



⇒ Sur votre GITHUB, un pull request est demandé !

⇒ Maintenant, essayez de faire le merge en local ! @ vos mains !

1. Sur GITBASH, Vérifiez dans quelle branche vous êtes maintenant ! surement sur la branche

‘ajustement’

2. Un pull Request est demandé sur GITHUB ! N’oubliez pas !

3. Sur GITBASH, revenez vers la branche master ! Expliquez le rendu de la commande ?

4. Maintenant demandez le pull ?

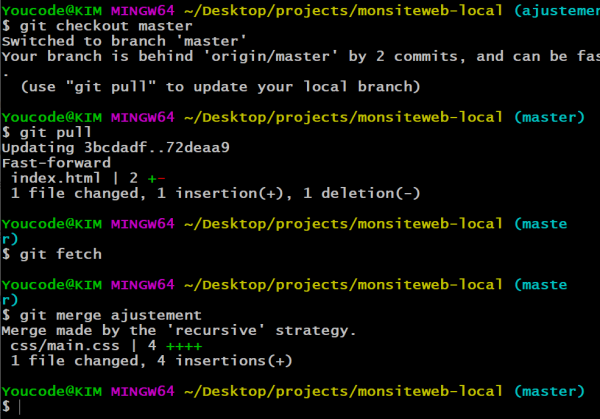
5. Une fois la commande du pull est exécutée, à votre avis quel type de merge on aura besoin

pour faire un merge sans conflits

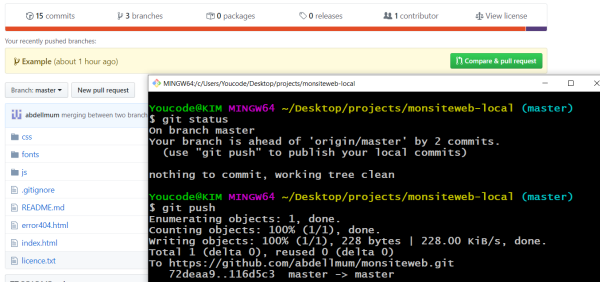
6. Faites le merge à travers GITBASH

7. Une fenêtre s’ouvre après l’exécution de la commande du merge ; ce qui signifie le message

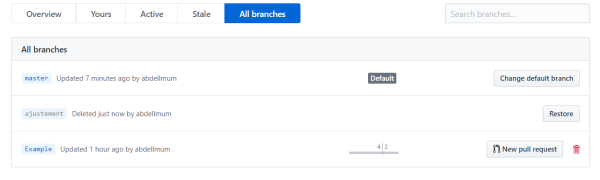
délivré ? à quoi sert cette fenêtre ?



10. Si tout est OK ! sur GITHUB le pull sera vérifier ? explorez le pull pour visualiser les changements ?



Sur GITHUB supprimez les branches que vous avez créé.

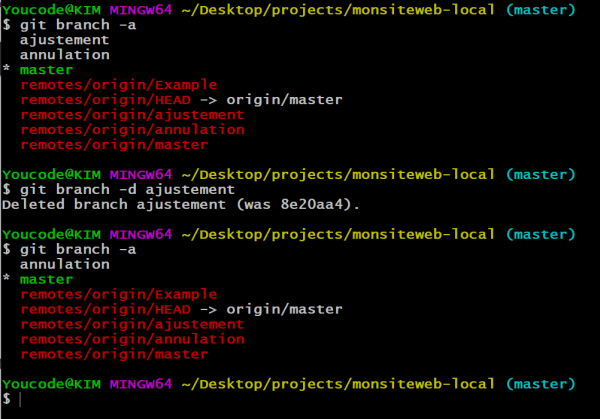


Sur le local on doit apporter les changements qu’on a fait sur GITHUB :

1. Tapez la commande git branch –a ? expliquez le message ?

2. Sur GITBASH supprimez les branches que vous avez créé.

3. Maintenant ! Tapez la commande git bash –a ? qu’est-ce que vous constatez ?



#5: fifth Step: The Cleaning up

Dans cette phase je vous laisse découvrir l’objectif ?

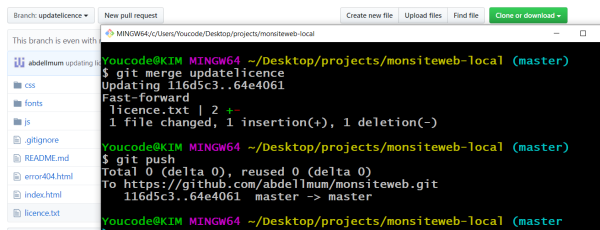
⇒ Sur GITHUB Créez une branche nommée ‘updatelicence, éditez le fichier Licence et changer Apache 2.0 par Apache 3.0

⇒ Sur GITBASH faites un pull global : git pull –-all ;

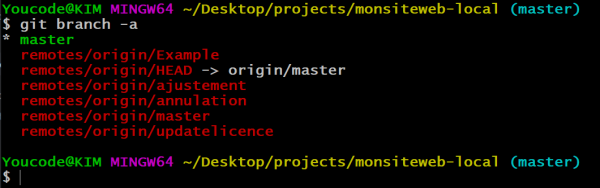
⇒ Maintenant faites le merge de la branche ‘updatelicence’.

⇒ Executez Git push

Git branch –d ‘updatelicence’



Exécutez la commande git bash –a ? qu’est-ce que vous constatez sur le remote et aussi sur GTHUB?

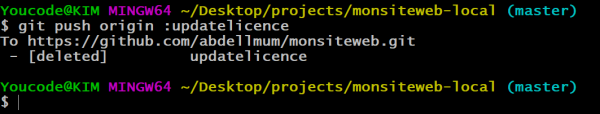


Challenge:

Maintenant sur GITBASH, trouvez la commande qui va nous permettre de supprimer la branche depuis le remote et ainsi de pushez les changements de la suppression simultanément ?

Vérifiez vos Repo (local et distant)

Pourriez-vous expliquez l’objectif de cette phase ?



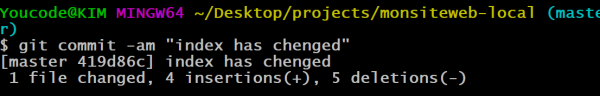
#6: sixth step: Rebasing

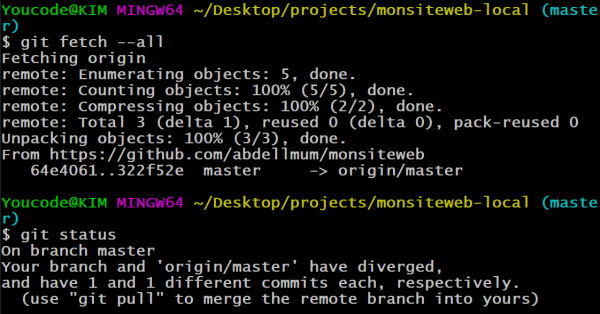
Sur GITHUB, Editez le fichier README.md, ajoutez la ligne ‘updates’ ! n’oubliez pas le commit

⇒ Sur GITBASH Editez le fichier index.html supprimez les lignes Et modifiez le titre du site : choisissez le titre que vous vouliez Faites le stagging et le commit en une seule ligne

⇒ Maintenant apportez les changements faites sur le Repo distant : fetch

⇒ Git status ? lisez les commentaires

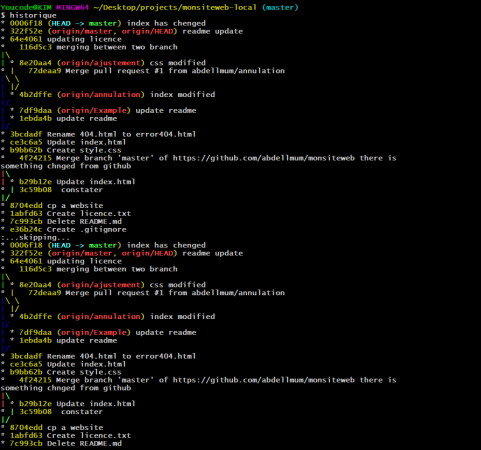


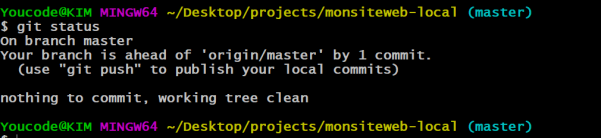


⇒ Maintenant si vous compreniez le commentaire, on a besoin de faire un rebase : git pull - - rebase

⇒ Exécutez l’alias que vous avez créé dans le scénario #1 (historique) ? qu’est-ce que vous

⇒ constatez sur l’arborescence des branches ?



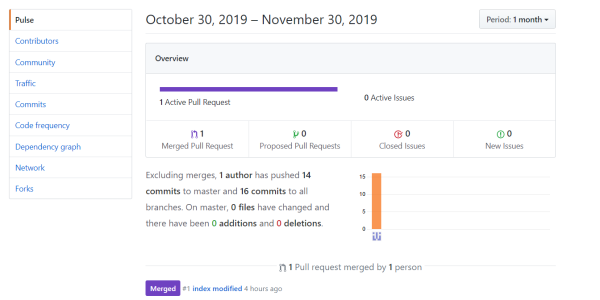


#7: seventh step GitHub Insights:

Sur Github vous avez un menu INSIGHTS ; Explorez votre activité à travers insights ⇒ Analysez le Network ou le timeline de votre workflow ?

⇒ Discutez entre vous équipe et formalisez ce que vous voyez sur le Network en une production

⇒ Timeline sur Tableau à marqueur ou trello : imaginez aussi que chaque membre du groupe à contribuer dans une tache du Timeline Network ( Like a Sprint planning)



#8 : eight step : Default branch and conflicts

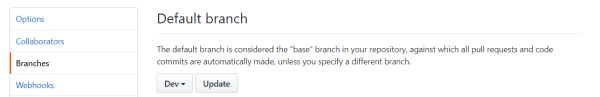
⇒ Sur github créez une branche du nom “Dev”

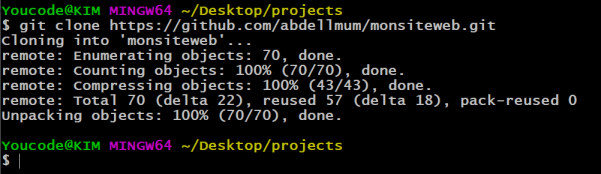
⇒ La Branche Dev deviendra la branche par défaut ; sur le menu « settings » mettez la branche Dev en mode par défaut

⇒ Ne faites aucun merge ni Rebase mais plutôt supprimez la branche feature depuis GITHUB

⇒ Sur GIT, supprimez récursivement et localement votre repo website

⇒ Maintenant Clonez via https ou SSH si vous l’avez réussi





⇒ Vérifiez si le contenu est récupéré

⇒ Vérifiez les branches et ou pointe le pointeur : HEAD

