**Scénario #1**



**Mustapha Kadouri Meriem Benrhabra**

**Abdelfattah Haissoun Soufiane Kadiri Raja Elhannioui**

[**https://github.com/hkidocode/demotest**](https://github.com/hkidocode/demotest)

[**https://github.com/hkidocode/Monsiteweb1**](https://github.com/hkidocode/Monsiteweb1)

[**https://trello.com/b/mSAAOirI/brief-projects**](https://trello.com/b/mSAAOirI/brief-projects)

**Product BACKLOG :**

# **1 : First Step : Immersion**

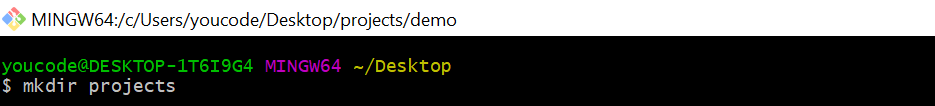
## Définition de git :

* Git est un outil de versioning qui permet de tracer l’évolution de votre projet et d’y apporter des modifications sereinement.
* Git permet d’avoir l’historique de l’avancement et les modifications du projet en cours de création.
* Option de branches pour travailler en souplesse.
* Gérer plusieurs fichiers ou projets.
* Pas besoin de se connecter à un serveur.

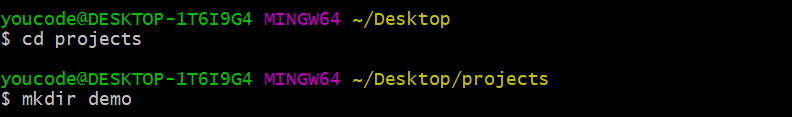
## Le fonctionnement du fichier caché «. git ».

* Lorsque vous créez votre repo local, avec **git init** ou **git clone**, Git initialise sa base de données, et la stocke dans un répertoire caché nommé **.git/**.

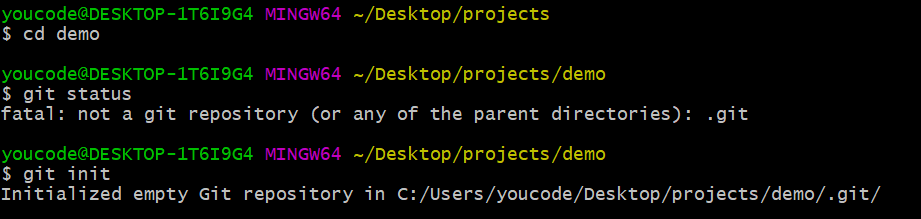
## Création d’un répertoire nommée « /projects » dans le bureau.

****

## Création d’un Repository Local sous le nom « /demo ».

****

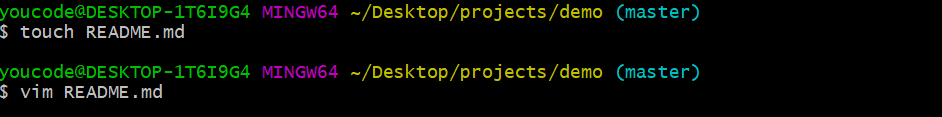
## Vérification de statut du répertoire « /demo ».

****

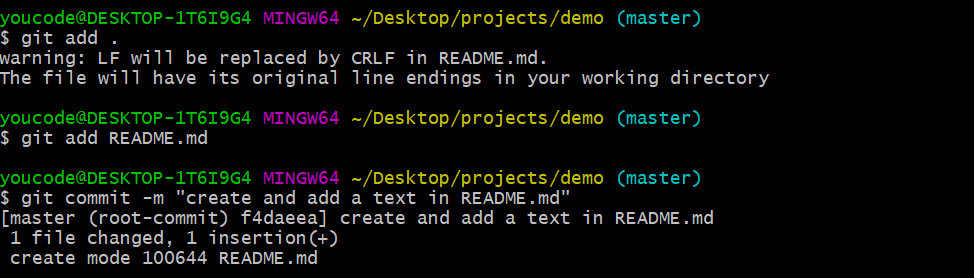
## Explication de commentaire

**« No commit yet**» nous n’avons pas encore effectuer aucune modification dans /demo.

## Création de fichier README.md

****

## Le stagging et le commiting avec un commentaire.

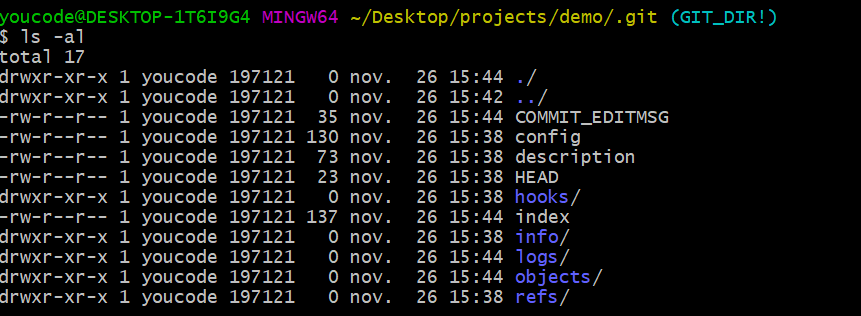
****

# **2 : Second Step : La découverte**

## Déplacement dans les fichiers de configuration « .git ».

****

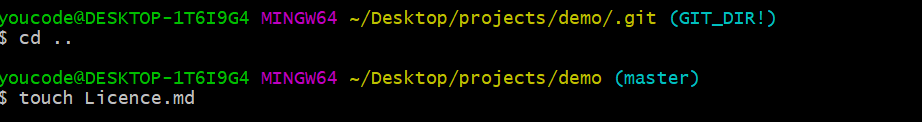
## Commande Ls –al :

****

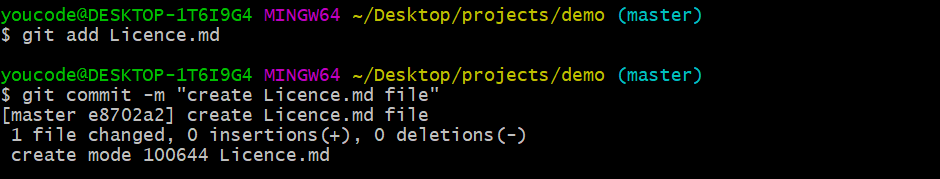
## Explication des : HEAD, LOGS, BRANCHES

* **HEAD :** ce fichier contient le chemin vers la référence qui indique la branche en cours
* **LOGS :** Le log de Git est un outil formidable pour analyser l’historique des commits et resituer un contexte. Il nous permet aussi bien de suivre un projet dans sa globalité que dans ses détails : fonctionnalités, correctifs, fichiers et répertoires, auteurs, dates…
* **BRANCHES :** Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits. La branche par défaut dans Git s’appelle master. Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.

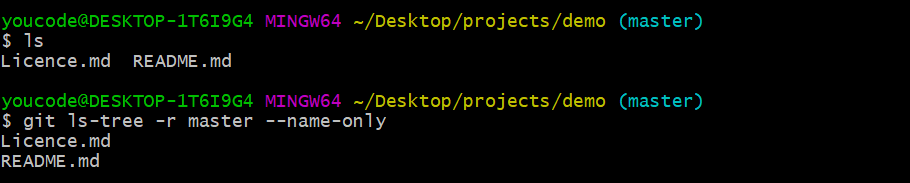
## Retour dans Repo et création de fichier « Licence.md ».

****

## Le commit :

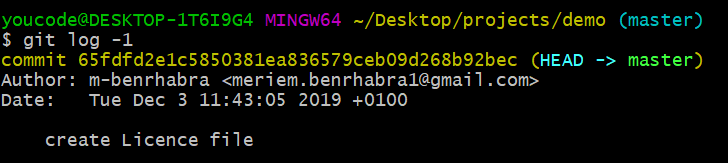
****

## Les fichiers traqués.

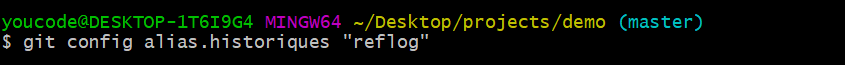
****

# **3 : third step : Historique**

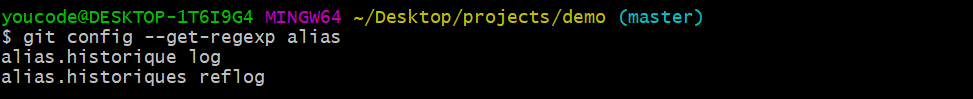
## Le dernier commit sur une ligne en ajoutant l’option d’affichage de la hiérarchie de la branche, avec les commit et leur branche aussi

****

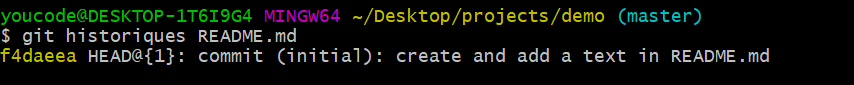
## Création d’un alias de la commande précédente le nom de l’alias est : historique

****

## La liste des alias

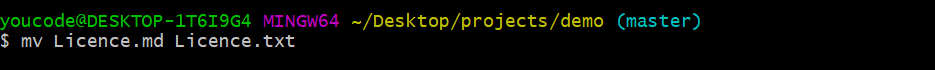
****

## L’historique des commit du fichier README.md avec l’alias.



# **4 : Fourth Step : Excluding files**

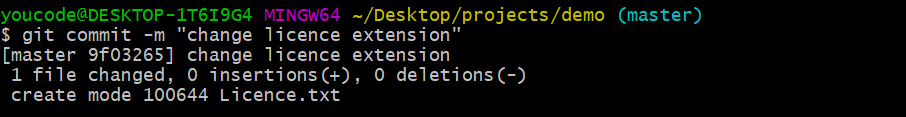
## Renommer le fichier Licence.md à Licence.txt



## Le stagging avec mise à jour (ne pas faire « git add . »)

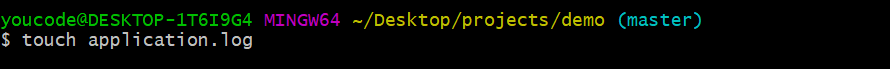
****

## Le commit :

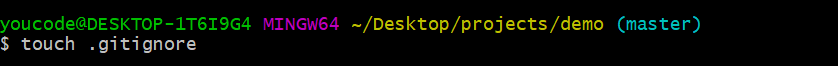
****

Supposons que nous développons sur notre plateforme, surement on a des fichiers qu’on veut exclure de notre arborescence du repository Local

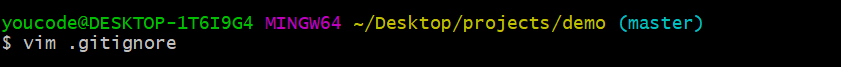
1. **Création d’un fichier nommé application.log**



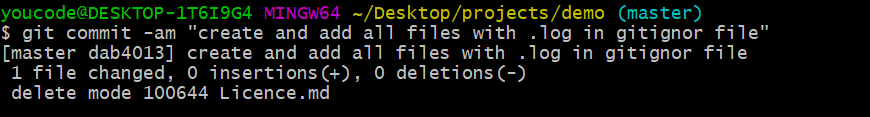
1. **Ne faites pas le stagging mais créez un fichier nommé « .gitignore »**



1. **Sur le fichier .gitignore  Ajoutez la ligne suivante « \*.log »**



1. **le stagging et le commit :**

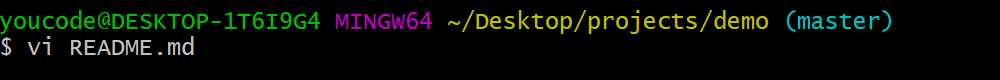
****

1. **Qu’est-ce que vous constatez ?**

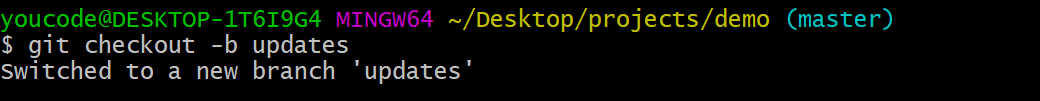
* Les fichiers qui sont dans .gitignore ne peuvent pas ajouter dans le dossier temporel .git

# **5 : fifth Step : Branching and Merging**

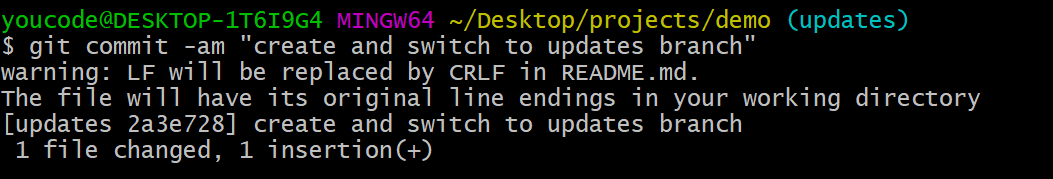
## Modification de fichier README.md :

****

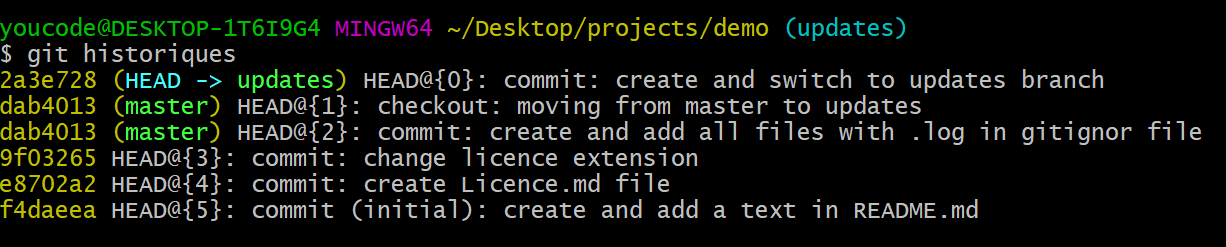
## Création d’une branche pour la modification du fichier README.md du nom ‘updates’

****

## Le stagging et le commit en une seule ligne.

****

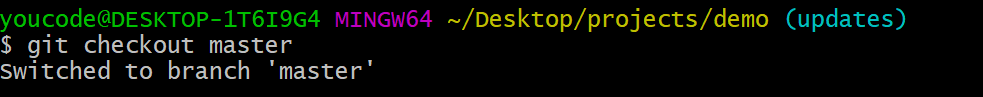
## L’historique avec l’alias



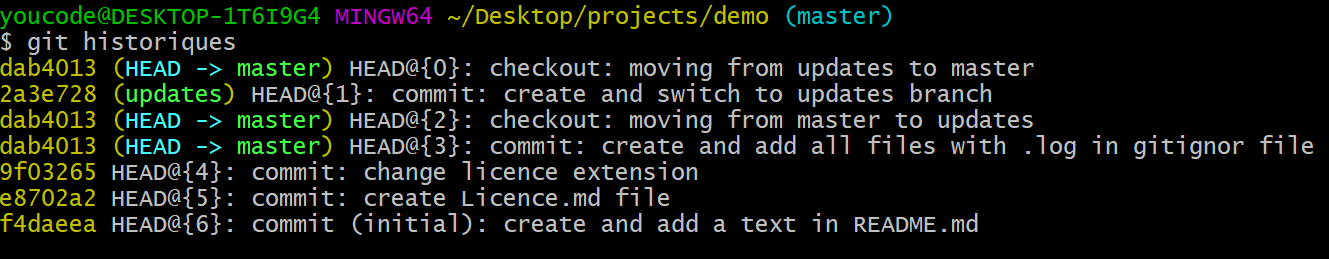
## Qu’est-ce que vous constatez ?

* On voit que la branche updates devenue dans la place de HEAD au lieu de master
* On remarque que la commande avec alias qui on a créé et toujours fonctionnée

## Retournez vers la branche Master

****

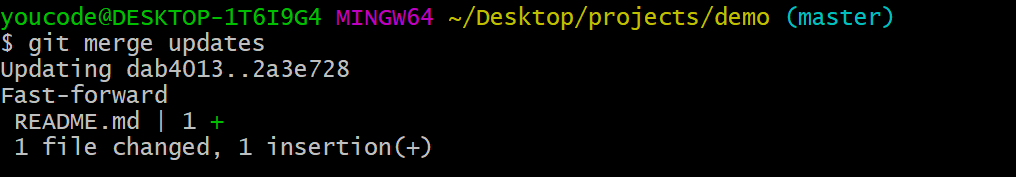
## L’historique avec l’alias

****

## Qu’est-ce que vous constatez

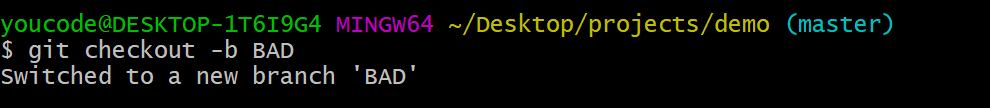
* On voit dans l’historique que le dernier commit créer par branche updates.

## Faites le merge

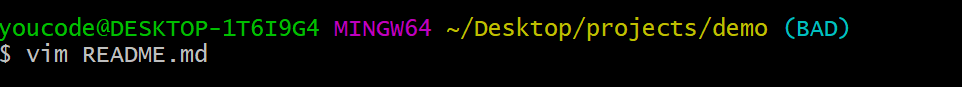
****

# **6 : sixth Step : Conflict Resolution**

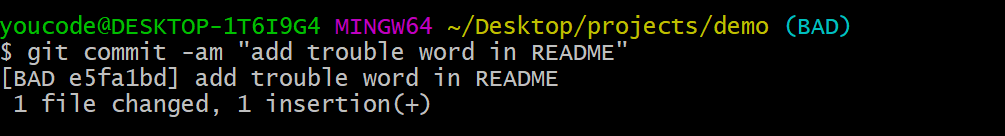
## Création d’une branche avec le nom ‘BAD’

****

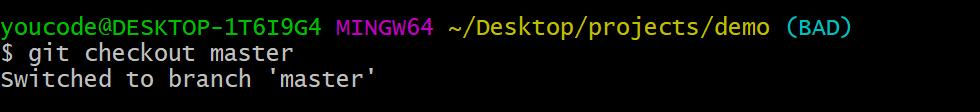
## Modifiez le fichier README.md et ajouter la ligne ‘Trouble’

****

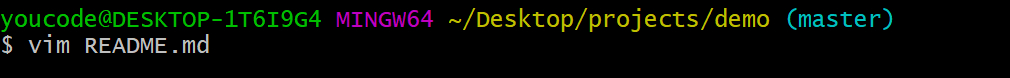
## Le stagging et le commit en une seule ligne



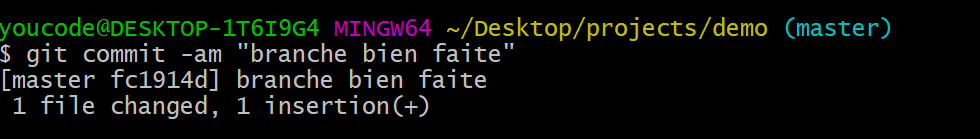
## Switchez faire la branche principale ‘master’



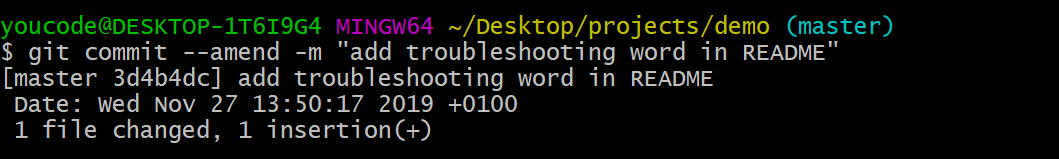
## Modifiez le fichier README.md et ajoutez la ligne ‘Troubleshooting’

****

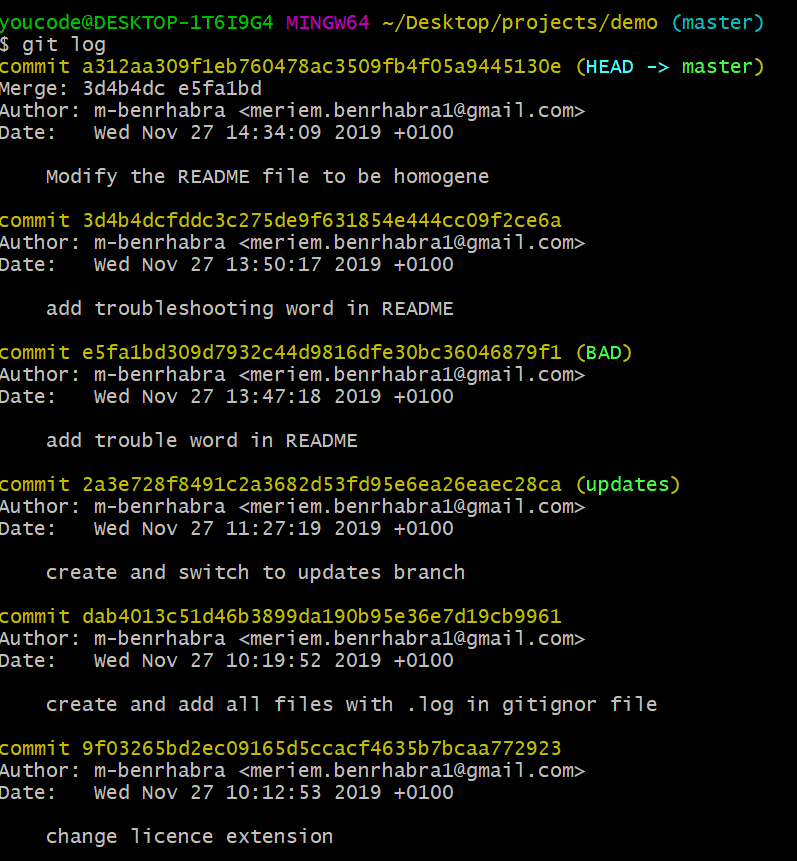
## Stagging/commiting avec commentaire ‘branche bien faite’



## Corriger le commentaire du dernier commit



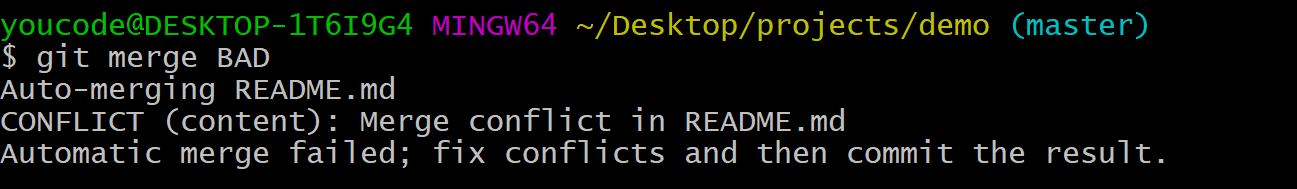
## Affichez l’historique



## Qu’est-ce que vous constatez ? expliquez l’objectif de ce petit scenario ?

* On voit que la dernière ligne de commande modifier le commentaire de dernier commit.
* Quand on fait une erreur dans un commentaire de commit, il peut être changé facilement.

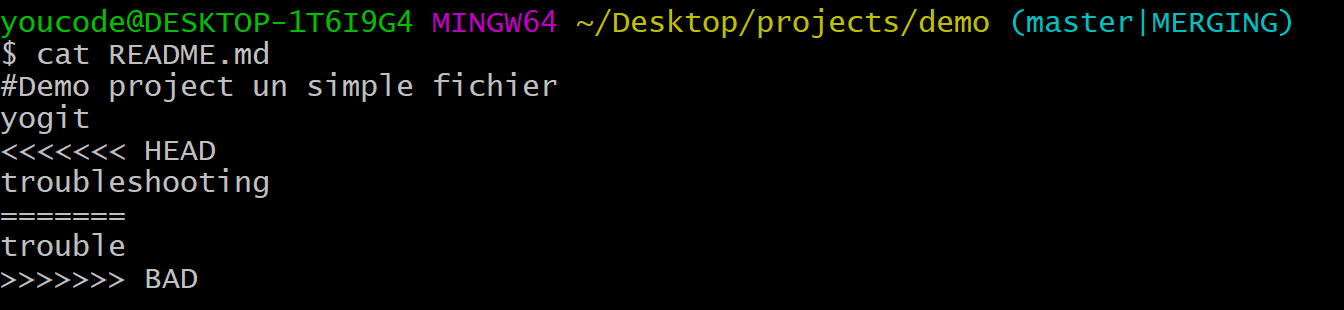
## Faites le merge de la branche ‘BAD’



## Expliquez le message ?

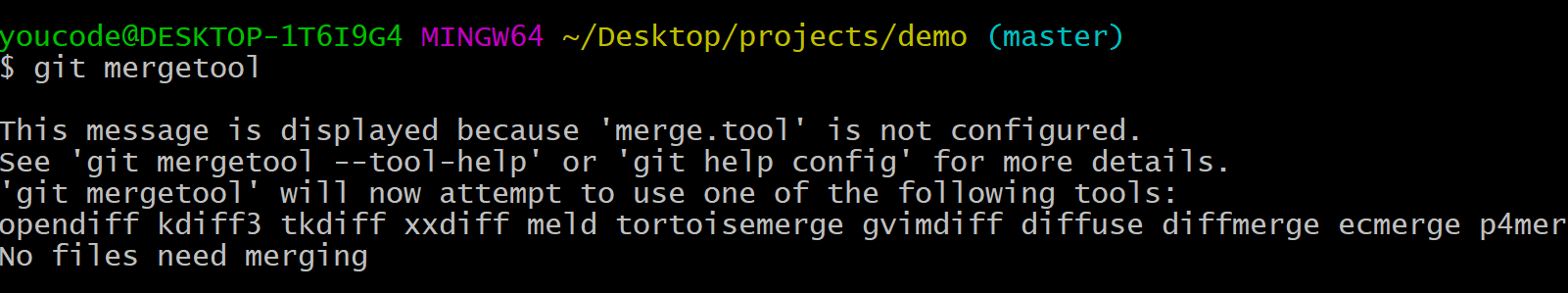
* Quand on fusionne la branche master avec la branche BAD, nous avons constaté que la fusionne automatique échoue à cause d’un conflit.

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



* La commande **cat README.md** afficher les deuxdéfirent version de branche master et branche BAD

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



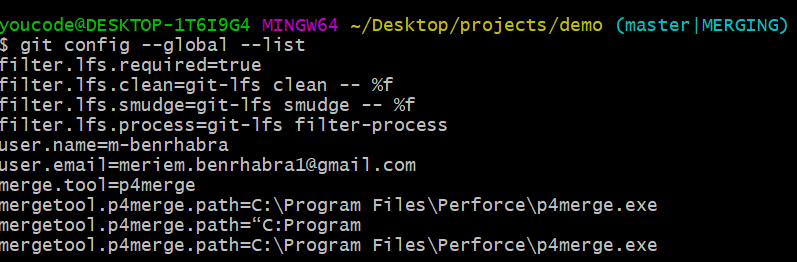
* La commande **git mergetool** n’a pas encore configurer

# **: seventh Step : merge tools**

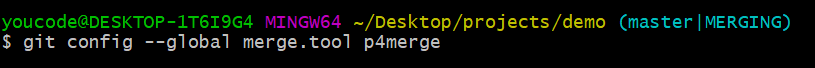
## Installation de p4merge sur le PC

<https://www.perforce.com/products/helix-core-apps/merge-diff-tool-p4merge>

## La commande : git config –global –list sur git bash



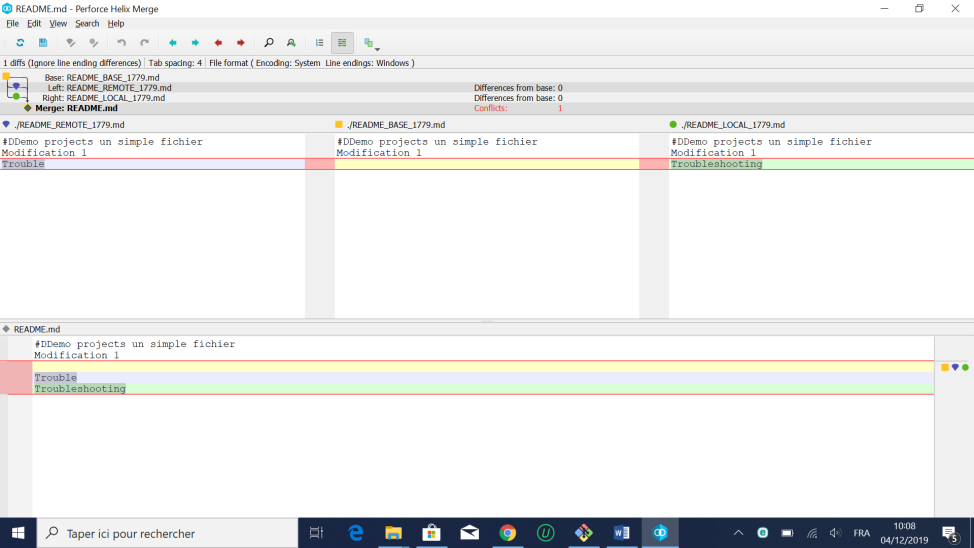
## La commande git config –global merge.tool p4merge



## La commande git config –global mergetool.p4merge.path “lien d’installation” ( .exe)



## La commande git mergetool

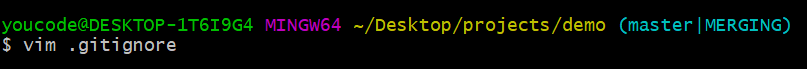


## Analysez ce que vous voyez ? et expliquez la plateforme ouverte en temps réel ?

* La commande git mergetool afficher une fenêtre devisée par trois version de fichier README.txt.
* La plateforme donne une accessibilité de choix entre les versions crée par les branches et choix compatible.

# **8 : eigth Step : Challenge**

## Sur le fichier **.gitignore** ; nous avons écrit une clause pour rejeter les fichiers indésirables et redondants



## Les fichiers : licence.txt Readme.md .gitignore :

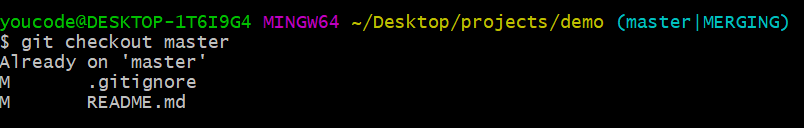


**Scénario #2**

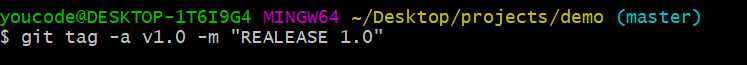
**Product BACKLOG #2 :**

# **1 : First Step : Tagging**

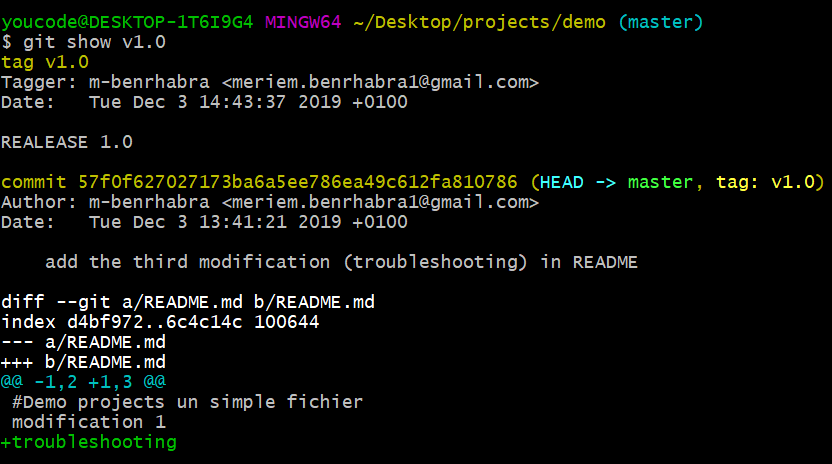
## La branche Principale



## Création d’un TAG avec un nom V1.0 et un commentaire ‘ REALEASE 1.0 ‘



## Les informations sur le TAG



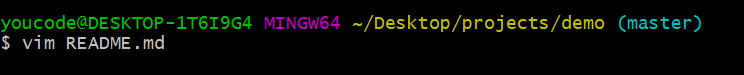
## Un TAG

* Généralement recommandé de créer des étiquettes annotées pour générer toute cette information mais si l’étiquette doit rester temporaire ou l’information supplémentaire n’est pas désirée, les étiquettes légères peuvent suffire.

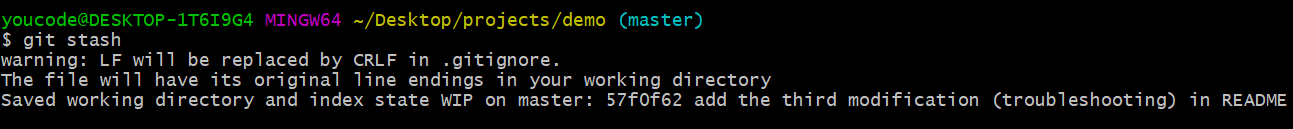
# **2 : Second Step : Stashing and Saving work in Progress**

## Modification du fichier README.md :

## 



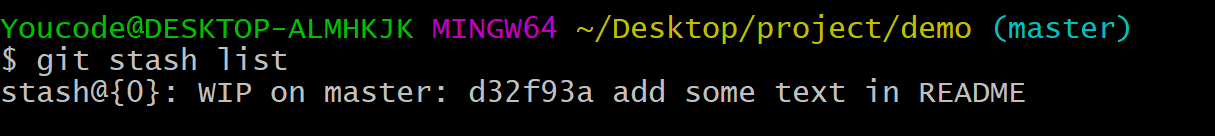
## La commande git Stash :



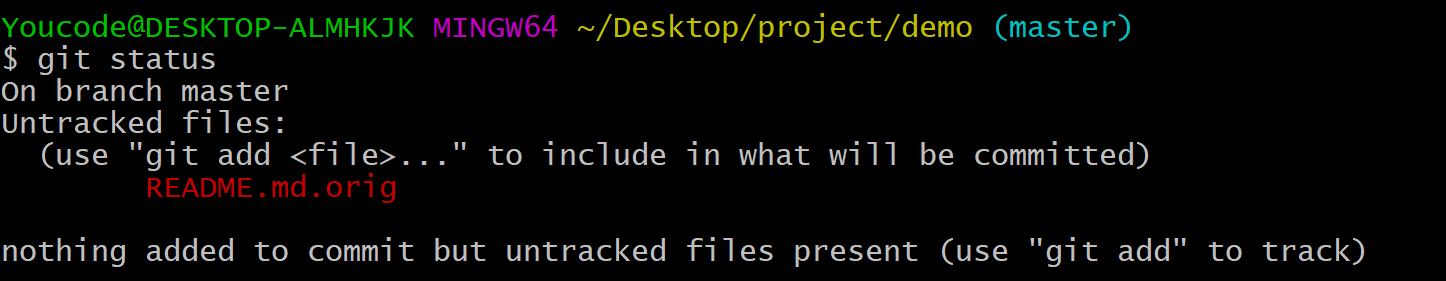
## Le fonctionnement de la commande git Stash :

* Git stash est une commande de Git qui permet de "**mettre de côté**" des modifications. Cette fonctionnalité est très pratique, notamment lorsque le développeur a besoin de changer de branche rapidement. Cette fonctionnalité lui permet de ne pas se presser à effectuer un commit bâclé pour pouvoir changer de branche.
* Git stash va récupérer les **modifications en cours**et les enregistrer dans un **conteneur** qu'on appellera un **stash**.

## La commande git stash list :



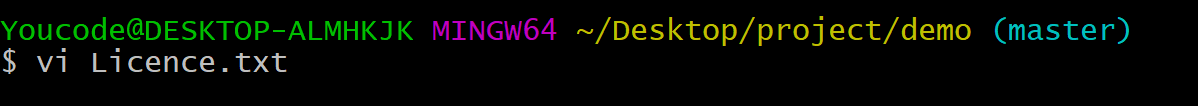
## La commande git status :



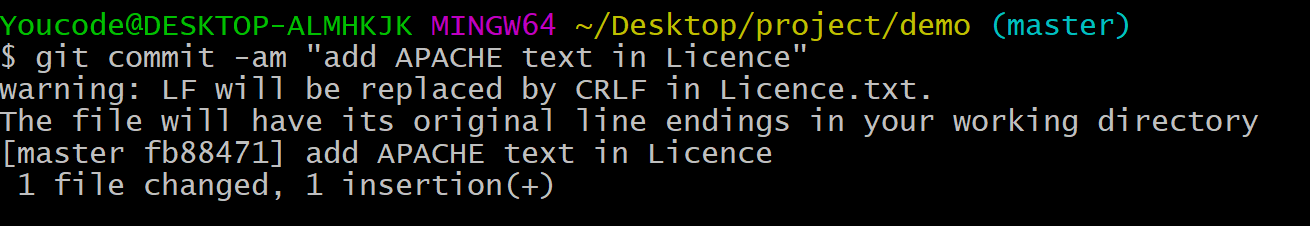
## Qu’est-ce que vous constatez ?

* On constate que le fichier **README.md.orig** n’est pas encore l’ajouter dans le dossier temporel **.git**

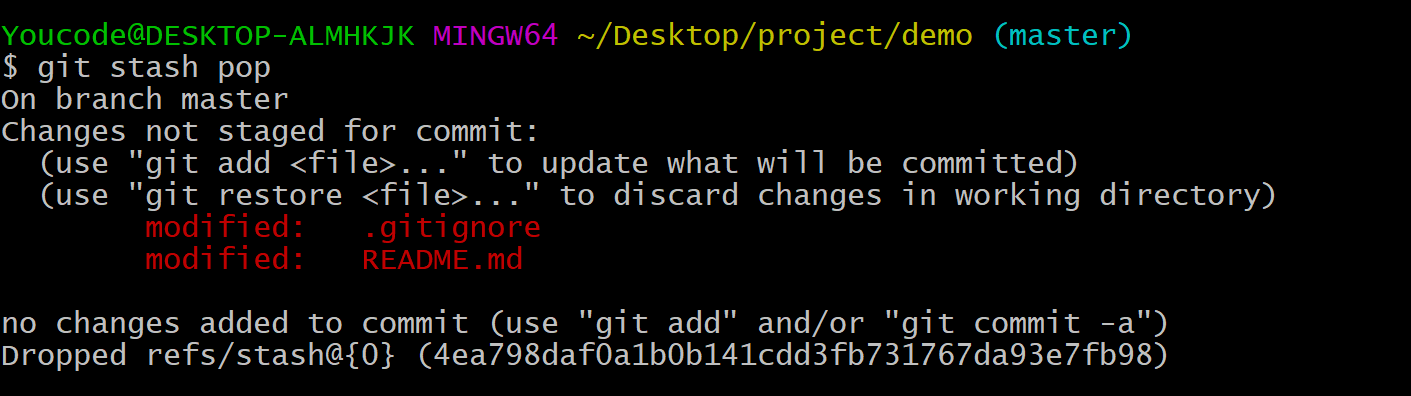
## Modification de fichier « Licence.txt » :



## Stagging et le commit en une seule ligne :



## La commande « git stash pop »



## Qu’est-ce que vous constatez ?

* On constate que la commande git stash pop reprend les changements qui ont ne peuvent pas valider par la commande **git commit**

# **3 : third step : Voyage sur GitHub, Local Repo to GitHub Repo**

## Création d’un repo github public :

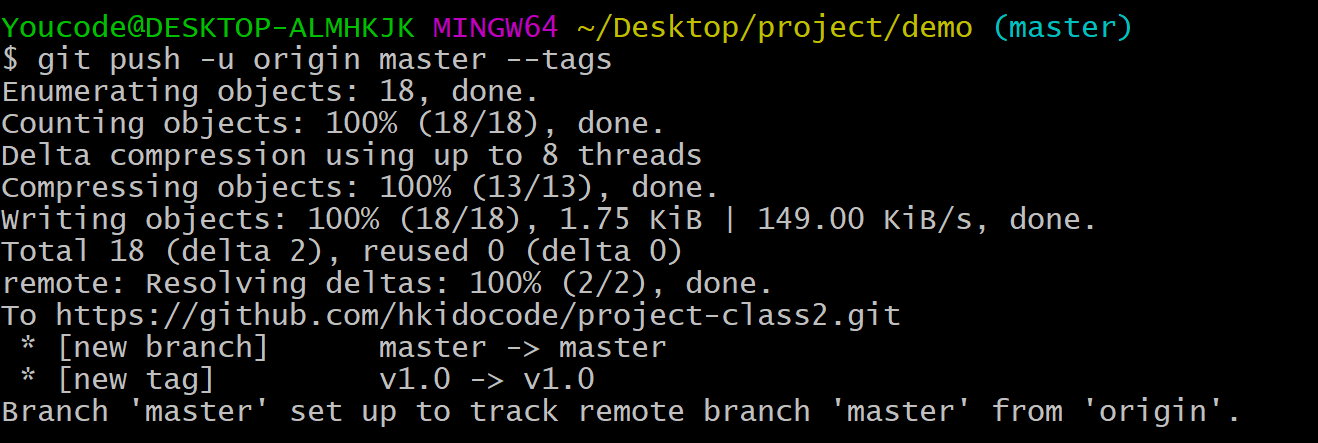
## Créez le remote en https



## Examinez le remote

Github

## La commande : git push –u origin master - -tags



## Explication du commande

* Push all tags from your local to the remote then add "--tags" to the git command and it will push all tags to the remote.

## Explication des options –u et - -tags

**–u :** Pour chaque branche mise à jour ou poussée avec succès, ajoutez une référence en amont (suivi), utilisée par git-pull (1) et sans autres commandes.

**- -tags :** Pousse toutes les balises git du référentiel local qui ne se trouvent pas dans le référentiel distant.

## Vérification de la liste des commits , les branches , les releases et les tags sur github

# **4 : Fourth Step : Mini challenge (optionnel)**

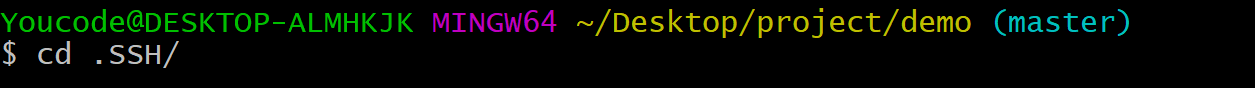
En examinant les types d’authentification sur GITHUB, on tombe sur l’authentification HTTPS et SSH, certes HTTPS est beaucoup plus facile à manager tandis que le SSH est plus sécurisé tandis que fiable en ce qui concerne les transferts cryptés.

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

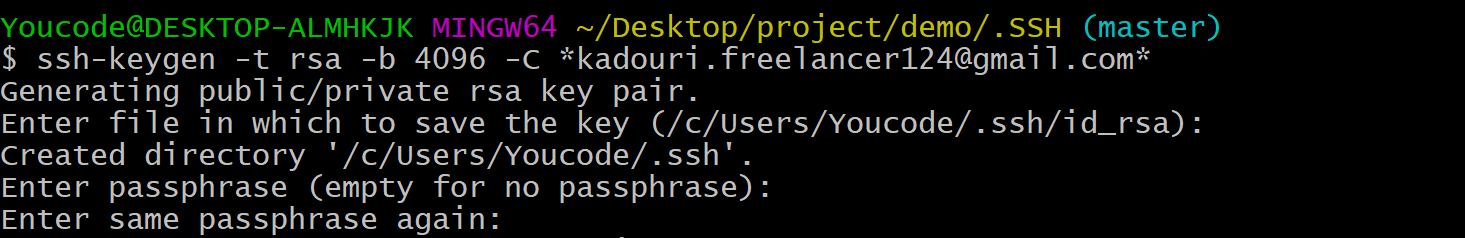
## Création d’un fichier sous git nommé **.SSH**

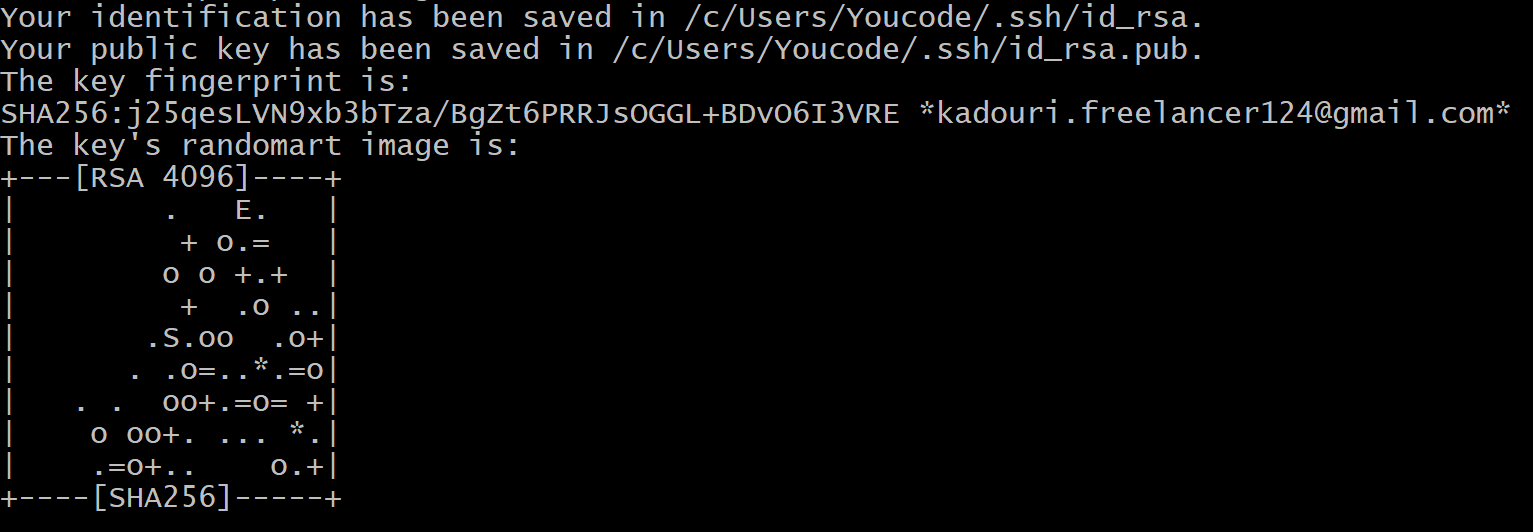


## Le fichier .SSH :



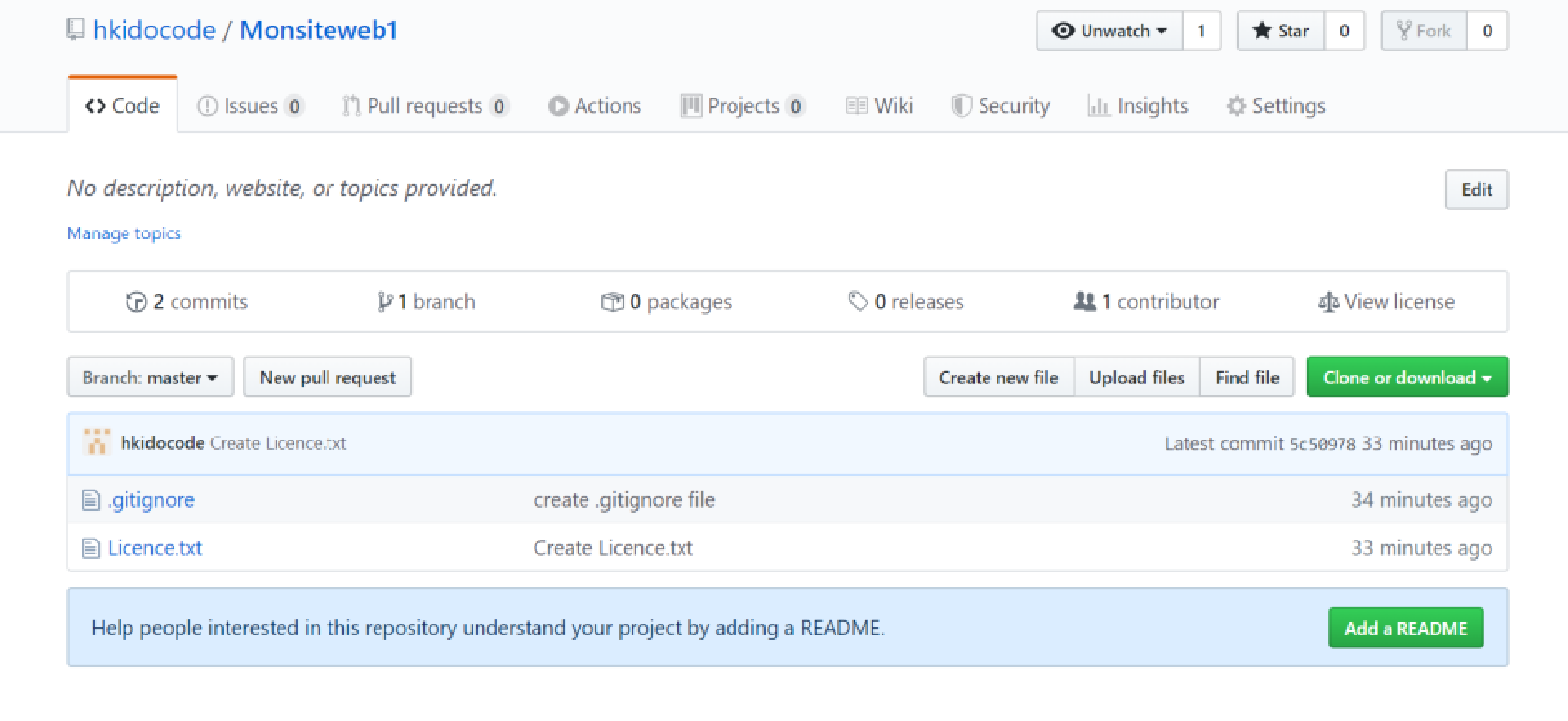
## Création d’une authentification sécurisé SSH entre le repo local et le repo distant (aide : Ssh –keygen-t rsa –C ' email’).



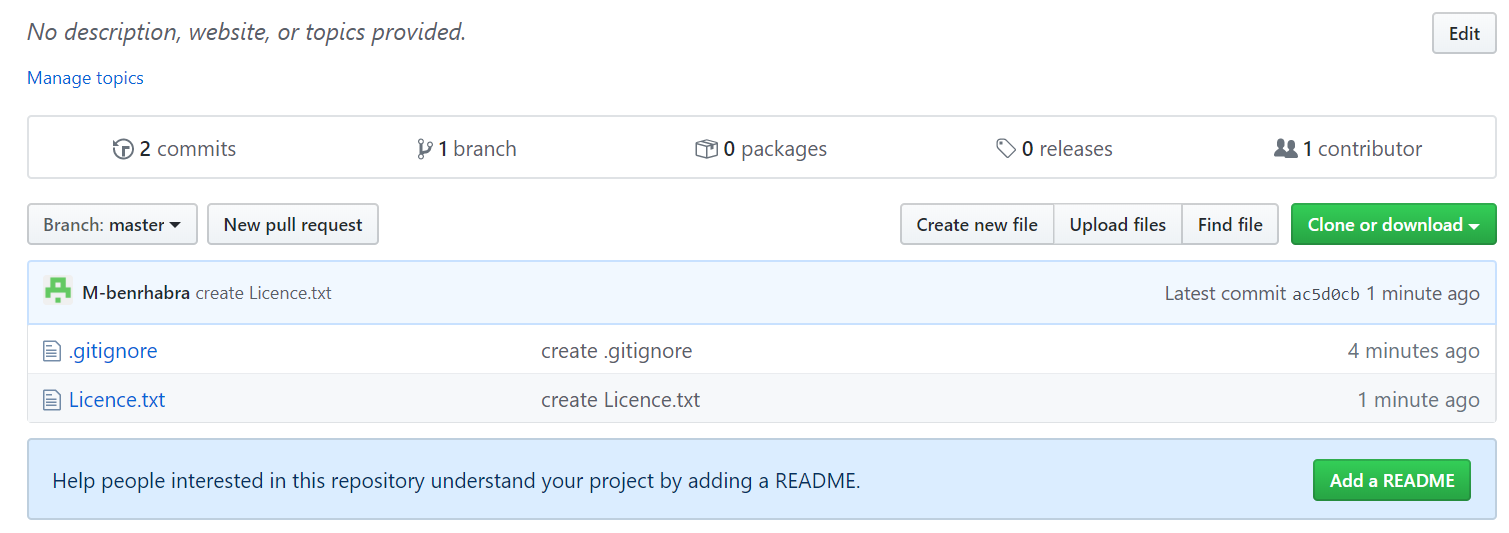


# **5 : Fifth Step : Création d’un local copy :**

## Création d’un autre repo nommé ( Monsiteweb) sur github



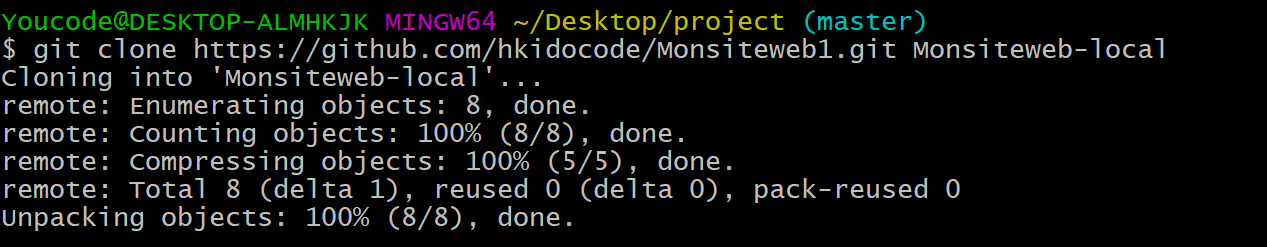
## L’ajout des fichiers : le fichier .gitignore et un fichier licence.txt ‘APACHE 2.0 ‘



## Le répertoire projets sous GIT :

* Par la commande **cd ..** (projects)

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

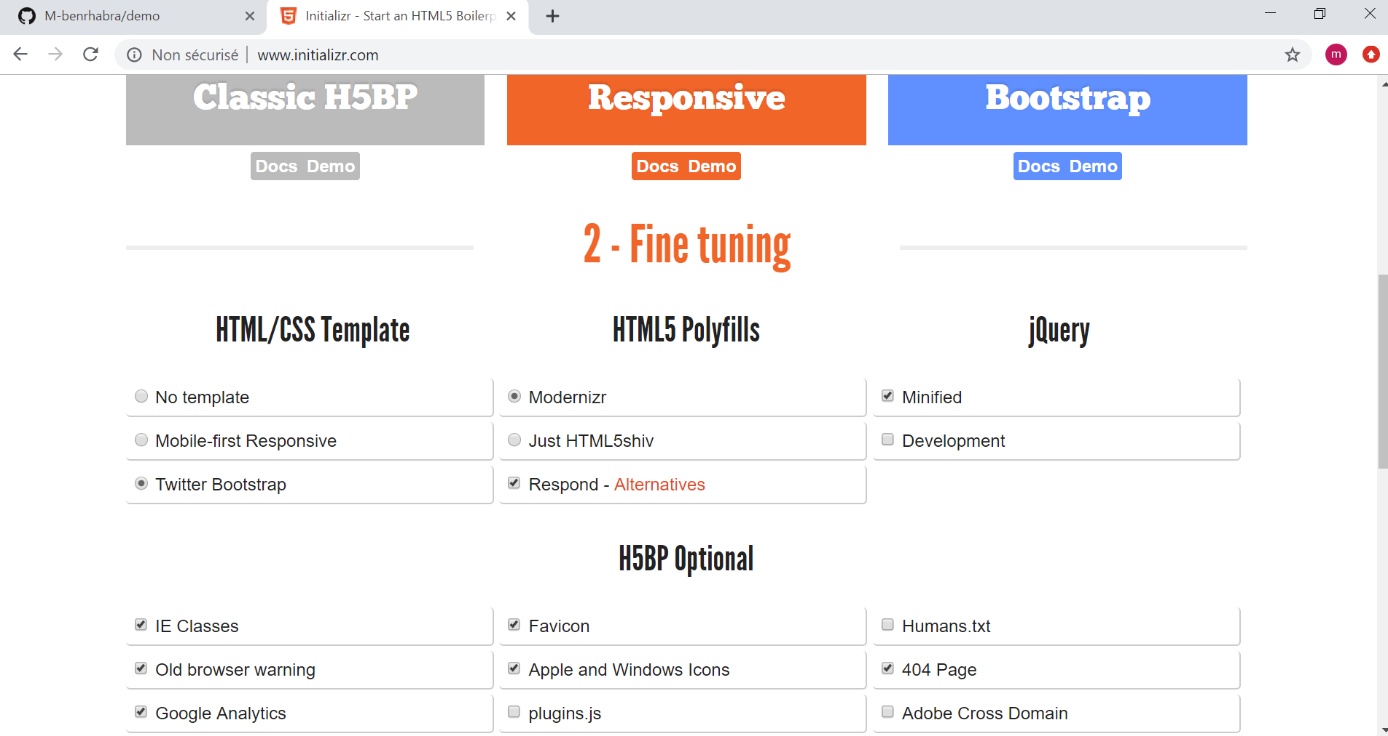
****

# **6 : sixth step : Sending the website :**

## Téléchargement du site web depuis le lien suivant :

<http://www.initializr.com/>

## Sur le site télécharger un site bootstrap avec le fichier **.htaccess** et le fichier **404.html**

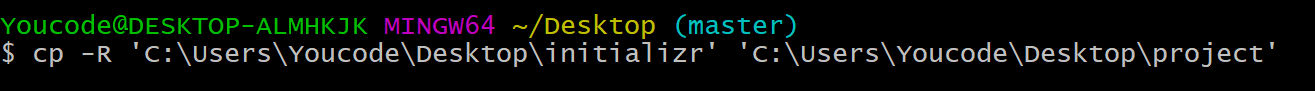


## L’analyse de l’arborescence.

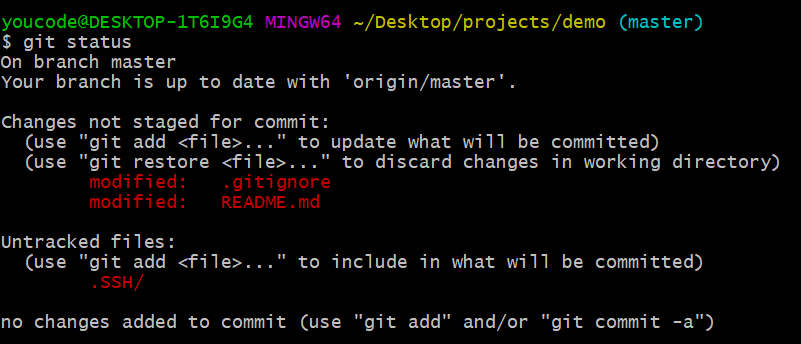
* L’arborescence de cette dossier initializr constituer de quatre dossiers {CSS ; FONTS ; IMAG ; JS}, deux fichiers HTML {404 ; INDEX}, un fichier HTACCESS, un fichier XML, trois images et un favicon.

Copiez le site télécharger dans votre repo local à travers une seule commande :

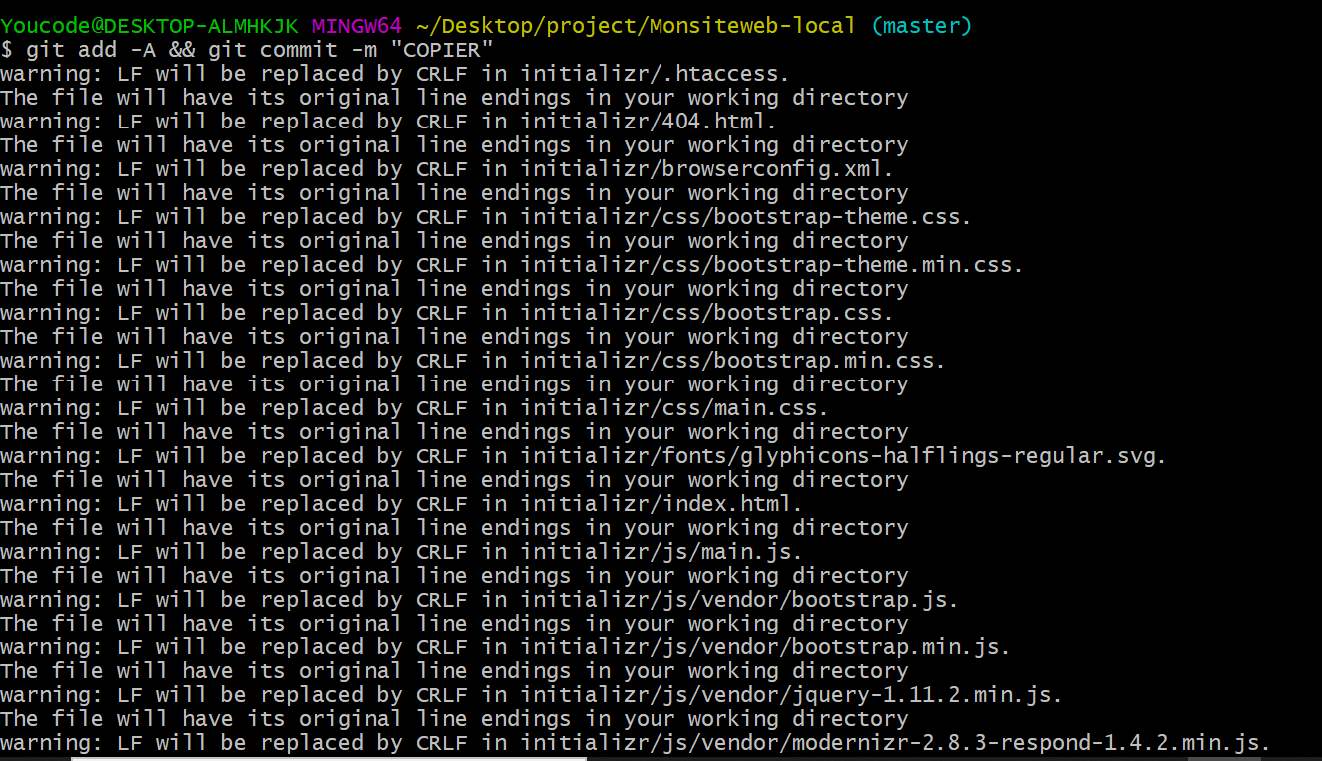
* Hint : **cp –R ‘~/DESKTOP/initialzr/\*’ . (Supposons que vous êtes sur votre repo local)**

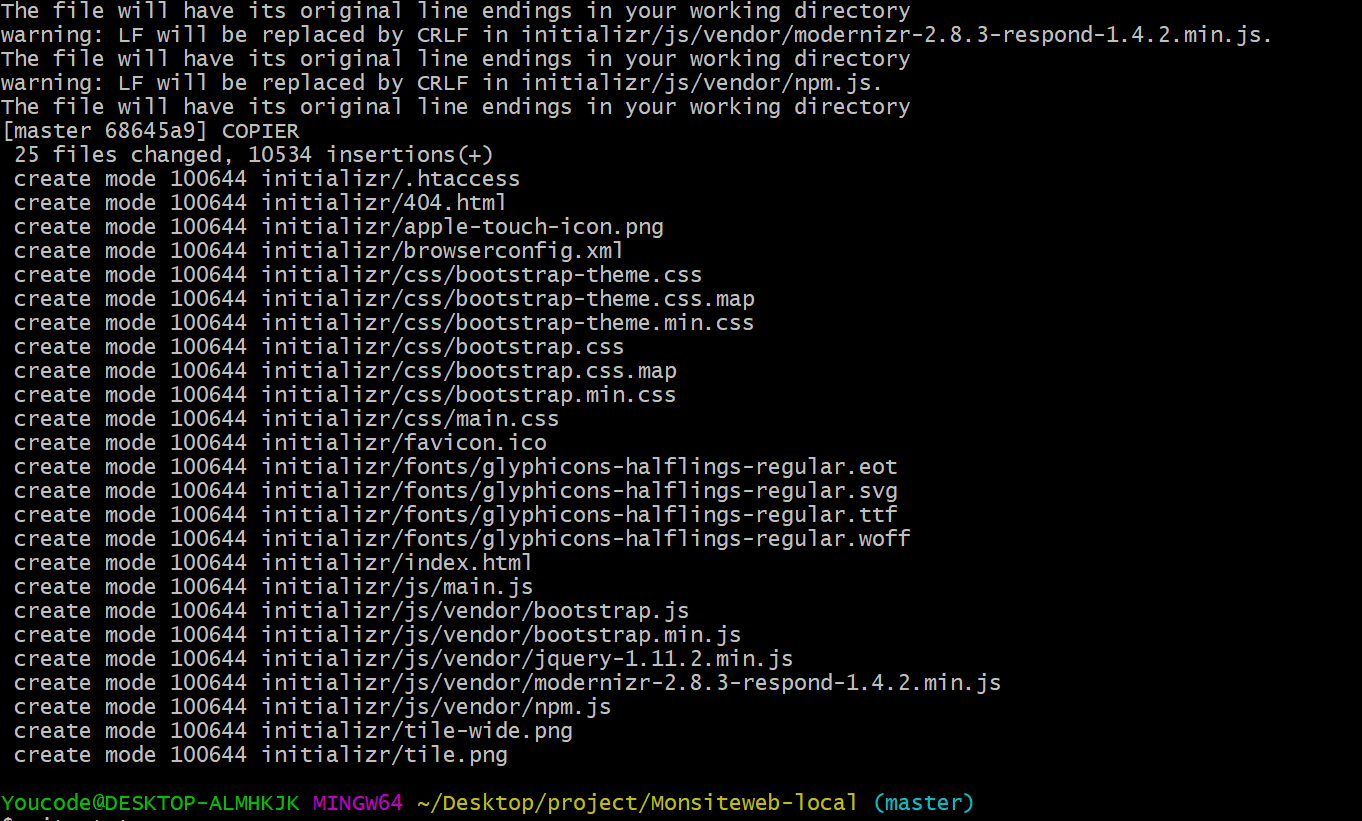


## Git status :

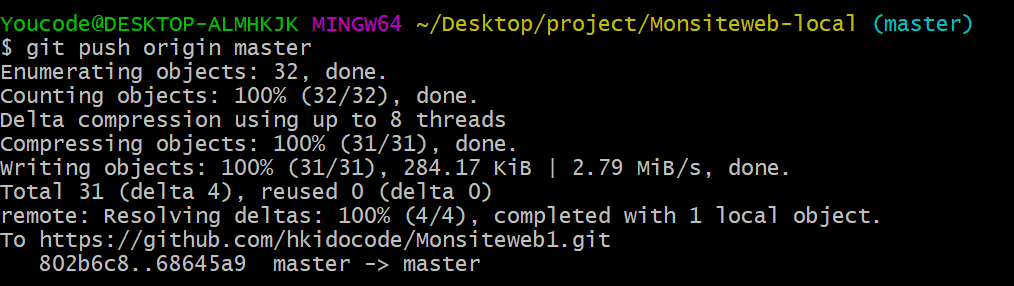


## Le stagging et le commit en une seule ligne

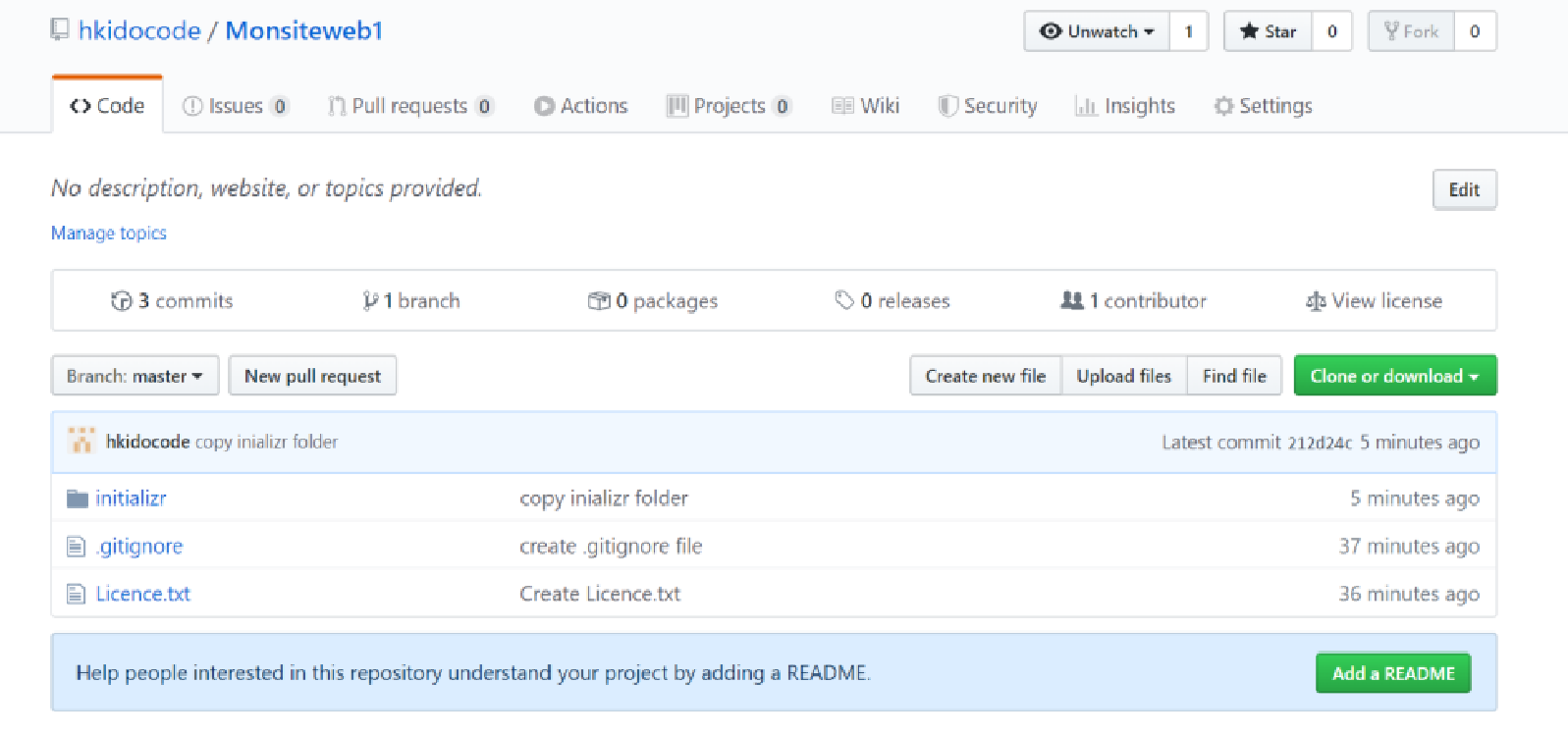




## Le push à github :



## Vérification de l’existence du site local sur votre repo github

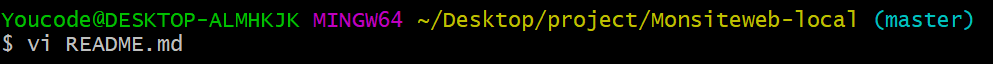


# **7 : seventh step : Fetch and pull :**

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

## Le commit sur github :

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



## Le stagging et le commit en une seule ligne

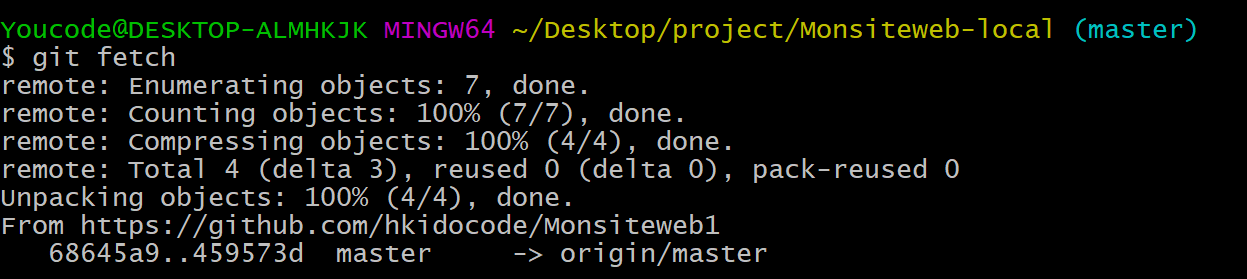


## Qu’est-ce que vous constatez ?

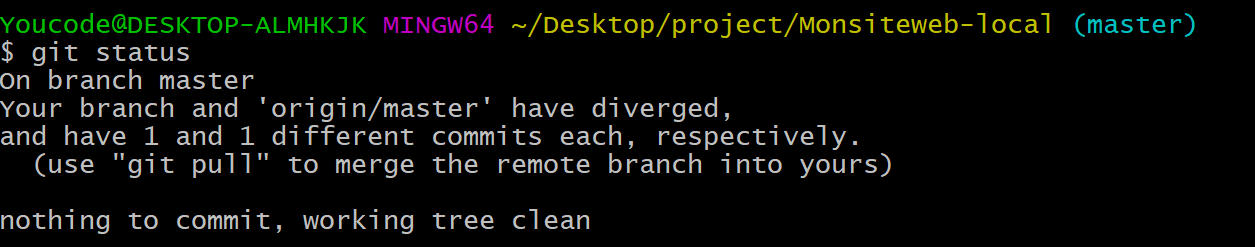
* On constate que on a fait deux insertion et une suppression.

**Trouble shooting :**

## La commande « git fetch »



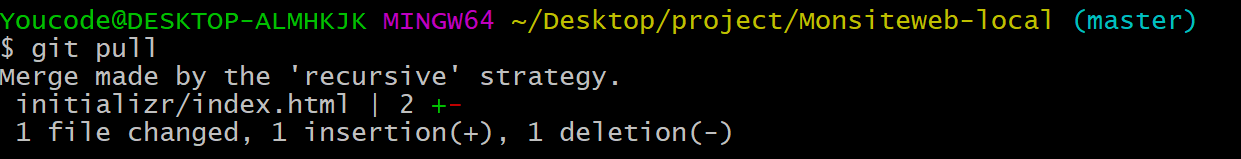
## Git status



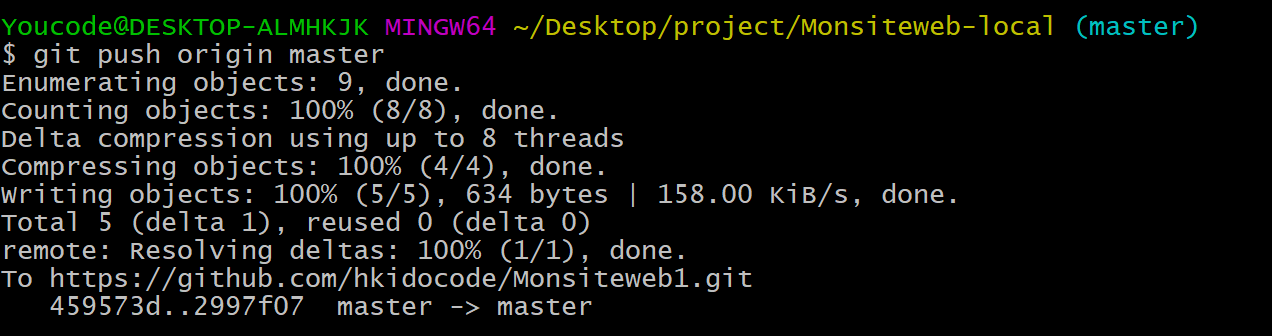
## Explication :

* On remarque qu’il y a deux différent commit fait par deux branches alors on doit faire la commande git pull pour mélanger les deux branches.

## Git pull



## Git push :

Vérification des commit sur github

**Scénario #3**



**Product BACKLOG 3 :**

**1 : First Step : Changes on Github**

Sur votre compte Github : faites les modifications suivantes

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

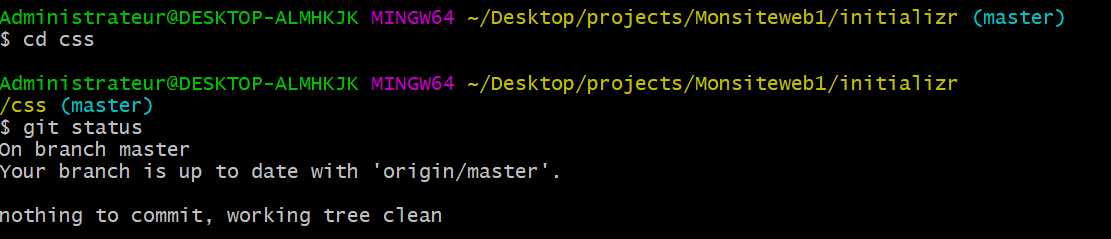
## Sur index.html ajoutez juste après la balise <body> une balise <H1> : <h1> modification récente </h1>

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

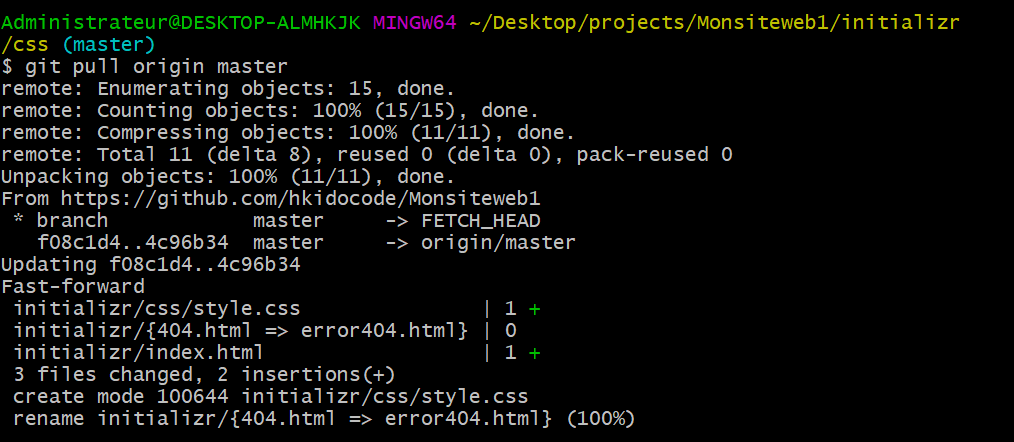
Maintenant la synchronisation avec votre repository local, les changements effectués doivent impérativement apparaitre sur votre repo LOCAL :

## Sur gitHub vérifiez la liste des commits.

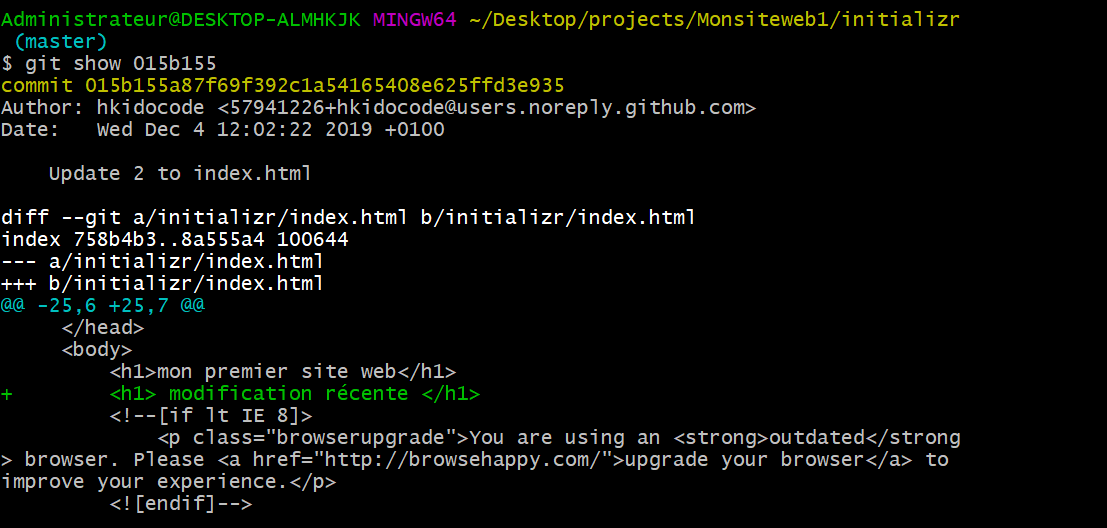
Sur GITBASH, vérifiez avec « git status »



## Le pull et la fusionne des 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’

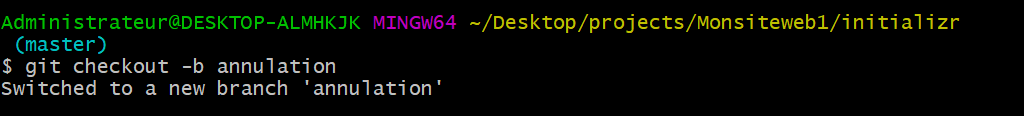
### Sur la branche ‘Example’ modifiez le fichier READM.md et faites-le commit (toujours sur GITHUB)

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

* Compare and pull request : To create a pull request and merge the branch with master

## 2. Repo local :

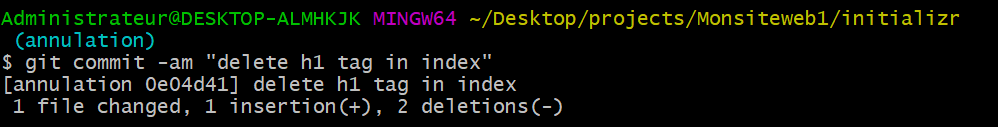
### Création d’une branche nommé ‘annulation’ Sur le repo local



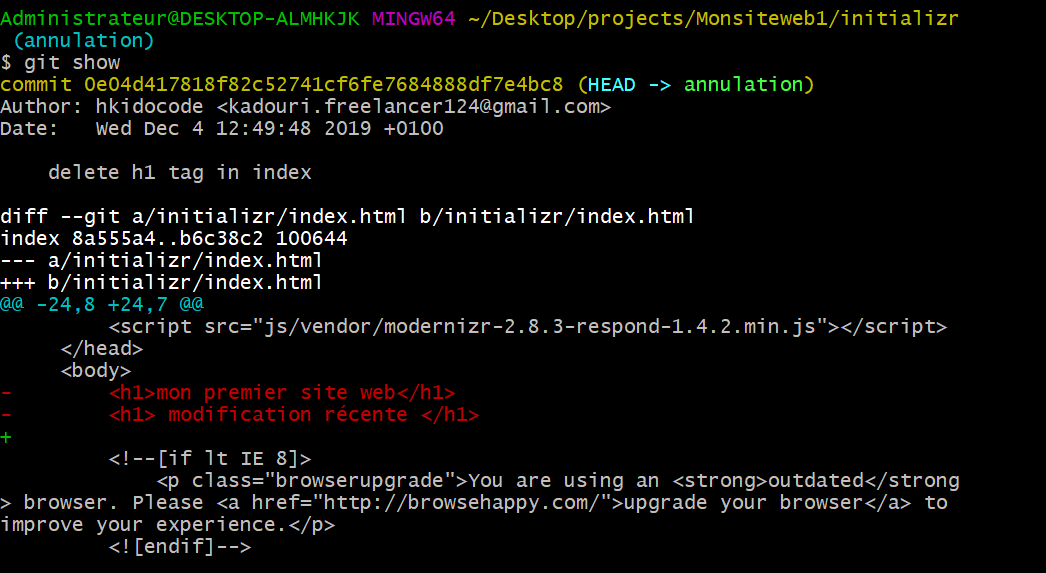
### Édition du fichier « index.html » et suppression de la balise H1 Sur le repo local



### Le commit et le stagging en une seule ligne.

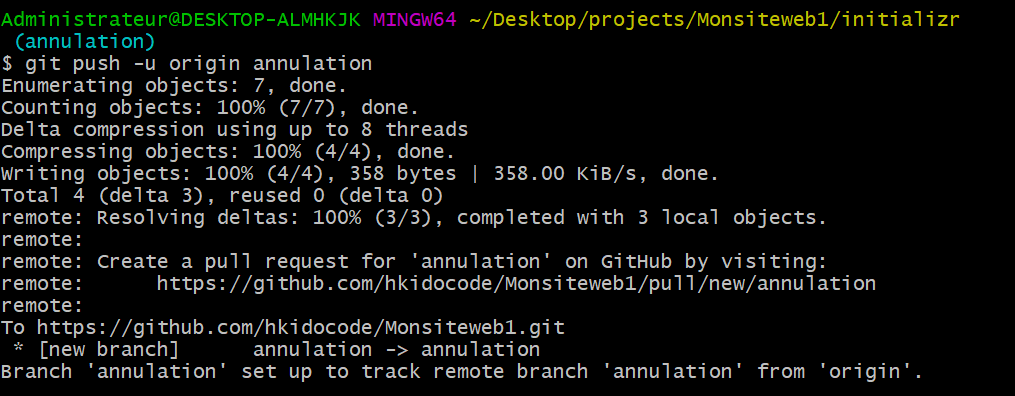


### « Git show » pour vérifier vos changements.



## 3. Vérification :

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



### Vérification du repo distant.

### Qu’est-ce que vous constatez ?

* On remarque qu’une nouvelle « pull request » était ajouté. Une branche ne fusionner pas automatiquement avec le repo principal avant la création une pull request.

# **3 : Third Step : compare pull Requests**

## Un message indiquant qu’une branche a été ajoutez avec un bouton « compare and pull request » sur le repo Distant

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

* Après la modification on delivre une pull request au chef ou master du projet pour la validation de la modification

## Création d’un pull request avec le commentaire ‘annulation à vérifier’

Une page s’ouvre, explorez la page ? qu’est-ce que vous constatez ?

Sur l’onglet ‘conversation’ vous pouvez aussi laisser des commentaires à la personne qui demande le pull request ; laissez un commentaire ?

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**

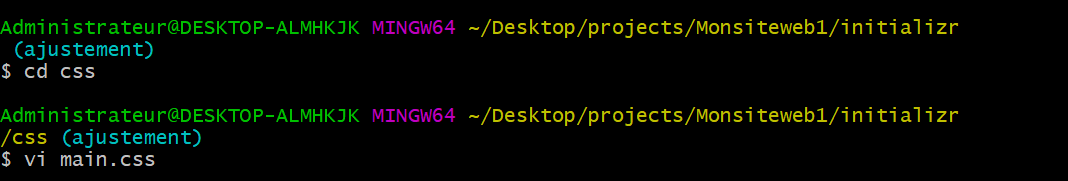
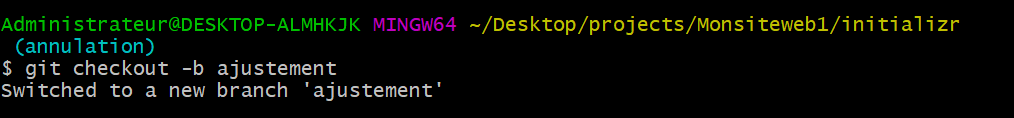
Pour finaliser l’objectif de cette étape, on va créer une branche nommé ‘ajustement’, ou on va modifier le fichier ‘main.css’ ( css/main.css) en ajoutant la ligne suivante :

**body {**

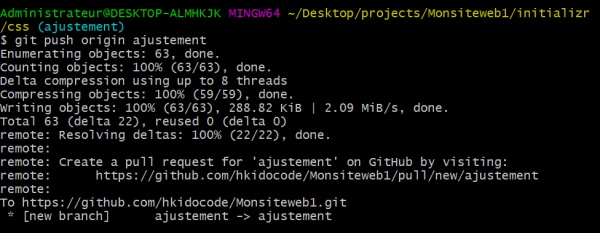
**Width : 100% ;**

**Height : 100%**

**}**



Push des modifications sur GITHUB.

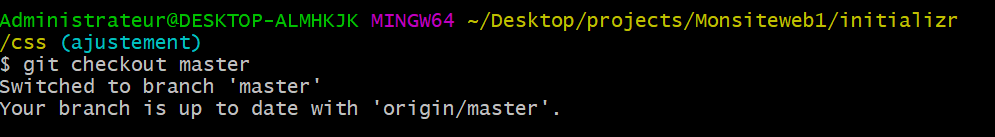


## Essayant de faire le merge en local ! @ vos mains !

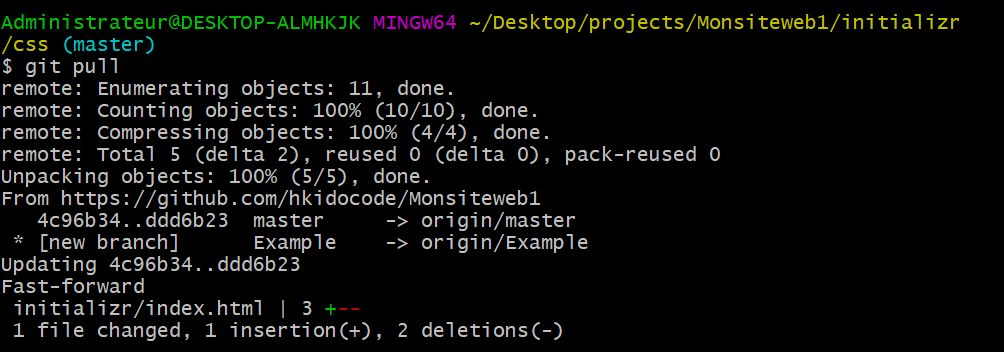
### 1. Sur GITBASH, Vérifiez dans quelle branche vous êtes maintenant ! surement sur la branche ‘ajustement’

### 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 ?

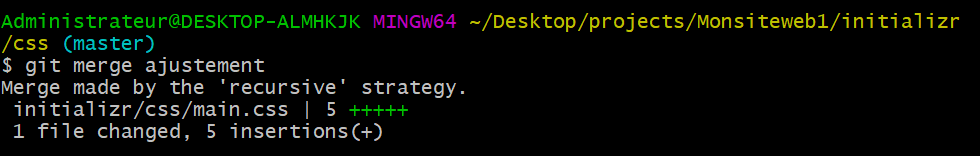


### 4. Maintenant demandez le pull ?

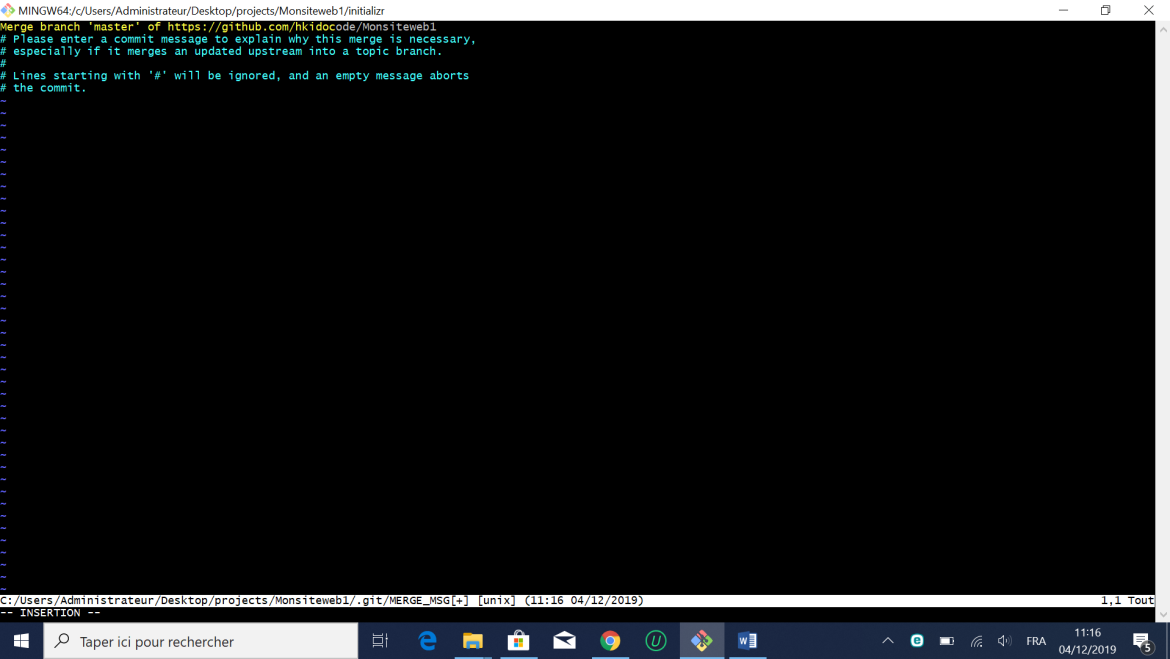


### 5. Une fois la commande du pull est exécutée, à votre avis quel type de merge on aura besoin pour faire un merge sans conflits

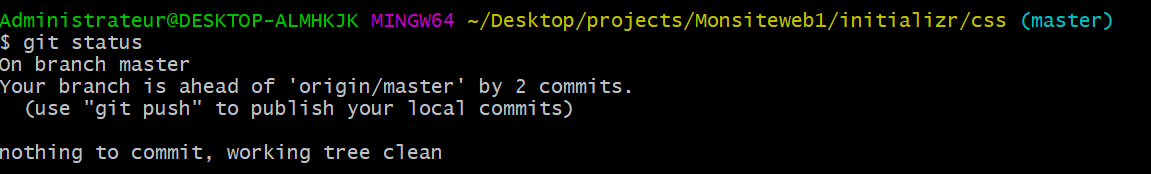
### 6. le merge à travers GITBASH



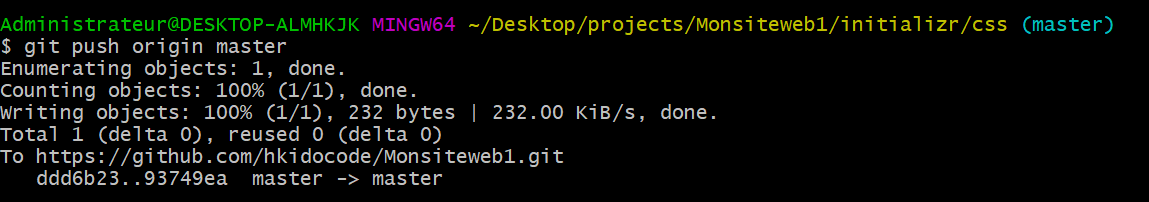
### 7. Une fenêtre s’ouvre après l’exécution de la commande du merge ; ce qui signifie le message délivré ? à quoi sert cette fenêtre ?



### 8. Git status :



### 9. Si tout est OK ! Pushez les changements ?



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

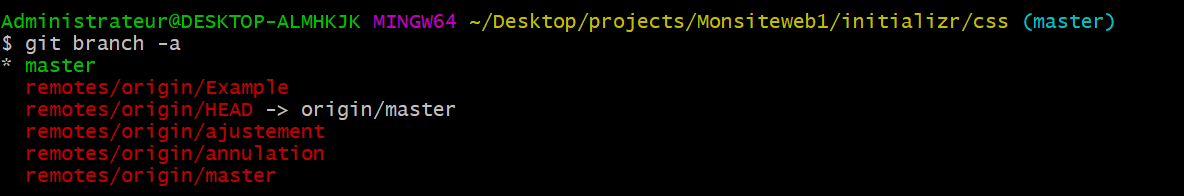
1. Tapez la commande git branch –a ? expliquez le message ?



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



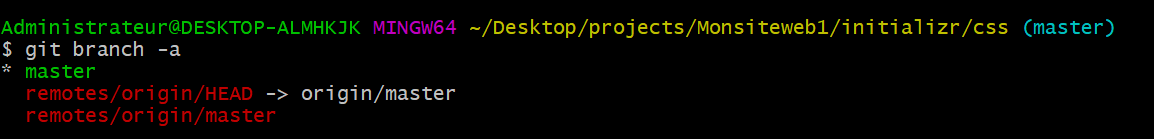
3. Maintenant ! Tapez la commande git bransh –a ? qu’est-ce que vous constatez ?



4. Ce que nous devons faire maintenant, est de faire appelez la suppression faites sur GITHUB et l’appliquez sur le local ! Trouvez la commande et exécutez ?



5. Tapez la commande git bransh –a ? qu’est-ce que vous constatez ?



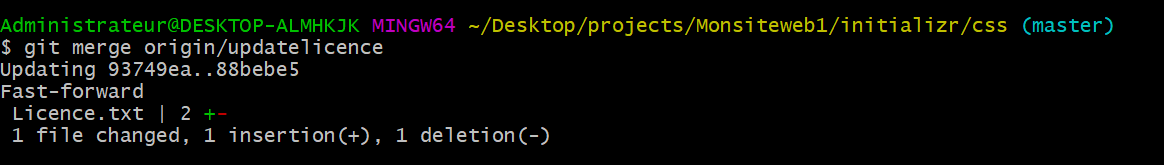
# **5 : fifth Step : The Cleaning up**

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

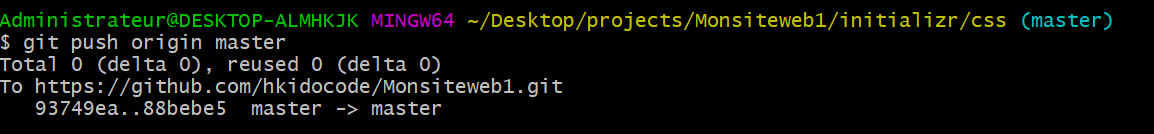
Sur GITBASH faites un pull global : git pull –-all



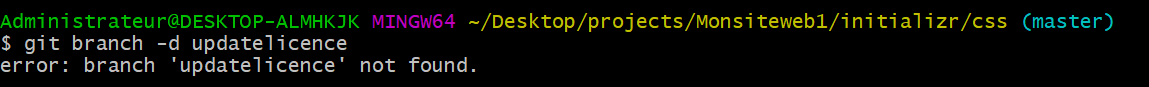
Le merge de la branche ‘updatelicence



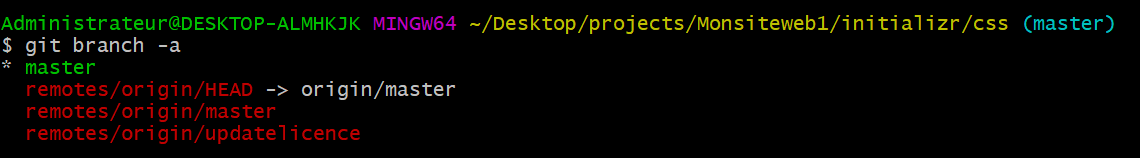
Exécutez Git push !



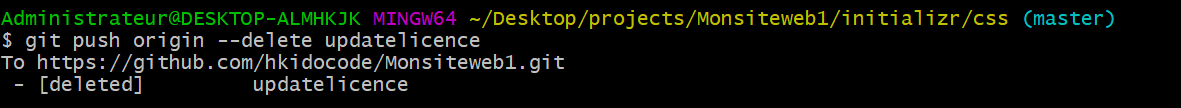
Git branch –d ‘updatelicence’



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



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)

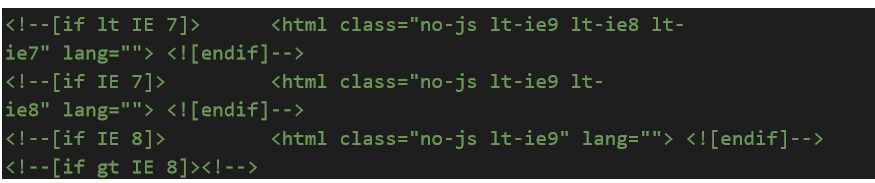


Pourriez-vous expliquez l’objectif de cette phase ?

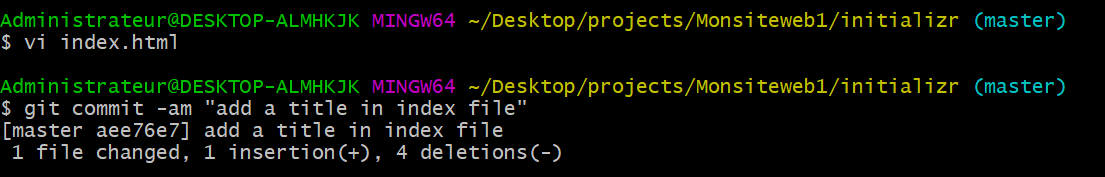
**6 : sixth step : Rebasing :**

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

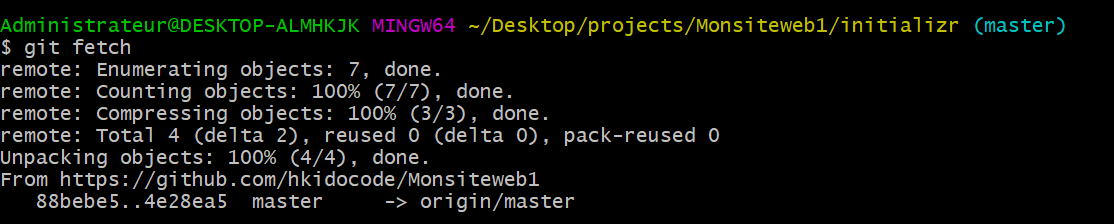
Sur GITBASH Editez le fichier index.html supprimez les lignes suivantes



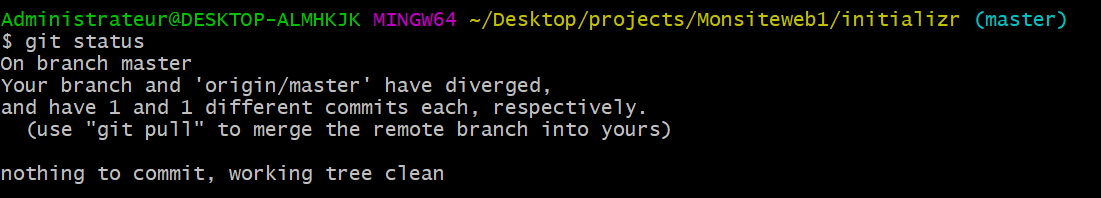
Et modifiez le titre du site : choisissez le titre que vous vouliez &Le stagging et le commit en une seule ligne



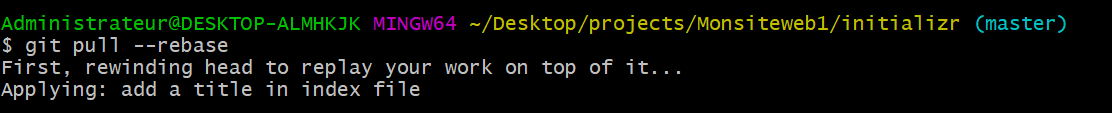
Maintenant apportez les changements faites sur le Repo distant : fetch



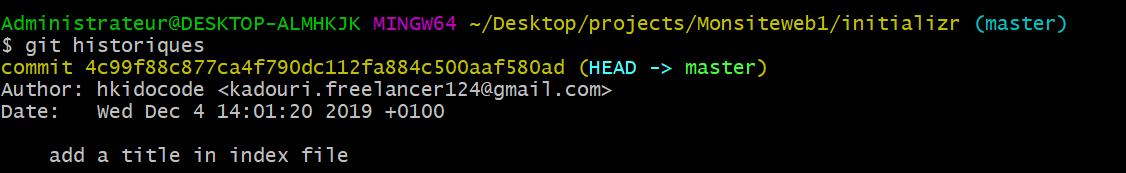
Git status ? lisez les commentaires



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 ?



Vérification de tous les changements :

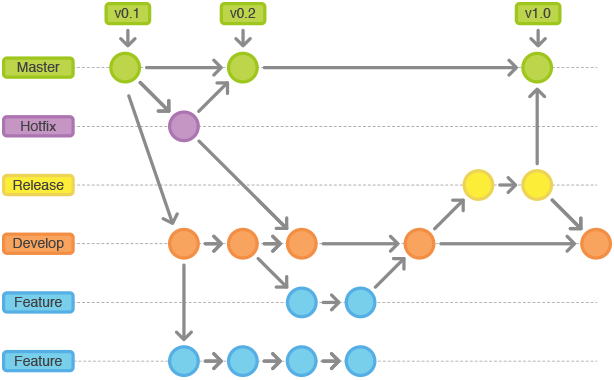
L’utilité du REBASE :

* Ils sont particulièrement utiles si nous avons besoin de corrections plus détaillées sur nos branches

**7 : seventh step GitHub Insights :**

Sur Github vous avez un menu INSIGHTS ; Explorez votre activité à travers insights :

## L’analysez du Network ou le timeline de votre workflow



La branche **master** est la branche qui va correspondre à notre environnement de production. Il est donc logique que l'on ne puisse y pousser nos modifications directement.

La branche**Develop** centralise toutes les nouvelles fonctionnalités qui seront livrées dans la prochaine version. Ici, il va falloir se forcer à ne pas y faire de modifications directement. Dans le cadre d'un gros projet, la branche Develop correspond en général à notre environnement de recette. L'environnement de recette est une copie du projet qui est censée partir en production et où les testeurs vont réaliser une batterie de tests afin d'être sûrs de ne pas envoyer de bugs en production.  
La branche master stocke l'historique officiel des versions, et la branche Develop sert de branche d'intégration pour les fonctionnalités.

Les branches**Feature** permettent de commencer à travailler sur une nouvelle fonctionnalité. La branche Feature est créée à partir de la branche Develop. Vous devrez créer pour chaque nouvelle fonctionnalité une branche Feature ! Lorsque nous avons fini de développer notre nouvelle fonctionnalité, il faudra alors la commiter puis la fusionner sur la branche Develop.

La branche **Hotfix**, quant à elle, va permettre de corriger un bug en production. En ce sens, elle sera créée à partir de la branche master, car c'est la branche master qui correspond à l'environnement de production. Une fois la branche Hotfix terminée, elle est mergée dans la branche Develop et dans la branche master.

La branche**Release** est créée à partir de la branche Develop en cas de livraison en production imminente. En effet, dans le cadre d'un projet un peu plus conséquent, il y a souvent plusieurs versions. Une fois que toutes les fonctionnalités d'une version ont été créées, c'est à ce moment que nous devons créer une branche Release. Elle va permettre de réaliser nos tests alors que d'autres développeurs pourront commencer à travailler sur la version suivante. Lorsque la branche Release est terminée, nous devons la merger dans la branche Develop et dans la branche master.

**8 : eigtht step : Default branch and conflicts**

Création d’une branche du nom “Dev” sur github

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

Créez une autre branche « demo », on va la considérer comme « feature » (sur github).

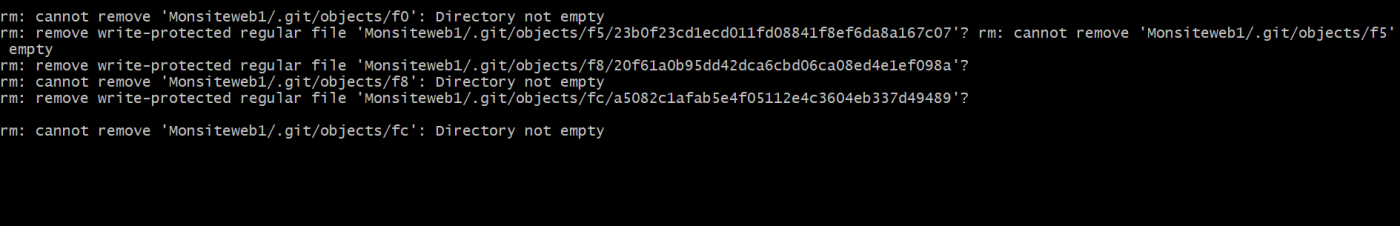
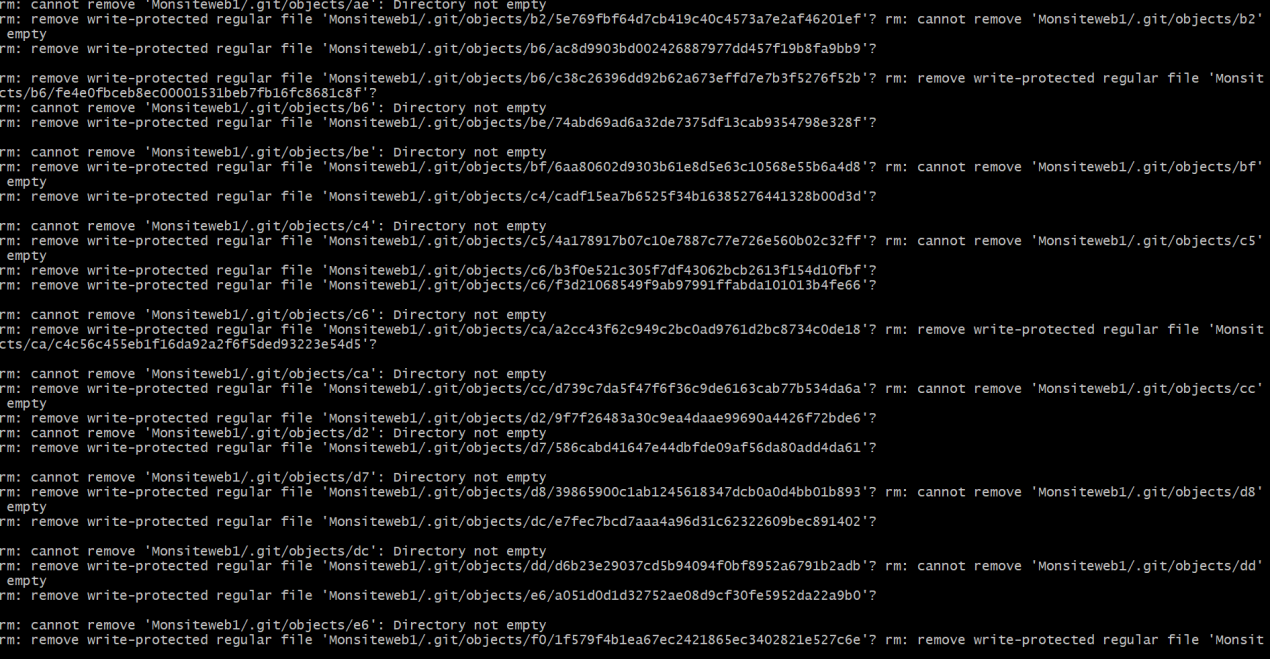
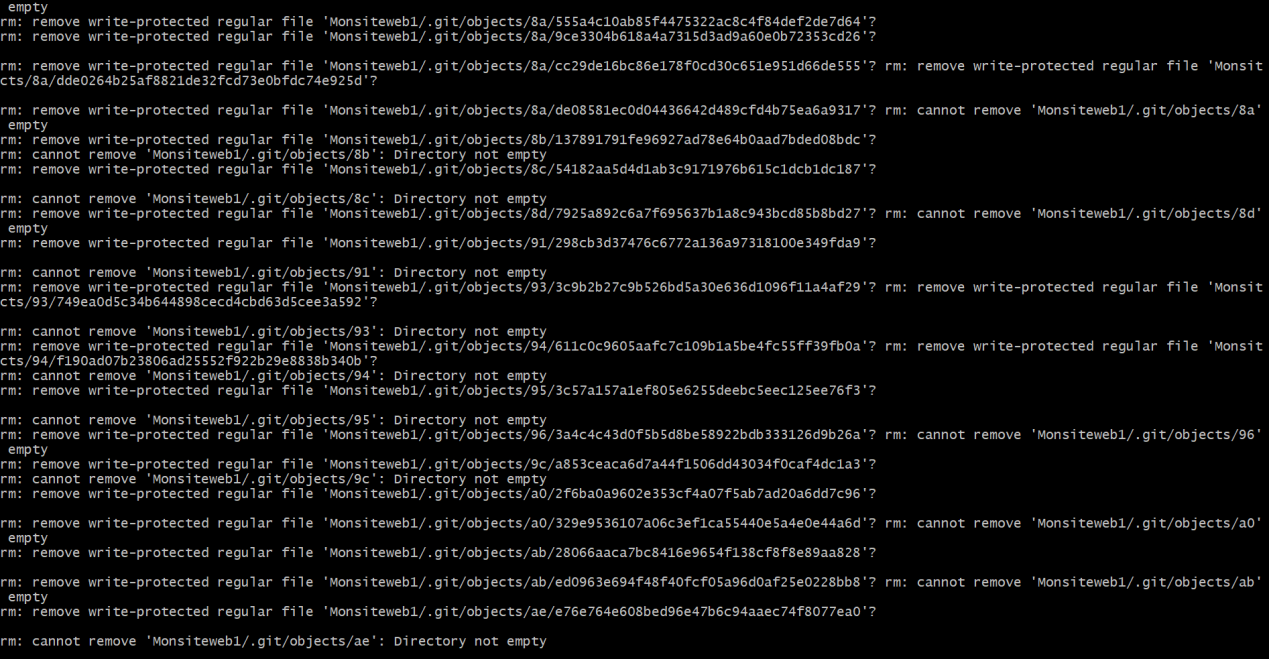
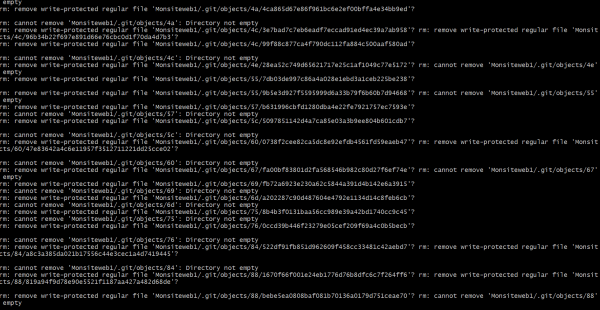
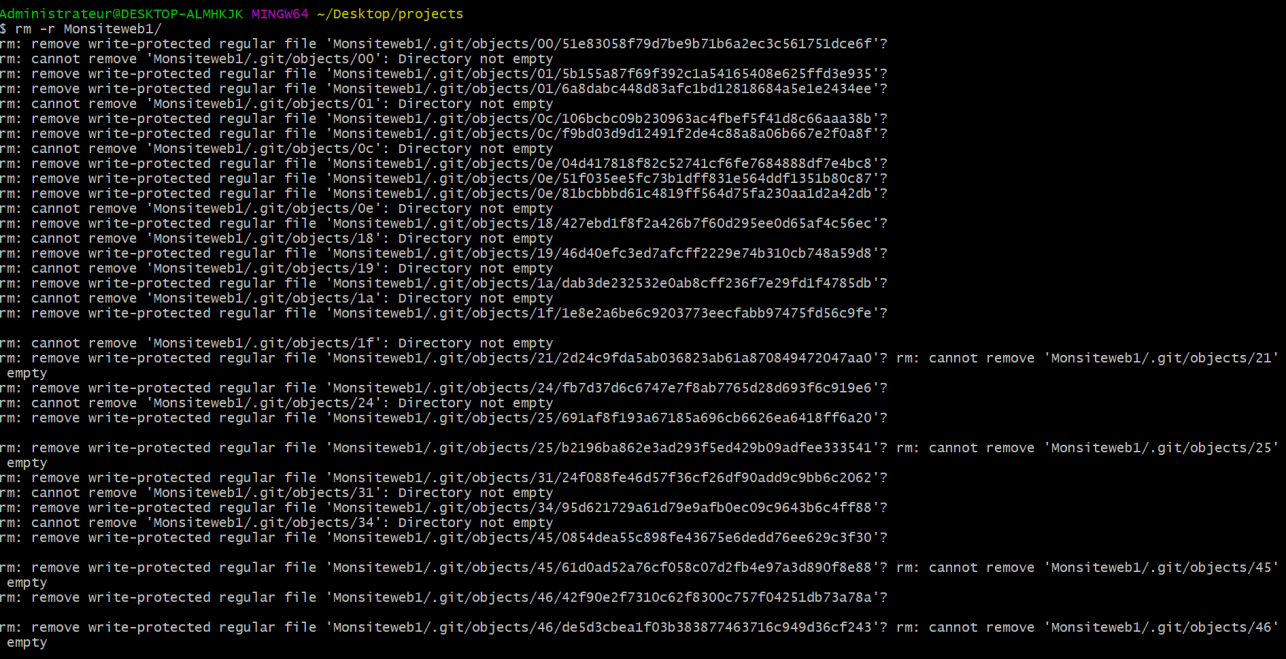
Avant de faire un pull Request sur le branche demo , Editez le fichier « README.md » , Ajoutez une ligne (sur github)

Revenez à la page d’accueil de votre repo ? à votre avis on a besoin dans ce cas de faire un merge ou bien un rebase ? pourquoi ?

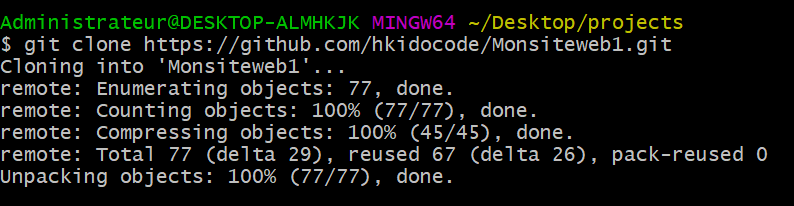
* Rebase

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



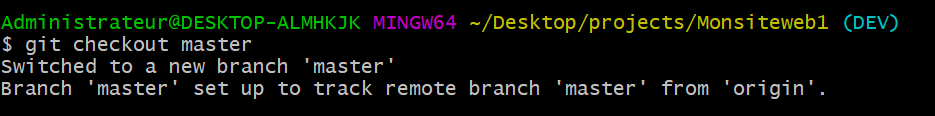
Vérification si le contenu est récupéré

* Par la commande **ls –al**

Vérification des branches et ou pointe le pointeur : HEAD

* Par la commande **git branch**

La commande git checkout master



Toutefois la branche par défaut est : ………………

* DEV