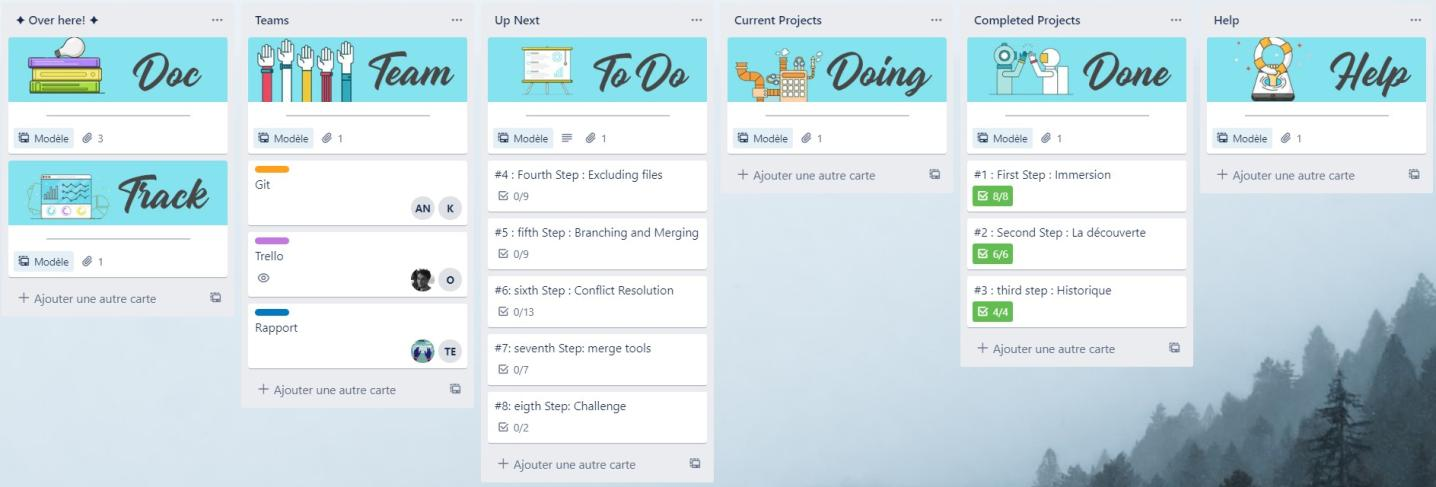
**Git en Quelques mots**

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé en 2005 par [Linus Torvalds](https://fr.wikipedia.org/wiki/Linus_Torvalds) lui-même, et distribué selon les termes de la [licence publique générale GNU](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU) version 2. En 2016, il s’agit du logiciel de gestion de versions le plus populaire, utilisé par plusieurs millions de personnes à travers le monde.

**Trello**

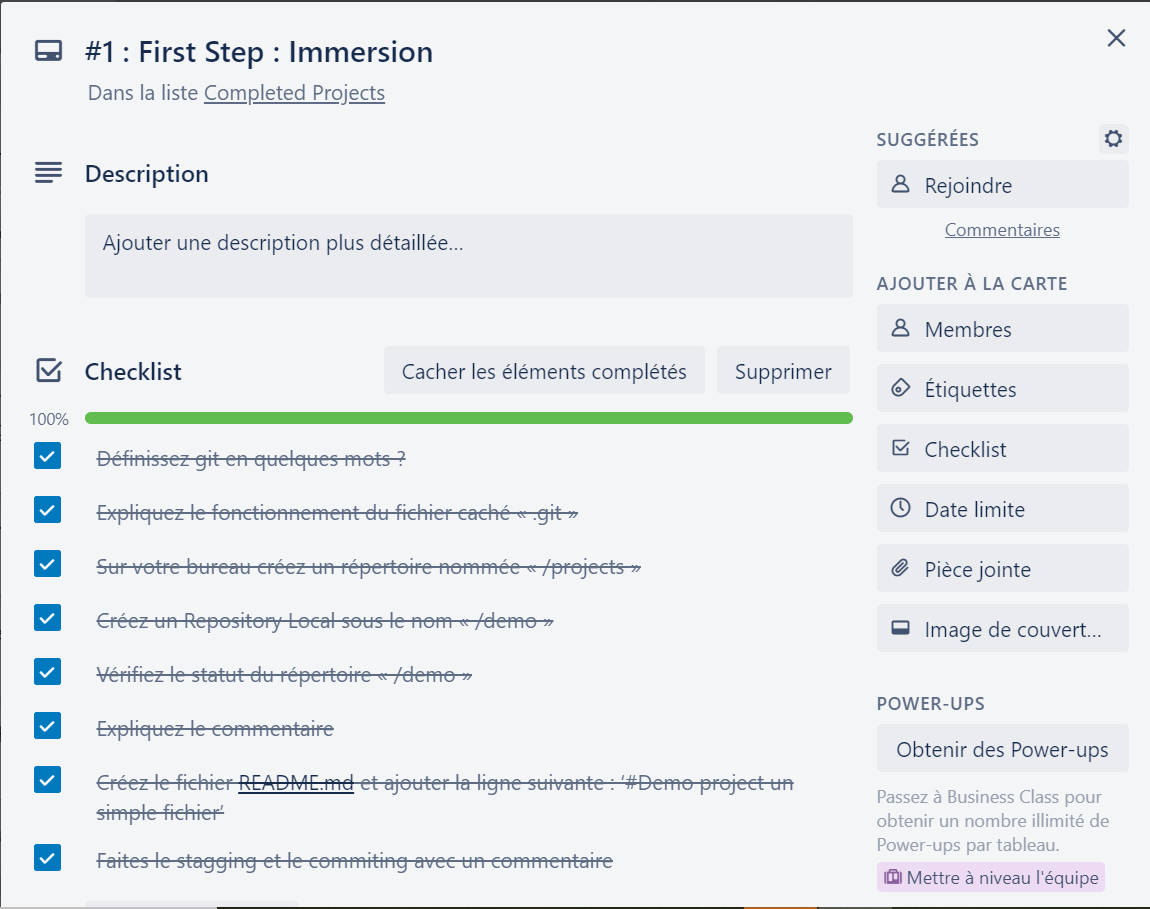
Trello est un outil de collaboration qui organise vos projets en tableaux.

Trello vous dit sur quoi on travaille, qui travaille sur quoi et où quelque chose est en cours.

****

Les *scénarios* fut répartie en *tasks et les tasks en items :*

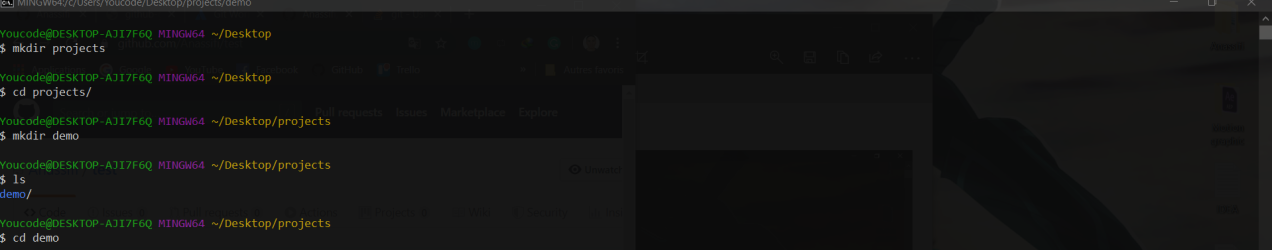
**



**#1 : First Step : Immersion**

**la fonctionnement du fichier caché .git**

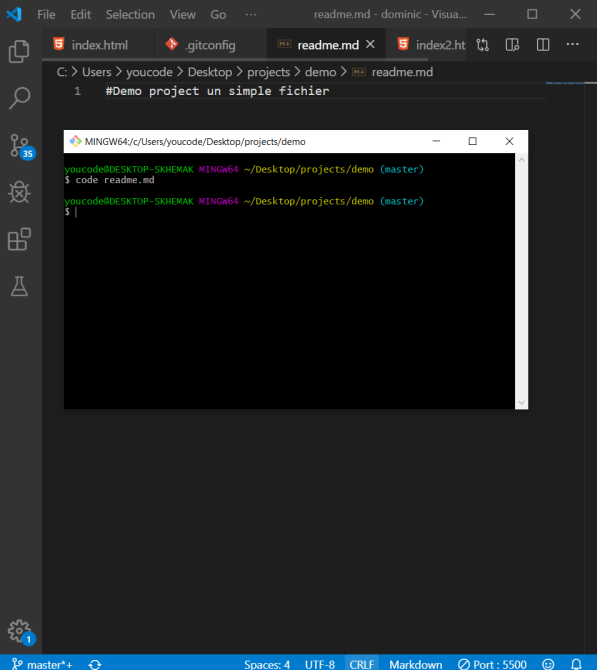
Le fichier .git contient toutes les données dont GIT a besoin pour gérer l’historique. Sauf rarissimes exceptions, vous ne modifierez jamais son contenu directement, mais uniquement en passant par les commandes GIT.

**les étape de création un repository :**

**Explication de commentaire :**

****

C’est le message d’état idéal pour Git. Être “up-to-date with ‘origin/master’.” signifie qu’il n’y a rien à pousser. “working directory clean” signifie que tous les fichiers de répertoire actuel sont gérés par git (ou sont intentionnellement ignorés via .gitignore) et que la version la plus récente du fichier a été validée.

**Creation de fichier README.md :**

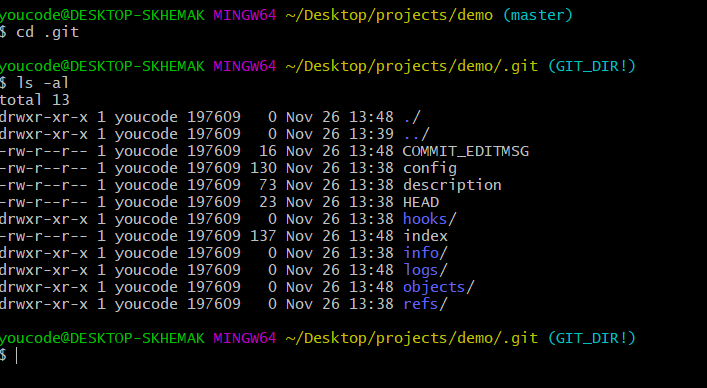
**#2 : Second Step : La découverte**

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

cd .git

**Tapez Ls –al**

.git.....ls-al



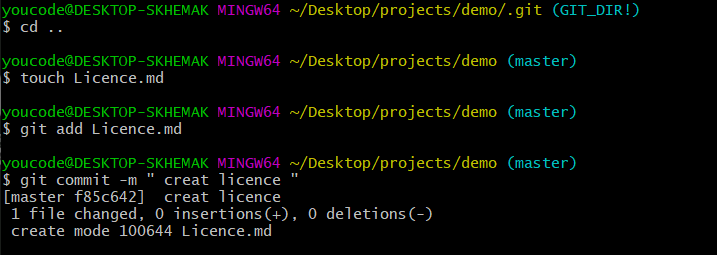
**Explications les clauses suivantes : HEAD, LOGS, BRANCHES**

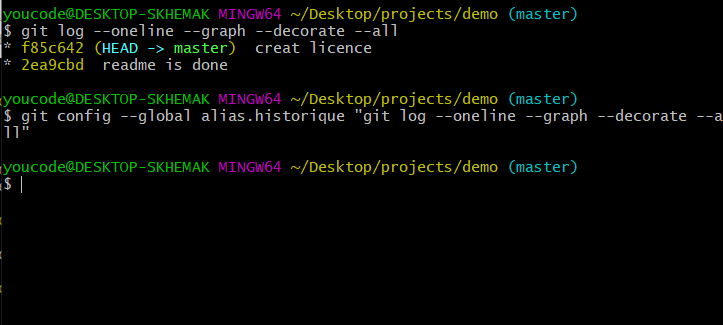
HEAD : HEAD est une référence au dernier commit de la branche de sortie actuelle. Vous pouvez considérer la tête comme la "branche actuelle". Lorsque vous changez de branche avec git checkout, la révision HEAD change pour pointer vers la pointe de la nouvelle branche.

LOGS : A Git Log est un enregistrement courant des commits. ... Métadonnées de l'auteur de validation: nom et adresse électronique de l'auteur du commit. Métadonnées de date de validation: horodatage de la date de validation. Titre / message de validation: aperçu de la validation tel qu'il est écrit dans le message de validation.

BRANCHES :Une branche dans Git est simplement un pointeur mobile léger sur l'un de ces commits. Le nom de branche par défaut dans Git est master. Lorsque vous effectuez initialement des commits, vous recevez une branche maître qui pointe vers le dernier commit que vous avez effectué. ... Plusieurs branches pointant dans l'historique des données du commit.

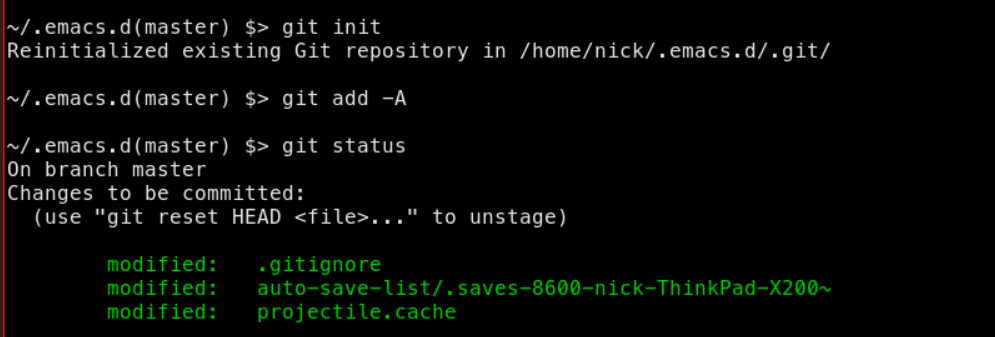
**Retournez vers votre Repo et créez le fichier « Licence.md»**



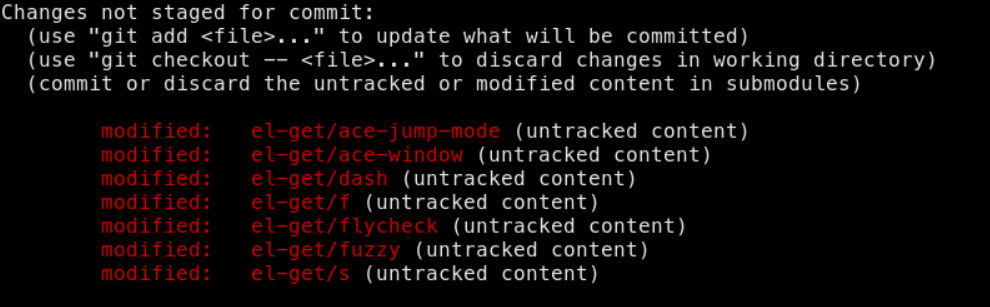
**Faites-le Commit** 

**Affichez les fichiers traqués.**

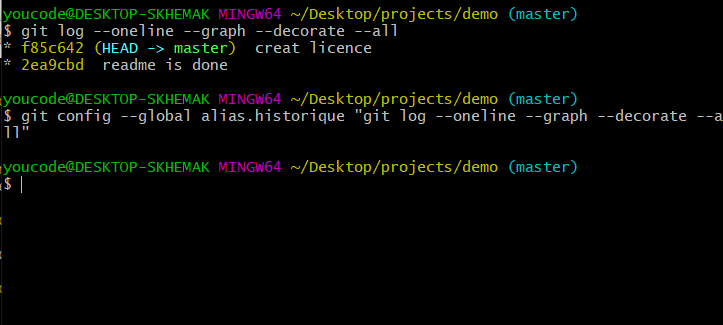
*Tracked :*

**

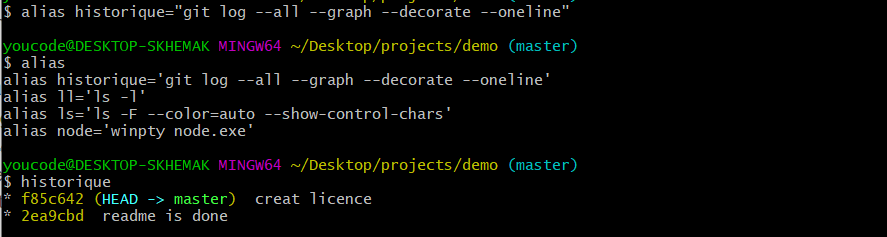
*Untracked :*

**

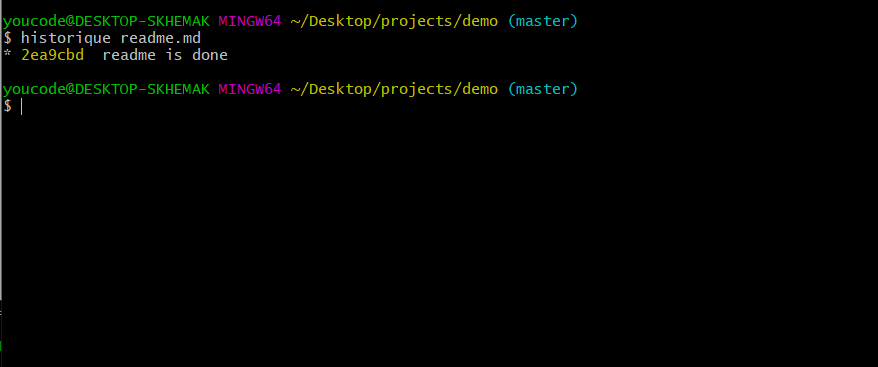
**#3 : third step : Historique**

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

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

**Affichage la liste des alias**

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

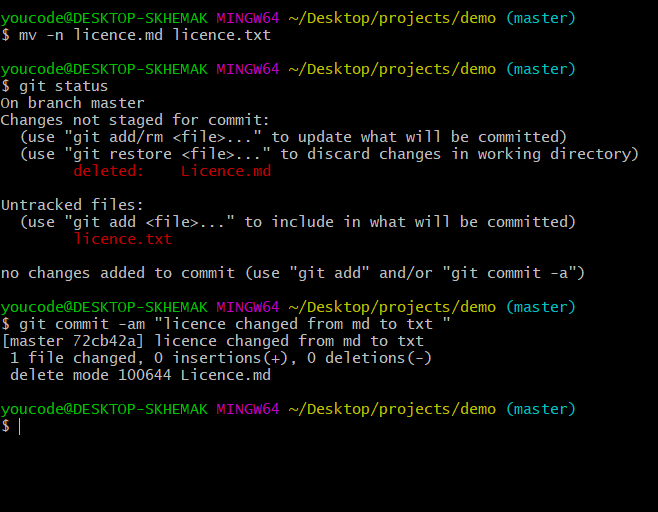
****

**#4 : 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.**

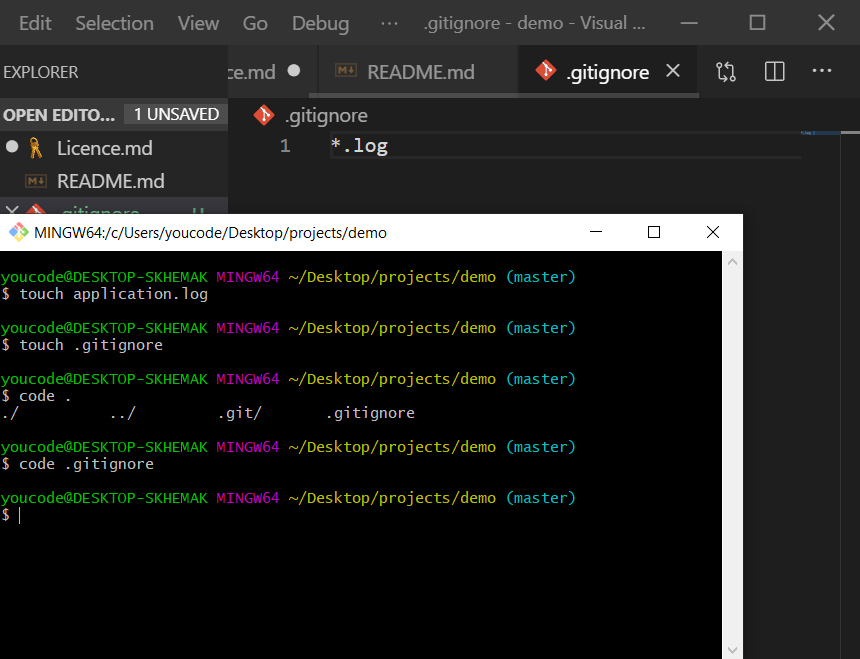
****

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

**1 Créez un fichier nommé application.log**

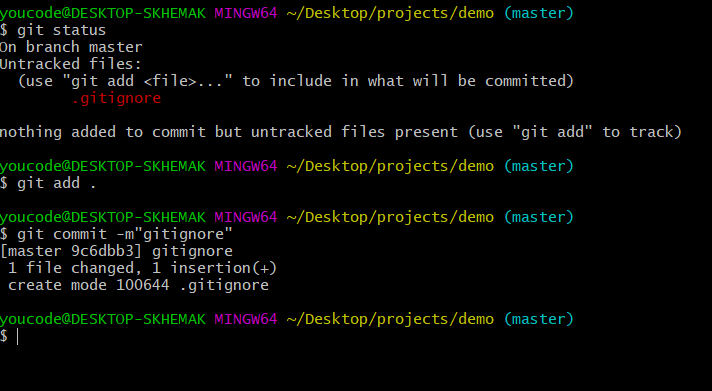
**2 Ne faites pas le stagging mais créez un fichier nommé « .gitignore »**

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

****

**4 Faites le staging et le commit**

**5 Qu’est-ce que vous constatez ?**

****

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

**The fast forward merging :** Une fusion rapide peut se produire lorsqu'il existe un chemin linéaire entre le bout de la branche actuelle et la branche cible. Au lieu de «fusionner» réellement les branches, tout ce que Git doit faire pour intégrer les historiques est de déplacer (c'est-à-dire, «avancer rapidement») de la branche actuelle jusqu'à la branche cible.

**The manual :**Une fois que les étapes de "préparation à la fusion" décrites précédemment ont été effectuées, une fusion peut être lancée en exécutant git merge <nom de branche> où <nom de branche> est le nom de la branche qui sera fusionnée dans la branche réceptrice.

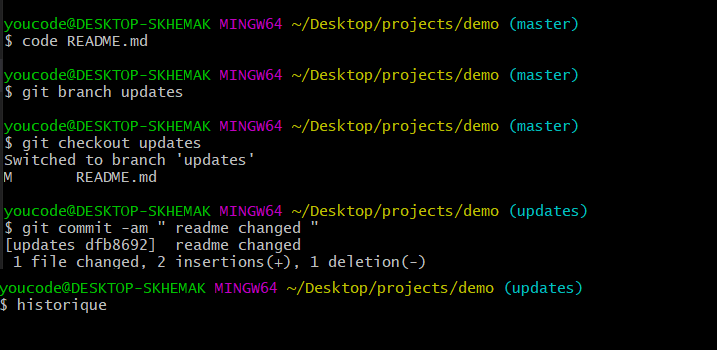
**The automatic :**La fusion Git combine des séquences de commits dans un historique unifié des commits. Il y a deux manières principales de fusionner Git: Fast Forward et Three way. Git peut fusionner automatiquement les commits, sauf si des modifications entrent en conflit dans les deux séquences de validation.

**Modifiez le fichier README.md**

**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**

**Qu’est-ce que vous constatez ?** 

**Retournez vers la branche Master.**

**Affichez l’historique avec l’alias**

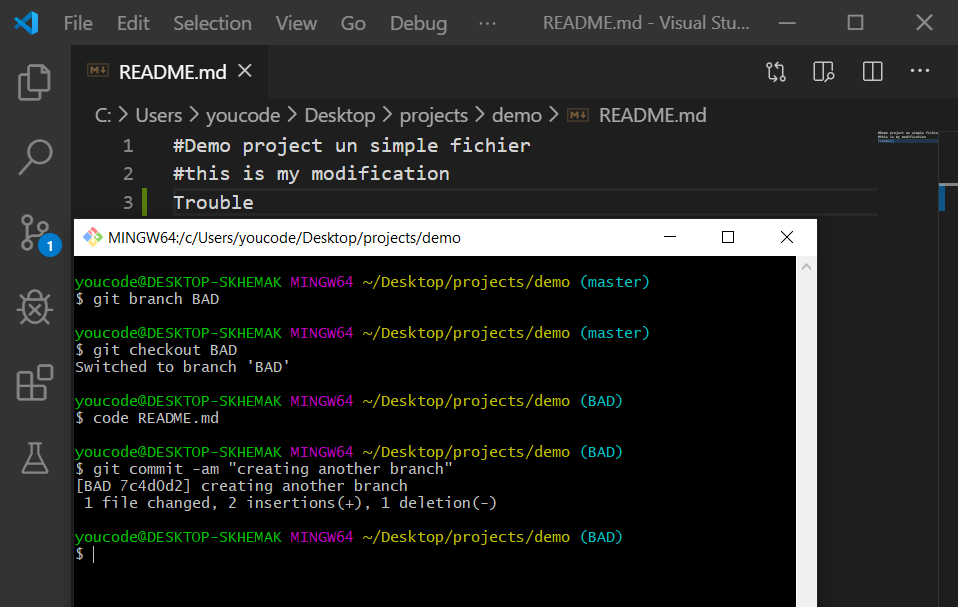
**Qu’est-ce que vous constatez**

**Maintenant faites le merge**

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

**Créez une branche avec le nom ‘BAD’**

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

**Faites le stagging et le commit en une seule ligne** 

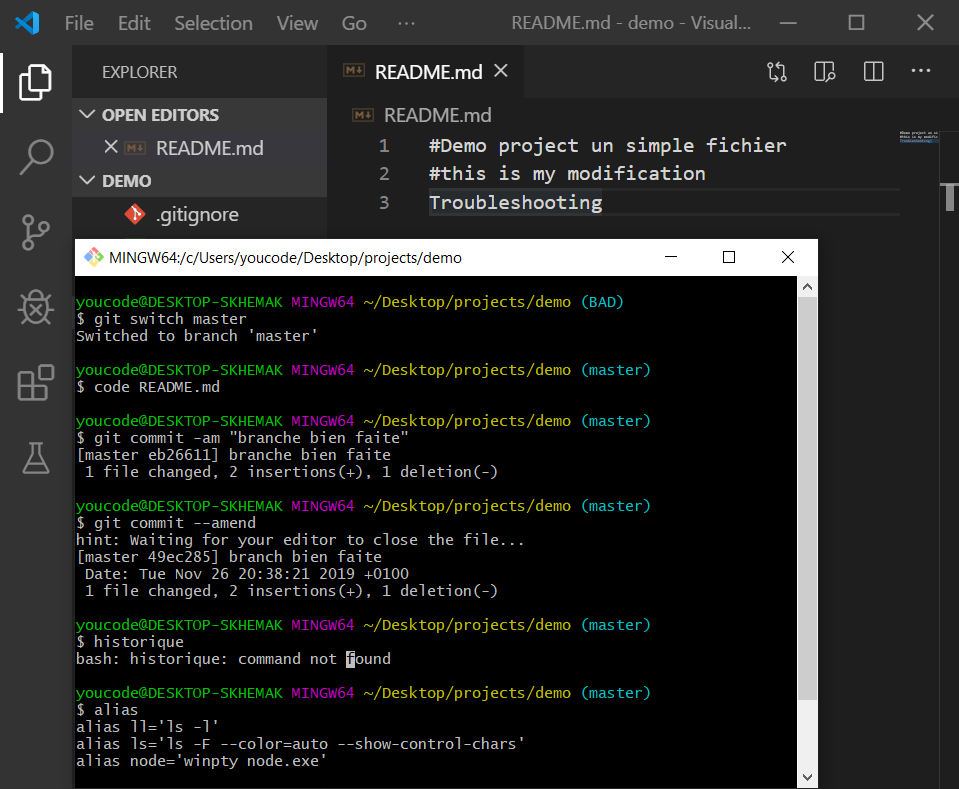
**Switchez faire la branche principale ‘master’**

**Maintenant modifiez le fichier README.md et ajoutez la ligne ‘Troubleshooting’**

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

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

**Affichez l’historique**

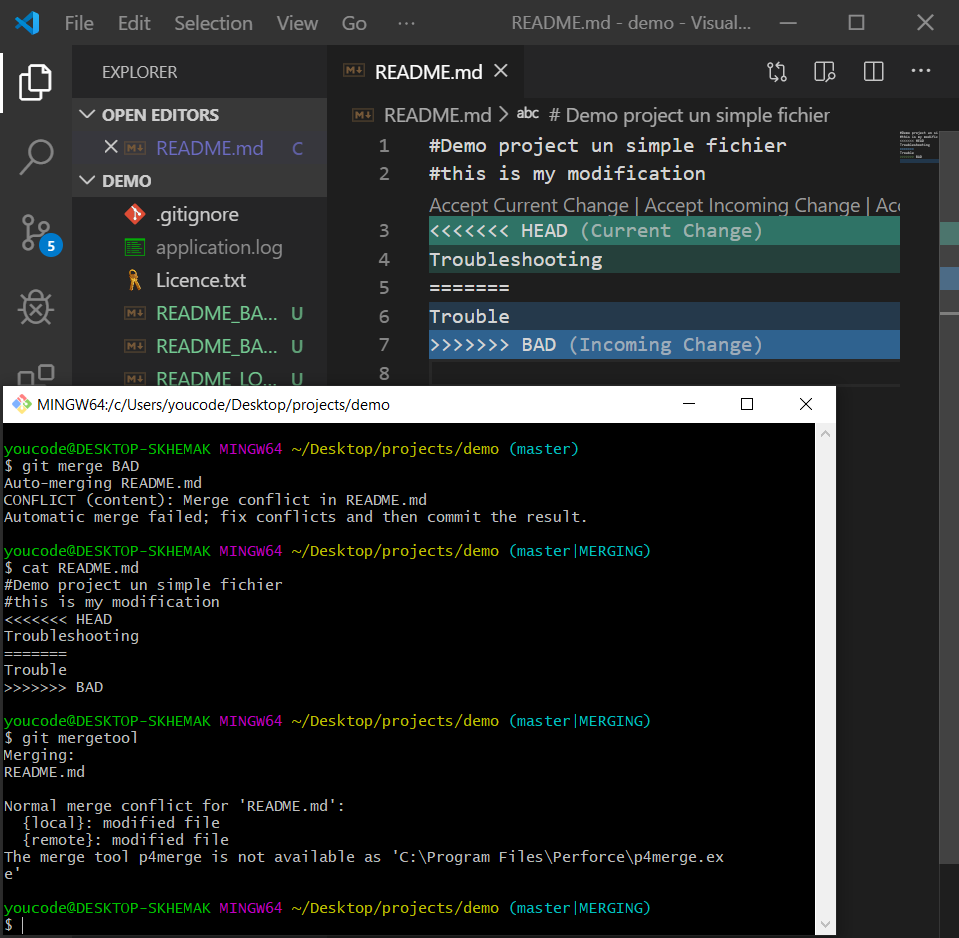
****

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

**Expliquez le message ?**

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

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

****

**#7: seventh Step: merge tools**

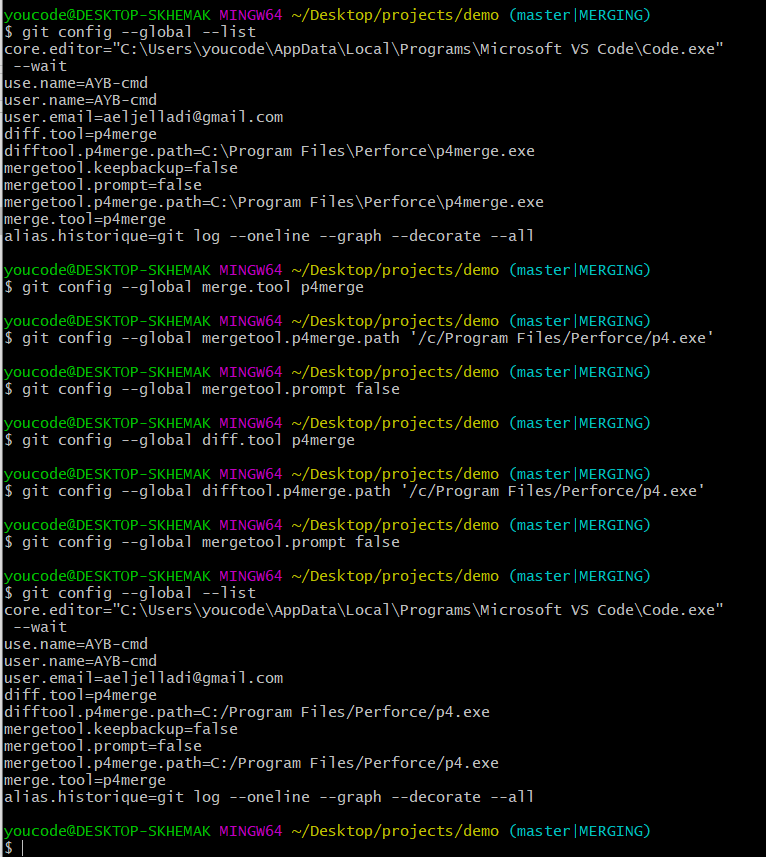
**Installez p4merge sur votre PC**

**Sur git bash tapez la commande : git config –global –list**

**Git config –global merge.tool p4merge**

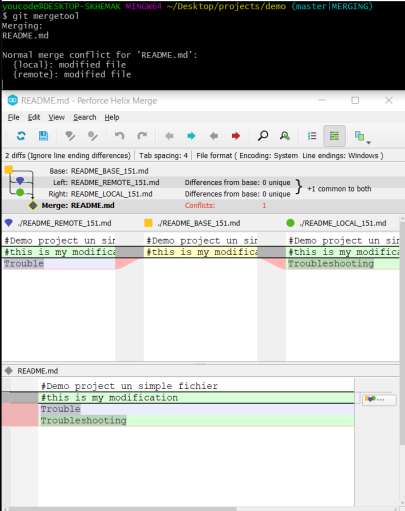
**Git config –global mergetool.p4merge.path “lien d’installation” ( .exe)**

**N’oubliez pas de bien configurer le prompt ? devinez comment**

****

**Tapez la commande git mergetool**

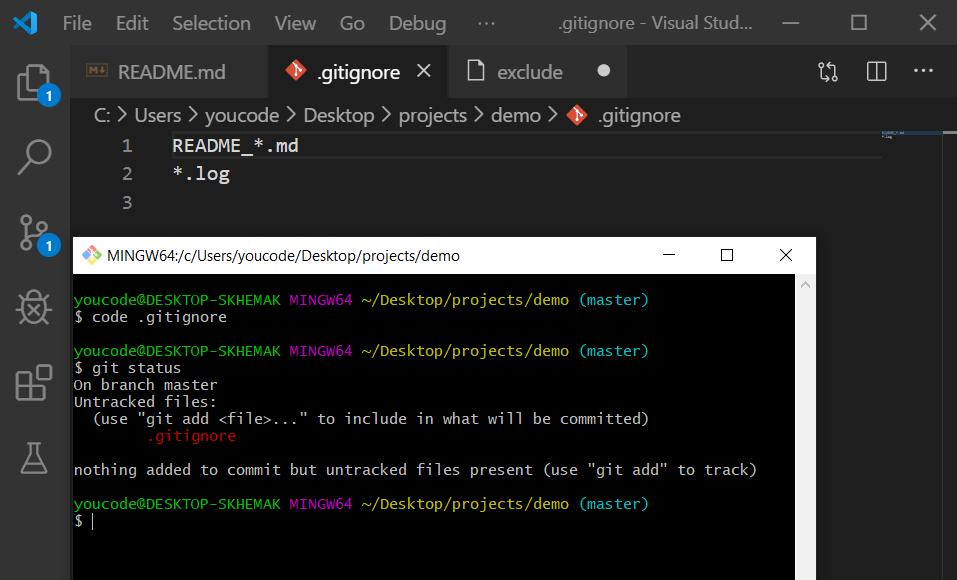
**Analysez ce que vous voyez ? et expliquez la plateforme ouverte en temps réel**

****

**#8: eigth Step: Challenge**

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

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

****