**EXERCICE PARIE 3 - Solution proposée**

1. **Qu'est-ce qu'un commit**

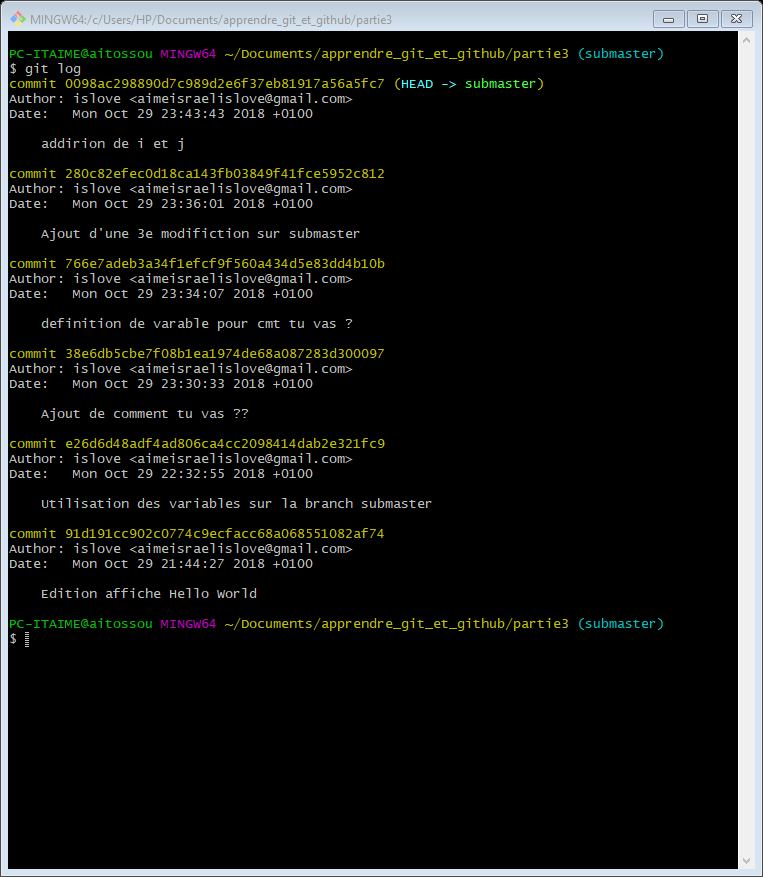
Dans tous les domaines les travaux que ce soit technique ou pas ont besoin d’être sauvegarder. Cette sauvegarde peut se faire sur un support papier ou numérique. Ainsi dans le domaine de l’informatique, notamment dans les systèmes de bases de données, de révision de fichier (par exemple du code) les termes d’archivage, de soumission, de validation ou encore l’anglais **commit** désignent l’enregistrement. Cet enregistrement entérine l’exécution de la tâche préalablement confiée, marquant à la fois la fin de la demande de transaction et le début de l’exécution de la tâche confiée, qui devra être exécutée automatiquement. Git est un système de contrôle de version distribué gratuit et en licence open source. Il a été conçu pour traiter aussi bien des petits ou de très gros projets avec rapidité et efficacité.

Commande : **git commit -a -m "message du commit"** (permet de faire les commits. L’option -a permet d’ajouter à l’index et l’option –m permet ajouter un message détaillant le commit).

1. **À quoi sert la commande : git log**

Après avoir créé plusieurs **commits** (sauvegardes) ou si vous avez cloné (copié) un dépôt ayant un historique de commits, vous souhaitez probablement revoir le fil des évènements. Pour ce faire, la commande **git log** est l'outil le plus basique et le plus puissant. Ainsi on dira que git log permet d’afficher toutes les sauvegardes, enregistrements effectués. Pour bien illustré ces propos voici une sauvegarde en capture d’écran.

Commande : **git log**



Il est clair sur cette figure que le git log en plus de donner l’information sur la date et l’heure, renseigne aussi sur l’auteur et fourni même son email. Remarquer que devant commit il y a un SHA une suite de lettres et de chiffres pour identifier de façon unique le **commit.**

1. **Qu'est-ce qu'une branche**

La branche dans Git est un concept trop **important**. Les branches permettent d'avoir un historique non linéaire et permettent de travailler pendant un moment sur une fonctionnalité particulière sans pour autant "polluer" (encombrement inutile) le flux principal (la branche maîtraisse). Elles peuvent servir pour travailler sur des fonctions spéciales mais aussi être utilisée pour garder une bonne organisation avec des branches qui correspondent à nos différents environnements (**master**, phpmaster, dev, bugfix...).

La commande branch permet de gérer tout ce qui a attrait aux branches (ajout, listing, suppression, renommage).

Voici un exemple de branches expliquées

**<disk-jaune>** La branche **master** (master est une convention de nom pour la branche principale) crée quand on initialise le repository, c’est automatique cette création.

Commande : git init

Les branches **<disk-jaune>** et **<disk-orange>** est un exemple de fonctionnalité que le développeur peut vouloir explorer.

Commande : git branch nom-de-branche (Permet de lister les branches) et git branch –b nom-de-branche (Permet de lister les branches et de s’y positionner)

**<disk-vert>** Cette nouvelle branche est la fusion de la branche master **<disk-jaune>** et **<disk-jaune>** de la branche.

Commande : git merge master (cela signifie que la branche courante est **<disk-jaune>**)

Ainsi la branche master est fusioné dans la branche **<disk-jaune> .**

Autres commandes:

git branch <ma-branche> **# Permet de créer une nouvelle branche <ma-branche>**

git branch -m <rename-branche> **# Renomme la branche courante en <rename-branche>**

git branch -d <ma-branche> **# Permet de supprimer une branche.**

Annexe :

Dans capture master est la branche principale et submaster est une autre branche. Remarquer le mot « HEAD » qui indique qu’elle est la branche courante.

Il faut noter qu’une branche reste une branche qu’elle soit master ou pas. Le nom master juste une convention de nom pour se retrouver quand on en a plusieurs.