OpenClassRoom : Git & Github

Partie 3 Activité

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

Faire un *commit* permet d'ajouter les dernières modifications apportées au code d'un projet au *repository* de ce projet. Généralement, le développeur fait un *commit* lorsque les nouvelles fonctionnalités qu'il a développées sont fonctionnelles, et évitera de faire des *commits* lors des états d'avancement intermédiaires où le code n'est pas fonctionnel en l'état.

Un *commit* peut être placé (*push*) sur un serveur distant (*remote*) afin qu'un tierce personne puisse récupérer la dernière version du code. Cela est utile lorsque plusieurs personnes travaillent sur un même projet avec du code en commun.

Une autre utilité des *commits* est de pouvoir rappeler d'anciens *commits* : le code est alors restauré à l'état où il se trouvait lors de cet ancien *commit*. Cela est fort utile dans le cas où, par exemple, les derniers développements ont abouti à un code dysfonctionnel.

Pour effectuer un commit avec un commentaire « mon commentaire » :

```
$ git commit -m "mon commentaire"
```

2. À quoi sert la commande *git log* ?

La commande *git log* permet d'afficher une liste des *commits* qui ont été effectués sur le *repository*, comme un historique. Cette liste contient pour chaque *commit* :

- Le *SHA* du *commit* : c'est une emprunte qui contient 40 caractères, et qui permet d'identifier un *commit*.
- Le nom et l'adresse mail de l'auteur du *commit*
- La date et l'heure du *commit*
- Le commentaire laissé par l'auteur du *commit*

Pour afficher l'historique des *commits* :

```
$ git log
```

3. Qu'est-ce qu'une branche ?

Par défaut, lorsque nous travaillons avec un *repository*, nous nous trouvons (sans forcément le savoir) sur une branche appelée *master*. On peut imaginer cette *branche* comme le tronc d'un arbre : c'est la version principale de notre code, sans doutes celle où se trouveront les versions amenées à être publiées (*release*).

Il arrive néanmoins que nous voulions faire des expérimentations sur notre code, mais sans toucher à la branche *master*, par exemple pour ne pas risquer de la rendre dysfonctionnelle, ou encore pour qu'un autre développeur puisse travailler dessus sans que nos tests n'affectent sa base de travail.

Pour cela git nous donne la possibilité de créer une branche : cela nous permet de partir du code de la branche *master*, et de faire des développements parallèles sans que celle-ci ne soit affectée. Si ces expérimentations sont concluantes, le développeur peut choisir de les appliquer à la branche *master* en fusionnant la branche qu'il a créé avec la branche *master*. La commande pour faire cette opération est *merge*.

Pour afficher les branches existantes sur le *repository* :

```
$ git branch
```

Pour créer une branche appelée « ma-branche » :

```
$ git branch ma-branche
```

Pour se positionner sur cette branche :

```
$ git checkout ma-branche
```

Pour créer la branche et s'y positionner en une seule commande :

```
$ git checkout -b ma-branche
```

Il est important de noter que *master* est en fait une branche comme les autres, c'est l'usage qui veut qu'elle soit notre branche principale.