Git_GitHub_Partie3_Activite

Consigne:

Le but est d'expliquer trois concepts de GIT pour un ami développeur web qui ne connait pas GIT.

Qu'est-ce qu'un commit

```
git commit
```

À quoi sert la commande git log

```
git log
```

Qu'est-ce qu'une branche

```
git branch develop
```

Introduction:

Pour rappel : GIT est un outil de "Versioning" (suivi de versions), qui permet de "suivre" les modifications apportées aux fichiers indexés dans un projet.

Tout d'abord, comme nous allons tenter d'expliquer le fonctionnement de GIT, j'ai crée dans une branche "develop", et dans cette branche , un fichier README.md (fichier important dans un "dépôt" GIT car c'est celui qui va donner le contenu d'un projet géré sur GITHUB)

Tous les fichiers demandés sont intégrés au repository "intro-git-github":

1. https://github.com/NLshaen/intro-git-github/blob/develop/Git_GitHub_Partie3_Activite.md
2. https://github.com/NLshaen/intro-git-github/blob/develop/Git_GitHub_Partie3_Activite.pdf
3. https://github.com/NLshaen/intro-git-github/blob/develop/Git_GitHub_Partie3_Activite.zip

Qu'est-ce qu'un commit ?

```
git commit
```

Pour expliquer ce qu'est un commit, reprenons la petite séquence suivante qui permet de créer un nouveau commit :

Lorsqu'on suit un projet avec "git", par exemple ici la réponse à un exercice, les modifications apportées aux fichiers sont listées dans le projet, et peuvent être affichées par la commande qui affiche l'état actuel du répertoire de travail du projet par rapport à l'index de "GIT":

```
git status
```

Cela signifie à ce stade que ces modifications existent bien dans l'état actuel du projet (si on l'affiche sur un serveur local, les modifications faites sont bien visibles), mais qu'elles n'ont pas encore été "validées" pour le prochain "commit".

Nous ajoutons tous les fichiers modifiés du répertoire courant à l'index, avec la commande:

```
git add .
```

Une variante de cette commande permet d'ajouter de nouveaux fichiers à l'index (ou de prendre en compte les fichiers supprimés):

```
git add -A .
```

Le "commit" lui-même est créé par la commande suivante, qui va créer un nouveau commit, et ajouter un commentaire pour décrire la raison et/ou le contenu des modifications:

```
git commit -m "Add file Git_GitHub_Partie3_Activite.md"
```

Un commit est un état précis des fichiers du projet indexés avec GIT. Cet état est identifié par une chaine de caractères réputée unique (un hash).

Un commit peut aussi bien consister en la modification d'un seul caractère, que l'implémentation d'une nouvelle fonction, cela ne dépend que de la manière de les utiliser !

À quoi sert la commande git log ?

```
git log
```

La commande "git log" permet d'afficher la liste des derniers commits de la branche ou nous nous trouvons (historique des dernières modifications).

L'historique précise l'auteur, la date et le commentaire, le hash , associé à chacun des "commits", et dispose de diverses options qui permettent d'affiner les résultats sur une période ou un répertoire précis.

Qu'est-ce qu'une branche ?

```
git branch develop
```

Une branche est une dérivation à partir du "tronc commun" d'un projet, avec lequel elle peut ensuite être fusionnée.

C'est une manière de cloisonner certains développements, comme par exemple le développement d'une nouvelle fonctionnalité ou d'un patch, mais aussi de maintenir et faire évoluer en parallèle plusieurs versions d'un même logiciel.

Une branche constitue un espace de travail isolé dans lequel il est possible d'apporter diverses évolutions sans perturber le développement de la partie principale du code.

Les branches peuvent servir aussi bien seul qu'en collaboration, en permettant de structurer et d'organiser le travail sur différentes parties d'un projet, ou à plusieurs niveaux de maturité de ce projet.

Il est possible de passer d'une branche à l'autre afin de travailler successivement dans plusieurs parties d'un projet, ou d'avancer à la fois sur des tâches urgentes, tout en progresssant sur des projets à moyen terme.

Ce qu'il faut bien noter, c'est que le répertoire de travail sera toujours dans l'état de la branche sur laquelle on est positionnée : si on repasse sur la branche "master" après avoir travaillé dans une autre branche, le contenu des fichiers sera modifié pour réprésenter l'état de la branche "master".

Le processus de création, travail et fusion d'une branche peut être résumé avec les commande suivantes :

Création d'une nouvelle branche "develop" (pour du development par exemple) et on vient se positionner sur la branche "develop"

```
git checkout -b develop
```

On travaille ensuite dans cette branche "develop", les "commits" étant dès lors enregistrés dans cette branche.

Pour changer de branche, on utilise

```
git checkout master
```

Une fois le travail dans une branche terminé, on se repositionne sur la branche d'origine pour y intégrer son travail, et nous faisons

un merge (fusion) des branches develop sur master.

```
git checkout master
git merge develop
```

Attention aux conflits :)