Utilisation de Git et GitHUB

Généralités

Pourquoi versionner votre code ?

Lorsque vous travaillez sur un projet de code, vous allez régulièrement y apporter des modifications, et par moments ces modifications vont provoquer des bugs. Lorsque vous revenez sur votre projet après quelques jours ou même quelques heures, il peut être difficile de vous souvenir des dernières modifications que vous avez effectuées et de retrouver vos repères dans votre code.

Définitions :

Un **commit** correspond donc à une **version** de votre code à un instant t.

La somme de tous les commits constitue l'historique de votre projet.

Utiliser un logiciel de versioning est considéré comme une habitude indispensable pour tout développeur digne de ce nom !

Installer Git

Rendez-vous sur https://gitforwindows.org/ et téléchargez la dernière version disponible. Une fois le fichier récupéré, lancez-le et suivez les instructions. Vous pouvez laisser toutes les configurations par défaut sauf pour l'éditeur.

Exécutez la commande suivante pour définir votre nom et l'email que vous utiliserez ensuite pour créer un compte gratuit sur Github:

```
git config --global user.name "Votre nom ou pseudo"
git config --global user.email "votre@email.com"
```

Pour vérifier que tout va bien, relancez votre console et tapez simplement 'git'. Si l'installation a fonctionné, vous devriez voir du texte en anglais expliquant l'utilisation de Git.

Petite astuce : utilisez la touche "Insert" de votre clavier pour coller du texte dans git bash (ctrl + v ne fonctionnera sur Windows)

Créer un Git

Créer un dossier qui contiendra votre repository : soit par l'explorateur de fichier soit par bash (la console de git, langage :linux)

- Pour activer un dossier comme repository git, Se placer dans ce dossier avec la console bash, puis taper

```
git init
```

- Pour gérer un repository, Git génère un index de tous les fichiers dont il doit faire le suivi. Lorsque vous créez un fichier dans un repository, vous devez donc l'ajouter à l'index Git à l'aide de la commande **git add nomDeVotreFichier.extension**. Par exemple :

```
git add checklist-vacances.md
```

Pour gagner du temps, vous pouvez ajouter tous les fichiers dans le répertoire courant en tapant

```
git add .
```

dans la console. Évidemment, faites bien attention quand vous utilisez ce raccourci à ne pas rajouter trop de fichiers à l'index.

- Lorsque vous modifiez votre repository, vous devez demander à Git d'enregistrer vos modifications en faisant un **git commit**. L'option `-m` vous permet de lui envoyer un message décrivant les modifications effectuées, ce qui s'avèrera très utile pour vous par la suite, you'll see! Par exemple :

```
git commit -m "Ajouté ma checklist-vacances.md (woohoo!)"
```

Visualiser les fichiers d'un commit

Pour revenir à une version de fichier entérieur.

Changer le pointeur du git local en tapant (préciser le code SHA du commit souhaité)

```
git checkout d986a1eb5bc574fb771601fd465b2dc7cd00bfb6
```

Double cliquer sur votre fichier comme habituellement

Pour revenir à la version actuelle

```
git checkout master
```

Gérer un git

Comment vous y retrouver dans l'historique de vos commits ?

Grâce à la commande `git log` qui vous affiche la liste de tous les commits que vous avez réalisés !

Le sens de lecture du log est le suivant : on part du commit le plus récent en haut de la liste, au commit le plus vieux en bas de la liste.

Le tip du chapitre

Petite astuce avant de passer à la suite si vous voulez gagner du temps, maintenant que vous êtes parés pour créer des milliers de commits sans vous perdre .

Jusqu'ici, lorsque vous mettez à jour un fichier dans votre repository, vous devez procéder en deux étapes :

1. Ajouter votre fichier à l'index avec la commande `git add`,
   ```
   git add checklist-vacances.md
   ```
2. Faire un commit qui décrit la mise à jour de votre fichier avec la commande `git commit`.
   ```
   git commit -m "Ajouté itinéraire dans checklist-vacances.md"
   ```

Et bien, si vous ne faites que mettre à jour un fichier que vous aviez déjà ajouté à l'index, vous pouvez condenser ces deux étapes de la façon suivante :

```
git commit -a -m "Ajouté itinéraire dans checklist-vacances.md"
```
L'option `-a` demande à Git de mettre à jour les fichiers déjà existants dans son index. Pratique, non ?

Connecter le Git local au Git Remote

Créer un repository sur GitHub

Connecter les repository

```
git remote add origin https://github.com/AFPA1701871/monPremierRepo.git
```

Envoyer les modifications locales indexées vers le remote dans la branche master

```
git push -u origin master
```

Pour récupérer en local les modifications faites en remote

```
git pull origin master
```

Créer des branches

Pour voir les branches présentes dans votre repo, utilisez la commande **`git branch`**. Elle vous retournera les branches présentes, et ajoutera une étoile devant la branche dans laquelle vous êtes placés.

- Pour créer une nouvelle branche, il vous suffit d'ajouter le nom de la branche à créer à la suite de la commande précédente :

```
git branch nouvelle-branche
```

- Pour vous placer dans une autre branche à l'intérieur de votre repo, vous allez avoir besoin d'un nouveau mot-clé : **checkout** :

```
git checkout nouvelle-branche
```

Petite astuce pour manipuler vos branches : vous pouvez utiliser la commande '**git checkout -b**' pour créer une branche et vous y positionner. Ainsi, au lieu de taper la commande suivante pour créer votre branche :

```
git branch ma-branche
```
, puis une deuxième commande pour vous y positionner :

```
git checkout ma-branche
```
, vous pouvez regrouper ces deux opérations en une seule commande :

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

Une fois la branche créée, faire une modif dans le code
Commiter les modif grace à

```
git commit -a -m "Ajout commentaire dans HTML"
```

Envoyer les modifs à distance, en tapant

```
git push --set-upstream origin ma-branche
```

Fusionner des branches avec le master

Lorsque vous travaillez sur plusieurs branches, il va souvent vous arriver de vouloir ajouter dans une branche A les mises à jour que vous avez faites dans une autre branche B. Pour cela, on se place dans la branche A :

```
git checkout brancheA
```

Puis on utilise la commande **git merge** :

```
git merge brancheB
```

Résoudre les conflits

Afin de choisir quelle version de fichier gardé quand il y a conflit,on peut utiliser les outils de GitHub pour repérer les conflits et faire les modifications en éditant le fichier.

Pour être plus efficace, on peut utiliser un outil externe de merge comme WinMerge. Il est graphique et permet de repérer les écarts entre les fichiers de 2 dossiers.

Ignorer des fichiers

Pour des raisons de sécurité et de clarté, il est important d'ignorer certains fichiers dans Git, tels que :

- Tous les fichiers de configuration (config.xml, databases.yml, .env...)
- Les fichiers et dossiers temporaires (tmp, temp/...)
- Les fichiers inutiles comme ceux créés par votre IDE ou votre OS (.DS_Store, .project...)

Le plus crucial est de ne **JAMAIS versionner une variable de configuration**, que ce soit un mot de passe, une clé secrète ou quoi que ce soit de ce type. Versionner une telle variable conduirait à une large faille de sécurité, surtout si vous mettez votre code en ligne sur GitHub !

Créez le fichier **.gitignore** pour y lister les fichiers que vous ne voulez pas versionner dans Git (les fichiers comprenant les variables de configuration, les clés d'APIs et autres clés secrètes, les mots de passe, etc.). Listez ces fichiers ligne par ligne dans .gitignore en indiquant leurs chemins complets, par exemple :

```
motsdepasse.txt
config/application.yml
```

Le fichier .gitignore doit être tracké comme vos autres fichiers dans Git : vous devez donc l'ajouter à l'index et le committer.