Annexe 2 - Utilisation de Git sur notre projet

Git est un logiciel de gestion de versions. Il facilite, pour chaque développeur, la synchronisation entre le dépôt local (le code qui est sur notre ordinateur) et le dépôt distant sur GitHub (le code commun). Il est ainsi beaucoup plus simple de travailler sur un même projet et d'éviter les décalages de versions. Dans le cadre de notre projet, nous utiliserons les fonctionnalités basiques de Git. Nous avons mis en place le protocole ci-dessous.

Notre dépôt est organisé avec les dossiers suivants :

— src : contient les fichiers python

— doc : fichiers de documentation (cahier des charges, diagrammes UML...)

— donnees : contient des fichiers de données, des fichiers tests et un dossier `exports` où seront générés tous les exports.

Avant de commencer à coder

— Dans Visual Studio Code, j'ouvre un terminal Git Bash

— Menu View > Terminal (ou CTRL+ù)

— je clique sur la petite flèche vers le bas à coté du +, puis sur Git Bash

— Normalement je suis placé directement dans le bon dossier et dans la console s'affiche :

```
idxxxx yyyyyy /p/projet-info-sources/Projet-info (main)
git pull      # permet de mettre à jour le dépôt local avec le dépôt distant
```

Je code

— je crée/modifie/supprime des fichiers python (ou autre)

— je teste que ça fonctionne bien

— une fois que j'ai un morceau de code qui fonctionne, je crée un **commit** (point de sauvegarde)

— il est très important de faire régulièrement un commit dès que quelque chose fonctionne bien. Cela évitera de perdre beaucoup de temps si ensuite par une action malheureuse, le code ne fonctionnait plus du tout. Dans ce cas, un simple retour arrière au dernier commit et l'on peut repartir sur de bonnes bases

```
git add .    # permet d'ajouter tous les nouveaux fichiers créés
git status   # pour voir les changements en cours
git commit -am «˜message explicite˜»     # Pour créer un commit
```

Si par la suite je fais une erreur, il est facile de revenir en arrière (au dernier commit) :

```
git reset --hard
```

Attention cette commande supprime toutes les modifications effectuées depuis le dernier commit. Par sécurité, je crée une copie du dossier **Projet-info** avant de lancer la commande.

Je partage mon travail

- Si personne n'a poussé du code entre temps, tout va bien, je vais pouvoir faire un **push**
- Par contre si le dépôt distant a été modifié, je dois synchroniser mon dépôt local avec les mises à jour effectuées sur le dépôt distant par d'autres membres de l'équipe
- Si vous n'avez pas touché aux mêmes fichiers, Git va effectuer la fusion tout seul lors du **git pull**
- Si vous avez touché au même fichier, ça se complique un peu. Git va dire qu'il n'a pas réussi de fusion automatique (*CONFLICT (content) : Merge conflict - Automatic merge failed*). Il faut ouvrir les fichiers en conflit et ceci apparaît :

```
<<<<<<< HEAD
« Les modifications que j'ai faites »
=======
« Les modifications faites par un autre membre de l'équipe »
>>>>>>>
```

- je modifie le fichier pour choisir quelle modification je garde
- je teste et je vérifie que tout est ok
- je recrée un commit `git commit -am « merge manuel »`
- et enfin, je peux faire `git push`

```
git pull           # pour récupérer les éventuelles modifications du dépôt distant
git status         # pour voir s'il n'y a pas de conflits
git push           # pousser son code vers le dépôt distant
```

Commandes Git : ce qu'il faut retenir

```
git status               # Voir ce qui est en cours
git pull                 # Copier dépôt distant vers dépôt local
git push                 # Copier dépôt local vers dépôt distant
git add .                # Avant un commit pour que git identifie les nouveaux fichiers
git commit -am « message » # Créer un point de sauvegarde
git diff                 # Avant de faire un commit, voir les différences
```