Projet Python M1 :

Synchronisation d'un dossier local sur un serveur FTP distant

VANDENBILCKE AUBIN

BENJAMIN BELLETOILE

GUISON VIANNEY

Organisation de l'équipe pour ce projet

Pour nous organiser sur ce projet nous avons utilisés un dépôt distant hébergé sur Gitlab. Pour la répartition des tâches , nous nous sommes divisés le travail de la façon suivante :

- Benjamin et Vianney se sont occupés de la communication avec le serveur FTP distant (Filezilla), du fichier de configuration pour les logs et de la gestion des paramètres d'entrées en console
- Aubin s'est occupé de l'adaptation du TP1 à ce projet afin de l'optimiser le plus possible

Nous avons ensuite uniquement dû rassembler chacune de ces parties pour obtenir un projet complet , notamment la partie de communication avec Filezilla (ajouts, suppressions, …) et la partie de gestion des évènements du répertoire local à surveiller.

Test effectués pour vérifier le fonctionnement

- Lancer notre script lorsque le répertoire distant existe déjà sur le serveur
- Lancer notre script même si des fichiers ou dossiers sont déjà stockés sur le serveur
- Créer un nouveau dossier ou fichier en local l'ajoute au serveur
- Supprimer un dossier ou fichier en local le supprime également sur le serveur
- Modifier un fichier (modification du contenu, modification de l'extension,…) en local le modifie également sur le serveur
- Déplacer ou supprimer un dossier complet en local(contenant lui-même d'autres dossiers ou fichiers) réalise la même opération sur le serveur

Sources

Elles sont envoyées à votre adresse mail, vous pouvez également les retrouver sur ce dépôt : **https://gitlab.com/Dypz/FilesMirror.git**

Bibliothèques supplémentaires à installer

Deux librairies supplémentaires sont nécessaires pour exécuter ce projet.

- ftplib qui implémente le côté client du protocole FTP
- argparse pour la gestion des paramètres d'entrées en console

Normalement celles-ci sont installés en même temps que l'interpréteur python ; cependant si jamais vous ne les possédez pas, elles sont installables grâce à la commande pip install library_name depuis une console.

Comment utiliser notre projet ?

Différents arguments sont proposés à l'utilisateur pour mettre en place son synchroniseur de fichier sur un serveur FTP distant , on trouve dans l'ordre :

- Toutes les informations concernant notre FTP rassemblées en une seule chaîne de caractères: on y trouve le FTP auquel se connecter, le nom d'utilisateur et le mot de passe pour accéder à notre FTP ainsi que le nom du répertoire où il faut effectuer la sauvegarde. **Exemple :** "localhost,aubin,aubin,test".
- Le répertoire local qui est à synchroniser sur notre serveur FTP distant. **Exemple :** "D:\Cours\Tests"
- La profondeur maximale : cette valeur définit, à partir du répertoire à synchroniser, jusqu'à quelle profondeur nous devons synchroniser celui-ci
- La fréquence de rafraîchissement : c'est l'intervalle de temps (en secondes) entre chaque synchronisation avec le serveur FTP distant
- La liste des extensions à exclure : si on ne veut pas synchroniser certaines extensions de fichiers dans notre répertoire local alors celles-ci sont à préciser en dernier argument car c'est un argument optionnel. Si l'utilisateur ne rentre aucune extensions à ignorer alors, par défaut, tous les fichiers du répertoire local seront synchronisés. **Exemple** : ".py" ".txt"

```
positional arguments:
  ftp_website           Full FTP Website(username,password,directory)
  local_directory       Directory we want to synchronize
  max_depth             Maximal depth to synchronize starting from the root
                        directory
  refresh_frequency     Refresh frequency to synchronize with FTP server (in
                        seconds)
  excluded_extensions   List of the extensions to excluded when synchronizing
                        (optional)
```

Au final l'exécution de notre synchroniseur de fichier s'exécutera en console de la manière suivante par exemple :

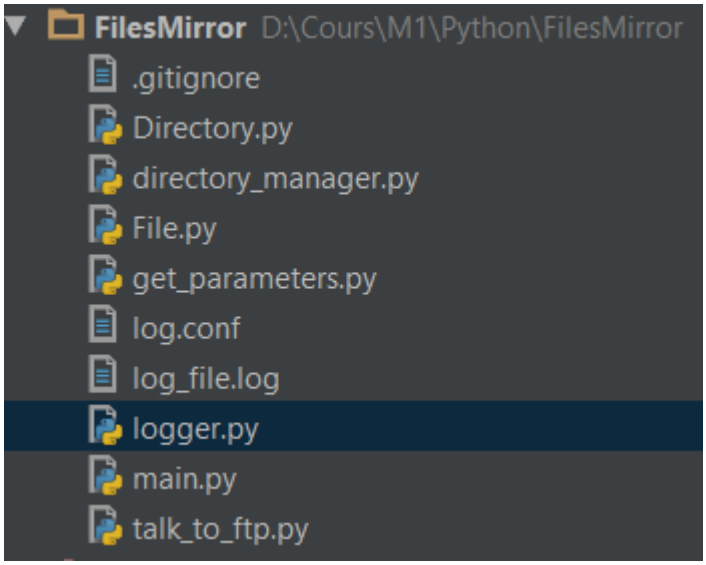
```
python main.py "localhost,aubin,aubin,test" "D:\Cours\Tests" 4 4 ".py" ".txt"
```

On peut également exécuter le script de la manière suivante pour définir le répertoire où on veut sauvegarder nos fichiers et dossiers à synchroniser sur le serveur FTP ("test\123" doit déjà être existant, A sera le nom du répertoire où la sauvegarde s'effectuera et il sera créer si besoin ; il est important de définir l'arborescence avec des "\" car ce séparateur est défini comme la valeur de os.path.sep utilisée dans le code)

```
python main.py "localhost,aubin,aubin,test\123\A" "D:\Cours\Tests" 4 4 ".py"
```

Organisation des sources :

Comme le montre la capture d'écran suivante, on trouve un certains nombres de fichiers dans notre projet : expliquons la fonction principale de chacun de ces fichiers.



- Directory et File sont des classes permettant de déterminer si le chemin sauvegardé correspond à un répertoire ou à un fichier ce qui est très utile au niveau de la synchronisation avec le serveur FTP distant car les méthodes sont différentes en fonction de la nature du chemin.
- get_parameters permet de récupérer les paramètres d'entrée de l'utilisateur qui sont fournis par la console (un exemple a été vu précédemment).
- talk_to_ftp est une classe permettant de communiquer avec le serveur FTP distant pour y effectuer différentes opérations. On utilise cette classe pour s'y connecter, se déconnecter, créer un fichier, créer un répertoire, supprimer un fichier, supprimer un répertoire sur le serveur.
- Logger est une classe statique utilisée pour avoir une trace des opérations effectuées : elle utilise notre fichier de configuration log.conf pour écrire dans un fichier de log (log_file.log) ainsi que dans la sortie standard une description de l'opération effectuée. Différents niveaux de traces sont mis à disposition dans cette classe (debug, warning, info, error, critical).
- Directory_manager est la classe au sein de laquelle tout le processus de synchronisation s'effectue : on y détecte les créations, modifications et suppressions de fichiers et/ou répertoires. Lorsqu'une de ces opérations est détectée, on y utilise alors la classe contenue dans talk_to_ftp pour mettre notre serveur FTP à jour en fonction de ce qui est détecté.

Quelques choix techniques

Synchronisation des fichiers et dossiers :

- Pour la synchronisation des fichiers, on avait choisi de se baser sur notre TP1 qui a au final subit un certain nombres de modifications pour l'optimiser le plus possible et l'adapter à ce projet. Ainsi on a choisi d'utiliser un dictionnaire (synchronize_dict) avec pour clé le chemin du fichier ou dossier et comme valeur une classe File ou Directory nous permettant de savoir rapidement à quelle type d'instance correspond le chemin en tant que clé. De plus, ces classes nous servent également pour détecter une éventuelle modification de fichier puisque l'un des attributs de la classe File correspond au timestamp de dernière modification de ce fichier. Ainsi à chaque synchronisation on va pouvoir vérifier si le timestamp au moment de la synchronisation est différent de celui enregistré dans l'instance de classe File. De plus, ce

dictionnaire étant un attribut de la classe, à chaque itération il suffit de tester si le fichier ou dossier trouvé est déjà présent ou non dans les clés du dictionnaire : si c'est le cas alors on se contente de vérifier sa date de dernière modification (timestamp) uniquement s'il s'agit d'une instance de File. Si ce n'est pas un fichier on passe à l'élément suivant. En revanche si le chemin actuel n'est pas présent dans les clés du dictionnaire c'est que celui-ci a été créé et on l'ajoute donc à notre dictionnaire avec comme valeur l'instance de classe correspondante (Directory ou File)

- Un autre point important est la gestion de la profondeur des chemins puisqu'on laisse la possibilité à l'utilisateur de définir une profondeur maximale pour le répertoire à synchroniser. Pour cela, il suffit de soustraire le nombre de 'os.path.sep' du chemin actuel au nombre de celui du répertoire à synchroniser donné en paramètre d'entrée : si celui-ci est supérieur à la profondeur maximale définie par l'utilisateur alors n'enregistre pas cet élément sur le serveur FTP ni dans notre dictionnaire.
- Evoquons maintenant comment nous détectons la suppression de fichiers / répertoires. A chaque synchronisation on stocke dans une liste l'ensemble des fichiers et dossiers parcourus (paths_explored) ; une fois qu'ils ont tous été parcourus on compare la longueur de la liste de clés de notre dictionnaire avec la taille de cette liste : si elle est différente c'est qu'une instance a été supprimée (ça ne peut être un ajout car on aura déjà ajouté les nouveaux fichiers et répertoires dans notre dictionnaire à ce moment-là). Ainsi à l'aide d'une liste de compréhension on obtient la liste des chemins ayant été supprimés : si ce chemin est un fichier alors on le supprime immédiatement. En revanche, s'il s'agit d'un dossier on supprime tout son contenu avant de le supprimer lui-même : ce processus est réalisé par la méthode remove_all_in_directory de DirectoryManager. On stocke également dans une liste tous les fichiers et dossiers ayant été supprimés (to_remove_from_dict) pour les supprimer de notre dictionnaire une fois que l'on a fini d'itérer sur celui-ci : en effet, on ne peut modifier une instance sur laquelle on est actuellement en train d'itérer d'où l'utilisation de cette liste.
- Enfin, à chaque fois qu'on détecte un changement, une suppression ou une modification de fichier / répertoire alors on se contente d'appeler la fonction appropriée de la classe 'TalkToFTP'.
- Tout ce processus se passe dans la fonction search_updates de la classe directory_manager.

<u>Connexion et intéractions avec le serveur FTP :</u>

- La connexion au serveur ftp est limité dans le temps. Pour éviter de se faire déconnecter automatiquement, on se connecte seulement quand on a besoin d'accéder au ftp (après une modification par exemple) et on se déconnecte à la fin du transfert.
- L'utilisateur peut choisir de créer une copie de son miroir dans un dossier qu'on crée à la racine du serveur, ou alors de créer son miroir dans un dossier qu'on crée au sein d'un chemin sur le serveur (dans ce cas, au lieu d'indiquer un nom de dossier, l'utilisateur indiquera un chemin avec comme séparateur "\" comme évoqué précédemment dans la notice d'utilisation).
- A l'initialisation, on vérifie que le dossier dans lequel on veut créer notre miroir n'existe pas déjà. S'il existe, on vérifie qu'il contient bien tous les élément du dossier qu'on veut copier. S'il manque un élément, on le copie. S'il l'élément est déjà présent, on ne le recrée pas de nouveau.
- Après l'initialisation, on fait tourner une boucle dans laquelle on vérifie s'il y a eu un ajout, une suppression ou une modification dans notre dossier local. Pour cela, on s'est basé sur notre premier TP qui consistait à détecter des changement dans un dossier.