1 Cycle de développement avec git

Introduction

Ce document a comme objectif d'être un guide d'utilisation de Git lors du développement de code. Ce guide n'est cependant pas parfait, il a été simplifié afin que n'importe qui ayant des connaissances théoriques de base git soit en mesure d'intégrer git dans son développement de code. De plus, ce guide contient quelques étapes qui sont propres à Python (encadré en bleu dans la figure 1.1). En ignorant les encadrés bleus, ces étapes peuvent être appliquées à n'importe quel langage. Il est important de noter que pour que le tout fonctionne, il faut que Git soit installé. Afin que ce guide soit indépendant du langage de programmation, de l'environnement de développement intégré et de tout outil facilitant l'utilisation de Git, l'ensemble de ce guide passera par l'intermédiaire de l'invite de commande. Puisqu'aucune compétence préalable n'est requise pour l'utilisation de ce guide, tous les chemins vers les différentes commandes seront en PATH absolut (donc à partir de C : sur Windows. Pour terminer, le document a été basé sur l'utilisation de Windows. Cependant, les étapes et les principes sont les mêmes pour tous les systèmes d'exploitation.

1.1 Clonage du dépot Git

Avant de contribuer à un code, il faut commencer par cloner le dépôt Git (dans le cas d'un nouveau projet, ce cycle prend pour acquis qu'un dépôt Git avec au moins un fichier README a préalablement été créé à partir du serveur hôte du dépôt Git (ex. Github, Gitlab). Pour ce faire, il faut simplement faire "git clone <url vers le dépôt git>" dans l'invite de commande ou le faire à partir de l'interface graphique de votre environnement de développement intégré (Pycharm, Vscode, etc.). Essentiellement, cette étape consiste à créer un dépôt Git local, qui fonctionnera en parallèle à celui du serveur hôte tout au long du développement.

1.2 Création de l'environnement virtuel

Après le clonage du dépôt git, il est important de créer un environnement virtuel Python. Un environnement virtuel permet d'utiliser un "Python" différent pour chaque projet. Avoir des modules qui sont propres à chacun des projets permet d'éviter qu'un vieux code ne fonctionne plus puisque tous les modules ont été mis à jour pour un autre projet. Il y a plusieurs manières différentes de faire un même environnement virtuel. Une manière qui fonctionnera toujours sera de premièrement créer un dossier vide se nommant "venv" dans votre projet python et d'effectuer la commande suivante dans un terminal : "\path\to\python\python.exe -m venv \path\to\new\environment\venv". Normalement, le Python de votre système devrait se situer à
"C :\Users\votre user\AppData\Local\Programs\Python\PythonXX\" où XX est la version du Python (ex. Python38 pour v.3.8). Le tout est sur Windows, mais le principe reste le même sur les autres systèmes d'exploitation, seules la localisation de Python et la syntaxe du terminal peuvent changer. Par la suite, il est important de s'assurer que l'environnement de développement intégré (Pycharm, Vscode, etc.) utilise le python.exe qui se situe dans le dossier venv que vous venez de faire, et non celui du système. Il ne faut pas oublier qu'il faut utiliser la version de python spécifiée par le créateur du code si celui-ci ne provient pas de vous.

1.3 Création de votre branche de développement

Avant de commencer à développer une nouvelle fonctionnalité au code, il faut vous créer une branche de développement associée à ce que vous voulez ajouter. Dans le cas d'un nouveau projet, il ne faut pas que le dépôt Git local et/ou sur le serveur soit vide pour que la création de branches fonctionne. Pour ce faire, il faut tout simplement effectuer les commandes suivantes : "cd path\to\your\project", "git branch <nom de la branche>". Souvent, cette commande peut s'effectuer directement par l'interface graphique de votre environnement de développement intégré (pycharm, vscode, etc.). Après avoir créé la branche de manière locale, il faut l'envoyer sur le dépôt Git du serveur en effectuant : "git push -u origin <nom de la branche>"

1.4 Sélection de la branche de travail et mise à jour des dépendances

Il faut ici s'assurer que nous travaillons sur la bonne branche. Pour ce faire, il faut effectuer les commandes suivantes : "cd path\to\your\project", "git checkout <nom de la branche>". Encore ici, cette commande peut s'effectuer directement par l'interface graphique de votre environnement de développement intégré (pycharm, vscode, etc.). Une fois sur la bonne branche, il faut s'assurer de faire un "git pull" pour avoir la version la plus à jour de la branche. Une fois que c'est fait, la prochaine étape est de s'assurer d'avoir tous les modules requis pour faire fonctionner le code. Si le fichier "requirements.txt" existe déjà, il suffit de simplement faire dans l'invite de commande "path\to\virtual\envirement\scripts\pip install -r path\to\your\project\requirements.txt" afin que Pip installe toutes les versions spécifiques de chacun des modules requis. En cas d'absence de requirements.txt, il suffit de faire "path\to \virtual\environment\scripts\pip install <nom du package>==X.XX" où X.XX est la version du module (il est possible de ne pas spécifier la version en retirant le ==, ce qui installera automatiquement la version la plus récente).

1.5 Développement du code

Normalement, il devrait y avoir un commit et un push (voir étape 6) par jour pour être capable de suivre adéquatement l'avancement du code. S'il y a trop de commits, il faudra annuler trop de commits pour faire un retour en arrière en cas d'erreur. Dans le sens opposé, un manque de commits engendrera une perte énorme en code pour faire un retour en arrière pour une simple erreur. De plus, tout au long du développement, il est important de garder en tête que tous les nouveaux fichiers produits devront peut-être se retrouver dans le .gitignore de l'étape 6.

1.6 Mise à jour des fichiers relatifs à Git

Directement après avoir terminé la session de développement, il est important de générer de nouveau le fichier requirements.txt afin que les nouvelles dépendances introduites dans le code soient incluses dans les modules requis. Pour ce faire, il faut simplement le générer à partir de Pip avec la commande suivante :
"path\to\virtual\envirement\scripts\pip freeze > path\to\your\project\requirements.txt". Par la suite, avant de sauvegarder les modifications, il est important de mettre à jour ou de créer le fichier .gitignore. Celui-ci permet de spécifier ce qui doit être inclus dans le Git et ce qu'il ne le devrait

pas. Cela permet donc d'avoir un contrôle sur ce qui se retrouve sur le dépôt git du serveur. Normalement, seulement du code, des figures, des PDF et des fichiers Git (.gitignore, README.rm et, etc.) devraient se retrouver sur le dépôt du serveur (essentiellement, tout ce qui ne se génère pas de manière automatique). Pour ceux qui ne sont pas à l'aise avec le .gitignore, il existe un générateur de .gitignore qui permet d'en générer un en fonction du langage ou de l'environnement de développement intégré (Pycharm, Vscode, etc.) utilisé. Aussitôt que votre code génère des fichiers automatiquement, ceux-ci doivent être ajoutés au .gitignore. Avec un .gitignore, il est possible d'ajouter de manière sécuritaire toutes les modifications au commit en effectuant les trois commandes suivantes :
"cd path\to\your\project", "git add *" et "git commit -m "message"". Lorsque que les changements sont sauvegardés par le commit, il faut tout de même les envoyer au dépôt git sur le serveur en effectuant la commande "git push". Si le développement de la fonctionnalité n'est pas terminé, il faudra retourner à l'étape 4. Si le tout est terminé, il faut continuer à l'étape 7.

1.7 Mise à jour du README.md

Avant de rendre disponible la nouvelle fonctionnalité, il faut qu'un utilisateur potentiel soit en mesure de l'utiliser. C'est l'utilité du fichier README.md. Ce dernier devrait contenir les informations relatives aux processus d'installation particulière pour que le code fonctionne (ex. requiert un programme externe et la version de python utilisé) en plus d'une description générale du code et de son utilisation. Voici un petit guide pour un README.md simpliste. Les modifications doivent ensuite être sauvegardées et partagées au dépôt Git sur le serveur en effectuant : "cd path\to\your\project", "git add *", "git commit -m "Update README.md"" et "git push".

1.8 Fusion entre la branche de développement et la branche principale

Une fois que la nouvelle fonctionnalité est complète, fonctionnelle et accessible à l'utilisateur, il faudra l'ajouter à la branche principale (souvent appelée master ou main). Pour ce faire, il faut faire les commandes suivantes : "cd path\to\your\project", "git checkout <nom de la branche principale>" et "git merge <nom de la branche>". Si vous ne connaissez pas le nom de votre branche principale, vous pouvez effectuer la commande suivante : "git branch", pour lister toutes vos branches. Votre branche principale sera celle ayant une * devant son nom. Pour que la fusion se fasse aussi sur le dépôt Git du serveur, il faut simplement faire la commande : "git push". Par la suite, puisque les branches ne seront plus utilisées, il est possible de supprimer la branche locale en effectuant "git branch -d <nom de la branche>" et supprimer celle sur le serveur en faisant "git push origin --delete <nom de la branche>". Si vous avez d'autres fonctionnalités à ajouter, il suffit de retourner à l'étape 3. Si ce n'est pas le cas, félicitations, vous avez terminé un projet ou une contribution en utilisant Git !

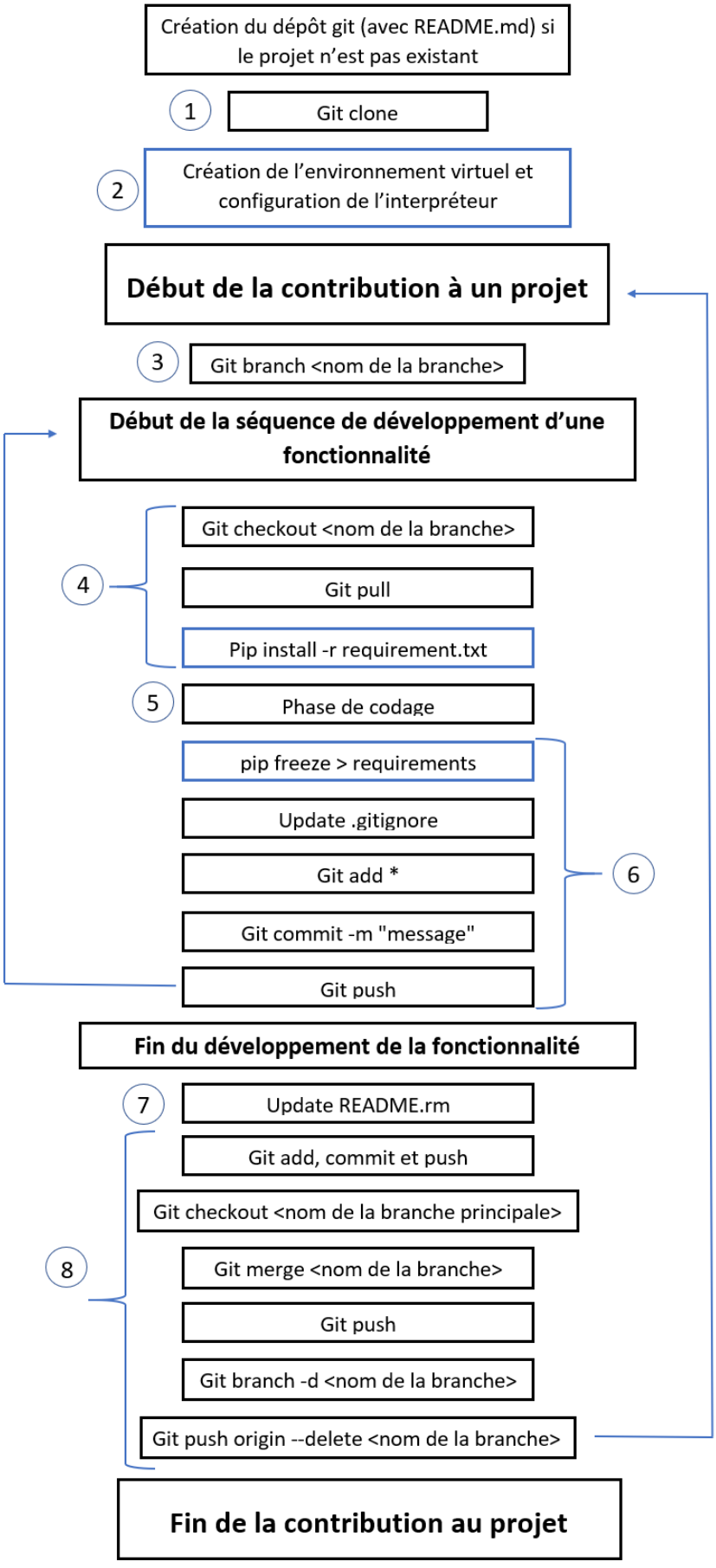

FIGURE 1.1: Schéma du cycle de développement de code intégrant git. Les encadrés bleus sont les étapes propres au langage Python, il est possible de les ignorer.