Projet informatique - Tutoriel git

Manon Ansart et Florian Scalvini

ESIREM, 2022

1 Git

1.1 Qu'est-ce que git ?

Git est un système de gestion de versions (version control system ou VCS en anglais), c'est à dire un système permettant de garder une trace de toutes les modifications effectuées sur un ensemble de fichiers. Le terme versioning est aussi employé couramment.

Git est utilisé en programmation pour suivre les modifications effectuées sur des fichiers de code. Un projet sur git s'appelle un dépôt (repository en anglais, souvent abrégé "repo"). Lorsque des modifications sont effectuées sur le code source, elles sont sauvegardées sous la forme d'un commit. Un commit contient l'ensemble des modifications effectuées depuis le dernier commit.

Git est un système gratuit et open source. D'autres systèmes de gestion de versions existent et sont utilisés en programmation, comme SVN, mais git reste le plus courant.

1.2 Quand et pourquoi utiliser git ?

Git a deux intérêts majeurs :

- faciliter le travail en groupe : git permet de récupérer sur sa machine les modifications (commit) effectuées par ses coéquipiers et de facilement fusionner les modifications faites en même temps par différentes personnes. Quand il est bien utilisé, git permet de comprendre plus facilement ce qui a été fait par qui et quand
- organiser son travail et conserver des versions antérieures du code : effectuer des commits régulièrement permet de mieux structurer l'avancée de son travail et de revoir ce qu'on a fait avant. Lorsque le code ne fonctionne plus, git permet de revenir facilement à une version antérieure, bien identifiée grâce aux commits

Donc, à l'avenir, quand devez-vous utiliser git ?

- quand vous développez quelque chose à plusieurs
- quand il y a un risque pour que vous vous mélangiez les pinceaux dans votre code ou si vous risquez de vous tromper et d'avoir besoin de revenir à une version antérieure. Donc, concrètement : pour tout projet de programmation de plus d'une journée

1.3 Et Github, c'est quoi ?

Github est un service en ligne qui permet d'héberger, visualiser et gérer des dépôts git. Par défaut, les dépôts présents sur Github sont publiques : tout le monde peut voir le code du projet, qui l'a modifié et quand. C'est une plate-forme couramment utilisée pour la mise à disposition de code open source. Les utilisateurs extérieurs au projet peuvent également faire des demandes pour effectuer eux-même des modifications sur le code du projet, ce qui contribue à la collaboration autour de l'open source.

Pour ce projet, nous vous proposons d'utiliser Github pour héberger votre dépôt git. Vous devez pour cela avoir un compte Github. Si vous n'en avez pas vous pouvez en créer un gratuitement (`https://github.com/`).

Par défaut, votre projet sera publique. Avoir un compte github actif avec du code visible peut être un point fort lorsque vous cherchez un poste d'ingénieur en informatique. Vous avez toutefois la possibilité de créer

des dépôts privés si vous le souhaitez. Cela nécessite un compte Pro, que vous pouvez avoir gratuitement en tant qu'étudiant (`https://education.github.com/discount_requests/student_application`).

1.4 Comment utiliser git ?

1.4.1 Processus

Git s'utilise à l'aide de lignes de commandes, à travers des interfaces graphiques dédiées ou dans des éditeurs de code. Pour ce projet nous allons utiliser git à travers PyCharm. **Vous êtes libres de le faire à l'aide d'un autre outil si vous le souhaitez**.

À la création du projet, plusieurs étapes doivent être réalisées pour mettre en place le dépôt :

- Création du dépôt git
- Publication du dépôt sur Github
- Mise en place du dépôt sur de nouvelles machines (pour les coéquipiers)

Ensuite, chaque membre de l'équipe utilise le dépôt en suivant les étapes suivantes :

- Vérifier si les co-équipiers ont fait des commits et les récupérer si besoin (**pull**)
- Coder et modifier les fichiers. Si cette étape dure longtemps, effectuer des pull régulièrement pour récupérer les éventuels commits
- Quand les modifications ont été effectuées pour une tâche, une fonction ou une unité de travail :
  - Sélectionner les fichiers sur lesquels on a fait des modifications et que l'on souhaite mettre dans le commit (**add**). Si plusieurs fichiers ont été modifiés, mais que certaines modifications n'ont rien à voir avec la tâche correspondant au commit, on peut choisir quels fichiers on ajoute.
  - Effectuer le commit en y associant un message décrivant ce qui a été effectué (**commit**). À ce stade le commit n'est présent qu'en local, sur votre machine, et n'est pas visible des autres membres de l'équipe
  - Publier le commit pour que les autres membres de l'équipe puissent le voir et récupérer les modifications (**push**)

1.4.2 Bonnes pratiques

Afin que le projet se passe bien et que git soit aussi utile que possible, il est recommandé de suivre les bonnes pratiques suivantes :

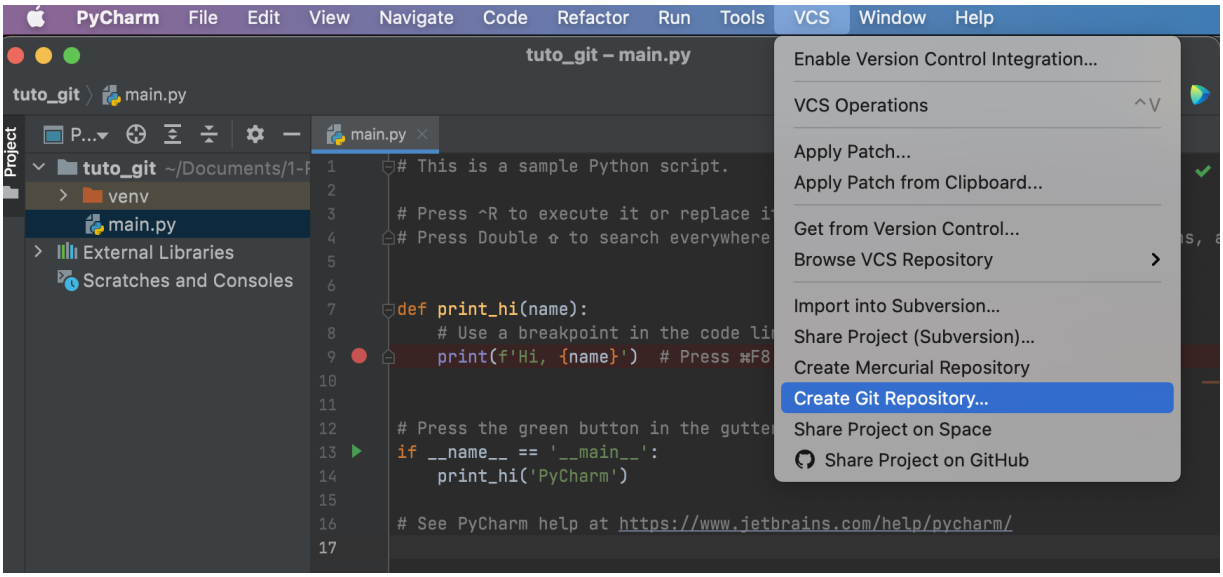
- Ne faire que des commits qui font passer le code à un état stable. Il est nécessaire qu'après chaque commit le code fonctionne et ne donne pas d'erreur. **Lancez donc toujours votre code avant de faire un commit**, et lancez vos tests si vous en avez.
- Faire des commits correspondant à des unités de travail : un commit n'est pas un ensemble de changements sans rapport les uns avec les autres, il doit rassembler des changements répondant au même objectif. **Un commit = un objectif**
- Faire des petits commits **le plus fréquemment possible**
- Choisir des **noms de commits appropriés**, décrivant de manière brève mais claire les changements effectués et la raison pour laquelle ils font été faits (tâche/objectif)
- Faire des pull fréquemment pour intégrer les dernières modifications et minimiser les conflits

2 Mise en pratique

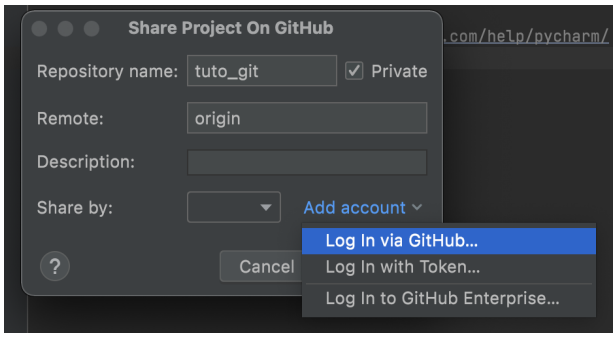
Pour cette mise en pratique, nous allons mettre en place un dépôt git en utilisant PyCharm et le projet que vous avez créé précédemment. Vous pouvez trouver des informations sur l'utilisation de git avec PyCharm dans la documentation de PyCharm (`https://www.jetbrains.com/help/pycharm/using-git-integration.html`)

2.1 Créer et publier le repo

1. Dans PyCharm, cliquez sur VSC -> Publier le projet sur github. Cette action permet à la fois de créer le dépôt git et de le publier sur github.



2. Une fenêtre apparaît et vous permet de configurer votre dépôt. Vous pouvez choisir le nom de votre dépôt et vous connecter à votre compte GitHub pour y mettre le dépôt (créez un compte si vous n'en avez pas).



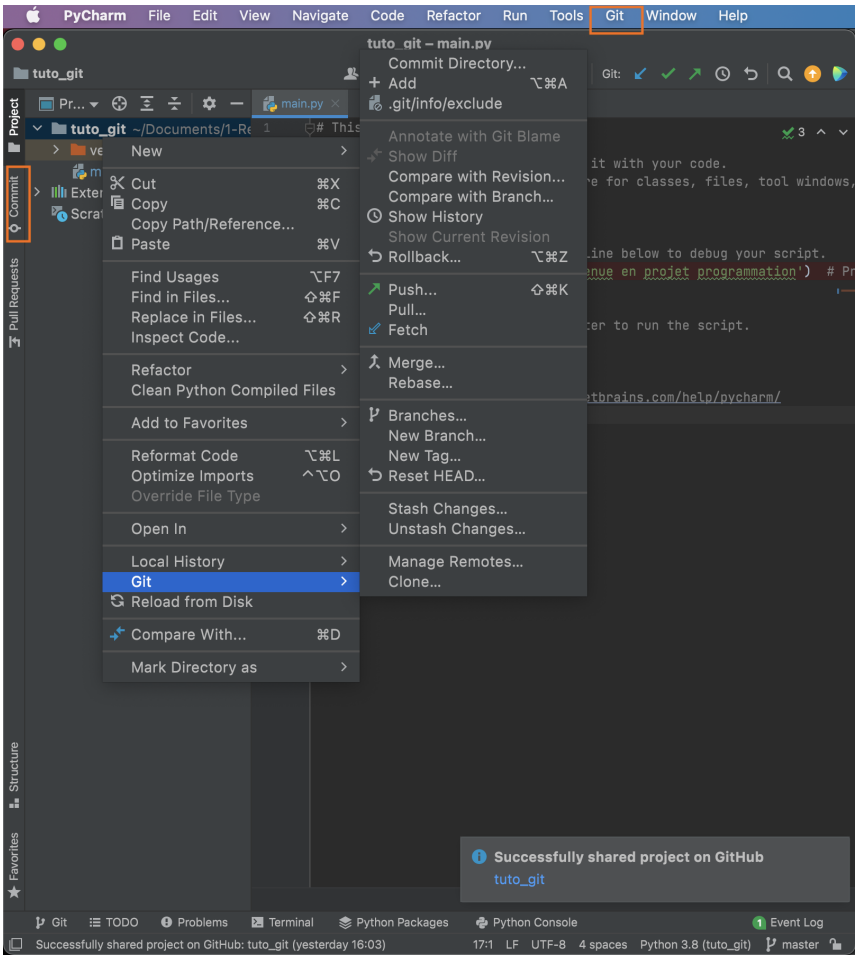
3. Une fenêtre vous demande ensuite quels fichiers vous voulez ajouter dans votre commit initial. Vous pouvez tous les sélectionner. Cliquez ensuite sur "Valider" et, félicitions ! Vous avez créez votre repo et fait votre premier commit.

2.2 Actions sur un repo existant

Au début du projet, vos fichiers apparaissent en blanc dans le panneau de gauche dans PyCharm. Ce sont des fichiers qui sont dans le même état que sur le git. Lorsque vous modifiez ces fichiers, ils apparaissent en bleu.

Une fois que votre projet PyCharm est lié à un dépôt git, il vous est possible d'effectuer des actions en lien avec git de plusieurs manières :

— Via la barre de menu en haut (onglet "Git")
— Via l'onglet "Commit" tout à gauche de la fenêtre
— En faisant un clic droit dans la liste des fichiers à gauche



Vous pouvez effectuer un commit en utilisant un de ces 3 moyens d'accès. Dans tous les cas vous aurez à gauche un panneau récapitulant les fichiers modifiés. Vous pouvez sélectionner les fichiers que vous souhaitez ajouter au commit et indiquer le message du commit. Vous avez le choix ensuite le choix entre effectuer votre commit uniquement ou effectuer votre commit et le publier via push. Si vous choisissez la première option, vous pourrez push votre commit plus tard en utilisant le menu Git.

Exercice :

1. Faites une modification dans votre code
2. Effectuez un premier commit (sans push).
3. Faites une autre modification.
4. Effectuez un deuxième commit (sans push).
5. Pushez ensuite les 2 commits.

Lorsque vous ajoutez un nouveau fichier celui-ci apparaît en rouge. Cela signifie que le dépôt git n'a pas connaissance de ce fichier, qui n'apparaîtra pas dans les fichiers modifiés pour les commits. Vous pouvez ajouter le fichier au dépôt en faisant un clic droit sur le ficher -> Git -> Add. Le fichier apparaîtra alors en vert. Il est ajouté au dépôt mais il n'y a pas encore été inclus dans un commit. Lors de la création du fichier, PyCharm vous proposera automatiquement d'ajouter le fichier au git. Dans ce cas vous n'avez pas besoin de faire l'ajout vous même.

Exercice :

1. Ajoutez un nouveau fichier et cliquez sur Annuler quand PyCharm propose l'ajout du ficher.
2. Ajoutez le fichier vous-même au dépôt.
3. Effectuez des modifications sur ce fichier (et sur d'autres si vous le souhaitez), effectuez un commit et pushez

2.3 Travail à plusieurs

Pour cette partie, vous allez collaborer avec un autre étudiant ou un binôme ayant créé un dépôt git différent du votre. Vous allez récupérer vos repos respectifs puis effectuer en parallèle des modifications

sur vos repos respectifs afin de voir la gestion des conflits.

2.3.1 Récupérer un dépôt existant

1. Partagez l'url de votre dépôt à l'autre groupe. Pour cela allez dans le panneau de fichiers projet à gauche et faites un clic droit sur le dossier de votre projet, puis Git -> Manage Remotes. Vous afficherez ainsi l'url de votre dépôt que vous pouvez envoyer ou dicter à l'autre groupe.
2. Après avoir récupéré l'url de l'autre dépôt, vous allez le mettre sur votre machine (on dit que vous **clonez** le dépôt). Pour cela
   (a) allez dans le menu Git et sélectionnez "Clone..."
   (b) Indiquez l'url du dépôt que vous souhaitez cloner
   (c) Indiquez l'emplacement dans lequel vous souhaitez mettre ce dépôt. Ne le mettez pas dans le votre de préférence.
   (d) Cliquez sur "Clone"
   (e) Le projet cloné sera automatiquement ouvert dans PyCharm. Ouvrez-le dans une nouvelle fenêtre.

2.3.2 Changements en parallèle sans conflits

1. En discutant avec l'autre groupe, faites une modification chacun de votre côté, dans le même projet mais en utilisant chacun un fichier différent.
2. Commitez chacun votre modification sans pusher.
3. Faites un pull (Git -> Pull... puis ok). Que se passe-t-il ? Pourquoi ?
4. Pushez chacun votre modification.
5. Une fois que l'autre groupe a pushé, faites un pull. Que se passe-t-il ?
6. Affichez l'event log (en bas à droite). En cliquant sur "view commit", affichez le commit qui a été effectué par l'autre groupe
7. Cliquez sur Git -> Show git log pour afficher tout l'historique des commits effectués

2.3.3 Résolution des conflits

Conflit après commit : dans cette partie, nous allons gérer le cas où un de vos coéquipiers a modifié un fichier que vous avez modifié également et commité (avant votre push)

1. En discutant avec l'autre groupe, faites une modification chacun de votre côté, dans le même projet et sur le même fichier, en ajoutant chacun une fonction différente par exemple.
2. Commitez chacun votre modification sans pusher.
3. L'un des deux groupes fait son push
4. L'autre groupe pull. PyCharm vous informe qu'il y a un conflit. Vous pouvez soit garder l'une ou l'autre des versions, soit les combiner, ce qui s'appelle un **merge**. On choisit cette troisième option. Utiliser l'interface de PyCharm pour combiner les autres commits et pusher la modification. Faites un pull des deux côtés
5. Afficher le git log (historique des commits)

Conflit avant commit : dans cette partie, nous allons gérer le cas où un de vos coéquipiers a modifié un fichier que vous avez modifié et où vous faites un pull avant votre commit

1. En discutant avec l'autre groupe, faites une modification chacun de votre côté, dans le même projet et sur le même fichier, en ajoutant chacun une fonction différente par exemple.
2. L'un des groupes commit et push
3. L'autre groupe pull. Pycharm vous informe qu'il y a un conflit. En cliquant sur l'événement en bas à gauche, vous pouvez visualiser la modification proposée, mais elle n'est pas encore intégrée dans votre code. Résolvez le conflit de la manière suivante :
   (a) Faites un clic droit sur le fichier que vous avez modifié et sélectionné Git -> Stash changes. Cela efface les changements que vous avez fait et les garde en mémoire pour les rajouter plus tard.
   (b) Faites un pull, qui va cette fois fonctionner puisque les changements que vous avez faits ne sont plus présents dans vos fichiers

(c) Faites un clic droit sur le fichier que vous avez modifié et sélectionné Git -> Unstash changes et sélectionnez votre stash. Choisissez "Apply stash". Cela rajoute vos changements comment si vous les effectuiez à nouveau.

(d) Comme ces changements sont sur un fichier qui a été modifié entre temps, vous avez un conflit similaire à ceux après commit. Résolvez-le comme précédemment.

4. Faites un commit et un push, ainsi qu'un pull pour l'autre groupe
5. Plutôt que d'utiliser ce processus, il est possible de commit ses changements après le pull qui a posé problème et d'utiliser la résolution de conflits précédemment décrite. Attention cependant : on ne commit que du code qui fonctionne, vous ne devez donc pas faire de commit si vous n'avez pas fini ce que vous étiez en train de faire.

Attention : Plus des changements ont été effectués par vos co-équipiers, plus il y a de conflits et plus ils sont difficiles à résoudre. C'est pourquoi il est important de faire des petits commits très fréquemment, de pusher régulièrement et de pull encore plus régulièrement.

2.4 Revenir en arrière

À n'importe quel moment vous pouvez ramener votre code à un état précédant. Pour cela il est préférable de ne pas avoir de changement locaux non commités pour éviter de les perdre.

1. Affichez le git log. Un tag "origin & master" affiche le dernier commit, qui est le commit actif.
2. Faites un clic droit sur n'importe quel autre commit et sélectionnez "Reset current branch to here...". Choisissez un reset "hard" : les fichiers reviennent à l'état du commit et vos changements locaux sont perdus. Dans le git log, l'étiquette "master" est maintenant sur le commit que vous avez choisi : c'est le commit auquel est votre machine. Le tag "origin/master" est cependant toujours sur le dernier commit : c'est le commit auquel en est le dépôt git accessible à tous.
3. Revenez au dernier commit de la même manière. Notez le déplacement de l'étiquette et vérifiez que tous vos fichiers sont bien revenus à leur état d'origine