Maxime Blanchard  
Mathieu Renaud  
Groupe 0002

**Travail Pratique 1**Travail présenté à monsieur François Bertrand   
  
Travail en Informatique  
402-101-SF

Département d’informatique  
Programme - Technique de l’informatique  
Cégep de Sainte-Foy  
16 Octobre 2017

Introduction à Git

**L’histoire**

C’est en 2005 que Linus Torvalds, fondateur du noyau Linux, crée le logiciel de gestion de version décentralisée. C’est au moment que BitKeeper, le gestionnaire de version qu’utilisait Linux, annonce qu’il serait dorénavant payant que Linus Torvalds se met alors à créer cette plateforme qui deviendra de loin le « version control system » (VCS) le plus utilisé.

**Pourquoi utiliser Git ?**

Plusieurs avantages démontrent pourquoi plus de 70% des utilisateurs de VCS utilisent Git. Regardons en surface les arguments positifs de ces utilisateurs :

1. Une gestion de versions décentralisée. Chaque utilisateur peut avoir le projet complet sur son propre ordinateur et être alors complètement indépendant du serveur principal sur lequel le projet a été initialement mis. De plus, l’utilisateur a le contrôle sur les fichiers avec lesquels il veut travailler et les rendre publiques.
2. La gestion des branches et de la façon de fusionner celles-ci se fait d’une façon très facile et rapide. De plus, la création de branches ne prend pas beaucoup d’espace de stockage et améliore alors la vitesse d’exécution.
3. Il y a beaucoup de services associés à Git comme Github ou BitBucket qui sont les plus populaires.

On peut également trouver des inconvénients à ce genre de VCS. Utiliser un système comme celui-ci peut être assez lourd pour la plupart des utilisateurs au début. C’est pourquoi qu’implanter Git dans une entreprise où les employés n’y sont pas familiers n’est pas la meilleure idée. Par contre, connaître ces technologies en tant que programmateur est rendu essentiel à ses outils. Certains disent que Git ne supporte pas si bien les fichiers très lourds, et on ne peut pas faire des « commits » sur plusieurs branches simultanément.

*Figure 1.1 - Le modèle Git (La gestion des versions décentralisés)*



Comment ça fonctionne ?

Ça peut sembler compliqué pour plusieurs lorsqu’on s’y met et qu’on veut apprendre le système au début. Par contre, lorsqu’on y comprend la logique, tout devient claire, et une fois qu’on y adhère, on ne voudra plus s’en passer. Les programmeurs, les entreprises, les designers, utilisent Git dans leur vie de tous les jours.

Pour résumer le fonctionnement de Git, on peut imaginer le tout en trois étapes distinctes. Premièrement, on a soit la création d’un dossier parent git, communément appelé « *repository*», ou on peut dupliquer (cloner), un dossier sur un serveur, tel que GitHub. Une fois la synchronisation faite, on devient alors propriétaire de notre propre projet, où l’on peut y faire des changements sur les fichiers qu’on veut, tout cela hors ligne, sur notre répertoire local.

Une fois qu’on a assez travaillé dans notre projet et on veut alors en faire une « photographie » afin que Git se rappelle de la version présente, c’est le moment de faire un commit des fichiers qui nous semblent important de prendre en considération. Effectivement, un avantage de git est qu’il comporte un emplacement de « stage », où on prépare notre « snapshot » de notre version. (voir la figure 1.2)

Pour terminer, tout ce qu’il nous reste à faire est d’envoyer (push) vers le serveur notre version actuelle. Cela peut se fera généralement sur une branche indépendante de la branche maitre. Si on est prêt à préparer la version de notre finale, on peut alors fusionner les branches ensembles afin de retourner celles-ci sur la branche maitre (master) (voir figure 1.3)

*Figure 1.2 – Les trois étapes au fonctionnement Git*

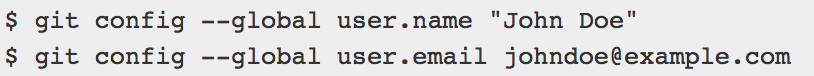


*Figure 1.3 – Démonstration du principe des branches Git*  
  


Comment utiliser Git

Malgré le fait que plusieurs interfaces visuelles sont maintenant disponibles pour nous faciliter la tâche à l’utilisation de ce système, la compréhension du terminal est essentielle pour arriver à nos fins. Veuillez noter que les instructions suivantes ne sont que la base de ce que peut faire Git. Le reste de l’apprentissage se fera avec l’expérience acquise avec vos différents projets.

1. Installer Git
   * Certains systèmes d’exploitation arrivent déjà avec Git installé. Dans le cas contraire, il est possible d’installer le logiciel par des installateurs, ou encore mieux par des gestionnaires de paquets, comme Xcode ou Homebrew pour MacOS, ou Chocolatey pour Windows.
2. Les réglages de votre identité
   * Afin de savoir qui a fait quels changements, vous inclusivement, il est alors préférable que chaque utilisateur affiche leur nom et leur adresse courriel avec la commande « git config »



1. S’approprier d’un répertoire Git
   * La première façon est de créer par vous-même un répertoire Git. Dans ce cas-ci, vous seriez celui qui partagera avec d’autres votre projet. Pour faire ceci, il vous faut simplement aller dans le dossier où vous voulez avoir votre répertoire de gestion de versions et faire la commande « git init ». un dossier caché (.git) se créedans ce dossier même afin de contrôler tous les changements qui se feront dès le moment de sa création.

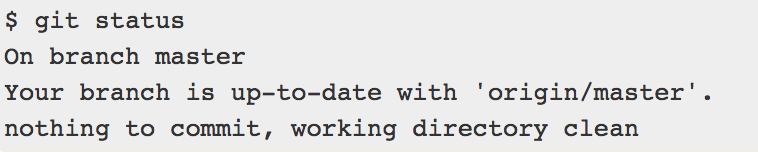


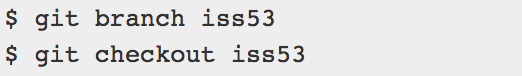
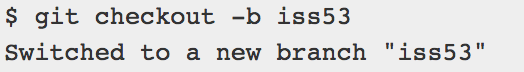
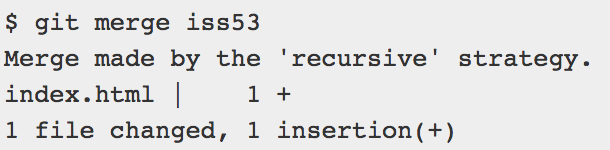
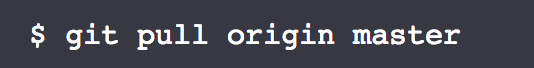


* + La deuxième façon est de cloner une copie déjà existante d’un répertoire Git sur un autre serveur. La façon la plus commune et la plus rependue est de mettre nos répertoires sur GitHub, un service d’hébergement web de gestion Git (voir deuxième section du document – GitHub). Dans ce cas où le répertoire est en ligne, vous n’avez qu’à ajouter le lien http à la commande « git clone ».



1. La commande « git status »
   * Cette commande est très importante, car elle permet de voir sur quelle branche on travaille présentement, et ainsi voir quels fichiers ont été modifiés sur la présente branche. Le plus souvent on l’utilise, mieux c’est ! À ce stade-ci, vous avez l’entièreté du projet sur votre dossier local, vous faites alors vos modifications personnelles.



1. Comprendre le système des branches
   * Certainement un des outils les plus importants de git est le système des branches. En fait, la plupart du temps, dès que vous débutez à travailler sur un nouveau projet que vous venez de cloner, vous devriez vous créer une branche pour ne pas faire de changement sur la branche principale (Master), pour ne pas rendre celle-ci instable. Pour ce faire nous allons utiliser la commande « git branch nom\_de\_la\_branche » pour créer la nouvelle branche.  
     
   * La dernière commande signifie que l’on entre dans la branche voulue. Ici, on veut commander à faire des changements dans la branche « iss53 », alors on checkout dans celle-ci.
   * Il est possible de sauver un peu d’énergie et de faire une ligne de code seulement pour faire ce chemin. La voici :  
     
2. Faire un « commit » sur notre branche courante
   * Il est toujours bien de faire la commande « git status » pour voir si on est bien en train de travailler sur la branche voulue. Si on veut voir l’étendue des branches existantes, on peut seulement utiliser « git branch ».
   * Maintenant que vous avez travaillé pendant un moment sur votre projet, vous sentez qu’il est temps de sauvegarder l’effort effectué. (Prendre note que cette partie est très simplifiée dans le but d’alléger l’apprentissage. Voir la documentation Git pour plus d’informations).
   * Comme nous avons vu plus haut, Git comporte un système de « staging area », qui nous permet de trier ce qu’on veut sauvegarder ou pas dans notre prochaine version. Dans un cas général, nous voulons sauvegarder tout ce que nous avons fait, ainsi que les nouveaux fichiers, images, dossiers ajoutés au projet. Pour que Git puisse reconnaître ces fichiers comme important, nous allons les ajouter au « tracking system » avec la commande « git add . » Ensuite, nous pouvons alors porter tous les changements à notre nouvelle version qui sera pointée par Git, avec la commande « git commit ». Il est conseillé pour sauver du temps de faire le tout en une seule commande.  
     
3. Fusionner notre branche personnelle avec la « master »
   * Bien sûr, on comprendra qu’à la fin d’un projet, on doit mettre la totalité de nos efforts en commun. On doit alors effectuer une action afin de fusionner notre branche dans la « master » pour mettre celle-ci à jour avec notre travail. La première étape consiste à se placer dans la branche qui va recevoir la fusion. Dans notre cas, nous allons sur la branche « master ». Avec la commande « git merge nom\_de\_la\_branche\_à\_recevoir » Git copie toutes les versions d’une branche vers l’autres, et donc toutes les modifications apportées au projet ensemble.   
     
4. Gérer les conflits de fusion de branches
   * Dans la dernière fusion plus haut, Git a été en mesure de gérer par lui-même les différences entre les deux versions sur chaque branche, afin d’en faire une seule version. Il a procédé comme on peut le voir par la stratégie « récursive ».
   * Cependant, il peut avoir des instances où Git n’est pas en mesure de gérer certains conflits. Pour parvenir à compléter la fusion, nous devons l’aider à lui faire comprendre quelles lignes de codes devrait-on garder et/ou, supprimer. Par exemple, Git aurait pu nous renvoyer un message comme celui-ci :  
     
   * La bonne nouvelle dans tout cela, c’est que Git nous renvoie le fichier index.html avec ces lignes en conflit. Nous n’avons qu’à régler par nous-même en supprimant les mauvaises lignes de codes, ainsi que les lignes de conflits Git. Ensuite, on refait les étapes vues plus haut pour faire un « commit, merge ». Le tout devrait alors fonctionner.
5. Travailler avec des répertoires sur un serveur en ligne.
   * Dans la majorité des cas, vous serez amené à travailler dans des projets qui sont hébergés sur un serveur en ligne pour faciliter le partage des fichiers et le travail d’équipe. Le concept qui suit est important afin de comprendre les deux commandes que vous utiliserez souvent pour aller porter vos changements en ligne, et copier ceux de vos collègues sur votre répertoire local.
   * La première nous permet d’actualiser notre projet local afin de travailler avec une copie à jour des fichiers qui sont en ligne. Cette commande simplifiée ici sera « git pull ». Le terme est simplifié puisque dans de plus gros projets avec plusieurs collaborateurs, il peut être utile d’utiliser la commande « git fetch » qui nous permet de voir ce qui s’est passé sur le projet d’origine sur le serveur, mais sans faire de changement sur votre dossier local.
   * Dans le cas de « git pull », cette commande fait exactement ce que « git fetch » effectue, mais en plus elle intègre directement le tout pour actualiser le projet sur votre ordinateur (votre répertoire local). Il est important de spécifier deux options; la première est le répertoire à vérifier, et la branche de celui-ci. (Dans 95% du temps, vous serez à vérifier l’état du répertoire « origin », ce qui veut dire le répertoire principal, sur lequel tout le monde font des changements).  
     
   * Finalement, la dernière commande qu’on voit est un peu la commande « commit », mais cette fois pour faire le changement sur le répertoire en ligne. Il est important d’avoir la dernière mise à jour du répertoire origine avant de pouvoir faire un « push ». C’est pourquoi nous voyons la dernière commande présentée avant. Après cette commande effectuée, tous vos collègues seront en mesure d’accéder à vos changements fait au projet.  
     
6. Apprendre à travailler avec un hébergeur en ligne spécialisé pour Git.
   * Dans la prochaine section, vous apprendrez alors les avantages à travailler avec un hébergeur comme GitHub qui nous permet de simplifier le travail d’équipe tout en utilisant la technologie Git.