École Polytechnique de Montréal
Département de génie informatique et génie logiciel
LOG1000 – Ingénierie logicielle

Travail pratique #1

Introduction à Git et Make

1. Mise en contexte théorique

Il n'est pas évident de partager le code rédigé au sein d'une équipe, particulièrement si l'équipe est grande et répartie dans plusieurs bureaux ou même pays. Des outils ont donc été développés pour faciliter ce partage. Parmi ces outils, le plus populaire en ce moment est Git.

Il n'est cependant pas suffisant de simplement partager le code. Idéalement, il faudrait aussi partager toutes les informations nécessaires qui permettent de transformer ce code source en programme exécutable : dépendances, commandes à exécuter, etc. Encore une fois, des outils ont été développés pour regrouper toutes ces informations en un seul endroit. L'outil Make n'est peut-être pas le plus populaire aujourd'hui, mais il existe beaucoup de logiciels dont la compilation est documentée avec Make ou en utilisant les mêmes principes de base.

2. Objectifs

Le but de ce travail pratique est de vous faire pratiquer à utiliser un outil de gestion de configuration, Git, et un outil d'automatisation de la construction du logiciel, Make. Les objectifs spécifiques sont de :

- Comprendre l'utilité et le fonctionnement des logiciels de gestion de configuration, aussi appelé outils de gestion de versions.
- Apprendre à utiliser l'outil de gestion de versions Git avec ses diverses commandes et comprendre les différentes situations possibles lors du développement d'un projet avec plusieurs programmeurEs.
- Comprendre l'utilité et le fonctionnement des fichiers de construction Makefile.
- Comprendre et modifier un fichier Makefile de base pour construire et déployer un logiciel en fonction des dépendances entre ses éléments.

3. Mise en contexte pratique

Vous avez construit un logiciel qui implémente la méthode de Newton[1] pour le calcul de la racine carrée. La méthode utilise une approche récursive afin d'approximer la valeur de la racine carrée : plus il y a d'itérations de récursion, plus l'approximation est précise.

Le problème est que vous êtes le seul à avoir le code ! Il faut maintenant le partager avec votre collègue. De plus, une partie du logiciel n'est pas encore été terminée : il faudra donc mettre le code à

1La méthode a en fait été développé initialement par Sharaf al-Dīn al-Ṭūsī, mathématicien perse du XIIe siècle.

jour, et possiblement ajuster la configuration de l'outil de construction du logiciel.

3.1. Outils nécessaires

L'énoncé de ce TP assume que vous avez les outils nécessaires installés, et que vous utilisez une plateforme Linux. Ces outils sont installés sur la machine virtuelle fournie par le département.

Si vous travaillez sur votre propre machine Linux par contre, voici les outils qui sont nécessaires pour ce TP, avec les commandes d'installation pour une machine Ubuntu/Debian :

- Git : sudo apt install git
- GITK : sudo apt install gitk
- GCC : sudo apt install gcc
- Make : sudo apt install build-essential

4. Travail à effectuer

Pour réaliser ce travail, vous devez absolument avoir communiqué avec les chargéEs de cours les matricules des personnes de votre équipe. Les équipes sont constituées de deux personnes. Voir : https://docs.google.com/spreadsheets/d/1K2NRGzPnT7mhkc_PvMYwqu4Kvgh-9BDBsAw7rSicMkg/edit#gid=0

La raison est qu'il faut que votre espace Git sur le serveur du département soit d'abord créé avant de pouvoir vous en servir. Assurez-vous donc d'avoir communiqué les informations, et surtout d'avoir fourni les bons matricules.

Cet espace Git vous servira pour tout le trimestre. Si vous devez changer d'équipe durant le trimestre, il est très important que vous le communiquez aux chargéEs de cours afin que le changement soit fait sur le serveur.

Pour l'ensemble de ce travail pratique, l'un des deux membres de l'équipe prendra le rôle d'Équipier 1, tandis que l'autre sera l'Équipier 2. Dans votre rapport, vous devez cependant spécifier clairement qui est Équipier 1 et qui est Équipier 2. Par exemple :

- Équipier 1 : André Martin #1294444
- Équipier 2 : Lisa Halpern #1295555

4.1. Git : Initialisation de votre répertoire

Votre répertoire Git est déjà en ligne si vous avez créé votre équipe en avance, et est disponible à l'URL suivante : https://githost.gi.polymtl.ca/git/log1000-XX (Remplacez XX par le numéro d'équipe que l'on vous a assigné). Cette URL va s'appeler URL_SERVEUR pour le reste de cet énoncé.

La coéquipièrEs #1 ou la coéquipièrE #2 doit d'abord créer une copie locale du répertoire Git et y ajouter le code source disponible sur Moodle dans le fichier d'archive « tp1.zip ». Pour faire une copie locale du répertoire Git, il faut d'abord que vous créez et que vous vous déplaciez dans le répertoire de votre choix, qui deviendra votre espace de travail pour de TP. Vous devez ensuite exécuter la commande suivante :

- git clone URL_SERVEUR

Entrez vos informations et naviguez dans le répertoire téléchargé. Votre répertoire Git sera également utilisé pour la remise de l'ensemble de vos travaux pratiques au courant de ce trimestre. Par la suite, copiez le contenu du fichier « tp1.zip » dans le répertoire local nouvellement créé et gardez la même hiérarchie de dossiers que celle du fichier d'archive.

Vous devez par la suite naviguer dans le répertoire nouvellement téléchargé.

Normalement, le clonage va ajouter automatiquement un « remote » appelé « origin » dans votre espace Git, qui indique d'où le code a été cloné. Vous pouvez visionner ensuite si le « remote » a été bien défini avec la commande suivante :

- git remote -v

Par défaut, ce « remote » sert à la fois pour le « fetch » (et le « pull ») et le « push ».

Une fois que vous avez mis en place les dossiers et fichiers, vous devez exécuter, à partir du dossier

contenant votre copie locale, les commandes suivantes :

- git add .
- git commit -m "votre message qui explique le commit"
- git push origin master

Ce qui permettra de pousser sur le serveur le code que vous venez de soumettre (« commit ») sur la branche par défaut « master ».

Prenez soin de mettre un commentaire pertinent lorsque vous faites un « commit ». La pertinence de vos commentaires sera évaluée (voir grille d'évaluation).

L'équipièrE ayant travaillé jusqu'à maintenant doit finalement exécuter la commande suivante, afin de s'assurer qu'il est à jour avec le contenu du serveur.

- git fetch origin
- git branch -a

Vous devriez voir les deux branches suivantes :

- master : qui est la branche par défaut dans le dépôt local,
- remotes/origin/master : qui est la branche par défaut sur le serveur distant identifié par « origin ».

Normalement, vous ne devriez voir que ces deux branches. Si vous avez une branche appelée « origin/master », celle-ci contient le code obtenu par le « fetch » et indique que du nouveau code a été obtenu du serveur. Cette branche doit être fusionnée avec la commande « git merge origin/master », qui fusionne cette branche avec la branche dans laquelle vous vous retrouvez en ce moment.

À ce point, le ou la deuxième coéquipièrE doit faire un « git clone » en utilisant la même méthodologie que décrite précédemment sur son propre ordinateur. Le répertoire local Git devrait contenir tous les fichiers que le premier coéquipier a soumis et partagé sur le serveur. Les coéquipièrEs 1 et 2 devraient avoir une copie locale du répertoire Git avec les mêmes fichiers sources.

Finalement, l'unE des deux coéquipièrEs doit exécuter la commande suivante :

- git log

Par la suite, le ou la coéquipière doit prendre une capture d'écran de la sortie provenant du terminal et la mettre dans le rapport sous la **question 4.1.1**. Sur la machine virtuelle Fedora 28, vous pouvez faire une capture d'écran de deux manières différentes :

- En appuyant sur la touche "PrintScrn" du clavier, une capture de l'écran au complet est faite et mise dans un fichier PNG dans le répertoire /home/gigl/Images
- En appuyant sur les touches "Ctrl+Alt+PrintScrn" du clavier, une capture de l'écran au complet est faite et mise dans un fichier PNG sur le bureau de la machine qui héberge la machine virtuelle (donc votre machine Windows ou Apple).

Répondez également à la **question 4.1.2** qui est : Quelle est la différence entre les deux commandes suivantes :

- git log
- git log –p

Note, pour quitter le « git log -p » vous pouvez simplement entrer la commande « q ».

4.2. Git : Modification et gestion de conflits

Les deux coéquipièrEs

Chaque coéquipièrE doit se créer une branche et se déplacer sur cette branche avec les commandes ci-dessous. Remplacez « NOM_DE_LA_BRANCHE » par un nom approprié. Utilisez des noms différents pour les deux coéquipièrEs.

- git branch NOM_DE_LA_BRANCHE
- git checkout NOM_DE_LA_BRANCHE

CoéquipièrE #1

Le ou la coéquipièrE #1 ajoute des commentaires au début du fichier « newton.h ». Suite à cette modification, le ou la coéquipièrE #1 doit exécuter la commande suivante :

- git status

Prenez une capture d'écran de la sortie et collez-la à la **question 4.2.1**.

Est-ce que cette sortie est la sortie attendue ? Expliquez. Placez votre réponse à la **question 4.2.2**.

Le ou la coéquipièrE #1 exécute les commandes suivantes, où « NOM_DE_LA_BRANCHE » correspond au nom de la branche du ou de la coéquipièrE #1 :

- git add .
- git commit -m "votre message qui explique le commit"
- git push origin NOM_DE_LA_BRANCHE

À ce point, le code de la branche a été poussé sur le serveur. Le ou la coéquipièrE #1 exécute la commande suivante :

- git merge master

Prenez une capture d'écran de la sortie et collez-la à la **question 4.2.3**.

Est-ce que cette sortie est la sortie attendue ? Expliquez. Placez votre réponse à la **question 4.2.4**.

Le ou la coéquipièrE #1 exécute les commandes suivantes, où « NOM_DE_LA_BRANCHE » correspond au nom de la branche du ou de la coéquipièrE #1 :

- git checkout master
- git merge NOM_DE_LA_BRANCHE
- git push origin master

Prenez une capture d'écran de la sortie et collez-la à la **question 4.2.5**.

Est-ce que cette sortie est normale ? Expliquez. Placez votre réponse à la **question 4.2.6**.

CoéquipièrE #2

Le ou la coéquipièrE #2 ajoute des commentaires à la fin du fichier « newton.h ». Suite à cette modification, le ou la coéquipièrE #2 doit exécuter les commandes suivantes, où NOM_DE_LA_BRANCHE correspond au nom de la branche du ou de la coéquipièrE #2 :

- git add .

- git commit -m “votre message qui explique le commit”

Le ou la coéquipièrE #2 exécute les commandes suivantes, où « NOM_DE_LA_BRANCHE » correspond au nom de la branche du ou de la coéquipièrE #2 :

- git checkout master
- git merge NOM_DE_LA_BRANCHE
- git fetch origin
- git merge origin/master

Prenez une capture d’écran de la sortie et collez-la à la **question 4.2.7**.

Est-ce que cette sortie est normale ? Expliquez. Placez votre réponse à la **question 4.2.8**.

UnE des deux coéquipièrEs

Sur le terminal d’unE des deux coéquipièrEs, tapez la commande suivante :

- gitk

L’application GITK permet de visualiser graphiquement les commits et les branches qui ont été faits sur Git. Notez que GITK ne présente que les branches locales, et non les branches qui se trouvent sur le serveur. Comme nous n’avons pas fait de « fetch » ni de « pull » des nouvelles branches poussées sur le serveur, GITK ne peut pas les voir.

Une utilité du GITK est de faciliter le « diff », soit d’obtenir la différence entre deux commit. La vidéo suivante explique comment faire :

- https://www.youtube.com/watch?v=PrgdepMiMLU [18 secondes]

Cette vidéo pourrait vous être utile pour le reste du TP.

Le ou la coéquipièrEs #2

Avant de passer à l’étape suivante, il faut s’assurer d’être synchronisés. Le ou la coéquipièrE #2 exécute la commande suivante :

- git push origin master

Prenez une capture d’écran de la sortie de la coéquipièrE #2 et collez-la à la **question 4.2.9**.

Le ou la coéquipièrEs #1

Le ou la coéquipièrE #1 exécute la commande suivante :

- git fetch origin
- git merge origin/master

Prenez une capture d’écran de la sortie du ou de la coéquipièrE #1 et collez-la à la **question 4.2.10**.

Est-ce que cette sortie est normale ? Expliquez. Placez votre réponse à la **question 4.2.11**.

Les deux coéquipièrEs

Cette fois-ci, les deux coéquipièrEs modifient la même ligne de commentaire, soit une ligne de commentaire précédant la fonction « int main(int argc, char** argv) » dans le fichier « main.c ». Assurez-vous de modifier la même ligne !

Les deux coéquipièrEs exécutent ensuite les commandes suivantes :

- git add .
- git commit -m “votre message qui explique le commit”

CoéquipièrE #1

Ensuite, le ou la coéquipièrE #1 exécute la commande suivante :

- git push origin master

CoéquipièrE #2

Et finalement, le ou la coéquipièrE #2 exécute la même commande, soit :

- git push origin master

Prenez une capture d’écran de la sortie du ou de la coéquipièrE #2 et collez-la à la **question 4.2.12**.

On essaie de pousser du code sur le serveur, mais quelqu’unE d’autre nous a devancé ! Il faut d’abord obtenir ce nouveau code. Le ou la coéquipièrE #2 exécute les commandes suivantes :

- git fetch origin
- git merge origin/master
- git status

Comme la même ligne a été modifiée par les deux coéquipièrEs, Git ne sait pas quoi faire. Il faut donc résoudre le conflit. Prenez une capture d’écran de la sortie du ou de la coéquipièrE #2 et collez-la à la **question 4.2.13**.

Le ou la coéquipière #2 ouvre le fichier « main.c » et fait la modification (en effaçant les marques ajoutées par Git) afin de régler le conflit. Gardez les deux commentaires (union des changements). Le ou la coéquipièrE exécute ensuite les commandes suivantes :

- git add .
- git commit -m “votre message qui explique le commit”
- git push origin master

Prenez une capture d’écran de la sortie du ou de la coéquipièrE #2 et collez-la à la **question 4.2.14**.

Et voilà, le conflit est résolu et la décision prise (quelle version du code doit-on garder) a été publiée sur le serveur.

CoéquipièrE #1

Il ne reste plus qu’à le ou la coéquipièrE #1 de se synchroniser avec son ou sa partenaire. Le ou la coéquipièrE #1 exécute les commandes suivantes :

- git fetch origin
- git merge origin/master

Et voilà, les deux coéquipièrE sont maintenant synchroniséEs.

4.3. Make : Modification du code

Les deux coéquipièrEs

Les deux coéquipièrEs exécutent les commandes suivantes :

- make
- ./newton 49 3

La première commande compile le code. Si celle-ci produit une erreur, c'est possiblement parce que vous n'avez pas installé les outils nécessaires. Relisez la section 3.1 sur le sujet.

La deuxième commande exécute l'application produite par la compilation du code source, et l'édition de liens (« linking ») des fichiers objets. On fournit deux paramètres à l'application lors de son exécution, soit la valeur « 49 » et la valeur « 3 ».

Prenez une capture d'écran de la sortie de chaque coéquipièrE et collez-la à la **question 4.3.1** et à la **question 4.3.2**.

UnE des deux coéquipièrEs

Vous êtes maintenant capables de produire l'application sur demande. Vous pouvez donc la modifier et observer l'effet du changement.

L'application actuelle fait une approximation du calcul de la racine carrée. Le travail que vous devez faire est de modifier le « main.c » afin d'indiquer à quel point cette approximation est bonne, en comparant avec le calcul de la racine carrée de la librairie « cmath ». Le code a déjà été rédigé et se trouve dans un fichier appelé « comparison.h » et « comparison.c ».

Votre travail consiste donc à modifier le fichier « main.c », et possiblement le « Makefile », afin d'intégrer une mesure de la qualité de l'approximation. N'oubliez pas de modifier les règles existantes si nécessaire !

Une fois le travail fait, le ou la coéquipièrE qui aura fait le travail exécute la commande suivante :

- make clean
- make
- ./newton 49 3

Prenez une capture d'écran de la sortie du ou de la coéquipièrE qui a fait le travail et collez-la à la **question 4.3.3**.

Il ne reste plus qu'à partager cette modification sur le serveur. Le ou la coéquipièrE qui aura fait le travail exécute les commandes suivantes :

- make clean
- git add .
- git commit -m "votre message qui explique le commit"
- git fetch origin
- git push origin master

Normalement, le « git fetch origin » ne devrait rien faire, vu que l'autre coéquipièrE n'a rien poussé sur le serveur.

Prenez une capture d'écran de la sortie du ou de la coéquipièrE qui a fait le travail et collez-la à la **question 4.3.4**.

4.4. Make : Ajustement du Makefile

UnE des deux coéquipièrEs

Le Makefile, dans sa forme actuelle, a l'avantage d'être très explicite vu que chaque fichier est identifié. Il peut cependant être laborieux à modifier. Afin de faciliter le travail futur, unE des deux coéquipièrEs doit modifier le Makefile pour utiliser le plus de variables et de symboles ($@, $<, $^) que possible. Limitez-vous cependant aux symboles vus en classe.

Une fois que la modification est faite, le ou la coéquipièrE qui a fait le travail exécute les commandes suivantes :

- make clean
- make
- ./newton 49 3

Prenez une capture d'écran de la sortie du ou de la coéquipièrE qui a fait le travail et collez-la à la **question 4.4.1**.

Il faut par la suite publier vos changements sur le serveur. Le ou la coéquipièrE qui aura fait le travail exécute les commandes suivantes :

- git add .
- git commit -m "votre message qui explique le commit"
- git fetch origin
- git push origin master

Et nous pourrons donc voir le contenu de votre Makefile final.

4.5. Make : Ajout de nouvelles commandes

Il est possible d'exécuter n'importe quelle commande avec Make. L'objectif de cette partie est d'ajouter une fonctionnalité à votre Makefile pour répondre à un nouveau besoin de votre équipe de développement.

Voici les exigences à implémenter :

- Vous devez créer une nouvelle règle, qui sera appelée sur un « make install ».
- Cette règle devra créer un répertoire « install » si celui-ci n'existe pas déjà.
- Cette règle devra compiler l'exécutable « newton » si celui-ci n'existe pas déjà ou si l'exécutable actuel n'est pas à jour avec les dernières modifications du code source.
- Cette règle devra copier l'exécutable « newton » dans le répertoire « install », quitte à remplacer l'exécutable déjà présent s'il est là.

Une fois votre Makefile modifié et fonctionnel, assurez-vous de pousser la dernière version de celui-ci sur le serveur Git.

Par la suite, le ou la coéquipièrE qui a fait le travail exécute les commandes suivantes :

- make clean
- make install

Prenez une capture d'écran de la sortie du ou de la coéquipièrE qui a fait le travail et collez-la à la **question 4.5.1**.

5. Rétroaction

Nous travaillons à l'amélioration continue des travaux pratiques de LOG1000. Ces questions peuvent être répondues très brièvement.

- Combien de temps avez-vous passé sur le travail pratique, en heures-personnes, en sachant que deux personnes travaillant pendant trois heures correspondent à six heures-personnes. Est-ce que l'effort demandé pour ce travail pratique est adéquat ? Placez votre réponse dans la **question 5.1**.
- Avez-vous des recommandations à faire afin d'améliorer ce travail pratique ? Placez votre réponse dans la **question 5.2**.

6. Astuces

6.1. Valider que la soumission s'est bien faite

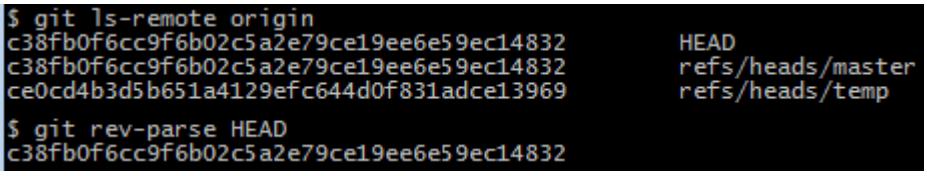
Si vous voulez vous assurer que vous avez bien fait votre remise sur le serveur vous pouvez faire les commandes suivantes :

- git ls-remote origin

En supposant que le nom de votre « remote » est « origin ». Cette commande indique le nom des commits pour les branches, incluant HEAD. Vous pouvez vérifier que le commit pointé par le HEAD sur le « remote » est le même qu'en local avec la commande :

- git rev-parse HEAD

Tel que démontré dans la figure ci-dessous. On voit que l'ID du commit pointé par HEAD sur le serveur est le même ID du commit pointé par HEAD dans notre dépôt local.



6.2. Erreur commune à éviter

Comme vous avez un accès SSH sur le serveur, il est possible d'exécuter des commandes Git directement sur le serveur. **Vous ne devriez jamais exécuter des commandes Git directement sur le serveur pour ce travail pratique !** il s'agit d'un excellente manière de corrompre votre espace Git. En pratique, exécuter des commandes directement sur le serveur n'est fait qu'en dernier recours et avec beaucoup de parcimonie. Si vous n'arrivez pas à produire les sorties d'écran demandées, expliquez dans le gabarit la raison de votre problème.

6.3. Guide sur l'utilisation des Makefile

Voici un petit guide d'introduction sur les Makefile, si les notes de cours sont insuffisantes pour vous : https://opensource.com/article/18/8/what-how-makefile

7. Livrable à remettre, procédure de remise et retard

Votre rapport devra être contenu dans un fichier nommé « rapport.pdf », qui doit être soumis à travers votre répertoire Git, dans un dossier appelé TP1. Veuillez mettre vos réponses dans le fichier « gabarit.doc » qui est fournis avec les fichiers sources. N'oubliez pas de faire « git add », « git commit » et « git push » sur le rapport à la fin de votre TP pour vous assurer que le rapport soit soumis !

Seule une remise électronique est exigée. **La remise doit se faire sur Git avant midi (12h00), le 4 février 2019**. Si jamais il y avait des problèmes avec votre serveur Git, envoyez votre rapport par courriel à votre chargéE de travaux pratiques en expliquant la situation. Les courriels des chargéEs de travaux pratiques se trouvent sur le site Moodle du cours.

7. Barème de correction

Le barème de correction est décrit dans le gabarit.