POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

GIT : EXEMPLE D'UTILISATION SIMPLE

25 SEPTEMBRE 2023

JÉRÔME COLLIN, CHARLES DE LAFONTAINE, MERIAM BEN RABIA
GIGL | Polytechnique Montréal

N.B. Le masculin est utilisé pour alléger le texte.

TABLE DES MATIÈRES

GIT : EXEMPLE D'UTILISATION SIMPLE

1. CONTEXTE

L'apprentissage de *Git* n'est pas simple puisque l'outil lui-même est complexe tout comme les processus de développement logiciel dans lesquels il est utilisé au quotidien à travers le monde. Il convient alors d'y aller par étapes et de débuter pas un exemple simple et représentatif.

L'exemple est basé sur une équipe de deux, **Julie** et **Ahmed**, qui travaillent de façon très séparée et ne s'entendent pas très bien ! Ce conflit d'équipe mènera à un conflit lors de la fusion du code avec *Git*… L'exemple est aussi entremêlé de commandes *Linux* assez courantes pour illustrer une situation représentative de programmation.

On suppose aussi qu'il y a un serveur *Git* déjà en place et configuré correctement à l'adresse fictive **`https://serveur-git.polymtl.ca/`** et qui contient un sous-répertoire appelé **`ing1234`** pour le cours du même nom et que Julie et Ahmed possèdent leur espace privé appelé **`ing1234-07`** sous **`ing1234`** sur ce serveur puisqu'ils suivent ce cours, lui aussi fictif.

2. JULIE, LA MATINALE, DÉBUTE LE TRAVAIL TÔT AU DÉBUT DE LA JOURNÉE…

Pour débuter, Julie ouvre un terminal sous *Linux* et lance la commande `pwd` pour déterminer à quel endroit elle se situe dans son arborescence de fichiers sur son ordinateur. Par la suite, elle crée un répertoire local appelé **`ing1234`** et navigue jusqu'à ce répertoire (`cd`). Aucune interaction n'a encore eu lieu avec *Git*, mais il s'agit simplement de bonnes pratiques pour bien ordonner des documents à venir dans un endroit séparé concernant ce cours des autres cours ou activités sur l'ordinateur de Julie.

Julie prend ensuite une copie de l'entrepôt avec la commande *Git* appelée **`clone`**. Ceci réplique l'arborescence de l'entrepôt localement sur l'ordinateur de Julie. Le répertoire **`ing1234-07`** apparaît. Sous ce répertoire, la commande *Linux* `ls -a` montre aussi qu'il y a un répertoire appelé **`.git`** qui est justement utilisé par *Git* pour y stocker l'information nécessaire à son fonctionnement. Bien que ce ne soit pas habituel, un fichier **`LISEZMOI.txt`** (**`README.txt`**) est aussi créé. Julie va ajouter un répertoire appelé **`travail1`** (en supposant qu'il y aura plus d'un travail à faire dans ce cours).

```
$ pwd
/home/treju
$ mkdir ing1234
$ cd ing1234
$ git clone https://serveur-git.polymtl.ca/ing1234
$ ls
ing1234-07
$ cd ing1234-07
$ ls
LISEZMOI.txt
$ ls -a
.    ..    .git    LISEZMOI.txt
$ mkdir travail1
$ cd travail1
$ pwd
/home/treju/ing1234/ing1234-07/travail1
```

Julie va, par la suite, aller chercher avec un fureteur internet des fichiers mis en place sur le site du cours par le professeur et les placer dans son répertoire de travail actuel. Elle se retrouve donc avec un *Makefile* et un fichier *C++* appelé **`exempleSimple.cpp`**. Ce dernier fichier sera renommé **`travail1.cpp`** pour lui donner un nom plus significatif. Elle peut par la suite démarrer l'éditeur *VS Code* et commencer à compiler son code (ce qu'elle peut avoir à faire plusieurs fois au fur et à mesure de ses modifications au fichier **`travail1.cpp`** ou au *Makefile* lui-même).

```
$ ls
Makefile   exempleSimple.cpp
$ mv exempleSimple.cpp travail1.cpp   [renomme le fichier]
$ ls
Makefile   travail1.cpp
$ code . &
$ make                                [ou make install si nécessaire]
```

Arrive le moment où Julie a un code qui fonctionne, mais n'est pas complet. Elle a toutefois un résultat approprié pour une première version à placer sous *Git*. Elle enchaîne alors les commandes **`git add`**, **`git commit`** et **`git push`**. Elle vérifie même son résultat avec les commandes **`git log`** et **`git status`** juste pour être certaine qu'elle a bien soumis ses modifications au serveur.

```
$ ls
Makefile    travail1.cpp    travail1.o    travail1.exe     ...
$ git add Makefile travail1.cpp
$ git commit -m "fonction, JT: première partie du devoir"
$ git push
$ git log travail1.cpp | grep JT
fonction, JT: contrôle de la led
$ git status
On branch master
Your branch is up to date with 'origin/master'.
...
Untracked files:
(use "git add <file>..." to include in what will be committed)
travail1.o
travail1.exe
...
```

Julie termine son travail et ne communique pas ses changements à Ahmed par courriel ou autre forme de messagerie électronique, ni même de vive voix.

3. AHMED POURSUIT LE TRAVAIL PLUS TARD EN SOIRÉE…

Comme Ahmed débute, il doit faire localement sur son ordinateur un peu le travail qu'a eu à faire Julie au départ. Ahmed va placer ses fichiers pour ce cours sous le répertoire **`projet1`**, et nom **`ing1234`** ce qui ne change rien puisque ceci représente un choix de l'usager et n'est pas visible sur le serveur *Git* de toute façon. Après avoir pris copie de l'ensemble des fichiers du serveur avec la commande **`git clone`**, la commande **`ls -alF`** permet à Ahmed de soupçonner que du travail a déjà été fait.

Ahmed consulte le contenu du fichier **`LISEZMOI.txt`** et fait ce qui y est inscrit: il supprime le fichier, non seulement localement sur son ordinateur, mais aussi sur le serveur *Git* car il aime tenir ses fichiers et répertoires de façon très propre sans rien de superflu et d'inutile.

La commande **`git log`** lui permet de constater ce qu'a fait Julie en lisant le commentaire accompagnant la première version du fichier **`travail1.cpp`**.

```
$ mkdir projet1
$ cd projet1
$ git clone https://serveur-git.polymtl.ca/ing1234
$ cd ing1234-07
$ ls -alF
-rwxr-xr-x    1      bouah  dialout       4096   Aug 27 17:16  .git/
-rw-r--r--    1      bouah  dialout       15644  Aug 27 17:16  LISEZMOI.txt
-rwxr-xr-x    1      bouah  dialout       4096   Aug 27 17:16  travail/
$ cat LISEZMOI.txt
Premier fichier de votre entrepôt... Vous pouvez le supprimer.
$ git rm LISEZMOI.txt
$ git add .
$ git commit -m "AB: ménage..."
$ cd t                [Ahmed presse TAB pour l'autocomplétion « travail »]
$ ls
Makefile   travail1.cpp
```

```
$ git log travail1.cpp
commit     27daac560d18de86cbd2c79fd932a8d07c009f09
Author:    Julie Tremblay <julie.tremblay@polymtl.ca>
Date:      Mon Aug 27 02:30:51 2022 +0000
fonction, JT: première partie du devoir
```

Ahmed poursuit en regardant rapidement le code écrit par Julie avec la commande *Linux* **more**. Il peut à son tour démarrer *VS Code* et poursuivre le travail débuté par Julie. Il pourra compiler plusieurs fois au fur et à mesure des modifications à son code. Éventuellement, lui aussi sera satisfait de ce qu'il obtient et effectuera les commandes **git add**, **git commit** et **git push** pour créer une deuxième version sur le serveur. Comme Julie, il vérifie son travail avec la commande **git log**.

```
$ more travail1.cpp
#include <stdio.h>
main()
...
$ code . &
$ make                          [Commande répétée à plusieurs reprises…]
$ git add lab1.cpp
$ git commit -m "fonction, AB: ajout de la deuxième partie"
$ git push
$ git log lab1.cpp
...
fonction, JT: première partie du devoir
...
fonction, AB: ajout de la deuxième partie
```

Ahmed termine sa session de travail et ne communique pas non plus ses progrès à Julie.

4. JULIE REVIENT FINIR SON TRAVAIL

Julie revient le lendemain matin pour terminer son travail en pensant qu'Ahmed n'a rien entre temps et modifie localement ses fichiers de la veille. À la fin, elle veut créer ce qu'elle croit sera la deuxième version.

```
$ cd ing1234
$ cd ing1234-07                                    [Sans aucun git pull]
$ cd travail
$ code . &
$ make                          [Commande répétée à plusieurs reprises...]
$ git add travail1.cpp
$ git commit -m "JT: deuxième partie du travail"
$ git push
#include <stdio.h>
main()
...
$ code . &
$ make                          [Commande répétée à plusieurs reprises...]
$ git add lab1.cpp
$ git commit -m "fonction, AB: ajout de la deuxième partie"
$ git push                                                  ⚠ [Conflit !]
$ git log lab1.cpp
```

Julie, après la commande **`cd ing1234-07`**, n'a pas fait de commande **`git pull`** pour récupérer les modifications effectuées par Ahmed et sa version 2. Résultat: elle travaille avec un entrepôt *Git* qui **n'est pas à jour localement sur son ordinateur**. En essayant d'envoyer sa version sur le serveur, celui-ci ne sera pas en mesure de réconcilier ce qu'il a avec ce que soumet Julie. Le conflit survient immédiatement.

Évidemment, Julie n'a pas utilisé la bonne commande **`git pull`** au bon moment, mais l'exemple illustre aussi que ce conflit survient, par la faible communication entre les deux membres de l'équipe sur le partage des étapes du travail à effectuer.

5. POUR ALLER PLUS LOIN

Il existe quelques bons résumés sur une page des commandes Git :

- Un bon résumé pour débuter avec *Git* style « *cheat sheet* »
- Aide-mémoire des commandes *Git*
- Aide-mémoire de *GitLab*
- Entraînement sur *GitHub*