Département Informatique
R5.07 – TP Git 1 (2ème partie)
Responsable : X.Roirand
Durée : 90mn machine (pour la 2nde partie)

Le but de ce TP va être de travailler sur un gestionnaire de version appelé Git et de traiter les cas courant d'utilisation. La bonne maitrise d'un gestionnaire de version est la base dans la CI/CD.

1) Connection sur la machine distante

Pour se connecter sur la machine distante, il faut utiliser une connections ssh. C'est une connexion sécurisée qui permet de ne pas faire passer le mot de passe en clair sur le réseau. Vous pouvez utiliser le programme « putty » sur windows ou « ssh » sur linux.

La connexion se fait en 2 étapes, dans le cadre de nos TP. il faut d'abord se connecter sur une sandbox dont l'adresse IP est 149.202.85.73. Le login est student et le mot de passe ?Student_56. Une fois sur cette sandbox, connectez-vous sur votre machine distante, pour cela il faut connaitre le numéro que l'enseignant vous a donné, et ajouter ce numéro au nom : machine pour obtenir le nom de la machine sur laquelle vous devez vous connecter.

Exemple : si votre numero est le 13 alors il faut faire un ssh sur la machine machine13

Ensuite le login et mot de passe sur votre machine distante sont :

login : student
mot de passe : ?<votre numero>Student_56

donc si votre numero est le 13 alors le mot de passe est ?13Student 56, comme suit :

login : student
mot de passe : ?13Student_56

Une fois sur votre machine distante, vous pouvez l'administrer totalement.

Pour pouvoir noter le TP, il faut que vous mettiez dans un fichier nommé /root/info.txt
Votre nom et prénom, ou vos noms et prénoms si vous êtes plusieurs. Attention, pas de fichier = zéro !

7) Reset d'un fichier sur la version du repository

   Modifiez le fichier system/config1.txt en rajoutant une ligne.

   Vérifiez qu'avec "git status" le fichier est bien dans la section "Changes not staged for commit".

   Le but ici va être de revenir à la version du repository, donc celle qui était là avant que vous rajoutiez votre ligne car par exemple ici on s'est trompé et on veut retrouver le fichier d'origine, pour ca utilisez la commande "git checkout <chemin_du_fichier>.

   Une fois cette commande tapée, un "git status" devrait nous dire que tout est clean "nothing to commit, working tree clean".

8) Ajout d'un fichier oublié dans un commit (avant de l'avoir poussé)

   Cas courant, vous avez preparé un commit avec des modifications et vous vous apercevez que vous avez oublié un fichier dans la liste des fichiers du commit. Pour rajouter le fichier, il

suffit de faire un “git add” et ensuite “git commit –amend”. Cette option “— amend” permet de dire à Git que le fichier doit etre ajouté dans la liste du dernier commit. ATTENTION: cela ne fonctionne que si le commit n’a pas été envoyé vers Gitlab.

9) Annulation d’un commit (avant de l’avoir poussé)

On va voir le cas ou on vient de faire un commit qu’on n’a pas encore poussé et on veut annuler ce commit car finalement on n’en veut plus, pour x raisons.

Pour cela, créer un fichier system/config3.txt et rajoutez dedans 2 ou 3 lignes. Rajoutez ce fichier dans un commit, et tapez avec “git log” pour vous assurer que le commit est bien présent et notez le commit id du commit precedent, ex:

```
myhost> git log

commit 34b0a206a4feac8779cfc775cee9a58de5ebc5c9 (HEAD -> master,
origin/master, origin/HEAD)
Author: Xavier Roirand <xavier@toto.com>
Date:      Thu Feb 6 17:22:41 2020 +0100

    Re Re Bla bla

commit
24eee27ac4aa61cdad8c3d4d86a95ea978500903
Author: Xavier Roirand <xavier@toto.com> Date: Thu
Feb 6 17:04:14 2020 +0100

[…]
```

Ici le commit id précedent est le 24eee27ac4aa61cdad8c3d4d86a95ea978500903.

Maintenant utilisez la commande “git reset <votre_commit_id>” qui doit annuler votre commit.

Vérifiez avec la commande “git log” que le commit à annuler a bien disparu. Notez que le fichier system/config3.txt est toujours present.

Note: il aurait été possible de revenir complètement en arrière avec la suppression des modifications faites par le commit (ici la creation du fichier system/config3.txt) avec l’ajout de l’option “–hard” dans la command “git reset”.

Recommencez en rajoutant le fichier system/config3.txt dans un commit et faite la même étape que précédemment mais avec un hard reset, et constatez la différence avec l’étape précédente.

10) Gestion d’un conflit

Nous allons voir ici le cas le plus courant et le plus intéressant qui est:

Comment résoudre un conflit, c’est à dire, comment faire pour gérer un cas ou quelqu’un a modifié un fichier, a fait un commit et l’a poussé dans Gitlab alors qu’on a aussi un commit qui contient une modification sur le même fichier et donc on ne peut pas pousser notre commit tant que le conflit n’est pas résolu.

Il y a deux solutions pour résoudre ce problème:

Premiere solution: rentrer le + vite ses commits pour éviter que le conflit soit à résoudre par soit et laisser ca aux autres. Sérieusement c’est une option intéressante car elle nous fait prendre conscience qu’un commit ne doit pas contenir trop de modifications mais plutôt un minimum de modifications, quitte à faire plein de commits. Pourquoi ? Parce qu’il est plus

facile de revenir en arrière sur 2 ou 3 commits quand une regression dans le fonctionnement du programme arrive plutôt qu'un commit qui contient 100 fichiers puisqu 'il va falloir trouver ou parmi les 100 fichiers se trouvent l'erreur. Donc attendre et cumuler plein de modifications dans un commit n'est certainement pas une bonne idée, en entreprise on privilégie des petits commits.

Deuxième solution: résoudre le conflit manuellement. C'est ce que l'on fait si on n'a pas réussi à faire la solution numéro 1.

Pour simuler le problème on va utiliser l'interface Gitlab pour modifier un fichier, faire pareil en local en modifiant le même fichier mais différement, puis tenter de pousser son commit local, et normalement on devrait avoir un souci à résoudre.

On commence par modifier le fichier info.txt dans Gitlab en rajoutant la ligne

Modification par Gitlab

Pour ce faire, allez sur "Repository", puis "Files" et ensuite cliquez sur "info.txt" et le boutton "Edit", faite la modification, puis "Commit change".

Ensuite on fait une autre modification en local, on rajoute la ligne:

Modification en local

Puis on créé un commit avec cette modification et on fait un "git pull –rebase". Souvenez-vous, cette commande permet de répercuter en local toutes les modifications faites à distance, donc cette faites dans Gitlab et donc on devrait avoir un problème comme ceci:

```
$ git pull --rebase
Enter passphrase for key '/c/Users/Xavier/.ssh/id_rsa':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

```
From ssh://gitlab.iu-vannes.org:12220/toto/project1
   69358bd..0055642 master       -> origin/master
First, rewinding head to replay your work on top of it...
Applying: local modif.
Using index info to reconstruct a base tree...
M       info.txt
Falling back to patching base and 3-way merge...
Auto-merging info.txt
CONFLICT (content): Merge conflict in info.txt
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch Patch failed
at 0001 local modif.
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase -- abort".
```

La partie en gras au-dessus est celle qui signifie qu'on a un souci avec un conflit, forcément le même fichier a été modifié de source différentes (Gitlab vs local).

Pour résoudre le conflit manuellement, il faut éditer le fichier (ou les fichiers) en conflit, ici le fichier info.txt, et voici ce qu'on trouve dedans:

```
[…]
<<<<<<< HEAD
Modification par Gitlab
=======
Modification en local
>>>>>>> test de conflit, modif local.
```

La syntaxe affichée est en 2 parties:

```
<<<<<< HEAD
……
```

Jusqu'a ====

Est ce qui vient de Gitlab

Et tout ce qui est de ====

```
…….
>>>>>>
```

Est ce qui est en local

L'idée pour la resolution est que l'utilisateur (vous) modifies le fichier en décidant ce qui doit êtré gardé de ce qui ne doit pas être pas gardé. Ici Il y a 3 solutions:

Solution 1) le fichier final doit être:

```
[…]
Modification Gitlab
```

Si par exemple on s'apercoit que la ligne "Modification en local" qu'on voulait rajouter n'a pas lieu d'être (par exemple la ligne Modification Gitlab fait la même chose que ce

qu’on voulait mais en mieux) alors on ne garde pas ce qu’on voulait mettre.

Solution 2) le fichier final doit être:

[…]
Modification en local

Si par exemple on s’apercoit que la ligne “Modification Gitlab” n’a pas lieu d’être dans ce fichier et qu’on ne veut garder que notre modification à nous.

Solution 3) le fichier final doit être:

[…]
Modification Gitlab
Modification en local

Ou

[…]
Modification en local
Modification Gitlab

Si par exemple on s’apercoit que la ligne “Modification Gitlab” doit être gardé mais aussi la ligne “Modification en local” qu’on avait rajouté. Après reste à gérer l’ordre des lignes mais aussi des solutions plus complexes ou parfois un algo a été modifié d’un coté et de l’autre on rajoute des choses dans l’algo. En general c’est cette solution là qui arrive 90% du temps et c’est aussi celle là qu’on va utiliser ici.

Bref une fois qu’on est content et qu’on a gardé tout ce qu’on voulait et supprimer le reste (cela inclut les lignes <<<<< et >>>>>> et ====== qu’on doit virer) alors on fait un “git add” du fichier ainsi modifié.

L’étape suivant est de valider le “git rebase”. Souvenez-vous, on était en train de faire “git pull –rebase” quand le conflit est arrive. Pour ca on fait “git rebase –continue”. Si jamais on ne sait plus trop dans quell état le repository local est, on fait un “git status” et ca nous dit:

```
rebase in progress; onto 0055642
You are currently rebasing branch 'master' on '0055642'.
  (all conflicts fixed: run "git rebase --continue")

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   info.txt
```

On voit ici la commande “git rebase –continue” a faire ou “git rebase –abort” si on veut annuler le git pull.

Donc faites la commande “git rebase –continue”

Vérifiez ensuite avec “git log” que le commit que vous venez de faire est bien listé

Faites un push du commit et verifiez dans Gitlab qu’il est bien présent et que le fichier info.txt contient bien les modifications que vous avez fait.

11) Travail sur une branche

On va pouvoir passer au choses sérieuses et bosser sur une branche. C'est ce qu'on fait en géneral quand on développe de nouvelles fonctionnalités.

Créez une branche nommée "feature1" dans votre repository local. Modifiez maintenant le fichier info.txt pour rajouter la ligne:

ligne feature1

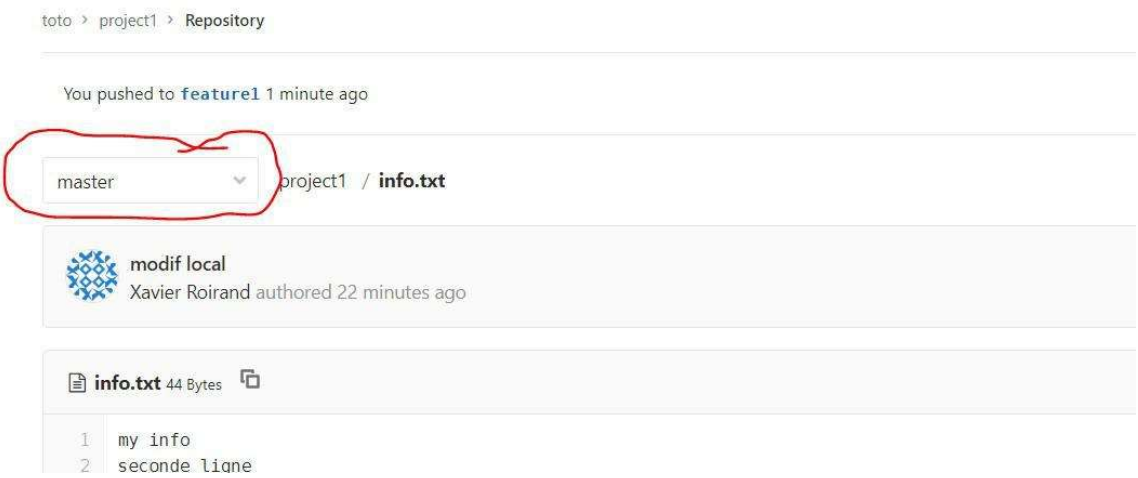
Créez un commit et ensuite poussez ce commit sur Gitlab.

Vous devriez avoir un problème car la branche n'existe pas sur Gitlab, donc Git ne sait pas dans quelle branche remote (Gitlab) pousser ce commit. Il y a du coup 2 solutions:

> Indiquer a Git lors du push, c'est la solution propose par le message affiché lors de votre git push qui a échoué.
> Indiquer à Git comment ce push et les futurs push doivent être fait en faisant une commande du genre "git branch …", c'est cette commande la qui doit être privilégiée.

Allez sur Gitlab et verifiez que dans "Repository" et "Branches", vous voyez bien 2 branches, la branche "master" qui est la branche principale qui a été créée au depart et la branche "feature1" qui est celle que vous venez de créer.

Allez voir aussi le contenu du fichier "info.txt" et choisissez grace a la liste déroulante:



la branche "master", puis la branche "feature1" et constatez que le contenu du fichier diffère selon la branche choisie.

12) Merge entre 2 branches

Quand on développe des fonctionnalités on les fait dans des branches et ensuite on merge les branches c'est à dire qu'on prend les modifications (par rapport à la branche de reference qui est master) d'une branche de développement et on les insert dans une autre branche (souvent la branche master).

Comme dans la vie d'un développeur lors d'un merge request (operation qui consiste à

merger 2 branches), les choses en général ne se passent pas sans conflit, on va volontairement créér un conflit, en modifiant en local le fichier “info.txt”, pour cela:

Trouvez la commande git qui permet d’afficher la branche courante et verifiez que vous êtes bien dans la branche “feature1”. Changez de branche et mettez vous dans la branche “master”. Modifiez le fichier info.txt en rajoutant la ligne:

ligne master

Ajouter ce fichier dans un commit et poussez le commit.

Donc si on se pose 2 secondes et qu’on réfléchit à la situation. On a fait une 1ere modification du fichier “info.txt” dans la branche feature1, puis une autre modification dans le fichier “info.txt” dans la branche master. Lorsqu’on va merger les branches, on devrait donc avoir un conflit à résoudre, similaire à celui qu’on a eu lors du dernier “git pull –rebase”.

Pour faire cette operation de merge request on peut le faire en local ou dans Gitlab. Pour simplifier ici on va le faire dans Gitlab.

Dans Gitlab, allez dans “Repository”, puis “Branches” et ensuite vous devez voir les branches master et feature1. Sur la ligne “feature1” à droit vous devez trouver un bouton “Merge request”, cliquez dessus. Renseignez le champ description et cliquez sur “Merge request”.

Vous devez arrivez sur un écran ou il est affiché: “There are merge conflicts” et le bouton “Merge” est désactivé car il faut d’abord résoudre le conflit.

Cliquez sur le bouton “Resolve conflicts” et vous avez alors un écran ou on vous propose plusieurs solutions comme lors du “git pull –rebase”:

1ere solution) garder le fichier info.txt de la branche master
2eme solution) garder le fichier info.txt de la branche feature1
3eme solution) “Edit inline” qui permet de choisir les bouts qu’on veut garder du fichier info.txt de la branche “master” et ce qu’on veut garder du fichier de la branche “feature1”

Choisissez cette dernière possibilité et modifie le fichier pour avoir les 2 lignes “ligne feature1” et “ligne master” en mm temps dans le fichier, puis cliquez sur “Commit to source branch et ensuite sur le bouton “Merge” et “Close merge request”.

Maintenant allez dans votre repository local et faites une mise à jour de votre repository (c’est toujours Gitlab la reference pour la mise à jour) et constatez que la merge request est aussi présente dans votre repository et que le fichier info.txt contient bien les 2 lignes “ligne feature1” et “ligne master”, well done