Découverte de Git

Yoann Dufresne, Mikaël Salson, Jean-Stéphane Varré

M1 Informatique, Université Lille, septembre 2018

L'objectif de ce TP est de prendre en main Git et de préparer un dépôt pour le développement du PJE.

Contenu du TP

1 Créer un dépôt sur le GitLab du FIL (à faire individuellement) 1

2 Premiers pas (à faire individuellement) 1
- 2.1 Commit et push . . . 1
- 2.2 Passons aux branches . . . 2
- 2.3 De l'université à la maison et vice-versa . . . 3

3 Jouer à plusieurs 5
- 3.1 Chacun sa branche . . . 5
  - 3.1.1 Dépôt initial . . . 5
  - 3.1.2 Créer un premier projet fonctionnel à plusieurs . . . 5
- 3.2 Gérer les conflits, contribuer sur les mêmes fichiers . . . 5
  - 3.2.1 Dépôt initial . . . 5
  - 3.2.2 Développement d'une nouvelle impression . . . 6
  - 3.2.3 Développement d'une nouvelle fonctionnalité de calcul . . . 6
  - 3.2.4 Mise en place d'une nouvelle version . . . 6
  - 3.2.5 Aller sur la branche master sans abandonner ni diffuser les développements en cours . . . 6
  - 3.2.6 Récupérez la version courante sans abandonner, ni diffuser les développements en cours . 6
- 3.3 Utilisation de Gitlab pour contribuer à plusieurs . . . 6

4 Projet PJE 7

1 Créer un dépôt sur le GitLab du FIL (à faire individuellement)

— Connectez-vous au GitLab du FIL (`gitlab-etu.fil.univ-lille1.fr`) avec vos identifiants du FIL.
— Créez un projet nommé `test1`
— Exécutez les instructions indiquées dans 'Git global setup' et 'Create a new repository' en ligne de commande
— Sur l'interface GitLab, observez le contenu de l'ensemble des sous-menus du menu 'Repository'

2 Premiers pas (à faire individuellement)

2.1 Commit et push

Situation : je travaille seul sur mon dépôt qui n'est que sur une machine.

Q 1. Exécutez les commandes Git permettant d'aboutir à un dépôt dont la vue 'Files' est la suivante :

master test1 / + Find File

Name	Last Update	Last Commit	History
README.md	less than a minute ago	add README	

README.md

mais dont `git status` donne le résultat ci-dessous.

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

new file:   hello.c
```

Q 2. Constatez que la vue 'Graph' est la suivante :

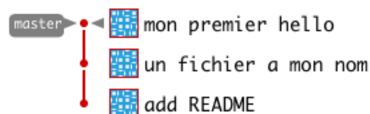
Q 3. Exécutez les commandes Git permettant d'aboutir à un dépôt dont `git status` fournit le résultat suivant :

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Q 4. Exécutez les commandes Git permettant d'aboutir à un dépôt dont `git status` fournit le résultat suivant :

```
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

et la vue 'Graph :



2.2 Passons aux branches

Situation : je travaille seul sur mon dépôt qui n'est que sur une machine et je veux faire un test de développement.

Q 5. Exécutez les commandes Git permettant d'aboutir à un dépôt dont la vue 'Graph est :

et le contenu de `hello.c` :

```
#include <stdio.h>

int main (int argc, char **argv) {
   printf("Hello world!\n");
   return 1;
}
```

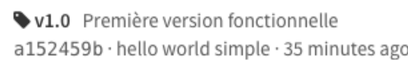
Q 6. Compilez pour obtenir un programme `hello`.

Q 7. Revenez sur la branche `master`. Que contient `hello.c` ? Pourquoi voit-on le programme `hello` ?

Situation : mon test de développement est opérationnel, je le mets sur la branche principale, et j'en fais une version.

Q 8. Exécutez les commandes Git permettant d'aboutir à un dépôt dont la vue 'Graph est :

Q 9. Exécutez les commandes Git permettant d'aboutir à un dépôt dont la vue 'Tags' est :



2.3 De l'université à la maison et vice-versa

Situation : je travaille seul sur mon dépôt et je veux poursuivre mon travail chez moi.

Q 10. Pour simuler le travail ailleurs, créez un répertoire 'maison' quelque part sur votre compte puis rendez-vous y.

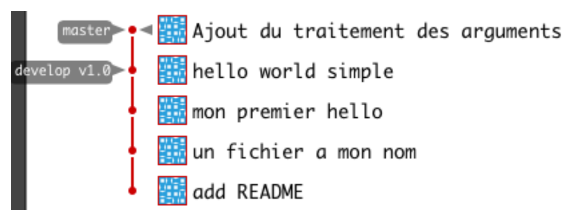
Q 11. Dans ce répertoire obtenez une instance de votre dépôt.

Q 12. En ne réalisant des modifications que dans ce répertoire, transformez votre `hello.c` de manière à ce que ce soit :

```
#include <stdio.h>

int main (int argc, char **argv) {
   if (argc == 0) {
      return 0;
   } else if (argc == 1) {
      printf("Hello world!\n");
   } else {
      printf("%s\n",argv[1]);
   }
   return 1;
}
```

puis faites-en sorte que la vue 'Graph soit celle-ci :

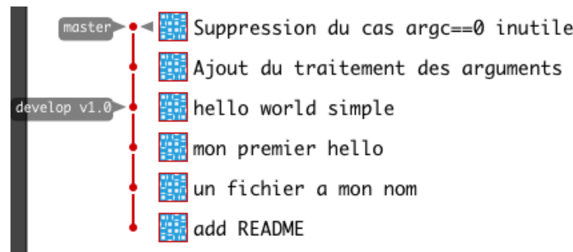


Q 13. Retournons à l'université : rendez-vous dans votre répertoire `test1` d'origine. Exécutez la commande Git permettant de mettre à jour votre dépôt. Vérifiez que vous avez récupéré les modifications.

Q 14. Les lignes :

```
   if (argc == 0) {
      return 0;
   } else
```

sont en fait inutiles. Supprimez-les et mettez à jour votre dépôt. La vue 'Graph' doit être :

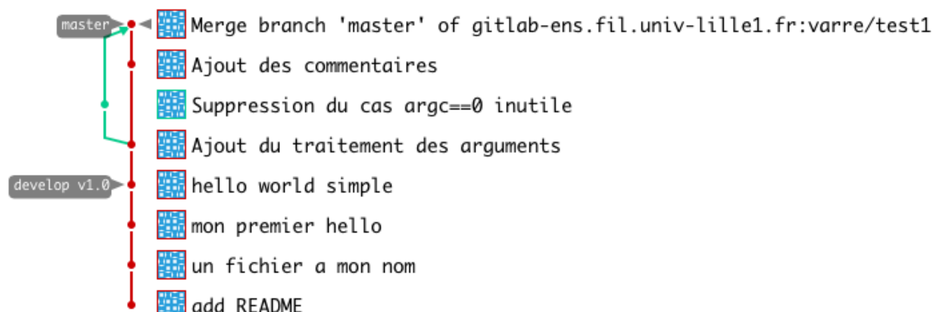


Q 15. Retournez à la maison mais **ne mettez pas à jour** le dépôt. Sous les remarques insistantes de votre enseignant, vous allez ajouter quelques commentaires à votre fichier `hello.c`, en début de fichier :

```
/**
 * Programme qui affiche Hello World! ou bien le premier
 * argument donné sur la ligne de commande.
 */
#include <stdio.h>

int main (int argc, char **argv) {
   if (argc == 0) {
      return 0;
   } else if (argc == 1) {
      printf("Hello world!\n");
   } else {
      printf("%s\n",argv[1]);
   }
   return 1;
}
```

Q 16. Enregistrez les modifications dans l'index (exécutez un `commit`). Puis poussez les modifications. Pourquoi cela ne fonctionne-t-il pas ? Exécutez les commandes Git permettant de réaliser le push. Observez attentivement la vue 'Graph qui devrait ressembler à cela :



Q 17. Enfin, retournez à la version 1.0 (sans effacer les modifications de la version courante, bien sûr). Vérifiez que `hello.c` a le bon contenu.

Q 18. Vous pouvez supprimer ce projet test en allant dans les paramètres du projet *Settings*, puis *Advanced Settings*, puis Remove Project.

3 Jouer à plusieurs

Le travail dans cette partie est à réaliser en groupe de 4 étudiants, chacun devant sa machine. On nommera les étudiants *etu1*, *etu2*, *etu3*, *etu4*.

Le but du projet est de créer un exécutable qui demande la saisie d'un nombre, en calcule le carré, et l'affiche. Chacune des trois fonctionnalités (lecture, calcul, impression) se fera dans un module séparé. Le programme principal sera développé par le quatrième étudiant. Dans un second temps, on ne travaillera plus par développeur mais par fonctionnalité.

3.1 Chacun sa branche

Situation : chacun fait un développement du projet et travaille seul sur sa fonctionnalité

3.1.1 Dépôt initial

Q 19. *etu1* crée un dépôt nommé 'test_a_4' et l'initialise en suivant la procédure comme avant.

Q 20. *etu1* invite *etu2*, *etu3* et *etu4* à rejoindre le projet.

Q 21. *etu2*, *etu3* et *etu4* rappatrient la version courante du projet.

etu1, *etu2*, *etu3* et *etu4* se sont mis d'accord pour que leurs développements soient réalisés dans la branche '`develop`', alors que la branche '`master`' sera réservée pour les versions fonctionnelles du projet. De plus ils se sont entendus sur le fait qu'une sous-branche de `develop` sera crée pour chaque développeur.

Q 22. *etu2* crée la branche '`develop`'.

Q 23. *etu1*, *etu3*, *etu4* passent sur la branche '`develop`'.

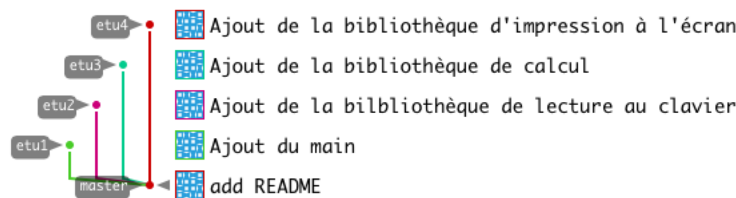
Q 24. *etu1*, *etu2* *etu3* *etu4* créent une branche propre.

3.1.2 Créer un premier projet fonctionnel à plusieurs

Q 25. En même temps :

- *etu1* crée un fichier pour le programme principal
- *etu2* crée les fichiers pour la lecture au clavier
- *etu3* crée les fichiers pour le calcul
- *etu4* crée les fichiers pour l'impression à l'écran

Poussez vos modifications dans vos branches respectives. La vue 'Graph devrait ressembler à cela :



Q 26. Déplacez-vous dans la branche `develop`. Faites une fusion de vos branches *etu1*, *etu2*, *etu3*, *etu4* dans la branche `develop`. Pourquoi un message de commit n'est pas réclamé pour toutes les branches ?

Q 27. Validez le bon fonctionnement du programme. Lorsque c'est correct, fusionnez dans la branche `master` et l'étiqueter `v1.0`.

3.2 Gérer les conflits, contribuer sur les mêmes fichiers

Situation : on crée de nouvelles fonctionnalités indépendamment sur deux branches, on travaille à plusieurs sur une même fonctionnalité

3.2.1 Dépôt initial

On part de la version 1.0 développée ci-dessus.

3.2.2 Développement d'une nouvelle impression

etu2 et *etu4* prennent en charge le développement d'une nouvelle fonctionnalité au niveau de l'impression : on aura désormais une option d'impression verbeuse ou non du résultat, il faudra donc modifier le main et la bibliothèque d'impression.

Q 28. Pour ce faire ils créent une branche nommée `print_verbose` et y font leur développement. *etu2* et *etu4* se répartissent le travail mais dans cette même branche.

Q 29. Les modifications sont enregistrées (commit) et poussées (push) sur cette branche. *etu2* et *etu4* doivent avoir la même version fonctionnelle.

3.2.3 Développement d'une nouvelle fonctionnalité de calcul

Dans le même temps *etu1* et *etu3* prennent en charge le développement d'une nouvelle fonctionnalité au niveau du calcul : on pourra désormais calculer la racine carrée du nombre, il faudra donc modifier le main et la bibliothèque de calcul.

Q 30. Il développeront dans la branche `square_root`. *etu1* et *etu3* se répartissent le travail mais dans cette même branche.

Q 31. Les modifications sont enregistrées (commit) et poussées (push) sur cette branche. *etu1* et *etu3* doivent avoir la même version fonctionnelle.

3.2.4 Mise en place d'une nouvelle version

Q 32. *etu1* et *etu3* sont les premiers à avoir fini leur développement, ils fusionnent donc dans la branche master et étiquettent cette version 2.0.

3.2.5 Aller sur la branche master sans abandonner ni diffuser les développements en cours

Q 33. *etu2* et *etu4* se rendent compte qu'il serait bon de compléter le fichier main avec le nom des auteurs ! *etu2* se décide à le faire immédiatement.

Q 34. Informé des modifications faites par *etu1* et *etu3*, *etu2* est pressé tester le travail de ses collègues, **avant même** de faire un commit du main. *etu2* veut donc aller voir dans la branche `master` et réalise un `git checkout master`. Mais Git ne l'entend pas ainsi. Que se passe-t-il ? Comment résoudre ?

Q 35. Une fois trouvé et après avoir testé, *etu2* revient dans la branche `print_verbose` et enregistre et diffuse les modifications du main sur le dépôt.

3.2.6 Récupérez la version courante sans abandonner, ni diffuser les développements en cours

etu2 et *etu4* apprennent que *etu1* et *etu3* ont terminé leur développement et ils aimeraient récupérer cette version. Comment peuvent-ils faire ?

Q 36. Faites en sorte de récupérer les modifications de la version `v2.0`. Puis poussez les modifications sur `master`.

Q 37. Une fois cela fait créer une version `v3.0` sur master.

Q 38. Assurez-vous que lorsqu'on change de version, on retouve bien le code correspondant.

3.3 Utilisation de Gitlab pour contribuer à plusieurs

Il est généralement bon d'avoir une branche par fonctionnalité

Q 39. *etu1* et *etu2* se mettent en tête d'ajouter une nouvelle fonctionnalité au programme : calculer le cube d'un nombre. Ils développent la fonctionnalité et la pousse sur une branche nommée `cube-nombre`.

Q 40. Ils créent ensuite une *merge request* sur Gitlab afin de demander la fusion de cette branche dans la branche principale.

Q 41. *etu3* et *etu4* sont chargés du traitement de cette *merge request*. Ils relisent donc le code, font éventuellement des commentaires dessus, directement dans Gitlab. Il peut y avoir plusieurs itérations de cette manière-là. La *merge request* Gitlab permettra (pas ici) de voir si le nombre de tests a évolué ou de voir si des tests ont échoué.

4 Projet PJE

Q 42. Réalisez un *fork* du dépôt `salson/PJEA-backbone` présent sur le Gitlab du FIL (l'un des membres du groupe le *fork*, les autres sont invités).

Q 43. Suivez précisément les étapes détaillées dans le fichier `README.md`