TD outil de développement logiciel : git et gitlab

Rendre un rapport au format pdf + code sous forme d'archive à la fin du TP sous Celene ; un rapport par étudiant

Consignes générales

- Tout ce qui est nécessaire au TP est installé dans la VM. Il y a un compte "sti" mot de passe "sti"
- Vous rendez vos rapports pour le contrôle continu sur Celene, avant minuit le jour du TP, sous forme d'archive contenant un rapport en pdf ainsi que tout code utile
- **Par défaut le clavier est en anglais,** `setxkbmap fr` **résoudra ce problème.**
- Aujourd'hui vous aurez besoin d'avoir votre clé installée sur gitlab et sur votre machine. Si vous ne l'avez pas déjà fait, un petit tutoriel est disponible ici `https://docs.gitlab.com/ce/ssh/`

Prologue au TP

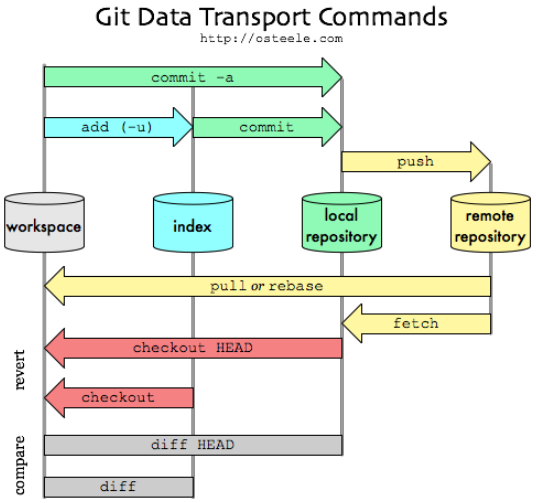
Git

Git fait partie de la famille d'outils dédiée à la gestion de version, comme ces prédécesseurs svn, cvs, et consort.

La particularité de git est de garder 2 types de dépôts (*repository*) séparés que chaque développeur synchronise de manière explicite. Dans chaque dépôt sont stockés les différentes versions du projet. A chaque commit, git stocke localement une nouvelle version. La version courante est enregistrée localement dans un pointeur appelé *HEAD* (pensez à une tête de lecture).

Les versions sont organisées en branches (ce qui fait que les versions sont organisées en préordre, pas en arbre, vous qui aimez les maths). Par défaut la banche courante est MASTER. C'est celle où vous commitez. Mais vous pouvez créer une nouvelle branche, et ensuite commiter dedans. Si vous voulez fusionner les branches il vous faudra faire un *merge*.

Certaines commandes n'ont qu'une portée locale (commit, stash par exemple) et d'autre ne servent qu'à interagir entre les 2 types de dépôt (clone, push, pull, fetch par exemple). La figure suivante vous donne un récapitulatif des principales fonctions de git et leur portée.



Il y a 4 notions importantes sur cette figure :

- l'espace de travail : c'est vos modifications courantes
- L'index, c'est une zone logique de préparation de votre commit. Typiquement quand vous faires un add pour ajouter un fichier nouveau au prochain commit, celui-ci est mis dans l'index

— Le dépôt local, sur lequel vous faites vos checkout et vos commit
— Le dépôt distant (un alias pour le désigner est origin) sur lequel vous poussez ou récupérez vos modifications.

Gitlab et Github

Gitlab est un logiciel de forge (*une "forge est un système de gestion de développement collaboratif de logiciel"* dixit Wikipedia) centré autour de git, mais qui fait plein d'autres choses (gestions de bugs, intégration continue, enfin bref, tout ce que fait une forge). Il y a une version libre de gitlab et une version payante, on peut avoir des solutions hébergées si l'on ne possède pas de ressources pour le faire.

Github est aussi un logiciel de forge centré sur gitlab, mais il est 1) propriétaire 2) hébergé.

Donc, gitlab et github sont les endroits qui vont accueillir vos dépôts distants git. gitlab vous propose en tant qu'administrateur un certain nombre de fonctionnalités bien pratiques, comme par exemple de définir des groupes, qui seront des espaces dans lesquels vous pouvez des droits étendus à certains utilisateurs.

1 Quelques échauffements avec git

vous aimez les nombres premiers, vous êtes décidé à partager votre passion avec vos amis passionnés d'arithmétique, et décidez d'unir vos forces pour améliorer encore votre petite bibliothèque et votre fantastique site Web.

Dans un premier temps, vous décidez d'héberger votre projet sur le serveur gitlab de l'école.

— Allez sur le gitlab de l'école `https://gitlab.insa-cvl/fr` et créez votre projet dans le groupe *outilsDeDev* (click sur le petit icône de type *hamburger menu*, puis groupe, puis new project). Le plus dur sera de trouver un nom original pour éviter les confits de noms avec vos camarades, pourquoi pas votre login ?
— A partir des informations de la page d'accueil de votre nouveau projet, importez l'ensemble de vos créations jusqu'à présent sur gitlab.
— Maintenant vérifiez que ça marche. Créez localement un deuxième répertoire pour y cloner votre projet (ce répertoire sera appelé "REP2" pour la suite de l'énoncé). Clonez votre projet dans ce répertoire
— Modifiez quelque chose dans votre répertoire original
— Modifez quelque chose d'autre dans REP2.
— Commitez votre modification dans votre répertoire original
— Commitez votre modification dans REP2
— Pushez la modif faite dans le répertoire original
— Tentre de pusher la modif faite dans REP2.

Question Ca ne marche pas. Expliquez brièvement pourquoi. Résolvez votre problème par un pull. Donnez la liste de commande utilisée pour pusher votee modif de REP2.

— Modifiez quelque chose dans votre répertoire original et faites un commit et un push. Modifiez REP2 sans faire de commit.

Question Faites un pull : que dit git ? Faites un remisage de vos modifications (stash), un pull et réappliquez vos changements. Poussez le tout. Donnez la liste de commande pour parvenir à ce push.

— Modifiez quelque chose dans votre répertoire original et faites un commit et un push. Faites encore une modif dans votre répertoire original sans faire de commit.

Question Faites un fetch : que constatez vous quand à la différence des commandes git diff et git diff master origin/master ?

Question Faites un merge et poussez le tout, donnez la liste de commandes pour faire tout cela.

2 A propos des branches

Quelques explications sur les branches

Vous trouverez une bonne petite explication des branches à cette adresse `https://git-scm.com/book/fr/v1/Les-branches-avec-Git-Ce-qu-est-une-branche` . L'idée est de créer un nouveau poin-

teur vers un commit particulier, et potentiellement diverger et créer une nouvelle liste de commit différente du mastern, jusqu'à un potentiel merge des différentes branches.

Travail sur les branches

— Créez une branche "petitebranche" dans votre projet. Pour cela, il vous faut utiliser la commande `git branch petitebranche`
— Commitez un petit quelque chose puis visualisez les branches dans git gui

Question Que concluez vous sur branch ? Faites maintenant un commit dans branch. Pour basculer dans branch, faites un checkout. Donnez la liste des commandes pour arriver à rajouter un commit dans "petitebranche".

Question Il va maintenant vous falloir fusionner les 2 branches de développement (master et petite branche"). Mour cela vous devez travailler à partir du master et faire un merge de votre branche "petitebranche". Donnez la suite d'instructions git pour y arriver.

3 Travaillons avec un ou des amis

— Trouvez vous un ami, par exemple, votre voisin de gauche. Invitez le à participer à votre projet pour qu'il puisse faire des modifications sur votre projet mais sans être trop impactant.

Question Quel rôle avez-vous attribué à votre ami ? Justifiez votre choix.

Question Faites de manière concurrente un commit chacun sur votre projet votre ami et vous. Tentez d'intégrer les modifications de chacun. Décrivez la solution choisie.

Question Si vous aviez plusieurs modifications à faire sur votre projet avec plusieurs développeurs, dans quelles conditions à votre avis créeriez vous des branches ?

Question Vous êtes toujours l'ami de quelqu'un. En tant qu'ami, trouvez un moyen sur gitlab de demander à votre ami chef de projet d'ajouter plus de math à son projet (car vous aimez les maths). Regardez du côté des issues pour cela. Donnez une copie d'écran montrant votre issue.

Question En tant que chef de projet, vous trouvez l'idée de mettre plus d'arithmétique dans le projet très bonne. Trouvez le moyen d'assigner cette tâche à l'ami qui vous la suggère. Donnez une copie d'écran montrant l'assignation de l'issue.