Objectif : Se familiariser avec le logiciel de gestion de version Git.

1 Environnement et quelques commandes élémentaires

Pour prendre en main l'outil git, nous allons privilégier la ligne de commande afin de bien comprendre le fonctionnement de l'outil, en s'affranchissant ainsi de la variété des outils graphiques (simples surcouches aux commandes).

Dans la pratique, il est courant d'utiliser une version graphique souvent intégrée au logiciel de développement (e.g. eclipse, visual studio, . . .). Dans notre cas, on utilisera le logiiel git bash : il s'agit d'un interpréteur de commandes (supportant donc les commandes posix élémentaires) et pemettant d'invoquer les commandes git.

Afin de créer et de modifier des fichiers (hors utilisation de git), on pourra utiliser les commandes posix :

— touch file.txt pour créer un fichier,

— echo "ligne" » file.txt pour écrire une nouvelle ligne en fin de fichier,

— mkdir rep pour créer un répertoire,

— cd rep pour changer de répertoire,

— pwd pour afficher le chemin courant,

— ls -al pour consulter le contenu d'un répertoire,

— cat pour afficher le contenu d'un fichier.

La documentation officielle est disponible en local (git help <nom-option>) et en ligne (http ://git-scm.com/docs).

2 Mise en pratique

L'objectif est de faire évoluer le contenu d'un fichier, en gardant une trace des différents états intermédiaires, ainsi qu'en étiquettant des versions importantes du fichier.

Le scénario à réaliser est le suivant :

Exercice 1

1. Créer un répertoire project.

2. Initialiser, dans le répertoire projet, un dépôt git avec la commande git init (exécutée depuis le répertoire project).

3. Configurer ce dépôt avec votre nom et votre email, ce qui permettra d'identifier l'auteur

des modifications :

nico@pc:~/projet$ git config --global user.name "nom"

nico@pc:~/projet$ git config --global user.email "nom@email.com"

4. Créer un fichier file.txt avec une ligne ligne01 et indexer ce fichier. Dans quels états sont les fichiers du dépôt.

5. Prendre un instantané avec un commit. On pourra donner comme commentaire : version1.

6. Ajouter la seconde ligne suivante au fichier : ligne02 ; indexer, puis commiter cette nouvelle version.

7. Quelle commande permet d'afficher l'historique des modifications, i.e. l'ensemble des commits ?

8. On considère cette version correspond à un livrable du code que l'on va nommer v0.1. Nommer ce commit avec le tag v0.1.

9. Ajouter une nouvelle ligne ligne03 au fichier. Quelle commande permet d'afficher les différences entre le répertoire de travail et le dernier commit ?

10. Interpréter le résultat précédent.

11. Historiser cette modification dans un nouveau commit.

12. Revenir à la version marquée avec le tag v0.1. Quel est le contenu du fichier ?

13. Taper la commande git log. Comment revenir au commit contenant les 3 lignes ?

14. Ajouter et commiter une nouvelle ligne ligne04. Marquer cette version comme v0.2. Quelle commande permet d'afficher les différentes versions tagguées.

15. Consulter graphiquement le dépôt : « git gui »

16. (Optionnnel) Consulter graphiquement le dépôt avec eclipse : créer un projet (new project) à l'endroit de votre répertoire ; consulter l'historique de votre fichier (bouton droit, Team, Show in History).

3 Développement avec des branches

A partir d'une version donnée du code, on souhaite pouvoir poursuivre le développement principal, tout s'autorisant de partir sur des développements annexes sur une période plus courte. Il peut typiquement s'agir d'une correction d'un bug majeur qui peut nécessiter plusieurs étapes et affecter plusieurs fichiers.

On considère le fichier nommé lignes.txt suivant :

On constate qu'il y a une erreur (un bug) sur les deux premières lignes : il manque la lettre l devant igne01. Pour corriger cette erreur, nous allons travailler sur une branche secondaire afin de ne pas bloquer les développements sur la branche principale.

Exercice 2

1. Créer le dépôt, configurer-le avec votre nom et votre email, intégrer le fichier lignes.txt avec les 3 lignes indiquées.

2. Créer une branche correction, et afficher les branches disponibles sur le dépôt ainsi que votre état.

3. Sur quelle branche êtes-vous ?

4. Placez-vous sur la branche correction, et reafficher les branches disponibles. Quelles différences observez-vous ?

5. Dans quel état sont les fichiers ?

6. Vous êtes toujours sur la branche correction : éditer le fichier afin d'ajouter le l manquant sur la première ligne seulement. Commiter cette modification avec le message

lignes.txt : ligne01 corrigée.

7. On suppose que le développement se poursuit sur la branche principale (master) : placer vous sur cette branche. Consulter le contenu du fichier. La correction apportée ne doit pas apparaître car celle-ci a été faite sur la branche correction.

8. Ajouter une nouvelle ligne au fichier (ligne04) et commiter cet ajout.

9. Revenez sur la branche correction, corriger la ligne02 du fichier texte et commiter la modification. Quel est le contenu du fichier texte.

10. Nous allons intégrer les corrections apportées sur la branche bug à la branche principale :

— positionnez-vous sur la branche master ;

— afficher le contenu du fichier,

— intègrer les modifications de la branche correction avec la commande :

nico@pc:~/projet$ git merge correction -m "fusion".

11. Afficher l'état du fichier après modification : les deux premières sont corrigées, et la ligne04

doit apparaître (ajoutée en parallèle du correctif sur la branche correction).

12. Afficher graphiquement toutes les évolutions avec une interface graphique.

4 Développement collaboratif

Afin de se familiariser avec le développement collaboratif, nous allons considérer un projet avec trois dépôts (dans trois répertoires distincts) : deux dépôts locaux (repo1 et repo2 associés aux utilisateurs nom1 et nom2) et un dépôt distant remote. Cette situation est illustrée par la figure 1

FIGURE 1 – Développement collaboratifs avec 3 dépôts.

Exercice 3

1.

(a) Créer un répertoire repo1, aller dans ce répertoire (cd repo1), puis initialisation un dépôt git, et configurez-le avec le nom nom1 et le email nom1@email.com (avec l'option local).

(b) Créer un fichier file.txt avec une seule ligne ligne01. Ajouter puis historiser ce fichier dans un commit commenté avec "first commit".

(c) Afficher les logs (git log)

2.

(a) Remonter dans l'arborescence.

(b) Créer un réperoire remote puis aller dans ce répertoire. Nous allons initialiser le dépôt remote par clonage du repo1 avec la commande suivante. Avant de taper cette commande, répondre à ces deux questions :

— que désigne le dernier point dans la commande précédente ?

— quel est l'objectif de l'option –bare de commande git clone ?

nico@pc:~remote$ git clone --bare chemin_vers_repo1 .

(c) Afficher ensuite les log de ce dépôt.

3.

(a) Remonter dans l'arborescence.

(b) Créer un répertoire repo2 puis aller dans ce répertoire.

(c) Initialiser le dépôt repo2 par clonage du dépôt remote.

(d) Configurer ensuite ce dépôt avec le nom nom2 et le email nom2@email.com (avec l'option local).

(e) Afficher ensuite le log.

4.

(a) Depuis le répertoire repo2 : afficher le contenu du fichier.

(b) Ajouter la ligne ensuite une ligne ligne02 en fin de fichier : ajouter puis commiter cette modification.

(c) Afficher le log. Quelle sont les version présentes ? qui les a réalisées ?

5. Mise à jour du dépôt distant (remote) depuis le dépôt courant repo2 :

(a) rester dans le dépôt repo2,

(b) pousser (push sans option) la modification précédente vers le dépôt remote.

6. Vérification que la mise à jour a bien été intégrée au dépôt remote :

(a) placez-vous dans le répertoire remote,

(b) consulter le log.

(c) pour quelle raison aucun fichier ne se trouve dans le répertoire de travail remote.

7. On souhaite récupérer en local dans repo1 les modifications apportées au remote depuis repo2.

(a) Placez-vous dans le répertoire repo1.

(b) Pour commencer, il faut ajouter remote comme dépôt distant à repo1. En effet, contrairement au dépôt repo2, repo1 n'a pas été initialisé par clonage (c'est remote qui a cloné repo1 et non l'inverse). Pour cela, utilisez la commande git remote add distant chemin-vers-remote

(c) Vérifier qu'il est bien enregistré comme dépôt distant avec une commande git remote.

8. Toujours depuis repo1, vous pouvez récupérer en local des dernières modifications disponibles sur le remote avec une commande comme git fetch distant. Cela crée une branche locale :

(a) affichez la liste des branches (git branch -a),

(b) puis positionnez-vous sur la branche nouvellement créée (checkout)

(c) et consulter le contenu du fichier.

9. Revenez sur la branche principale avec checkout master. Consulter l'état du fichier.

Intégrez ensuite les modifications importées sur la branche (merge distant/master).

10. Consulter à nouveau l'état du fichier.

11. Ajouter une nouvelle ligne puis pousser sur le dépôt distant (push distant master:master).

12. La mise à jour de repo1 peut paraître fastidieuse car repo1 n'est pas le clone du remote

(c'est l'inverse).

Si repo1 était le clone du remote, un certain nombre de choses auraient été configurées automatiquement, simplifiant considérablement les commandes. Par exemple, il a fallu explicitement ajouter un remote à repo1 : en cas de clone du dépôt distant (e.g. cas de repo2), le dépôt est automatiquement configuré (nom : origin), et la commande fetch utilise ce dépôt distant par défaut (inutile de préciser fetch distant). Pour illustrer cela, retournons dans le repo2 qui est un clone du remote, avec certains éléments automatiquement configurés :

(a) Placez-vous dans repo2

(b) Récupérer les modifications apportées par repo1 sur remote : git pull (équivalent

de fetch enchaîné avec merge)

(c) Vérifier que le fichier est mis à jour.

(d) Modifier ce fichier (ajout ligne04 en fin de fichier) et commiter la modification

(e) Pousser sur le remote : git push.

13. De retour dans le repo1, vous pouvez récupérer ces modifications : git pull distant master