EXERCICES

GIT AVANCE, GITHUB, GITLAB

Table des matières

OBJECTIFS

Ce cahier d'exercices vous aidera à maitriser des commandes avancées de GIT ainsi que le travail en équipe sur GitLab ou GitHub.

GIT BASH (WINDOWS) OU TERMINAL (MAC ET LINUX)

Les exercices s'effectuent dans une fenêtre git bash (sur windows) ou un terminal (mac ou linux) positionnés sur le bon répertoire (local, shared ou cloned).

Pour positionner la fenêtre git bash sur le bon répertoire on peut faire un clic droit (dans le navigateur de fichier sous windows) sur le dossier dans lequel se placer (local par exemple), et lancer la commande 'Open Git Bash Here'.

REMARQUE

Certaines commandes lancées retournent un résultat sur plusieurs pages.

GIT utilise un 'pager' par défaut qui attend que l'utilisateur saisisse un 'espace' pour passer à la page suivante.

Pour sortir du pager sans consulter toutes les pages, **appuyer sur la touche 'q'** (ie : quit).

EXEMPLE DE CODE A GERER

Afin de rendre ces exercices plus réalistes, on créera du code 'algorithmique' qui affiche une fenêtre avec un bouton par exemple dedans. Cet algorithme pourra aussi être écrit en java, en Html, (ou tout autre language) si vous le connaissez afin de pouvoir l'exécuter.

Si vous ne connaissez pas de langage approprié, vous pouvez conserver cet exemple algorithmique (fichiers .algo).

Fichier Helloworld.algo :

```
Algo HelloWorld {
    fonction principale {
            // Création de la fenetre
            Créer une fenetre de titre 'Hello world'
            Mettre la dimension à 200, 200
            Configurer la fenetre sur 'exitOnClose'
            // Création du label de texte
            Créer un label avec du texte centré et contenant le texte 'Hello world'
            Créer la couleur jaune
            Mettre la couleur jaune sur le label
            // Finalisation
            Ajouter le lable sur la fenetre
            Afficher la fenêtre
}
}
```

Et pour le fichier des constantes par exemple

```
Constantes {
            constante HELLO_WORLD_TITLE = « Title Hello world »
            constante HELLO_WORLD_TEXT  = « Hello world text »
 }
```

GIT 200 : MISE EN PLACE D'UN HISTORIQUE GIT

Dans un répertoire de travail, créer un dépôt git local, et y gérer un ou plusieurs fichiers de code (algo comme ci dessus ou code réel exécutable).

On effectuera les modifications suivantes :

Dans la branche master :

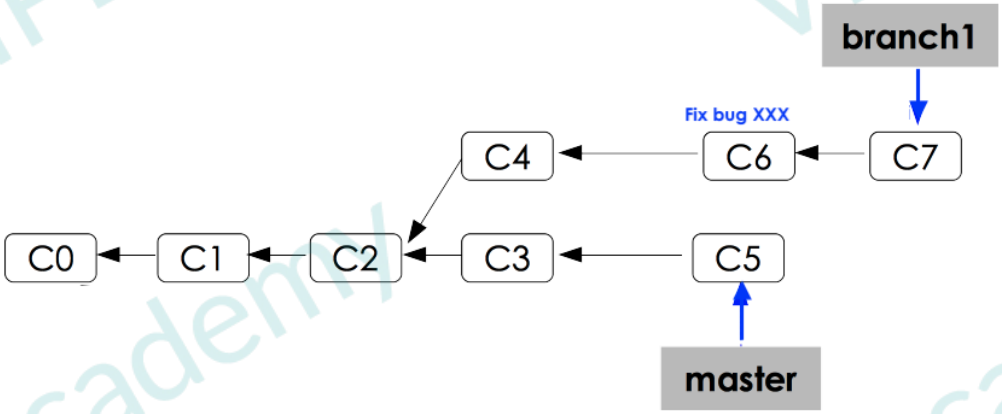
- ✓ C1 : ajout du/des premier(s) fichiers(s) de code
- ✓ C2 : extraction des constantes (Cf ci dessous)
- ✓ C3 : mise à jour taille fenetre et ajustement du texte des constantes
- ✓ C5 : fond d'écran en jaune

Dans la branche 'branch1' (créée à partir de C2)

- ✓ C4 : couleur du texte principal en bleu
- ✓ C6 : couleur de fond en vert (fix_XXX)
- ✓ C7 : fenetre large et alignement à gauche.

Ajouter le tag '**fix_XXX**' sur le commit C6

Ces modifications se résument dans ce schéma :



COMPLEMENT

Pour C2, faire par exemple la création d'un fichier de constantes (constantes.algo)

```
Constantes {
  constante HELLO_WORLD = « Hello world »
}
```

puis remplacer ces constantes dans le code de Helloworld et commiter le tout…

```
Utiliser les constantes de Constantes
Algo HelloWorld {
    fonction principale {
        // Création de la fenetre
        Créer une fenetre de titre HELLO_WORLD
        Mettre la dimension à 200, 200
        Configurer la fenetre sur 'exitOnClose'
        // Création du label de texte
        Créer un label avec du texte centré et contenant le texte HELLO_WORLD
        Créer la couleur jaune
        Mettre la couleur jaune sur le label
        // Finalisation
        Ajouter le lable sur la fenetre
        Afficher la fenêtre
}
}
```

Pour les autres commits, changer dans le code de l'algo les noms de couleur, ou ajouter des lignes pour dire qu'on met la couleur de fond à vert par exemple ….

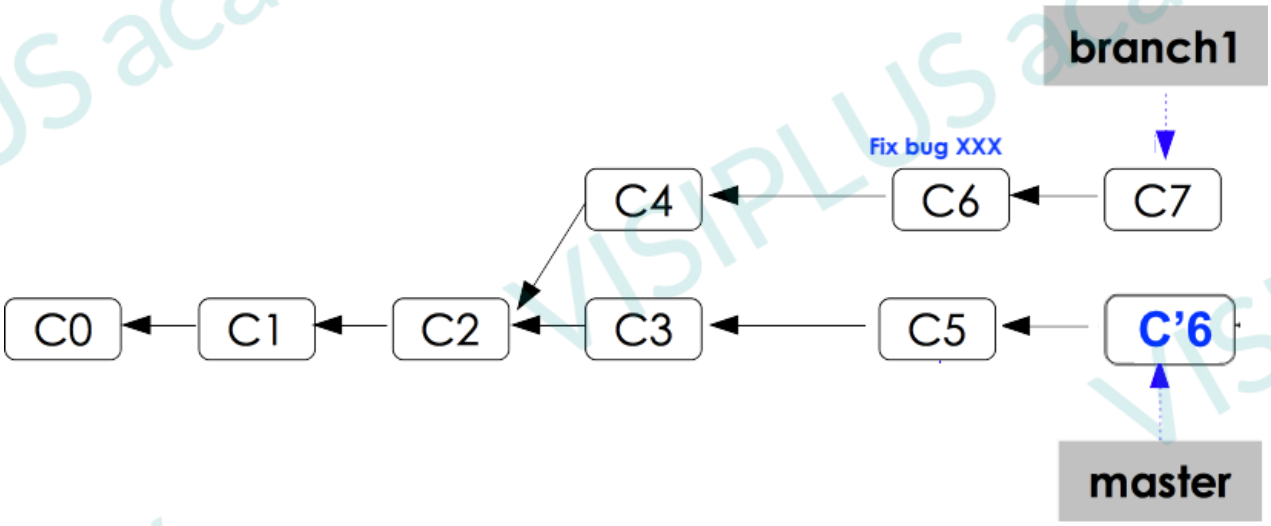
GIT_210 : EFFECTUER UN CHERRY PICK

Le commit taggé Fix_XXX contient une correction qui nous intéresse et qui devrait être intégrée dans master. Effectivement, la branche 1 ne va pas aboutir et elle risque de ne pas être fusionnée. Par contre ce commit qu'elle contient doit être récupéré.

Effectuer le cherry-pick du commit 'Fix_xxx' pour créer un commit « C'6 » dans la branche master et mettant donc la couleur de fond en vert.

S'il y a des conflits, réduisez les et finalisez le cherry-pick.

On doit obtenir au final l'historique suivant :



GIT_220 : COUPER ET RESUSCITER UNE BRANCHE.

La branche 1 finalement n'est plus utile. Il faut la couper. Exécuter la commande et consulter l'historique.

Mais on reçoit un appel, une fois que la branche est coupée, qui nous dit que finalement il fallait la garder !!!

Faire le nécessaire pour retrouver la branche dans l'historique.

En profiter, une fois la branche retrouvée, d'ajouter une note sur le commit « C7 » indiquant simplement que la branche a été retrouvée.

Afficher cette note :

GIT_230 : CREATION D'UN HOOK

Installation du hook

A chaque commit, on veut stocker un fichier complémentaire automatiquement contenant le texte : « commit vérifié le <date et heure du jour>», si on a choisi 'y' dans le dialogue qui le demande.

En vous inspirant du code shell ci dessous, qui fait une sauvegarde d'une base de donnée si on a choisit 'y' comme réponse à la question, adapter ce code pour créer ce fichier dans 'suivi/commitInfo.txt' (au lieu de sqldump/$DBNAME.sql.zip).

Ajouter le hook à l'endroit adéquat et contrôler son exécution correcte quand on répond 'y' à la question ou quand on répond ' n ' ou enter.

```
#!/bin/bash

# Change the following values to suit your local setup.
# Add this file on your .git/hooks/ and name it as 'pre-commit'
#

# The name of a database user with read access to the database.
DBUSER=xxxxxx

# The password associated with the above user. Leave commented if none.
DBPASS=xxxxx

# The database associated with this repository.
DBNAME=xxxxx

DBHOST=localhost

# The path relative to the repository root in which to store the sql dump.
DBPATH=sqldump

read -p "Sauvegarde de la base dans ce commit (y/[n]) ?" yn < /dev/tty
yn=${yn:n}
case $yn in
        [Yy]* )
            echo mysqldump -h $DBHOST -u $DBUSER -p$DBPASS $DBNAME > $DBPATH/$DBNAME.sql
            mysqldump -h $DBHOST -u $DBUSER -p$DBPASS $DBNAME > $DBPATH/$DBNAME.sql
            zip $DBPATH/$DBNAME.sql.zip $DBPATH/$DBNAME.sql
            git add $DBPATH/$DBNAME.sql.zip
            exit 0;;
        [Nn]* )
            exit 0;;
esac
exit 0
```

Partage du hook

Mettre ensuite en place un mécanisme de transfert du hook (avec un fichier readme.txt qui explique comment l'utiliser) pour les autres développeurs. Faire cette modificaiton dans la branche master et sous la forme d'un commit dont le commentaire sera : « C8 : mise en place du mécanisme de partage du hook « .

GIT_240 : PUBLICATION DU PROJET DU GITHUB

Accéder à votre compte GitHub (le créer si besoin), et créer un nouveau dépôt. Nous appellerons ce dépôt 'ghDistant' pour le désigner dans la suite de cet exercice.

***Attention** : ne pas ajouter de fichiers (README, licences ou autre) dans un premier commit. Le dépôt doit être vide.*

Publier le dépôt local créé dans les exercices précédents sur ghDistant.

GIT_250: TRAVAIL SIMULTANE ET SYNCHRONISATION GITHUB

Directement dans l'interface Github, éditer un des fichiers de la branche master et faire le commit (faire commencer le commentaire par C20)

En local, modifier différemment le même fichier de la branche master (faire commencer le commentaire par C10)

Faire le push du dépôt local vers github. Gérer le problème afin d'obtenir au final sur le dépôt GitHub (et sur le dépôt local) la suite de commit unique suivante dans la branche master :

C8 ← C20 ← C10

GIT_260: FORK ET PULL REQUEST SANS CONFLIT

Pour cet exercice, vous devez créer un second compte github. Nommez le comme votre compte d'origine suivi d'un '2' par exemple.

Connectez vous sur le compte2 et naviguez sur le compte d'origine pour forker le projet (aller sur https://www.github.com/compte/projet en remplaçant compte par votre login et projet par le nom du depot créé pour l'exercice et cliquer dans fork).

Créer une issue dans projet (sur compte1), pour dire qu'il faut passer finalement la couleur de fond en cyan.

Dans compte2, directement via les éditeurs de github, modifiez le fichier du projet forké pour mettre à jour la couleur cyan de fond comme demandé dans l'issue créée. Faites un commit en mettant dans le commentaire « Close #numeroIssue » pour dire qu'on a fermé l'issue correspondante.

Créer une pull request et faire le merge dans compte1 après avoir échangé des commentaires dans la pull request pour constater ce qui est possible et les mails envoyés par github.

GIT_270 : PULL REQUEST AVEC CONFLIT.

Ouvrir une issue pour demander de changer la taille de la fenêtre d'origine et une autre issue pour ajouter un label qui écrit la date du jour à coté du label helloworld et qui change aussi la taille de la fenetre d'origine.

Dans compte 2, corriger l'issue d'ajout de label de date et changement de taille, et proposer une pull request. Ne pas encore l'intégrer.

Dans compte, corriger l'issue de taille seulement (mettre une taille de fenêtre différente de celle corrigée pour l'issue 2) et faire le commit.

Constater l'état de la pull request dans compte1.

Cloner le dépôt forké de compte2 localement. Ajouter l'URL d'upstream sur compte et faire un fetch. Effectuer la mise à jour en faisant un merge et en résolvant le conflit, puis pusher sur origin. Constater que la pull request est maintenant synchronisable et l'intégrer dans compte.

GIT_280 : PUBLICATION DU PROJET DU GITLAB

Accéder à votre compte GitLab (le créer si besoin), et créer un nouveau dépôt. Nous appellerons ce dépôt 'gitlabDistant' pour le désigner dans la suite de cet exercice.

Publier le dépôt local créé dans les exercices précédents sur gitlabDistant (ajouter simplement la nouvelle URL en lui donnant l'ID de gitlab). Votre dépôt local a maintenant 3 remote URL : origin (le fork), upstream (github), et gitlab (gitlab).

On pourra ainsi travailler sur les différents dépôts pour les synchroniser ensemble.

GIT_290: CREATION D'UNE MERGE REQUEST SUR GITLAB

Pour faire une merge request, il faut créer un second compte gitlab et comme pour l'exercice de GitHub, forker le projet gitlab d'origine dans le second compte.

Pour cet exercice, on utilisera l'interface de gitlab dans le compte2 pour modifier un fichier et proposer une merge request que l'on intégrera.

On pourra ensuite faire un fetch gitlab pour constater l'évolution des modifications dans le compte local et imaginer la suite de la synchronisation.