TP – Git avançé

Industrialisez votre chaîne de production logicielle C/C++

Amabis

v1.0.0

Echoes Technologies
18-22 rue d'Arras
92000 Nanterre
+33 1 75 61 41 60

Sommaire

A propos

Ce document est le support des Travaux Pratiques de la formation Gestion de versions (avançé) de l'offre Echoes Software Factory de la société Echoes Technologies.

Prérequis

- Un poste de travail sous Windows ou Linux
- Git installé
- Git extensions installé

Création d'un compte sur Github

Aller sur github et créer un compte : https://github.com/

TP1 : Commandes de base de git

Fork d'un dépôt Github

Aller sur l'adresse suivante et faire un fork de ce dépôt (bouton Fork en haut à droite) : https://github.com/snmsts/roswell

Copier l'url du nouveau dépôt créé (textbox 'HTTPS clone URL' sur la droite) et faire un clone de celui-ci en local :

```
git clone monDepot
```

1. Placez-vous sur le dossier racine de ce dépôt puis afficher les branches locales présentes:

```
git branch
```

2. Affichez maintenant l'ensemble des branches (locales + distantes) du dépôt, que remarquez-vous ?

```
git branch -a
```

3. Créez une branche locale suivant la branche distante nommée 'go'

```
git branch go origin/go
```

4. Affichez de nouveau la liste des branches, êtes-vous sur la branche créée ?
5. Déplacez-vous sur la nouvelle branche :

```
git checkout go
```

6. Vérifiez que vous êtes bien sur la branche souhaitée.
7. Revenez sur la branche master

```
git checkout master
```

8. Créez une nouvelle branche locale et placez-vous dessus directement avec la commande ci-dessous :

```
git checkout –b maBranche
```

9. Créez un nouveau fichier
10. Vérifiez l'état de votre espace de travail

```
git status
```

11. Quel est l'état de votre fichier ?
12. Ajoutez ce fichier à l'index git

```
git add monFichier
```

13. Quel est le nouvel état du fichier ?
14. Créez un nouveau commit :

```
git commit -m "mon message"
```

15. Quel est maintenant l'état de votre espace de travail ?
16. Modifiez le contenu du fichier existant et créez également un nouveau fichier
17. Essayez de faire un nouveau commit, que se passe-t-il ?


```
git commit -m "blabla"
```

18. Essayez de nouveau avec la commande suivante :

```
git commit -am "blabla"
```

19. Que s'est-il passé selon vous ? Quel est l'état de votre répertoire de travail ?
20. Ignorez le dernier fichier créé, en ajoutant le nom de ce fichier dans le fichier .gitignore
21. Vérifiez que ce fichier est bien ignoré (avec la commande git status)
22. Affichez l'historique des derniers commits :

```
git log
```

23. Utilisez la commande git ci-dessous pour vous créer un alias git affichant les commits comme un graphe :

```
git config --global alias.lg "log --graph --pretty=format:'%Cgreen%h%Creset
-%Creset %s%C(yellow)%d %Cblue(%aN, %cr)%Creset' --abbrev-commit --
date=relative"
```

24. Utilisez cette nouvelle commande pour visualiser les commits sous forme de graphe :

```
git lg
```

25. Affichez l'ensemble des branches (locales et distantes), que remarquez-vous ?

```
git branch –a
```

26. Ajoutez votre branche au dépôt distant

```
git push
```

27. Affichez de nouveau l'ensemble des branches pour vérifier qu'elle a bien été ajoutée. Vous pouvez également aller sur votre page github pour voir l'état du dépôt distant

TP 2 : Utilisations de git flow

Installation de git flow

Installer git flow en suivant les instructions de https://github.com/nvie/gitflow/wiki/Installation

Initialisation de git flow

Initialiser git flow (garder les préférences par défaut).

TP

Faire un fork du dépôt https://github.com/sebastien-carreau/tp_git-avance

Le dépôt contient plusieurs dossiers (nommés 'entrees', 'plats' et 'desserts') contenant des recettes de cuisines (un fichier par recette).

1. Quelles sont les branches présentes sur le dépôt ?
2. Quel sont les nombres de commits et tags pour chaque branche ?
3. Créez une nouvelle branche de feature pour ajouter une recette, puis terminez la branche, expliquez les différentes étapes réalisées par git flow.
4. Une erreur s'est glissée dans la recette de la Salade de pommes de terre camarguaise, dans les ingrédients sont indiqués 40 œufs au lieu de 4. Créez une nouvelle branche hotfix, corrigez l'erreur puis terminez ce hotfix. Expliquez les différentes étapes réalisées par git flow.
5. Créez ensuite une nouvelle release incluant ce hotfix.
6. Sur la branche develop, modifiez une des recettes sans faire de commit. Vous vous rendez compte que vous auriez dû créer une branche de feature pour cela. Il vous faut mettre de côté les modifications réalisées pour créer la nouvelle branche de feature, avec quelle commande git faites-vous cela ?
7. Créez la nouvelle branche de feature, appliquez les modifications mises de côté et faire un commit.
8. Modifiez une des recettes et faire le commit. Finalement cette recette était mieux avant, annuler le dernier commit. Quelle commande avez-vous utilisez ? Si ce commit avait déjà été poussé sur la branche publique, qu'auriez-vous fait ?
9. Imaginons que dans votre feature vous ayez besoin d'un commit particulier venant d'une autre feature. Cette dernière n'étant pas terminée elle n'a pas encore été mergée sur develop. Comment pouvez-vous récupérer les modifications de ce commit sur votre branche ?
10. Pushez sur le dépôt distant.
11. Cloner une seconde fois le dépôt dans un autre répertoire et refaites les étapes 3-4-5 sans utiliser git flow.

TP 3 : Débusquer rapidement l'origine d'un bug avec git bisect

Initialisation

Décompressez le fichier bisect-demo.zip où bon vous semble ; il crée un dossier `bisect-demo` dans lequel vous n'avez plus qu'à ouvrir une ligne de commande (sous Windows, préférez le Git Bash). Ce dépôt contient plus de 1 000 commits répartis sur environ un an et, quelque part là-dedans, un bug s'est glissé.

En effet, si vous exécutez `./demo.sh`, il affiche un `KO` tout penaud. Alors qu'il devrait afficher glorieusement `OK`. Ce souci remonte à assez loin, et nous allons utiliser `git bisect` pour le débusquer.

Git bisect

1. Vérifiez que le script ne fonctionne pas sur le dernier commit
2. Démarrez Git bisect à ce commit (bad)
3. Ici on n'a aucune idée du dernier commit valable, alors on va prendre le commit initial, trouvez l'identifiant de ce commit
4. Déplacez-vous sur celui-ci et vérifiez l'exécution du script
5. Identifiez le comme valable (good), git bisect ainsi initialisé va commencer à faire de nombreux checkout
6. Après chaque checkout, exécuter le script et indiquez si il est bon (good) ou non (bad), jusqu'à trouver le commit qui a introduit le problème
7. Quel est l'identifiant de ce commit, sa date ainsi que son auteur ?
8. Terminez le bisect
9. Dans ce cas précis où l'erreur était simplement une modification d'un fichier, trouvez deux autres méthodes simples et rapides pour trouver le commit défectueux (avec une seule commande)

TP 4 : Dépôt partagé

Par groupes de 2 ou 3 personnes, choisissez le dépôt d'une des personnes et travaillez tous dessus. Faites des modifications/push/pull et créez des conflits pour s'habituer à les résoudre. Familiarisez-vous également avec les autres commandes git (chery-pick, revert, reset, merge, rebase, stash, clean…)

Pour autoriser des personnes à faire des push sur son dépôt github vous pouvez suivre les instructions du lien suivant : https://help.github.com/articles/adding-collaborators-to-a-personal-repository/

TP 5 : Refaire l'histoire avec git rebase -i

La situation

Cette année Bob s'organise, il se fait des fiches de révision pour son examen d'histoire. Du coup, en bon programmeur, il décide de les rédiger en markdown pour pouvoir les enrichir avec des illustrations et autres bricoles.

Il a ainsi rédigé 5 fichiers différents pour 5 périodes différentes :

- la révolution industrielle (la_revolution_industrielle.md) ;
- la Première Guerre mondiale (la_premiere_guerre_mondiale.md) ;
- l'entre-deux-guerres (entre_guerre.md) ;
- la Seconde Guerre mondiale (la_seconde_guerre_mondiale.md) ;
- les Trente Glorieuses (30_glorieuses.md).

Seulement il y a un problème, Bob a été malade pendant l'année. Du coup, ses fichiers ont été écrits dans le désordre : il a rédigé ses cours au fur et à mesure puis, quand il avait du temps, il a fait du rattrapage en prenant les cours d'Alice.

Il se retrouve alors avec l'historique de commits git suivant :

```
1 218bfcb Conclusion de la première guerre mondiale - rattrapage
2 b1cc9fd L'entre deux guerres
3 70c08d1 Début de la seconde guerre mondiale - rattrapage
4 ce99651 Les 30 glorieuses
5 61f6523 La seconde guerre mondiale - fin
6 b52fa42 La première guerre mondiale
7 8c310df La révolution industrielle
8 47901da Ajout des titres des parties
9 5b56868 Creation des fichiers du programme d'histoire
```

C'est cet historique que nous allons essayer de réordonner. On peut voir que les premiers commits correspondent au début du programme scolaire, puis ensuite Bob a été malade. Il a alors fait une pause, puis repris le cours dans l'ordre. Ensuite, il a rattrapé les cours manquants en recopiant les notes d'un autre élève, en partant du plus récent dans le programme au plus vieux.

L'historique montrée est antéchronologique, le commit le plus récent (celui que l'on vient de faire) est en haut de la liste, le plus vieux est en bas.

Le dépôt exemple

Le dépôt git de ce tp est présent dans le fichier zip tuto_git_rebase.zip.

Dans ce dépôt, vous devez trouver deux branches qui ont pour noms Bob et Alice. La branche Bob possède les écrits de Bob, tandis que celle d'Alice possède quelques anecdotes qu'elle a voulu lui donner plus tard. Pour l'instant, concentrez-vous sur la branche de Bob.

```
git checkout Bob
```

Vous pouvez alors vérifier les commits et leur mauvais rangement. Vous devriez obtenir la liste que l'on a vue plus tôt.

Les objectifs

Nos objectifs seront donc multiples. Afin de conserver un dépôt propre, nous allons effectuer une à une les opérations suivantes :

- mise dans l'ordre chronologique du cours des différents fichiers
- fusion des commits consécutifs traitant du même fichier et de la même partie "logique"
- fusion de la branche d'Alice pour "enrichir" notre branche de son contenu

Déplacer des commits

À l'heure actuelle, on a un historique un peu farfelu. Dans l'idéal, on aimerait que les éléments se suivent chronologiquement, voire que l'on ait uniquement un seul commit par période.

Ainsi, on va essayer d'obtenir le résultat suivant :

```
 ----------- AVANT -----------
1 218bfcb Conclusion de la première guerre mondiale - rattrapage
2 b1cc9fd L'entre deux guerres
3 70c08d1 Début de la seconde guerre mondiale - rattrapage
4 ce99651 Les 30 glorieuses
5 61f6523 La seconde guerre mondiale - fin
6 b52fa42 La première guerre mondiale
7 8c310df La révolution industrielle
8 47901da Ajout des titres des parties
9 5b56868 Creation des fichiers du programme d'histoire
10
11 ----------- APRES -----------
12 ce99651 Les 30 glorieuses
13 61f6523 La seconde guerre mondiale - fin
14 70c08d1 Début de la seconde guerre mondiale - rattrapage
15 b1cc9fd L'entre deux guerres
16 218bfcb Conclusion de la première guerre mondiale - rattrapage
17 b52fa42 La première guerre mondiale
18 8c310df La révolution industrielle
19 47901da Ajout des titres des parties
20 5b56868 Creation des fichiers du programme d'histoire
21
```

Comme vous pouvez le constater, de nombreux commits ont littéralement changé de place ! C'est ça que nous allons faire ici, déplacer des commits !

Et aussi impressionnant que cela puisse paraître, il va suffire d'utiliser une seule commande à bon escient pour le faire : `git rebase`. Mais attention, on ne l'utilise pas n'importe comment.

Pour l'utiliser, on va devoir lui spécifier le commit **le plus ancien** devant rester tel quel. Dans notre cas, nous souhaitons tout remettre en ordre jusqu'à "La premiere guerre mondiale" (b52fa42) qui, lui, ne bouge pas. On va alors lancer le rebase **en mode interactif** jusqu'à ce commit :

```
git rebase -i b52fa42^
```

Attention à ne pas oublier l'option `-i` pour le mode **interactif** ainsi que le `^` après l'identifiant du commit ! Ce dernier sert à indiquer que l'on veut remonter jusqu'à ce commit *inclus*.

Une nouvelle fenêtre s'ouvre alors avec plein de choses passionnantes :

```
 1 pick b52fa42 La première guerre mondiale
 2 pick 61f6523 La seconde guerre mondiale - fin
 3 pick ce99651 Les 30 glorieuses
 4 pick 70c08d1 Début de la seconde guerre mondiale - rattrapage
 5 pick b1cc9fd L'entre deux guerres
 6 pick 218bfcb Conclusion de la première guerre mondiale - rattrapage
 7
 8 # Rebase 8c310df..218bfcb onto 8c310df
 9 #
10 # Commands:
11 #  p, pick = use commit
12 #  r, reword = use commit, but edit the commit message
13 #  e, edit = use commit, but stop for amending
14 #  s, squash = use commit, but meld into previous commit
15 #  f, fixup = like "squash", but discard this commit's log message
16 #  x, exec = run command (the rest of the line) using shell
17 #
18 # If you remove a line here THAT COMMIT WILL BE LOST.
19 # However, if you remove everything, the rebase will be aborted.
```

Dans cet affichage, vous avez la liste des commits jusqu'au dernier que vous souhaitez garder tel quel. L'opération est maintenant simple, il va falloir déplacer les lignes pour les mettre dans l'ordre que vous voulez. L'ordre en question sera celui que l'on a vu juste au-dessus. Laissez les "pick" en début de ligne, ils sont là pour signifier que vous souhaitez utiliser le commit en question.

Vous devriez obtenir quelque chose comme ça avant de valider :

```
1 pick b52fa42 La première guerre mondiale
2 pick 218bfcb Conclusion de la première guerre mondiale - rattrapage
3 pick b1cc9fd L'entre deux guerres
4 pick 70c08d1 Début de la seconde guerre mondiale - rattrapage
5 pick 61f6523 La seconde guerre mondiale - fin
6 pick ce99651 Les 30 glorieuses
7
8 # Et en dessous le blabla précédent
```

Sauvegardez puis quittez l'éditeur. Le rebase se lance alors automatiquement… et vous crie dessus, c'est un échec !

Si vous utilisez la commande `git status` vous allez voir qu'il existe un conflit sur le fichier "la_seconde_guerre_mondiale.md". En l'ouvrant, vous verrez des marqueurs `<<<<<<`, `======` et `>>>>>` que git a rajoutés pour vous signaler les endroits où il n'arrive pas à faire une chose logique.

C'est donc à vous de jouer en éditant manuellement le fichier, pour qu'il ait l'allure escomptée. En l'occurrence, c'est simplement une ligne blanche qui l'ennuie. Supprimez-là, ainsi que les marqueurs, puis sauvegarder le fichier.

Nous allons maintenant signaler à git que le conflit est résolu en faisant un :

```
git add la_seconde_guerre_mondiale.md
```

Cela nous permet de rajouter le fichier dans l'index, puis on lui demande de continuer le rebase avec :

```
git rebase --continue
```

Git vous demandera alors de confirmer le message de commit (ce que vous ferez), puis continuera son bonhomme de chemin.

Un autre conflit similaire apparait alors, résolvez-le de la même manière.

À la fin, git doit afficher le message `Successfully rebased and updated refs/heads/Bob.` pour nous informer que tout va bien.

Si vous réaffichez votre historique, vos commits sont maintenant dans l'ordre !

Fusionner des commits

Cette fois-ci nous allons **fusionner** des commits pour réduire ces derniers, et surtout les rendre cohérents!

On va donc chercher à atteindre le schéma suivant :

```
 1 -------- AVANT --------
 2 56701fc Les 30 glorieuses
 3 a63009c Début de la seconde guerre mondiale - rattrapage
 4 752c8cd L'entre deux guerres
 5 328d49e Conclusion de la première guerre mondiale - rattrapage
 6 b52fa42 La première guerre mondiale
 7 8c310df La révolution industrielle
 8 47901da Ajout des titres des parties
 9 5b56868 Creation des fichiers du programme d'histoire
10
11 -------- APRES --------
12 d55d7d3 Les 30 glorieuses
13 3107653 La seconde guerre mondiale
14 ca137f6 L'entre deux guerres
15 ebfa63b La première guerre mondiale
16 8c310df La révolution industrielle
17 47901da Ajout des titres des parties
18 5b56868 Creation des fichiers du programme d'histoire
```

Nous allons aussi en profiter pour mettre à jour un message de commit !

Là encore, c'est la magie de la commande `rebase` qui va nous être utile. Comme précédemment, on va la lancer sur le dernier commit qui ne bouge pas, donc `b52fa42 La première guerre mondiale`. Ce qui nous donne :


```
git rebase -i b52fa42^
```

La machine se met en route et nous affiche le menu permettant de faire les modifications. Nous allons cette fois-ci lui dire de fusionner le commit 8f3d90c avec son prédécesseur et nous en profiterons pour éditer le message de commit de e4b80f96. On utilisera pour cela le mot-clé "fixup" ou "squash" pour fusionner (le dernier permet de changer le message de commit lors de la fusion), et nous utiliserons "reword" pour éditer juste le message du second commit à modifier.

Voici la séquence :

```
pick b52fa42 La première guerre mondiale
fixup 328d49e Conclusion de la première guerre mondiale - rattrapage
pick 752c8cd L'entre deux guerres
reword a63009c Début de la seconde guerre mondiale - rattrapage
pick 56701fc Les 30 glorieuses
```

Sauvegardez, quittez, puis laissez la magie opérer ! Lors du processus, l'éditeur devrait apparaître pour vous demander le nouveau commit pour l'opération de "reword".

Merger une autre branche

Une dernière fonction bien pratique de l'outil rebase est le *merge* (fusion) entre des branches. Ainsi, si vous travaillez sur une branche pour développer quelque chose, mais que vous voulez récupérer le contenu d'une autre branche pour mettre à jour la vôtre (vous synchroniser avec *master* par exemple), `rebase` peut vous y aider.

Cette fois-ci, on va se servir de rebase non pas en indiquant un commit mais en indiquant la branche que l'on aimerait récupérer dans notre travail. En l'occurrence, on va chercher à récupérer les modifications d'Alice (branche du même nom) qui a pris notre cours puis y a rajouté quelques anecdotes dans son coin.

Voici la petite commande à lancer :

```
git rebase Alice
```

Cette fois-ci, pas besoin du mode "interactif" `-i`.

Évidemment, il peut arriver que des conflits se présentent, comme dans ce cas précis.

Essayez de les corriger avec l'éditeur, puis il suffit de faire un `git add <lefichier>` suivi d'un `git rebase --continue` pour continuer le rebase !

Vous voilà maintenant synchronisés avec la branche d'Alice.