

TD 2c — Annuler et remettre à jour

Guide détaillé, étape par étape

Le TD simule une fausse manœuvre, l'enregistre par un commit, puis l'annule avec `git revert`. Il ajoute ensuite un commit sur la branche `main_code`, et met cette branche à jour sur `develop` avec `git rebase`. Il correspond aux questions 18 à 20 de la feuille, et dure une quinzaine de minutes.

On travaille dans le dépôt du TD 2a, tel que le TD 2b l'a laissé, `~/Desktop/info01/cours2/2a_premier_depot/travail/projet_2`, dans Git Bash. Le TD n'a pas de fichier de départ.

Étape	Objectif	Questions	Ce qu'on fait
1	enregistrer une erreur dans l'historique	18	vider les deux fichiers de <code>src</code> , et enregistrer ce commit
2	annuler un commit sans réécrire l'historique	19	repérer le commit fautif, et l'annuler avec <code>git revert</code>
3	mettre une branche à jour sur une autre	20	faire un commit sur <code>main_code</code> , puis la mettre à jour sur <code>develop</code> avec <code>git rebase</code>

Chaque étape commence par un encadré qui la résume. Ce que chaque étape fait constater est expliqué à la fin du guide, dans « Ce que le TD fait constater » : faire l'étape d'abord, et noter ce qu'on observe, avant de lire l'explication.

1 · La fausse manœuvre

À faire : sur develop, effacer tout le contenu de `src/main.py` et de `src/operations.py` ; enregistrer les deux fichiers vides ; faire un commit.

À obtenir : un commit suppression du code au sommet de develop.

```
cd ~/Desktop/info01/cours2/2a_premier_depot/travail/projet_2
git checkout develop
```

Dans VS Code, ouvrir `src/main.py`, tout sélectionner (Ctrl + A), supprimer, enregistrer. Faire de même pour `src/operations.py`. Dans le terminal :

```
git status
```

`git status` liste les deux fichiers comme modifié.

```
git add src
git commit -m "suppression du code"
```

Vérification : `git commit` affiche 2 files changed, 22 deletions(-).

La feuille pose la question : comment revenir en arrière ? Le commit est enregistré ; le contenu des deux fichiers n'est plus dans la copie de travail, mais il est dans les commits précédents.

2 · Annuler le commit

À faire : repérer l'identifiant du commit fautif avec `git log` ; l'annuler avec `git revert` ; accepter le message dans l'éditeur ; afficher le graphe.

À obtenir : un commit Revert "suppression du code", et les deux fichiers de nouveau remplis.

Repérer le commit

```
git log --oneline
```

```
b6cc6db suppression du code
482fb9c Merge branch 'operations' into develop
a655591 affichage du titre
...
```

Le commit fautif est le premier de la liste, le plus récent. Noter ses sept caractères, ici `b6cc6db` ; ils sont différents sur chaque poste.

L'annuler

```
git revert b6cc6db
```

Remplacer `b6cc6db` par l'identifiant relevé. Git ouvre l'éditeur dans le terminal, avec le message Revert "suppression du code" déjà écrit. Comme au TD 2b : Échap, puis `:wq`, puis Entrée.

```
[develop 06b3d3b] Revert "suppression du code"
2 files changed, 22 insertions(+)
```

```
git llog
```

Vérification : le graphe commence par ces trois lignes, et `src/main.py` contient de nouveau le programme :

```
* 06b3d3b (HEAD -> develop) Revert "suppression du code"
* b6cc6db suppression du code
* 482fb9c Merge branch 'operations' into develop
```

À noter : si l'identifiant du commit créé par `git revert` est celui que vous aviez relevé ; si le commit suppression du code est encore dans le graphe.

3 · Un commit sur main_code, puis le rebase

À faire : se placer sur main_code ; ajouter un commentaire en première ligne de src/main.py et faire un commit ; afficher le graphe de toutes les branches ; lancer `git rebase develop` ; afficher de nouveau le graphe.

À obtenir : le commit commentaire en tête de main.py au sommet de develop, avec un autre identifiant qu'avant le rebase.

Le commit sur main_code

```
git checkout main_code
```

Dans VS Code, ouvrir src/main.py et ajouter en première ligne, au-dessus du texte entre triples guillemets :

```
# Point d'entrée de la calculatrice : affiche le titre.
```

Enregistrer, puis :

```
git commit -am "commentaire en tête de main.py"
git llog --all
```

-a ajoute à la zone de préparation les fichiers suivis qui ont été modifiés, ce qui évite le `git add`. Le graphe montre la branche main_code, partie de affichage du titre, à côté de develop :

```
* 06b3d3b (develop) Revert "suppression du code"
* b6cc6db suppression du code
* 482fb9c Merge branch 'operations' into develop
|\
| * 4a4a187 (operations) ajout des quatre opérations
| | * e8f1afe (HEAD -> main_code) commentaire en tête de main.py
| | /
| /|
* | a655591 affichage du titre
| /
* d3c6642 (documentation) description du projet
* 1bab660 (master) ajout du README
```

À noter : l'identifiant du commit commentaire en tête de main.py, ici e8f1afe, et le commit dont il part.

Le rebase

```
git rebase develop
git llog --all
```

`git rebase develop` met à jour la branche courante, main_code, sur develop. Git affiche :

```
Rebasage et mise à jour de refs/heads/main_code avec succès.
```

Vérification : le commit du commentaire est maintenant au sommet de develop, et le graphe ne montre plus qu'une ligne au-dessus du merge :

```
* 29a4b05 (HEAD -> main_code) commentaire en tête de main.py
* 06b3d3b (develop) Revert "suppression du code"
* b6cc6db suppression du code
* 482fb9c Merge branch 'operations' into develop
```

À noter : l'identifiant du commit du commentaire après le rebase.

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

git revert ajoute un commit

Le commit créé par `git revert` a son propre identifiant, différent de celui du commit annulé. Il est ajouté au sommet de `develop`, et fait l'inverse du commit visé : celui-ci avait retiré 22 lignes, le nouveau commit les remet. Le projet revient au contenu d'avant la suppression, et le commit fautif reste dans l'historique, avec sa correction au-dessus.

Aucun commit n'est effacé : l'historique s'allonge, et tout état passé reste accessible. Une commande qui retire des commits d'une branche existe, `git reset`, mais elle n'est pas vue dans cette séance.

Le rebase refait le commit

Avant le rebase, le commit du commentaire, `e8f1afe`, avait pour parent affichage du titre : `main_code` était partie de là, avant la fusion, la suppression et l'annulation faites sur `develop`. Le rebase a refait ce commit au bout de `develop`. Le nouveau commit, `29a4b05`, a le même message et apporte la même modification, mais son parent est maintenant le commit Revert "suppression du code". L'identifiant d'un commit est calculé sur son contenu et sur son parent : le parent a changé, donc l'identifiant aussi. C'est ce que montrent les commits « bis » de la partie « Branches, fusion et conflits ».

L'ancien commit `e8f1afe` n'appartient plus à aucune branche, et `git llog --all` ne l'affiche plus. L'historique de `main_code` a été réécrit : une branche que quelqu'un d'autre a déjà récupérée ne se rebase pas, parce que sa copie contiendrait encore l'ancien commit. Le cours 6 y revient.

Le commit du commentaire est ce qui rend le rebase visible. Sans lui, `main_code` ne contiendrait que l'affichage du titre, que `develop` contient déjà depuis le TD 2b : le rebase n'aurait rien à refaire, et il avancerait seulement `main_code` jusqu'au sommet de `develop`.

La branche `main_code` est maintenant à jour sur `develop` : le TD 2d y ajoute un commit.