

TD 2d – Créer et résoudre un conflit

Guide détaillé, étape par étape

Le TD écrit deux versions différentes de la saisie d'une opération dans `src/main.py`, sur deux branches, puis fusionne les deux branches dans `develop`. Les deux versions modifient les mêmes lignes : la seconde fusion s'arrête sur un conflit, que l'on résout. Le TD correspond aux questions 21 à 25 de la feuille, et dure environ vingt-cinq minutes.

On travaille dans le dépôt du TD 2a, tel que le TD 2c l'a laissé, `~/Desktop/info01/cours2/2a_premier_depot/travail/projet_2`, dans Git Bash. Les deux versions sont fournies dans le dossier du TD :

```
2d_conflits\  
├─ depart\  
│   ├── main_regex.py      la saisie lue avec une expression régulière  
│   └─ main_operations.py  la saisie découpée à la main  
├─ README.md  
└─ td_2d_conflits.pdf
```

Étape	Objectif	Questions	Ce qu'on fait
1	modifier un fichier sur une branche	21	la première version, sur <code>main_code</code>
2	modifier la même ligne sur une autre branche	22 et 23	la seconde version, sur une nouvelle branche <code>main_code_bis</code>
3	fusionner deux modifications de la même ligne	24	fusionner les deux branches dans <code>develop</code> : le conflit
4	résoudre un conflit, puis terminer la fusion	25	résoudre le conflit, et terminer la fusion

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 première version, sur main_code

À faire : sur main_code, remplacer le contenu de src/main.py par celui de depart/main_regex.py ; faire un commit ; ne pas le fusionner.

À obtenir : un commit lecture de l'opération avec une regex sur main_code.

```
cd ~/Desktop/info01/cours2/2a_premier_depot/travail/projet_2
git checkout main_code
cp ../../2d_conflits/depart/main_regex.py src/main.py
git status
```

cp remplace src/main.py par le fichier fourni, qui reprend le titre de la question 11 et y ajoute la saisie. Le commentaire ajouté au TD 2c disparaît du fichier ; il reste dans l'historique. git status liste src/main.py comme modifié.

```
git add src/main.py
git commit -m "lecture de l'opération avec une regex"
```

La nouvelle version lit une ligne tapée au clavier, y cherche une opération entre deux nombres avec une expression régulière, puis appelle les fonctions de operations.py. Pour l'essayer : python src/main.py, puis taper 12-5 et Entrée ; le programme affiche 7.0.

2 · La seconde version, sur main_code_bis

À faire : revenir sur develop ; y créer main_code_bis ; remplacer src/main.py par depart/main_operations.py ; faire un commit.

À obtenir : un commit lecture de l'opération par découpage sur main_code_bis.

```
git checkout develop
git checkout -b main_code_bis
```

À noter : ce que contient src/main.py sur cette branche, et si la version de l'étape 1 y est.

```
cp ../../../../2d_conflits/depart/main_operations.py src/main.py
git add src/main.py
git commit -m "lecture de l'opération par découpage"
```

Cette version fait le même travail autrement : elle découpe la ligne tapée autour du signe de l'opération, et vérifie que les deux morceaux sont des nombres.

3 · Fusionner les deux branches : le conflit

À faire : revenir sur develop ; y fusionner main_code_bis, puis main_code.

À obtenir : le second merge s'arrête sur un conflit dans src/main.py.

```
git checkout develop
git merge main_code_bis
```

Le premier merge passe, en avance rapide (Fast-forward).

```
git merge main_code
```

```
Fusion automatique de src/main.py
CONFLIT (contenu) : Conflit de fusion dans src/main.py
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

```
git status
```

```
Sur la branche develop
Vous avez des chemins non fusionnés.
(réglez les conflits puis lancez "git commit")
(utilisez "git merge --abort" pour annuler la fusion)
```

```
Chemins non fusionnés :
(utilisez "git add <fichier>..." pour marquer comme résolu)
modifié des deux côtés : src/main.py
```

L'invite de Git Bash se termine maintenant par (develop|MERGING) : la fusion est en cours.

4 · Résoudre le conflit

À faire : ouvrir `src/main.py` ; pour chaque conflit, choisir la version à garder et supprimer les marqueurs ; vérifier que le programme se lance ; marquer le fichier comme résolu ; terminer la fusion.

À obtenir : un commit Merge branch 'main_code' into develop, et un programme qui calcule.

Lire les marqueurs

Ouvrir `src/main.py` dans VS Code. Le fichier contient deux conflits : le début du fichier, où les deux versions ont un texte d'introduction et des import différents, et la boucle de saisie. Chacun a la forme suivante :

```
<<<<<<< HEAD
    (la version de develop, venue de main_code_bis)
=====
    (la version de main_code, qui arrive)
>>>>>>> main_code
```

À noter : ce qui se passe si l'on lance `python src/main.py` à ce moment.

Choisir

Pour chaque conflit, garder l'une des deux versions : celle qui est entre `<<<<<<< HEAD` et `=====`, ou celle qui est entre `=====` et `>>>>>>> main_code`. Supprimer l'autre, puis les trois lignes de marqueurs. Garder la même version dans les deux conflits : les import du début doivent correspondre au code de la boucle, qui appelle `add(...)` dans la version de `main_code` et `op.add(...)` dans celle de `main_code_bis`.

VS Code affiche au-dessus de chaque conflit des liens cliquables : Accept Current Change garde la version de HEAD, Accept Incoming Change celle qui arrive, et chacun supprime les marqueurs du conflit.

Enregistrer, puis vérifier que le fichier ne contient plus de marqueur et que le programme calcule :

```
grep -n "<<<<<<<\|=====\|>>>>>>>" src/main.py
python src/main.py
```

Vérification : `grep` n'affiche rien ; le programme affiche son titre, attend une opération, et affiche 7.0 pour 12-5.

Terminer la fusion

```
git add src/main.py
git merge --continue
```

Git ouvre l'éditeur avec le message Merge branch 'main_code' into develop : Échap, :wq, Entrée.

```
git llog
```

Vérification : le graphe commence ainsi :

```
* 4ced033 (HEAD -> develop) Merge branch 'main_code' into develop
|\
| * 6745df8 (main_code) lecture de l'opération avec une regex
| * 29a4b05 commentaire en tête de main.py
* | 4d853e1 (main_code_bis) lecture de l'opération par découpage
|/
* 06b3d3b Revert "suppression du code"
```

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

Étape 2 : chaque branche a son contenu

Sur `main_code_bis`, créée depuis `develop`, `src/main.py` ne contient que le titre de la question 11 : la version de l'étape 1 est sur `main_code`, et n'a pas été fusionnée. Les deux versions sont écrites à partir du même état du fichier, comme le feraient deux personnes qui travaillent en même temps.

Pourquoi le second merge s'arrête

Le premier merge est une avance rapide : `develop` n'avait pas avancé depuis la création de `main_code_bis`. Le second doit réunir deux modifications du même fichier, faites à partir du même état. Git réunit seul des modifications qui portent sur des lignes différentes. Ici, les deux branches ont remplacé les mêmes lignes, de deux façons différentes : git ne peut pas choisir, et le merge s'arrête. Il écrit alors les deux versions dans le fichier, à chaque endroit où elles diffèrent, et la fusion reste en cours jusqu'à ce que la personne qui fusionne ait choisi.

Les marqueurs sont du texte

Lancé avec les marqueurs, le programme s'arrête sur une erreur :

```
<<<<<<< HEAD
^
SyntaxError: invalid syntax
```

Les lignes `<<<<<<<`, `===== et >>>>>>> sont écrites dans le fichier comme n'importe quel texte. Git ne vérifie pas qu'elles ont disparu : git add marque le fichier comme résolu même s'il en contient encore. Un fichier Python qui en contient ne s'exécute plus. Oublier d'en supprimer une est l'erreur la plus fréquente du TD ; grep les retrouve.`

Résoudre, c'est écrire un nouvel état

La version gardée n'est pas imposée : on peut garder l'une, l'autre, ou écrire une version qui combine les deux. Le commit de fusion enregistre ce que contient le fichier au moment de `git add`, et c'est ce contenu que `develop` porte ensuite. Tant que la fusion n'est pas terminée, `git merge --abort` remet `develop` dans l'état d'avant le merge.