

TD 2b — Branches et fusions

Guide détaillé, étape par étape

Le TD organise le projet du TD 2a en branches, comme le décrit la partie « Lire et tenir un dépôt » : une branche `develop` qui réunit le travail, et une branche par tâche. On écrit la description du projet sur une branche, le programme principal sur une deuxième et les fonctions de calcul sur une troisième, comme si deux personnes travaillaient en parallèle, puis on fusionne les trois dans `develop`. Le TD correspond aux questions 6 à 17 de la feuille, et dure une trentaine de minutes.

On travaille dans le dépôt créé au TD 2a, `~/Desktop/info01/cours2/2a_premier_depot/travail/projet_2`, dans Git Bash.

Étape	Objectif	Questions	Ce qu'on fait
1	développer sur une branche, puis fusionner	6 à 8	créer <code>develop</code> et documentation, décrire le projet, fusionner
2	isoler le programme principal sur sa branche	9 à 12	créer <code>main_code</code> , y écrire le programme principal
3	isoler les fonctions de calcul sur une autre branche	13 à 16	créer <code>operations</code> , y écrire les fonctions de calcul
4	réunir deux branches dans une troisième	17	fusionner <code>main_code</code> , puis <code>operations</code> , dans <code>develop</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.

Chaque étape se termine par `git llog`. Le graphe qu'il affiche est la vérification du TD : le comparer à celui du guide.

1 · La branche documentation

À faire : créer develop depuis master, puis documentation depuis develop ; décrire le projet dans README.md ; faire un commit ; revenir sur develop et y fusionner documentation.

À obtenir : develop et documentation désignent le même commit, description du projet.

Se placer dans le projet

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

Vérification : l'invite se termine par projet_2 (master), et git status affiche rien à valider, la copie de travail est propre.

Créer les deux branches

```
git branch develop
git checkout develop
git checkout -b documentation
git branch
```

git branch develop crée la branche sans s'y placer ; git checkout develop s'y place. git checkout -b documentation fait les deux en une commande, à partir du commit courant, celui de develop. git branch liste les branches, et marque d'une étoile la branche courante :

```
develop
* documentation
master
```

Décrire le projet

Ouvrir README.md dans VS Code (File, Open Folder, puis le dossier projet_2). Sous la première ligne, écrire une phrase qui décrit le projet : une petite calculatrice en Python, qui fournit des fonctions mathématiques utiles. À la fin, écrire la ligne d'image Markdown du logo de l'école. L'archive ne fournit pas l'image : la ligne montre la syntaxe, et l'image ne s'affiche pas dans l'aperçu.

```
# Geo Calculatrice

Une petite calculatrice en Python, qui fournit des fonctions mathématiques utiles.

![Logo de l'école](logo.png)
```

Enregistrer le fichier (Ctrl + S), puis, dans le terminal :

```
git status
git diff
git add README.md
git commit -m "description du projet"
```

À noter : ce que git diff affiche avant le git add, et le signe qui commence les lignes ajoutées.

Fusionner dans develop

```
git checkout develop
git merge documentation
git llog
```

À noter : le mot que git merge affiche à sa deuxième ligne.

Vérification :

- * d3c6642 (HEAD -> develop, documentation) description du projet
- * 1bab660 (master) ajout du README

2 · La branche main_code

À faire : depuis develop, créer main_code ; y créer src/main.py avec le contenu de main_base.py ; faire un commit ; revenir sur develop.

À obtenir : le commit affichage du titre sur main_code, et le dossier src absent quand on est revenu sur develop.

```
git checkout -b main_code
mkdir src
cp ../../../../2b_branches/depart/main_base.py src/main.py
python src/main.py
```

Le chemin ../../../../2b_branches/depart/main_base.py remonte de projet_2 à travail, puis à 2a_premier_depot, puis à cours2, et redescend dans le dossier du TD 2b. La commande cp fait ce que demande la question 11 : copier le contenu de main_base.py dans src/main.py. Le fichier peut aussi être créé dans VS Code et le texte copié à la main.

Vérification : python src/main.py affiche le titre :

```
Geo Calculatrice
*****
```

```
git add src
git commit -m "affichage du titre"
git checkout develop
ls
```

À noter : ce que ls affiche sur develop, et si le dossier src en fait partie.

3 · La branche operations

À faire : depuis develop, créer operations ; y créer src/operations.py avec les quatre fonctions ; faire un commit ; revenir sur develop.

À obtenir : le commit ajout des quatre opérations sur operations.

```
git checkout -b operations
mkdir src
```

La branche operations part de develop, qui ne contient pas encore src/main.py : le dossier src n'existe pas sur cette branche, et mkdir src le crée. Créer ensuite src/operations.py dans VS Code, avec les quatre fonctions de la question 15 :

```
def add(a, b):
    return a + b

def mult(a, b):
    return a * b

def neg(a):
    return -a

def inv(a):
    return 1 / a
```

Fonction	Ce qu'elle renvoie
add(a, b)	la somme de a et b
mult(a, b)	le produit de a et b
neg(a)	l'opposé de a
inv(a)	l'inverse de a

Enregistrer, puis :

```
git add src
git commit -m "ajout des quatre opérations"
git checkout develop
```

4 · Fusionner dans develop

À faire : sur develop, fusionner main_code, afficher le graphe, puis fusionner operations, et afficher le graphe.

À obtenir : un commit de fusion Merge branch 'operations' into develop, qui a deux parents.

Le premier merge

```
git merge main_code
git llog
```

```
Mise à jour d3c6642..a655591
Fast-forward
 src/main.py | 8 +++++++
 1 file changed, 8 insertions(+)
 create mode 100644 src/main.py
```

Le second merge, et l'éditeur

```
git merge operations
```

Git ouvre un éditeur dans le terminal, avec un message déjà écrit : Merge branch 'operations' into develop. Sous Windows, c'est vim. Pour accepter le message et quitter : taper Échap, puis :wq, puis Entrée. Ne pas fermer la fenêtre du terminal, ce qui laisserait la fusion inachevée.

```
Merge made by the 'ort' strategy.
 src/operations.py | 14 ++++++++
 1 file changed, 14 insertions(+)
 create mode 100644 src/operations.py
```

Vérification : git llog affiche le graphe suivant ; les identifiants diffèrent d'un poste à l'autre.

```
* 482fb9c (HEAD -> develop) Merge branch 'operations' into develop
|\
| * 4a4a187 (operations) ajout des quatre opérations
* | a655591 (main_code) affichage du titre
|/
* d3c6642 (documentation) description du projet
* 1bab660 (master) ajout du README
```

ls src affiche maintenant main.py et operations.py.

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

Étape 1 : ce que `git diff` affiche

Avant le `git add`, `git diff` compare le fichier modifié au dernier commit. Les lignes ajoutées commencent par +, et la première ligne du fichier, inchangée, est reprise sans signe pour situer la modification. Après le `git add`, `git diff` n'affiche plus rien : la modification est dans la zone de préparation, et `git diff --staged` l'afficherait.

Une branche change les fichiers du dossier

Revenu sur `develop` à l'étape 2, `ls` n'affiche plus que `README.md` : le dossier `src` a disparu. Il n'est pas perdu : il est enregistré dans le commit de `main_code`, et `git checkout main_code` le fait réapparaître. Changer de branche remplace les fichiers du dossier de travail par ceux du dernier commit de la branche. C'est aussi pourquoi `src` doit être recréé à l'étape 3 : la branche `operations` part d'un commit qui ne le contient pas.

Deux sortes de fusion

Le premier merge affiche Fast-forward, une **avance rapide** : `develop` n'avait reçu aucun commit depuis la création de `main_code`. Il n'y avait rien à réunir, et `git` a déplacé le nom `develop` sur le dernier commit de `main_code`, sans créer de commit. Le merge de l'étape 1 était de la même sorte.

Le second merge crée un **commit de fusion**. Entre-temps, `develop` avait avancé avec le premier merge, et `operations` avait reçu son propre commit : les deux branches avaient chacune un commit que l'autre n'avait pas. Le commit de fusion a deux parents, `a655591` et `4a4a187`, ce que le graphe dessine par `| \` au-dessus et `| /` en dessous. Il contient les fichiers des deux branches : `main.py` et `operations.py`.

L'organisation du projet

Le projet suit l'organisation décrite dans « Lire et tenir un dépôt » : `master` n'a pas bougé depuis le TD 2a, `develop` réunit le travail, et chaque tâche a eu sa branche. `master` recevra la version terminée au TD 2e.