

# Git à plusieurs : mode d'emploi

Florian Legendre

Université de Poitiers

Année 2020 - 2021

# Table of Contents

- 1 Rappels sur les conflits d'édition
- 2 Gestion des conflits d'édition avec Git

# Table of Contents

- 1 Rappels sur les conflits d'édition
  - Quelques remarques préliminaires
  - Rappel de la notion de conflits d'édition
  - Quel est le rôle de Git dans la gestion de ces conflits ?

# Git n'est pas GitHub !

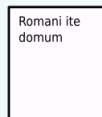
À l'instar des branches où je considérais que vous aviez un dépôt distant et où je vous montrais comment configurer vos branches locales/distantes depuis Git, ici je vais vous présenter une dernière notion de Git qui est indépendante du choix de dépôt distant que vous ferez.

Cette notion est celle de conflit d'éditions. Bien que le mot "conflit" puisse faire peur il s'agit de quelque-chose de tout à fait ordinaire quand on travaille à plusieurs sur un même projet. Git simplifie grandement la gestion de ces conflits.

# Rappel du problème



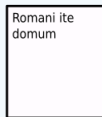
Jean



jean\_V1.docx

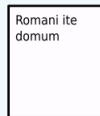


Anne



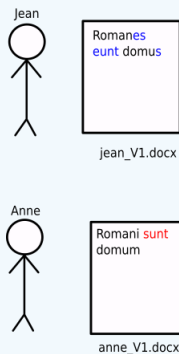
anne\_V1.docx

DropBox



notreDevoir\_V1.docx

# Rappel du problème



DropBox



# Rappel du problème



Jean

Romanes  
eunt domus

jean\_V1.docx



Anne

Romani sunt  
domum

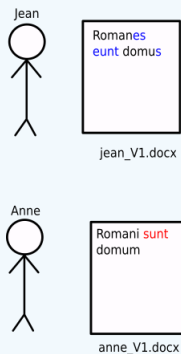
anne\_V1.docx

DropBox

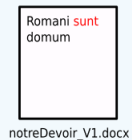


notreDevoir\_V1.docx

# Rappel du problème

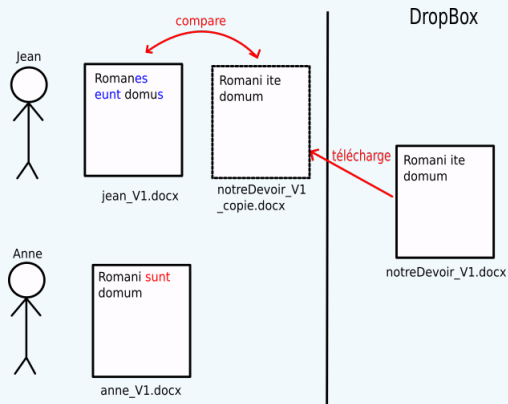


DropBox

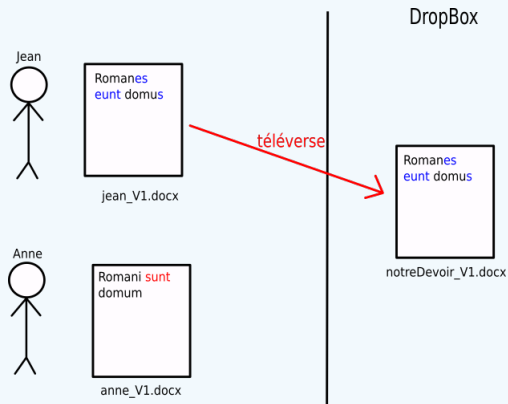




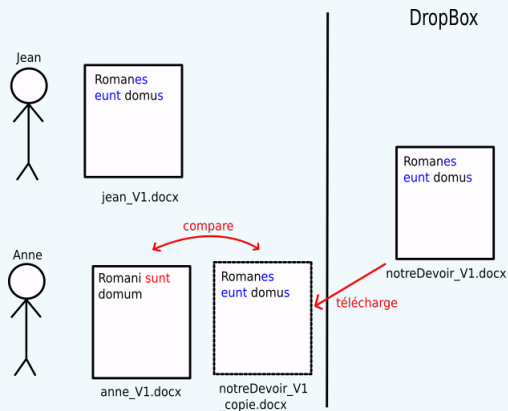
# Rappel de la solution "à la main"



# Rappel de la solution "à la main"



# Rappel de la solution "à la main"



# Rôle de Git

Git ne peut pas décider à votre place de ce qui doit être supprimé ou gardé ! Ou alors nous perdons tous notre travail car les ordinateurs peuvent coder...

Mais Git peut vous indiquer dans quels fichiers et à quelles lignes vos deux versions sont en conflit.

# Table of Contents

- 2 Gestion des conflits d'édition avec Git
  - Git fait la détection
  - Vous faites la résolution

# Un conflit détecté par Git

Un conflit survient quand on met à jour sa branche locale avec la branche distante suivie et qu'une ou plusieurs versions de la branche suivie fait état de zones éditées qui sont en conflit avec celles de notre branche locale :

```
crex@crex:~/projects/GitLearn(dev)$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 705 bytes | 16.00 KiB/s, done.
From github.com:Chuxclub/GitLearn
a223800..d261500 dev -> origin/dev
Auto-merging codes/test.txt
CONFLICT (content): Merge conflict in codes/test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

**Listing 1** – Exemple de détection automatique de conflit d'édition

# Un conflit détecté par Git

Pour savoir quels fichiers sont en conflits vous pouvez aussi utiliser la commande *git status* qui vous renseigne en plus sur la marche à suivre :

```
crex@crex:~/projects/GitLearn(dev)$ git status
On branch dev
Your branch and 'origin/dev' have diverged,
and have 2 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   codes/test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

**Listing 2** – Les fichiers en conflit indiqués par git status

# Un conflit détecté par Git

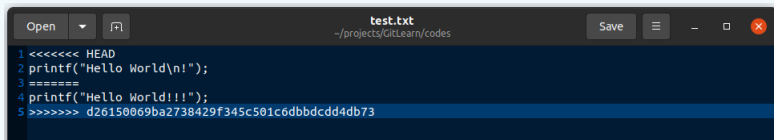
Normalement à chaque fois que vous mettez à jour votre branche locale en faisant un *git pull*, git crée un commit qui enregistre cette mise à jour (comme ça il est facile de défaire une mise à jour avec un *git revert <id>*).

Cependant, quand Git détecte un conflit il attend que vous ayiez résolu ce-dernier avant de faire un commit de la mise à jour.



# Un conflit détecté par Git

Lors d'un conflit Git écrit dans le fichier conflictuel comme ci-dessous :

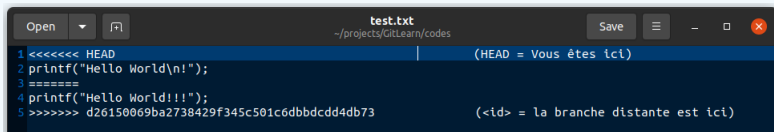


```
test.txt
~/projects/GitLearn/codes

1 <<<<<< HEAD
2 printf("Hello World\n!");
3 =====
4 printf("Hello World!!!");
5 >>>>>> d26150069ba2738429f345c501c6dbbdcdd4db73
```

# Un conflit détecté par Git

Vous remarquerez les deux parties créées par les <<<, === et >>> de Git. La première partie (HEAD) correspond à ce qu'il y a chez vous (Rappel : HEAD = "vous êtes ici pour Git"), la seconde partie correspond à ce qu'il y a sur la branche distante :



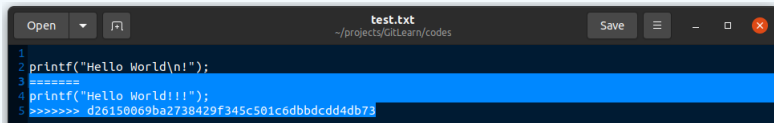
```
1 <<<<<< HEAD | (HEAD = Vous êtes ici)
2 printf("Hello World\n!");
3 =====
4 printf("Hello World!!!");
5 >>>>>> d26150069ba2738429f345c501c6dbbdcdd4db73 | (<id> = la branche distante est ici)
```

# Vous faites la résolution

Git vous a épargné la détection des conflits. Mais la gestion/résolution reste à votre charge ! Pour ce faire il suffit d'aller dans le fichier conflictuelle/édité par Git et de supprimer/garder les lignes qui vous intéressent.

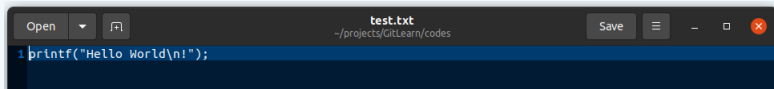
# Vous faites la résolution

Je peux faire ce que je veux lors d'une édition de conflits : garder tout ou partie ou rien du tout du travail de mon collègue ou le mien. Vous faites ce choix selon vos besoins. Ici je choisis de supprimer entièrement le travail qui était sur la branche distante :



```
test.txt
~/projects/GitLearn/codes

1
2 printf("Hello World\n!");
3 =====
4 printf("Hello World!!!");
5 >>>>>> d26150069ba2738429f345c501c6dbbdcdd4db73
```



```
test.txt
~/projects/GitLearn/codes

1 printf("Hello World\n!");
```

# Vous faites la résolution

Une fois que vous avez fini de choisir ce que vous vouliez garder ou non (discutez-en avec la personne concernée avant !) vous pouvez ajouter les fichiers au staging avec `git add <nom_fichier>` afin de préparer le commit que Git a suspendu en vous attendant :

---

