# Introduction à la gestion des branches avec Git

# Table des matières

# I Créer des branches

**Branche** Az Définition

Une branche est une divergence effectuée pour travailler séparément sur des fonctionnalités différentes.

Jusqu'à maintenant, on s'est contentés de travailler sur une seule branche.

**Intérêt des branches** Fondamental

- Gestion séparée des fonctionnalités
- Meilleure compréhensibilité des changements
- Meilleure traçabilité des bugs.

**Création d'un projet (dépôt distant)** Méthode

Pour le Gitlab de l'UTC :

- Se rendre sur https://gitlab.utc.fr
- Une fois connecté avec ses identifiants CAS, il suffit d'appuyer sur le bouton « New project » et de renseigner un nom et un niveau de visibilité

## Pour git, qu'est-ce qu'une branche ?

Pour git, une branche, c'est une étiquette un peu spéciale, sur laquelle on peut faire un peu plus d'opérations que d'habitude.

On peut se déplacer, effectuer un diff, un show

**La branche master** Fondamental

La branche master est la branche principale du projet. Elle est créée automatiquement au départ du projet et sa tête devrait être une versions fonctionnelle du projet.

# II Créer des branches et les utiliser

### Créer une branche

Méthode

La manière commune de créer une branche est d'utiliser

```
1 git branch ma_branche
```

Il existe d'autres manières de créer une branche, comme par exemple

```
1 git checkout -b ma_branche master
```

Cette commande dit :

1. Déplace toi sur la branche ma_branche.
2. l'option -b indique que la branche ma_branche n'existe pas et qu'il faut la créer.
3. master est le point de départ de la branche.

### Faire un commit sur une branche

Az Définition

Soit une branche ma_branche ayant pour dernier commit mon_commit, faire un nouveau commit sur la branche ma_branche signifie

- Créer un commit fils au commit mon_commit
- Déplacer l'étiquette de la branche sur le nouveau commit

### Vérifier la branche sur laquelle on est

Méthode

Pour vérifier la branche sur laquelle on est on peut utiliser la commande git branch sans utiliser d'argument. Elle liste toutes les branches disponibles et met une étoile sur la branche master

### Ne plus avoir à vérifier la branche sur laquelle on est

Conseil

Il existe de nombreuses manières de changer son prompt pour qu'il affiche en permanence l'état du dépôt sur lequel on est. Et notamment la branche sur laquelle on est.

# III Exercice : Application

## Question 1

[solution n°1 p. 14]

Créez un dossier init_git et créez-y un dépôt git. Mettez-y un fichier toto.txt. Commitez les modifications. Ensuite, créez trois branches et déplacez vous

### Indice :

```
mkdir init_git
cd init_git
git init
touch toto.txt
git add toto.txt
git commit -m "premier commit"
git branch alice
git branch bob
git checkout -b charlie
```

## Question 2

[solution n°2 p. 14]

Ajoutez une ligne contenant "pomme" dans votre fichier aller sur la branche alice et constater que le changement n'est pas fait sur la nouvelle branche

# IV La fusion de branches

**Fusion de branches** Méthode

Fusionner des branches, c'est réunir le travail de deux personnes différentes en une seule, pour le faire, on utilise la commande git merge, celle-ci crée en général un commit de fusion

Remarque

Le commit de fusion est le seul type de commit qui puisse avoir deux parents.

Syntaxe

```
1 git merge target_branch
```

La commande précédente met toutes les modifications de la branche cible dans la branche courante.

# V Exercice : Défi : Veille

## Question

[solution n°3 p. 14]

Reprenez votre dépôt et executez les commandes suivantes. Que font-elles ? Fusionnez les deux branches

```
git checkout alice
touch alicefile.txt
git add alicefile.txt
git commit -m "Create file for alice"
git checkout bob
touch bobfile.txt
git add bobfile.txt
git commit -m "Create file for bob"
```

### Indice :

Vous êtes encore dans la branche de bob, pour fusionner la branche de Alice dans celle de Bob, il faut utiliser git merge

# VI Rattraper son retard

**Rebaser** Az Définition

Rebaser, c'est réorganiser son historique et ses étiquettes.

**Rattraper son retard** Fondamental

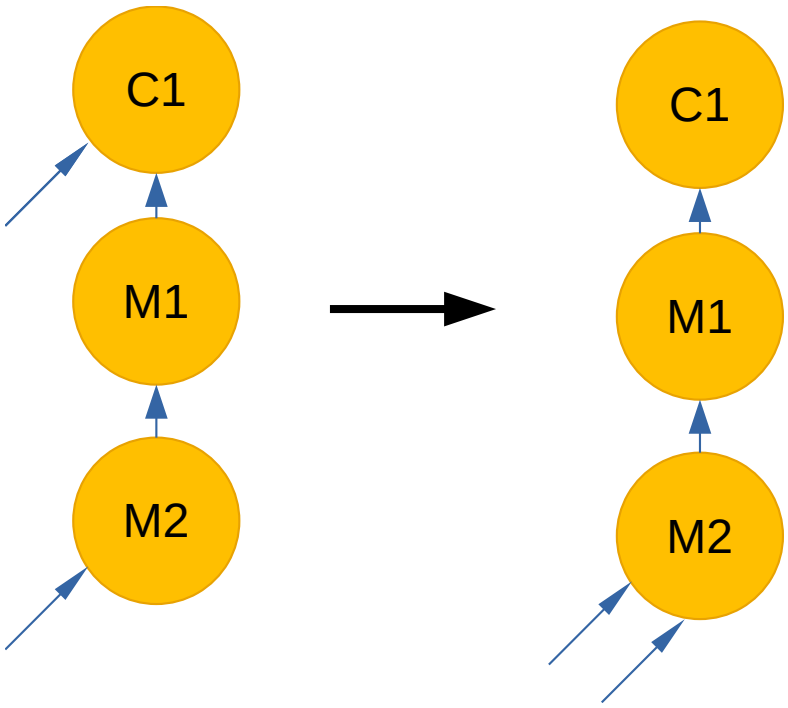
Pour rattraper son retard sur une branche distante, on effectue souvent un rebase.

Ce rebase prend l'ensemble des commits de la branche cible et les intègre dans l'historique de la branche courante.

**Le rebase est à consommer avec modération** Attention

Rebase, c'est modifier son historique, et c'est donc prendre le risque de ne plus être synchronisé avec les dépôts distants, c'est donc à utiliser avec beaucoup de précautions.

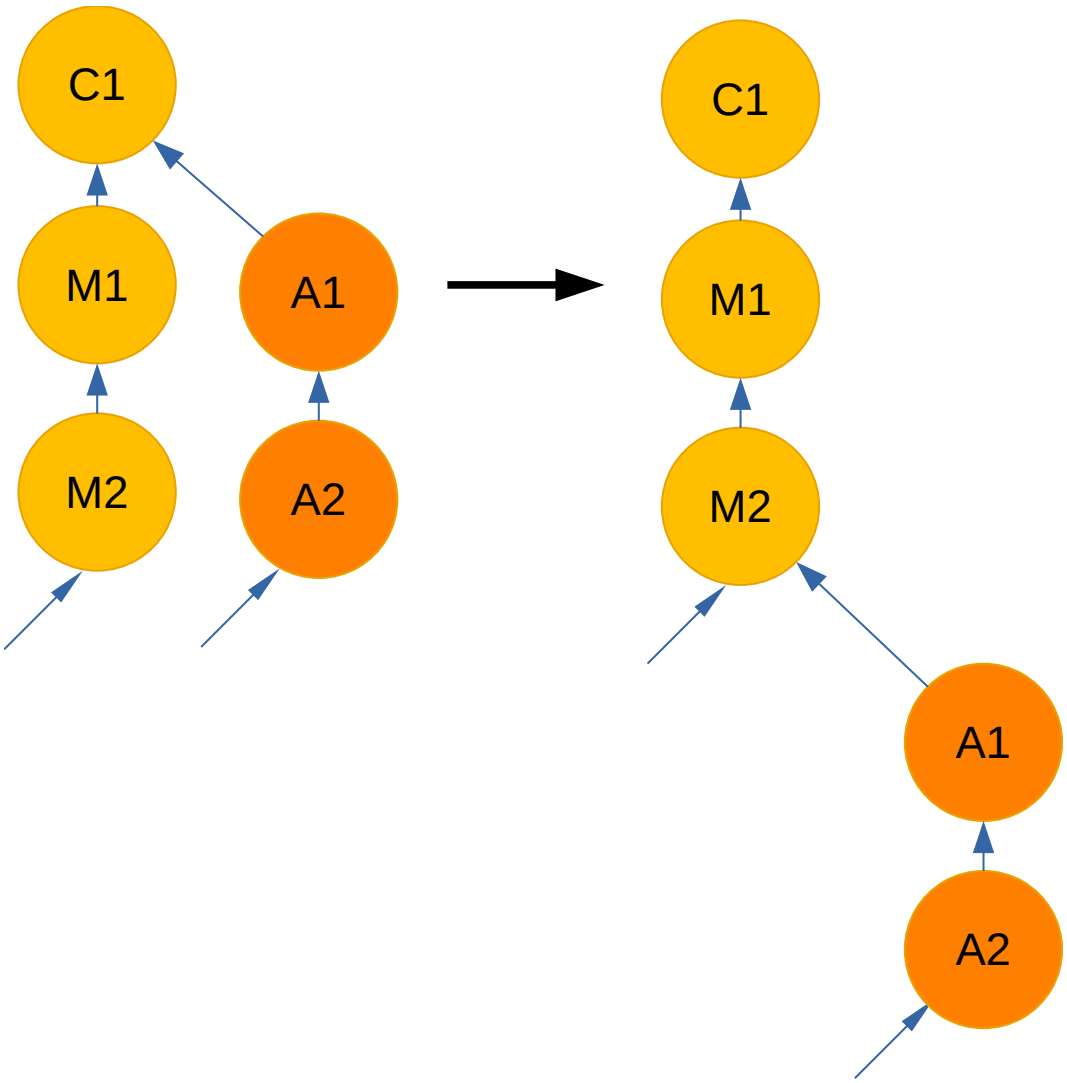
**Un rebase simple** Exemple




Si vous avez du retard sur une branche mais que vous n'avez rien fait, vous pouvez rebase sans problèmes.

**Un rebase un peu plus complexe**

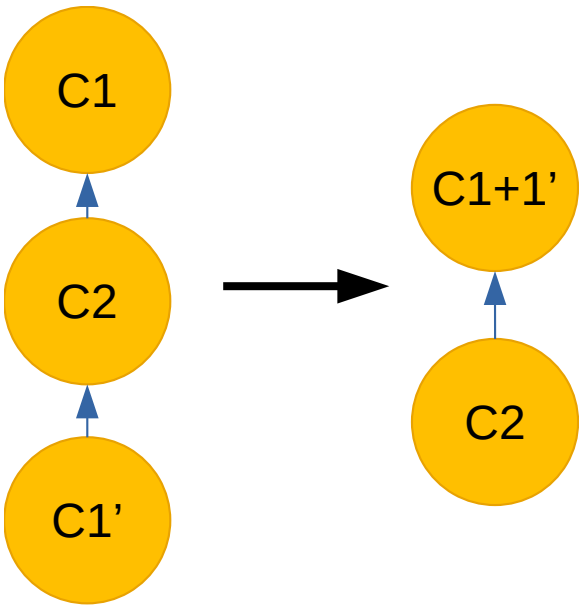
Exemple




Si vous avez du retard sur une branche et que vous avez effectué des commits, c'est un peu plus risqué, mais pas de panique

**Un rebase pour la forme**

Exemple




Le rebase, c'est aussi ça. Mais on ne va pas s'y appesantir.

# VII Exercice : Défi : Veille

## Question 1

[solution n°4 p. 14]

Retournez dans votre dépôt. Charlie est en retard sur alice et bob, faites lui rattraper son retard.

## Question 2

[solution n°5 p. 14]

Que font les commandes suivantes ?

```
git checkout alice
echo "poire" >> alicefile.txt
git commit -am "Update fruits for alicefile"
git checkout bob
echo "banane" >> alicefile.txt
git commit -am "Update fruits for alicefile"
```

Effectuez un merge de alice dans bob

### Indice :

Souvenez vous de la gestion de conflits.

# VIII La gestion des branches avec git

## Exercice 1

[solution n°6 p. 14]

Que fait la commande

```
1 git checkout -b ma_branche
```

- **A** Elle crée une branche ma_branche si ma_branche n'existe pas
- **B** Elle écrase la branche ma_branche avec une nouvelle si ma_branche existe
- **C** Elle me déplace vers la branche ma_branche si celle-ci n'existait pas avant
- **D** Elle me déplace vers la branche ma_branche si celle-ci existait déjà avant

## Exercice 2

[solution n°7 p. 15]

Quelles phrases sont vraies

- **A** La commande "git branch -D ma_branche" crée une branche nommée ma_branche
- **B** La commande "git merge alice" fusionne la branche courante avec la branche alice
- **C** La commande "git rebase" est une commande dangereuse
- **D** Si l'on doit résoudre des conflits, c'est parce que l'on a mal géré ses branches
- **E** La gestion arborescente est une spécificité de git

## Exercice 3

[solution n°8 p. 15]

C'est une bonne idée de rebase si...

- **A** Je n'ai pas produit de nouvelles versions depuis longtemps et les autres ont beaucoup avancé
- **B** J'ai effectué des modifications majeures et je dois les incorporer à un projet avant de push
- **C** J'ai effectué des modifications mineures et je dois les incorporer à un projet avant de push

## Exercice 4

[solution n°9 p. 15]

La branche master a vocation à :

**A** Contenir à terme une très large partie des commits

**B** Être fonctionnelle à tout moment

**C** Contenir la version la plus avancée du projet, qu'elle soit stable ou non

**D** servir de base pour les versions publiées d'un projet

# Solutions des exercices

## Solution n°1

[exercice p. 5]

## Solution n°2

[exercice p. 5]

```
1 echo "pomme" >> toto.txt
2 cat toto.txt
3 git commit -am "Add pomme to toto.txt"
4 git checkout alice
5 cat toto.txt
```

## Solution n°3

[exercice p. 7]

```
1 git merge alice
```

## Solution n°4

[exercice p. 11]

```
1 git checkout charlie
2 git rebase alice
```

## Solution n°5

[exercice p. 11]

## Solution n°6

[exercice p. 12]

Que fait la commande

```
1 git checkout -b ma_branche
```

**A** Elle crée une branche ma_branche si ma_branche n'existe pas

**B** Elle écrase la branche ma_branche avec une nouvelle si ma_branche existe

**C** Elle me déplace vers la branche ma_branche si celle-ci n'existait pas avant

**D** Elle me déplace vers la branche ma_branche si celle-ci existait déjà avant

Git checkotu -b créera une branche ma_branche et vous y déplacera. Si elle détecte que la branche existe déjà, elle renvoie une erreur et ne fait rien.

## Solution n°7

[exercice p. 12]

Quelles phrases sont vraies

**A**

La commande "git branch -D ma_branche" crée une branche nommée ma_branche

Cette commande SUPPRIME la branche ma_branche

**B** La commande "git merge alice" fusionne la branche courante avec la branche alice

**C** La commande "git rebase" est une commande dangereuse

**D**

Si l'on doit résoudre des conflits, c'est parce que l'on a mal géré ses branches

Il est tout à fait normal de devoir gérer des conflits quand certaines personnes doivent travailler dans la même partie du code.

**E**

La gestion arborescente est une spécificité de git

Pouvoir gérer des versions en arborescence est une fonctionnalité de base de beaucoup de gestionnaires de version.

## Solution n°8

[exercice p. 12]

C'est une bonne idée de rebase si...

**A**

Je n'ai pas produit de nouvelles versions depuis longtemps et les autres ont beaucoup avancé

**B**

J'ai effectué des modifications majeures et je dois les incorporer à un projet avant de push

**C**

J'ai effectué des modifications mineures et je dois les incorporer à un projet avant de push

## Solution n°9

[exercice p. 13]

La branche master a vocation à :

**A** Contenir à terme une très large partie des commits

**B** Être fonctionnelle à tout moment

**C** Contenir la version la plus avancée du projet, qu'elle soit stable ou non

| | | |
|---|---|---|
| **D** | | |
| ☐ | servir de base pour les versions publiées d'un projet | C'est souvent à partir de la branche master que l'on crée des versions. Si certaines versions nécessitent des correctifs spécifiques, on a pour coutume de créer des branches spécifiques pour appliquer des correctifs à des versions spécifiques. |