
git
Semifir
Adrien Vossough
https://www.semifir.com/

# tags

# Les Tags

**Un tag**

- C'est un alias (un nom) défini par un développeur
- Il pointe vers un commit.
- Il facilite l'identification d'un commit
- Il nomme un moment précis l’état du dépôt.
- Il permet de marquer des numéros de version sur des commits.

**Le numéro de version  du type: x.y.z**

- Le x va marquer une version majeure pour des modifications importantes ou encore induit une incompatibilité avec une version précédente.
- Le y va marquer une version mineure qui ajoute des fonctionnalités tout en conservant une compatibilité avec l’ancien système. (Attention au dépréciation de fonctionnalités)
- Le z va marquer un patch pour corrections de bugs sans ajout de fonctionnalités.

## Deux types de tags

1. Les tags légers stockent le nom du tag sur un commit donné.
2. Les tags annotés stockent davantage d'informations en conservant également la date de création et le créateur du tag.

créer un tag léger en se positionnant sur le commit

```
git tag nom_du_tag

# exemple :
git tag v1.22.91
```

## Tag avec hash du commit et tag annoté

créer un tag léger en spécifiant le hash du le commit

```
git tag v1.22.91 be33a4
```

Création d'un tag **annoté**

```
git tag -a nom_du_tag

# exemple
git tag -a v1.9

# avec message
git tag -a v1.9 - "message du tag"
```

## Lister les tags

```
git tag --list
# ou
git tag -l

# nombre de ligne du tag à afficher
git tag -l -n3

# détail d'un tag
git show v1.8.5
```

## Envoyer les tags sur le serveur et supprimer un tag

Envoie des tags sur le dépôt distant

```
git push origin –tags

# envoie des tags un à un

git push origin nom_du_tag
```

Suppression d'un tag

```
git tag -d nom_du_tag

# pour l'indiquer dans le dépôt

git push origin :refs/tags/nom_du_tag
```

# Branches

# Les branches

**Une branche**

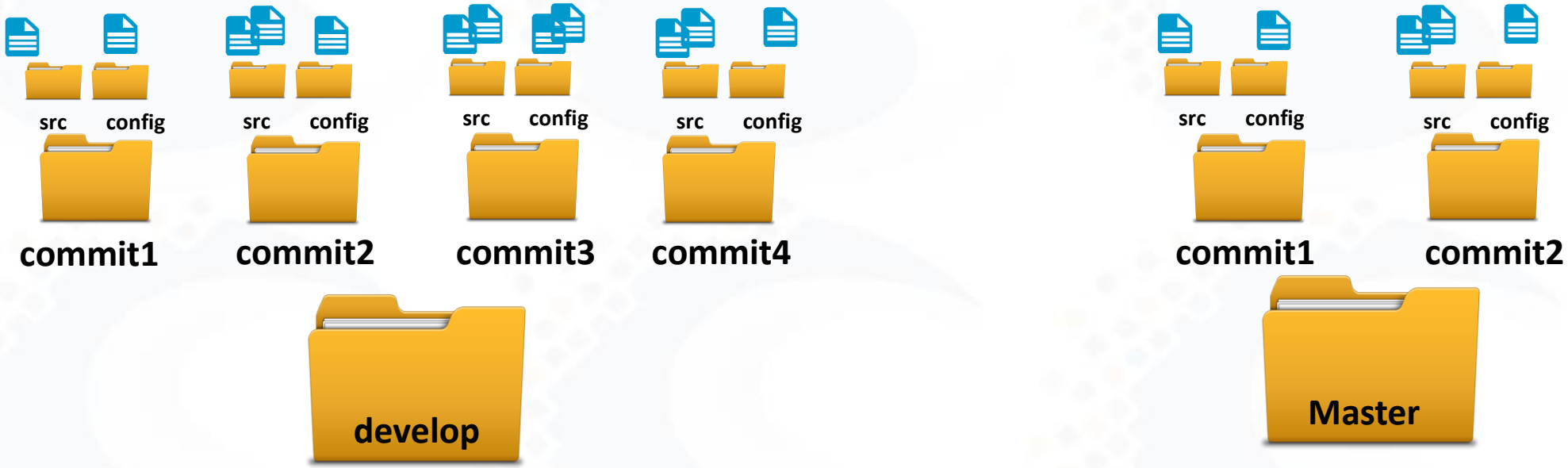
- correspond à une version parallèle de celle en cours de développement.
- peut servir à développer de nouvelles fonctionnalités
- Peut servir à corriger des bugs
- Permet de segmenter différentes versions en cours de développement.

Par défaut, sur un dépôt Git, une branche master est créée. C'est sur cette branche qu'il faut effectuer toutes les manipulations.
Les branches sont des alias dont la référence est dynamique contrairement aux tags qui sont statiques.

# Présentation

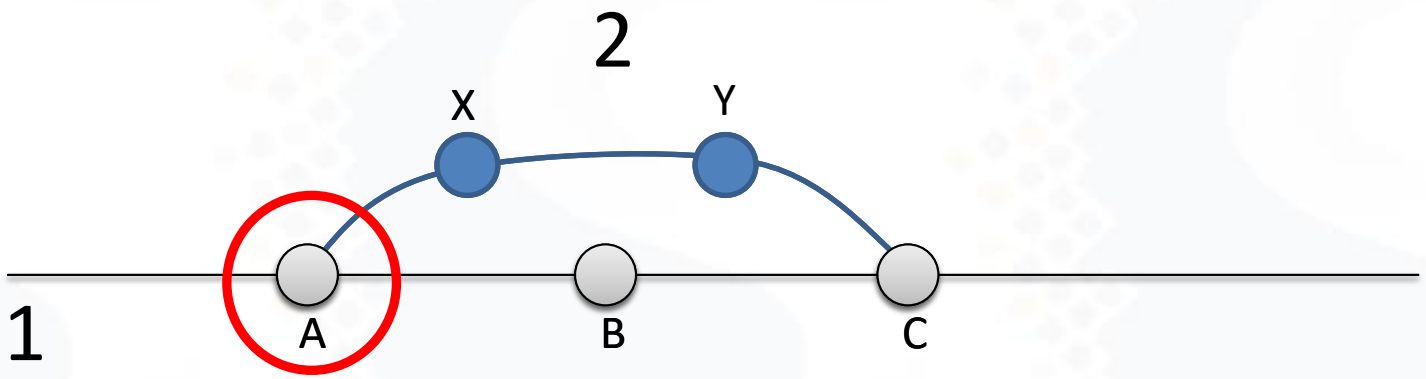
## Comment imaginer les branches ?

Nous pouvons imaginer une branche comme un répertoire.
Dans chacune des branches se trouve une version du projet sauvegardé sous forme d'un "commit"

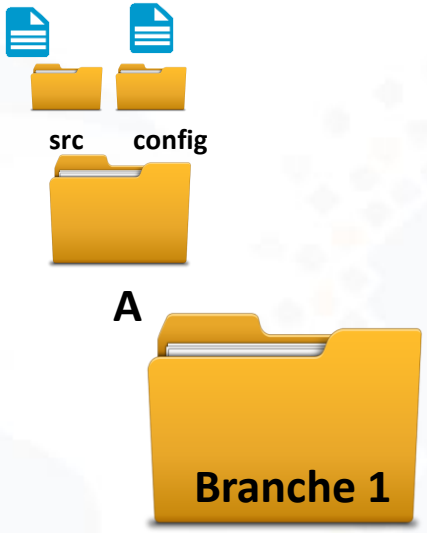


# Présentation

## Comment imaginer les branches ?



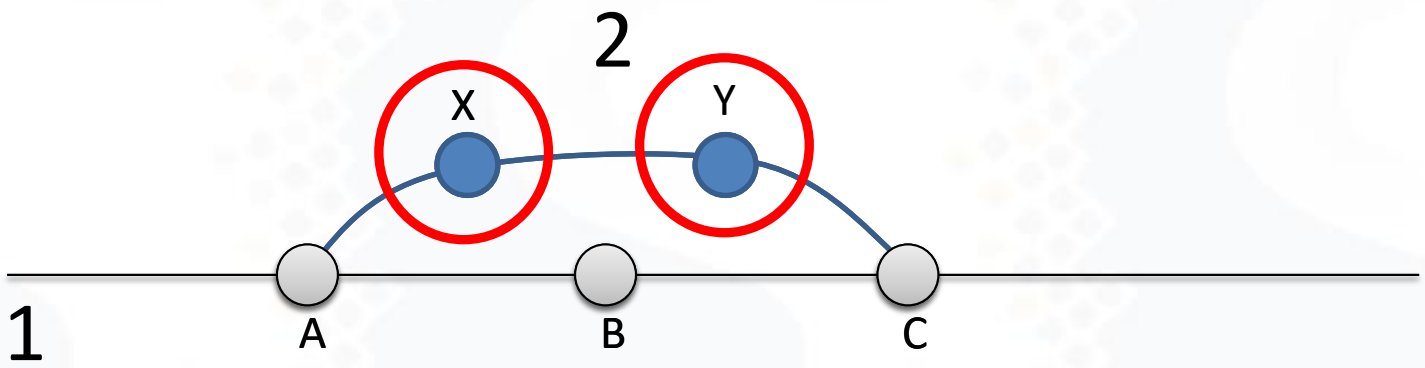

Le développeur a créé une branche, 2 à partir du commit A pour ajouter une nouvelle fonctionnalité.



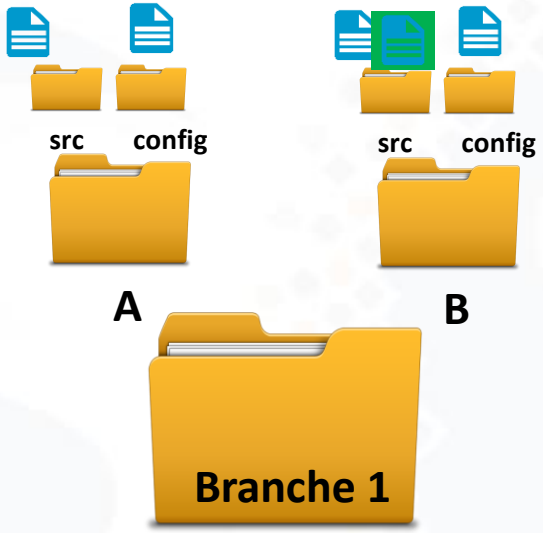

# Présentation

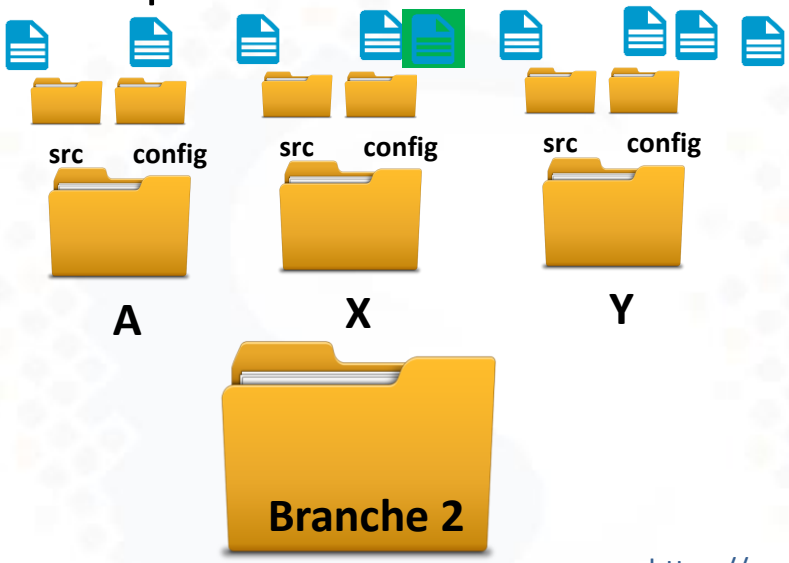
## Comment imaginer les branches ?

ù



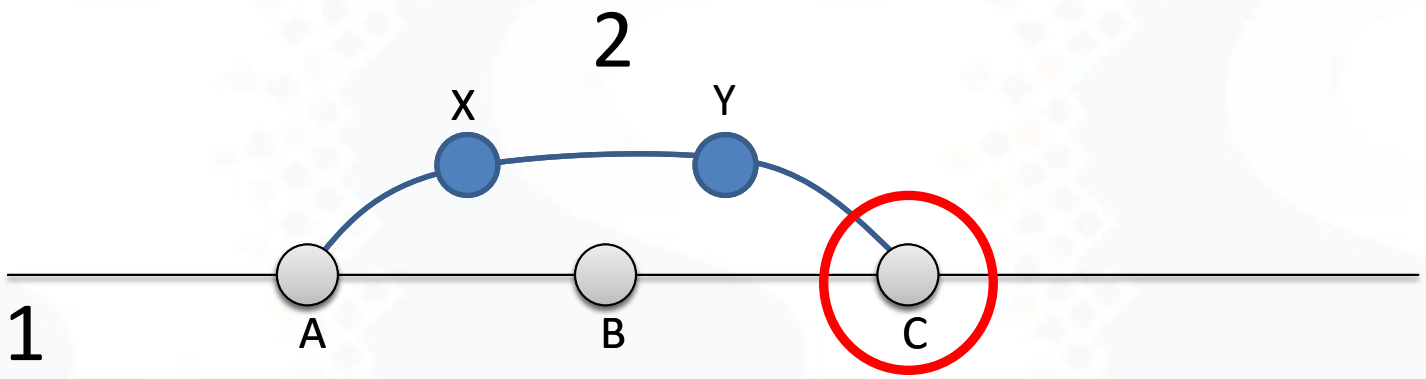

Le deuxième commit X correspond au premier commit après la création d'une branche, le commit Y a suivi. Le développeur fusionne sur la branche principal une fois qu'il a finit ses modifications



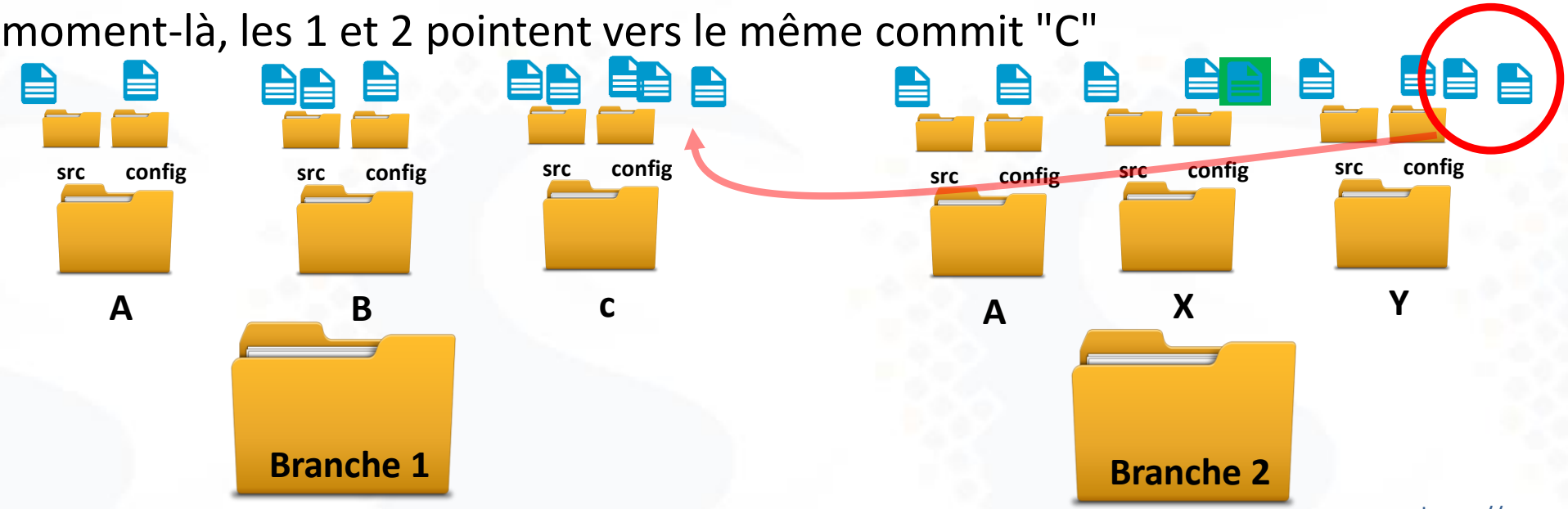



# Présentation

## Comment imaginer les branches ?



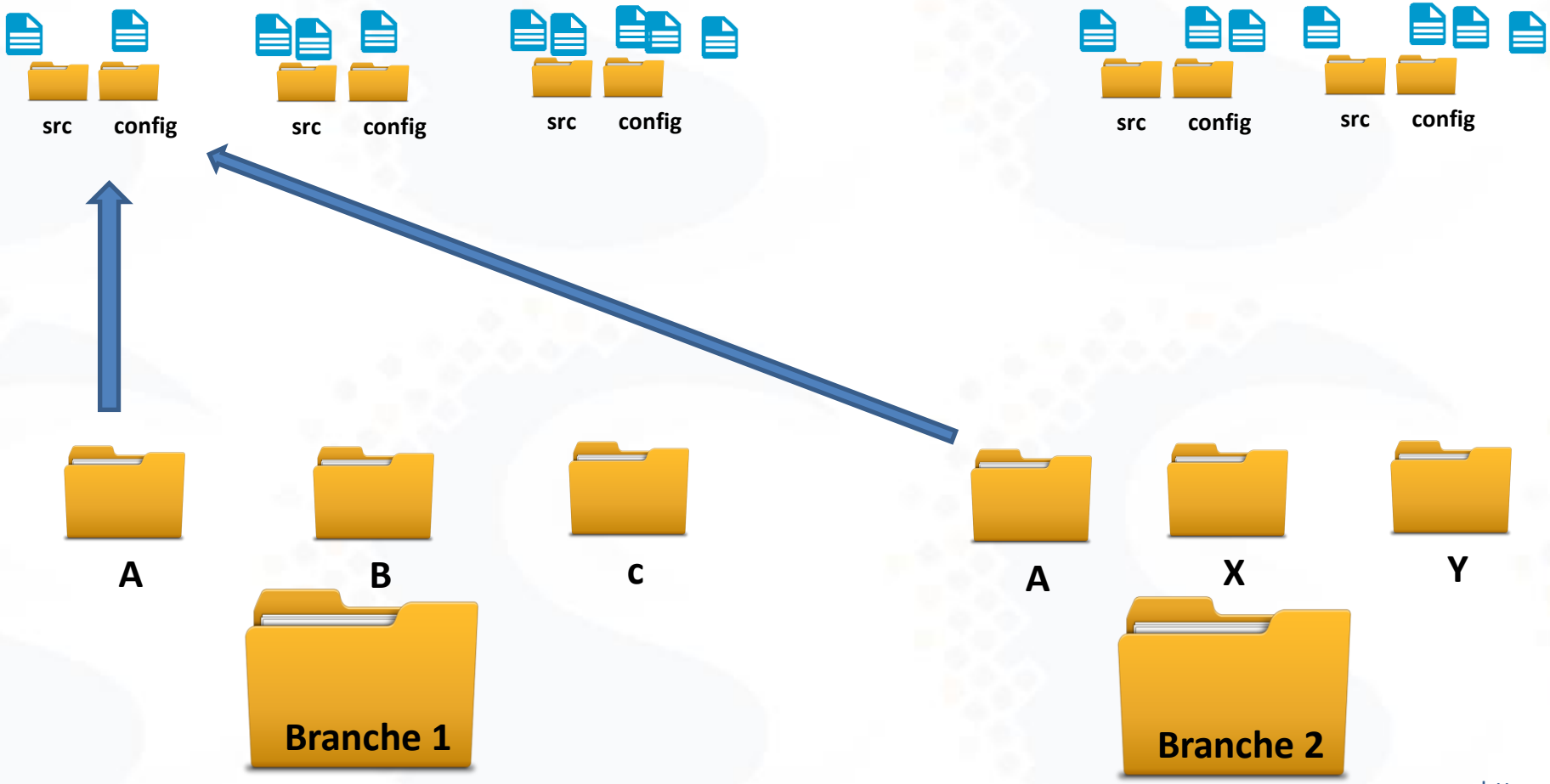

Une fois les modifications intégrées, le développeur va commiter pour enregistrer le résultat de la fusion. À ce moment-là, les 1 et 2 pointent vers le même commit "C"



# Présentation

## Comment imaginer les branches ?

## Dans la réalité, les branches sont des listes de pointeurs vers des objets



# Branches : commandes de base

# Se positionner sur une branche

Les branches permette de conserver et faire évoluer plusieurs versions du même projet.

**Le développeur doit changer de branche pour** :

1. corriger un bug prioritaire sur sa tâche actuelle,
2. travailler sur une nouvelle fonctionnalité (donc sur une autre branche),
3. fusionner pour mettre à jour la version de développement avec une nouvelle fonctionnalité

**Changer de branche (ou se placer sur une branche) signifie :**

1. Le répertoire de travail est modifié.
2. L'index reste intact.
3. La référence HEAD est mise à jour et correspond au commit le plus récent de la branche sélectionnée. Le prochain commit aura pour parent le nouveau HEAD et sera donc considéré comme un commit de la branche sur sélectionnée.

Avant de changer de branche

```
git status
```

# Commandes de base

## Lister / créer / se placer sur une branche

Liste des branches existantes

```
git branch
* Master

liste des branches avec le hash et le message du dernier commit
 git branch -v
```

Création d'une branche. Vérifier d'abored de quelle branche elle va dériver !

```
git branch nom_branche
```

Se placer sur une branche

```
git checkout nom_branche
```

# Commandes de base

## Créer et se placer sur une branche

```
git checkout -b connexion_sociale
```

Avant de changer de branche, il faut s'assurer que la branche actuelle n'a pas de modification à "commiter"

**Le nom d'une branche :**

1. Doit contenir que des caractères ASCII imprimables.
2. Ne doit pas commencer par un tiret.
3. Ne doit pas contenir deux points consécutifs.
4. Ne doit pas se terminer par un slash.

Tester le nom d'une branche

```
git check-ref-format --branch 2.4/fix/forms/35
```

# Commandes de base

## Créer et se placer sur une branche

```
git checkout -b connexion_sociale
```

Avant de changer de branche, il faut s'assurer que la branche actuelle n'a pas de modification à "commiter"

**Le nom d'une branche :**

1. Doit contenir que des caractères ASCII imprimables.
2. Ne doit pas commencer par un tiret.
3. Ne doit pas contenir deux points consécutifs.
4. Ne doit pas se terminer par un slash.

Tester le nom d'une branche

```
git check-ref-format --branch 2.4/fix/forms/35
```

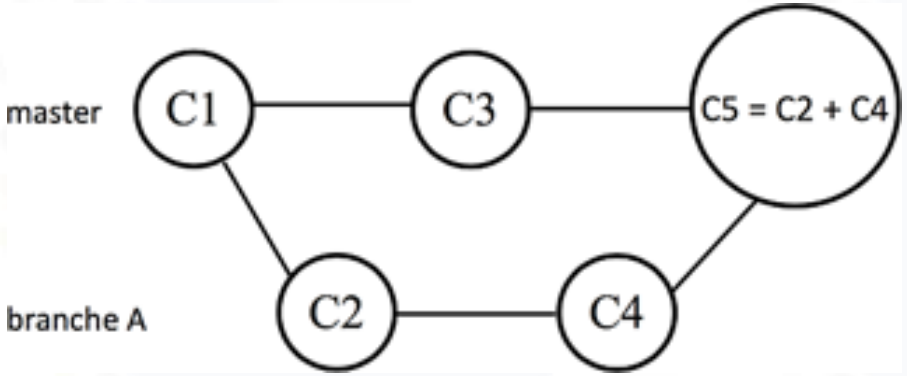
# Fusionner (merge) des branches

Les fusions, appelées merges,  est le fait de regrouper les différences entre une branche vers une autre.

**Exemple** : git pull récupère les commits de la branche distante, Git effectue une fusion avec la branche locale.

On veut fusionner une branche_a vers master :

```
git checkout master
git merge branche_a
```



# Commandes de base

## Fusionner (merge) des branches : fast forward (avance rapide)

1. Ajouter une fonctionnalité en passant par une branche puis la fusionner à Master

2. Git fait un Fast-forward et ne fusionne pas réellement les deux versions, il va avancer le pointeur de la branche master jusqu'à celui de la branche nouvelle_fonctionnalité.

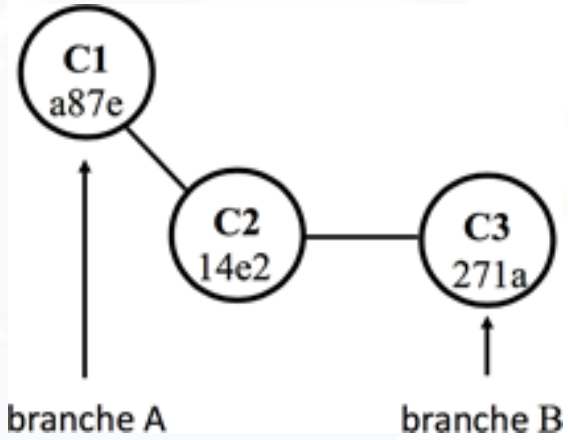
```
git checkout -b nouvelle_fonctionnalité

# …modification du projet…

git commit -am "Ajout de la fonctionnalite"
git checkout master
git merge nouvelle_fonctionnalité

Updating bba9ded..9ddef37
Fast-forward
fichier_principal.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

Git sait qu'en fusionnant les deux versions du projet, il obtiendrait la même version que celle pointée par la branche nouvelle_fonctionnalité.

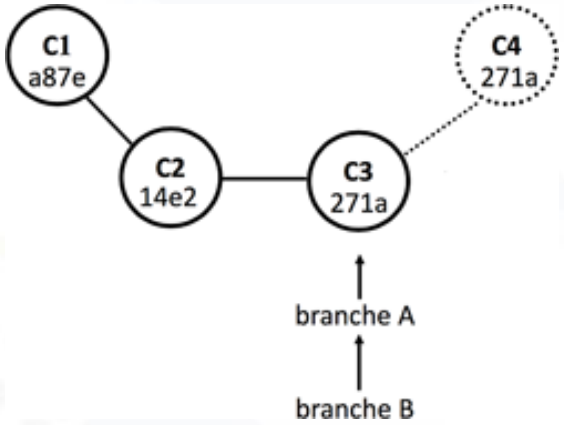
# Commandes de base

## Fusionner (merge) des branches : fast forward (avance rapide)

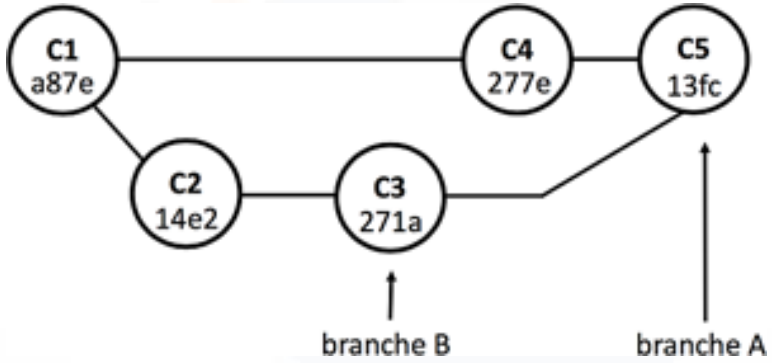
C1 : création de la branche B
C2 : premier commit de B
C3 : dernier commit de B




C4 merge de B vers A.
Comme C4 sera identique à C3, la branche A pointe C3

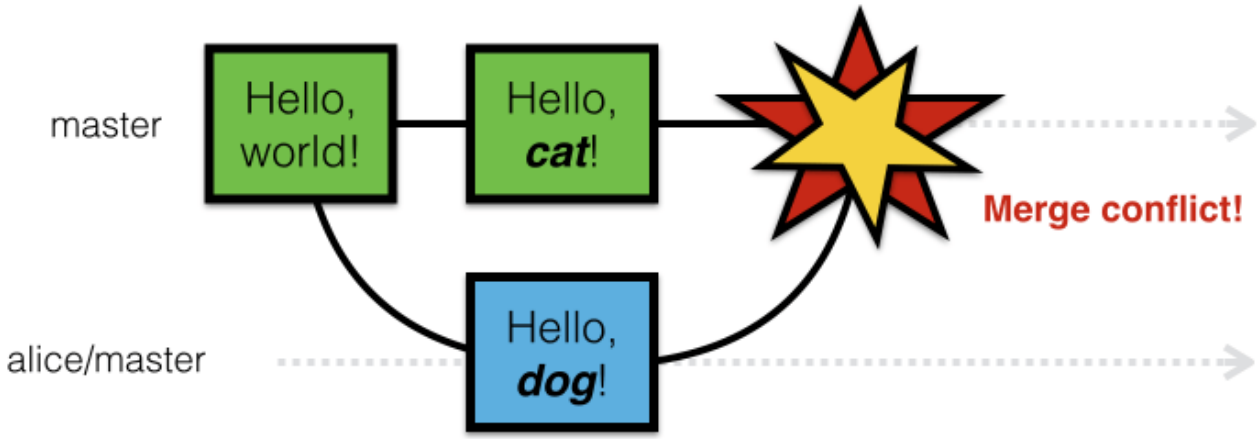



Sans fast-forward :



# Les conflits lors des merges :

Lorsqu'un fichier à évolué sur deux branches, parfois, lors d'un merge, Git ne peut pas appliquer les changements sur le fichier. C'est au développeur de le faire.



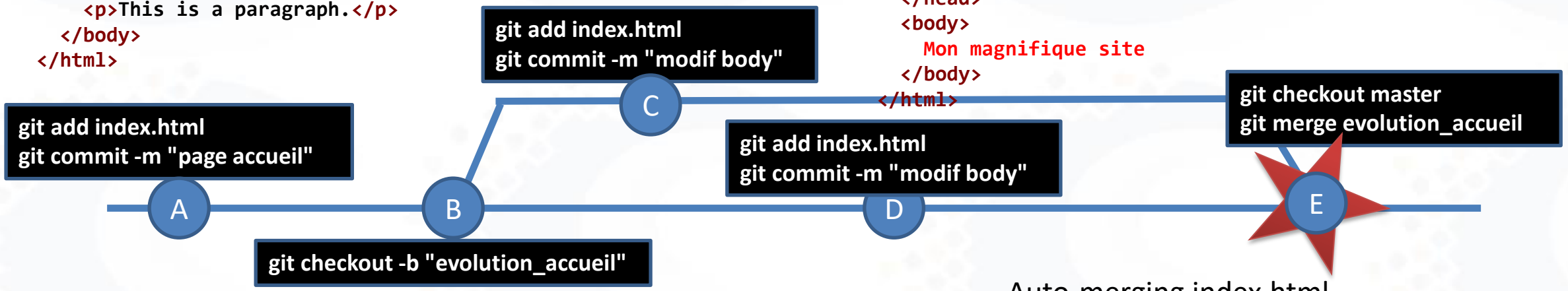
# Commandes de base

## nerges :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    Ajout d'un texte
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    Mon magnifique site
  </body>
</html>
```




Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed;
fix conflicts and then commit the result

# Les conflits lors des merges :

```
# savoir où se trouve le problème
git status

On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:    index.html

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

# Les conflits lors des merges :

<<<< : Début du contenu de la branche de base (ici HEAD, c'est master).
==== : séparation du contenu des deux branches.
>>>> : Fin du contenu de la branche à fusionner (ici evolution_accueil).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
<<<<<<< HEAD
    Mon magnifique site
=======
    Ajout d'un texte
>>>>>>> branch2
  </body>
</html>
```

# Les conflits lors des merges :

Pour gérer le conflit, nous devons échanger avec les développeurs à l'origine du problème et corriger le code.
Le conflit est considéré "réglé" lorsque plus aucun fichier ne comporte de "<<<", "===" et ">>>"

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    Mon magnifique site
  </body>
</html>
```

```
git add index.html

# vérifier s'il reste des problèmes
git status

git commit -m "correction ..."
```

# Supprimer une branche :

**Deux cas pour la suppression d'une branche :**

1. La branche a été fusionnée et aucune autre modification n'est prévue dans cette branche.

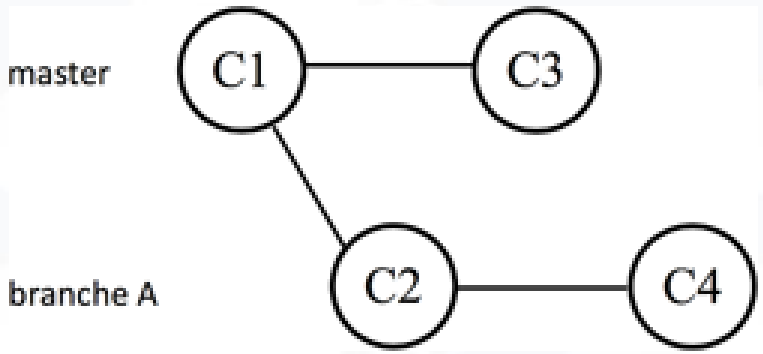
2. Les modifications dans cette branche ne sont plus à l'ordre du jour et elles ne seront jamais intégrées.

```
git branch -d nom_branche
```

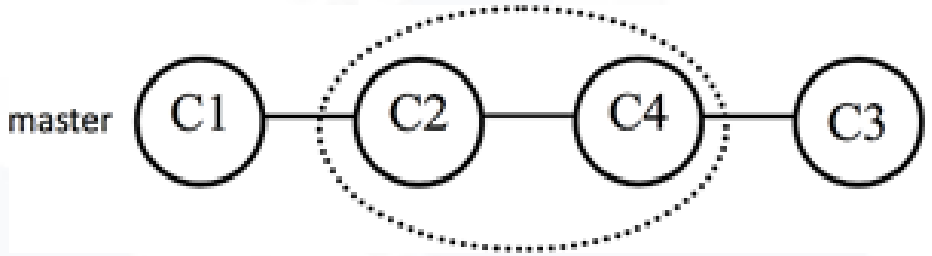
# Rebaser une branche :

**git rebase** permet d'intégrer les modifications d'une branche vers une autre.

1. Un merge a pour but d'intégrer plusieurs commits dans un commit unique représentant tous les commits mergés.
2. Un rebase est un réécriture d'historique




```
git checkout master
git rebase branche_a
```



# Rebaser une branche :

Rebaser la branche A dans la branche master, Git :

1. Identifie le dernier ancêtre commun des deux branches : C1.
2. Met de côté les modifications de la branche master qui n'existent pas dans la branche A.
3. Applique les commits de la branche A (C1 et C2) dans la branche master.
4. Réapplique les commits de master mis de côté.

Les commits de branche A sont ajoutés juste après le dernier ancêtre commun des deux branches, soit avant toutes les modifications à la branche master.

**Ne pas rebaser des commits envoyés sur un dépôt distant.**
**Un rebase va transformer des commits comme s'ils étaient supprimés pour être recréés, ce qui est interdit en distant !**