Module 4
Les tags et les branches

Contenu du module

- Qu'est-ce qu'un tag ?
  - Définition et usage
  - Bonnes pratiques d'utilisation des tags
- Travailler avec les tags
  - Numérotation des versions
  - Différents types de tags
  - Créer, lister, supprimer des tags
- Qu'est-ce qu'une branche ?
  - Principes et organisation
  - La branche « master »
  - Bonnes pratiques d'utilisation des branches
- Gestion des branches
  - Travailler avec les branches
  - Créer et fusionner des branches
- Les branches distantes

Qu'est-ce qu'un tag ?

- Alias/Nom défini par un développeur.
  - Il pointe vers un commit pour l'identifier facilement.
- Les tags sont utilisés pour nommer à des moments précis l'état du dépôt.
  - Ils permettent d'éviter l'utilisation des hashs SHA-1 qui ne sont pas explicites et difficiles à retenir.
- Les tags sont notamment utilisés pour marquer des numéros de version sur des commits.

Bonnes pratiques d'utilisation des tags

- Il est essentiel d'utiliser les tags à bon escient !
- Ils doivent correspondre à un état de mémoire significatif.
  - Version stabilisée à livrer, identification d'une fonctionnalité terminée, numéro de patch, …
- Une nomenclature de nommage doit être définie pour un projet.

Numérotation des versions

- Plusieurs méthodes …
  - Pas d'idéal ! Il faut juste rester cohérent !
- Le « versionning sémantique » est courant.
  - Le numéro de version se construit à partir de trois nombres séparés de points : **x.y.z**
  - **x** : version majeure.
    - Modifications importantes dans le fonctionnement de l'application (incompatibilités, …).
  - **y** : version mineure.
    - Ajoute des fonctionnalités tout en conservant une compatibilité avec l'ancien système.
  - **z** : patch.
    - Corrections de bugs sans ajout de fonctionnalités.

Les différents types de tags

- 2 types de tags :
  - Tags légers.
    - Ils stockent le nom du tag sur un commit donné.
  - Tags annotés.
    - Ils stockent le nom de la personne qui créé le tag (utilisateur Git).
    - Ils stockent également un message informatif sur le tag.
      - Informations relatives à la version, …

Création d'un tag

- Tag léger :
  - Option 1 :
    - Se positionner sur le commit :
      - `git checkout <numéro de commit>`
    - Créer le tag :
      - `git tag <nom du tag>`
  - Option 2 :
    - Créer le tag sans se positionner sur le commit :
      - `git tag <nom du tag> <numéro de commit>`
- Tag annoté :
  - Ajouter l'option –a à la commande git tag !
  - La saisie d'un message est nécessaire !
    - Soit de manière interactive
    - Soit en ajoutant l'option `–m "message"`

Lister les tags et leurs informations

- `git tag --list`
  - L'option –nx permet d'afficher les x lignes du message d'un tag nommé.
    - Par défaut x vaut 1
- `git show <nom du tag>`
  - Pour un tag léger :
    - Affiche les détails du commit auquel il est attaché.
  - Pour un tag annoté :
    - Affiche les détails du tag puis les détails du commit.

Supprimer un tag

- Il s'agit simplement de supprimer l'alias positionné sur le commit, et non pas le commit en tant que tel !
  - Supprimer un tag n'entraîne donc aucune perte dans l'historique du projet.
- Commande :
  - `git tag -d <nom du tag>`

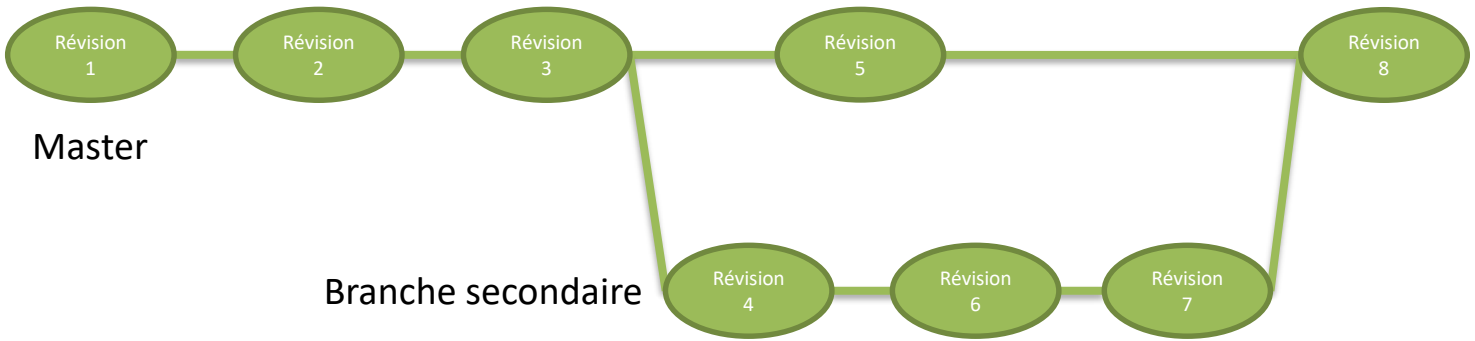
Qu'est ce qu'une branche ?

- Une branche est un simple pointeur vers un commit particulier.
- Tout dépôt Git contient une branche principale : `master`.
- La branche courante est nommée `HEAD`.

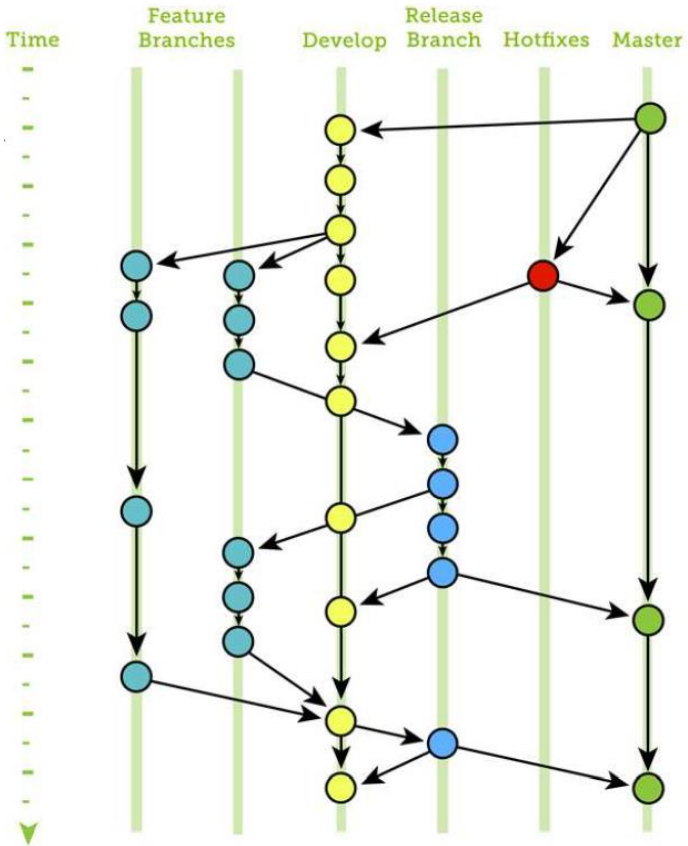
- Utilité :
  - Idée : Une fonctionnalité par branche.
  - Exemple :
    - Branche de développement (Dev), de maintenance, d'évolutions de fonctionnalités,...

Organisation des branches

- Les branches permettent de travailler sur une copie de l'historique sans risquer de modifier un code déjà testé et validé.
- Les modifications pourront être ensuite reportées sur une autre branche.
  - Notion de « merge ».



Exemple d'organisation



La branche `MASTER`

- La branche `master` n'est pas, d'un point de vue technique, une branche spéciale.
  - Elle est identique à toutes les autres branches !
- Chaque dépôt en a une car elle est créée par la commande git init.
  - Elle pourrait être renommée…
- Mais conventionnellement :
  - C'est LA branche de collaboration !

Bonnes pratiques d'utilisation des branches

- Pour chaque projet, il est important de définir :
  - Le rôle de chaque branche.
  - Une nomenclature de nommage.
  - Des responsabilités de gestion de branches.
  - Le périmètre de chacune des branches.
- On ne devrait jamais faire de modifications directement sur la branche MASTER !
  - C'est la branche ou les développeurs viendront ajouter leur travail issu d'autres branches.
  - Elle sert de branche de collaboration.

Travailler avec les branches

- Les commandes `git branch` et `git checkout` permettent de travailler avec les branches.
- Lister les branches existantes :
  - `git branch`
    - La branche courante est indiquée avec *
- Créer une branche :
  - `git branch <nom de la branche>`
- Changer de branche :
  - `git checkout <nom de la branche>`
- Créer une branche et se positionner dessus :
  - `git checkout -b <nom de la branche>`
- Renommer la branche courante :
  - `git branch -m <nouveau nom>`
- Renommer une branche :
  - `git branch -m <ancien nom> <nouveau nom>`

Changer de branche

- Pourquoi ?
  - Corriger un bug prioritaire sur sa tâche actuelle,
  - Continuer à travailler sur une nouvelle fonctionnalité (donc sur une autre branche),
  - Fusionner deux branches pour mettre à jour la version de développement avec une nouvelle fonctionnalité par exemple.
- Effets du changement de branche
  - Le répertoire de travail est modifié.
  - L'index reste intact.
  - La référence HEAD est mise à jour et correspond au commit le plus récent de la branche sur laquelle le développeur s'est placé. Le prochain commit aura pour commit parent le nouveau HEAD et sera donc considéré comme un commit de la branche sur laquelle le développeur s'est placé.
- Avant de changer de branche, il faut veiller à n'avoir aucune modification en cours dans le répertoire de travail et dans l'index !
  - Il est possible de vérifier les modifications en attente avec la commande `git status`.

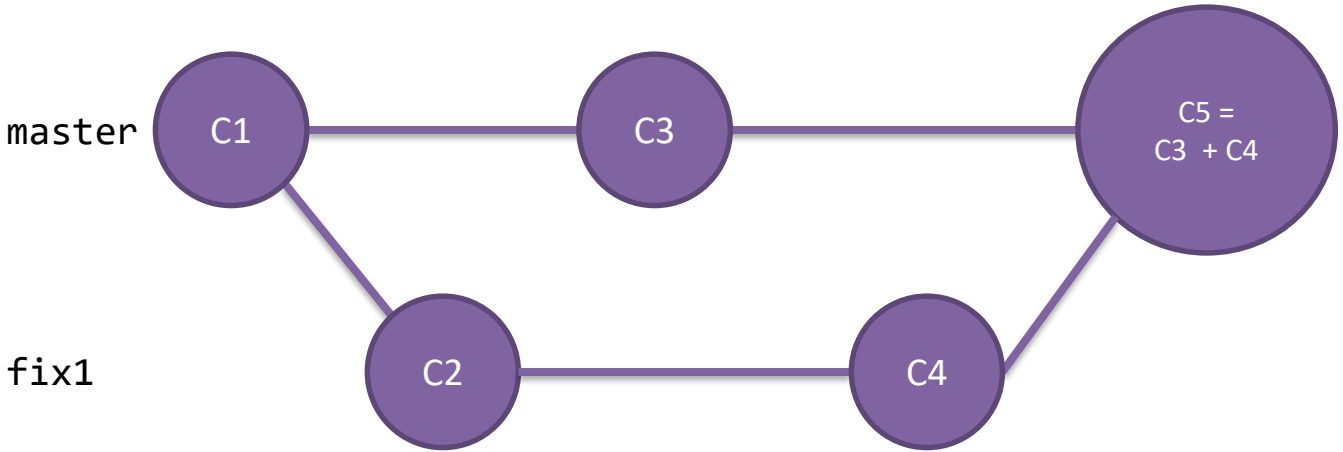
Fusionner des branches

- Intégrer les modifications d'une branche dans une autre.
  - « merge »
- Se poser la question : dans quelle branche dois-je intégrer les modifications de l'autre branche ?
- Avant de fusionner :
  - Ne pas avoir de modifications en cours ! (Tous les commits ont été faits.)
- Exemple :
  - On souhaite intégrer les modifications de la branche `fix1` dans la branche `master`.
    - Se positionner sur la branche cible (`master`)
      - `git checkout master`
    - Fusionner la branche `fix1` dans `master`
      - `git merge fix1`

Ce que fait la fusion…

- Recherche du dernier commit que les deux branches ont en commun.
  - « ancêtre commun »
- Création d'un nouveau commit
- Y apporter les modifications de la branche fusionné.
  - Ce commit est particulier car il hérite de deux parents : commit de « merge ».
  - Ces étapes sont totalement transparentes pour le développeur.

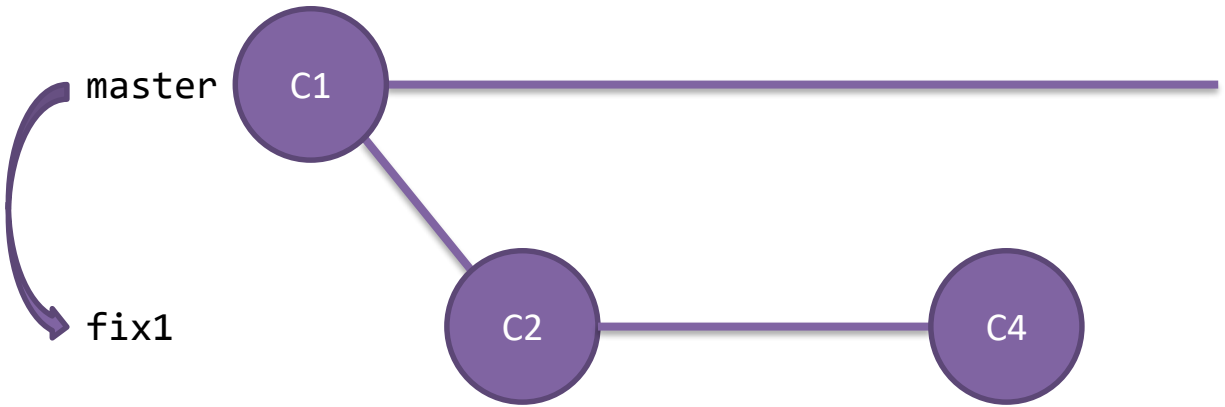
Exemple : commit de « merge »



- Création d'une branche nommée `fix1` à partir du commit C1 de la branche `master`.
  - C2 est le premier commit de `fix1`.
  - La vie de la branche principale (`master`) continue (commit C3).
- C4 représente le deuxième et dernier commit de `fix1`.
  - C'est le commit pointé par la référence `fix1`.
- C5 représente le commit de merge.
  - C'est le commit qui va réunir les modifications de `fix1` pour les intégrer dans la branche `master`.

L'avance rapide

- L'avance rapide est un système interne à Git qui lui permet de gagner en performances lorsque la branche cible de la fusion n'a pas subit de modification depuis la création de la branche à fusionner.
- Dans ce cas, aucun commit de « merge » n'est créé.
  - Le pointeur de la branche `master` est déplacé sur C4

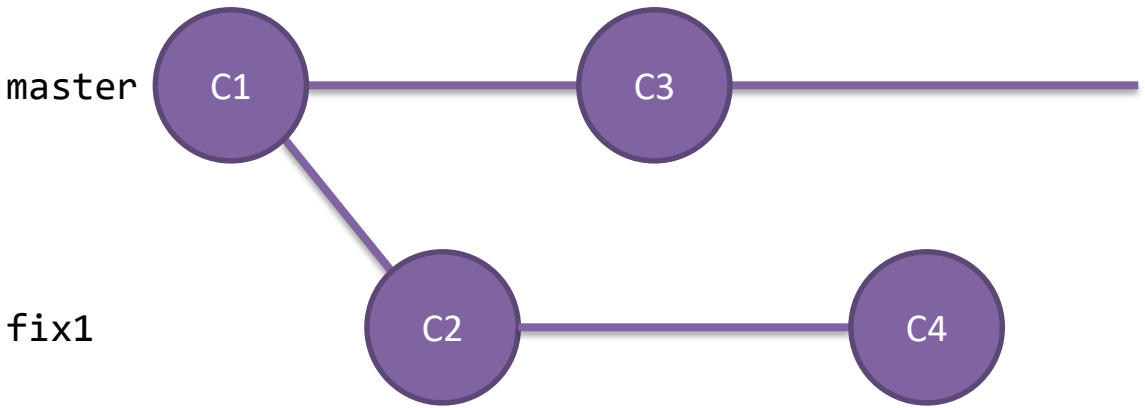


Supprimer une branche

- Cas qui nécessitent la suppression d'une branche :
  - La branche a été fusionnée dans master et aucune autre modification n'est prévue dans cette branche.
  - Les modifications dans cette branche ne sont plus à l'ordre du jour et elles ne seront jamais intégrées.
- Attention, il ne faut supprimer une branche que lorsqu'on est sûr et certain que les modifications qu'elle apporte n'ont et n'auront plus aucune valeur.
- Commande :
  - `git branch –d <nom de branche>`

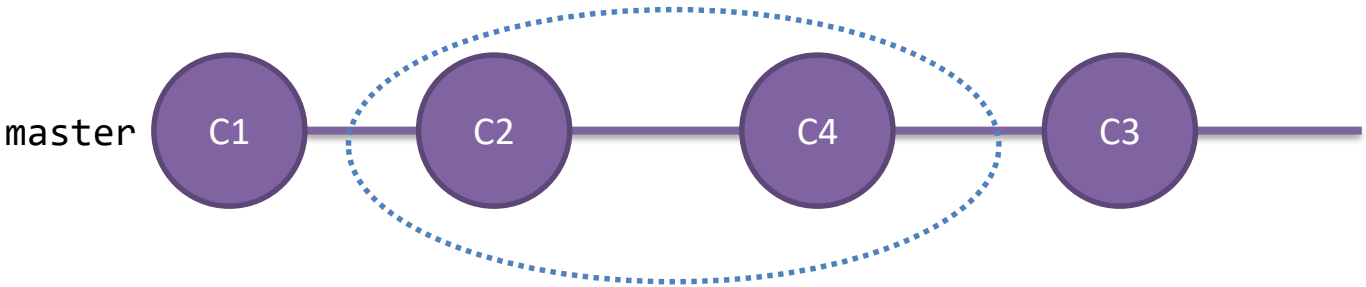
« Rebaser »

- Une approche pour intégrer les modifications d'une branche dans une autre !
  - Cela correspond à une réécriture d'historique !
- Au lieu d'intégrer toutes les modifications d'une branche dans un unique commit, il est possible de modifier la base d'une branche en incluant les commits d'une autre branche.
- Situation avant le rebase :



git rebase

- Opérations pour rebaser :
  - `git checkout master`
  - `git rebase fix1`
- Situation après le rebase :



Les conflits de fusion

- S'il y a eu des changements apportés aux 2 branches sur les mêmes lignes, il sera alors impossible de réaliser la fusion sans corriger le conflit au préalable.
- Corriger le(s) fichier(s) en conflit.
  - Penser à retirer les lignes <<<<, ==== et >>>>
- Une fois le conflit résolu, utiliser la commande `git add`.
- Terminer la fusion avec un `git commit`.

Travaux Pratiques

Travaux pratiques

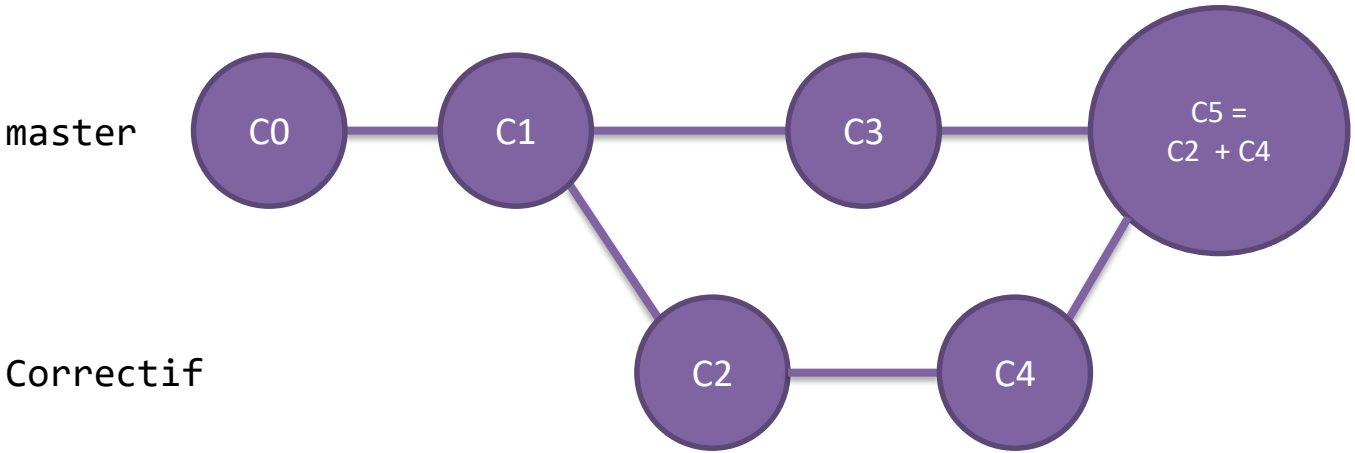
- Exercice 5 : Créer une version
  - Sur le dépôt it-git, validez tous les changements en cours vers le dépôt.
  - Créez une version/tag nommé : `v1.0.0`
    - Décrivez le contenu du projet dans le message du tag.
  - Ajoutez un fichier `webservers.xml`.
  - Y ajouter le contenu suivant :

```
<webservers>
    <server>Apache</server>
    <server>Nginx</server>
    <server>IIS</server>
</webservers>
```

  - Ajoutez le fichier au dépôt.
  - Créez une nouvelle version version/tag : v1.1.0 en utilisant un message décrivant le contenu du projet.
  - Visualisez les différents tags.

Travaux pratiques

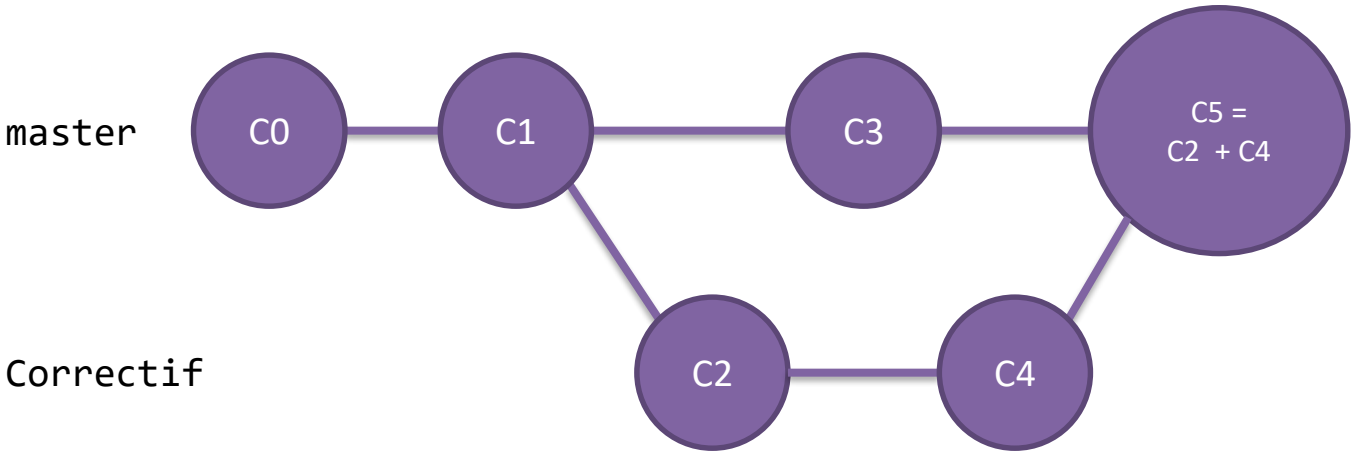
- Exercice 6 : Les branches



  - Créez et initialisez un nouveau dépôt `branches1`.
  - Utiliser les commandes appropriées pour créer la structure du schéma ci-dessus.
    - Chaque commit est associé à un nouveau fichier (C1, C2, ...).
    - Utilisez la commande git log et ses options appropriées pour visualiser chaque étape.

Travaux pratiques

- Exercice 7 : Les branches et les conflits



  - Créez et initialisez un nouveau dépôt `branches2`.
  - Utiliser les commandes appropriées pour créer la structure du schéma ci-dessus en modifiant un fichier unique à chaque commit.
    - Chaque commit est associé à une nouvelle ligne dans le fichier.
    - Utilisez la commande git log et ses options appropriées pour visualiser chaque étape.

Fin du module

- Avez-vous des questions