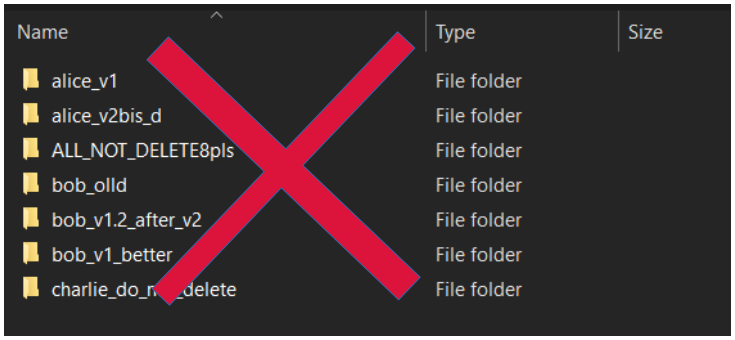
Club Informatique

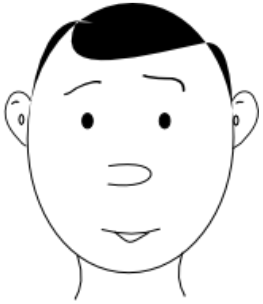
How to Git Gud

Par Arthur Desuert et Rémi Cellard

Contexte

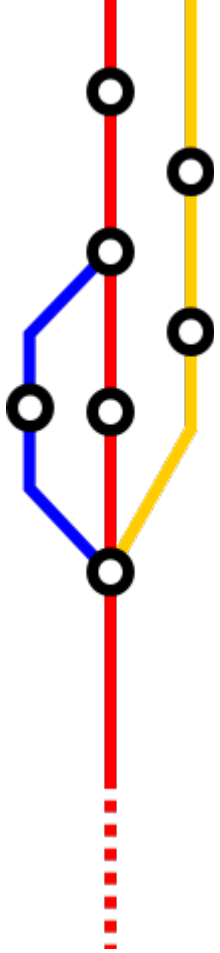
Alice



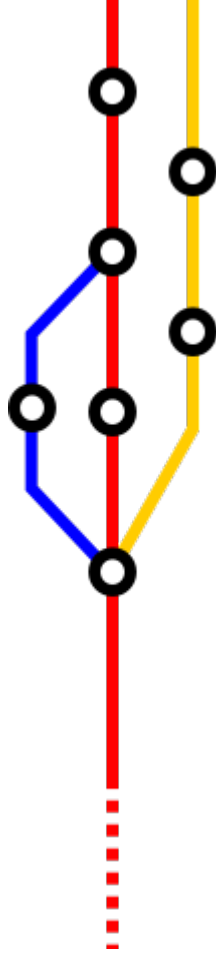


Bob

Charlie

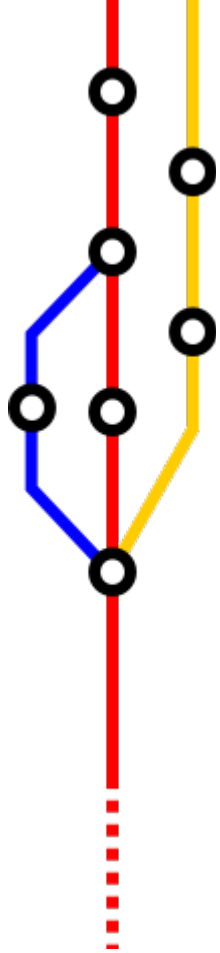


Sommaire



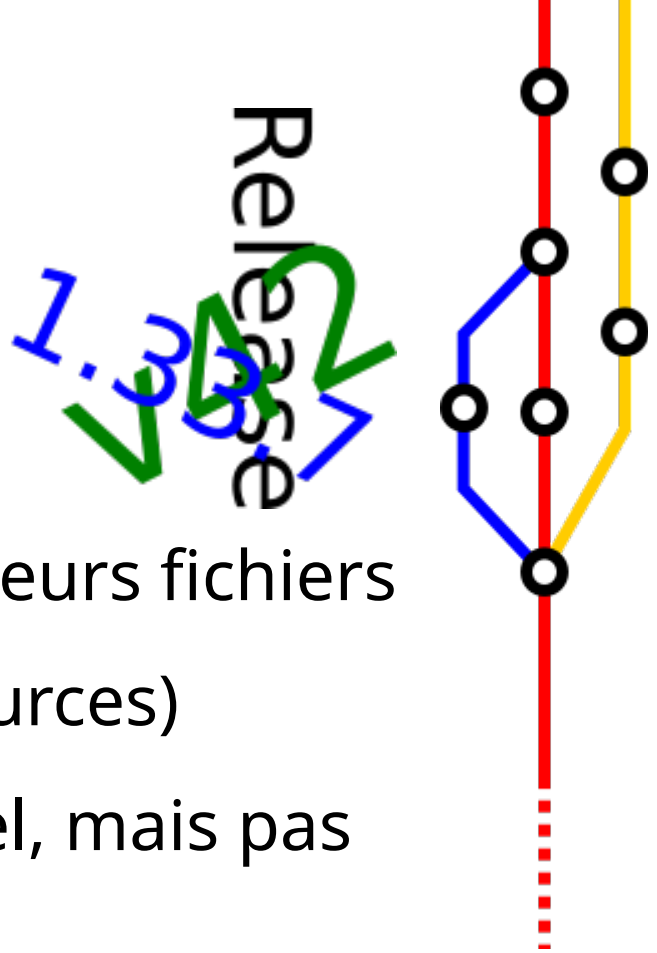
- Introduction
- Bases de Git
- Les branches
- Travail d’équipe avec Git

Introduction



Introduction

Gestion de versions



La gestion de versions, c'est quoi ?

- Sauvegarder l'évolution d'un ou plusieurs fichiers
- Principalement des fichiers texte (sources)
- Très utilisé en développement logiciel, mais pas que !
  - Scripts
  - Cahiers de laboratoire
  - Etc.

Introduction
Git

Le gestionnaire de versions Git :

- Créé en 2005 par Linus Torvalds
- Gratuit et open-source
- Assez répandu et apprécié

Version Control

All Respondents | Professional Developers

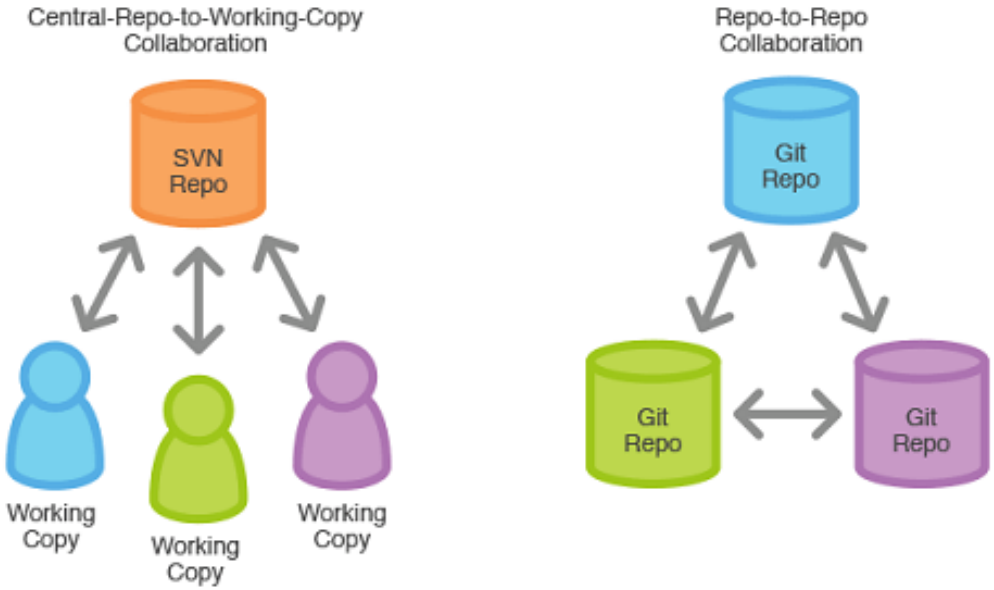
Git	87.2%
Subversion	16.1%

(cf https://insights.stackoverflow.com/survey/2018)

Introduction

Pourquoi Git ?

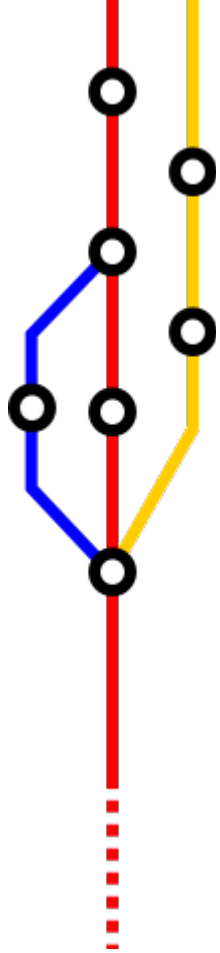
Modèle distribué (≠ Subversion)



→ Travail hors ligne possible !

Les bases de Git

Installation du client Git



Pour Linux (adapter selon votre distribution) :

 # apt install (-y) git | # dnf install git
 # pacman -S git

Pour Windows :

http://git-scm.com/download/win

Pour Mac :

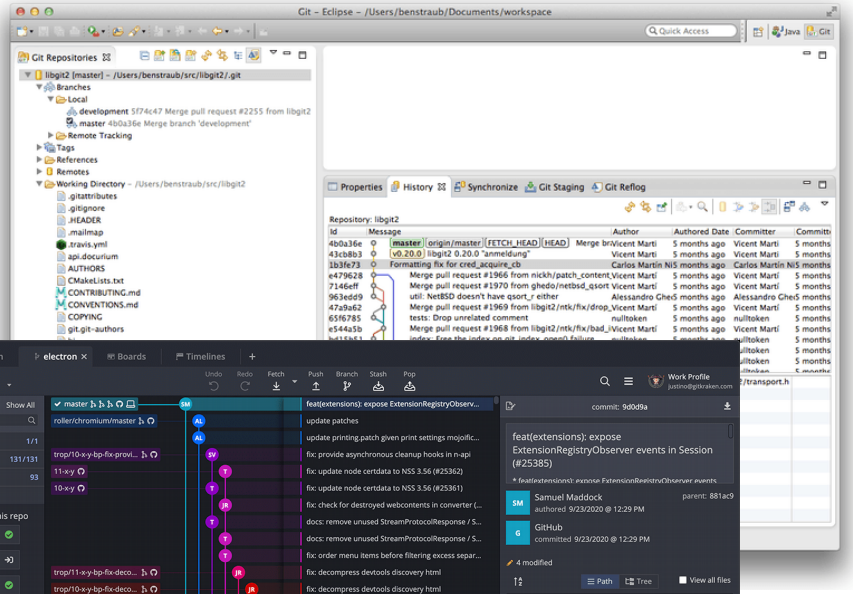
http://git-scm.com/download/mac

Les bases de Git

Clients Git graphiques

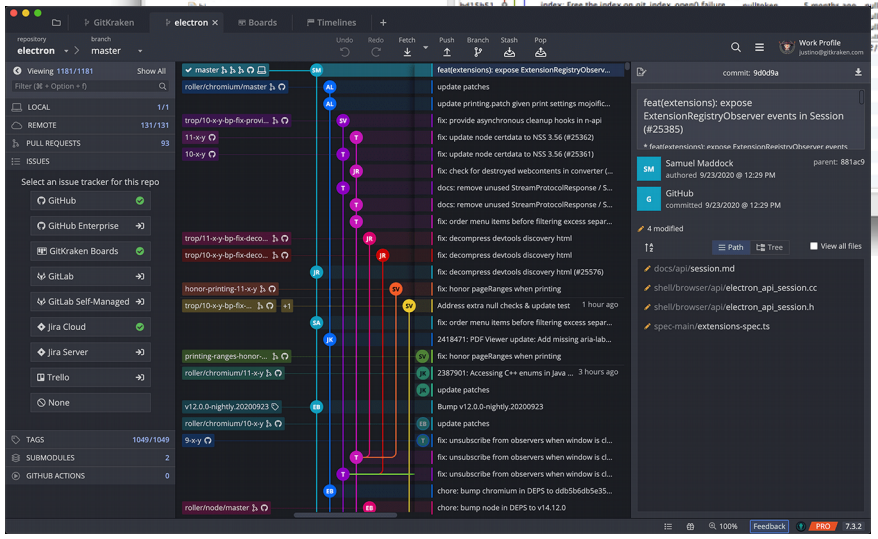
Intégré à un environnement (IDE) :

- Eclipse
- Matlab



Ou bien *standalone* :

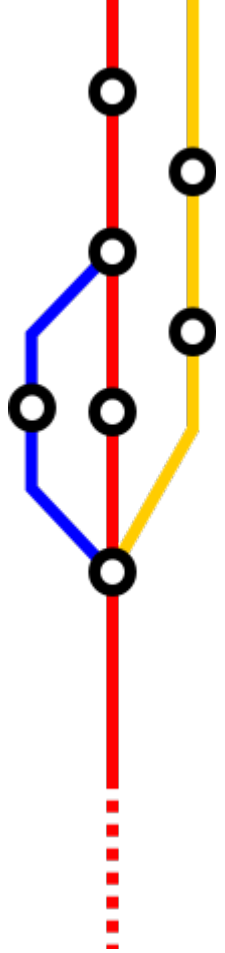
- git GUI
- GitKraken



Club Informatique

Les bases de Git

Création d'un utilisateur



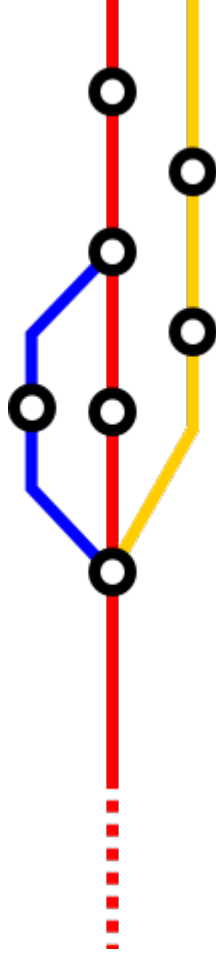
Déclarer son 'identité' à Git :

$ git config (--global) user.name "<pseudo_git>"

$ git config (--global) user.email "<email_git>"

/!\ Ces infos sont incluses dans les commits /!

Le premier dépôt (local)



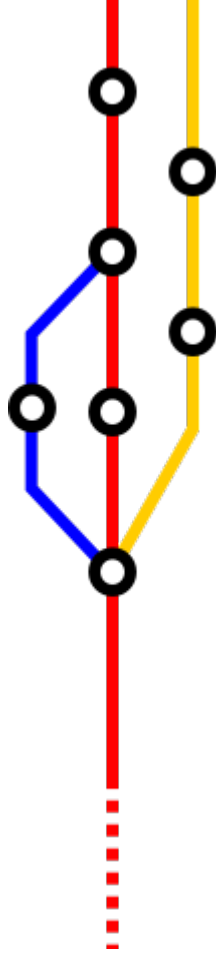
Au commencement, il y avait :

$ git init

Init. d'un dépôt dans le répertoire courant

→ Création d'un dossier .git

Fichiers de départ



Le **README** :

- Permet d'expliquer le contenu du dépôt, donner les directives de config, etc.
- Peut être rédigé en texte simple, Markdown,...

Le **.gitignore** :

- Permet d'ignorer automatiquement certains fichiers
- Très utile pour ne pas polluer son dépôt
- http://gitignore.io

Les opérations de base

Puis vinrent les premiers fichiers :

```
user@deb-dev:/tmp/session_git$ ls
file1  file2
```

Pour voir ce qu'en dit Git :

$ git status

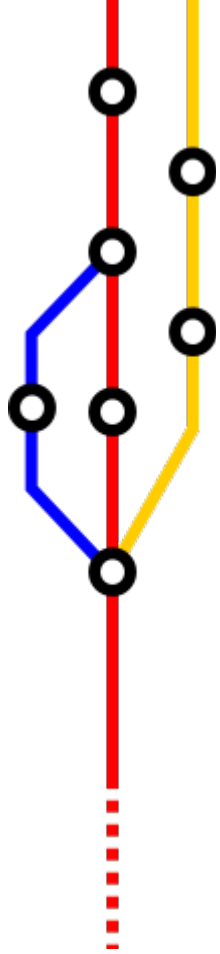
```
user@deb-dev:/tmp/session_git$ git status
Sur la branche master

Aucun commit

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        file1
        file2
```

Les opérations de base



2 étapes clés du 'workflow' Git :

1) Sélection des modifications :

$ git add <file1> <file2> ... <fileN>

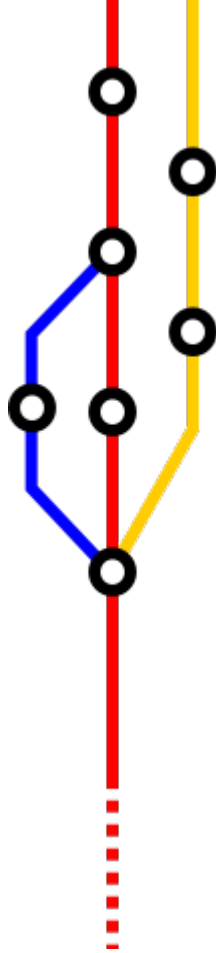
2) Validation de la sélection (le 'commit')

$ git commit (-m <message>)

Étape intermédiaire : vérifier le contenu de la sélection

$ git status

Les opérations de base



Quelques commandes utiles :

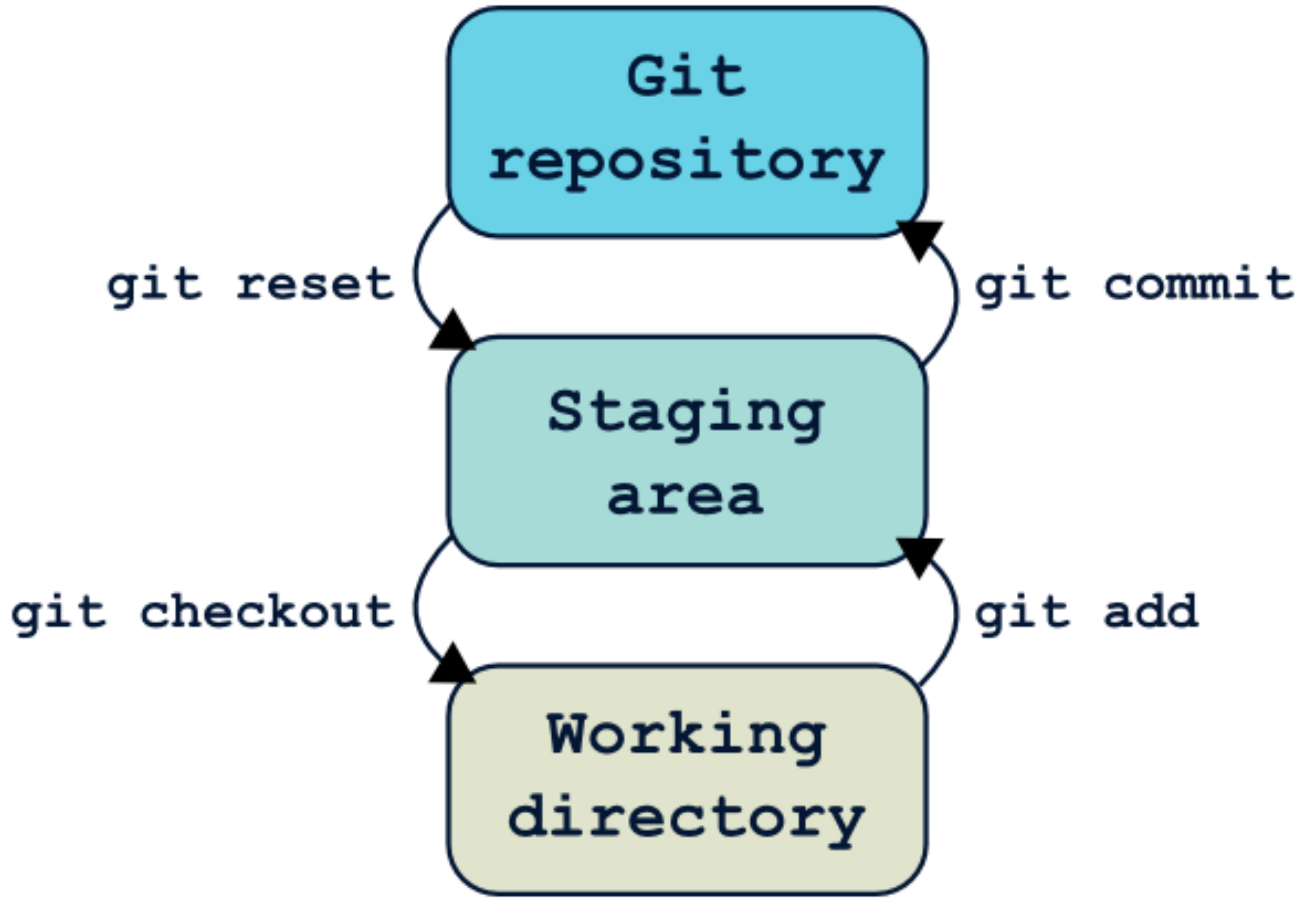
- Visualiser l'historique :

$ git log (<file>)

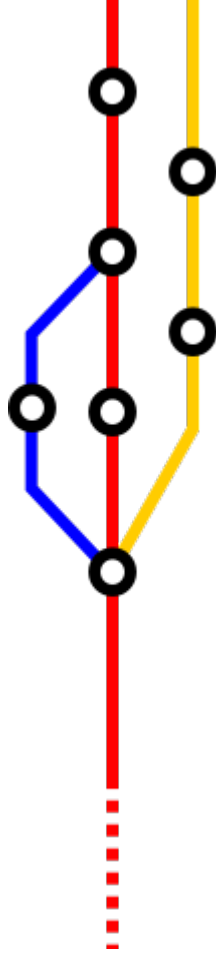
- Voir les différences :

$ git diff (<hash_c1> <hash_c2>)

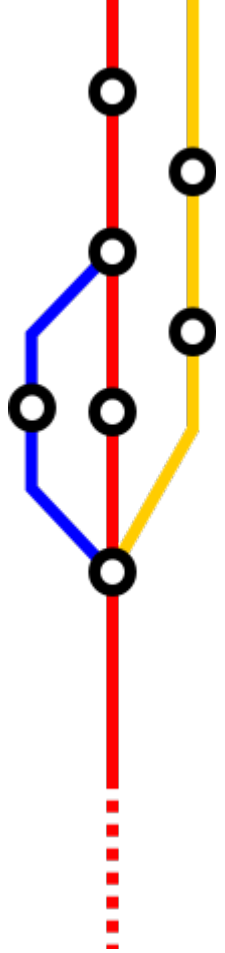
Les bases de Git
Synthèse



Les branches



Les branches
Création



Pour créer une branche (et se placer dessus) :

$ git checkout -b <nom_branche>

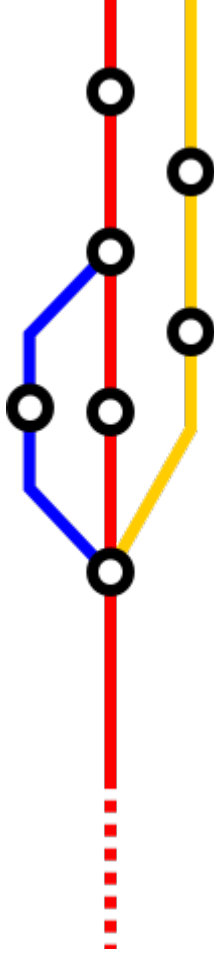
Pour lister les branches existantes :

$ git branch

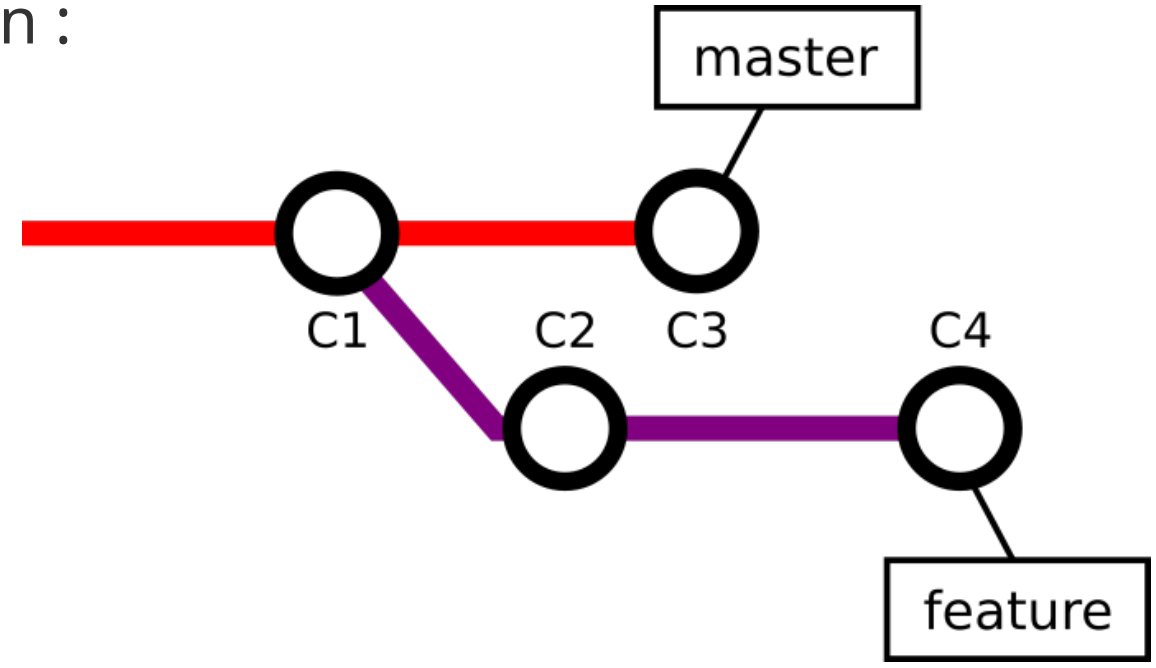
Pour passer d'une branche à l'autre :

$ git checkout <nom_branche>

Les branches
Fusion

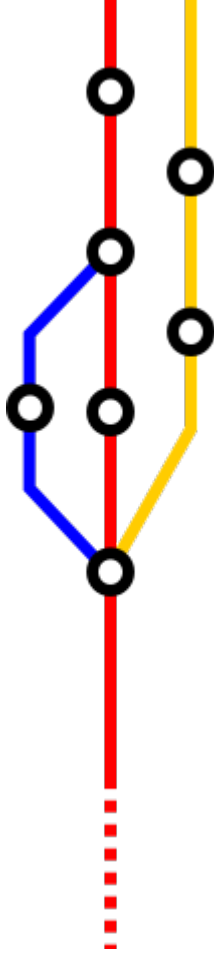


- Situation :



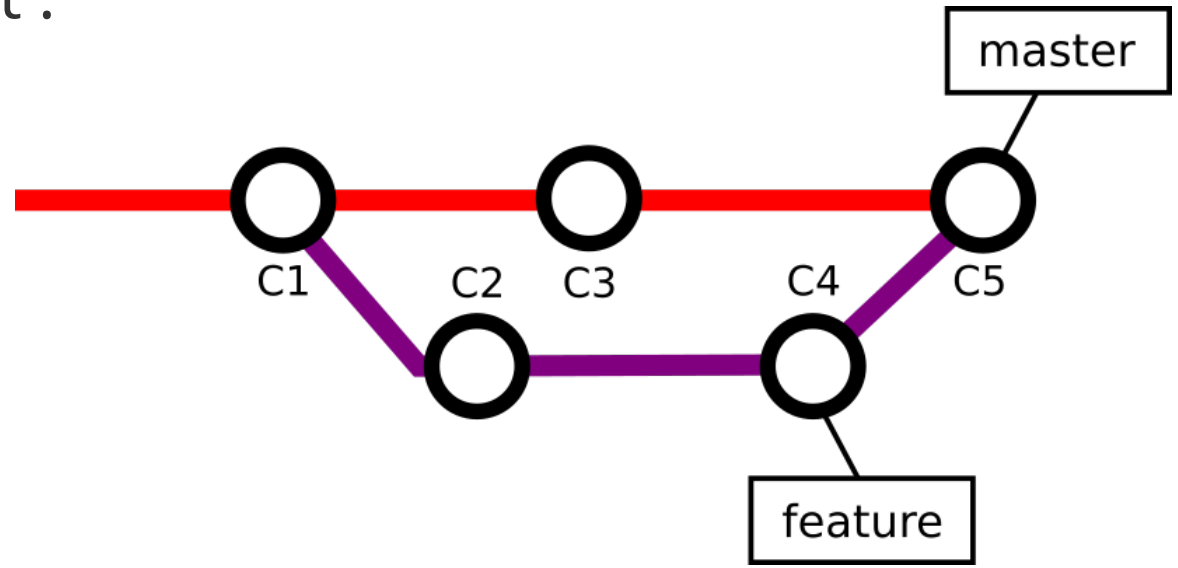
→ Pour mettre à jour la branche *master*, 2 solutions

Les branches
Fusion - merge



$ git checkout master && $ git merge feature

- Résultat :

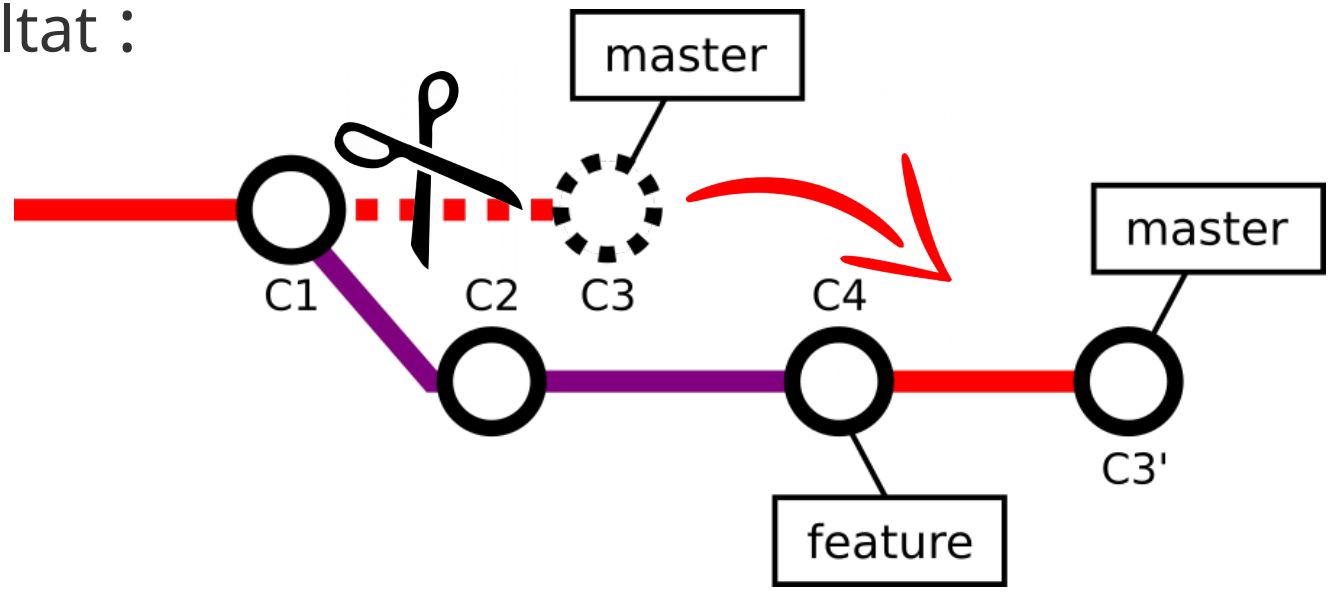


→ Historique préservé mais possible sac de nœuds !

Les branches
Fusion - rebase

$ git checkout master && $ git rebase feature

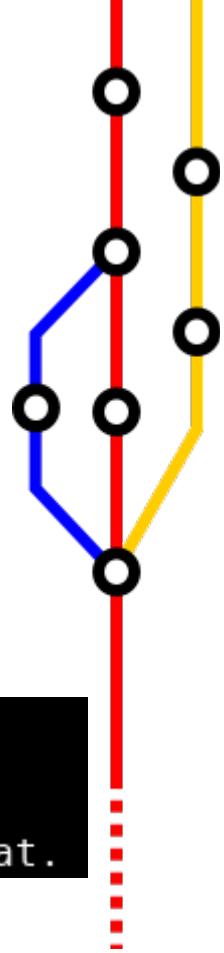
- Résultat :



→ Historique linéaire mais ré-écrit !

Les branches

Fusion - conflits



Qu'importe la méthode de fusion :

conflits possibles !

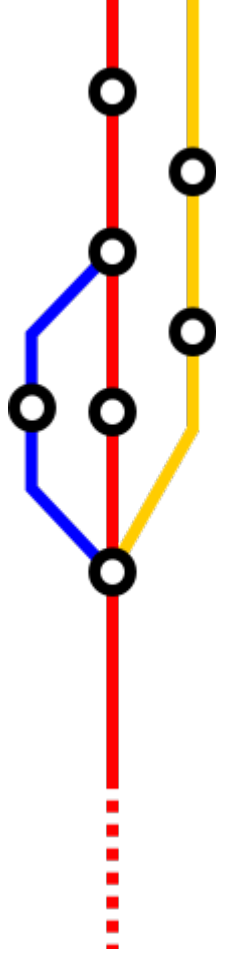
```
user@deb-dev:/tmp/session_git$ git merge feature
Fusion automatique de file2
CONFLIT (contenu) : Conflit de fusion dans file2
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

Source des conflits :

- Un même fichier modifié différemment sur les 2 branches

Les branches

Fusion – résolution de conflits



2 approches possibles :

- Abandonner la fusion :

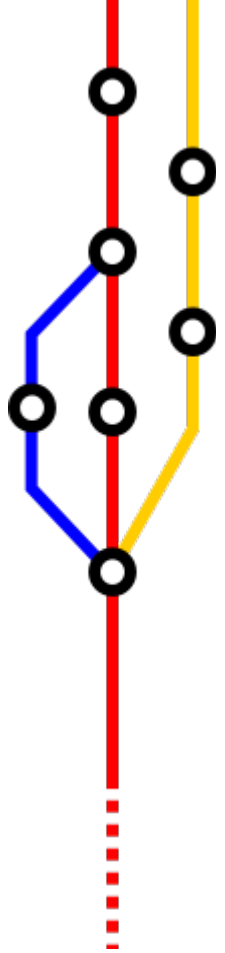
  $ git merge –-abort ou **$ git rebase --abort**

- Résoudre le conflit :

  1) Fusionner manuellement les fichiers en conflit
  2) Faire un commit

Outils : meld, kdiff3, votre éditeur favori,...

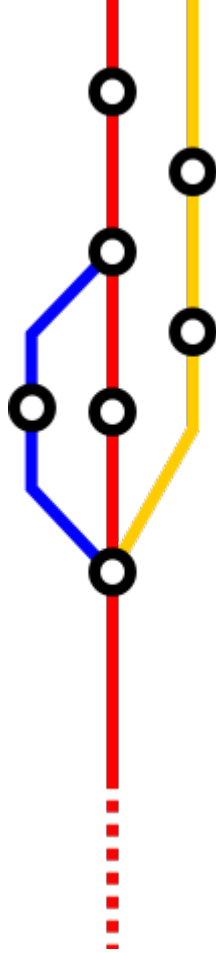
Les branches
Suppression



Pour supprimer une branche :

$ git branch -d <nom_branche>

Travail d'équipe avec Git



Les 'hébergeurs' Git

Il existe des 'hébergeurs' publiques :

- Github (Micro$oft)
- Framagit (Framasoft)

Auto-hébergement possible !

- Gitlab
- Serveur Git natif

Pousser un projet local

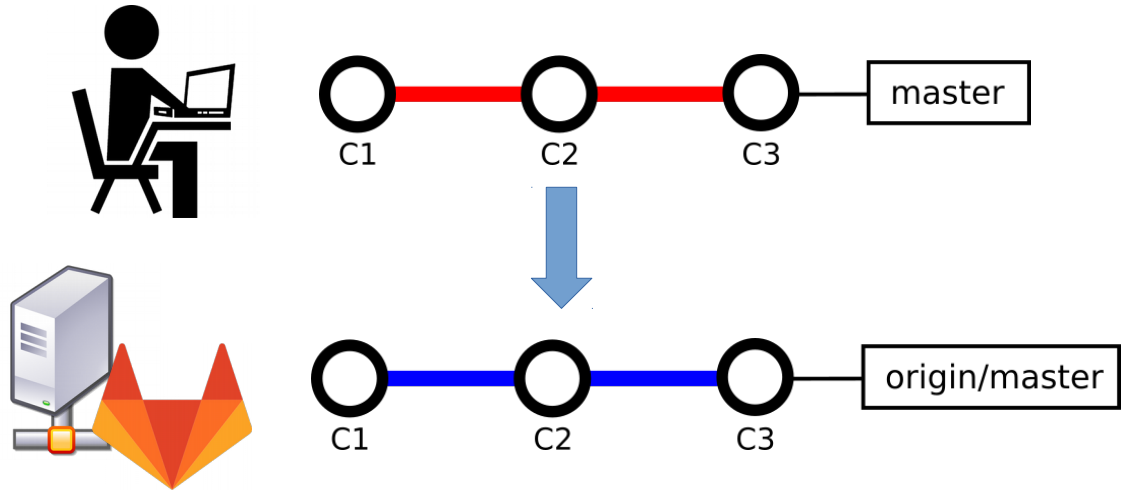
Après avoir créé un projet vide dans Gitlab :

1) Définir le serveur distant :

$ git remote add <label_serv.> <@_serv.>

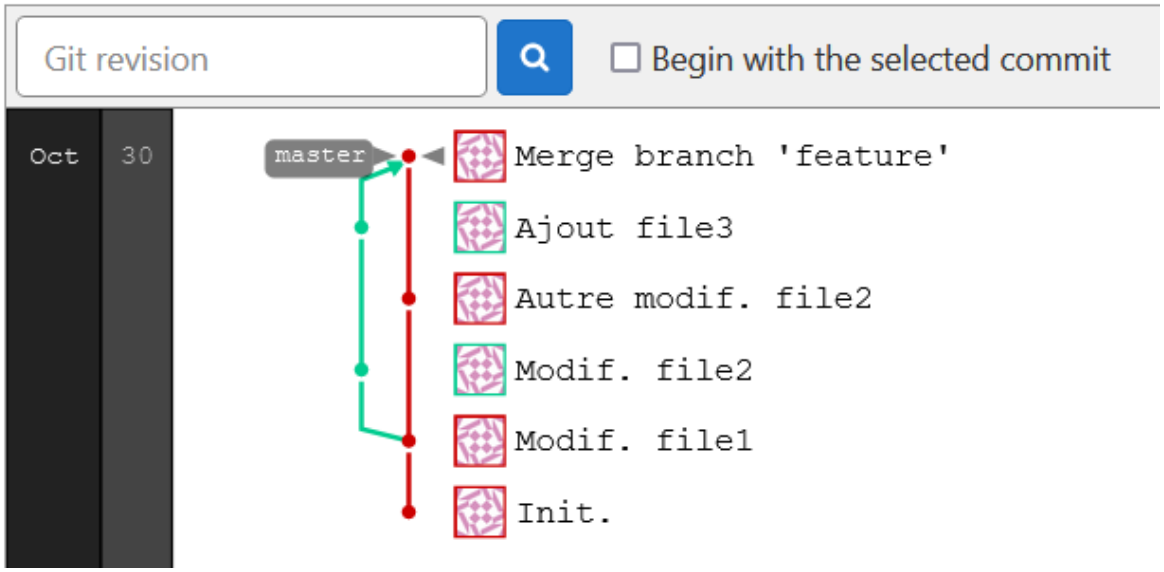
2) Pousser la branche 'master' sur le serveur distant :

$ git push -u <label_serv.> master



Travail d'équipe avec Git
Tour d'horizon de Gitlab

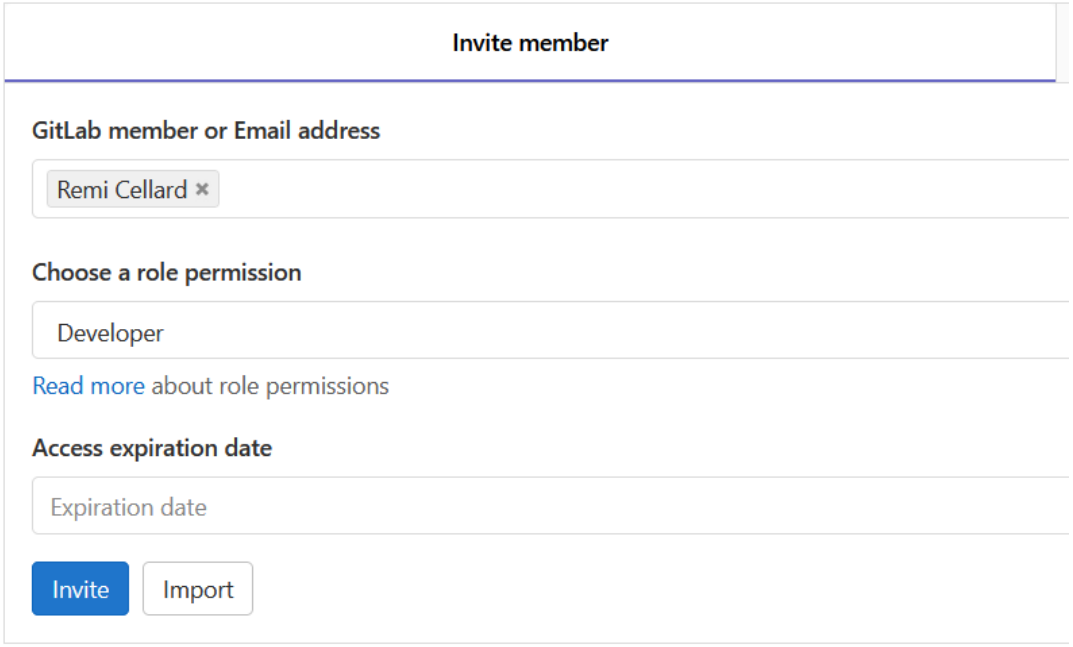
- Historique graphique (*Repository* > *Graph*) :



  - Visualiser l'auteur et le titre des commits
  - Accès rapide au détail de chaque commit

Tour d'horizon de Gitlab

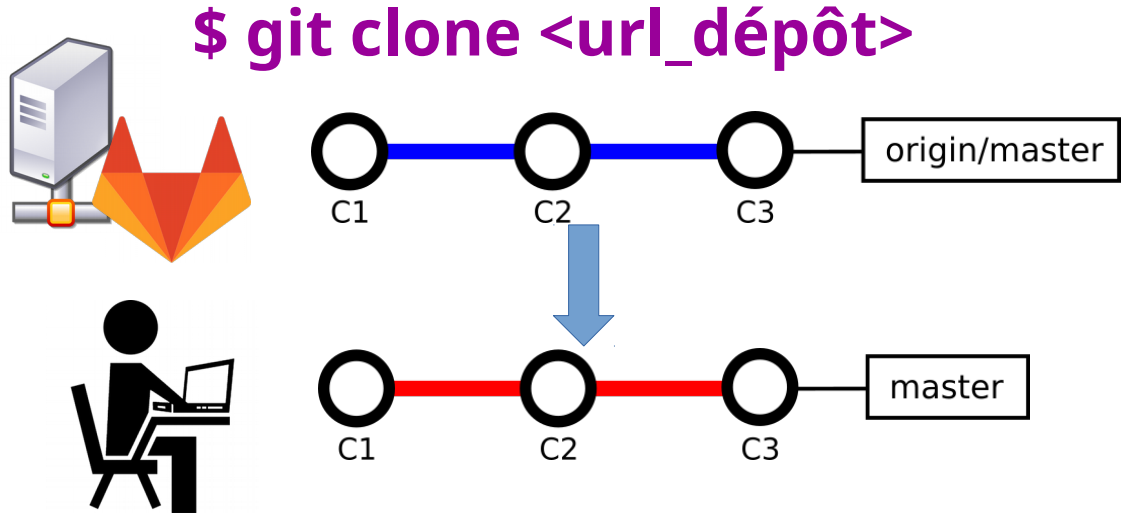
- Gestion des membres (*Members*) :



  - Inviter de nouveaux membres
  - Gérer les rôles

Récupérer un dépôt existant

Clonage d'un dépôt avec Git :



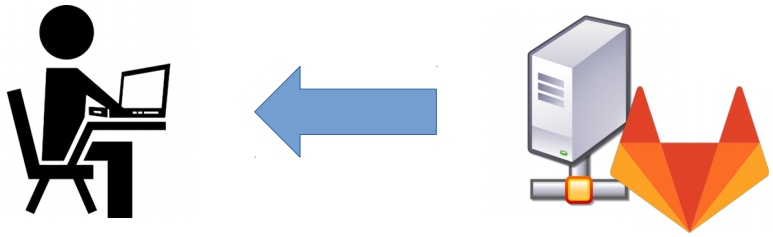
Remarques :

- URL doit pointer vers 'racine' du dépôt
- Clonage (par défaut) de tout l'historique

Se maintenir à jour

Pour synchroniser le dépôt local avec le dépôt distant :

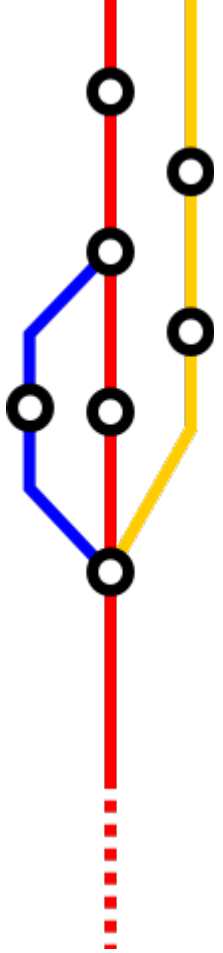
$ git pull



- Il s'agit d'une fusion de la branche distante dans la branche locale
- Si on a travaillé en local sur la branche :

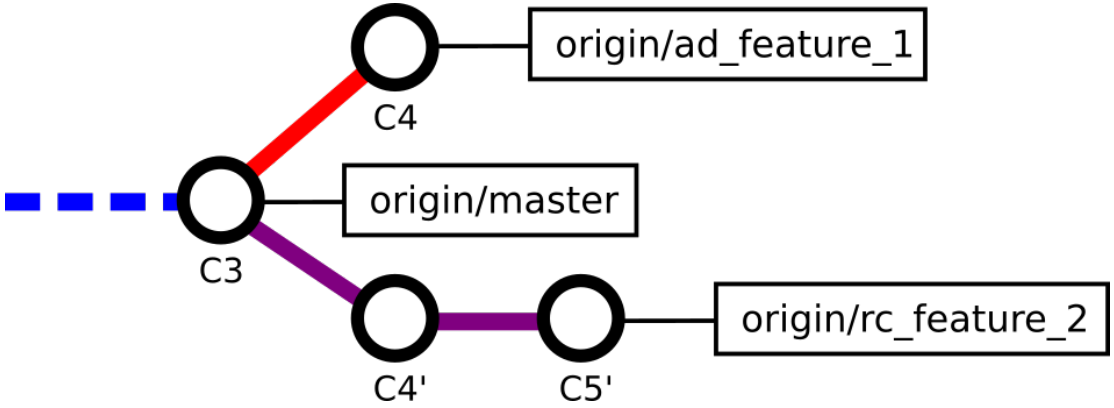
risque de conflits !

Travail d'équipe efficace



Ne jamais travailler en direct sur une branche partagée :

- Minimum une branche par membre



Ne pas faire de fusion directe sur une branche partagée :

- Exploiter le principe de *merge request*
- Généralement, un membre assigné aux fusions

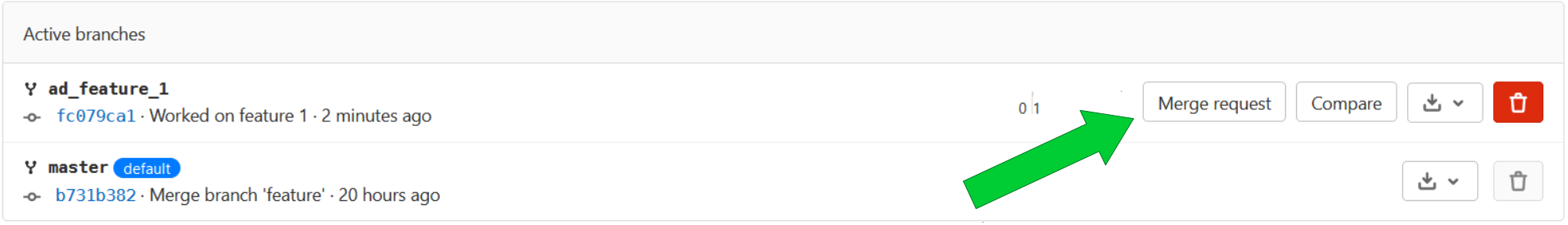
Travail d'équipe avec Git

Merge request – mode d'emploi

1) Pousser la branche locale à fusionner avec la branche partagée :

$ git push <nom_branche>

2) Sur Gitlab, créer une nouvelle *merge request* (*Repository* > *Branches*) :

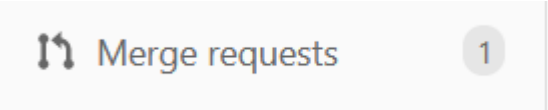


Club Informatique

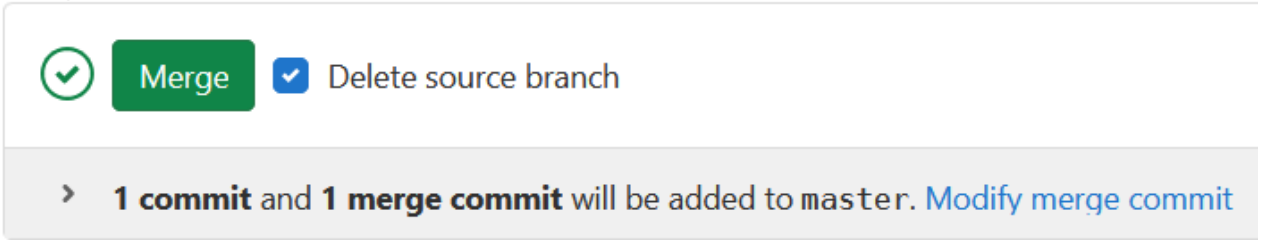
Merge request – mode d'emploi

Pour le membre en charge de la fusion :

- Examiner la *merge request*

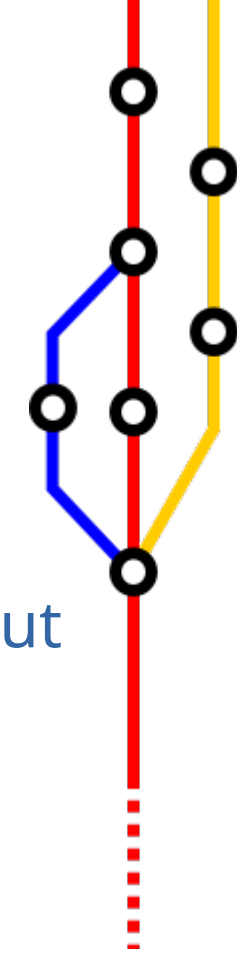


- Valider la *merge request*



- En cas de conflits, résolution possible directement depuis Gitlab

Liens et références



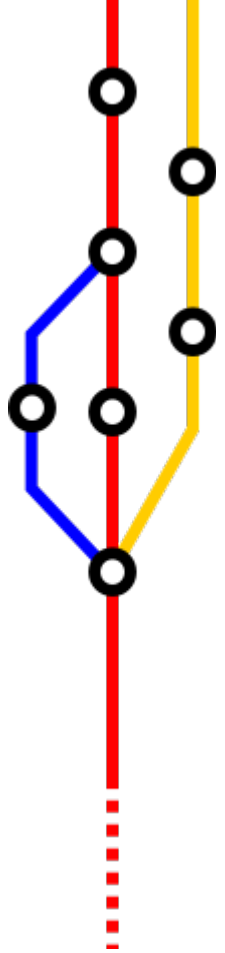
- **Formation Git de l'IUT de Valence :**

  https://github.com/benito103e/revealjs-formation-git-iut

- **Apprendre Git de façon ludique ! (FR)**

  https://learngitbranching.js.org/

Liens et références



- **Référence de la doc Git (EN) :**

https://git-scm.com/docs/

- **La GitHub cheat sheet (EN) :**

https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf

Merci de votre attention !