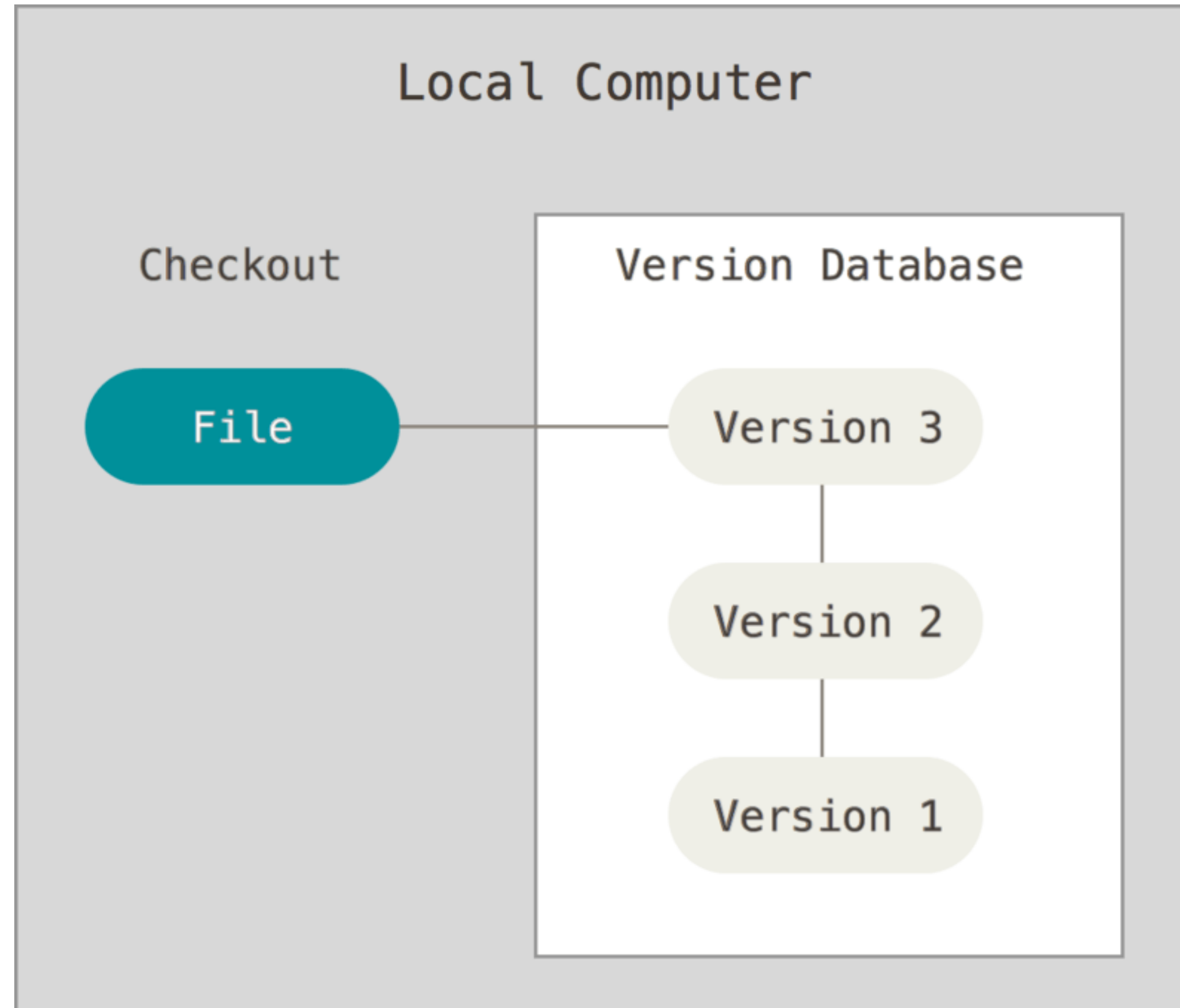
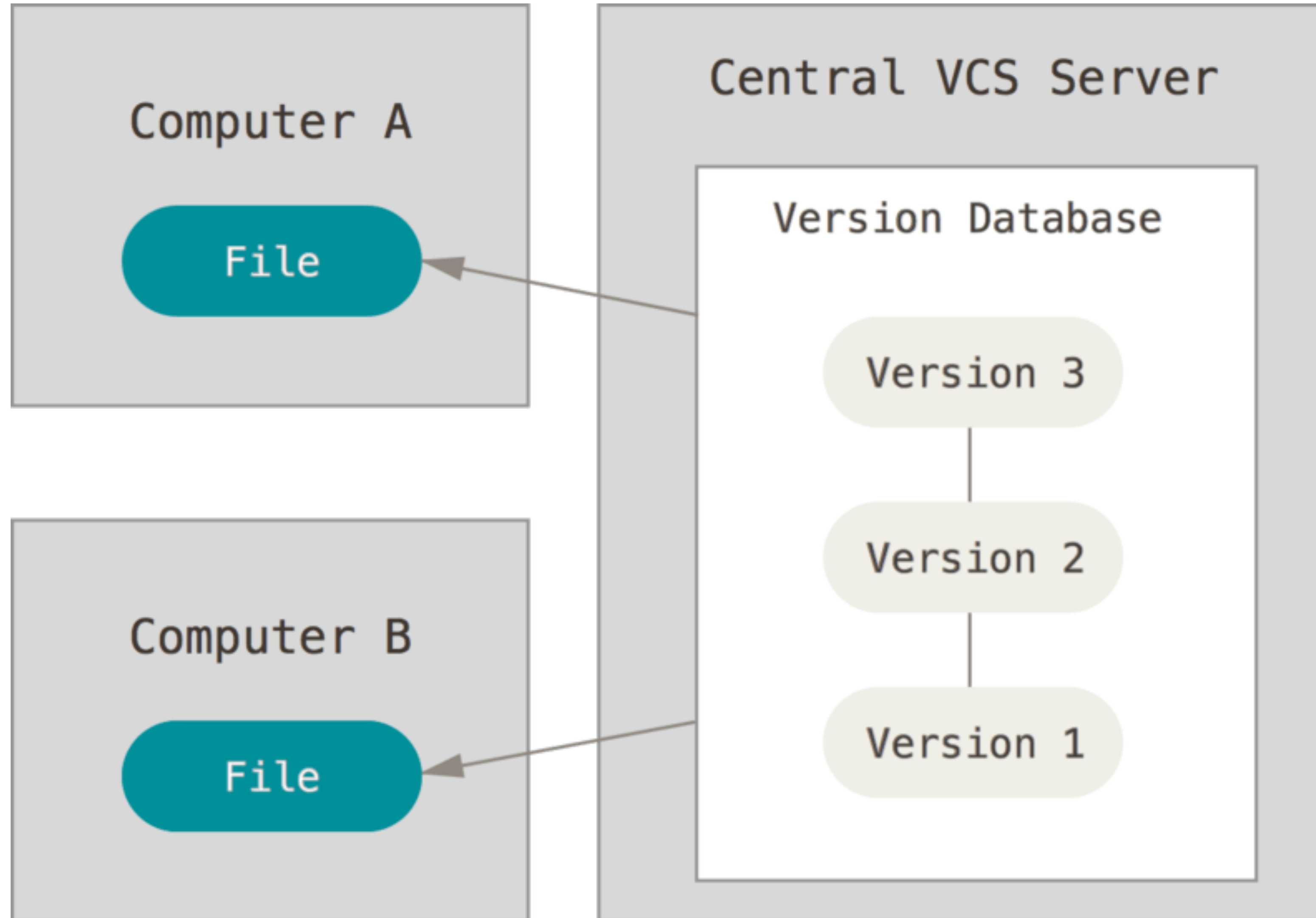


# Git pour PRG1

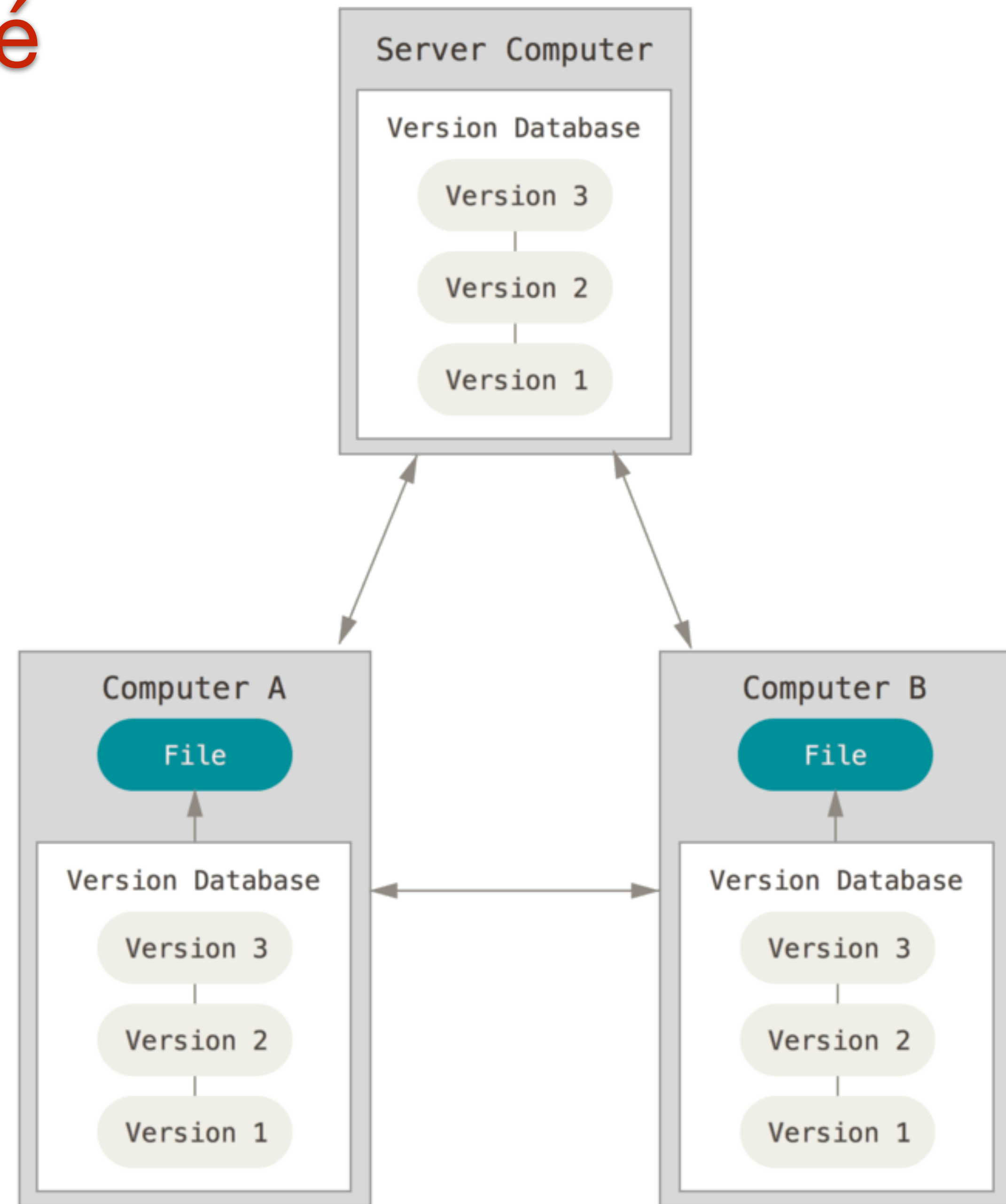
# VCS = Version Control Software



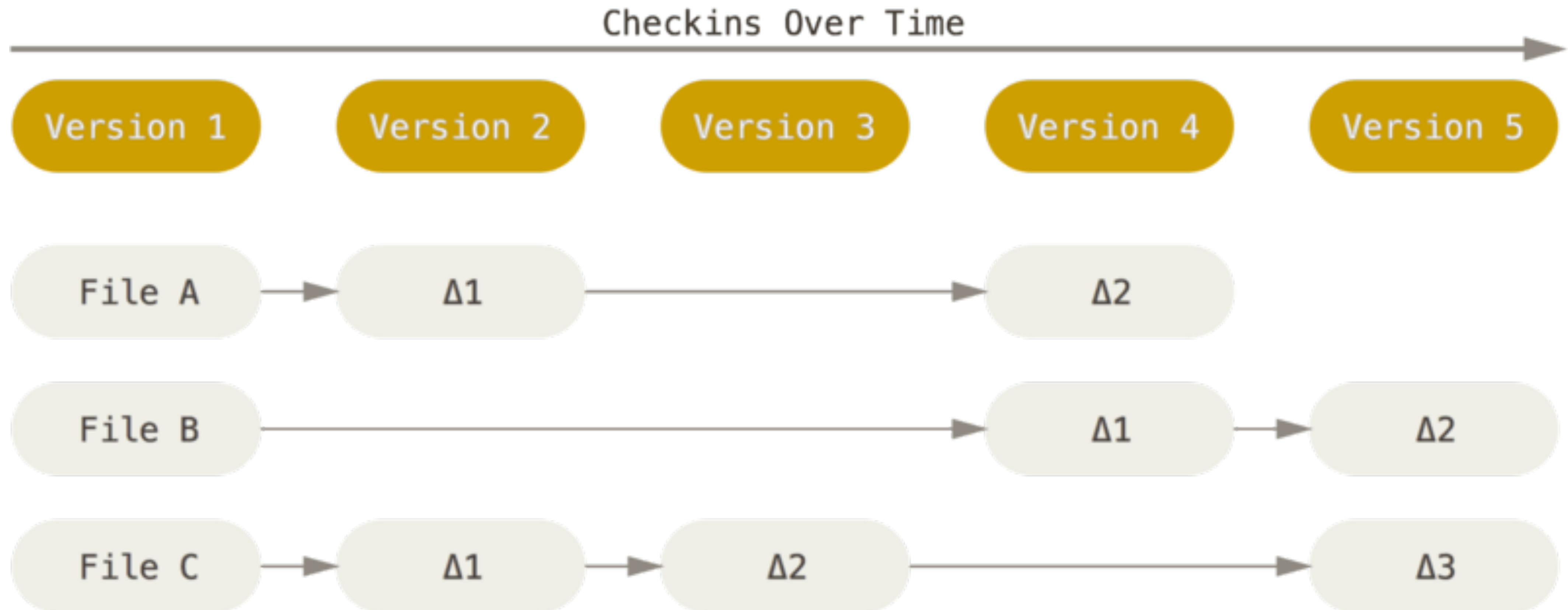
# VCS centralisé



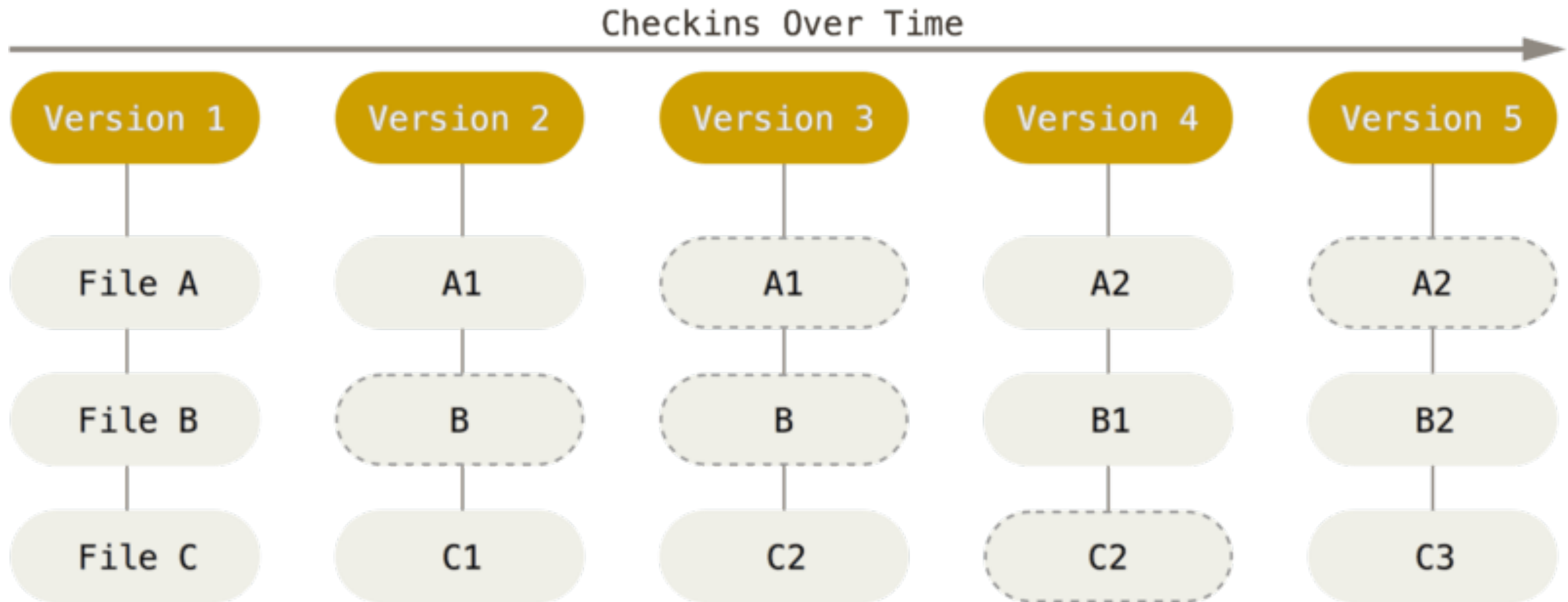
# Git = VCS distribué



# Les anciens VCS stockent des différences



# Git stocke les fichiers qui ont changé



# Démarrer un repo git

- ◆ Localement

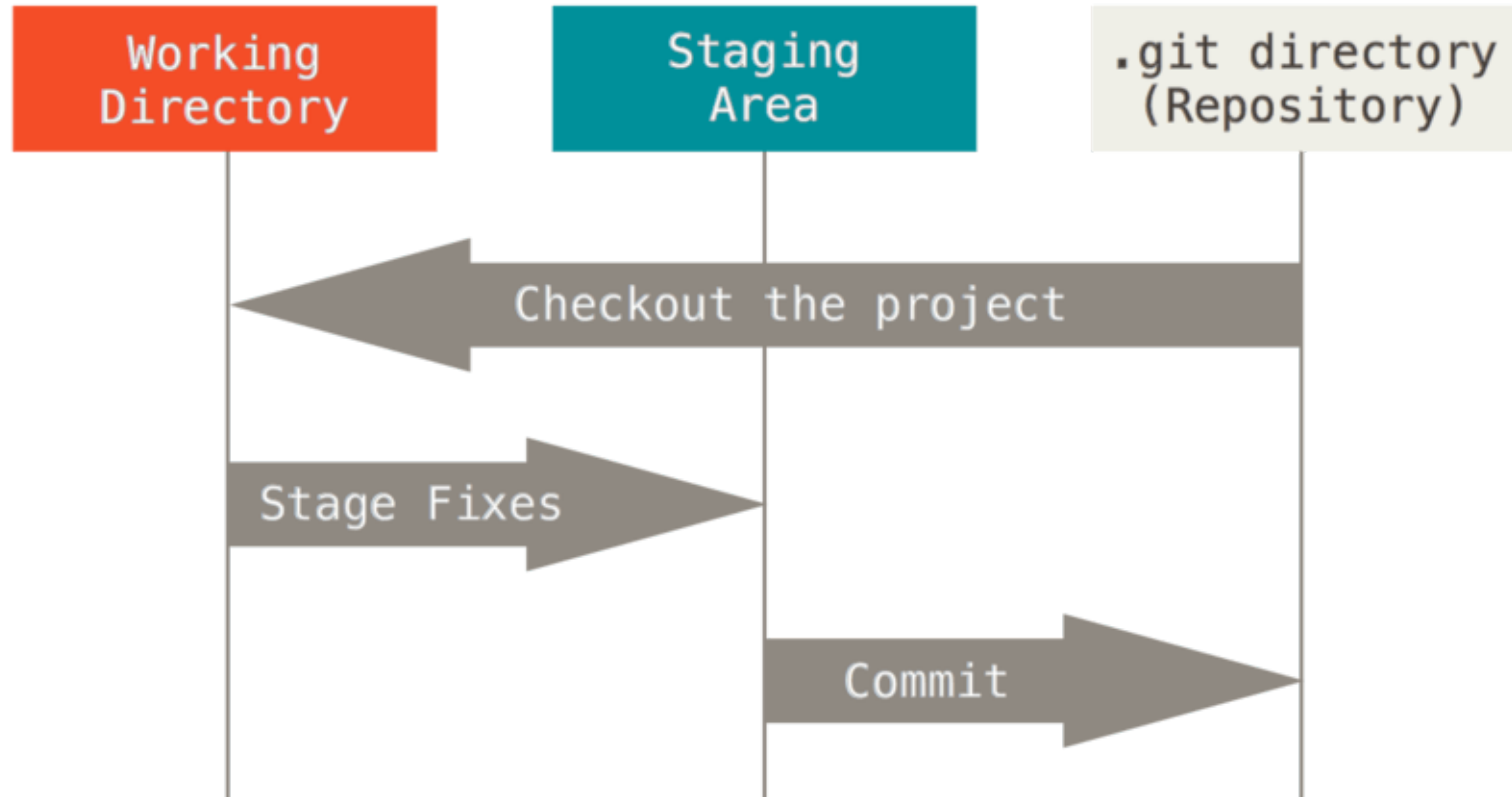
```
git init
```

- ◆ A partir d'un repo à distance

```
git clone https://github.com/PRG1-HEIGVD/PRG1\_Recueil\_Exercices.git
```

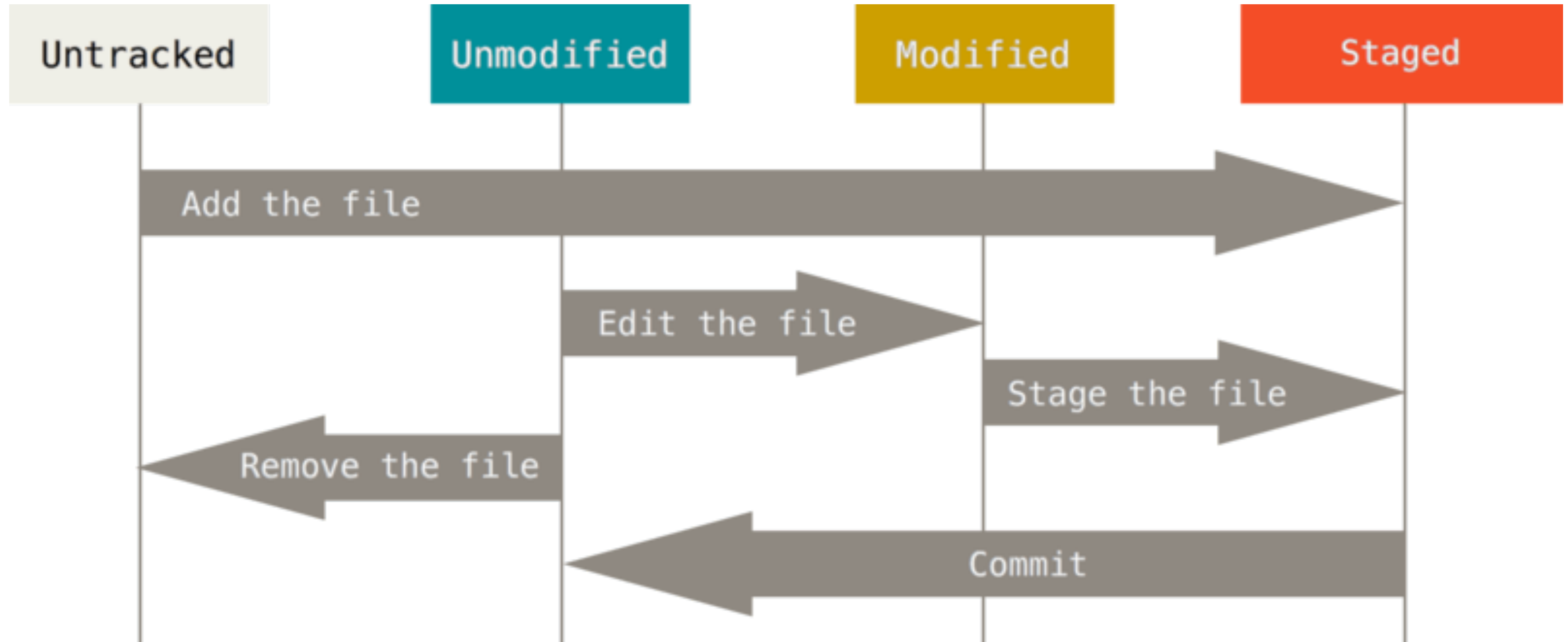


# Sections d'un projet Git





# Etats d'un fichiers



# Les commandes git

- ♦ `git add [fichiers]`
  - ♦ Place les fichiers et/ou répertoires dans la staging area
  - ♦ Valide depuis untracked ou depuis modified
  - ♦ Inopérant depuis unmodified
- ♦ `git commit -m "message à choix"`
  - ♦ Déplace les fichiers de la staging area au repository
  - ♦ Les fichiers commités sont maintenant unmodified

# Les commandes git (2)

- ♦ `git status`

- ♦ Indique les fichiers untracked, modified ou staged dans votre répertoire

- ♦ `git reset [fichier]`

- ♦ Annule un `git add` en retirant le fichier modifié de la staging area

- ♦ `git diff [fichier optionnel]`

- ♦ Affiche la différence entre le(s) fichiers courants et la version commitée

# Les commandes git (3)

- ♦ `git diff --staged`

- ♦ Affiche la différence entre les fichiers courants et la staging area

- ♦ `git checkout [fichier]`

- ♦ Ecrase le fichier courant avec la version stockée dans le repo

- ♦ `git rm [fichier]`

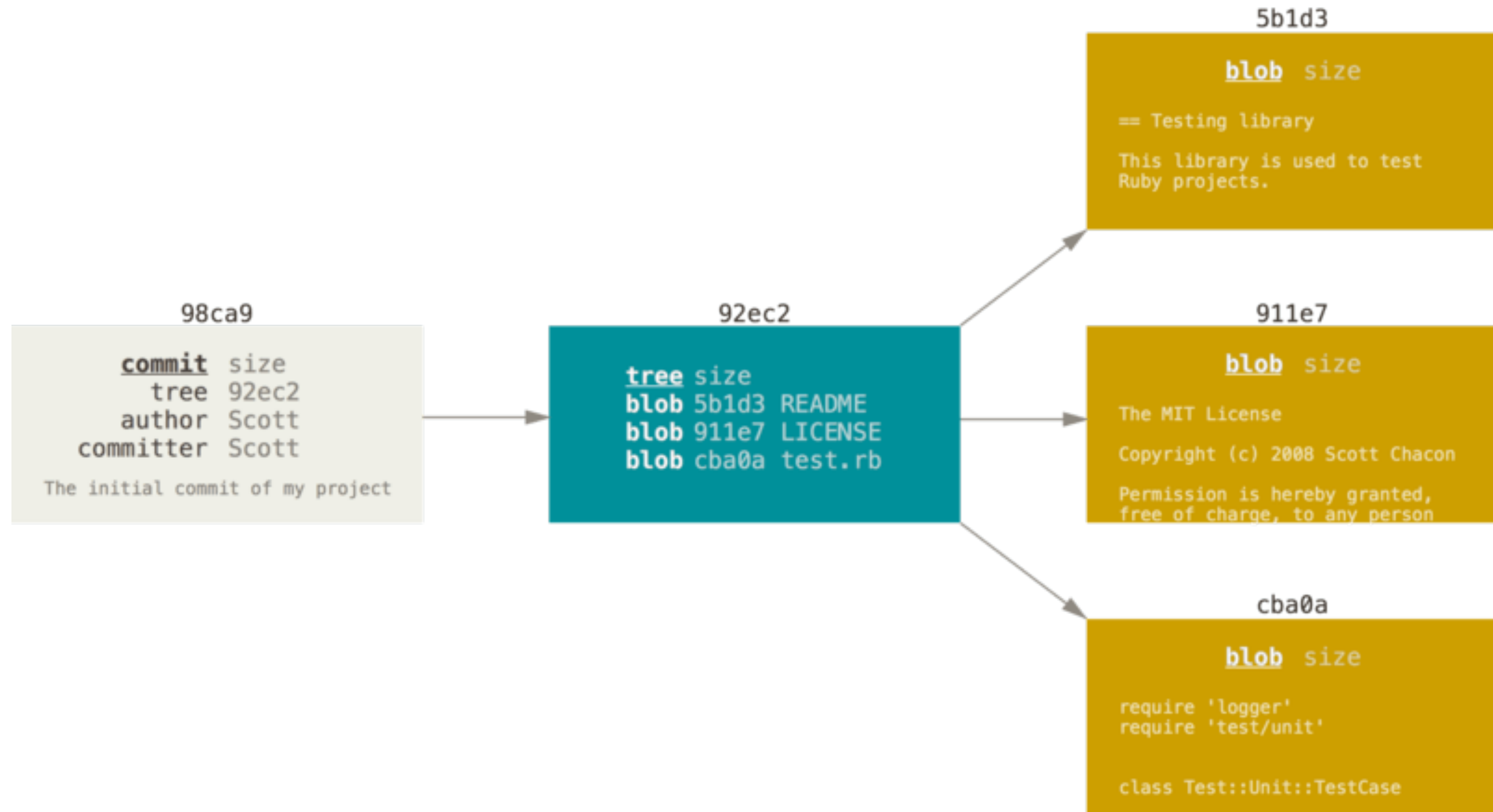
- ♦ Efface le fichier et le « stage » pour le prochain commit

- ♦ `git mv [fichier]`

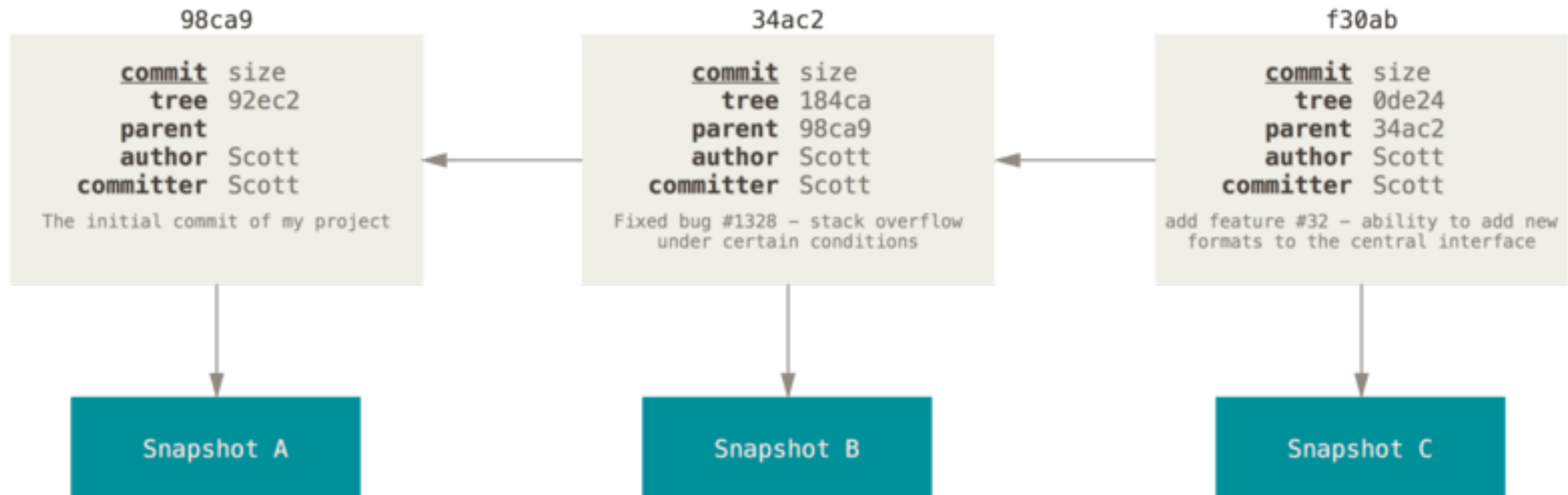
- ♦ Renomme le fichier et l'annonce pour le prochain commi

# Les branches

# Un commit dans le repository

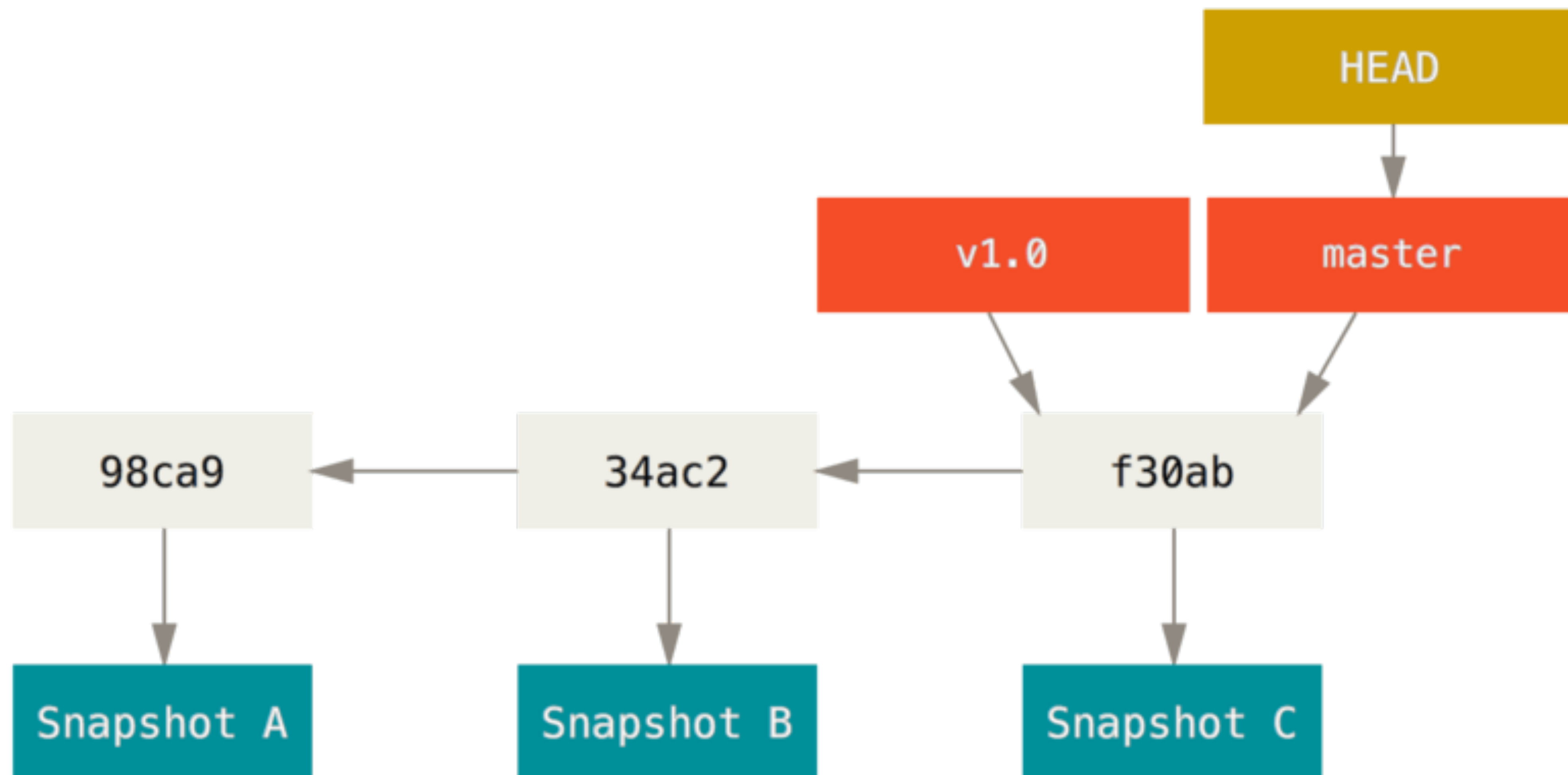


# Liste chaînée de commits





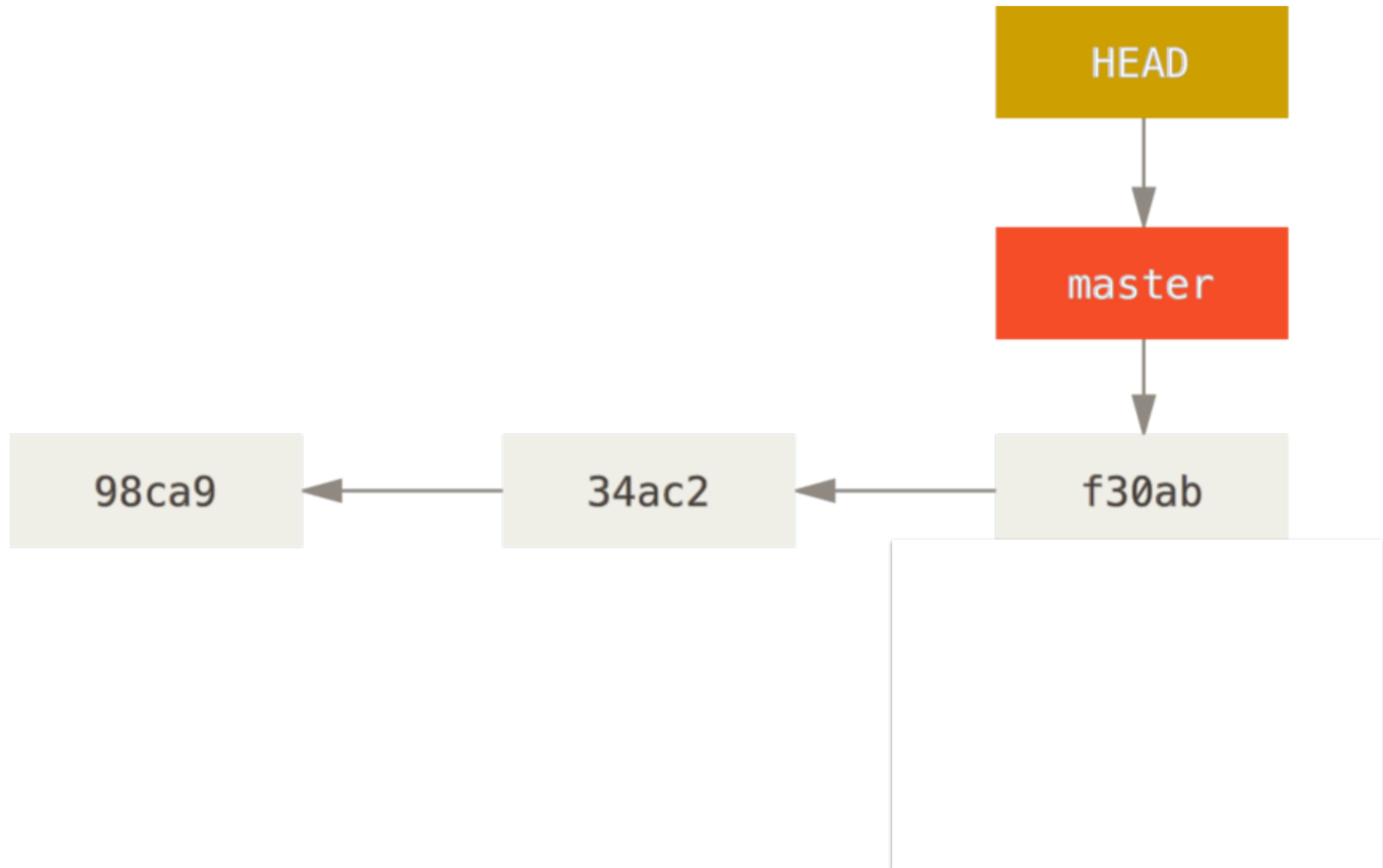
# Branche = pointeur vers un commit



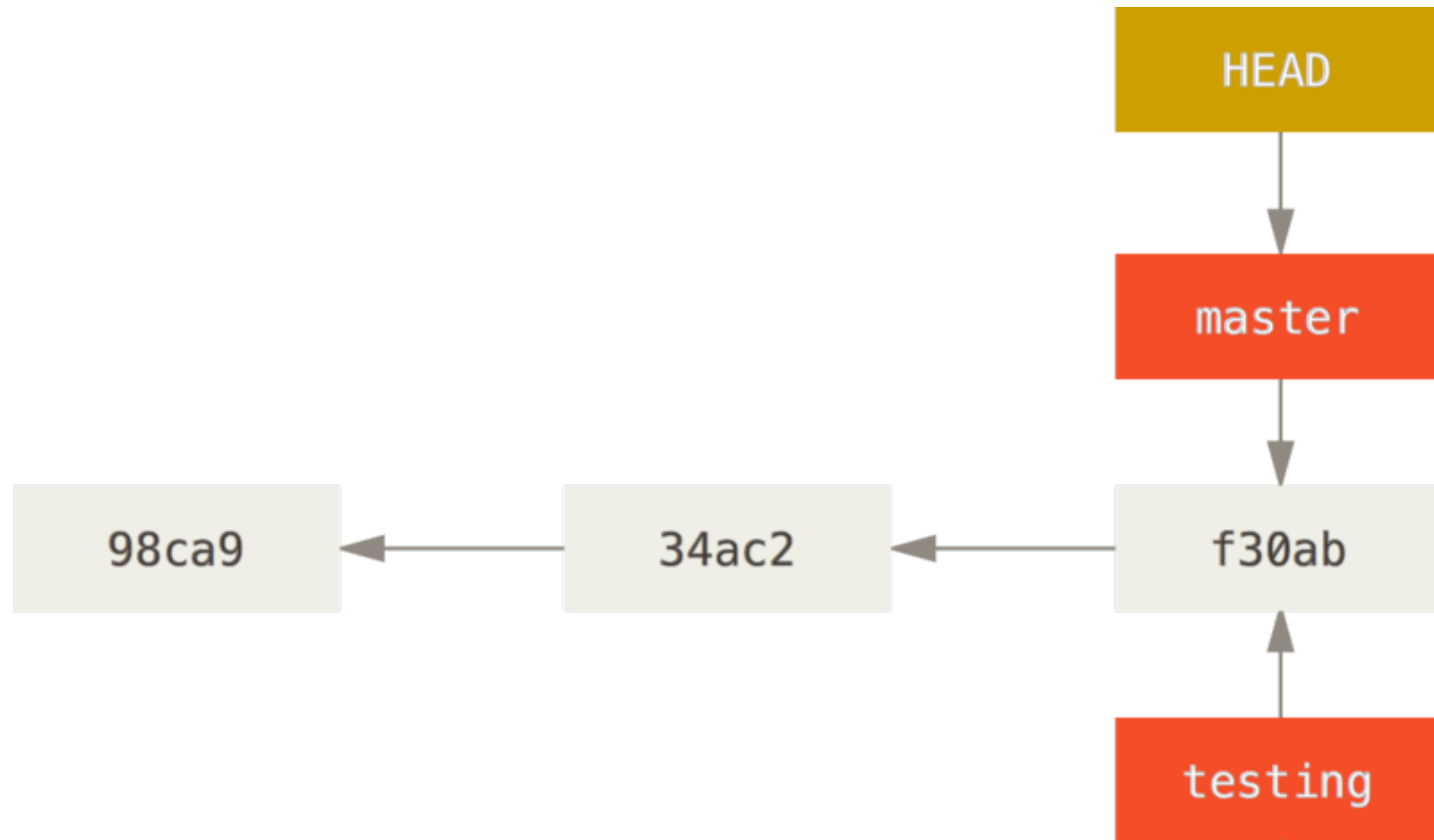
# Les commandes git (4)

- ♦ `git branch`
  - ♦ Liste les branches de votre repo
- ♦ `git branch [branch-name]`
  - ♦ Crée une nouvelle branche depuis l'emplacement courant dans le repo
- ♦ `git checkout [branch-name]`
  - ♦ Aller dans la branche donnée et modifier le contenu local en conséquence

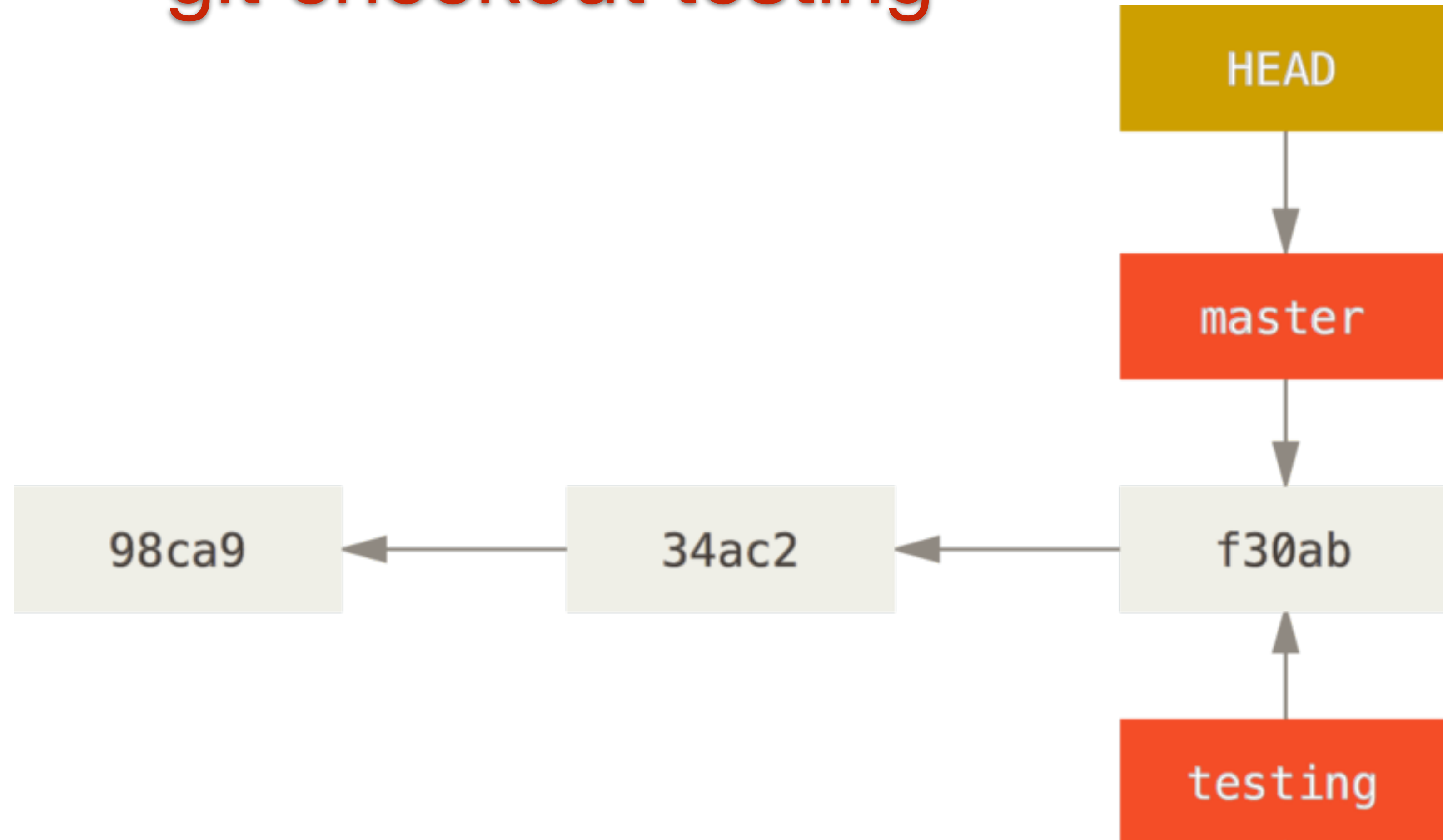
# git branch testing



# git branch testing



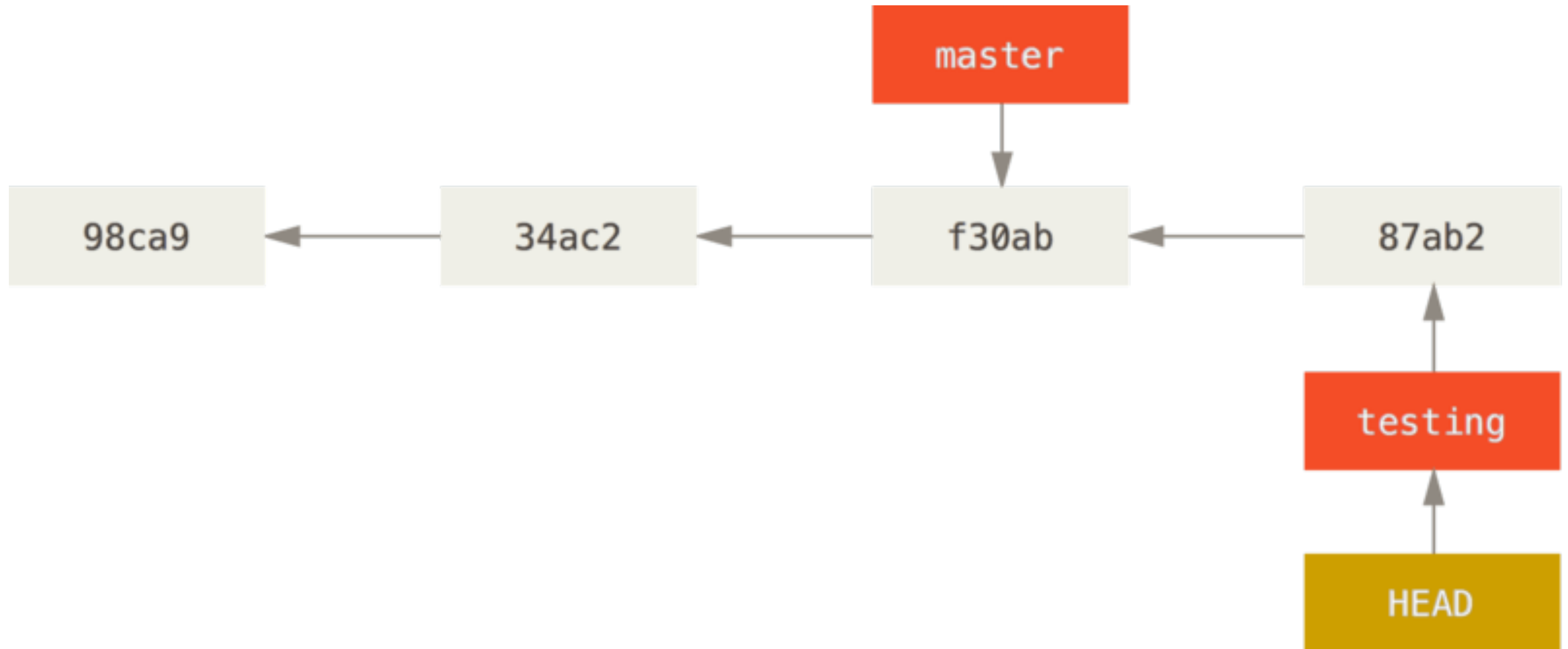
# git checkout testing



# git checkout testing

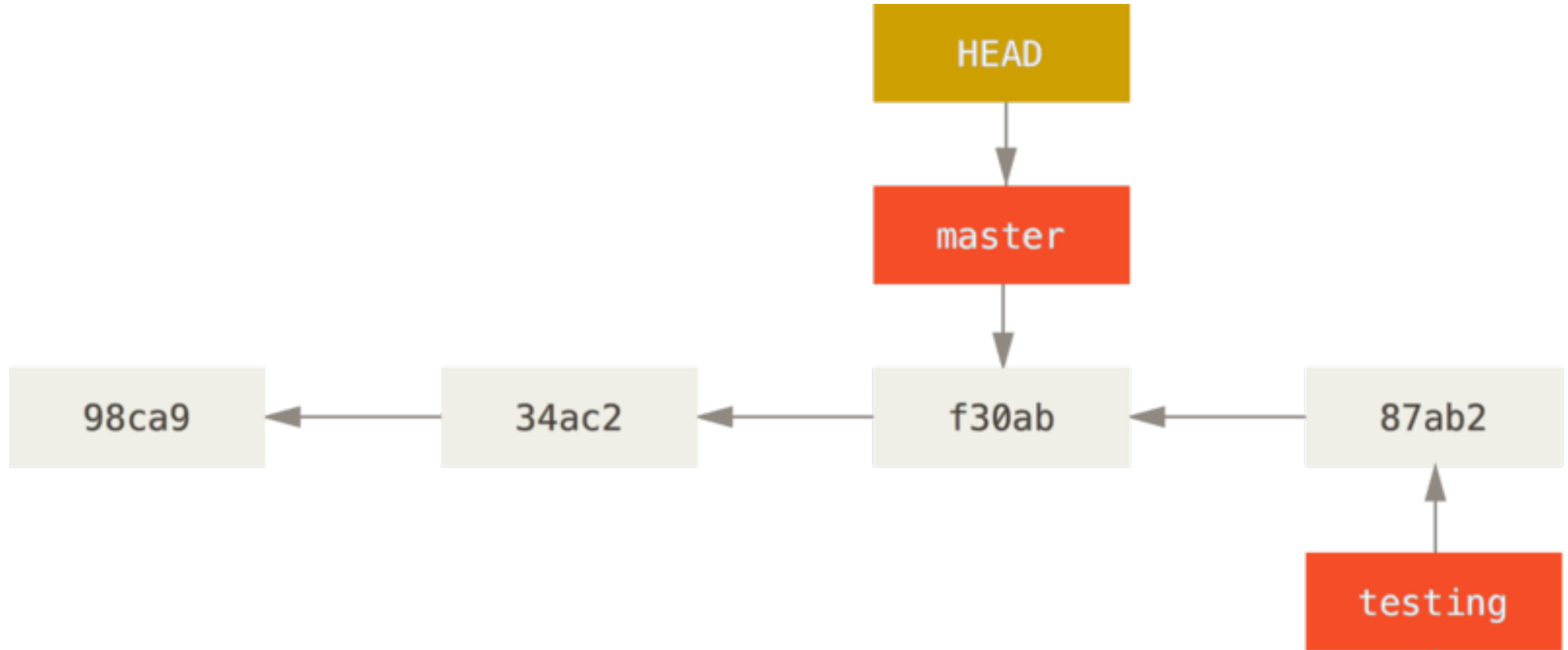


```
vim test.rb  
git commit -a -m 'made a change'
```

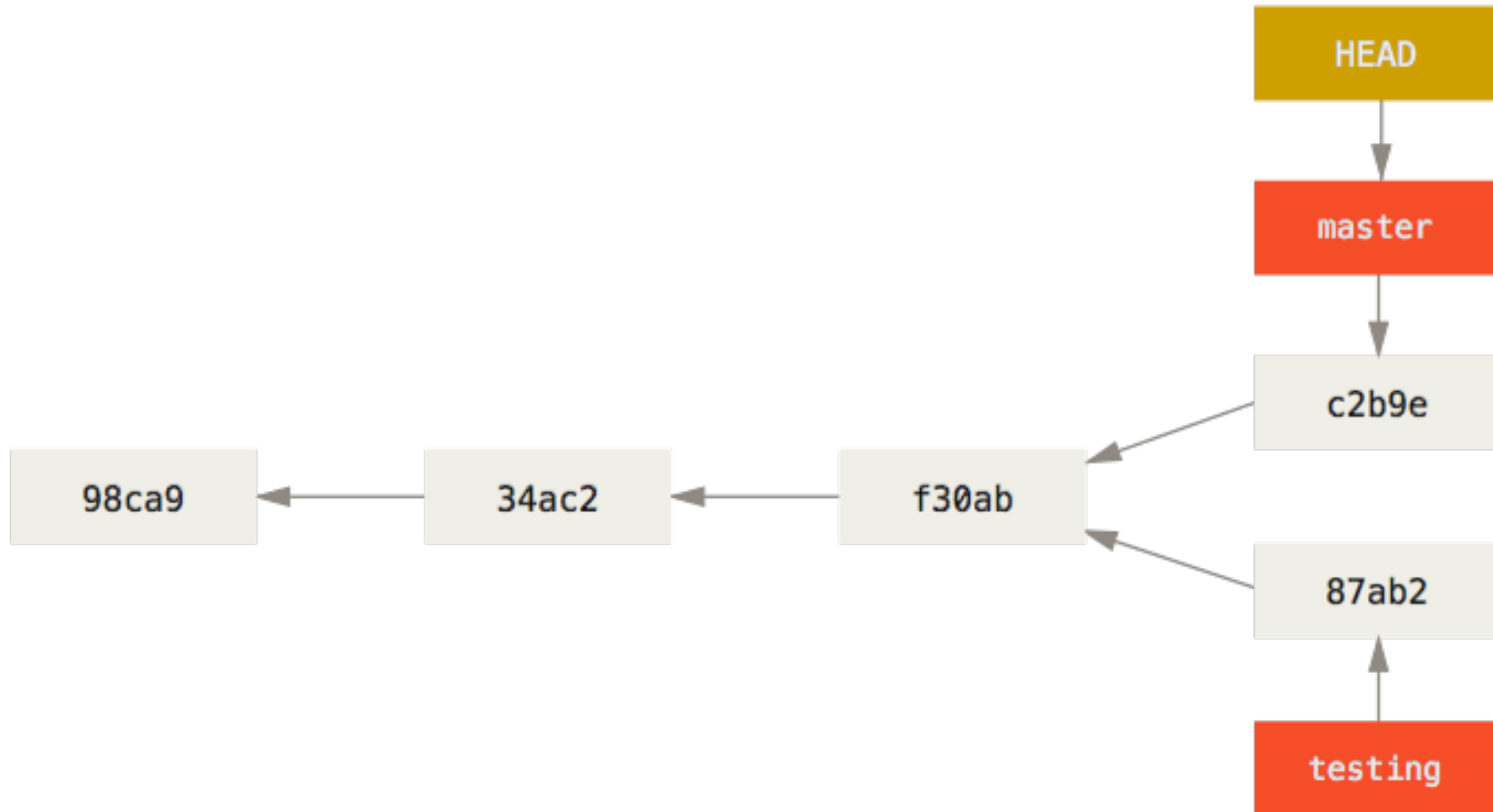




# git checkout master



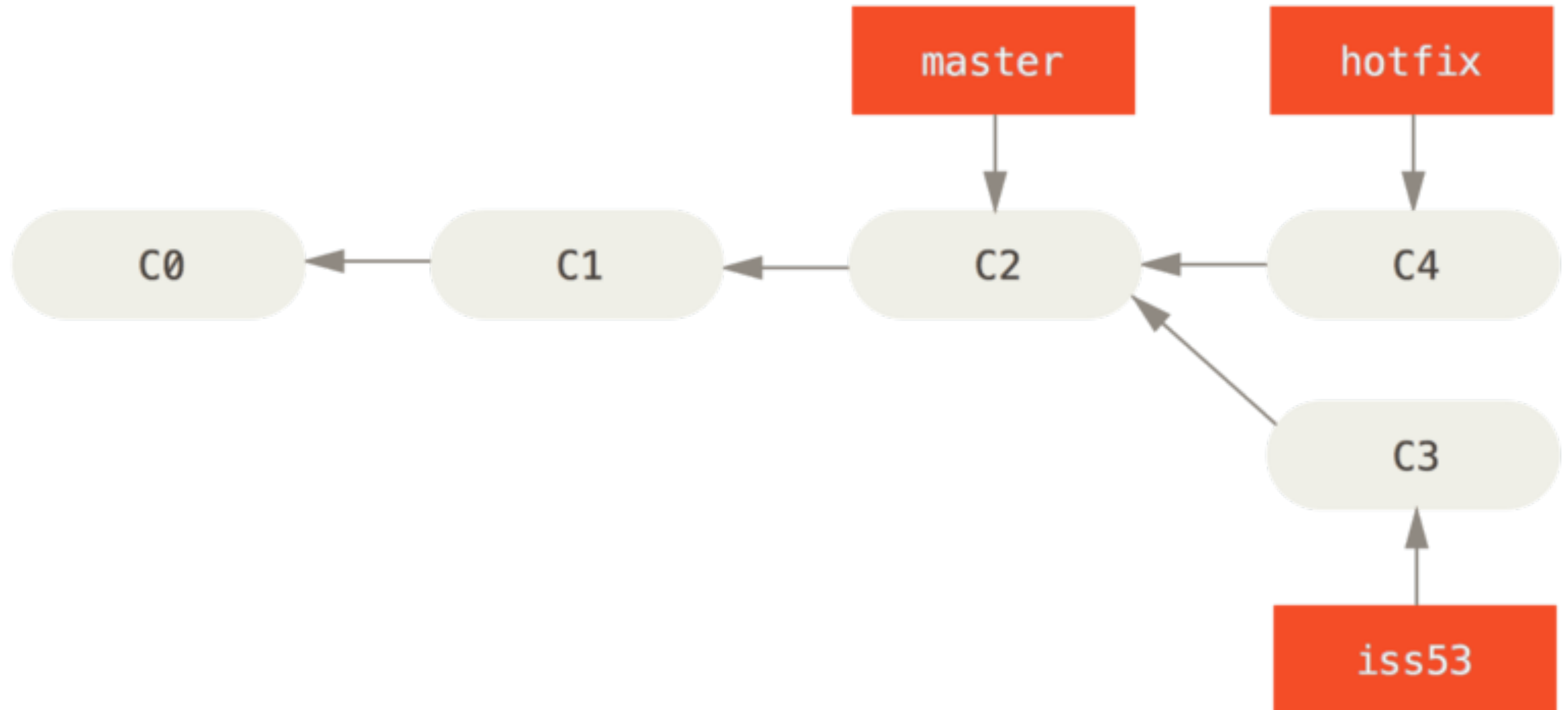
vim test.rb  
git commit -a -m 'made other changes'



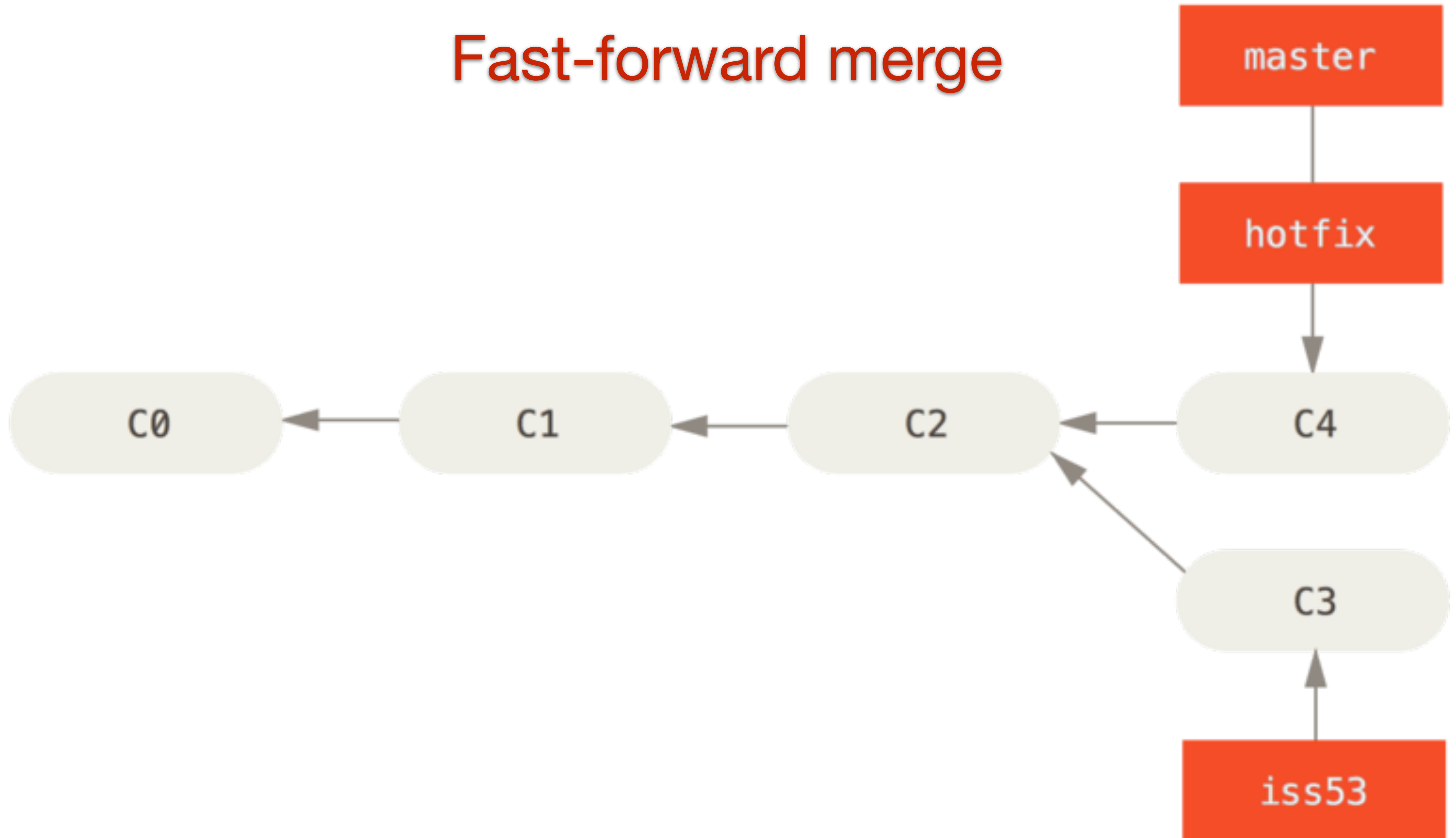
# Les commandes git (5)

- ♦ `git merge [branch]`
- ♦ Fusionne la branche nommée avec la branche courante

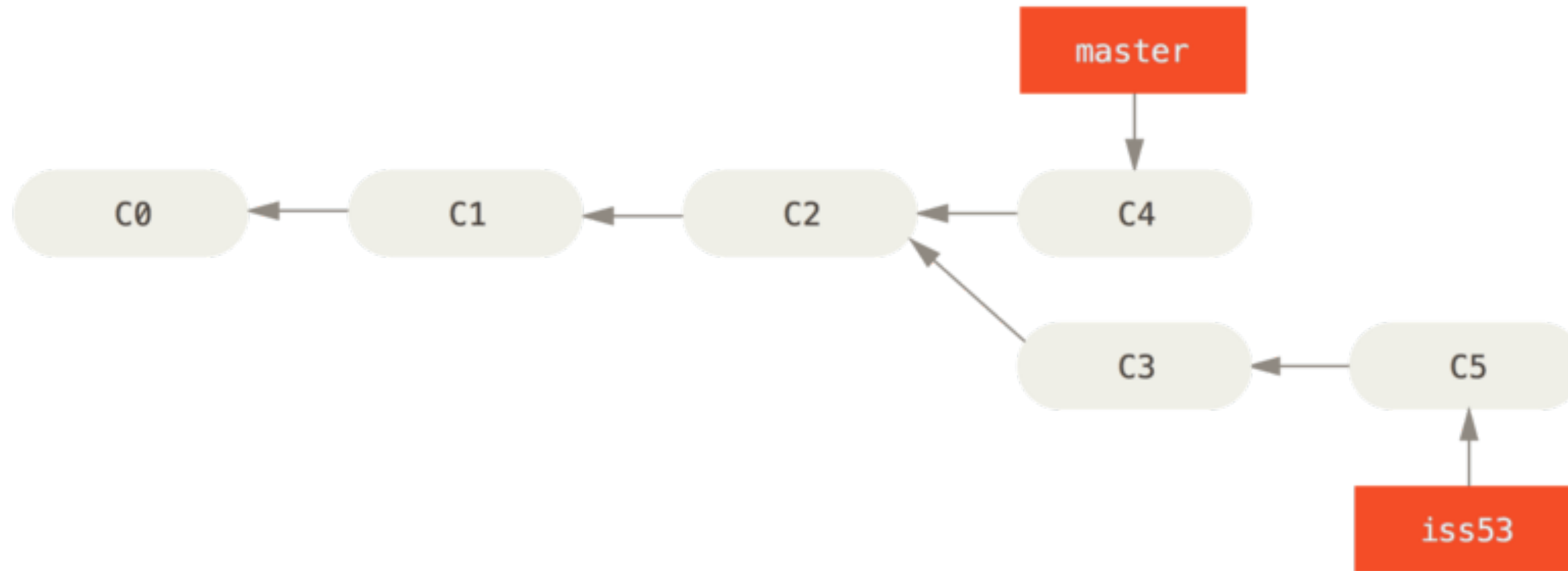
# Fast-forward merge



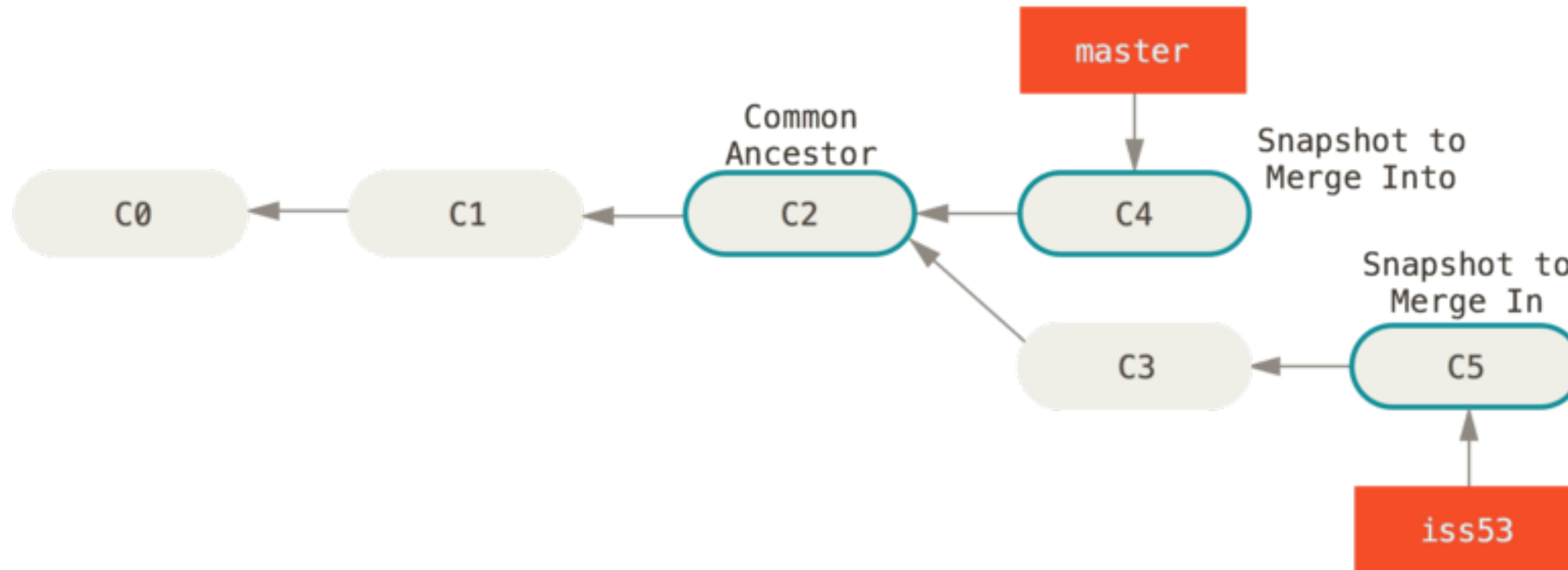
# Fast-forward merge



# Basic merge

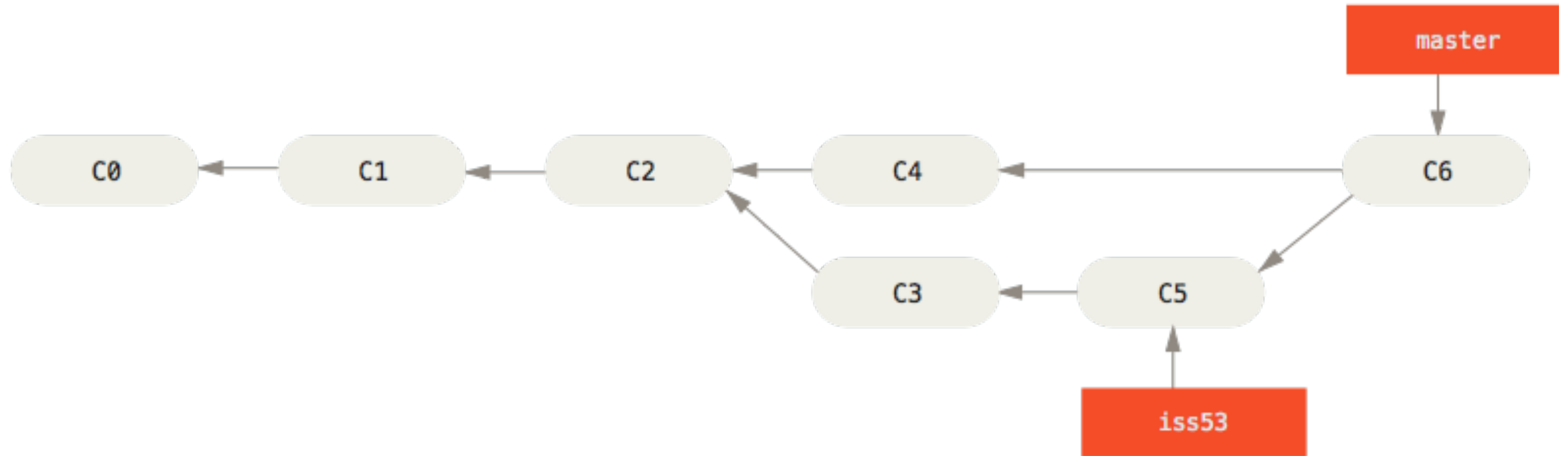


# Basic merge

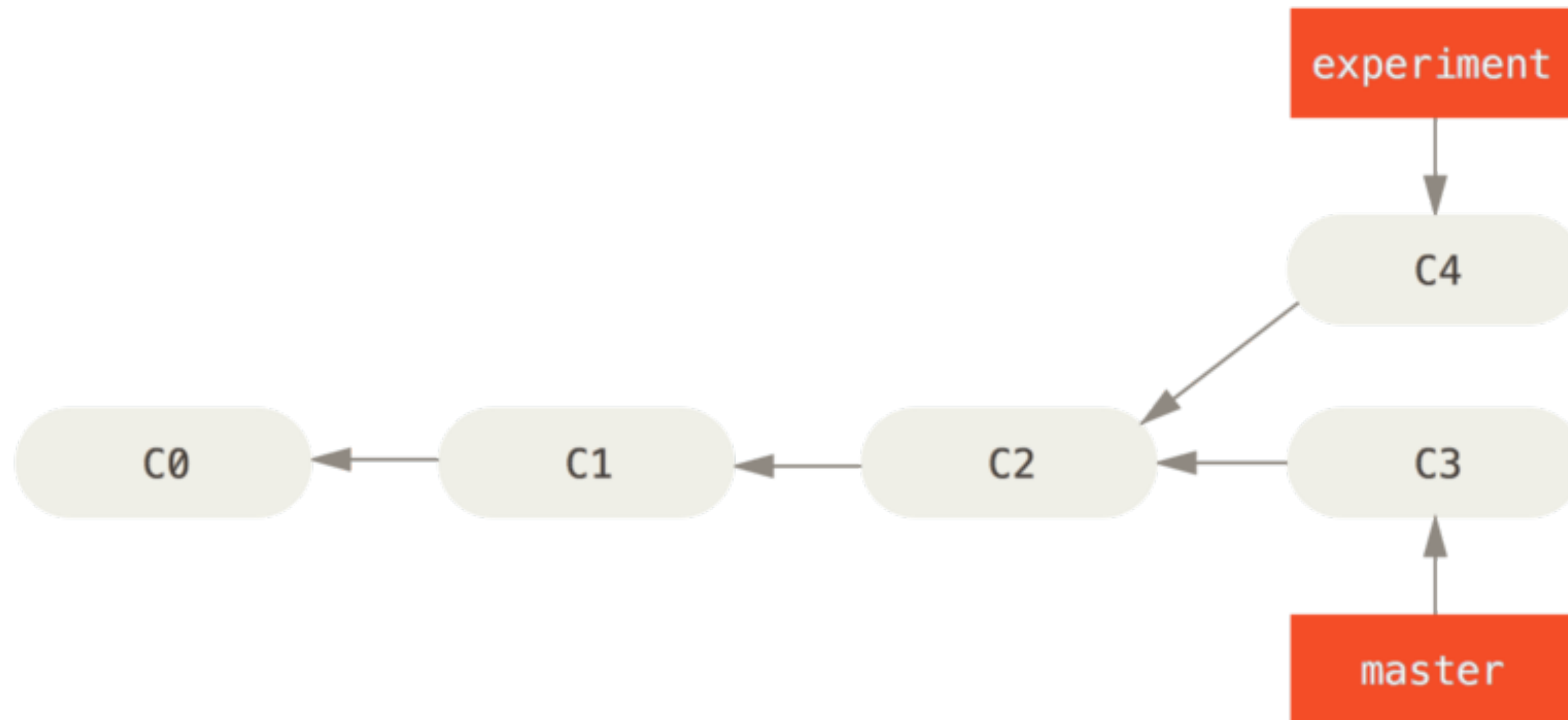




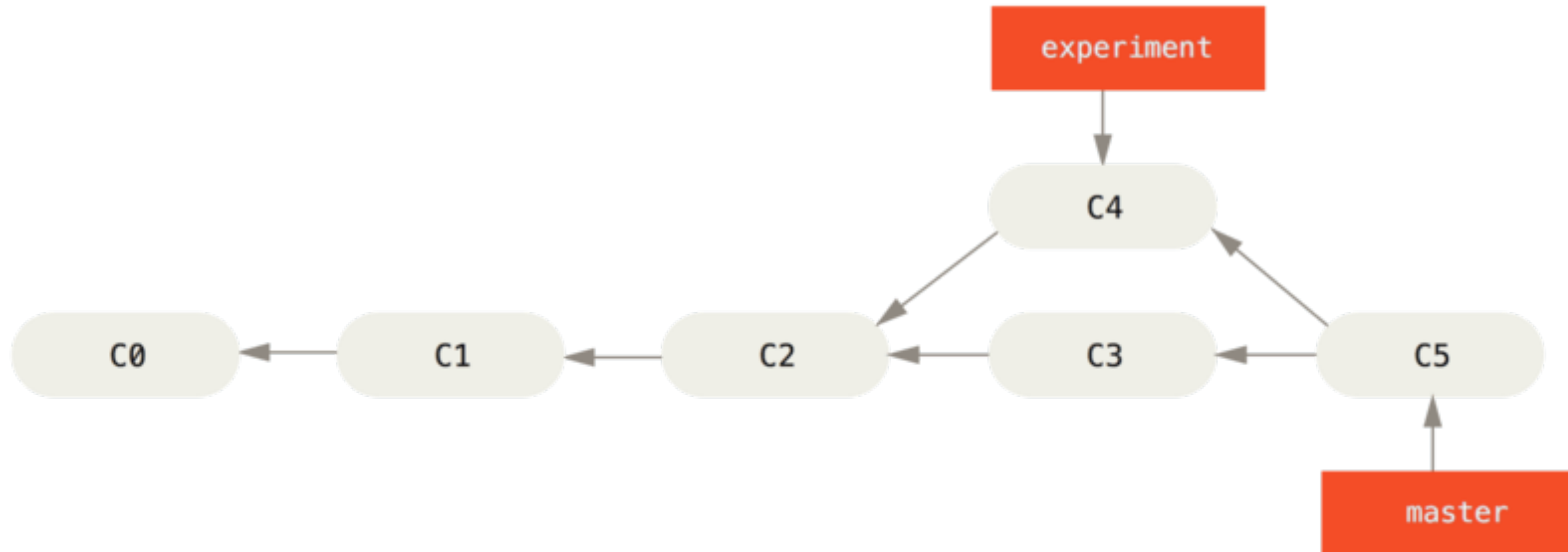
# Basic merge



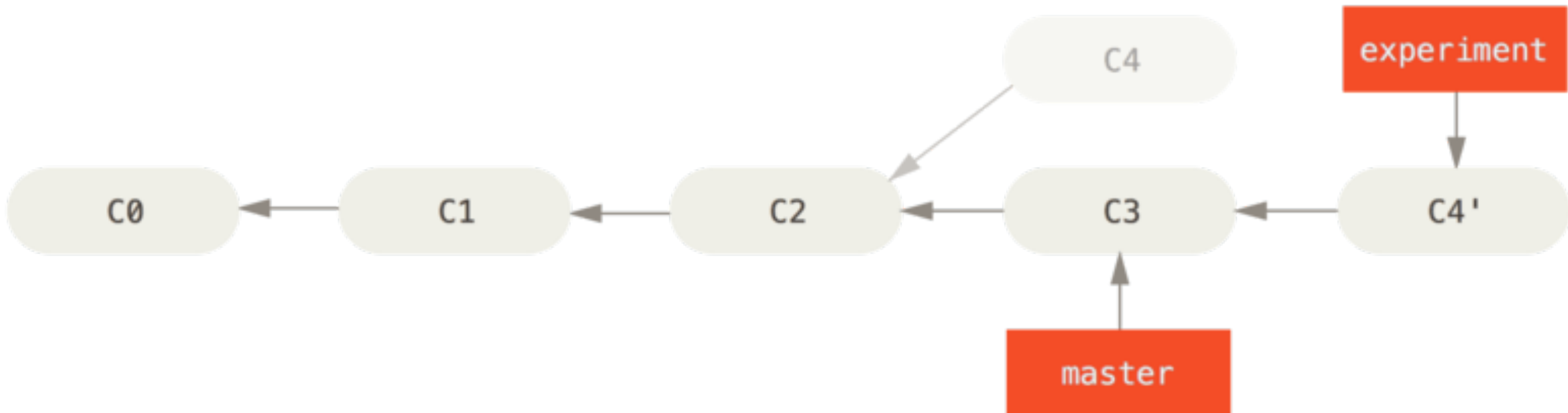
# merge vs. rebase



# merge vs. rebase

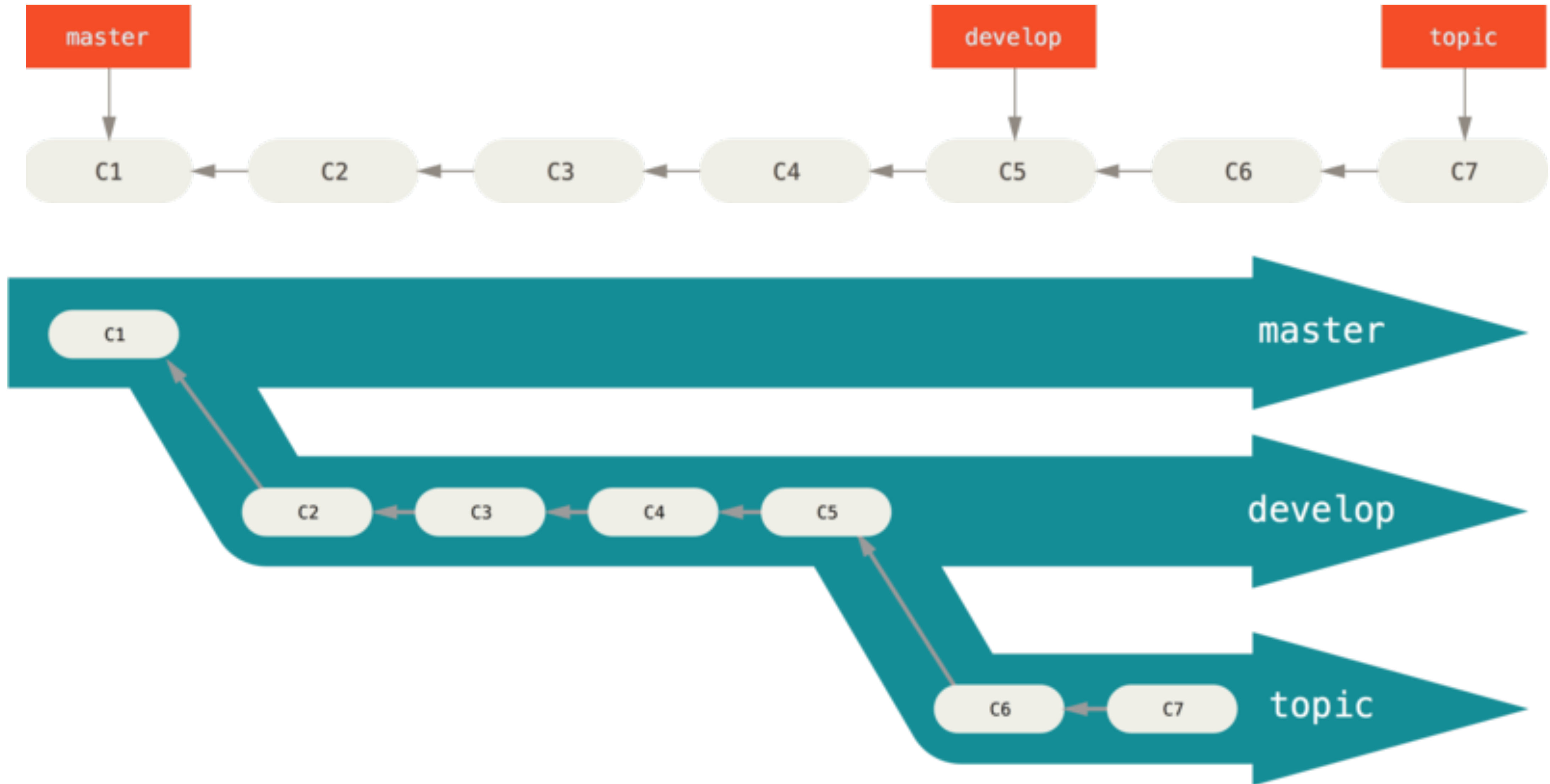


# merge vs. rebase

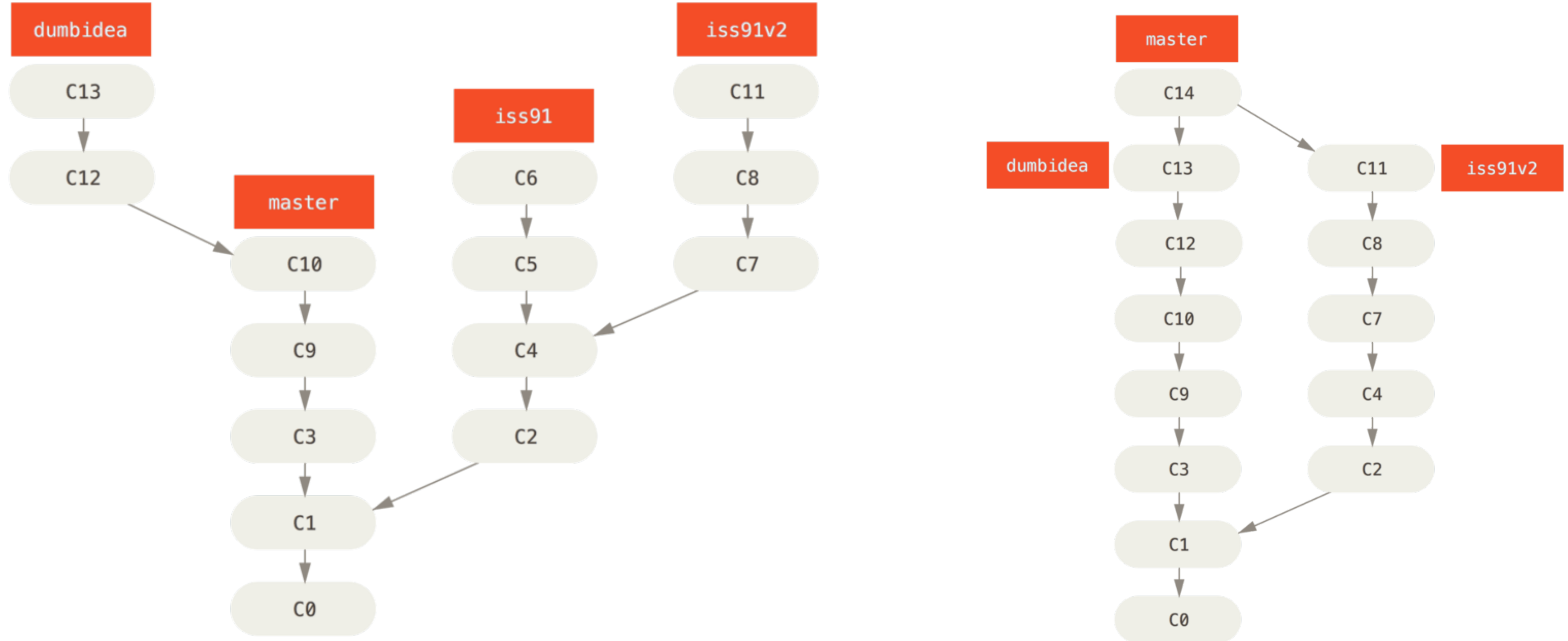


# Branches aux long court

heig-vd



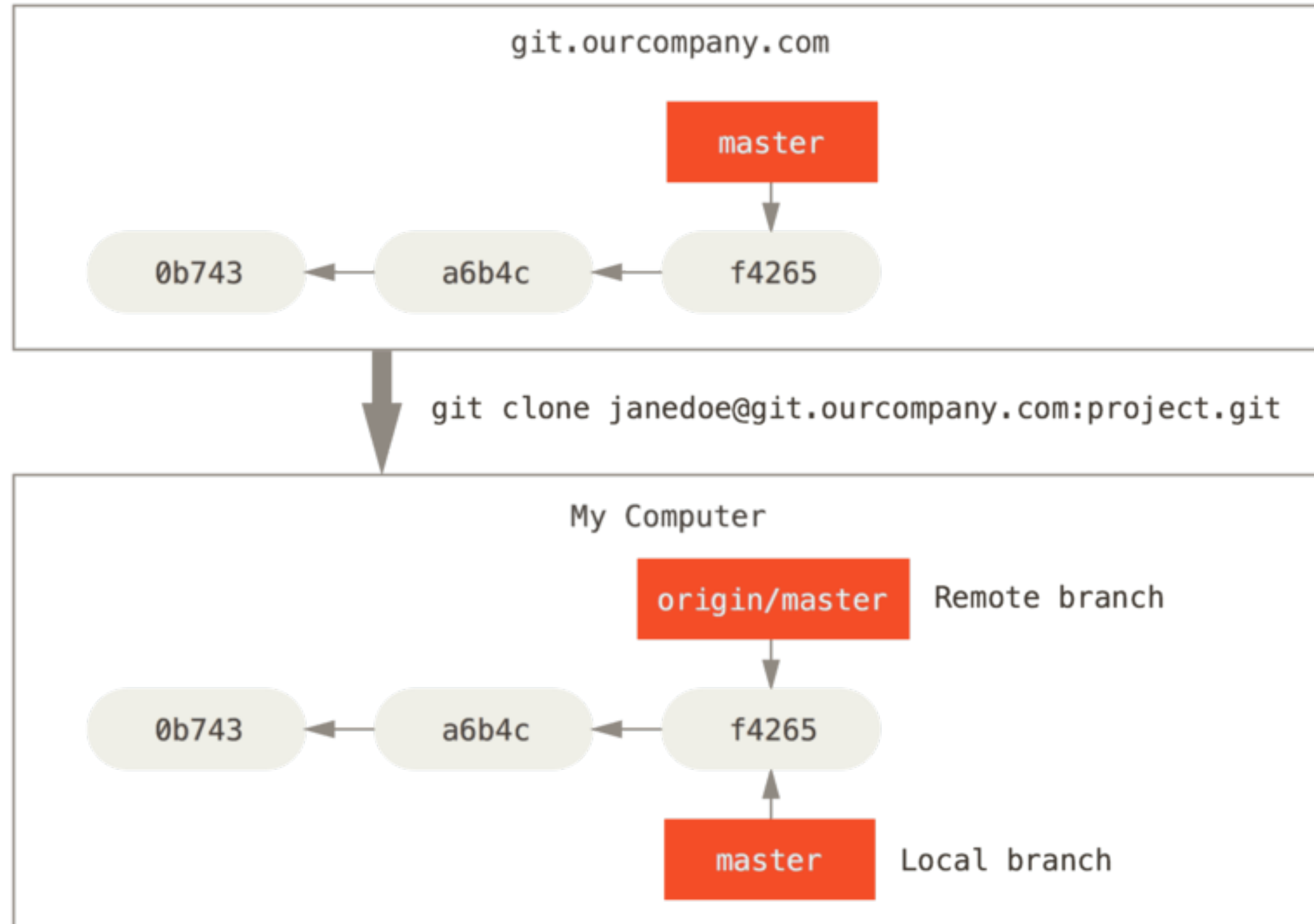
# Branches thématiques



À distance...



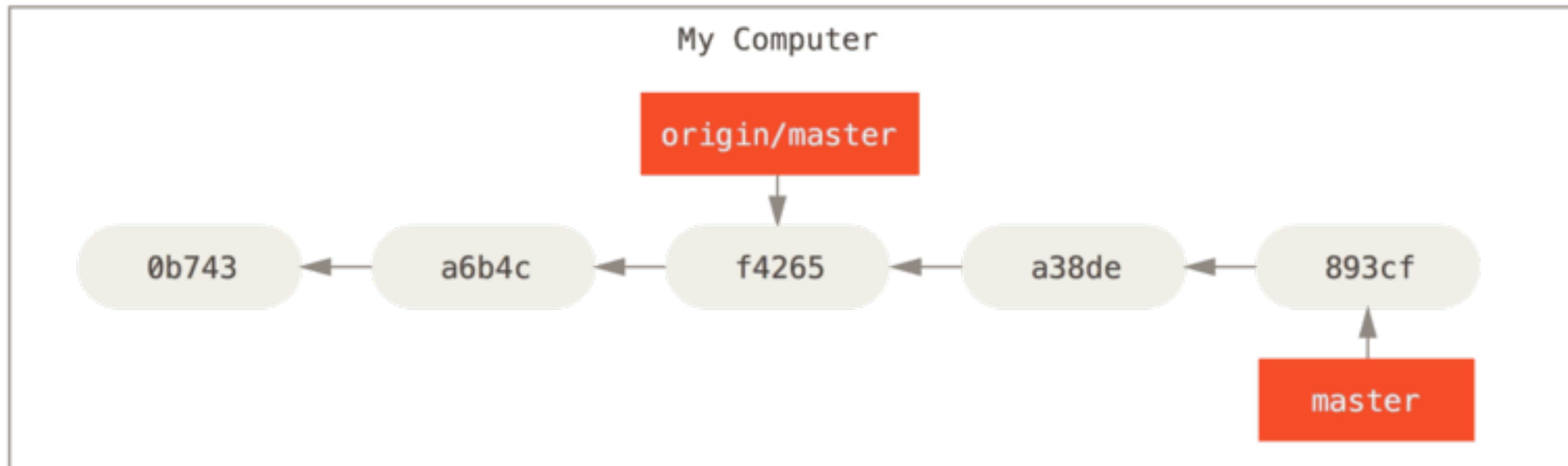
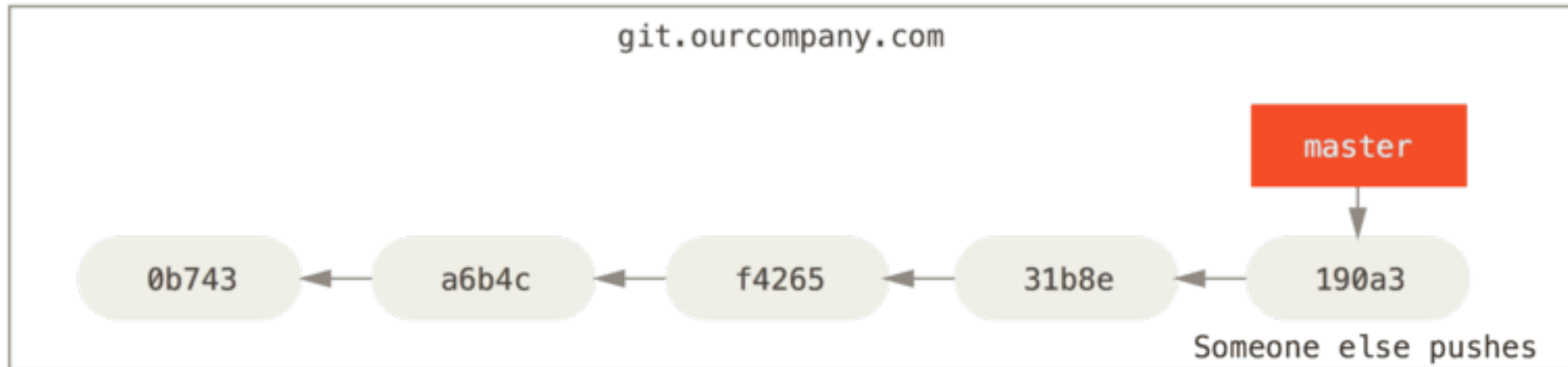
# Branches de suivi à distance



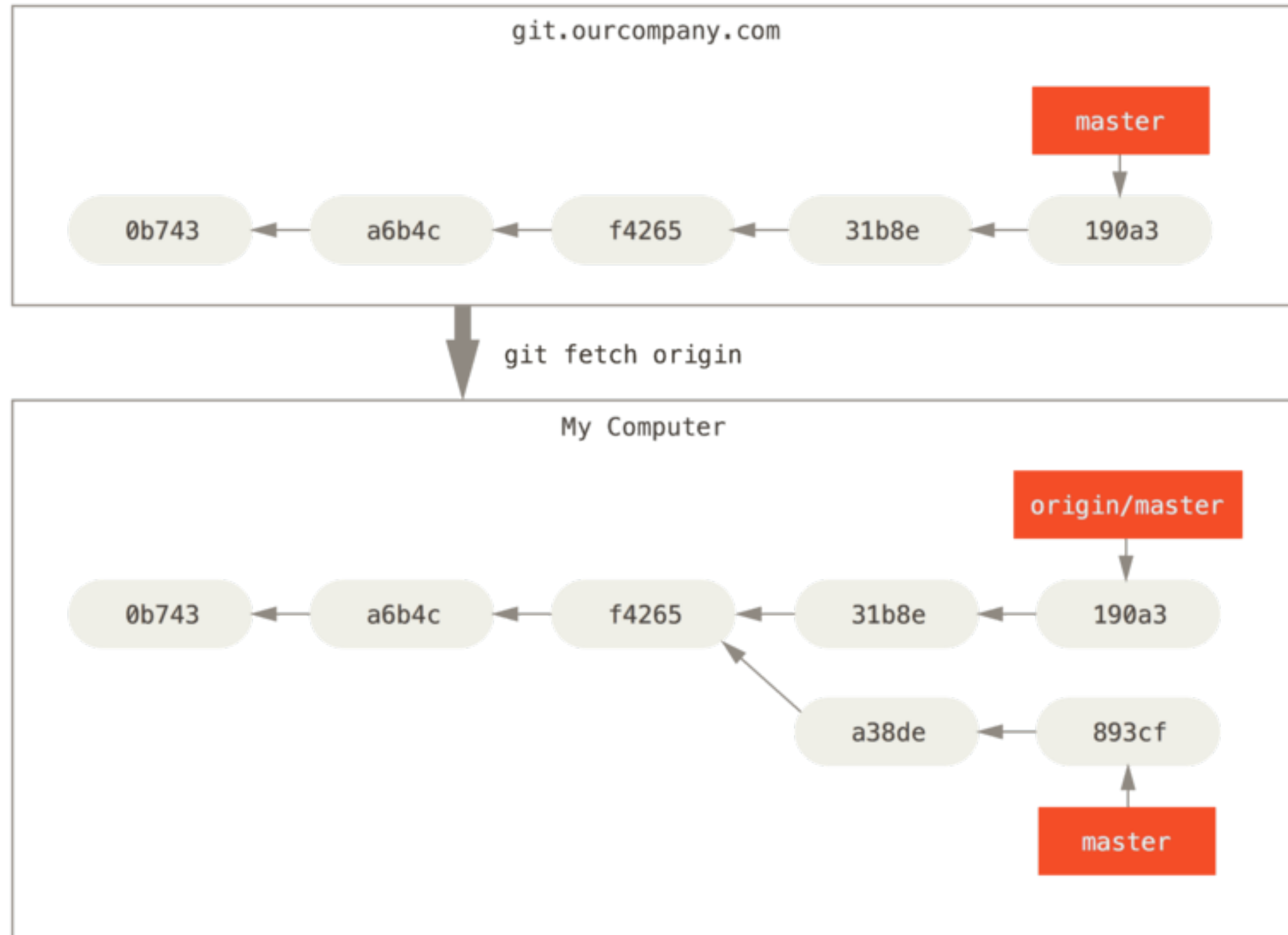
# Les commandes git (6)

- ◆ `git push`
  - ◆ Transmet la branche locale à la branche à distance
  - ◆ Si c'est la première fois que l'on `push`, il faut écrire `git push -u origin [branch]` pour créer le lien entre la branche locale et la branche sur le repo à distance surnommé `origin`
- ◆ `git fetch [alias]`
  - ◆ Copie localement les branches du repo à distance `alias`
- ◆ `git pull`
  - ◆ Effectue un `fetch` puis un `merge`

# Branches de suivi à distance (2)

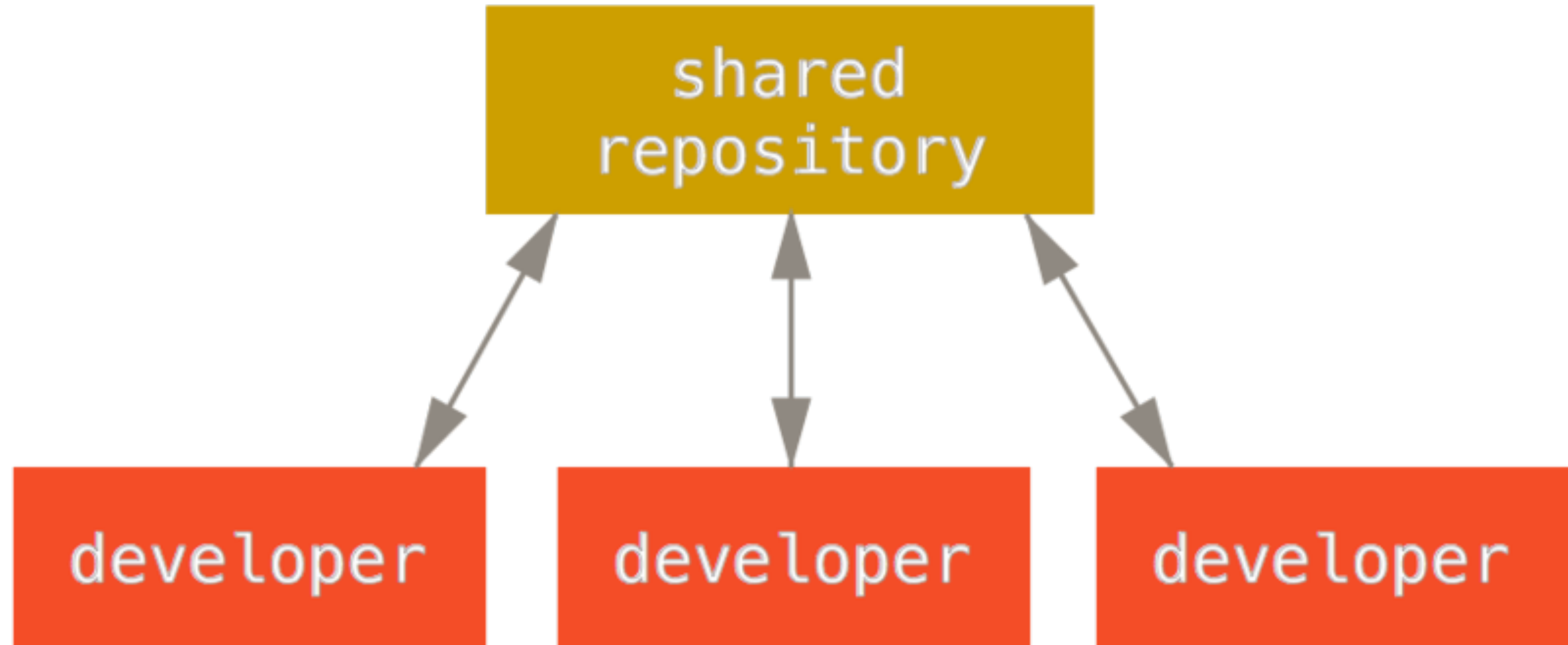


# Branches de suivi à distance (3)



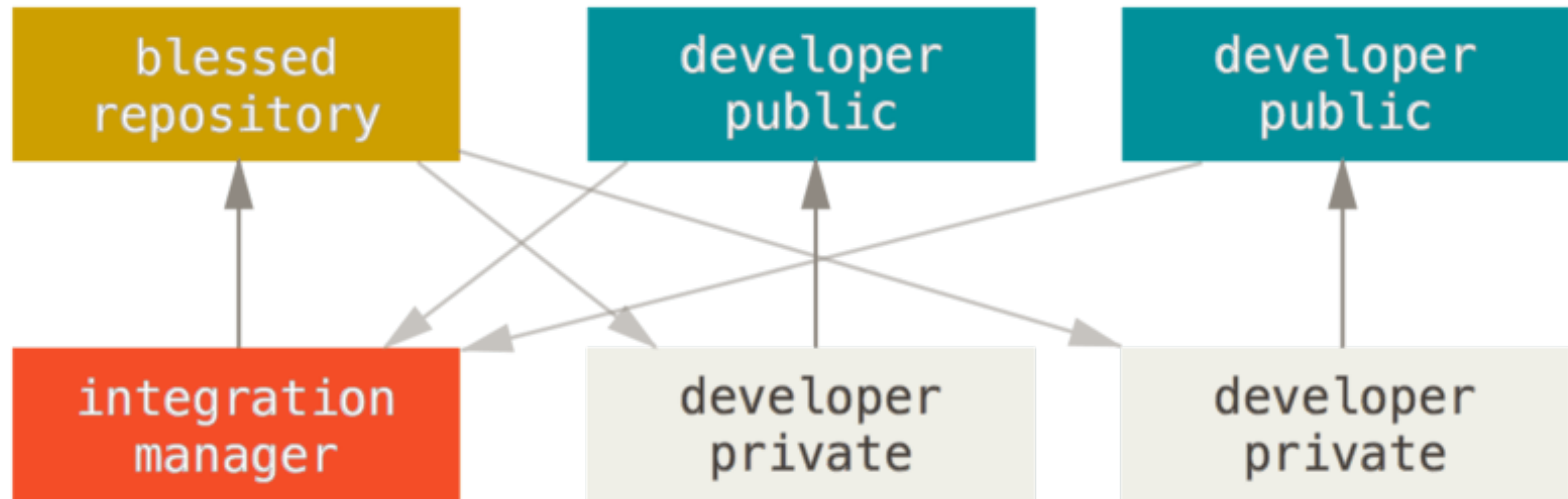
# Git distribué

# Gestion centralisée



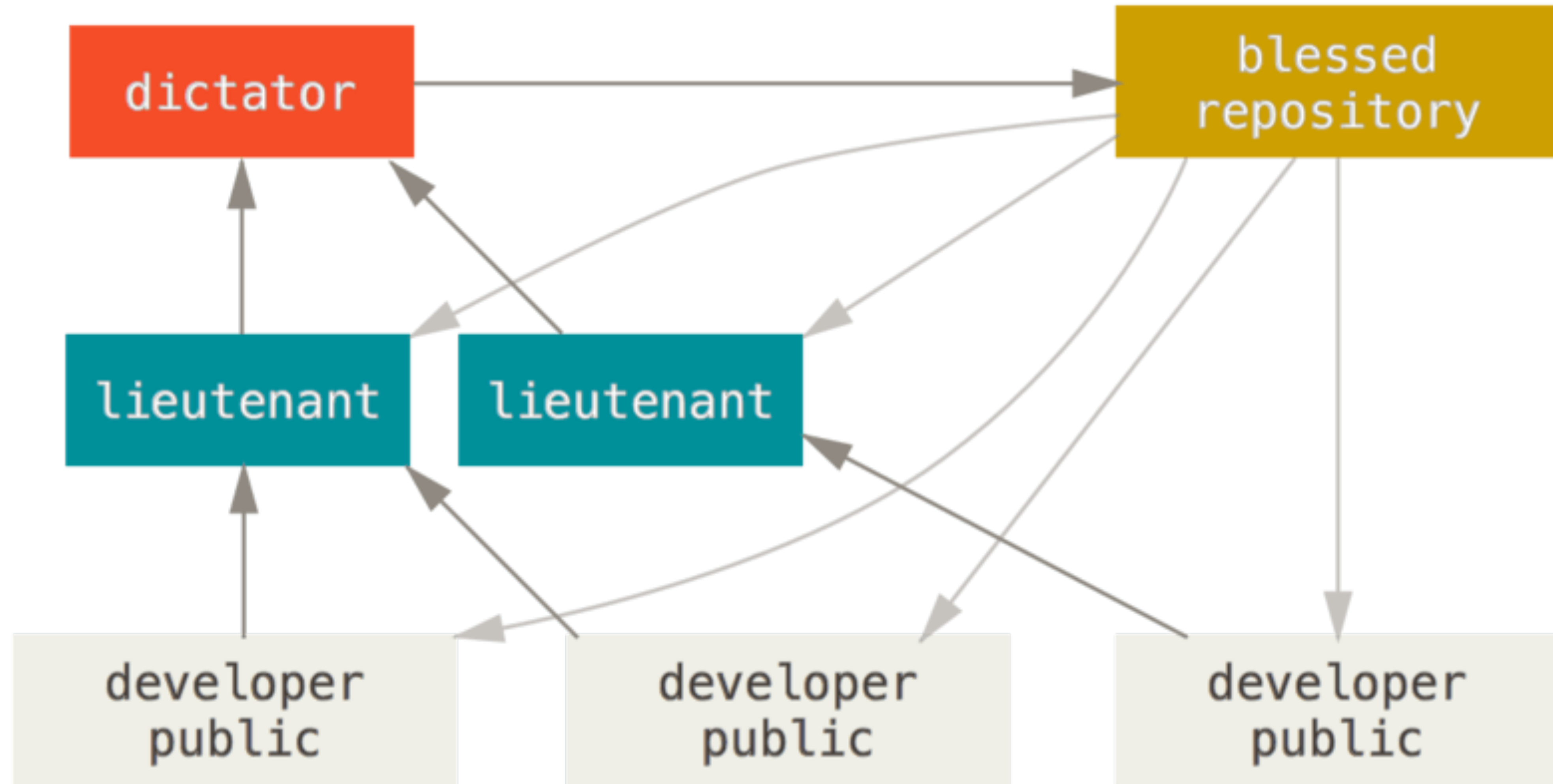


# Gestionnaire d'intégration





# Dictateur et lieutenants



# Et plus encore...

- ✦ Cette présentation ne couvre pas l'utilisation de fork et de pull\_request sur GitHub.
- ✦ Vous pouvez en apprendre plus sur ce sujet ici ...

<https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests>