

jeudi 07 décembre



git 4^e

Les branches

Créer un espace de travail local:

- **mkdir mon_projet_local**
- **cd mon_projet_local**
- **git init**

Vérifier sa configuration

- **git config --get-regexp alias**
 - **l -> log --oneline**
 - **ci -> commit**
 - **s -> status -s**
- **git config --get-regexp editor**

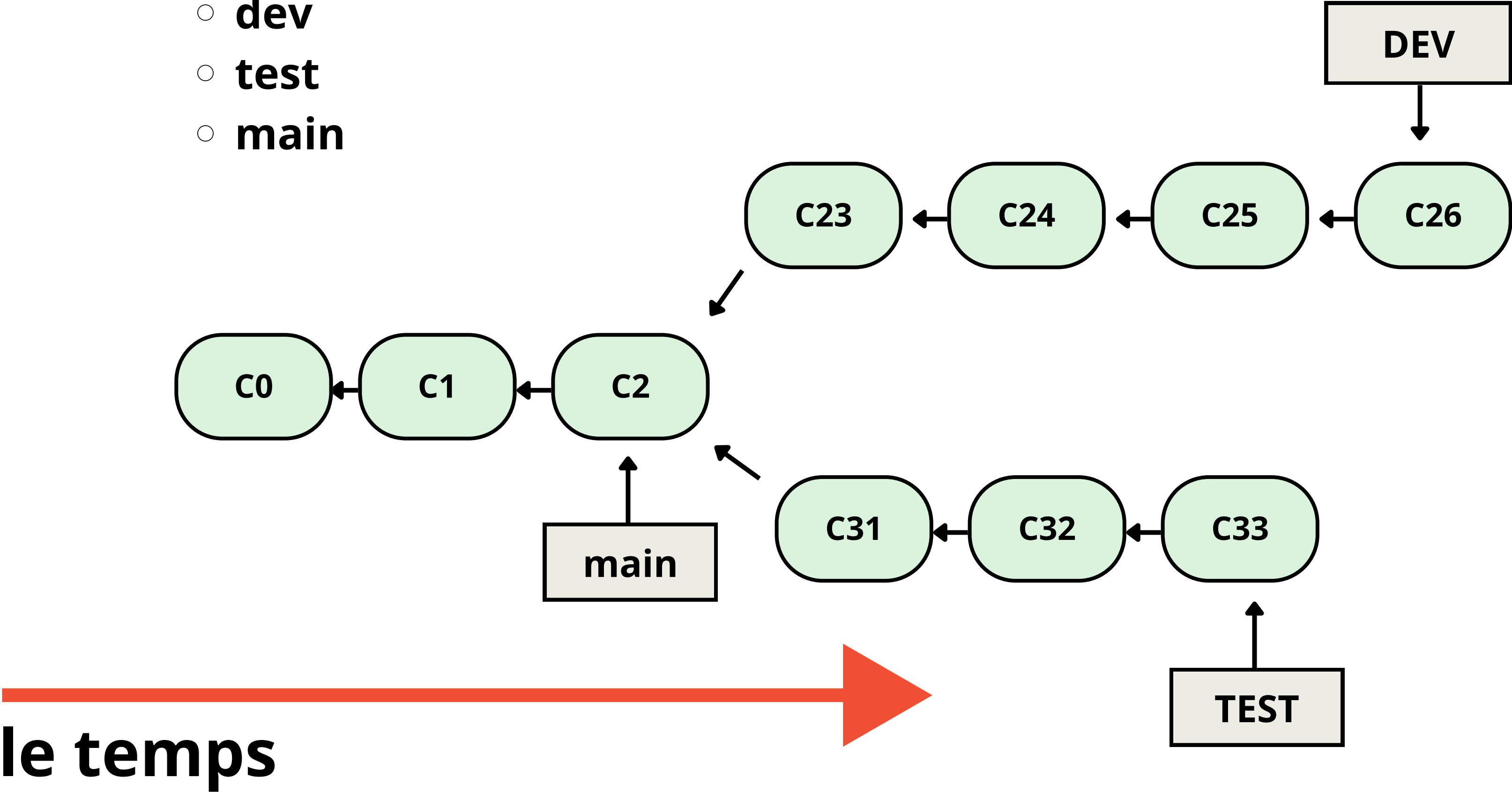
Pourquoi les branches:

Travailler en parallèle sur différents états du logiciel

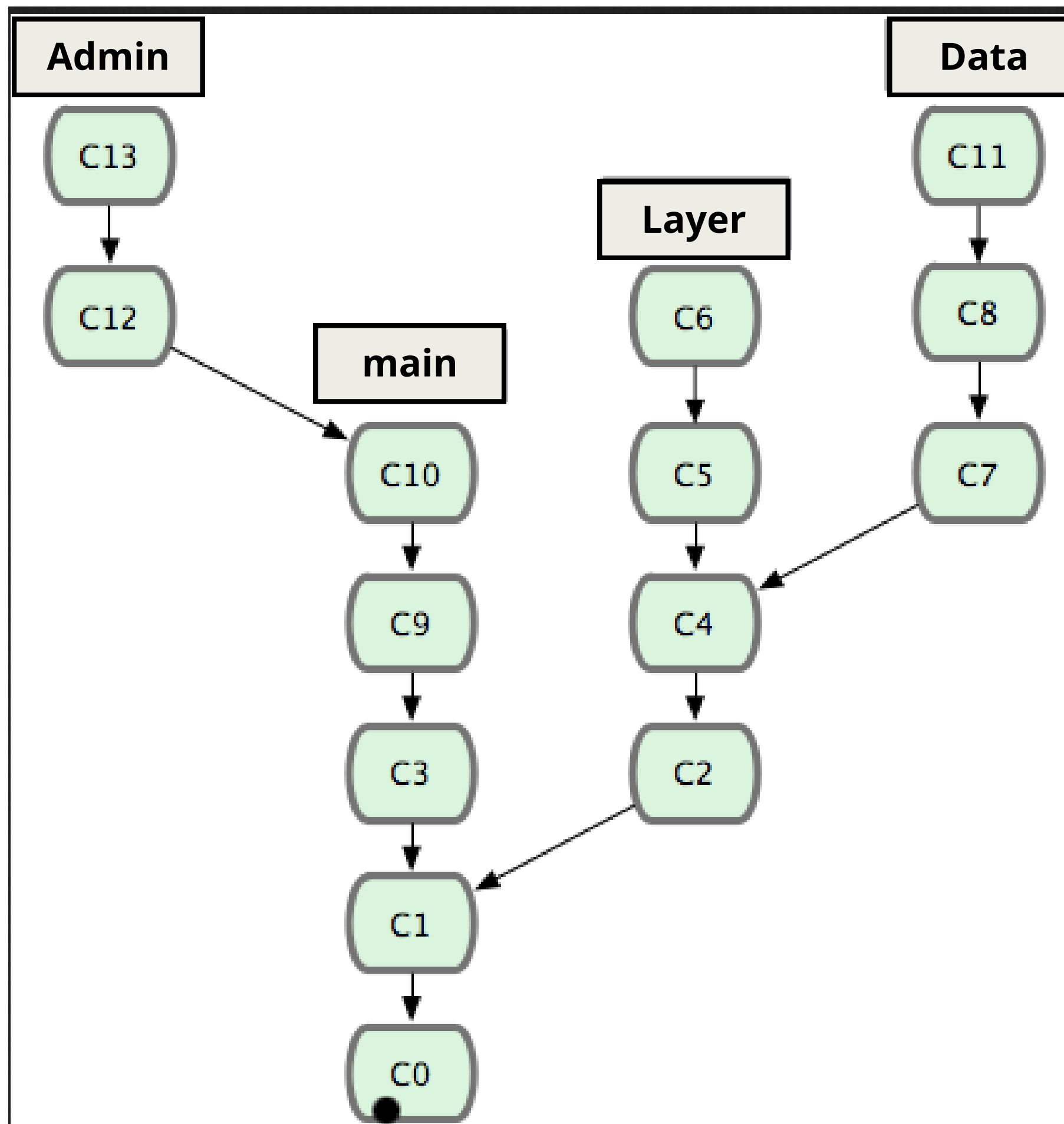
- **plusieurs versions :**
 - **dev**
 - **test**
 - **prod**
- **plusieurs fonctionnalités :**
 - **ajouter un layer**
 - **nouvelle source de données**
 - **interface admin**
- **Corriger un bug**
 - **sur la version de prod**
 - **en gardant test en l'état**

- **plusieurs versions :**

- **dev**
- **test**
- **main**

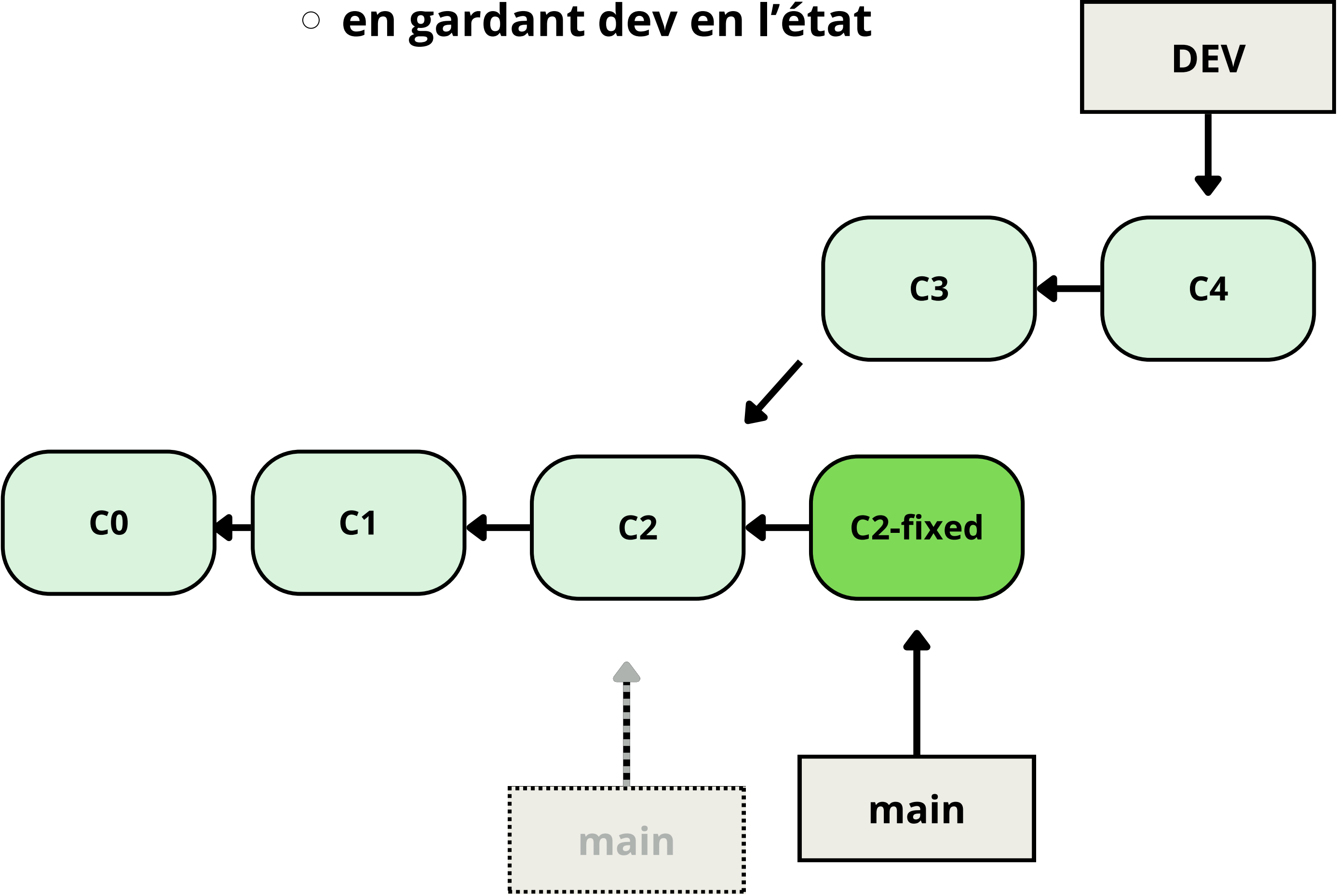


le temps



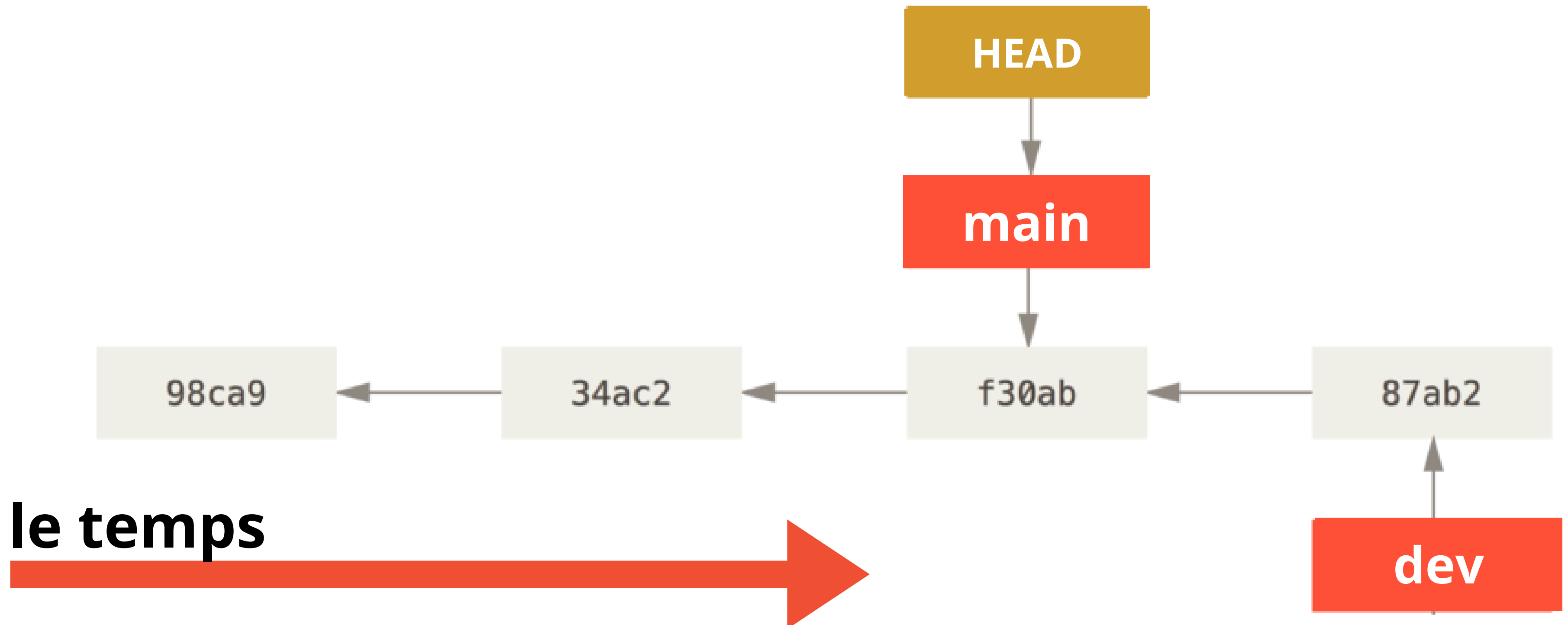
- **plusieurs fonctionnalités :**
 - **ajouter un layer**
 - **nouvelle source data**
 - **interface admin**

- **Corriger un bug**
 - **sur la version de prod**
 - **en gardant dev en l'état**



C'est quoi une branche

- une étiquette qui pointe vers un commit
- un contenu particulier du répertoire de travail



créer une nouvelle branche

- **vérifier le contenu de notre répertoire:**

- **dir .**
- **git s**
- **git l**

- **créer une branche nommée "dev"**

- **git branch dev**

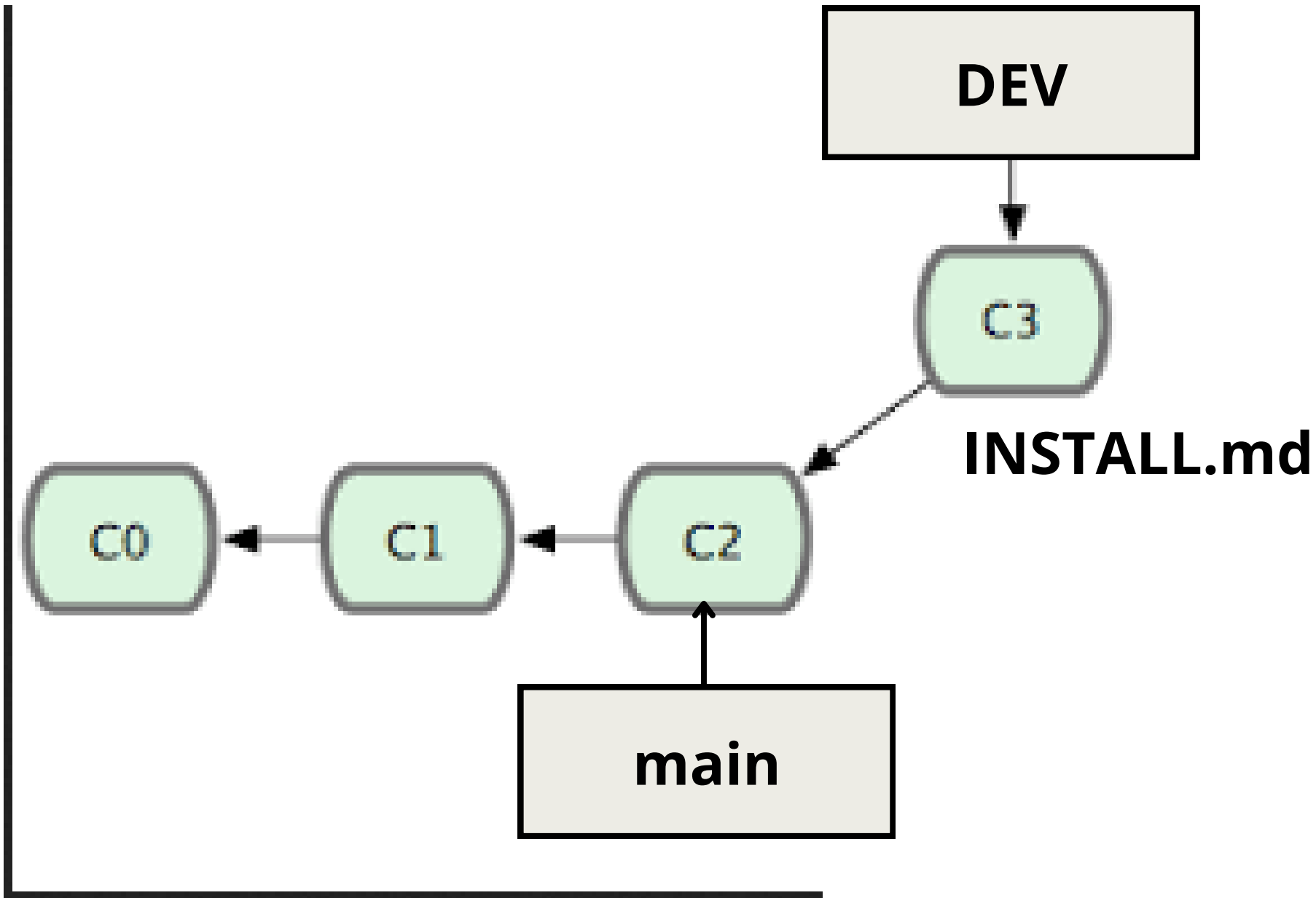
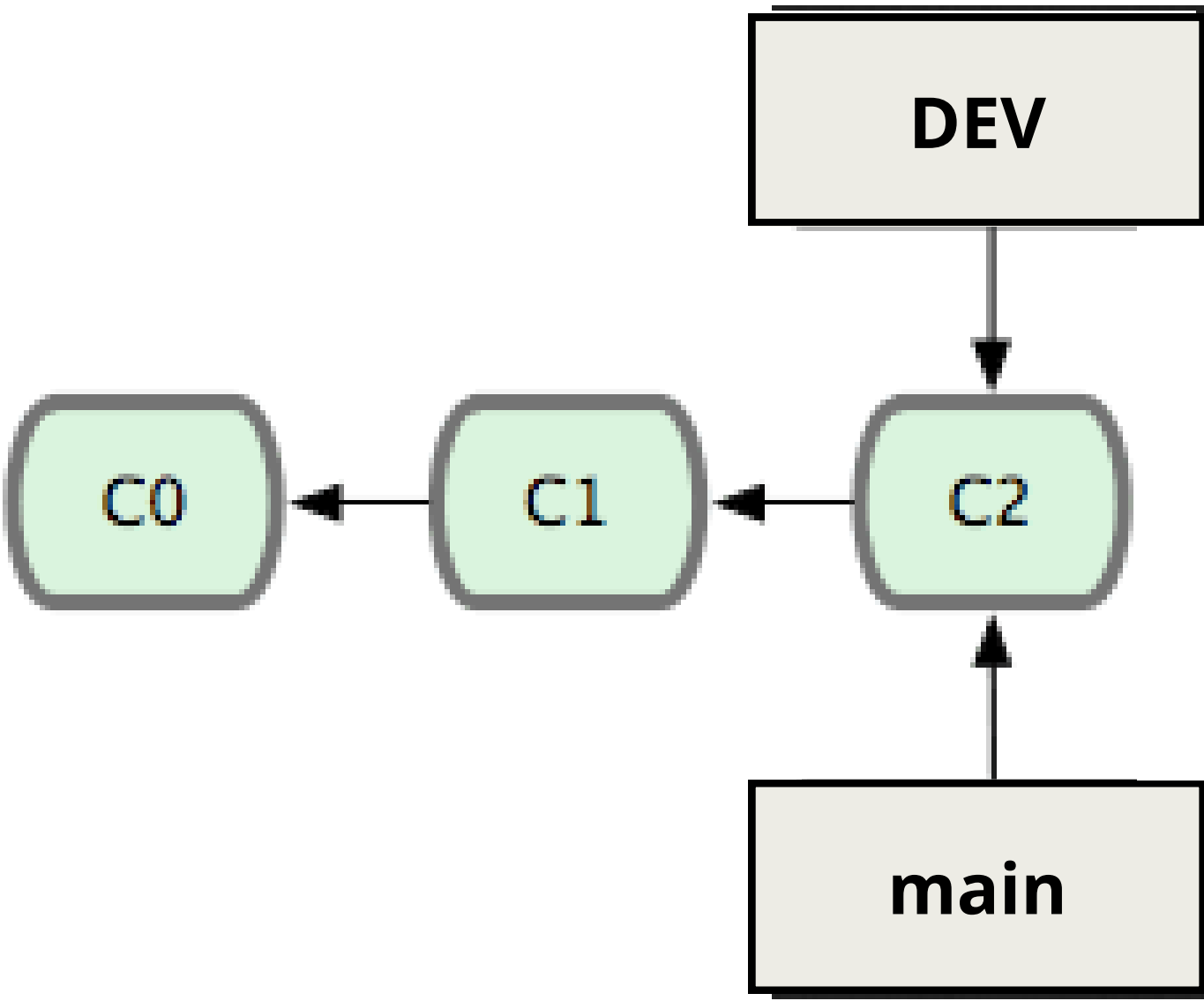
- **s'y déplacer**

- **git switch dev**

- **ajouter un fichier "INSTALL.md"**

- **echo "hello" >> INSTALL.md**
- **git add INSTALL.md; git ci -m "Un msg"**
- **git l**

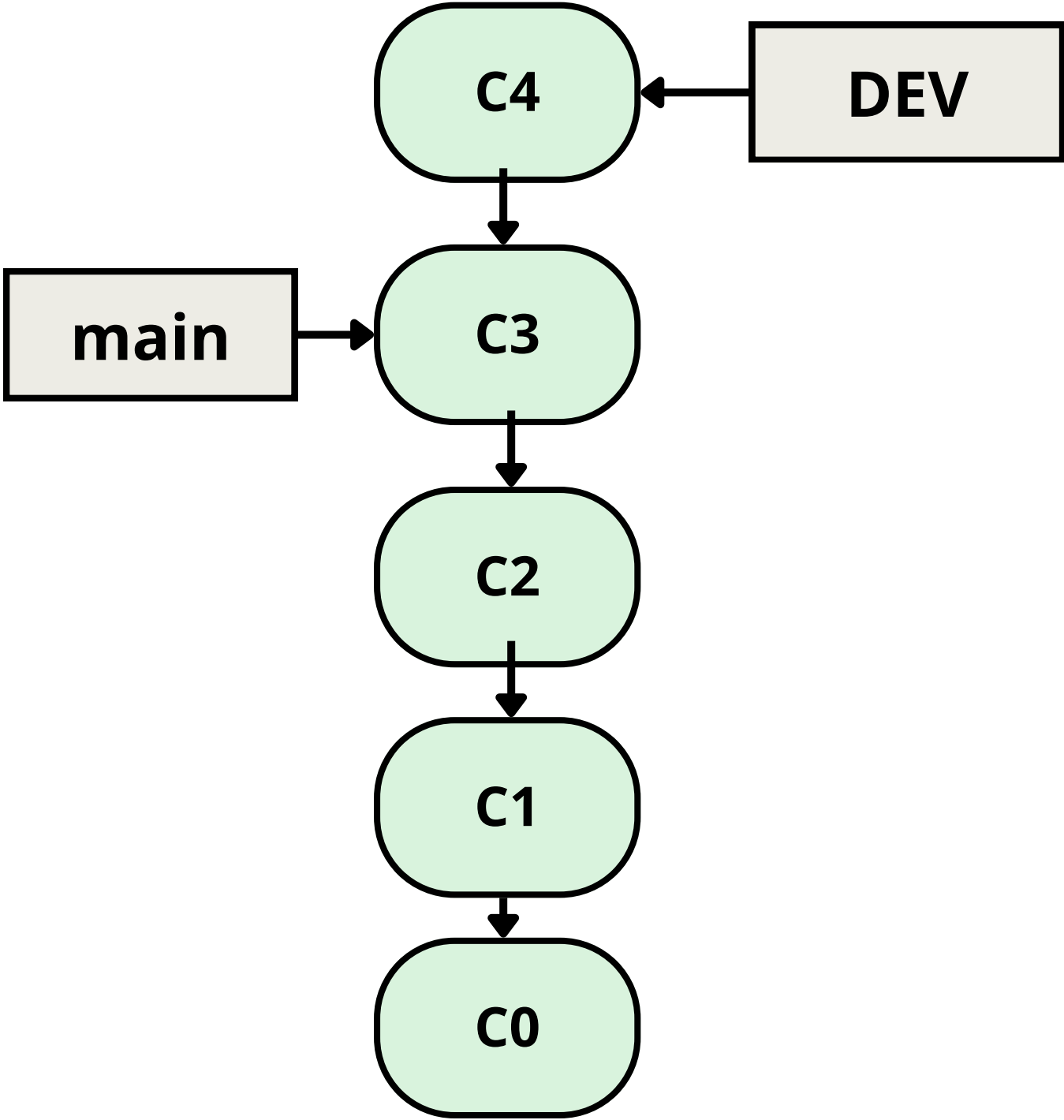
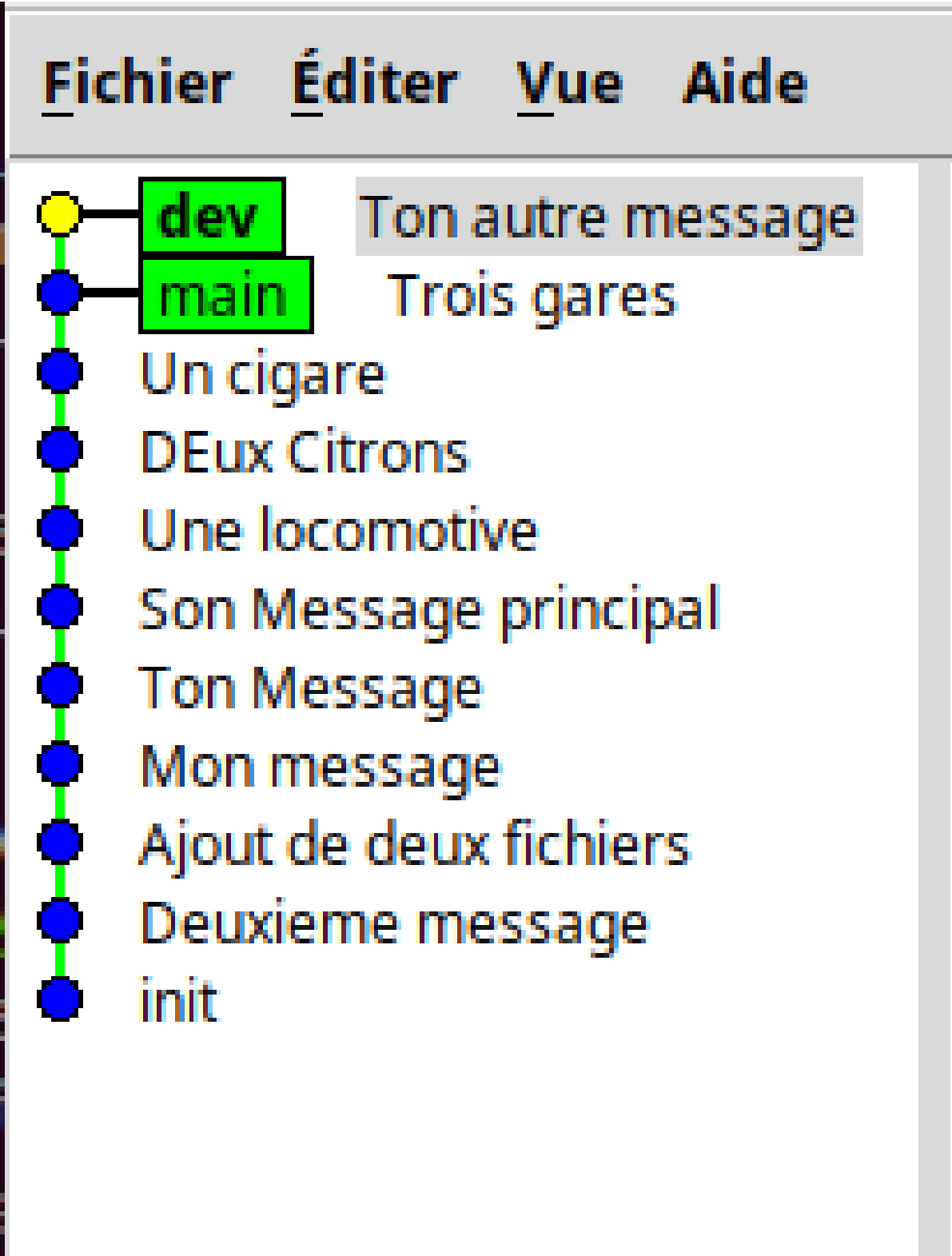
Créer la branche DEV



Committer dessus

gitk --all

le temps ↑



rebasculer sur main et commiter 1 fois

- `git switch main`
- `echo '{ "hello": "world" }' >> hello.json`
- `git add hello.json`
- `git ci -m "Add new json config file"`

quelles différences sur les workspaces ?

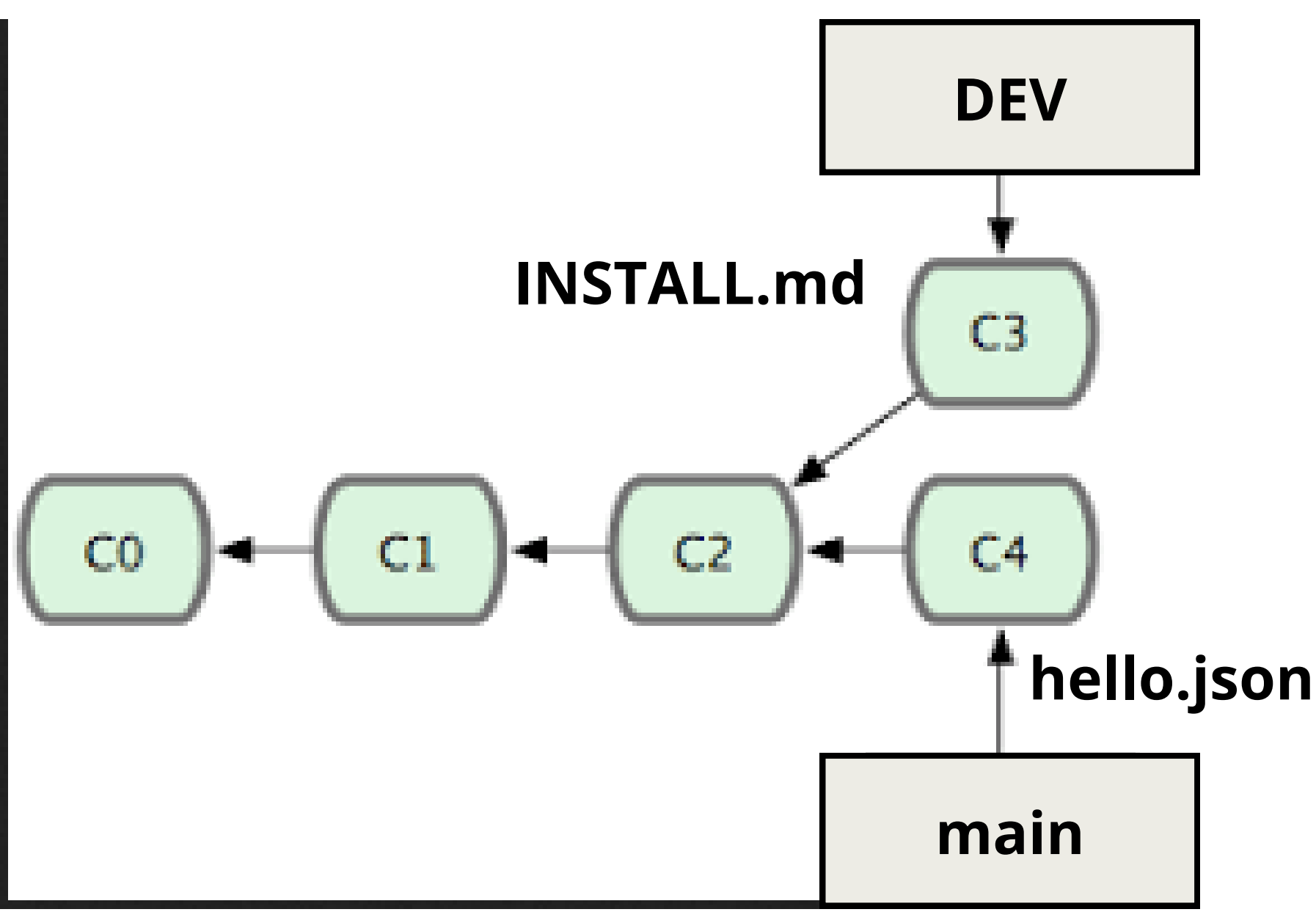
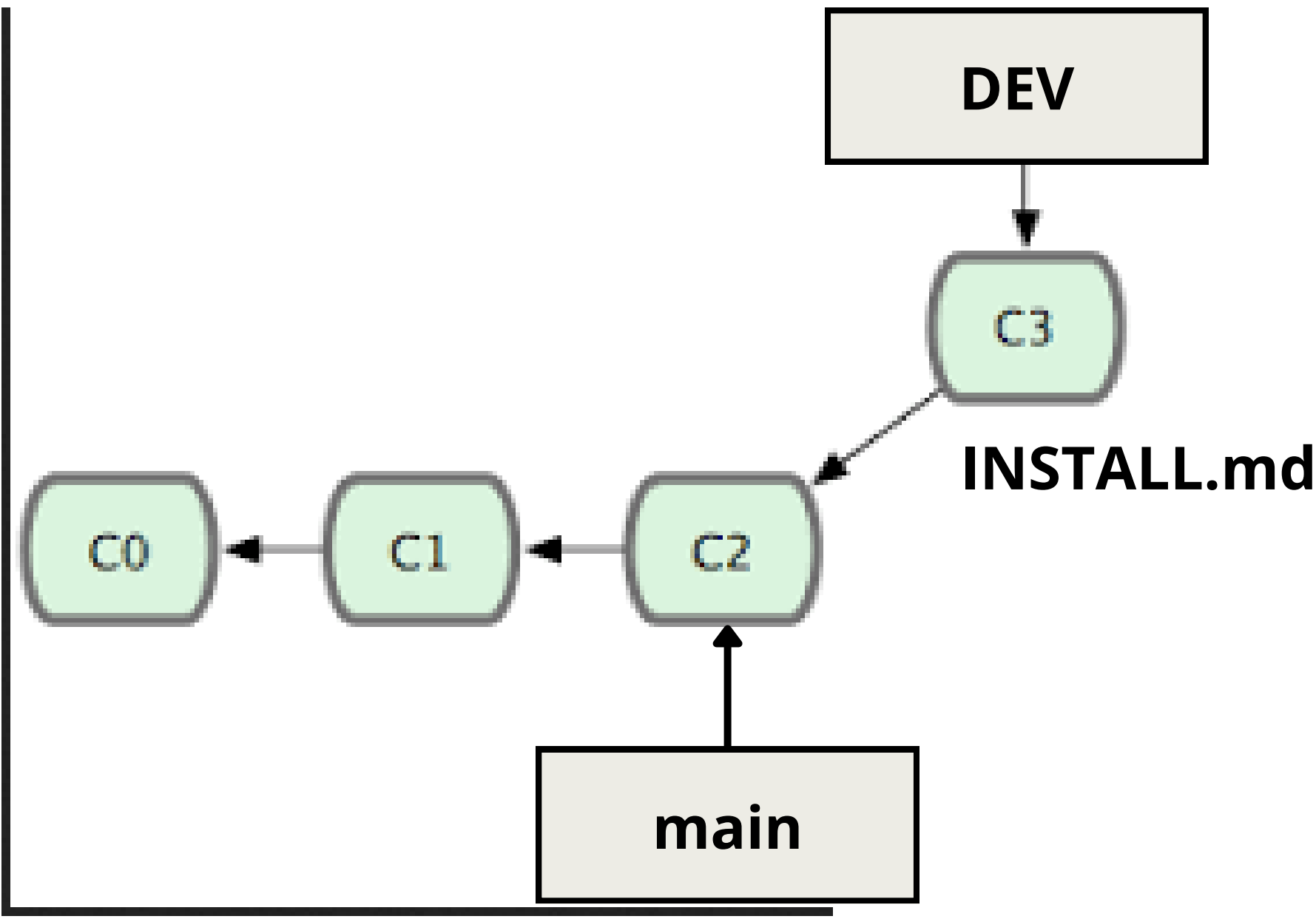
- `git br (-> check "main")`
- `dir . (-> check "hello.json")`
- `git switch dev`
- `dir . (-> no more "hello.json" new "INSTALL.md"`

rebasculer sur main et commiter

AVANT

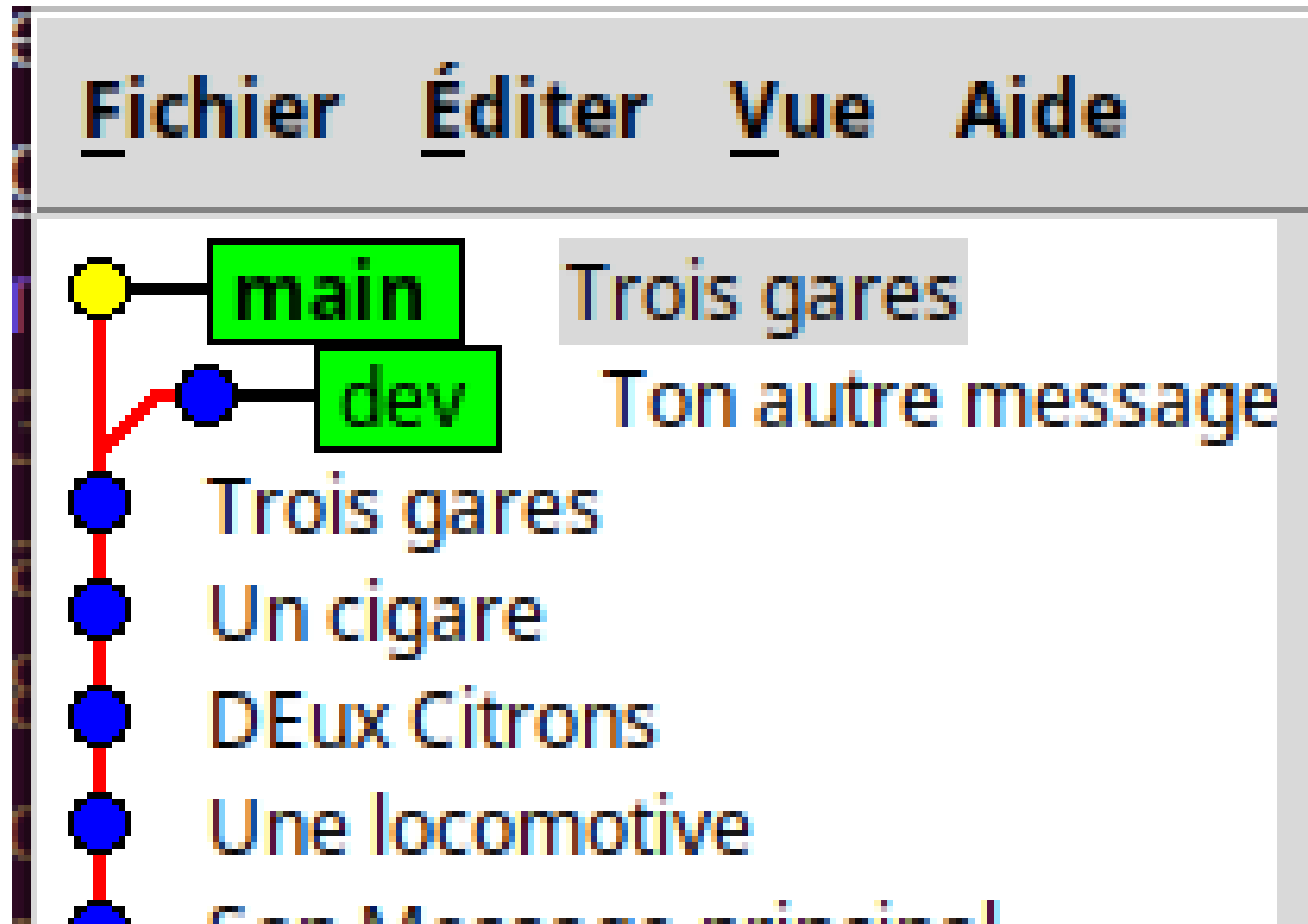


APRÈS



rebasculer sur main et commiter

gitk --all



Fusionner "main" et "dev"

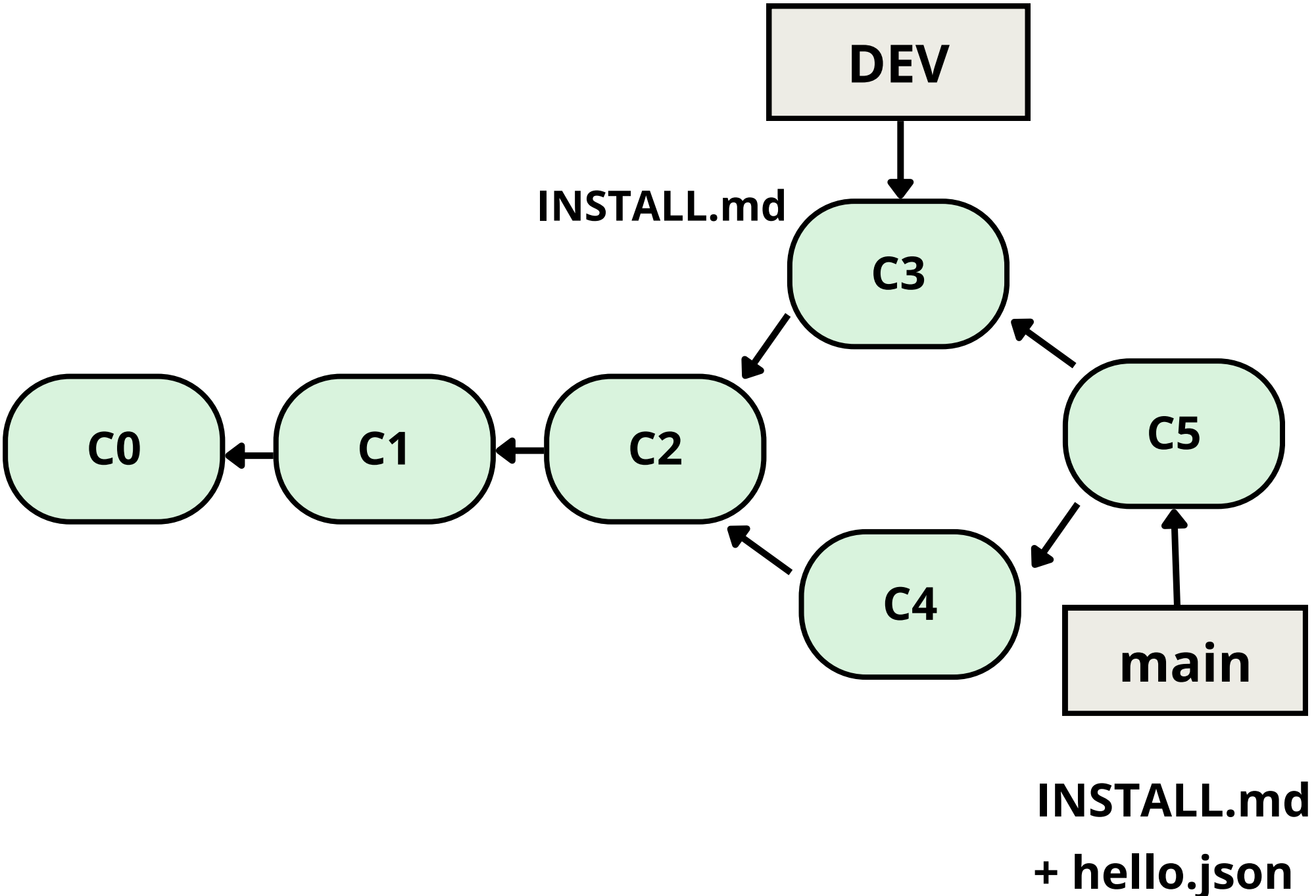
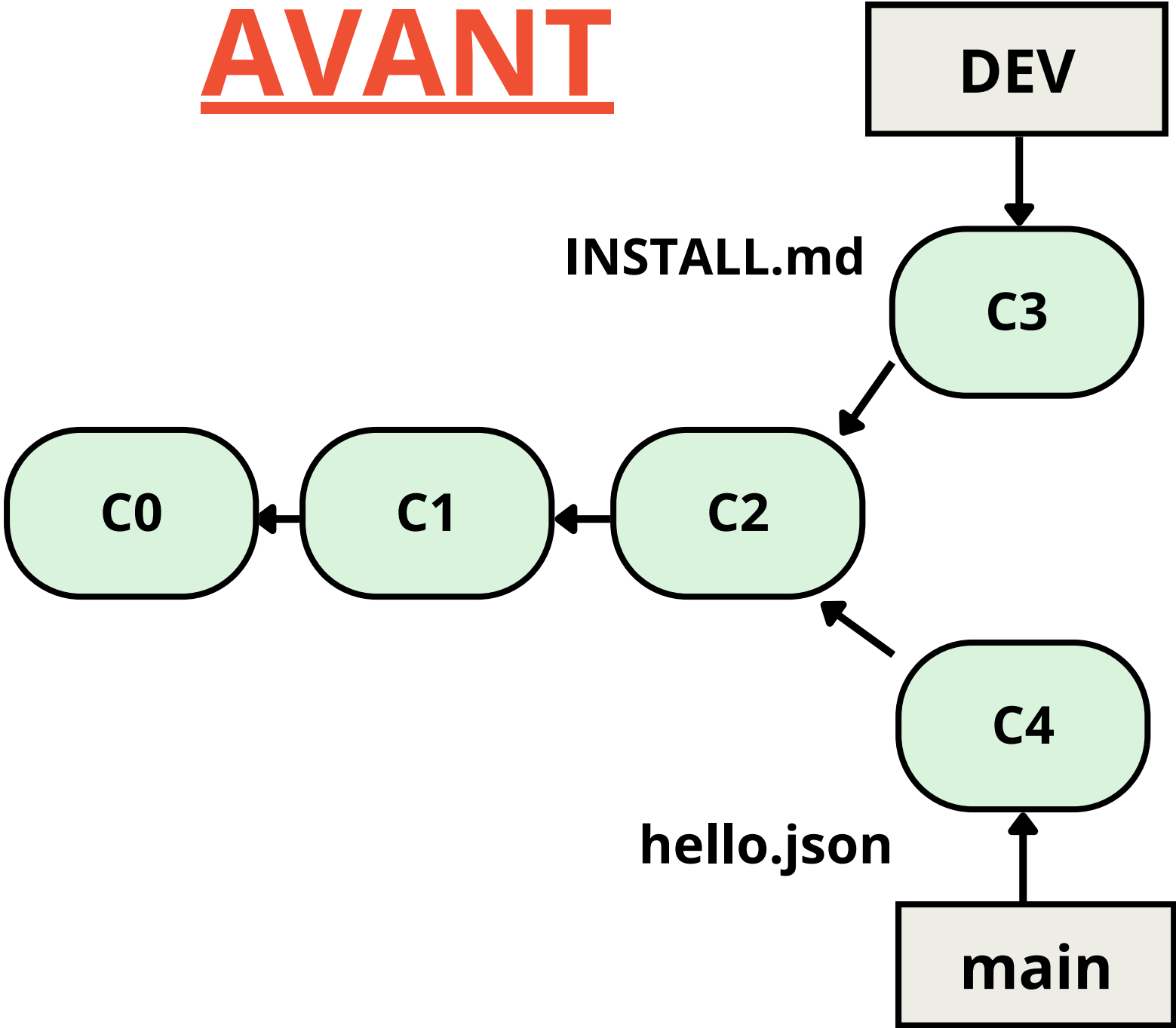
Récupérer dans **main** ce que l'on avait dans **DEV**
À Savoir: le fichier "INSTALL.md"

- **git switch main**
- **dir . (-> check no "INSTALL.md")**
- **git merge dev**
- **dir . (-> check "INSTALL.md" && "hello.json")**
- **git l**
- **git switch dev**
- **dir . (-> check "INSTALL.md" but no "hello.json")**

Fusionner "main" et "dev"

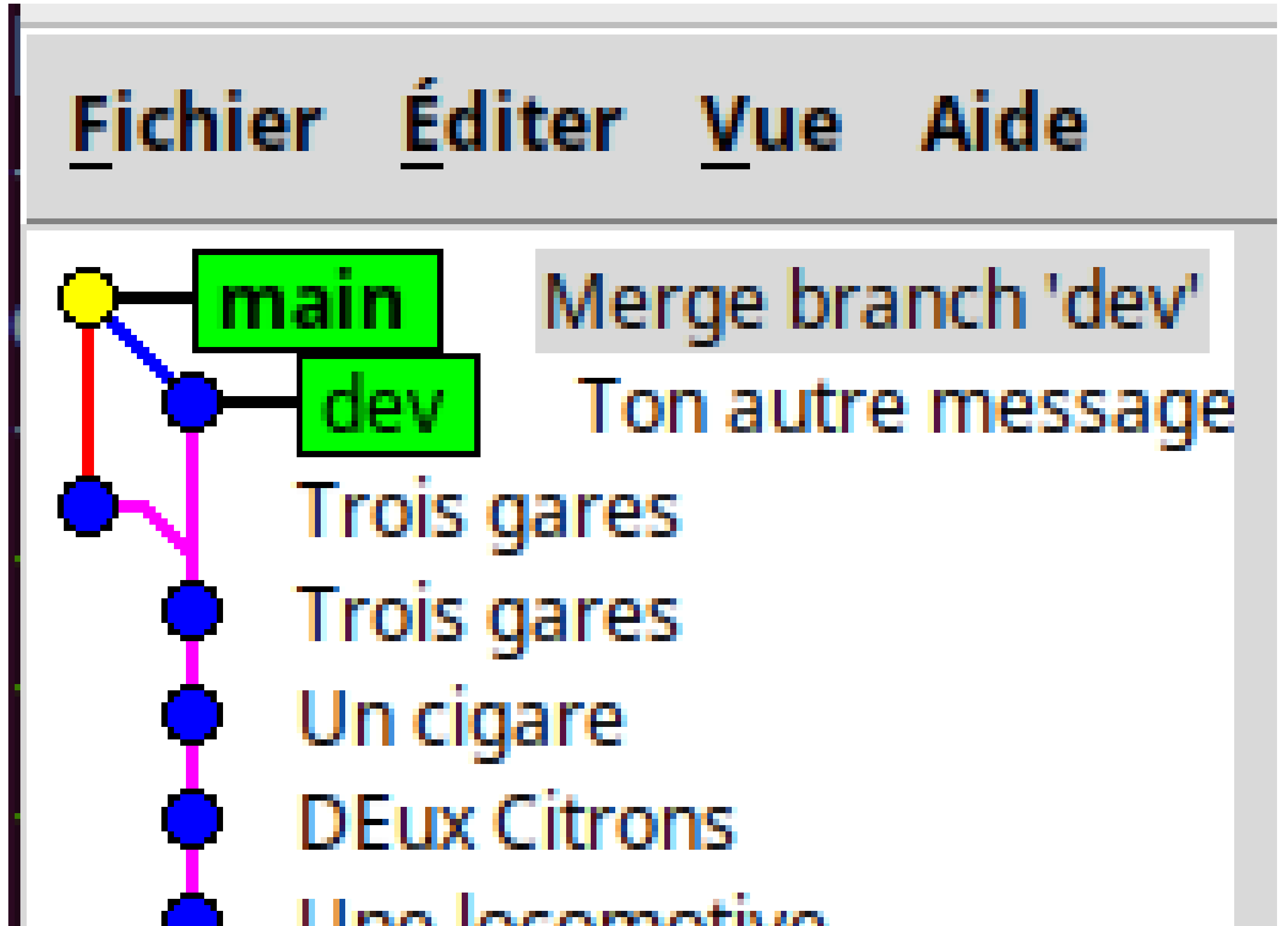
APRÉS

AVANT



Fusionner "main" et "dev"

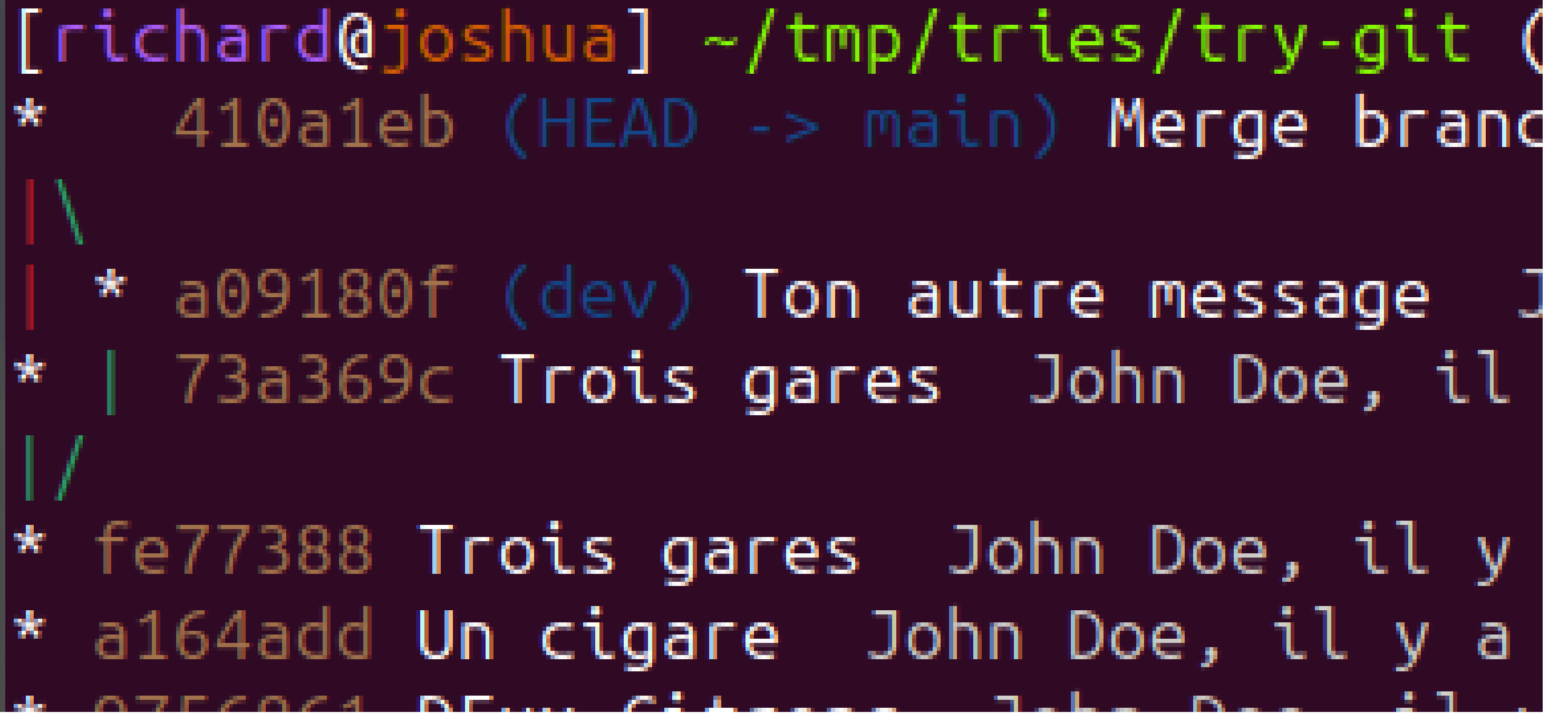
gitk --all



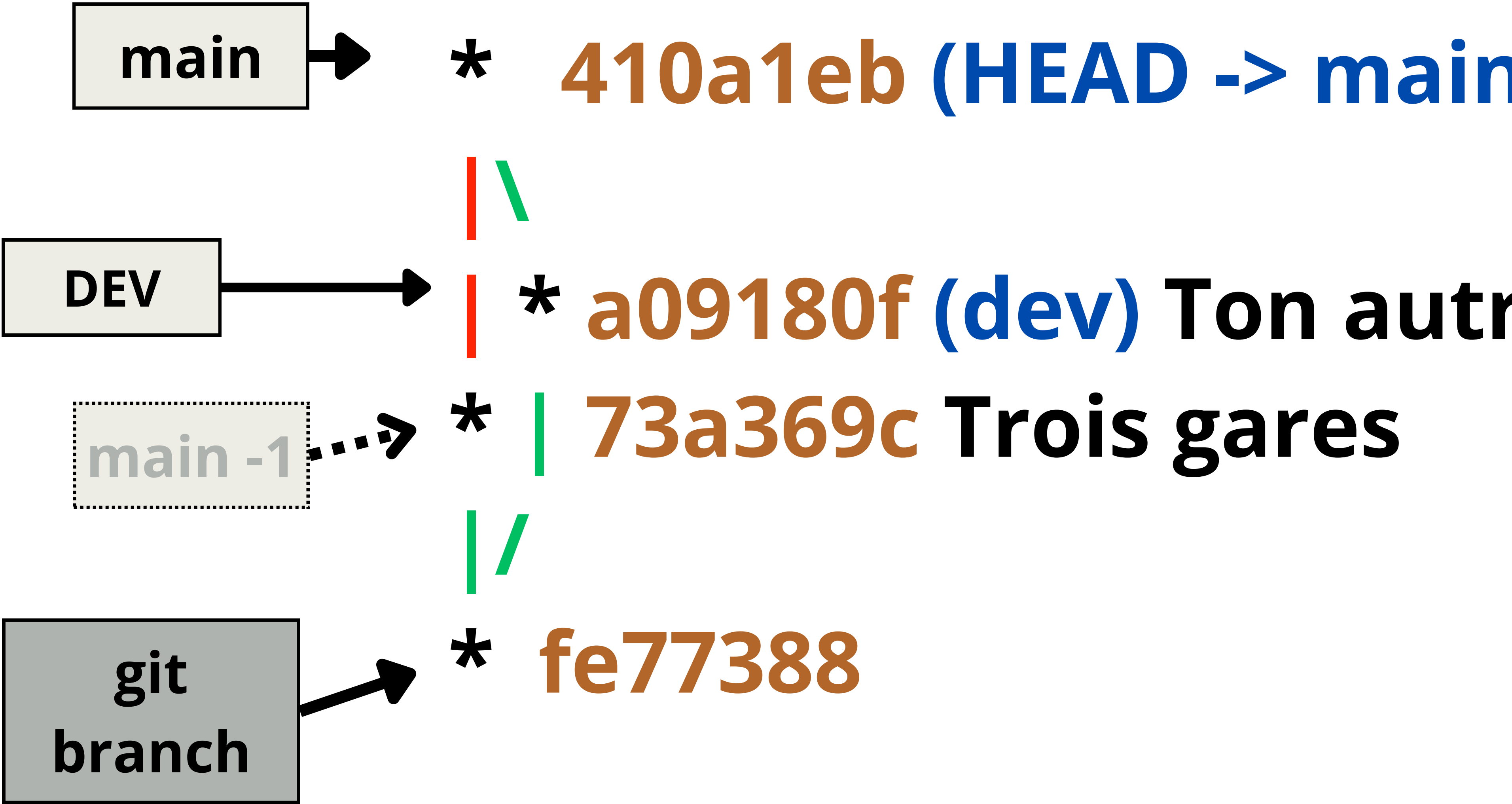
git config alias.lg log --graph --oneline

git lg

le temps



```
[richard@joshua] ~/tmp/tries/try-git (
* 410a1eb (HEAD -> main) Merge branch
| \
| * a09180f (dev) Ton autre message J
* | 73a369c Trois gares John Doe, il
| /
* fe77388 Trois gares John Doe, il y
* a164add Un cigare John Doe, il y a
* 0756061 Deux Cigarettes John Doe, il y
```



Le Fast Forward (“avance rapide”)

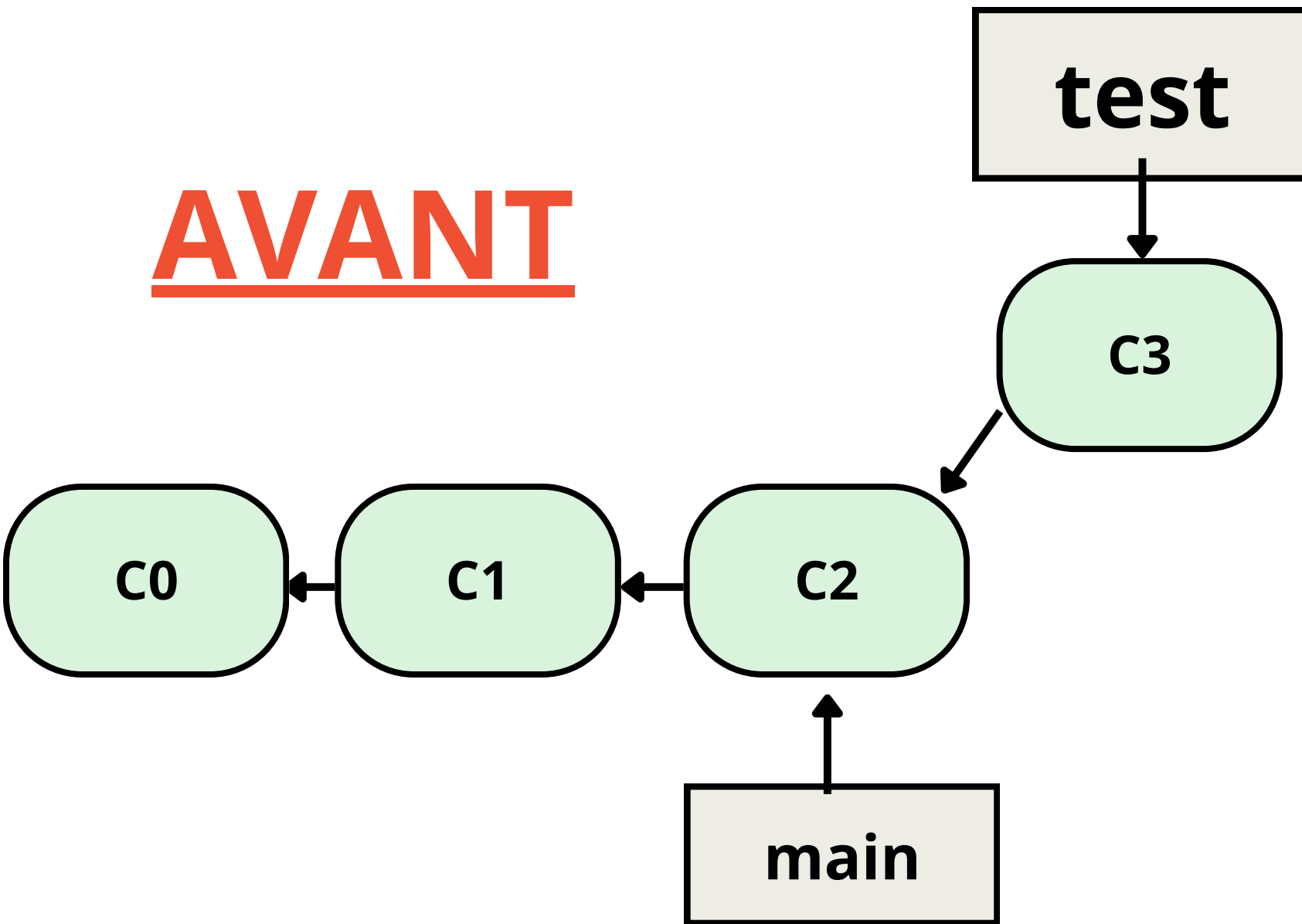
ou le merge rapide
et sans douleur

git merge --ff

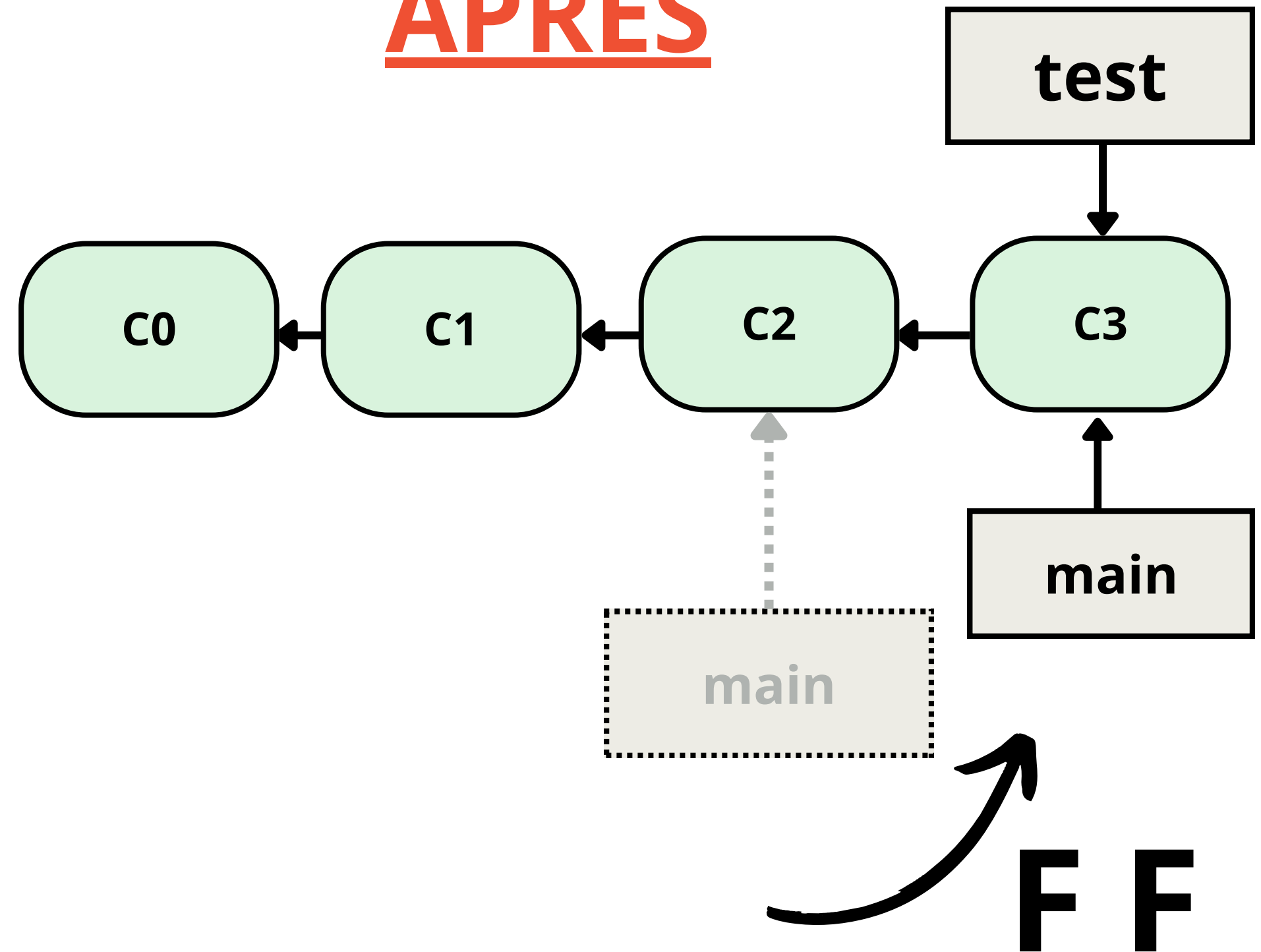
- **git switch main**
- **git switch -c test**
- **echo “c’est un test” >> test.txt**
- **git add test.txt; git ci -m “test”**
- **git switch main**
- **git merge test --ff**

Le Fast Forward

AVANT



APRÈS



Le NO Fast Forward

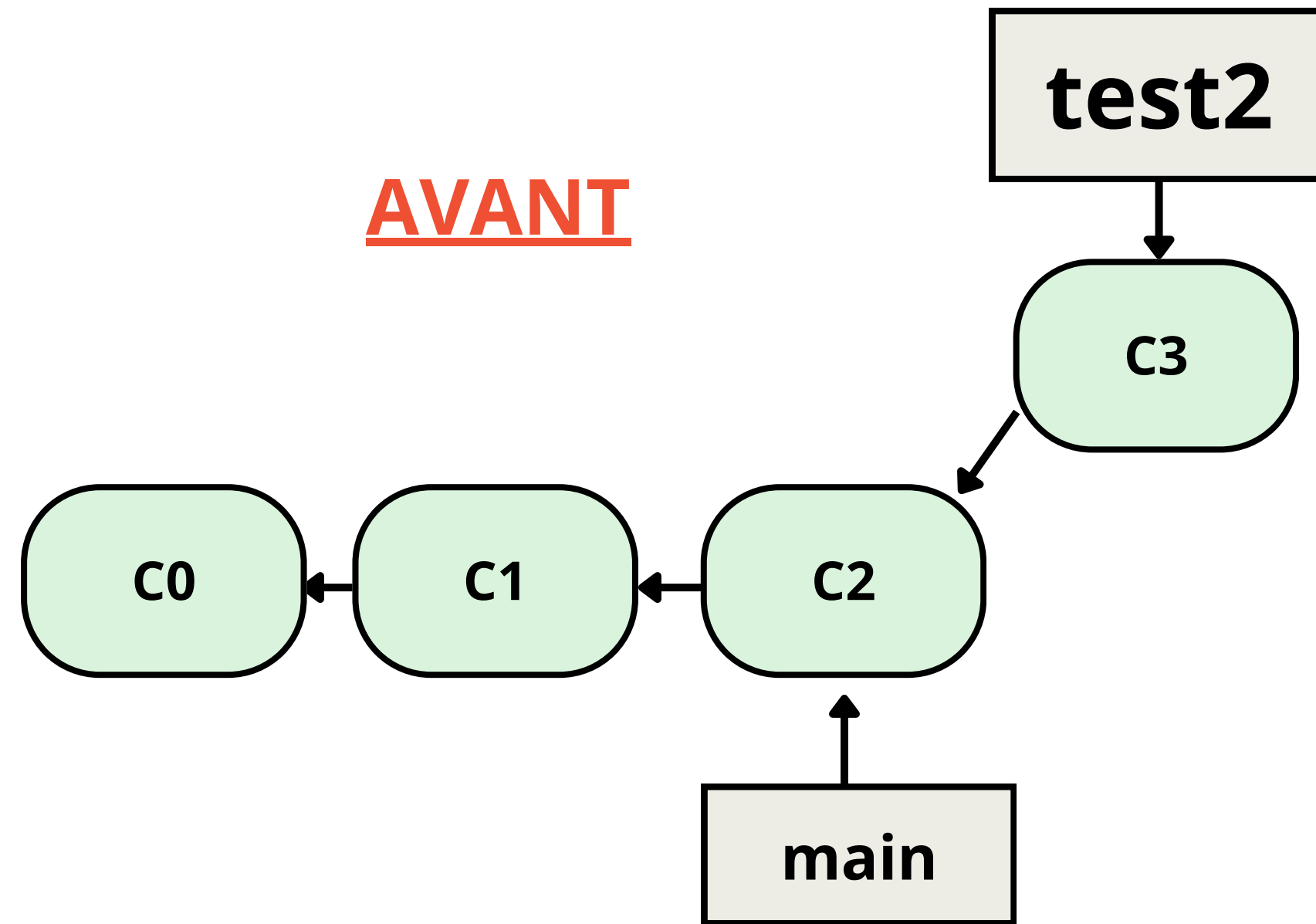
```
git merge --no-ff
```

crée un commit de merge forcé même quand un fast forward est possible

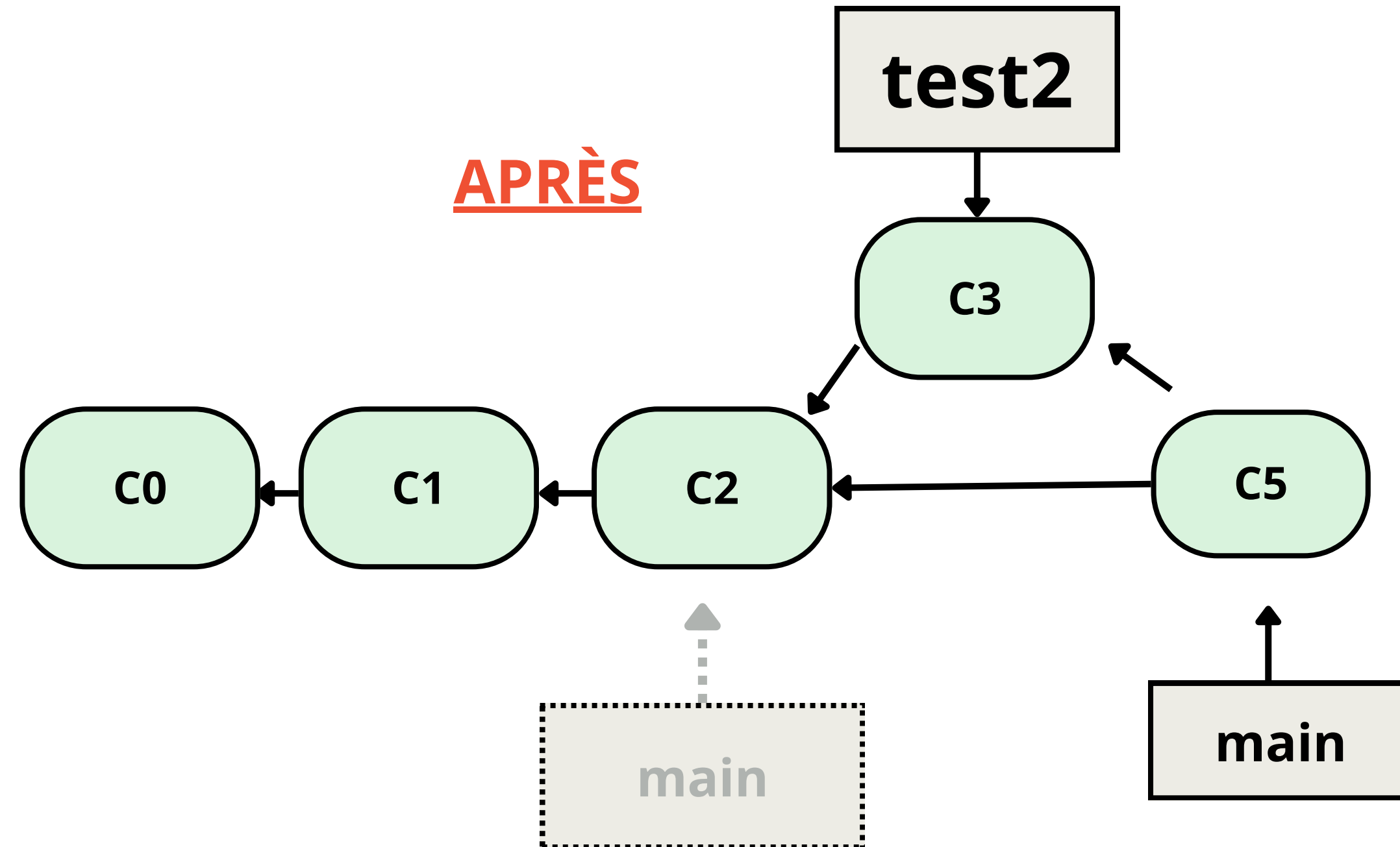
- **git switch main**
- **git switch -c test2**
- **echo "Deuxiem test" >> test2.txt**
- **git add test2.txt; git ci -m "test2"**
- **git switch main**
- **git merge test2 --no-ff**

Le NO Fast Forward

AVANT



APRÈS



Le diff

```
git switch main
```

Editer le fichier “README.md”
supprimer, rajouter des lignes.

```
git diff
```

Le diff

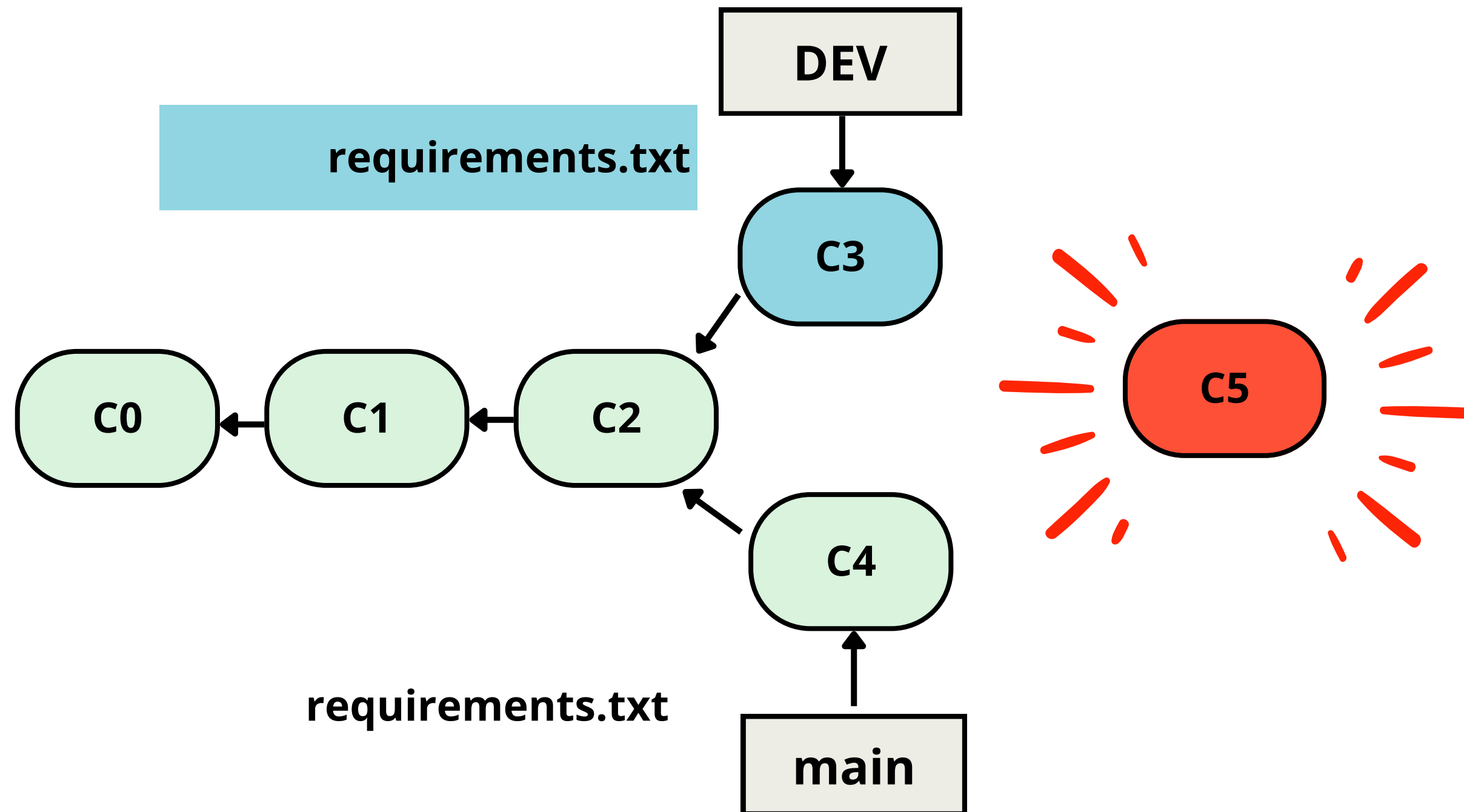
-Suppressions

+Ajouts

```
[richard@joshua] ~/tmp/tries/try-git (mainM?)$ git diff
diff --git a/README.md b/README.md
index b5eeeb4..e69b94d 100644
--- a/README.md
+++ b/README.md
@@ -1,7 +1,6 @@
 hoho
 ohoh
 hello
-hello
-hou
 hou
 trois gares
+Bonjour Monde
(END)
```


Le conflit: ou le merge impossible

lorsque deux branches ont des
modifications incompatibles
(les mêmes fichiers et les mêmes lignes)



Le conflit de merge : mise en place

```
git switch main
```

Editer un nouveau fichier "requirements.txt"

Flask==2.2.2

gunicorn==20.1.0

Jinja2==3.1.2

```
git add requirements.txt
```

```
git ci -m "New dependencies file"
```

Le conflit de merge : branching

```
git switch -c upgrade_deps
```

Editer le fichier "requirements.txt"

Flask==2.2.3



gunicorn==20.1.0

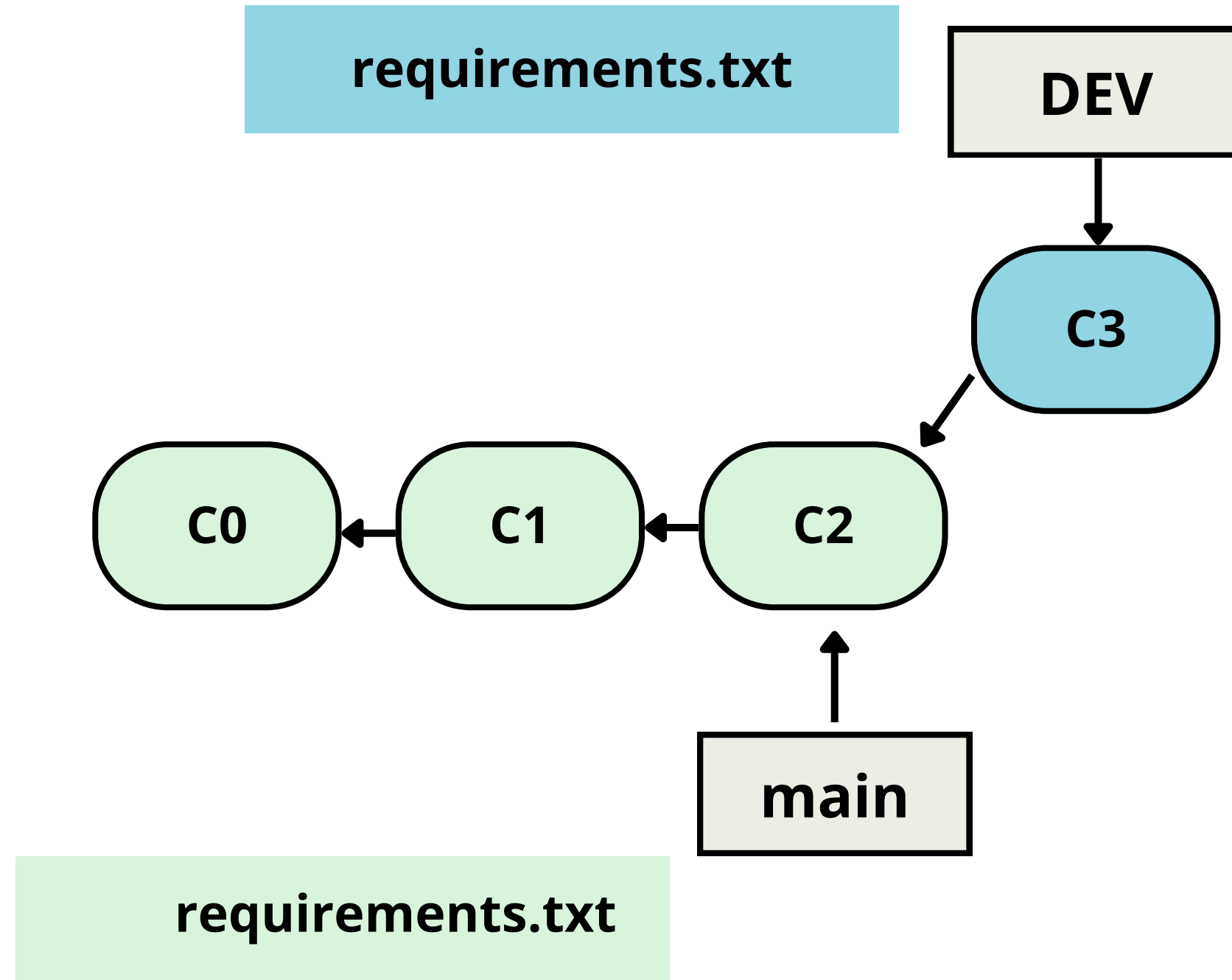
Jinja2==3.1.2

pandas==2.1.3



```
git add requirements.txt
```

```
git ci -m "Add pandas and upgrade"
```



Le conflit de merge : diverger

```
git switch main
```

Editer le fichier "requirements.txt"

Flask==2.2.4



gunicorn==20.1.0

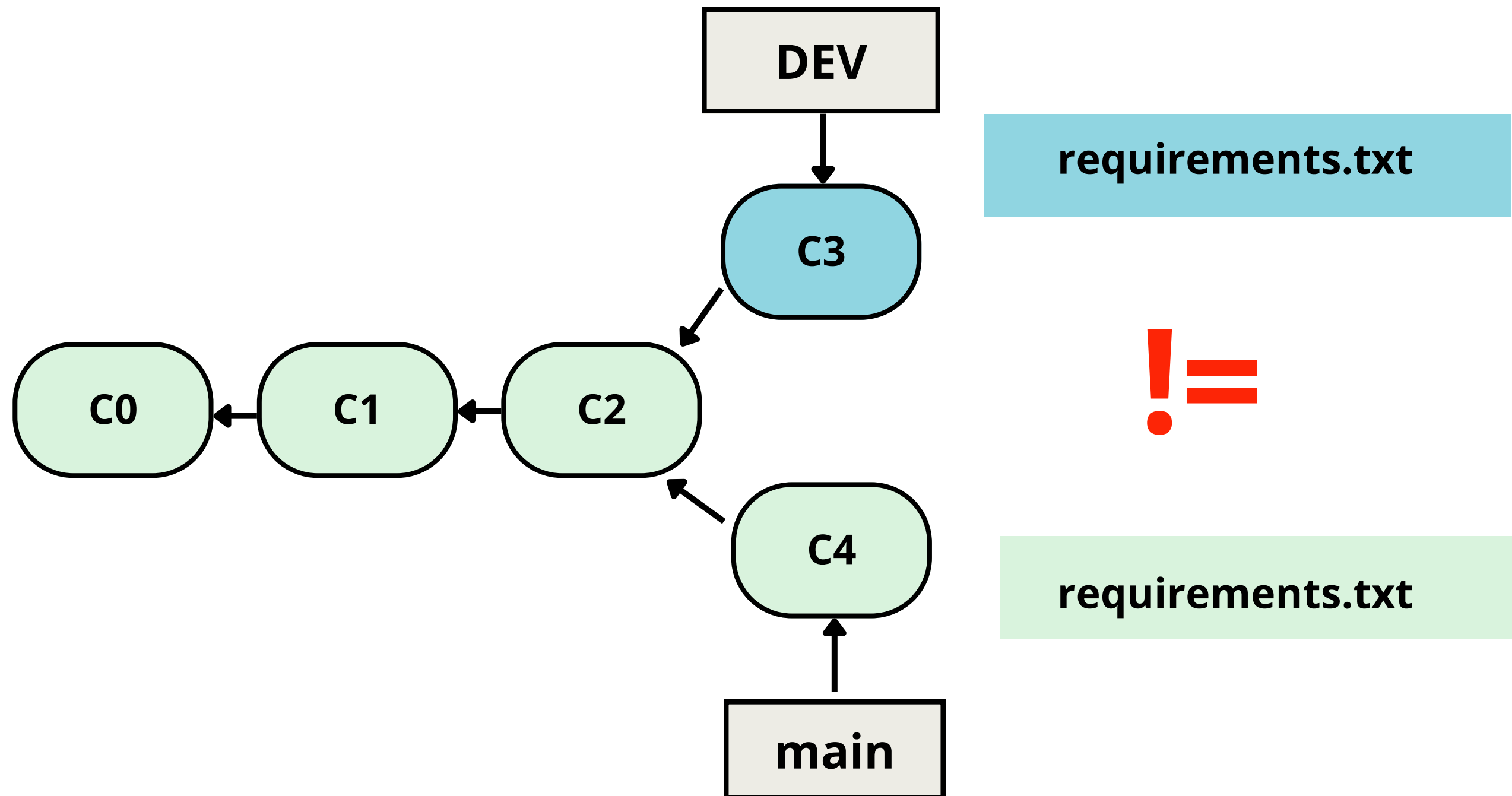
Jinja2==3.1.2

numpy==1.26.2



```
git add requirements.txt
```

```
git ci -m "Add numpy and upgrade" ◀-----
```



Le conflit de merge : différences

```
git diff main upgrade_deps
```

```
- Flask==2.2.3  
+Flask==2.2.4  
gunicorn==20.1.0  
Jinja2==3.1.2  
- pandas==2.1.3  
+numpy==1.26.2
```

Le conflit de merge : le drame

```
git merge upgrade_deps
```

```
Fusion automatique de requirements.txt  
CONFLIT (contenu) : Conflit de fusion dans requirements.txt  
La fusion automatique a échoué ; réglez les conflits et validez
```


Le conflit de merge : le status

```
git status -s
```

```
UU requirements.txt
```

```
git status
```

```
Vous avez des chemins non fusionnés.  
  (réglez les conflits puis lancez "git commit")  
  (utilisez "git merge --abort" pour annuler la fusion)
```

```
Chemins non fusionnés :  
  (utilisez "git add <fichier>..." pour marquer comme résolu)  
    modifié des deux côtés : requirements.txt
```

Le conflit de merge : l'affichage

<<<<<<<< HEAD

Flask==2.2.4

=====

Flask==2.2.3

>>>>>>>> upgrade



main

résolution: édition manuelle

<<<<<< HEAD

Flask==2.2.4

=====

Flask==2.2.3

>>>>>> upgrade

gunicorn==20.1.0

Jinja2==3.1.2

<<<<<< HEAD

numpy==1.26.2

=====

pandas==1.2.3

>>>>>> upgrade

Flask==2.2.4

gunicorn==20.1.0

Jinja2==3.1.2

numpy==1.26.2

pandas==1.2.3

résolution: git add

```
git status -s
```

```
UU requirements.txt
```

```
git add requirements.txt + git status
```

Sur la branche main

Votre branche est en avance sur 'origin/main' de 1 commit.
(utilisez "git push" pour publier vos commits locaux)

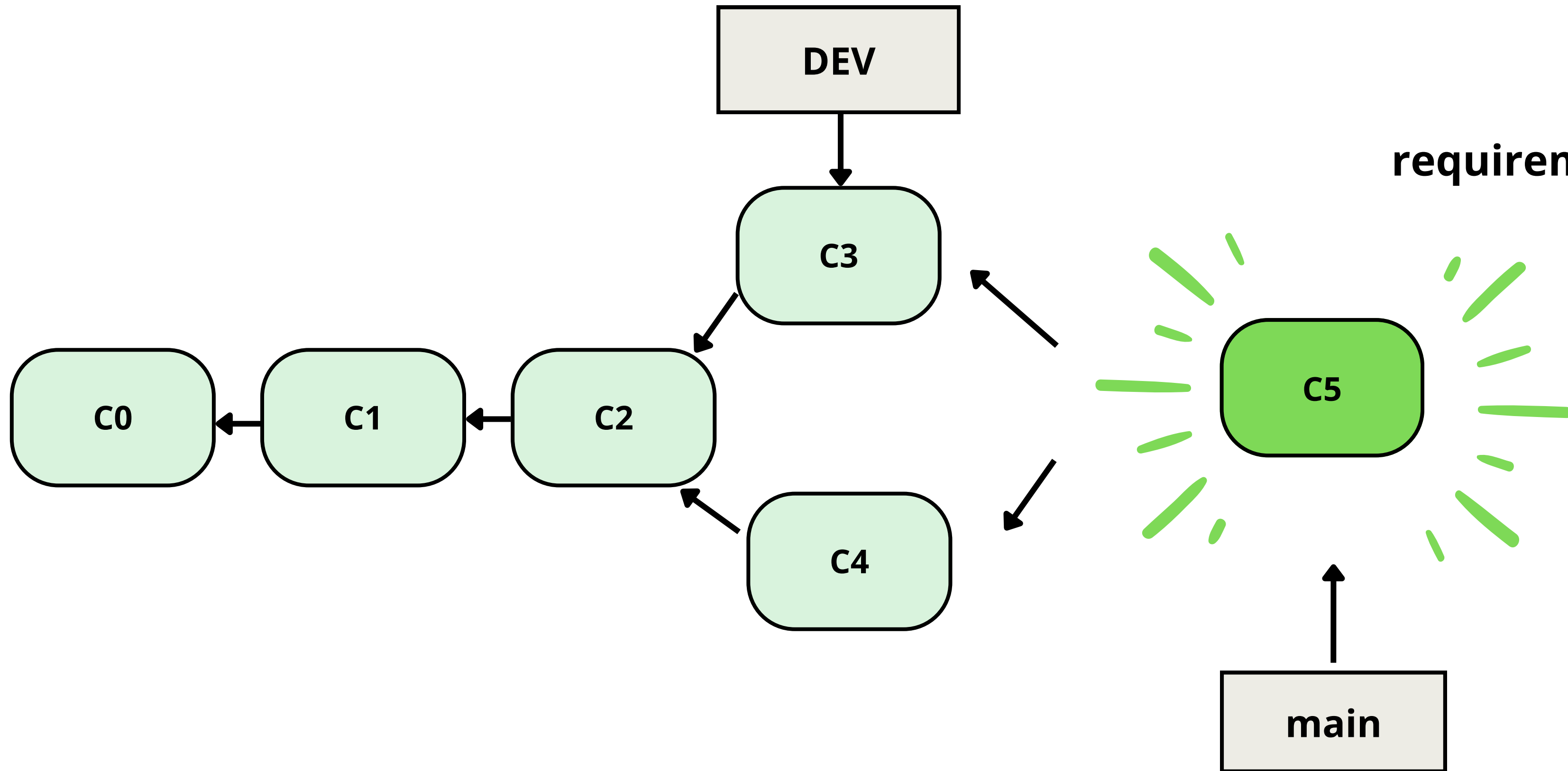
Tous les conflits sont réglés mais la fusion n'est pas terminée.
(utilisez "git commit" pour terminer la fusion)

Modifications qui seront validées :

modifié : requirements.txt

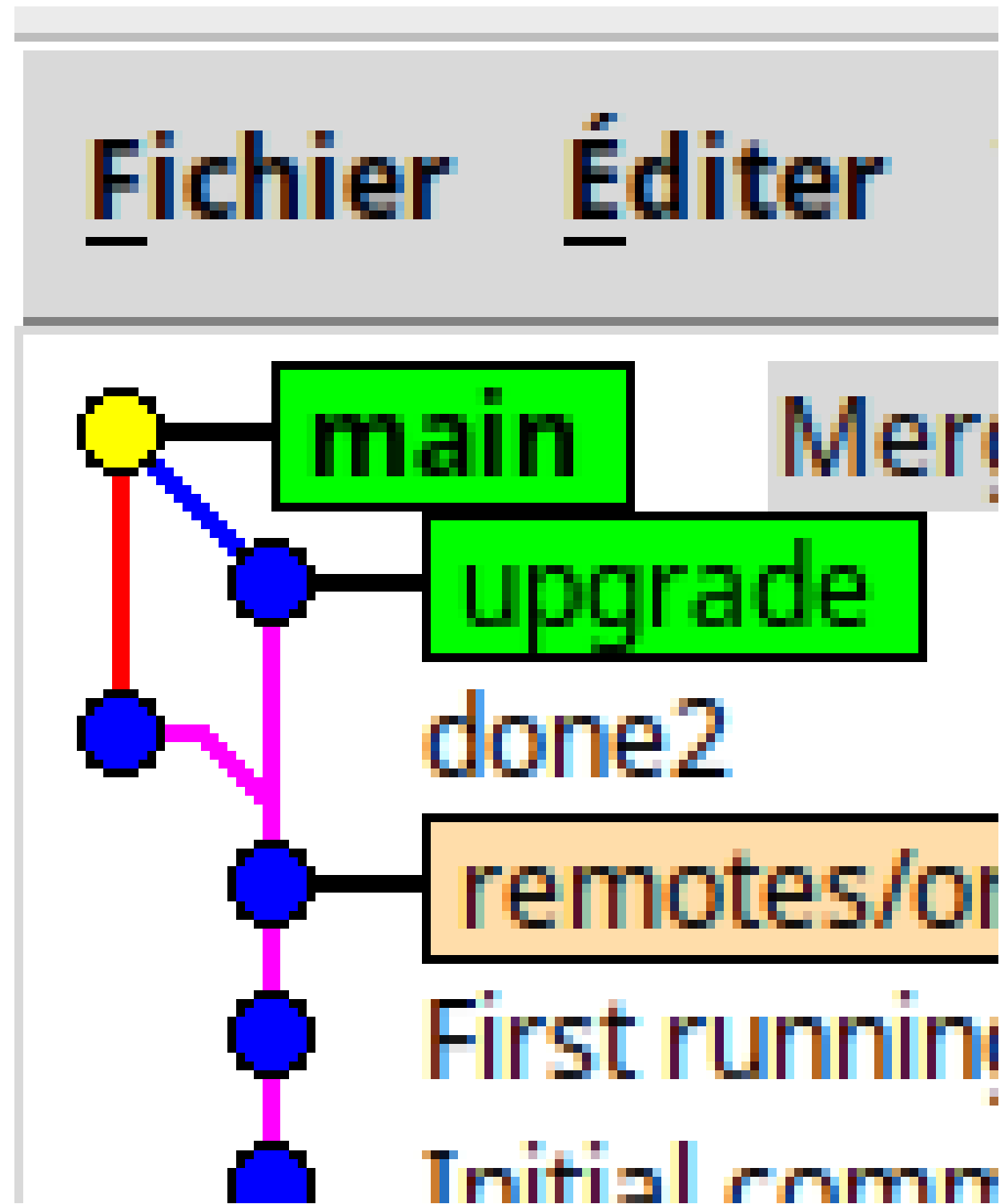
merge résolu

requirements.txt



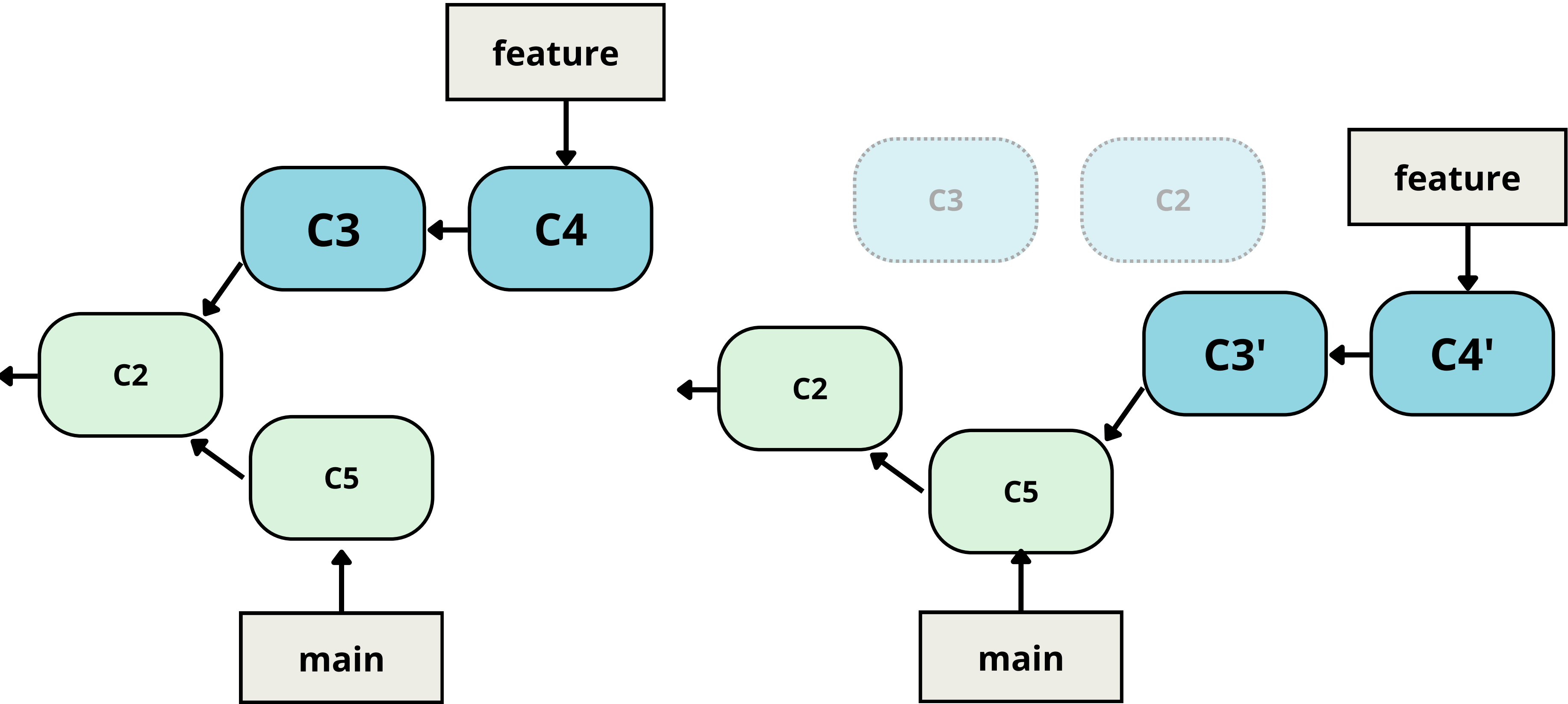
merge résolu

gitk --all



git rebase

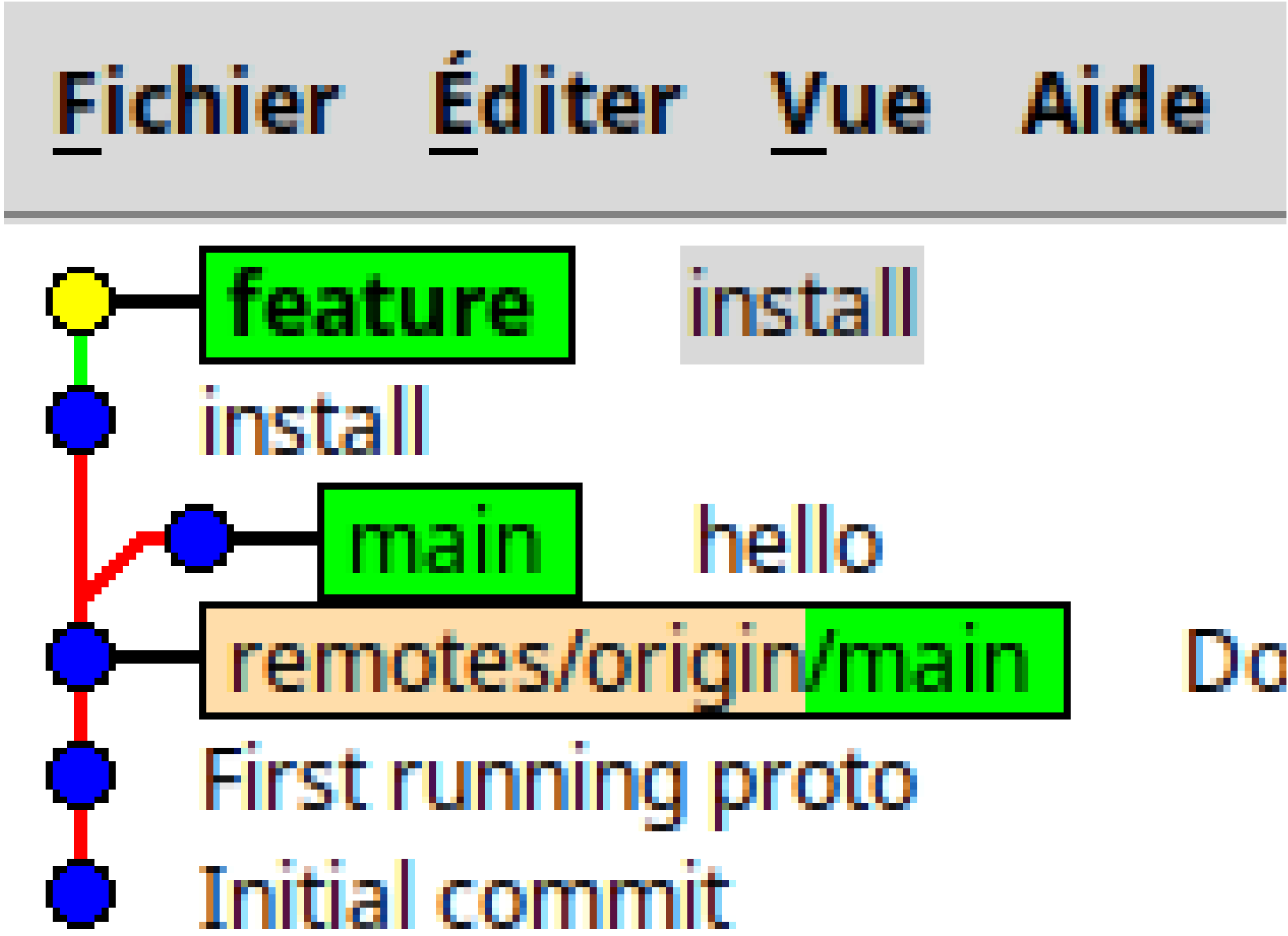
ou comment changer la base
d'une branche



git rebase: mise en place

- **git switch main**
- **git branch feature**
- **git commit**
- **git switch feature**
- **git commit; git commit**

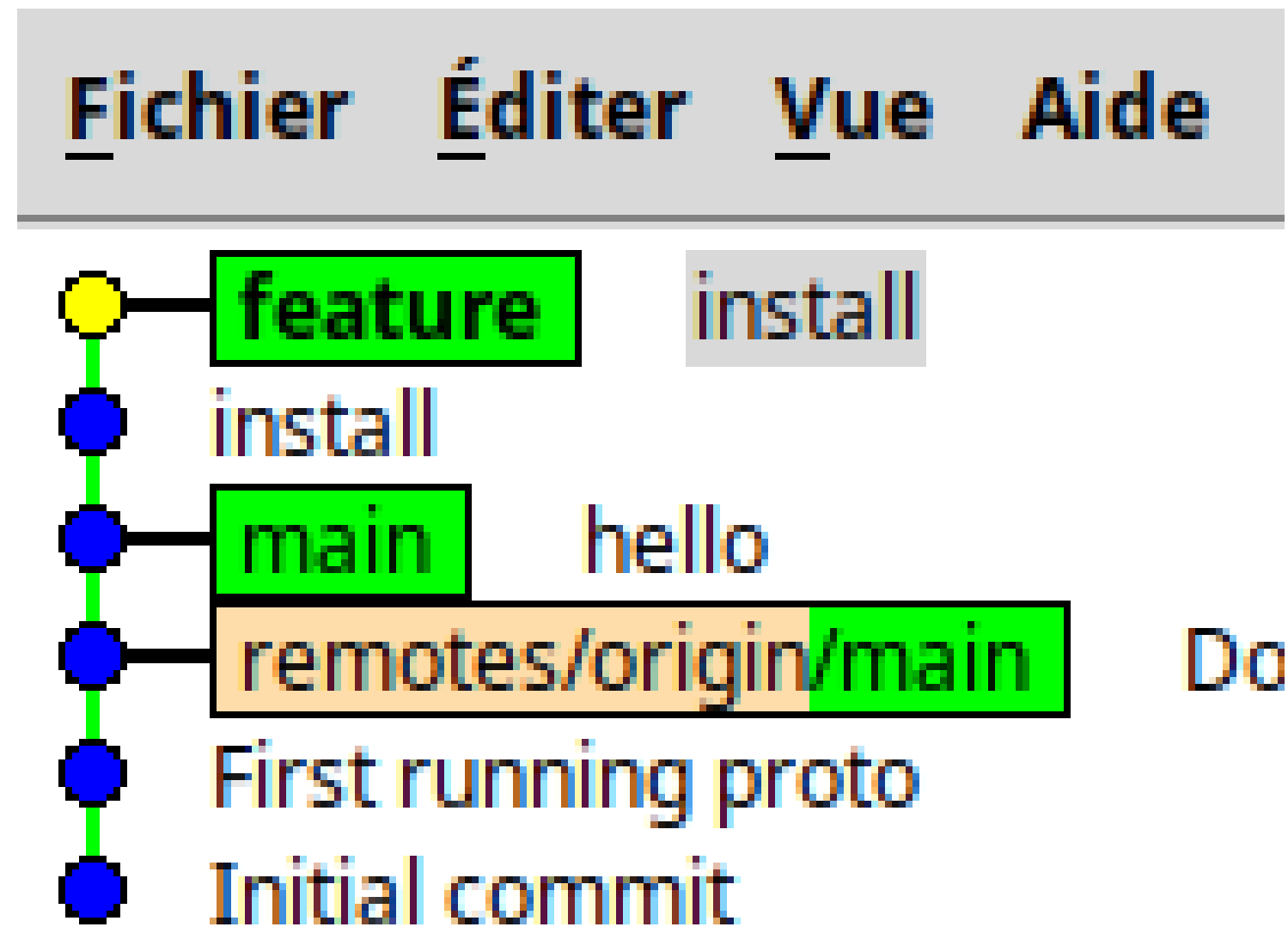
gitk --all



git rebase: do it

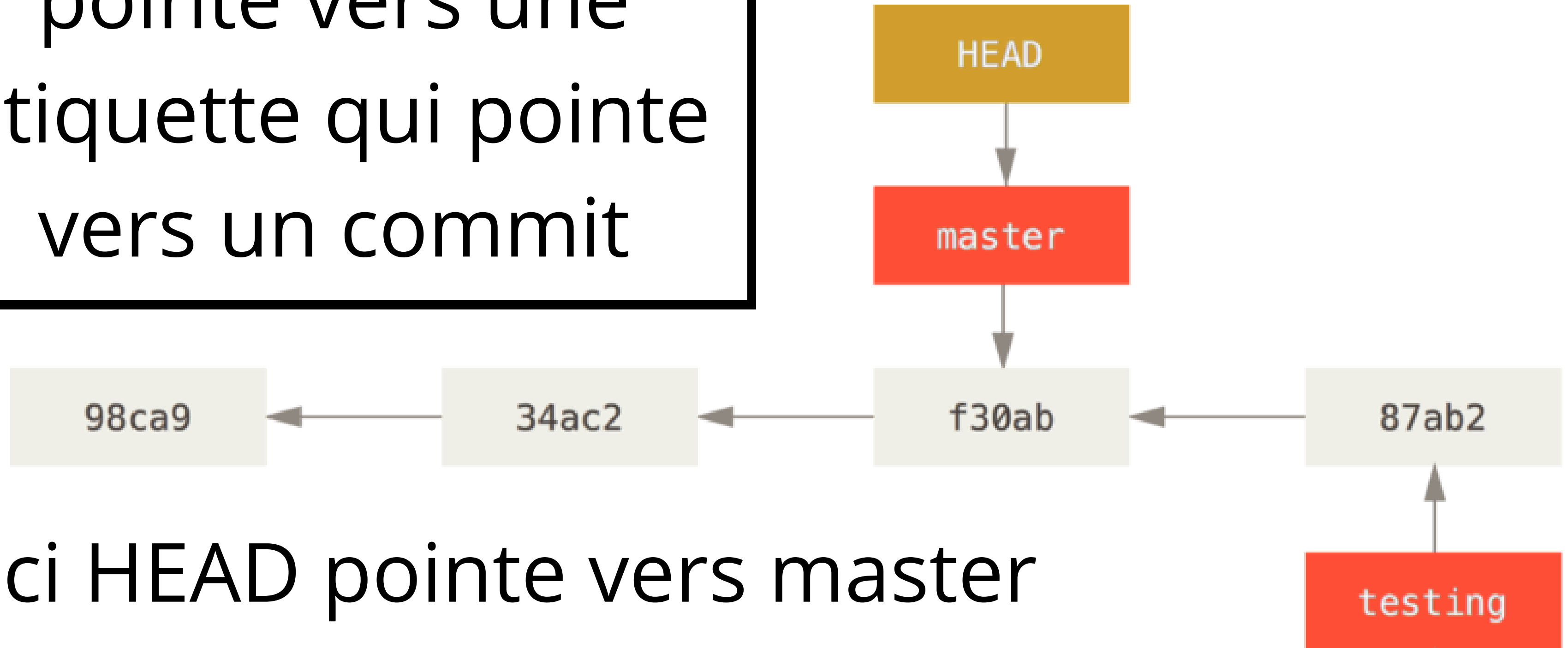
- git switch feature
- git rebase main

gitk --all



HEAD c'est quoi ?

une étiquette qui
pointe vers une
étiquette qui pointe
vers un commit



HEAD nous dit où on est.
C'est le "lieu" de notre
répertoire de travail.

HEAD c'est quoi ?

Si HEAD pointe vers "dev"
alors notre répertoire reflète
le contenu pointé par "dev"

HEAD c'est quoi ?

```
git swich main
```

```
git l
```

* **410a1eb** (HEAD -> **main**) Merge 'dev

| \

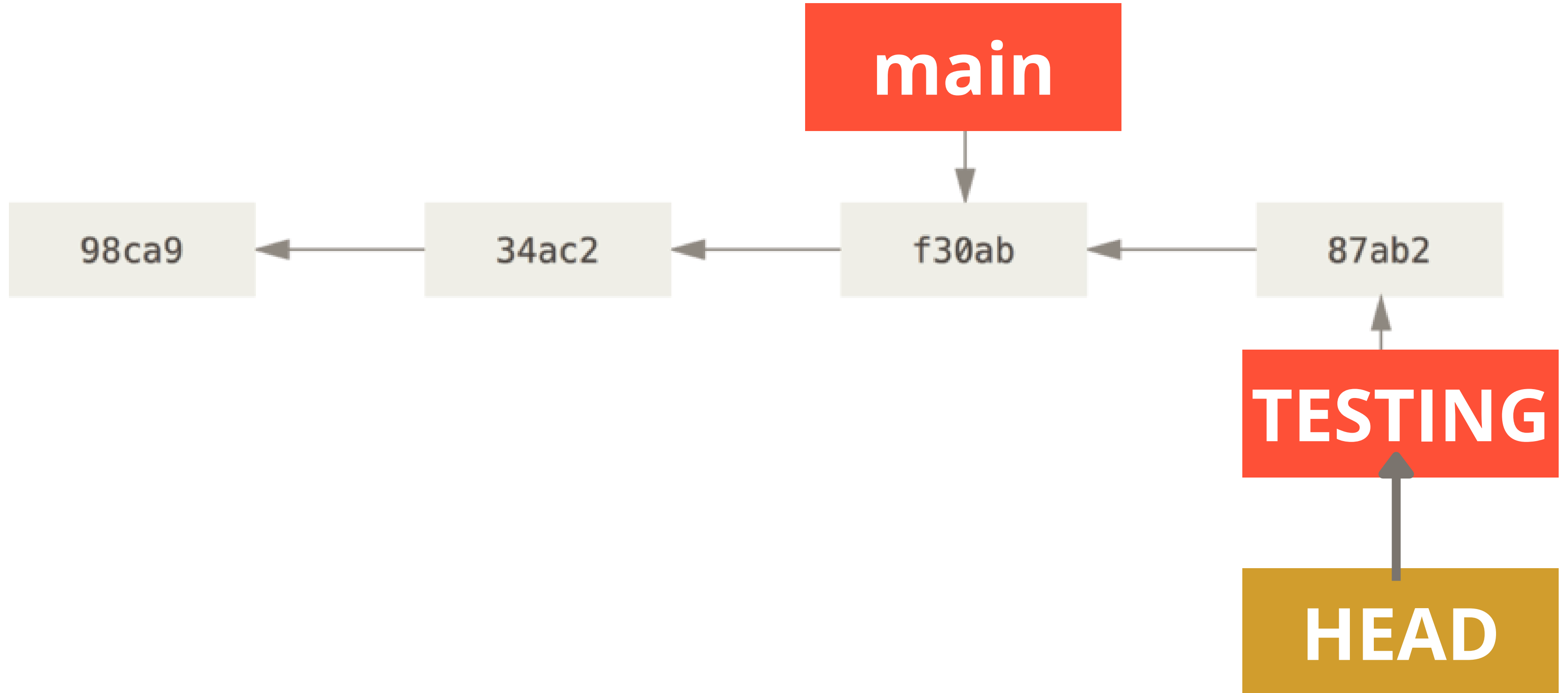
| * **a09180f** (**dev**) Ton autre

git switch et HEAD

git switch testing

- **extraie le commit dans le répertoire de travail**
- **positionne HEAD sur l'étiquette 'testing'**

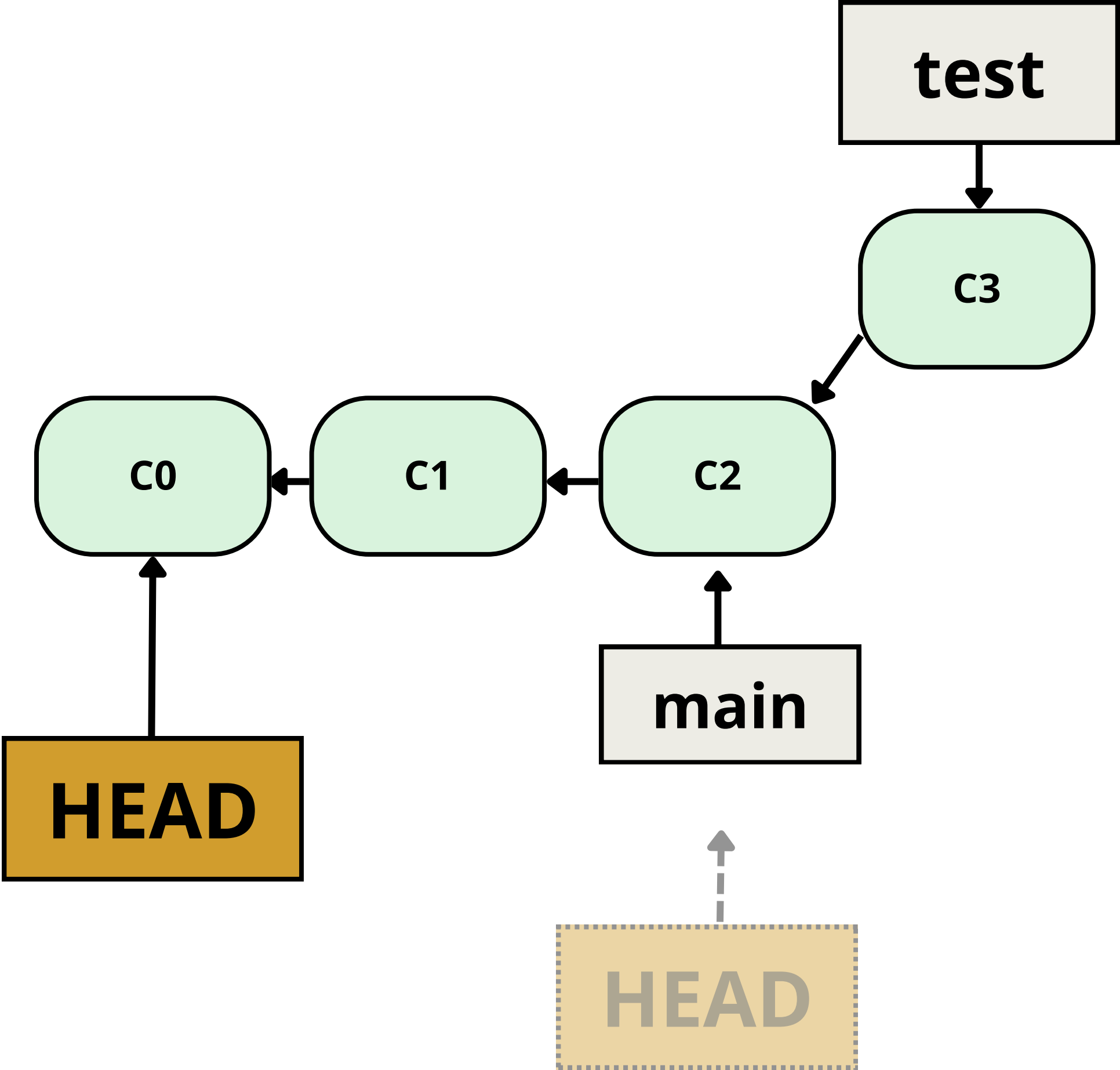
git switch testing



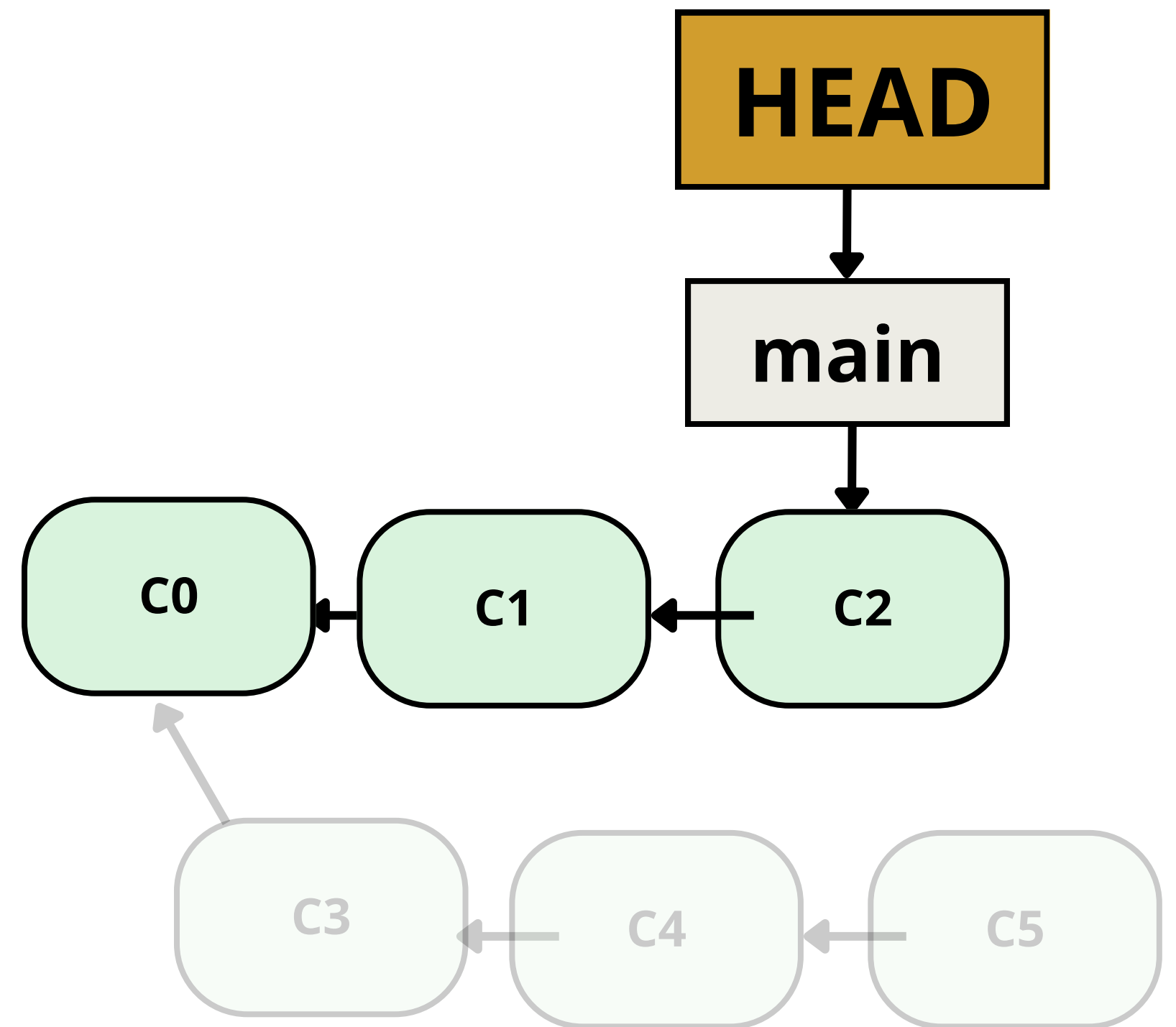
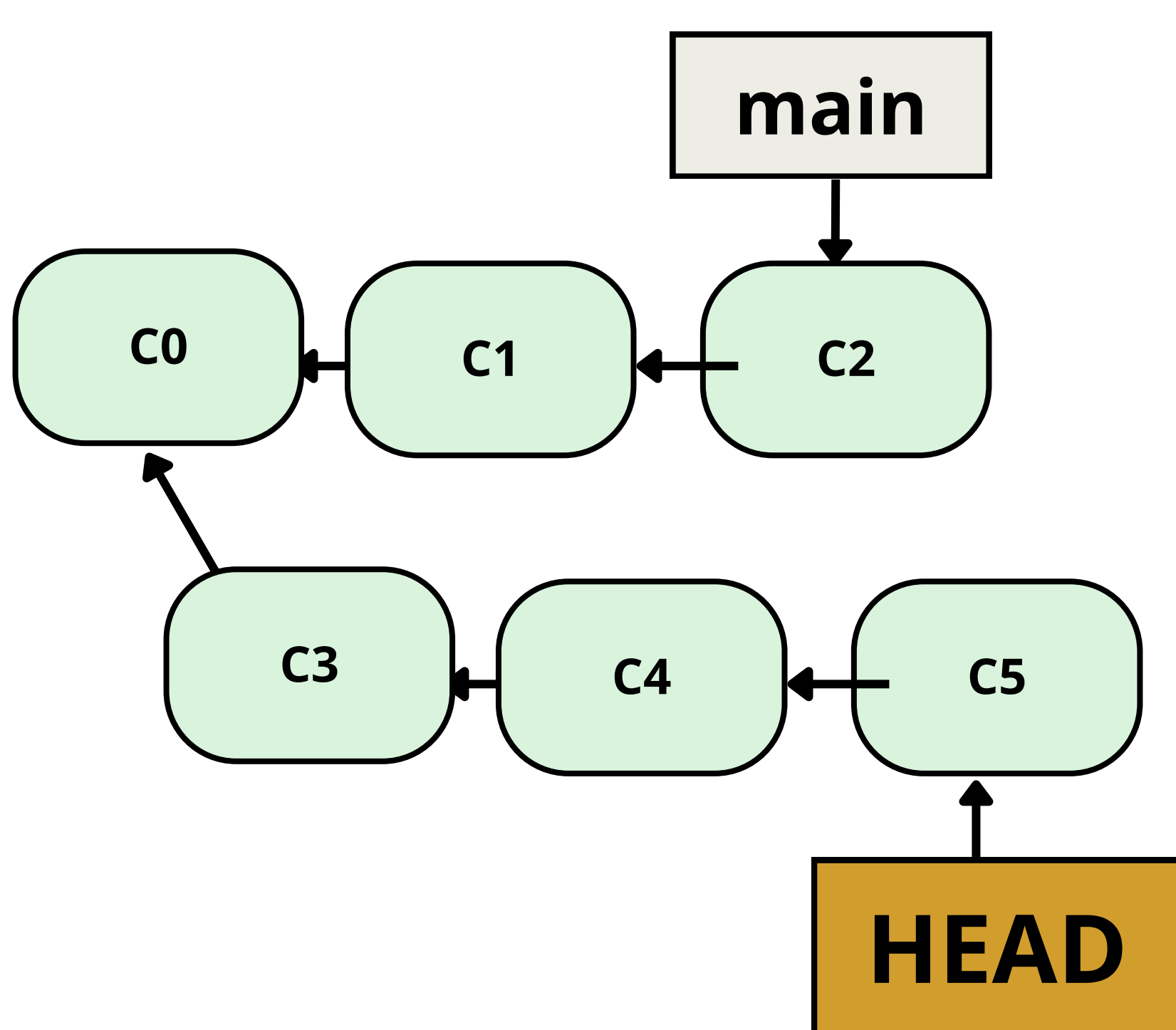
detached HEAD

État du répertoire de
travail qui n'est pointé
par aucune étiquette de
branche
(autre que HEAD)

detached HEAD



**on peut commiter des modifs mais sans
étiquette de branche on va perdre leur trace**



git checkout 2873979

```
[richard@joshua] ~/tmp/tries/tfgnc (feature)$ git co 28f3979
```

Note : basculement sur '28f3979'.

Vous êtes dans l'état « HEAD détachée ». Vous pouvez visiter, faire des modifications expérimentales et les valider. Il vous suffit de faire un autre basculement pour abandonner les commits que vous faites dans cet état sans impacter les autres branches

Si vous voulez créer une nouvelle branche pour conserver les commits que vous créez, il vous suffit d'utiliser l'option -c de la commande switch comme ceci :

```
git switch -c <nom-de-la-nouvelle-branche>
```

Ou annuler cette opération avec :

```
git switch -
```

Désactivez ce conseil en renseignant la variable de configuration advice.detachedHead à false

HEAD est maintenant sur 28f3979 install

```
git switch -c ma_nouvelle_branche
```

<https://learngitbranching.js.org>

