



## Références

- <https://git-scm.com/> : site officiel
- <https://marklodato.github.io/visual-git-guide/index-en.html> :  
Illustrations des commandes.
- <https://git-scm.com/book/en/v2> : Livre « *Pro Git* », gratuit.
- <http://git-school.github.io/visualizing-git>

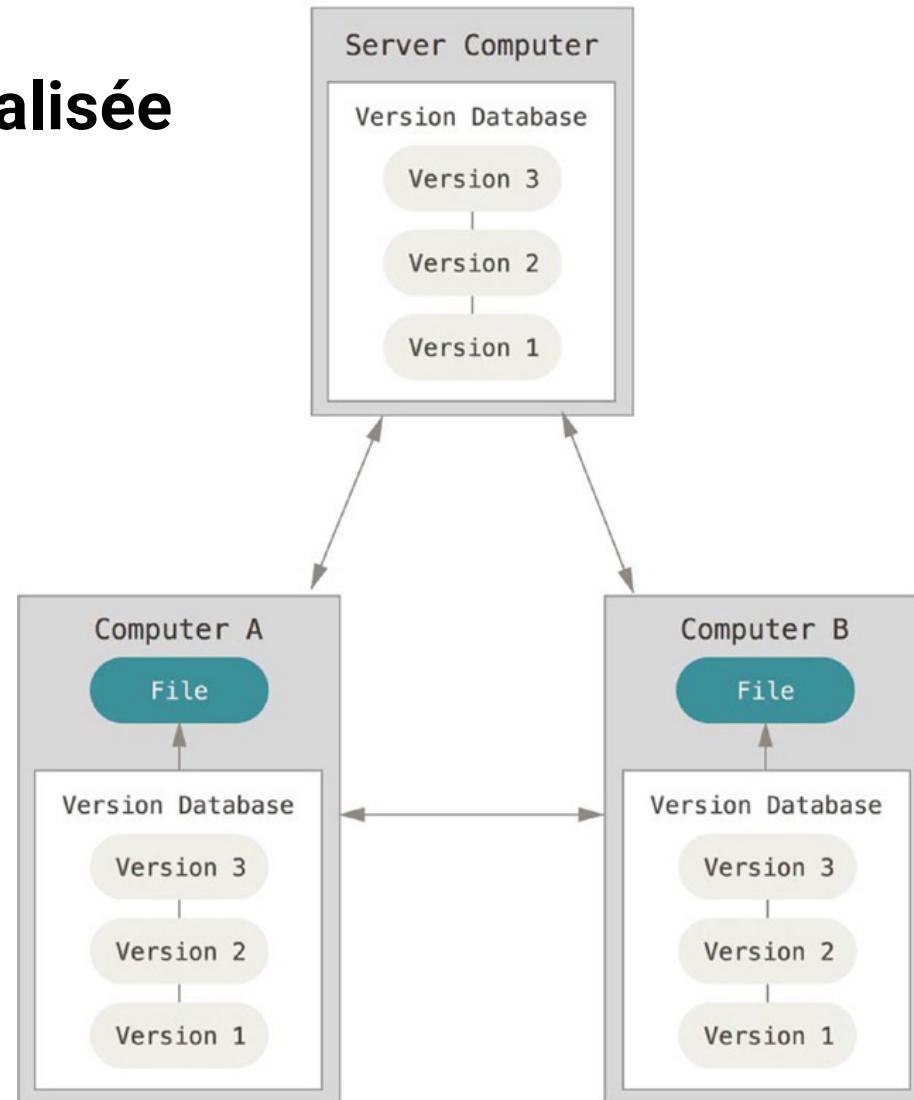
## Définition

*"Un logiciel de gestion de versions (ou VCS en anglais, pour Version Control System) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus."-Wikipédia*

## Avantages

- **Gestion du code source**
  - Sauvegarder l'évolution du code
  - Pouvoir revenir en arrière (version antérieure)
  - Comparer des versions
- **Collaboration du code source**
  - Travailler à plusieurs sur un projet
  - Pouvoir réaliser des modifications sans impacter le travail des autres

## Architecture décentralisée ou distribuée

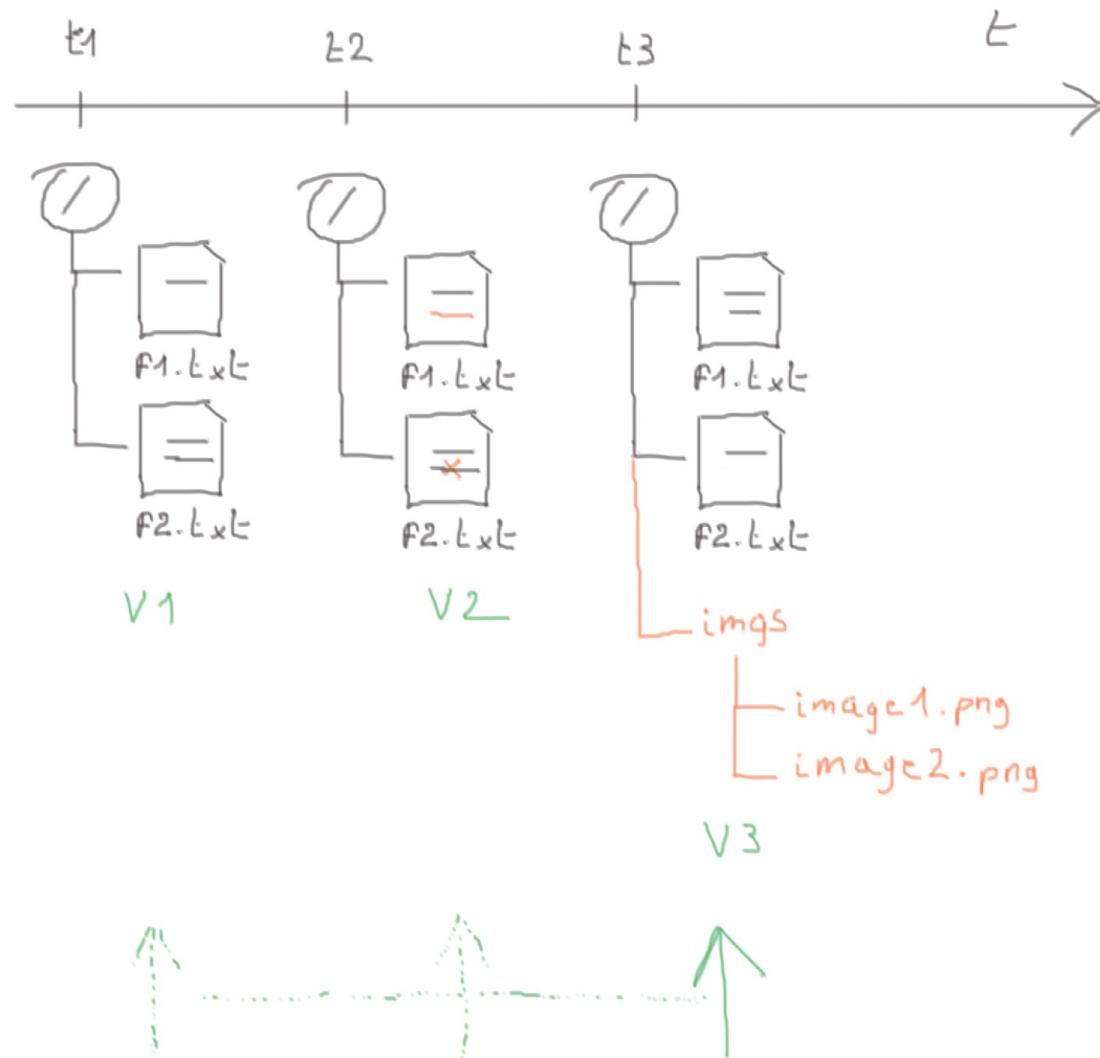




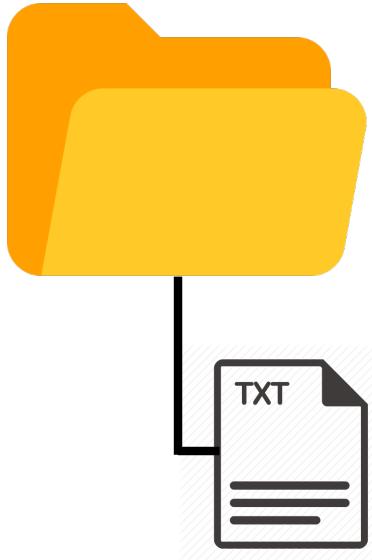
Git est un logiciel de gestion de versions décentralisée (DVCS-Distributed Version Control System).

# Concepts de base (théorie)

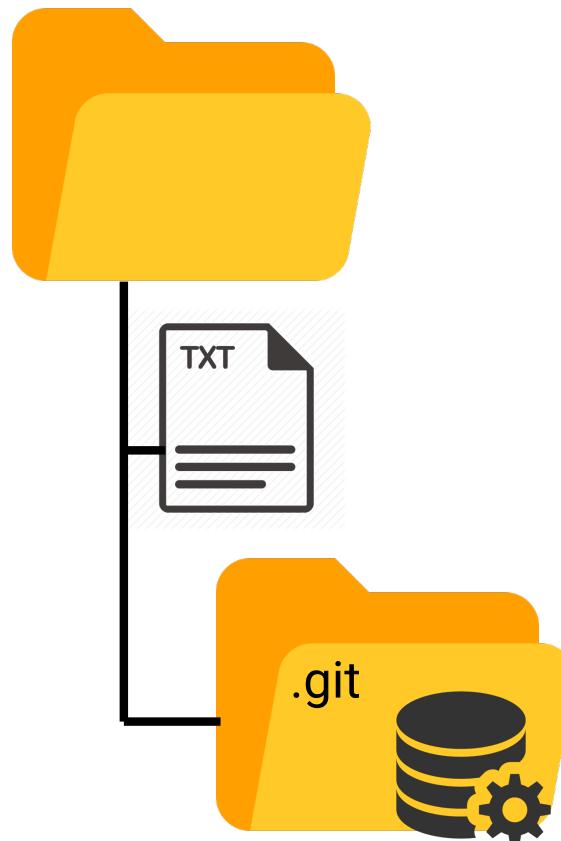
- 3 zones
- Objets Git
- Stockage des objets -> graphe orienté acyclique
- Références
- Garbage collector

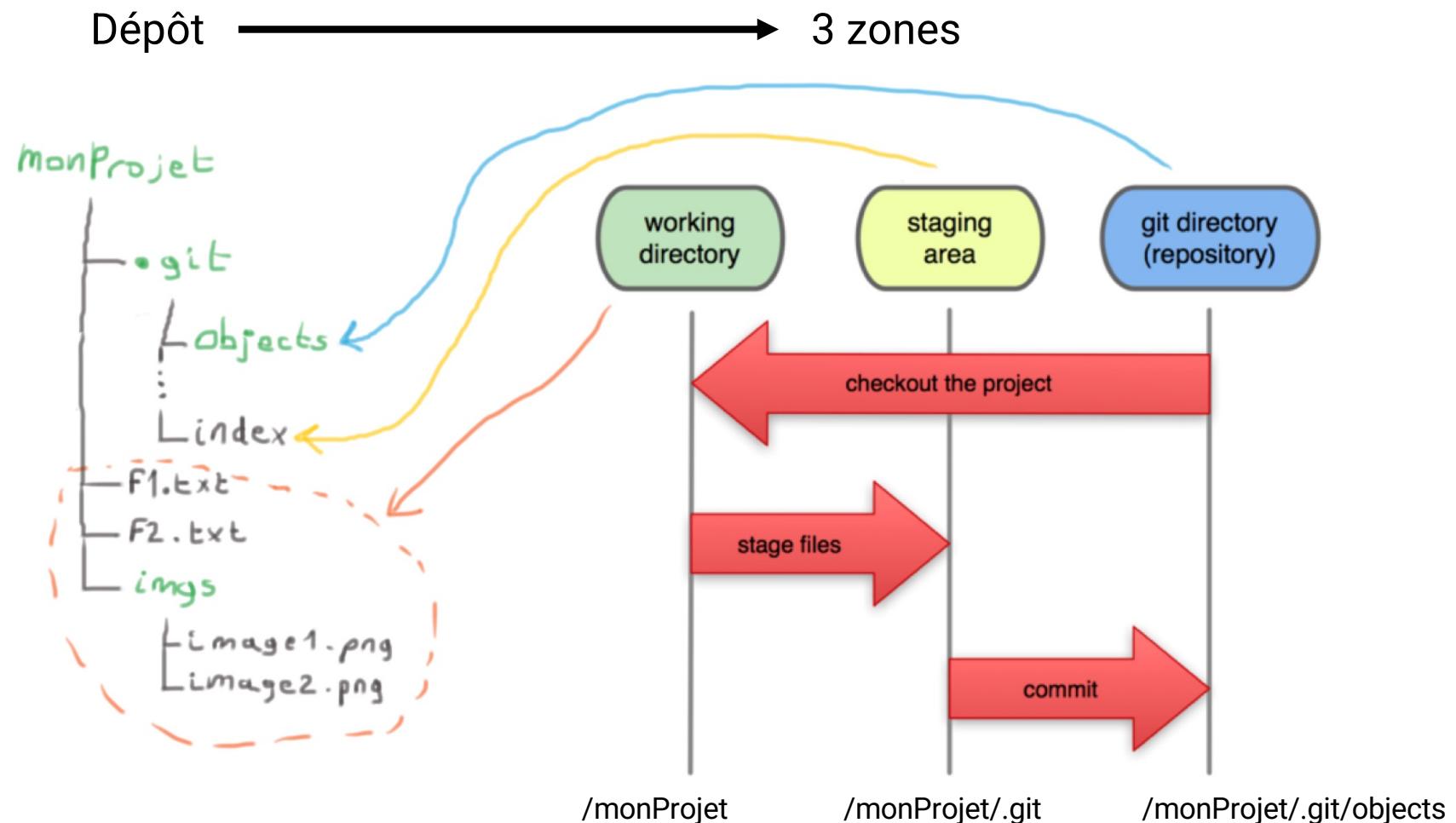


Répertoire « classique »

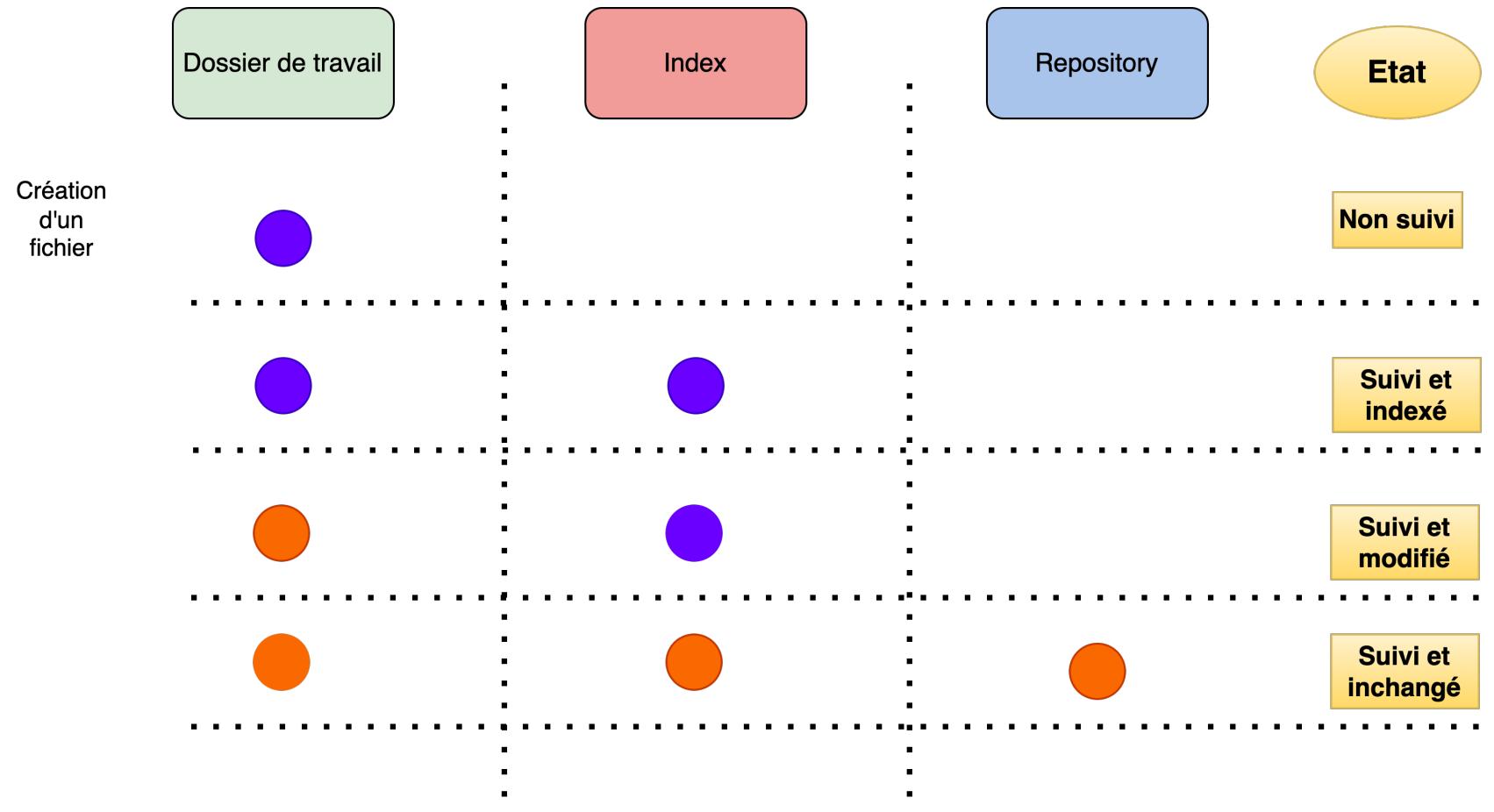


Dépôt Git

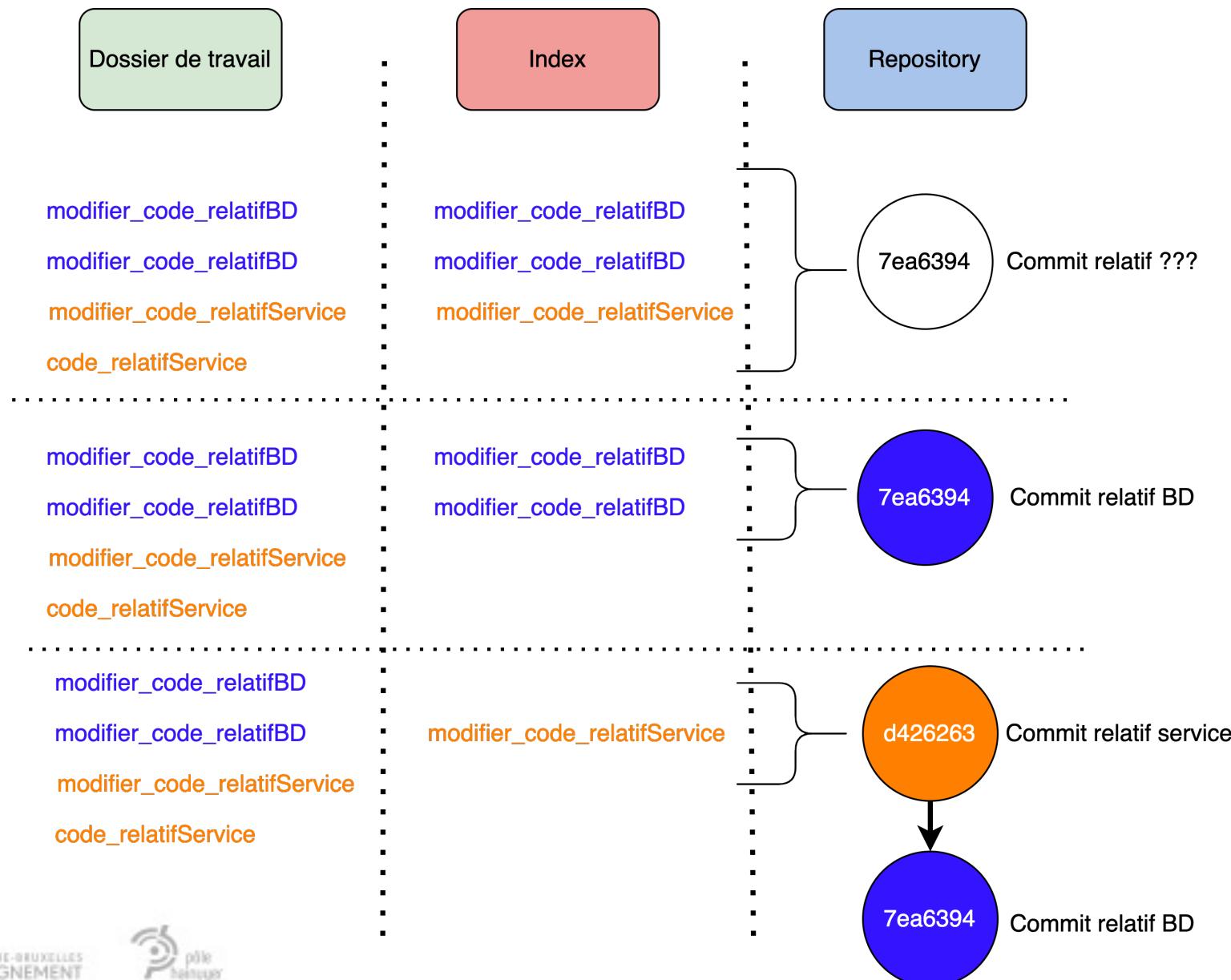




## Concepts de base – Etats d'un fichier



## Concepts de base – 3 zones



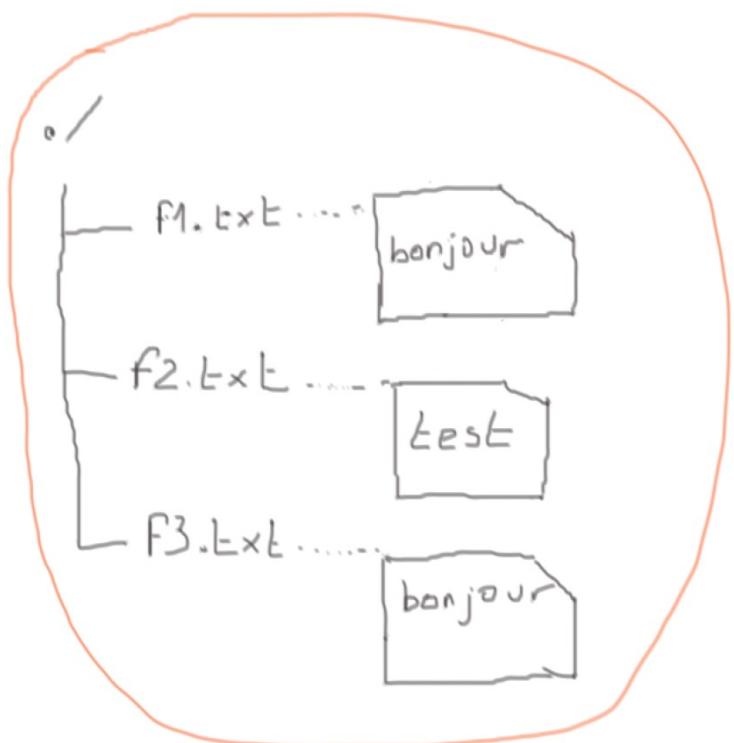
## Définition

*"SHA-1 (Secure Hash Algorithm) est une **fonction de hachage cryptographique** conçue par la National Security Agency des Etats-Unis (NSA), et publiée par le gouvernement des Etats-Unis comme un standard fédéral de traitement de l'information (Federal Information Processing Standard du National Institute of Standards and Technology (NIST)). Elle produit un résultat (appelé « hash » ou condensat) de 160 bits." - Wikipédia*

SHA1(contenu) => 2<sup>160</sup> possibilités => **garantie de l'unicité**

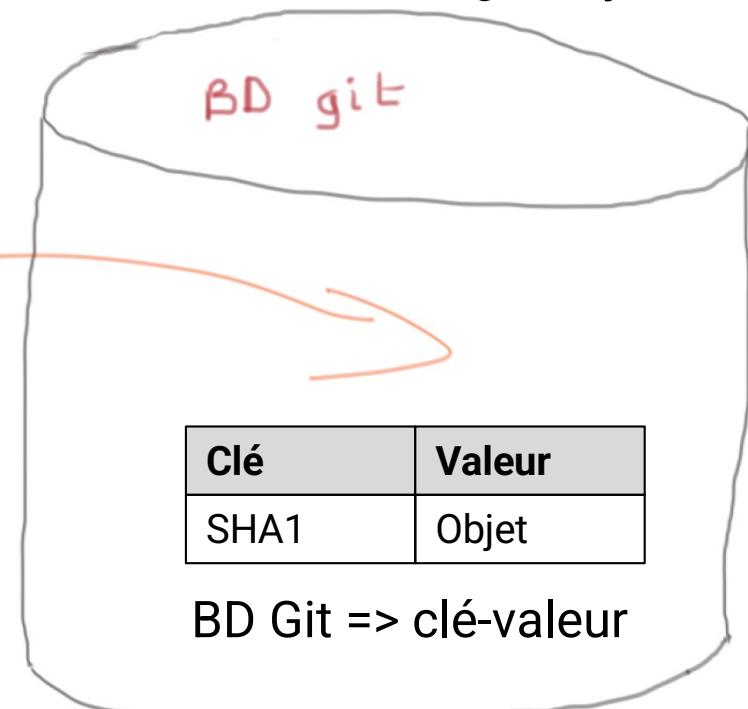
```
{14:34:16}~/tests/depot_git (master) $ echo "toto" | shasum  
e6e8ea7465f12e4d3b5a067a4c4dc698436b3478 -
```

## Comment Git stocke le projet ?

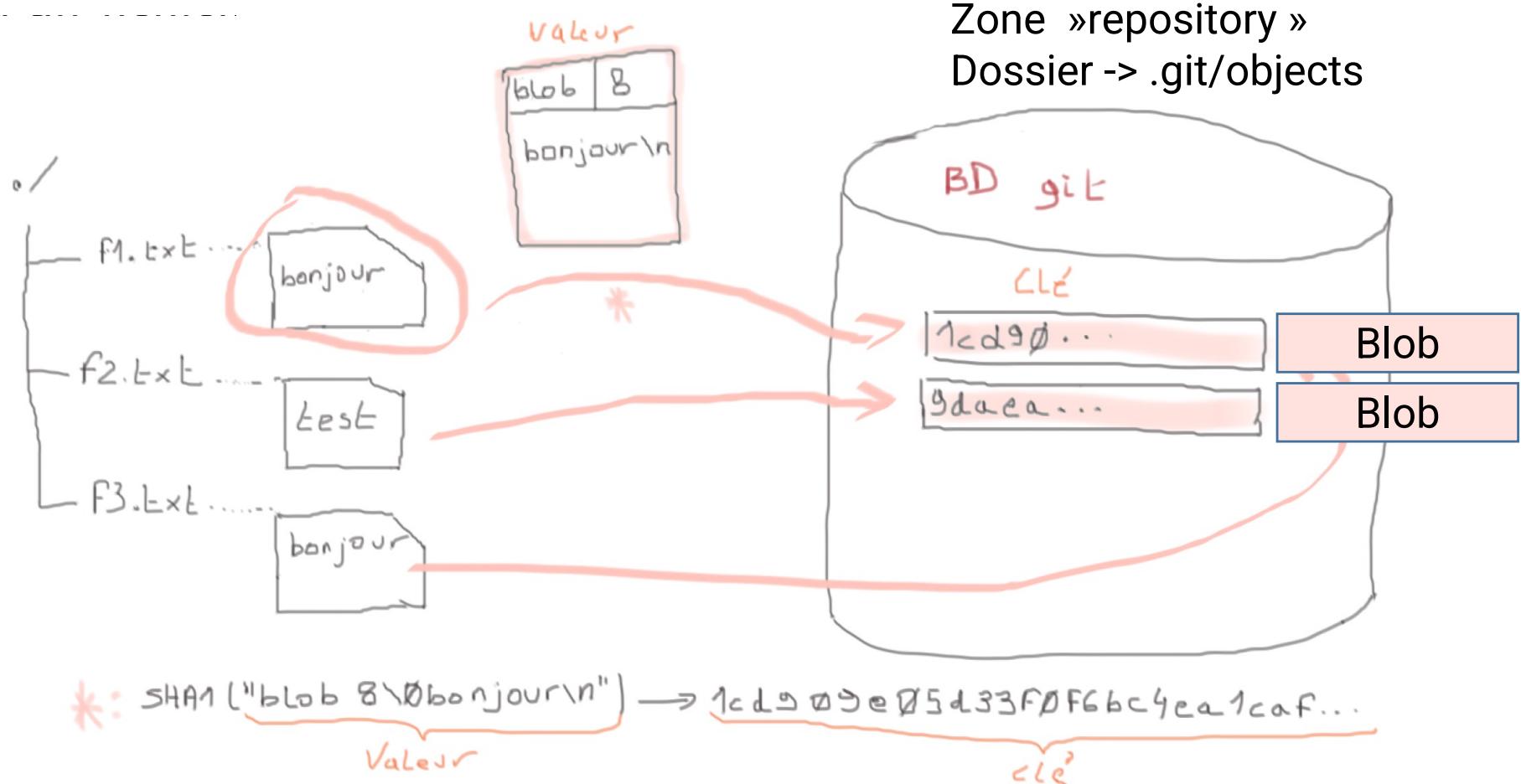


stocker?

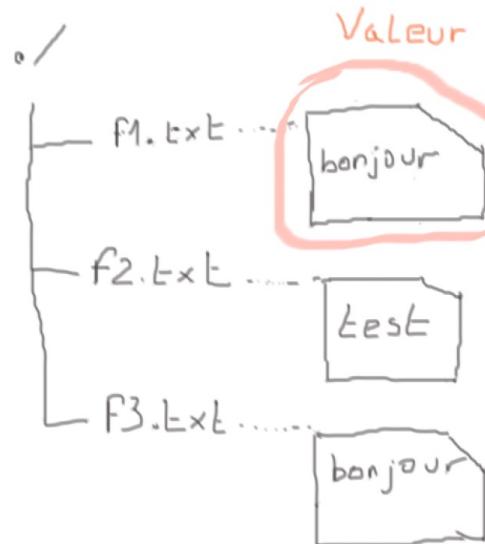
Zone « repository »  
Dossier -> .git/objects



**Blob** : binary large object (gros objet binaire) = stocke le contenu d'un fichier

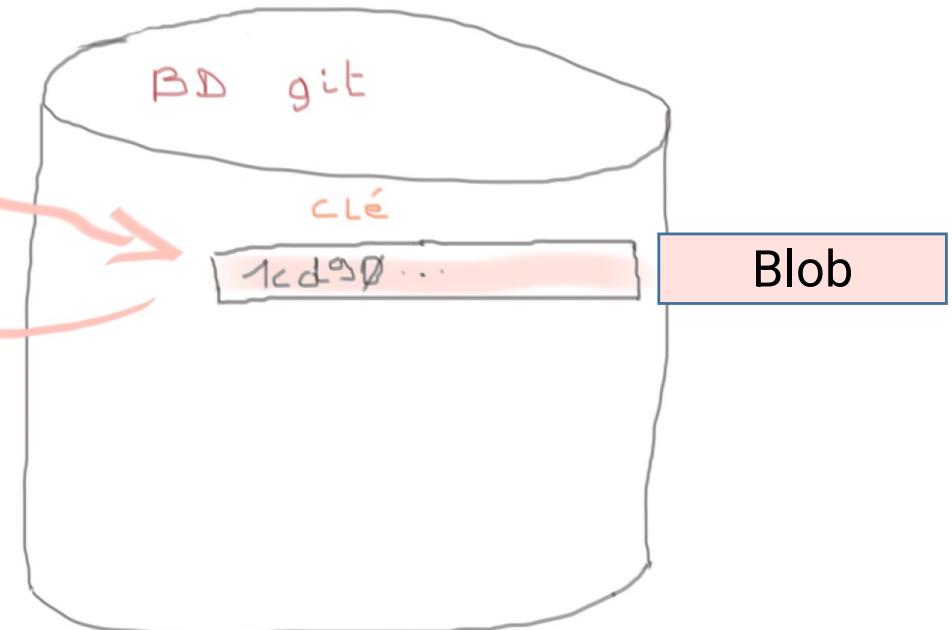


Récupérer le contenu d'un blob



PUT  
GET

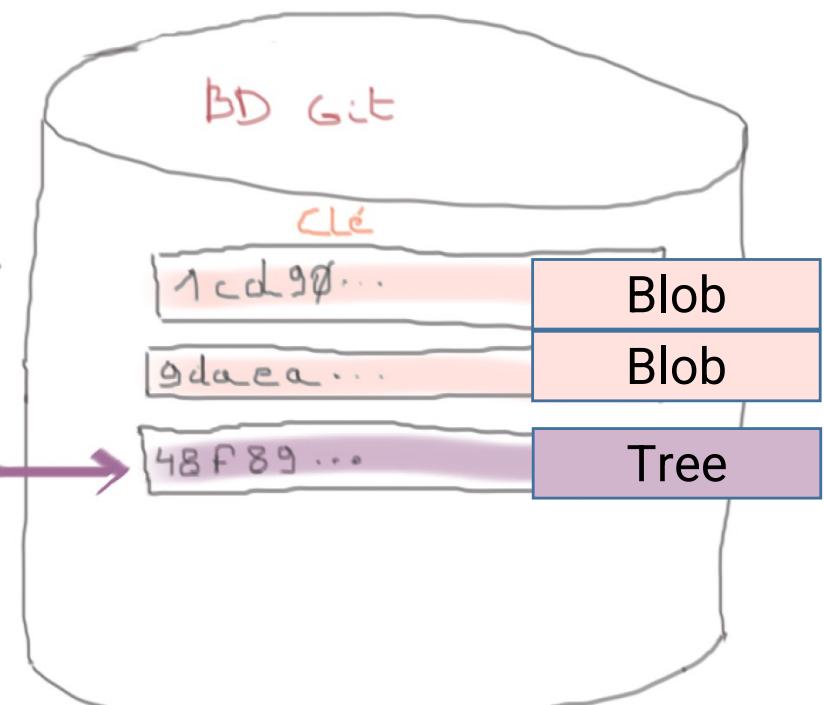
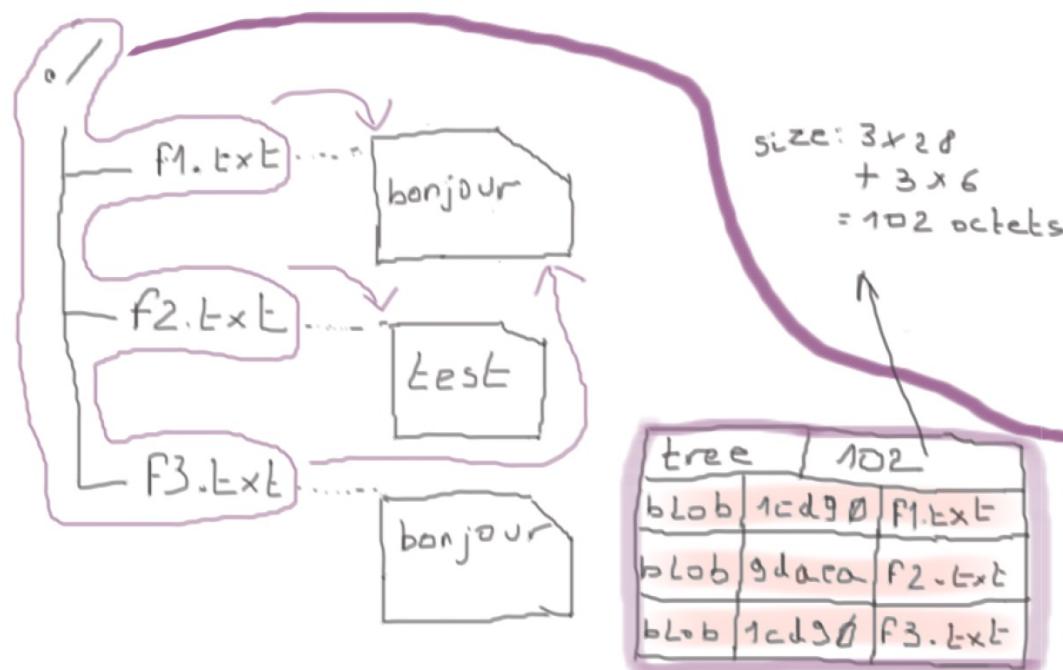
Zone »repository »  
Dossier -> .git/objects



GET: git cat-file -p 1cd90...

**Tree = Stocke l'arborescence du projet**

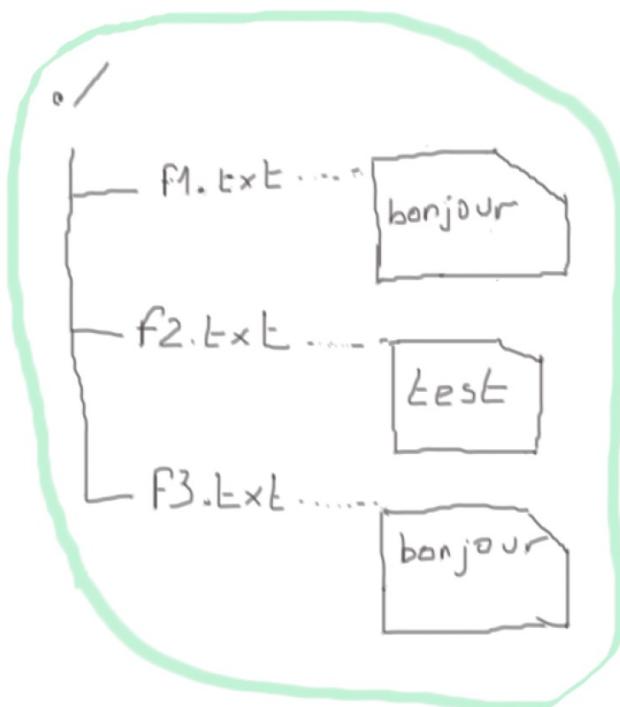
Zone »repository»  
Dossier -> .git/objects



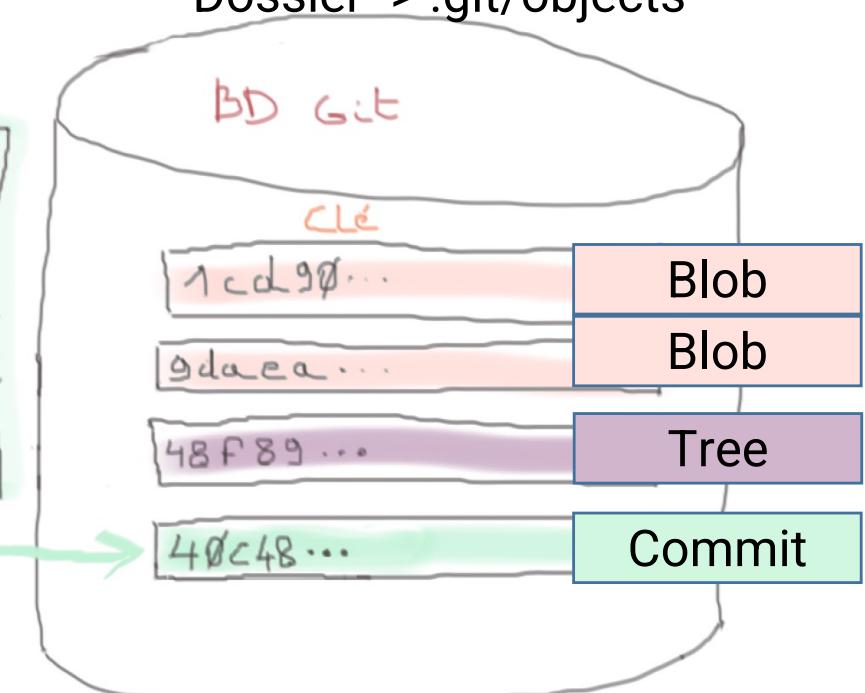
## Concepts de base – Objets Git - Commit

Stocke la référence d'un tree (arborescence) + nom de l'auteur + horodatage + message

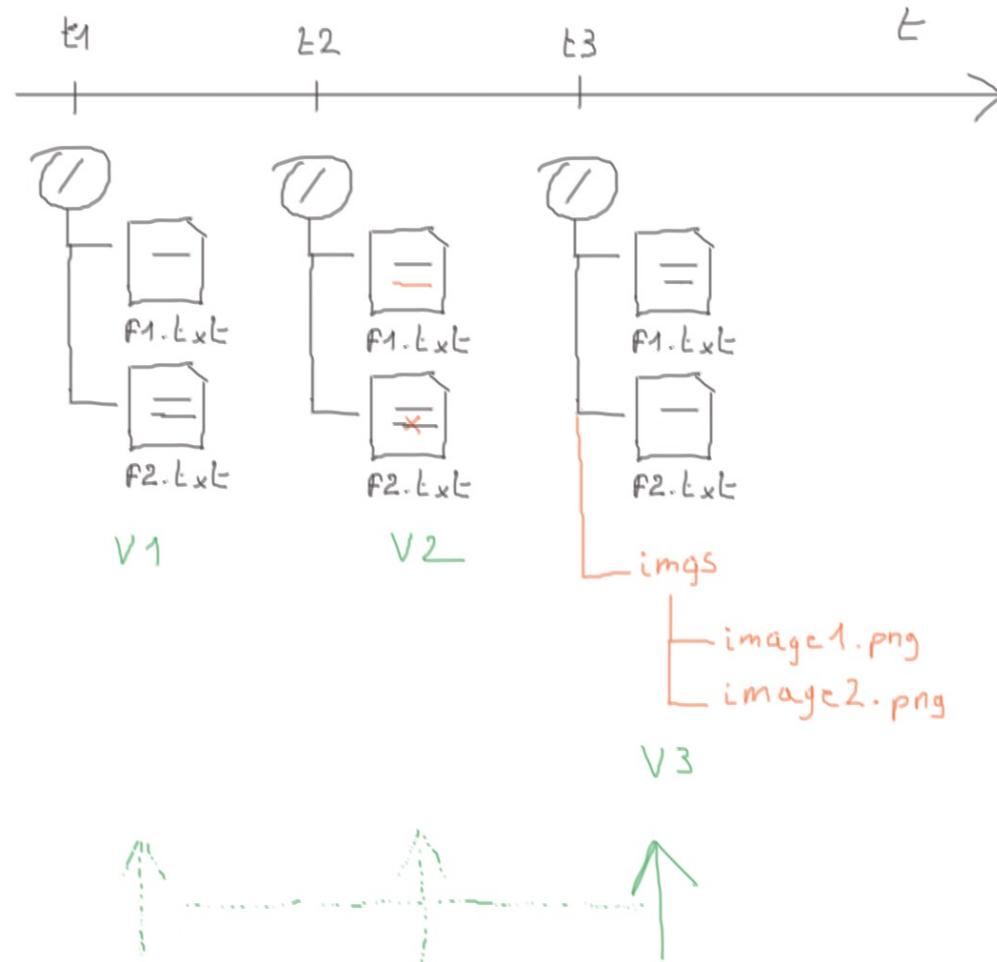
Zone »repository»  
Dossier -> .git/objects



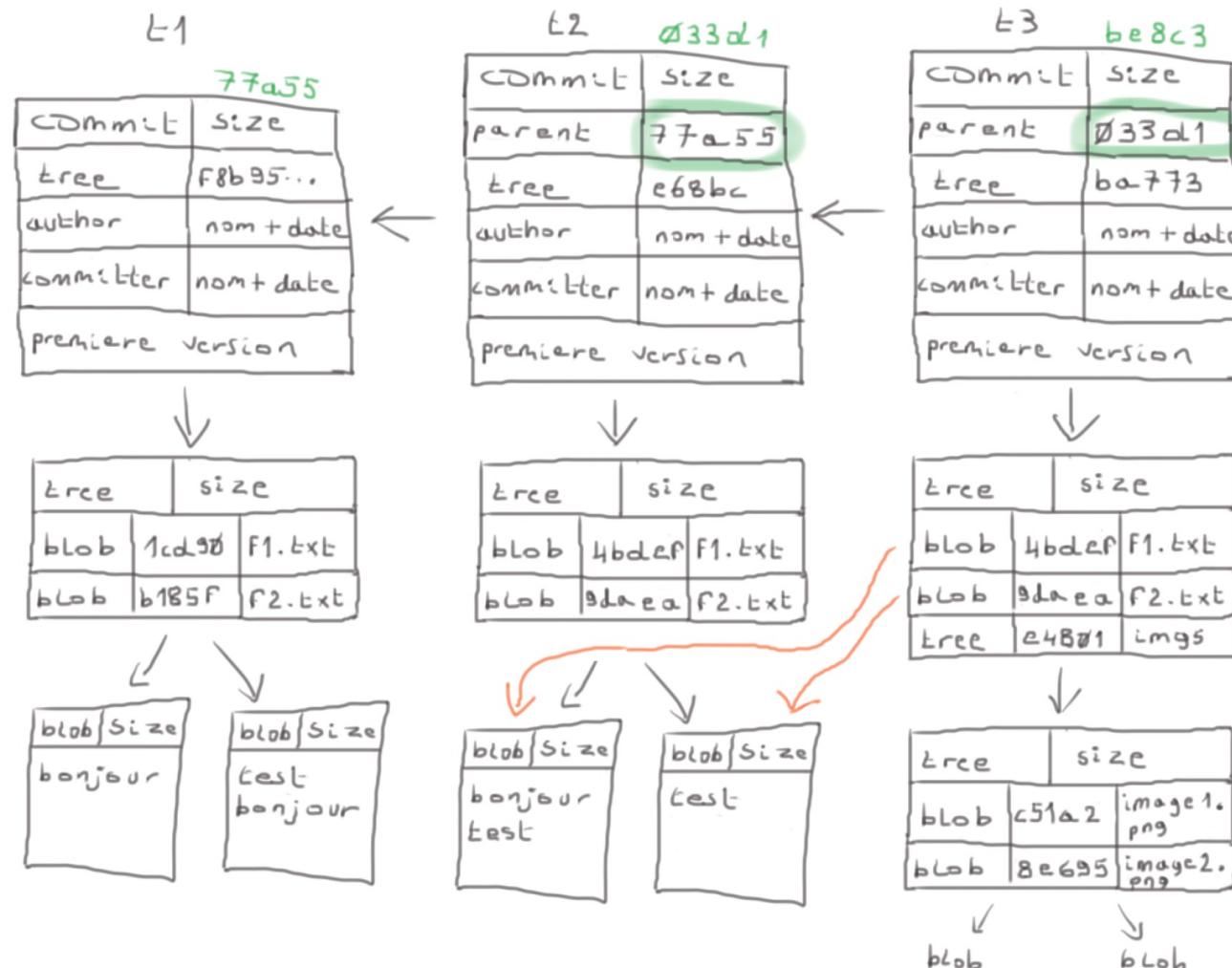
| Commit           | size       |
|------------------|------------|
| tree             | 48 F89     |
| author           | nom + date |
| committer        | nom + d    |
| premiere version |            |



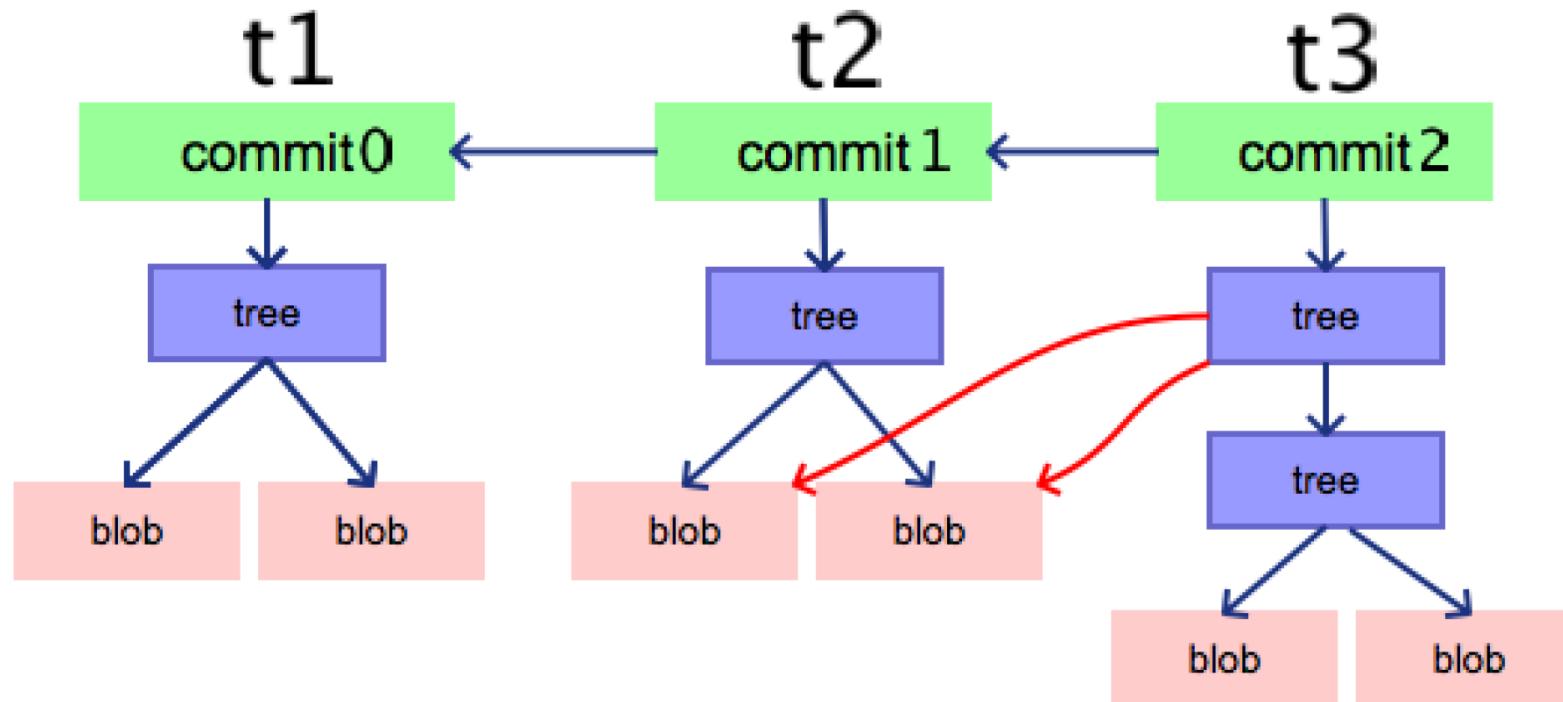
## Comment stocker les différents instantanés ?



## Comment stocker les différents instantanés ?

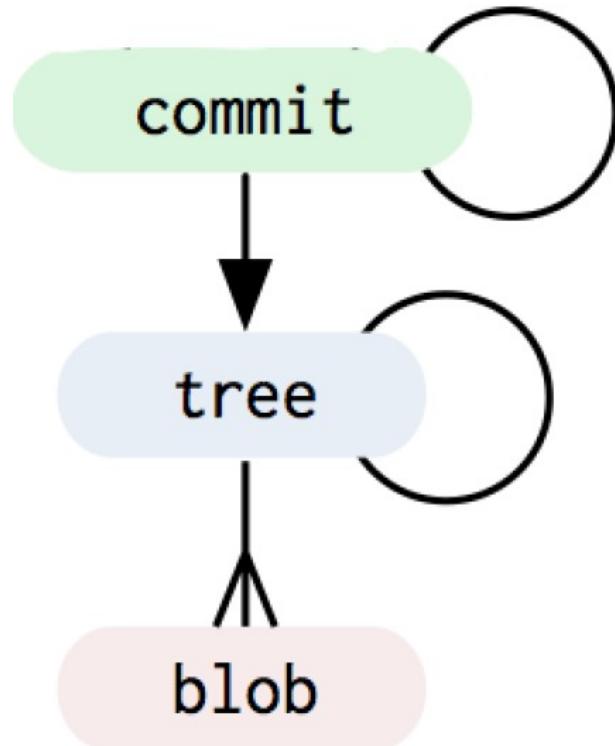


## Comment stocker les différents instantanés ?

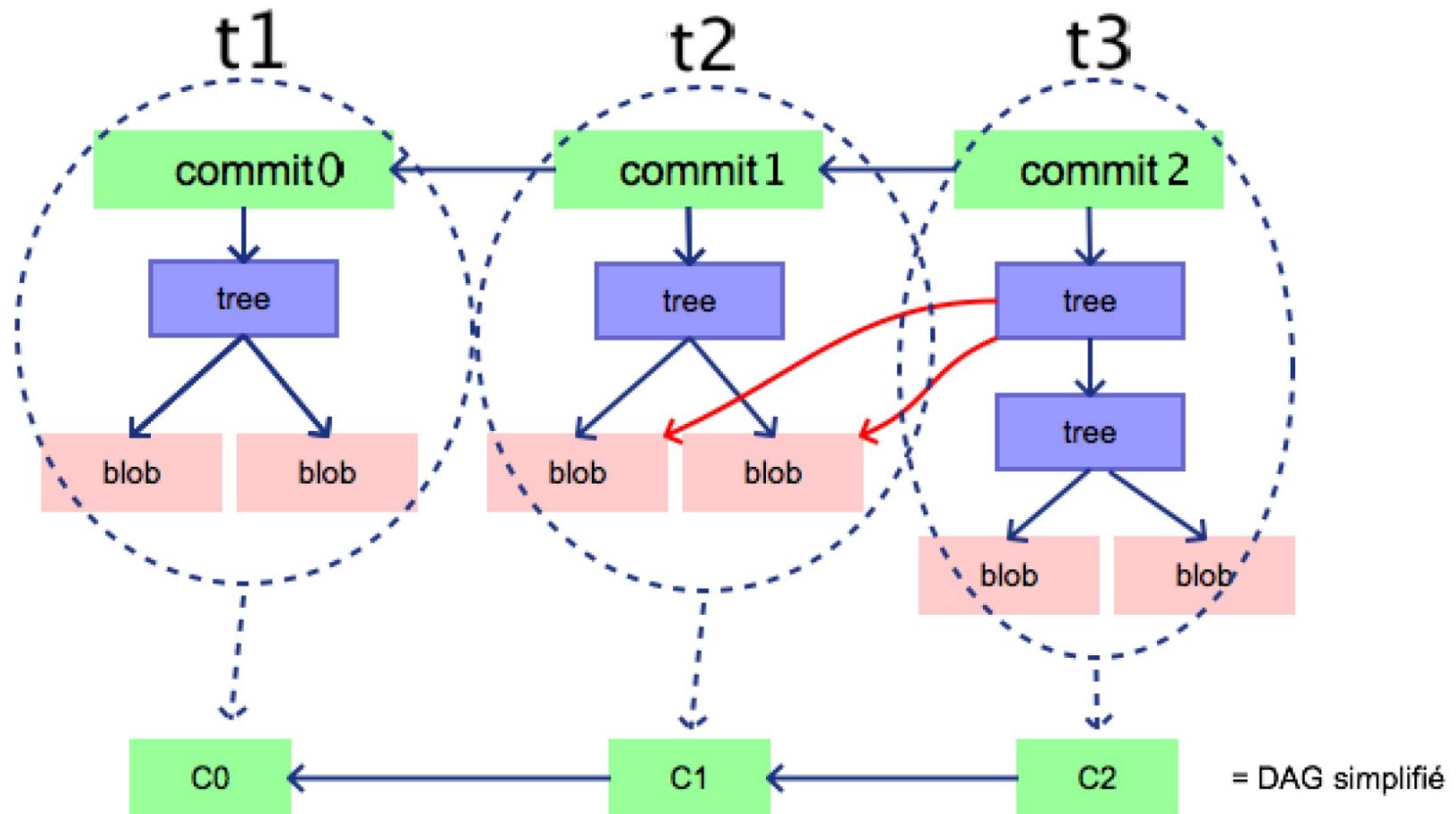


**Structure de données => DAG** : Directed Acyclic Graph (Graphe orienté acyclique) => graphe sans boucle fermée

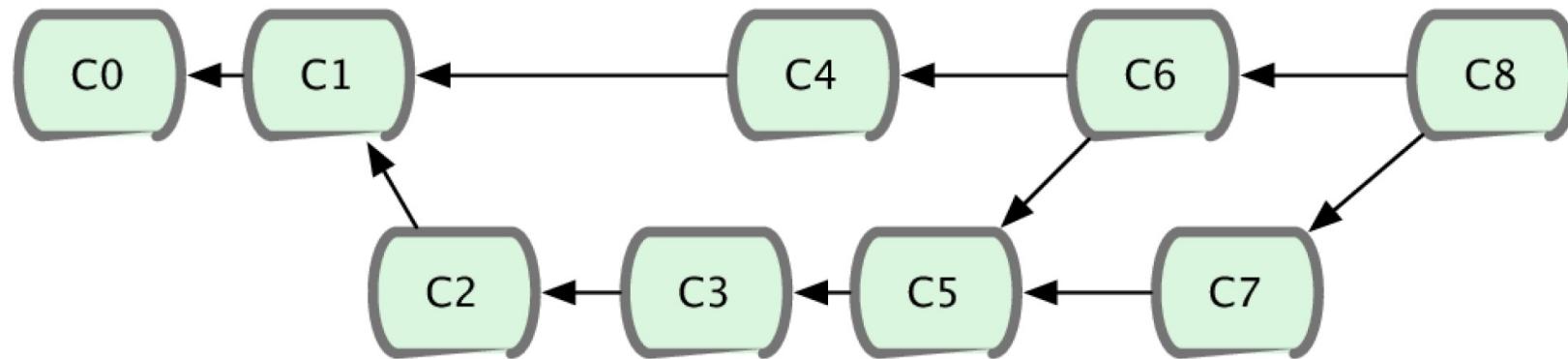
## Comment stocker les différents instantanés ?



### Comment stocker les différents instantanés ?



Comment stocker les différents instantanés ?

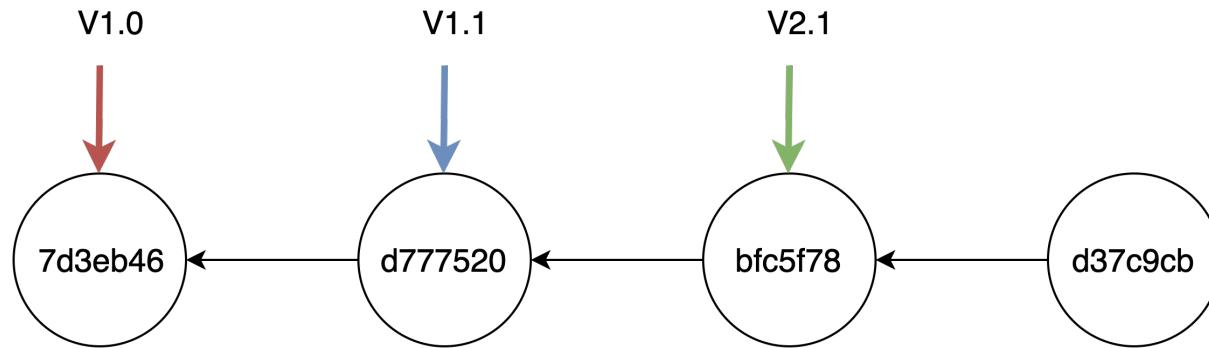


Graph (DAG) = l'ensemble des sauvegardes (commits)

## Différents types de référence

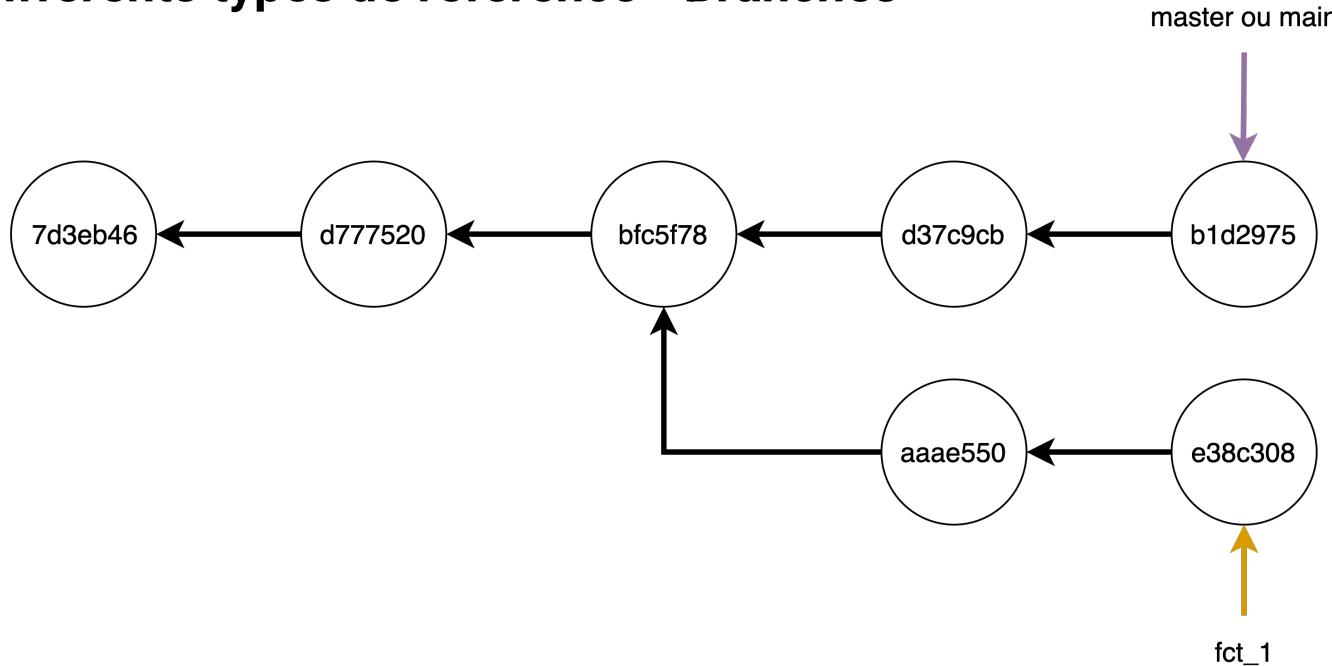
- Tag => marquer un commit
- Branches => permet de voir l'historique des commits
- HEAD => naviguer dans le graphe

## Différents types de référence - Tag



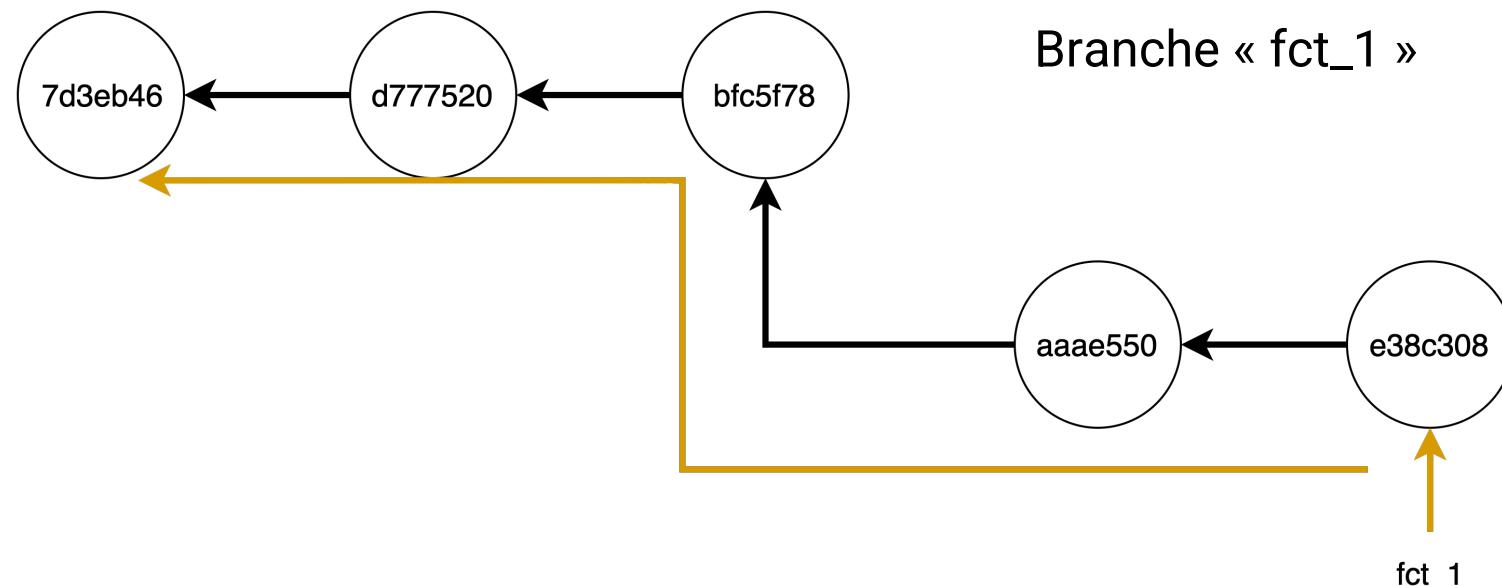
- Permet de marquer un commit particulier (p.e. : numéro de version)
- Pointeur fixe
- Tag = fichier contenant le SHA1 d'un commit spécifique.  
[.git/refs/tags/nomTag]

## Différents types de référence - Branches

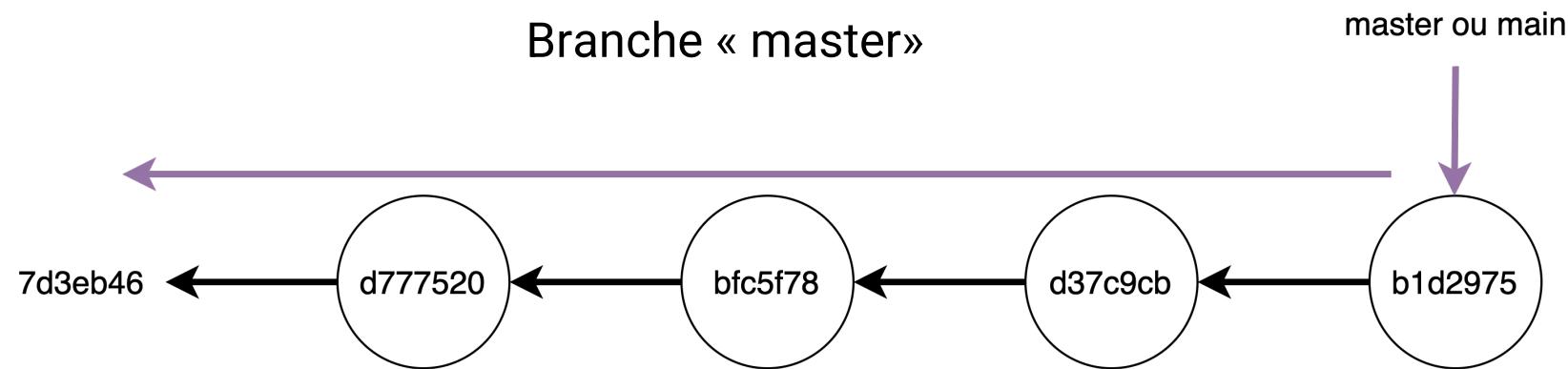


- Branche = référence qui pointe sur le SHA1 du dernier commit.
- Une branche est en fait un fichier de 20 octets contenant le SHA1
- Branche master ou main = Branche principale. [.git/refs/heads/master]
- A chaque commit, une branche se déplace automatiquement.

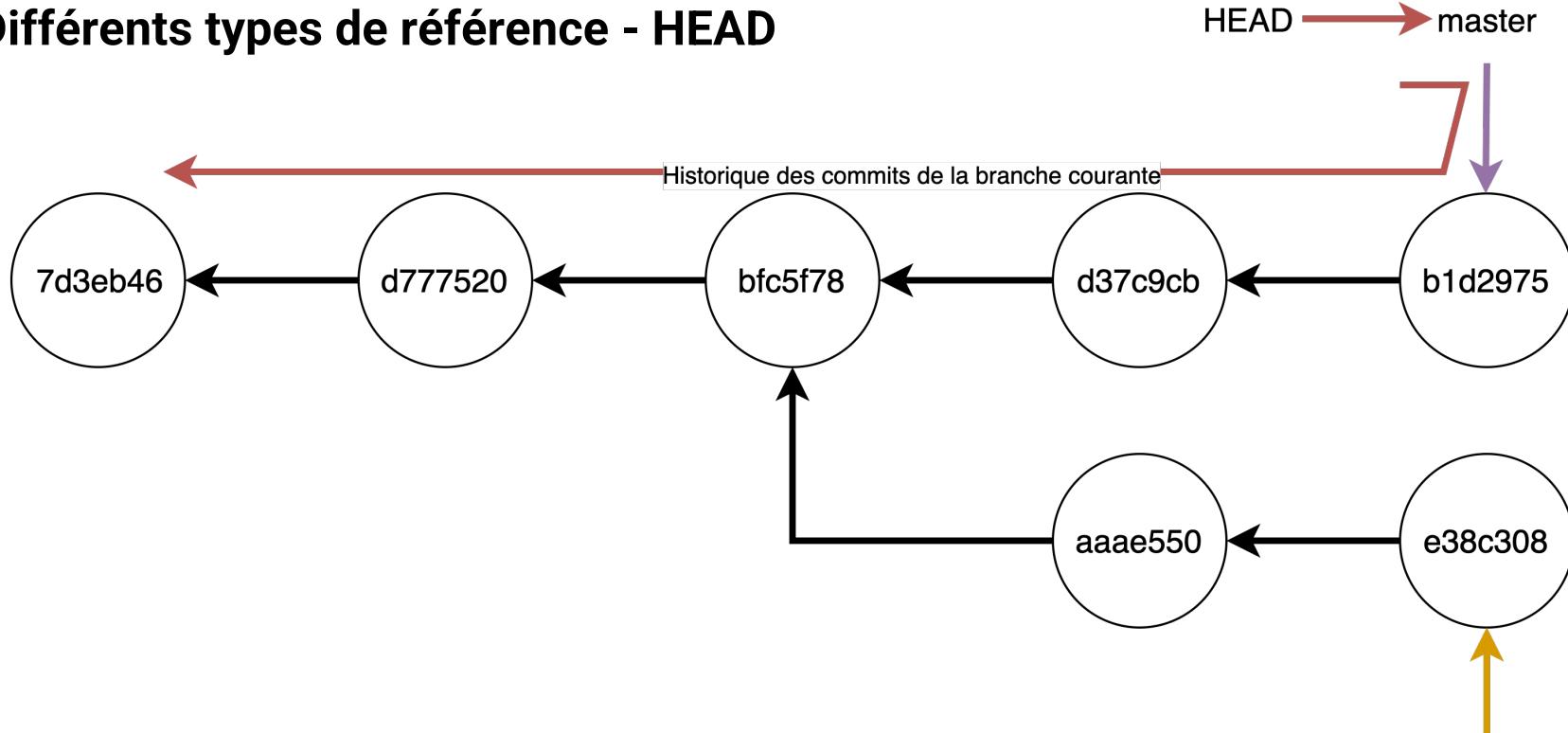
## Différents types de référence - Branches



## Différents types de référence - Branches

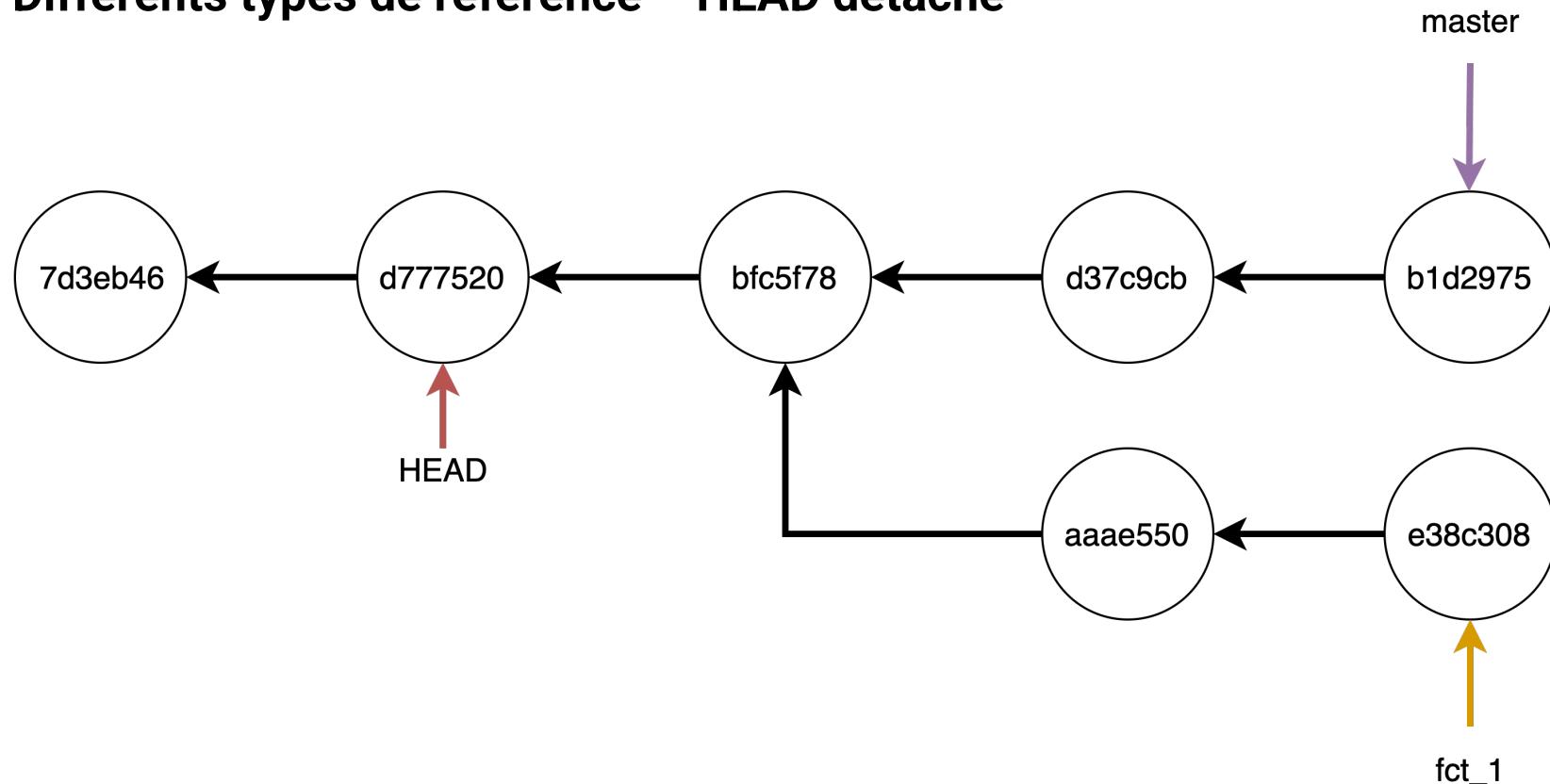


### Différents types de référence - HEAD



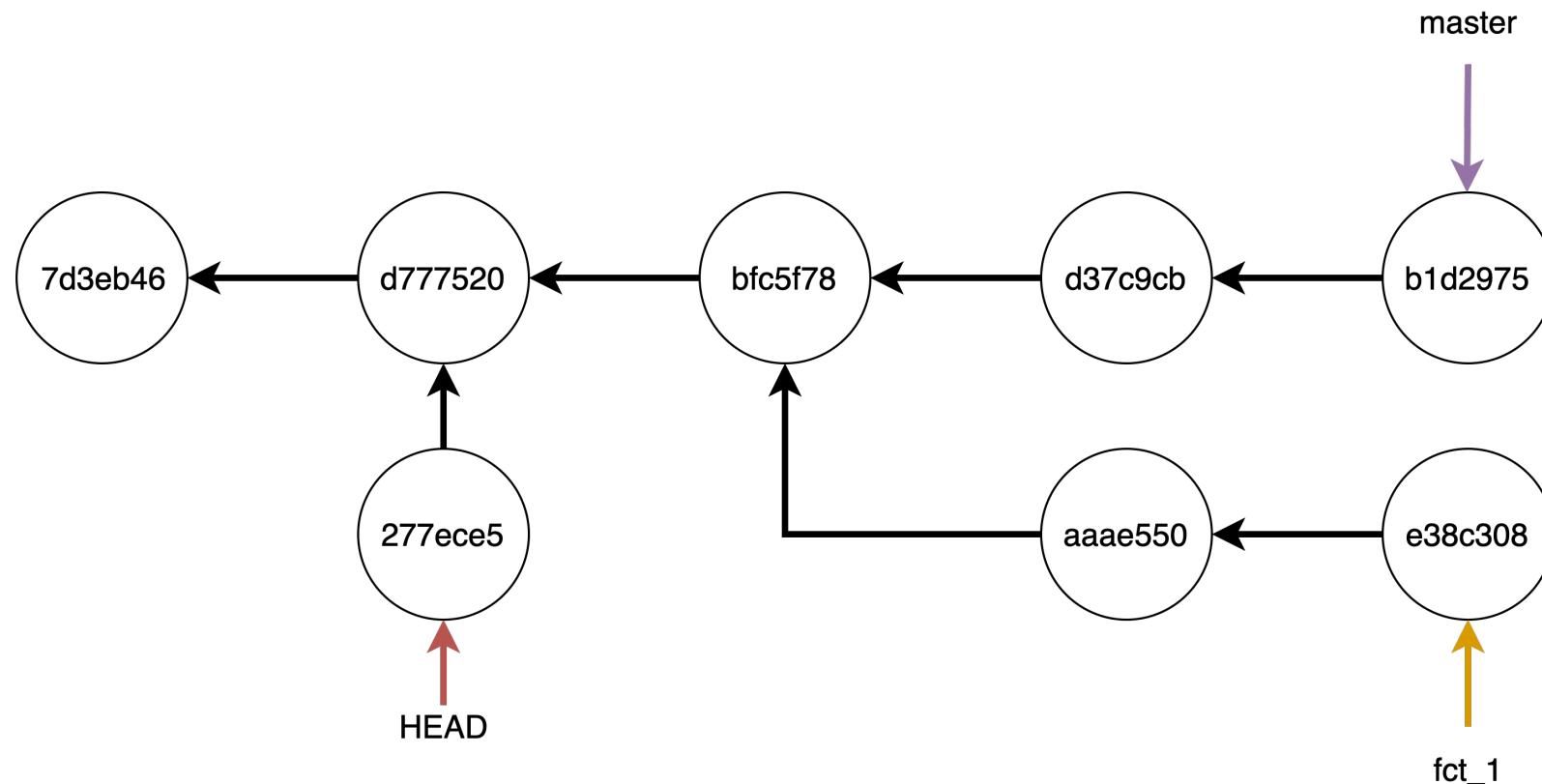
- Référence HEAD = fichier contenant la référence d'une branche [./HEAD].
- Donne l'état du répertoire de travail.
- Permet de naviguer dans le graphe en s'attachant à une branche => historique des commits de la branche courante

## Différents types de référence – HEAD détaché

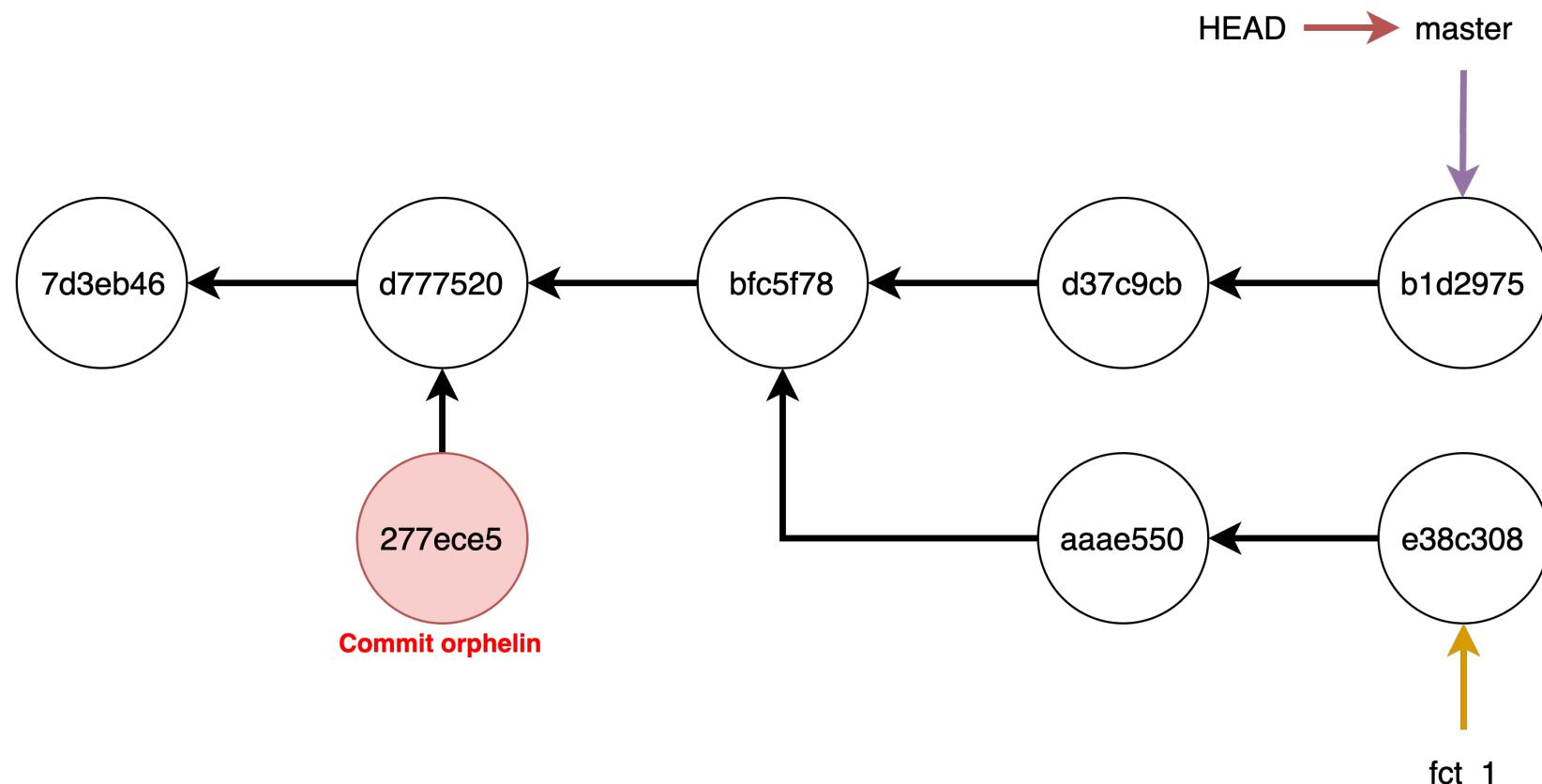


HEAD détaché = HEAD qui pointe directement sur un commit

## Différents types de référence – HEAD détaché – Crédit d'un commit

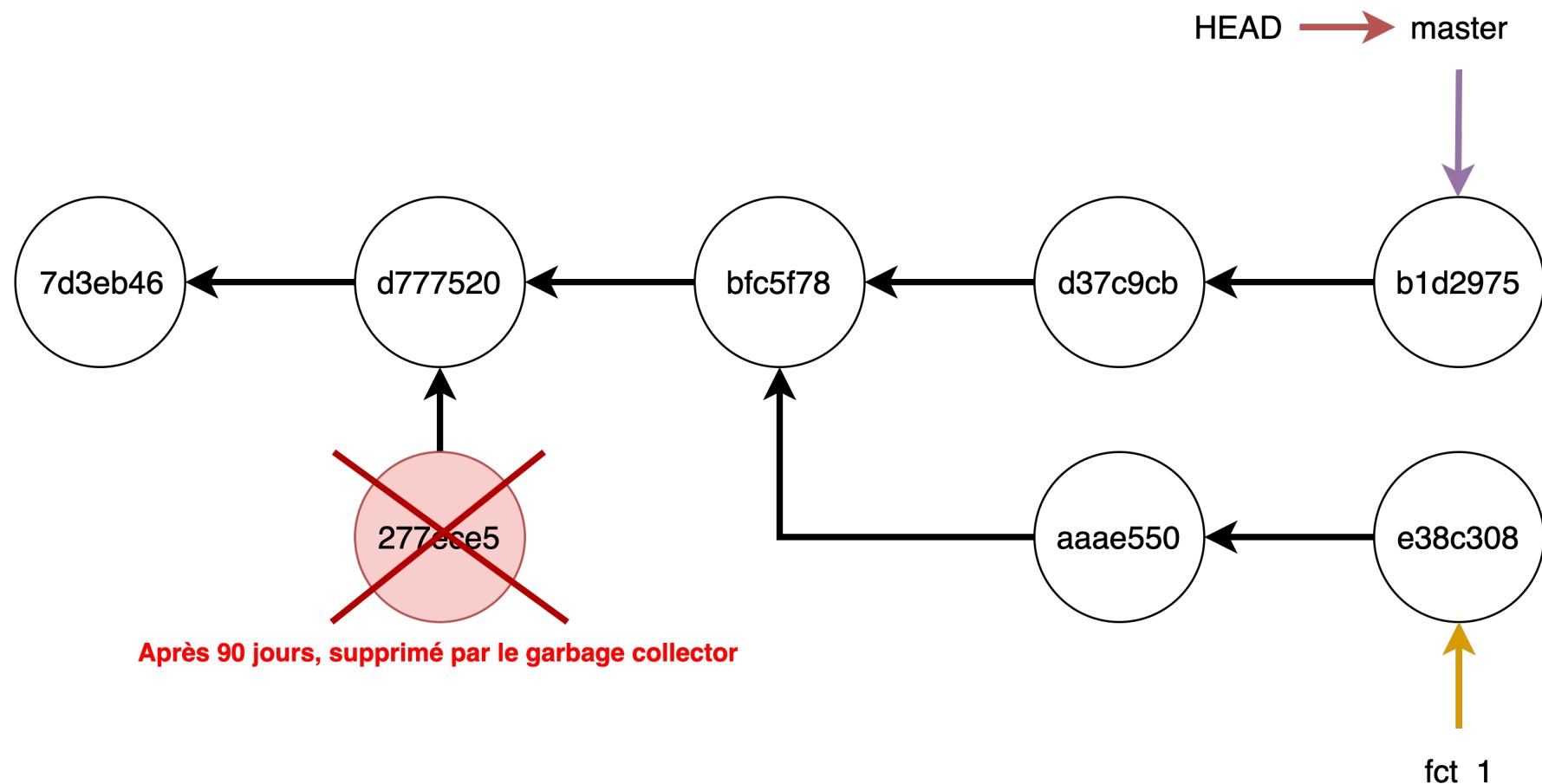


## Différents types de référence – HEAD détaché – Revenir sur master

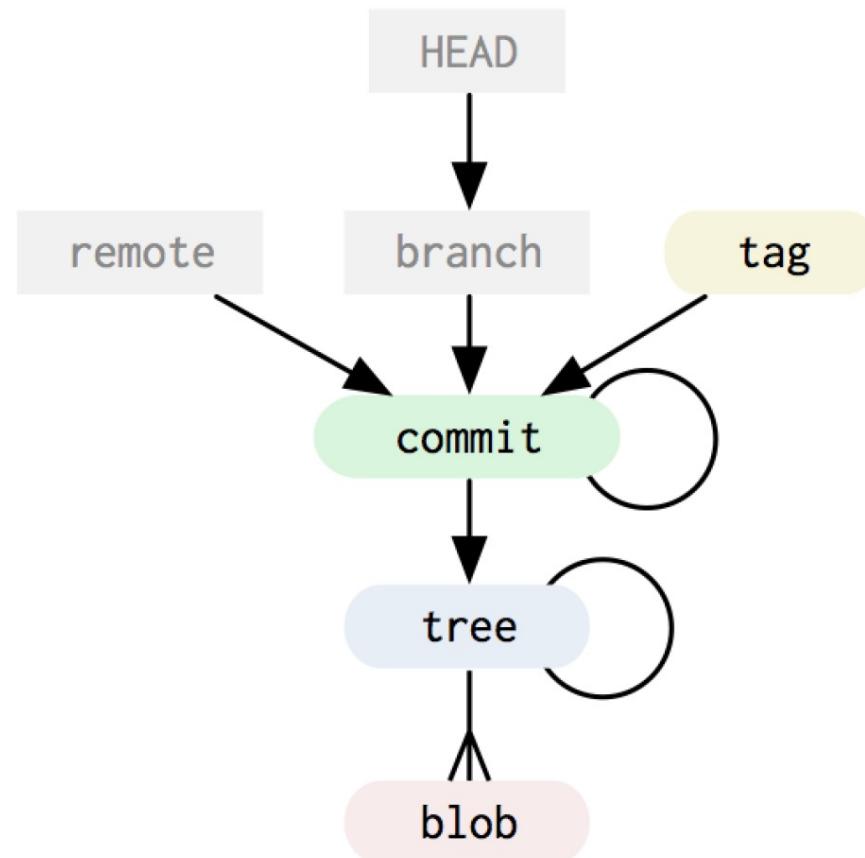


Commit orphelin= Commit qui n'est plus référencé

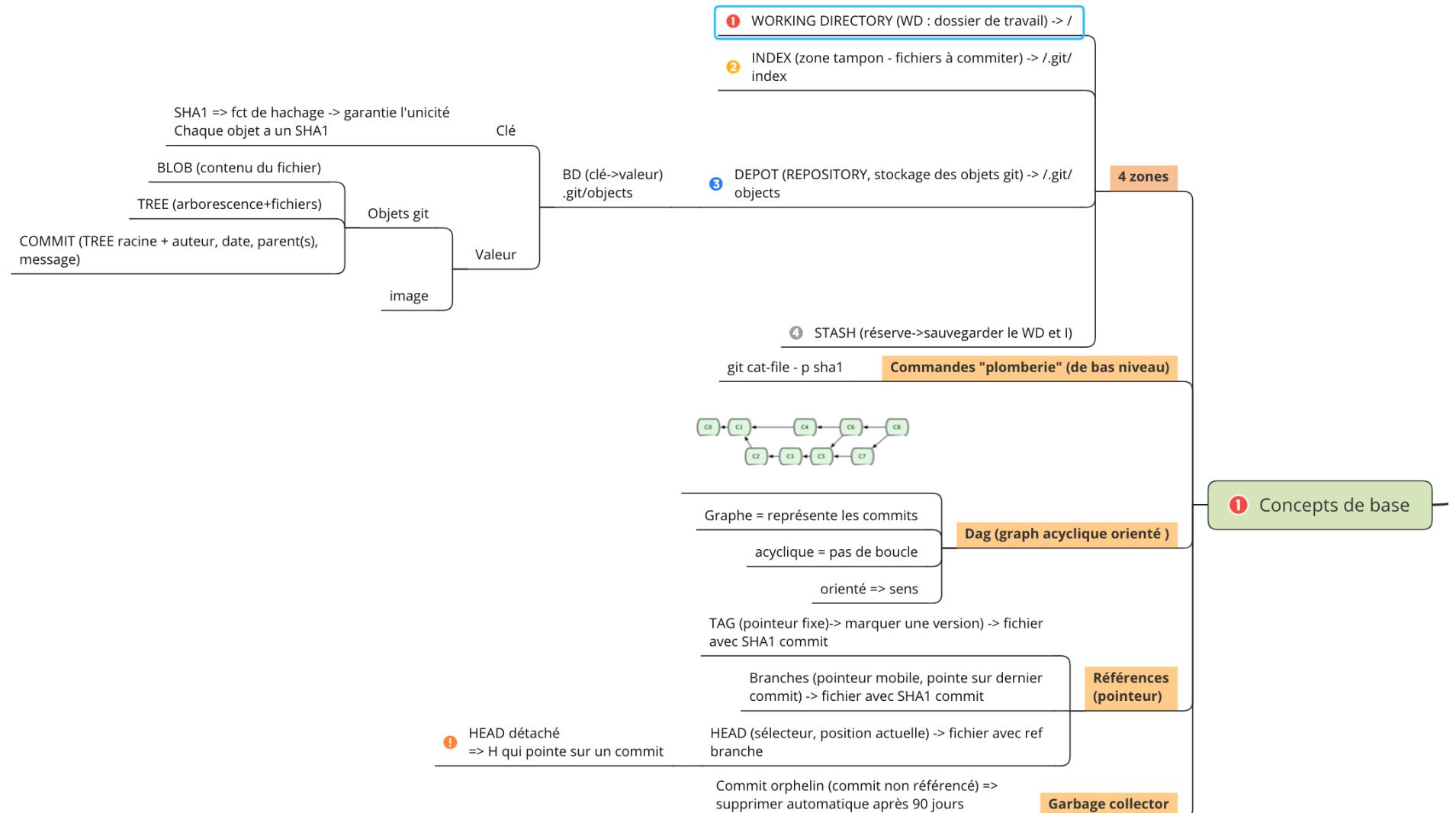
## Différents types de référence – HEAD détaché – Garbage collector



## Résumé objets & références



# Concepts de base – Synthèse



# Commandes

- **Système** : paramètres liés à tous les utilisateurs du système (fichier : /etc/gitconfig )

*Commande* : \$ git config --system <paramètre> <valeur>

- **Global** : paramètres spécifiques à chaque utilisateur (~/.gitconfig)

*Commande* : \$ git config --global <paramètre> <valeur>

- **Local** : paramètres spécifiques à un unique dépôt (.git/config)

*Commande* : \$ git config <paramètre> <valeur>

<paramètre> : section.nomParametre

### Identité

Commande :

```
$ git config --global user.email gianni.tricarico@heh.be  
$ git config --global user.name «TRIGianni»
```

Fichier : (~/.gitconfig)

**[user]**

**name** = TRIGianni  
**email** = gianni.tricarico@heh.be

**section**

**Paramètre = valeur**

**Configuration minimale pour réaliser des commits**

Fichier : (~/.gitconfig)

```
[alias]
lg = log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %$'

chk = checkout
```

\$ git chk      Équivalent à      \$ git checkout

Editer le fichier : /projet/.gitignore

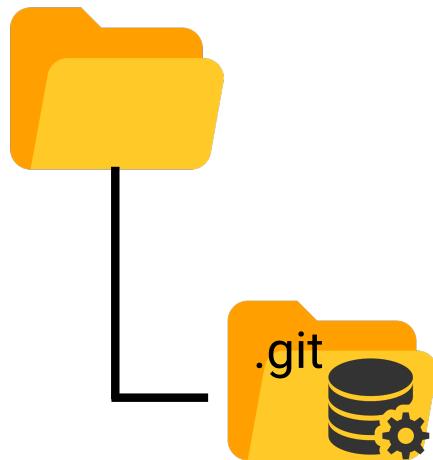
\$ cat .gitignore

```
# pas de fichier .a
*.a
# mais suivre lib.a malgré la règle précédente
!lib.a
# ignorer uniquement le fichier TODO à la racine du projet
/TODO
# ignorer tous les fichiers dans le répertoire build
build/
# ignorer doc/notes.txt, mais pas doc/server/arch.txt
doc/*.txt
# ignorer tous les fichiers .txt sous le répertoire doc/
doc/**/*.txt
```

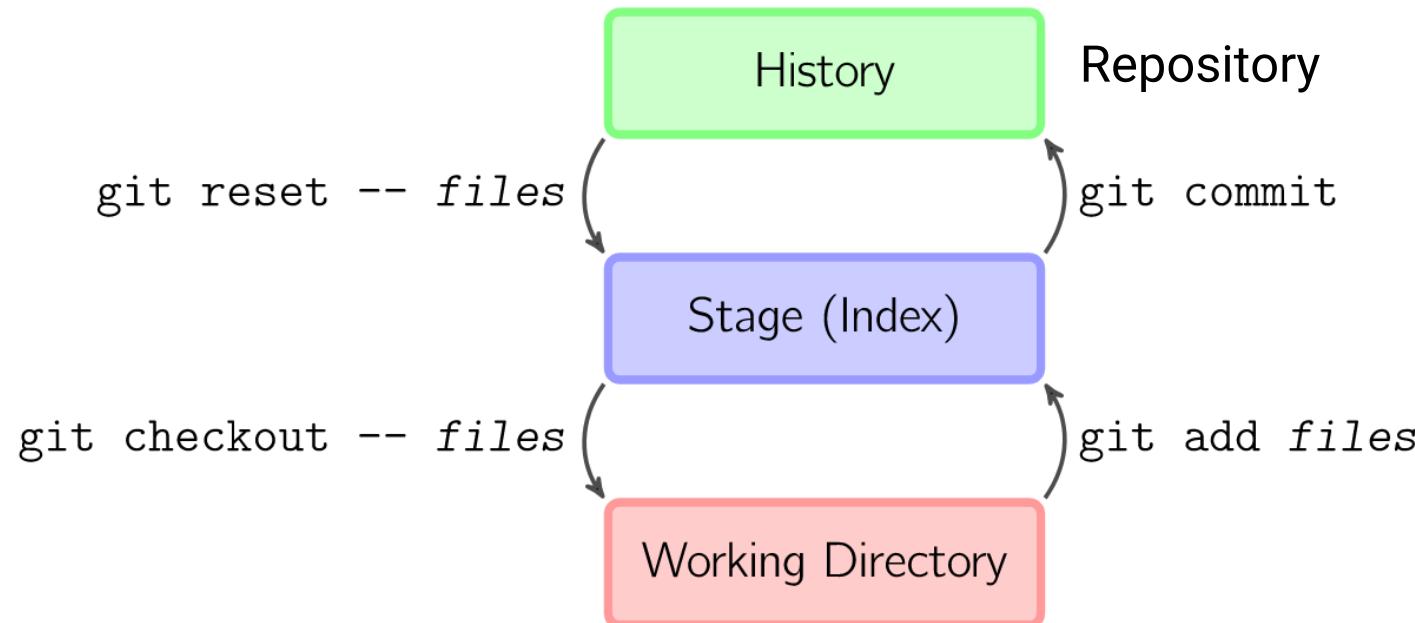
Règles prédéfinies : <https://github.com/github/gitignore>

```
$ git init [nom_dépôt]
```

Crée un dépôt local vide dans le dossier courant ou en créer un avec le nom spécifié.

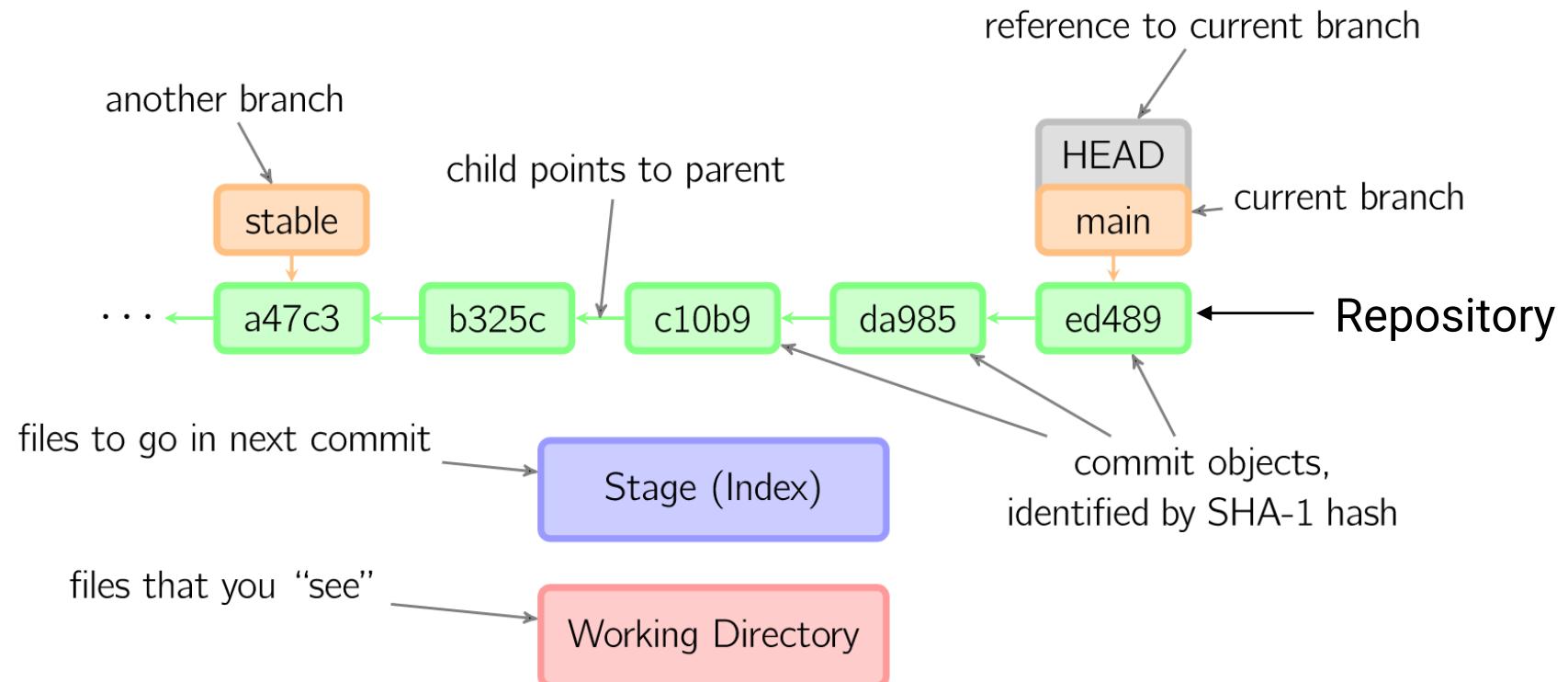


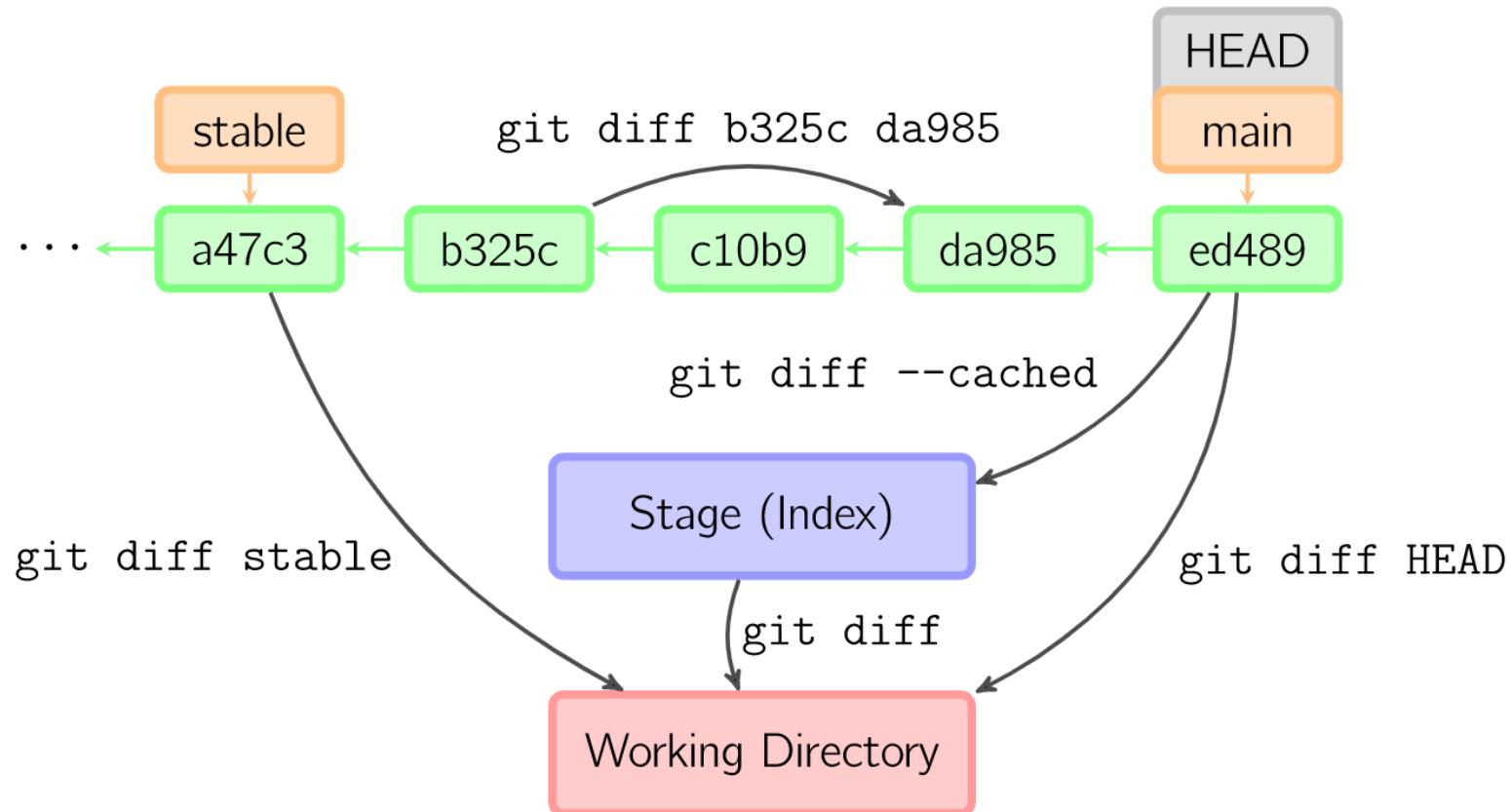
```
{21:52:25}~/tests/monProjet (master #) $ tree -a
.
└── .git
    ├── HEAD
    ├── config
    ├── description
    ├── hooks
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    └── pre-push.sample
    ├── pre-rebase.sample
    └── prepare-commit-msg.sample
        └── update.sample
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        └── heads
    └── tags
AD -- files
```

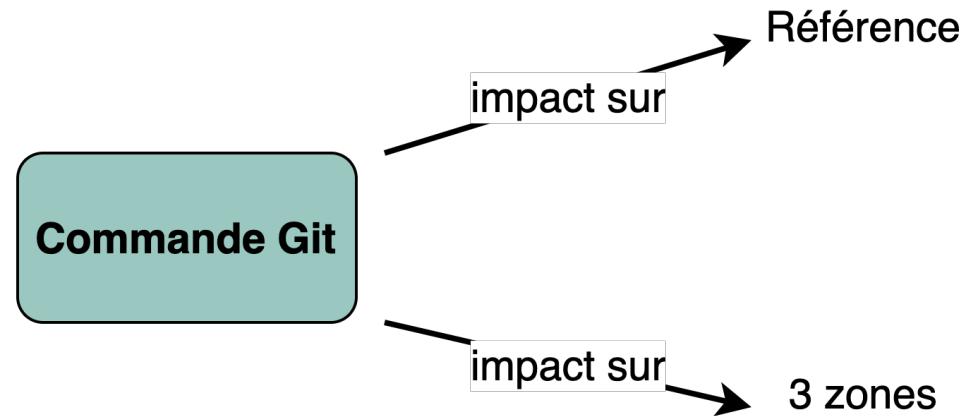


- *git add fichiers* : copie les *fichiers* vers le stage.
- *git commit* : fait un "cliché" du stage sous la forme d'un commit.
- *git reset – fichiers* : supprime les fichiers du stage ; i.e. elle copie les *fichiers* du dernier commit vers le stage.
- *git checkout – fichiers* : copie les *fichiers* du stage vers la working copy

# Démo



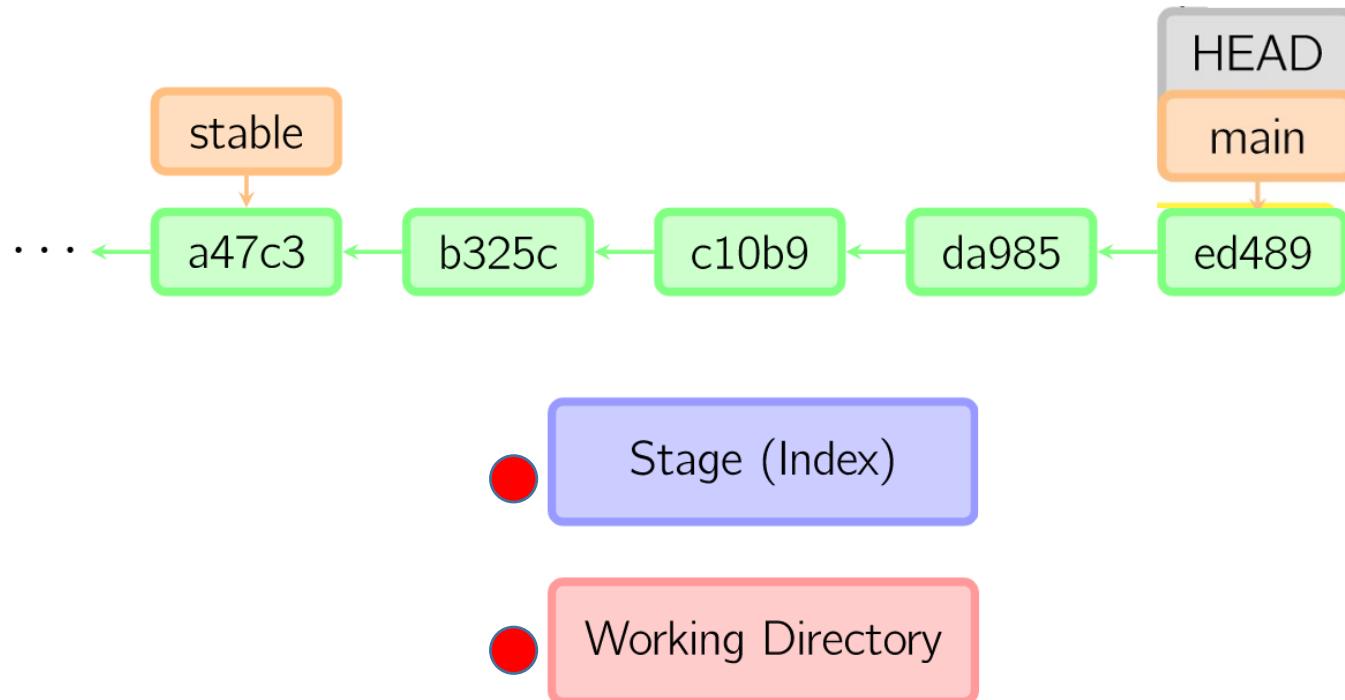




**Il n'existe aucune commande qui permet de supprimer ou modifier un commit.**

**Seul le garbage collector est capable de supprimer un commit orphelin.**

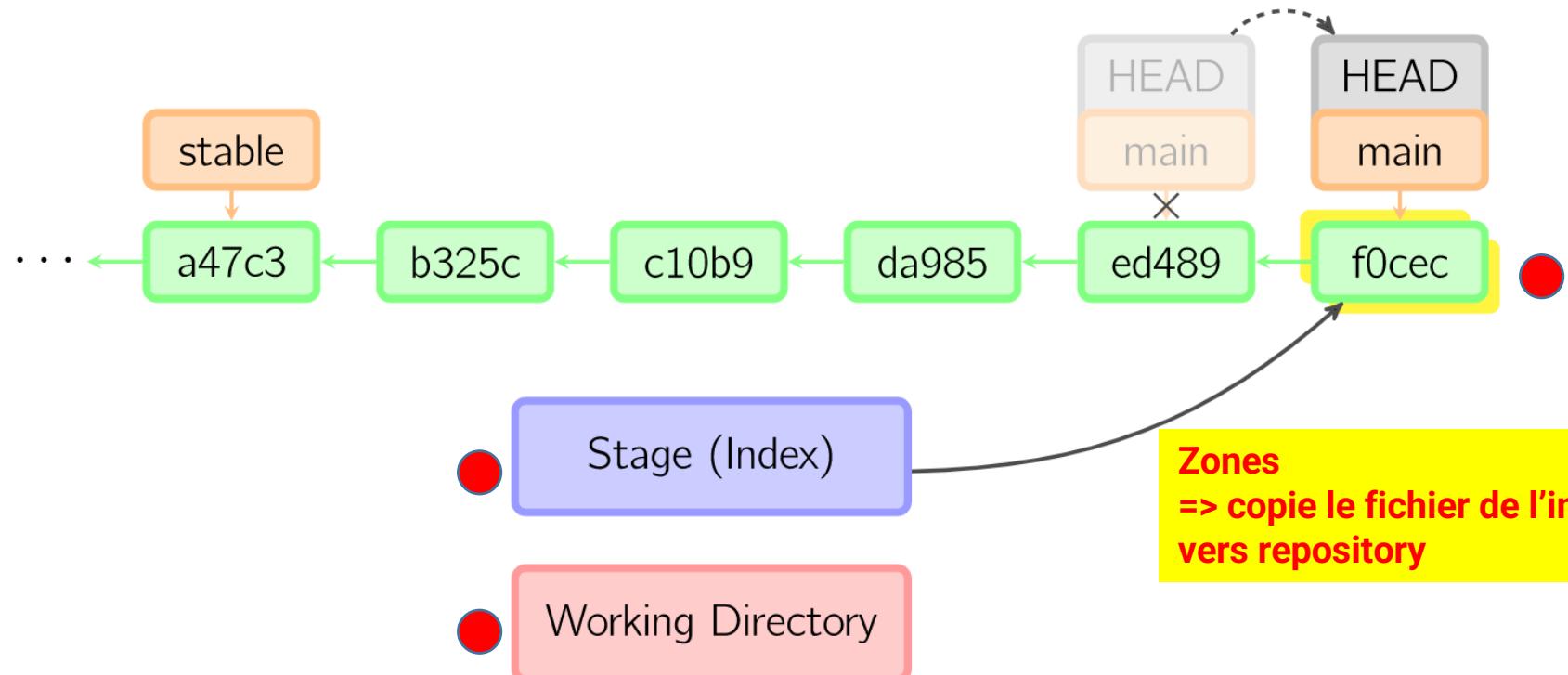
`git commit -m « message »`



Copie les fichiers présents dans la zone d'index vers le repository.

```
git commit -m « message »
```

Référence  
=> déplacement branche + HEAD



### git log

```
commit 6cd1a3779636182eef50c84efd0da2550b651d10 (HEAD -> master, origin/master)
Merge: 7ea6394 d426263
Author: TRIGianni <gianni.tricarico@heh.be>
Date:   Thu Sep 24 14:21:25 2020 +0200

    Merge pull request #2 from Multydrive/master

    test

commit d4262633c1e98298138c5021024f9e6aa7edf92e
Author: Andreas <andreas.basso@std.heh.be>
Date:   Thu Sep 24 14:14:23 2020 +0200

    git@github.com:Multydrive/devDemo.gitgit@github.com:Multydrive/devDemo.gitgi
    t@github.com:Multydrive/devDemo.git
    test

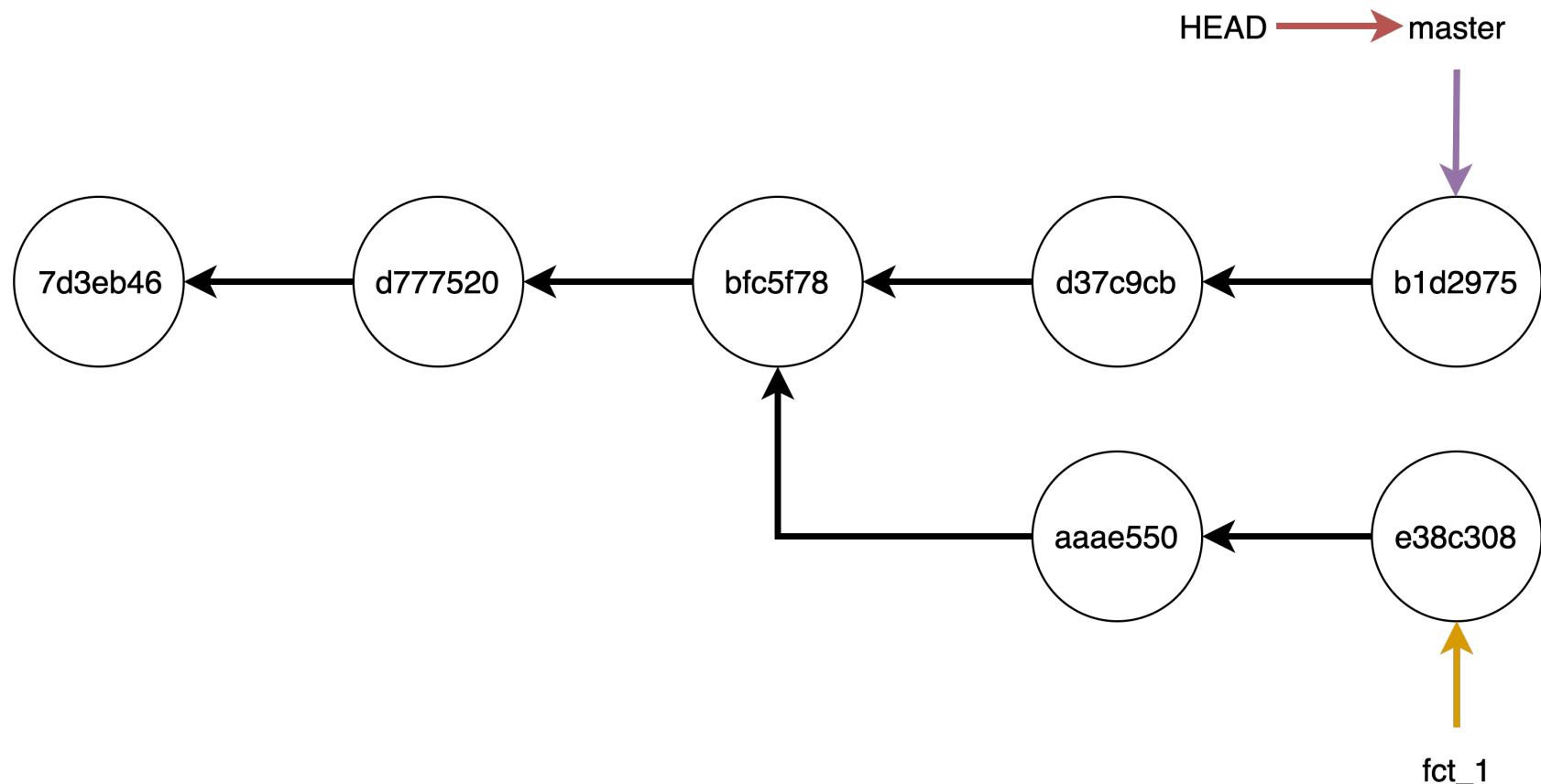
commit 7ea639403f260279a4b11c4637b80aae545097cb
Author: Gianni Tricarico <giannitricarico@MacBook-Pro-de-Gianni.local>
Date:   Wed Sep 23 10:11:07 2020 +0200

    ajout <p>
```

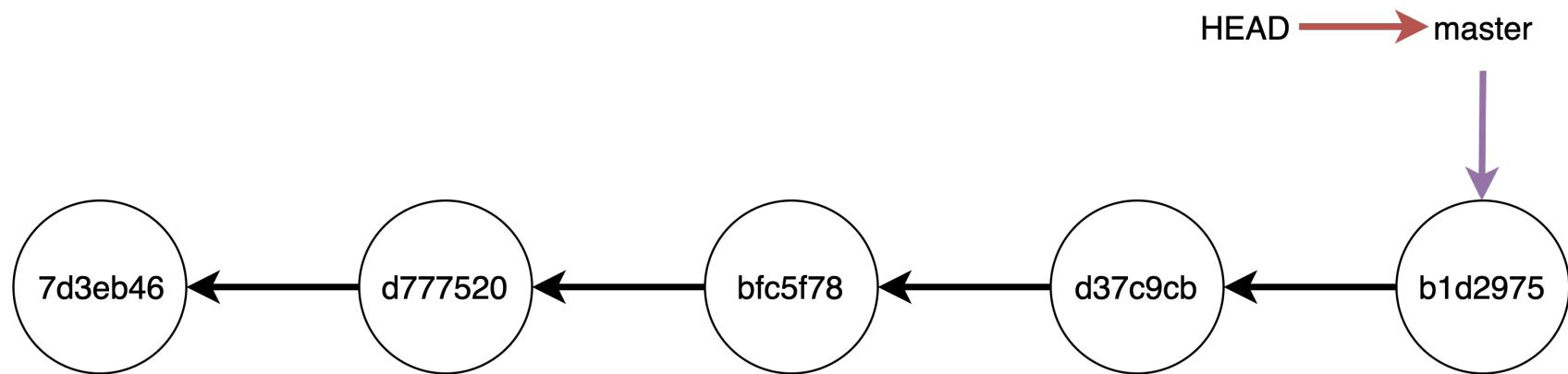
```
Git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset  
%s %Cgreen(%cr) %C(bold blue) %Creset' --abbrev-commit
```

```
* 6cd1a37 - (HEAD -> master, origin/master) Merge pull request #2 from Multydrive/master (1 year, 8  
months ago) <TRIGianni>  
|\  
| * d426263 - git@github.com:Multydrive/devDemo.gitgit@github.com:Multydrive/devDemo.gitgit@github.com  
:Multydrive/devDemo.git test (1 year, 8 months ago) <Andreas>  
|/  
* 7ea6394 - ajout <p> (1 year, 8 months ago) <Gianni Tricarico>  
* bd6fbff - fusion (1 year, 8 months ago) <Gianni Tricarico>  
|\  
| * 277ece5 - (style) ajout # (1 year, 8 months ago) <Gianni Tricarico>  
| * e38c308 - background-color : green (1 year, 8 months ago) <Gianni Tricarico>  
| * aaae550 - style.css + p (1 year, 8 months ago) <Gianni Tricarico>  
* | b1d2975 - ajout * (1 year, 8 months ago) <Gianni Tricarico>  
* | d37c9cb - correction bug (1 year, 8 months ago) <Gianni Tricarico>  
|/  
* bfc5f78 - ajout <p> (1 year, 8 months ago) <Gianni Tricarico>  
* d777520 - ajout balise h1 (1 year, 8 months ago) <Gianni Tricarico>  
* 7d3eb46 - (tag: V1.0) création du fichier index.html (1 year, 8 months ago) <Gianni Tricarico>
```

## Commandes – Visualiser l'historique des commits

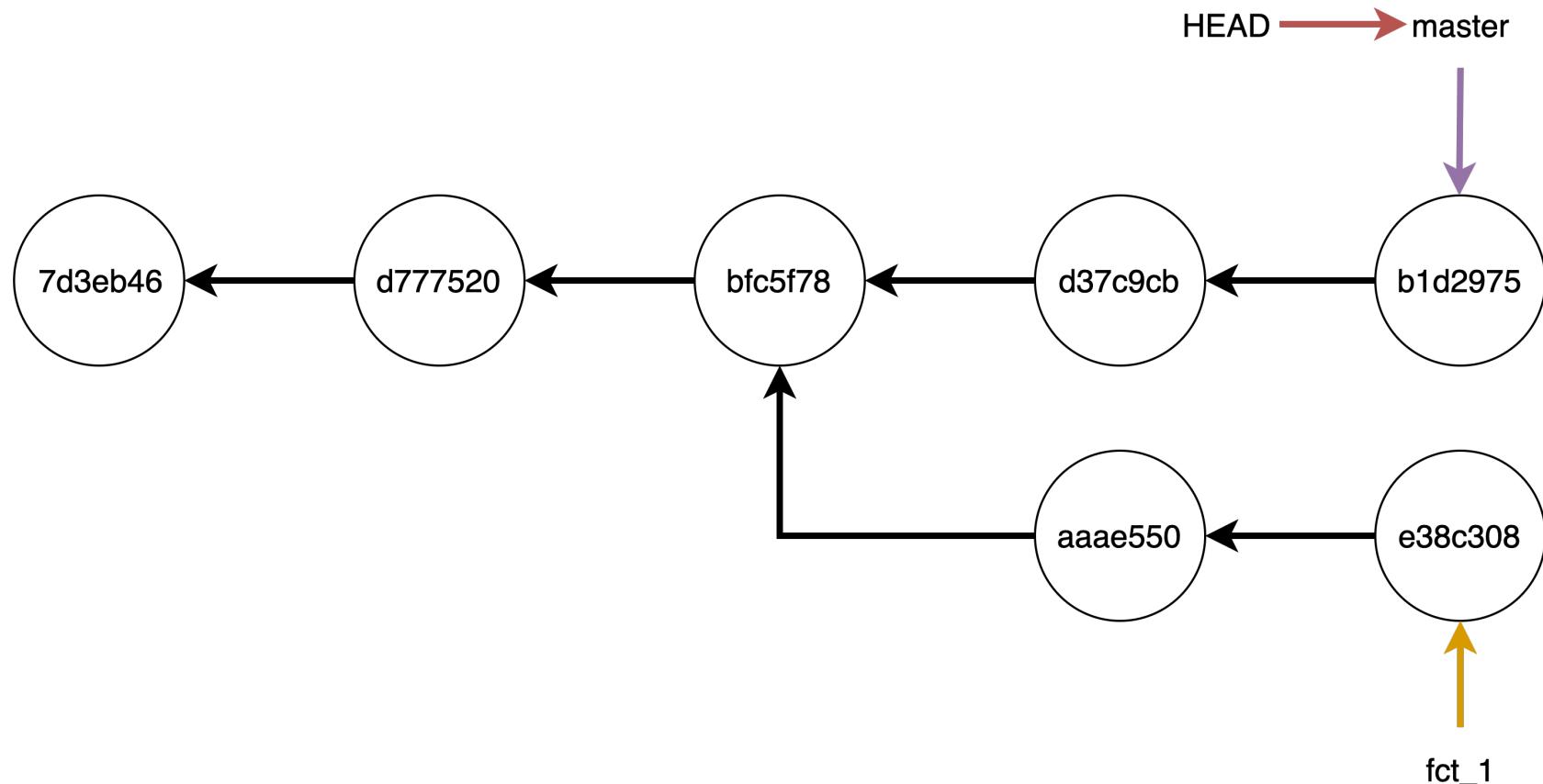


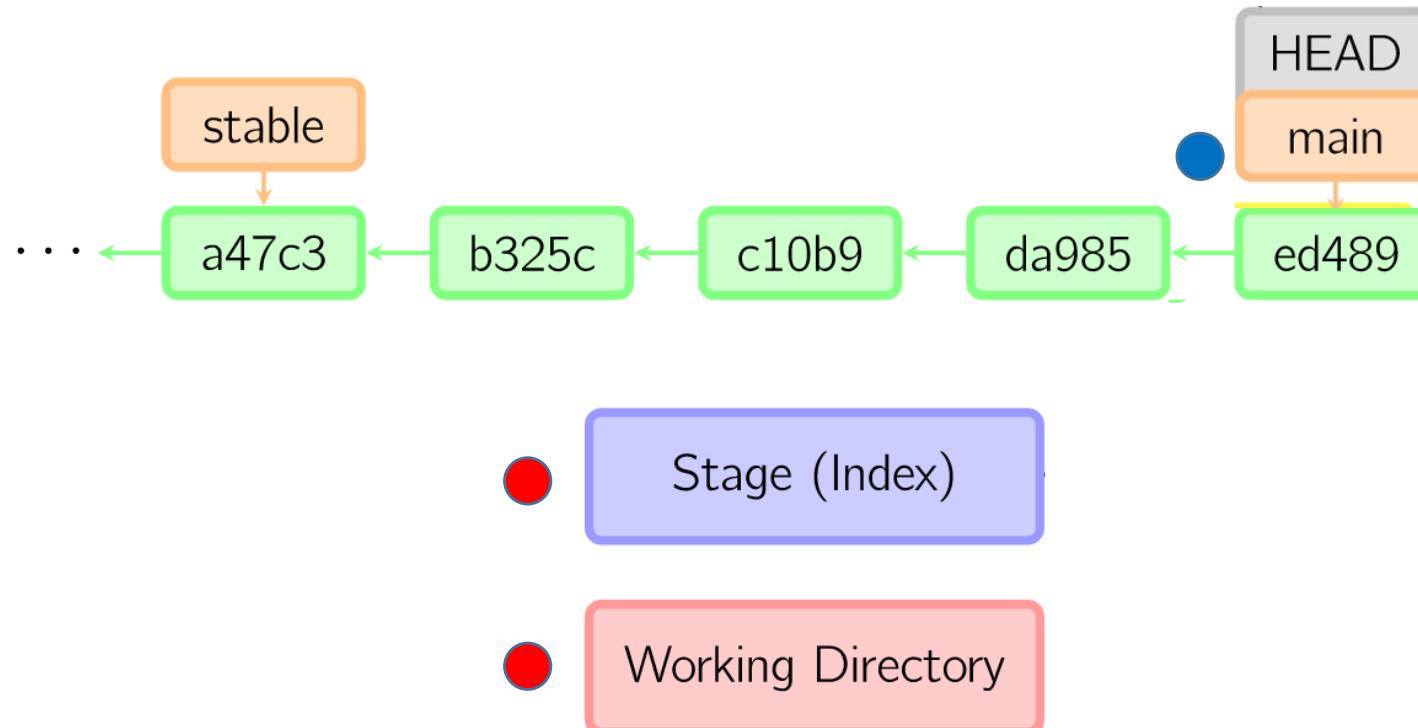
git log



Visualiser l'historique des commits de la branche courante.

git log --all

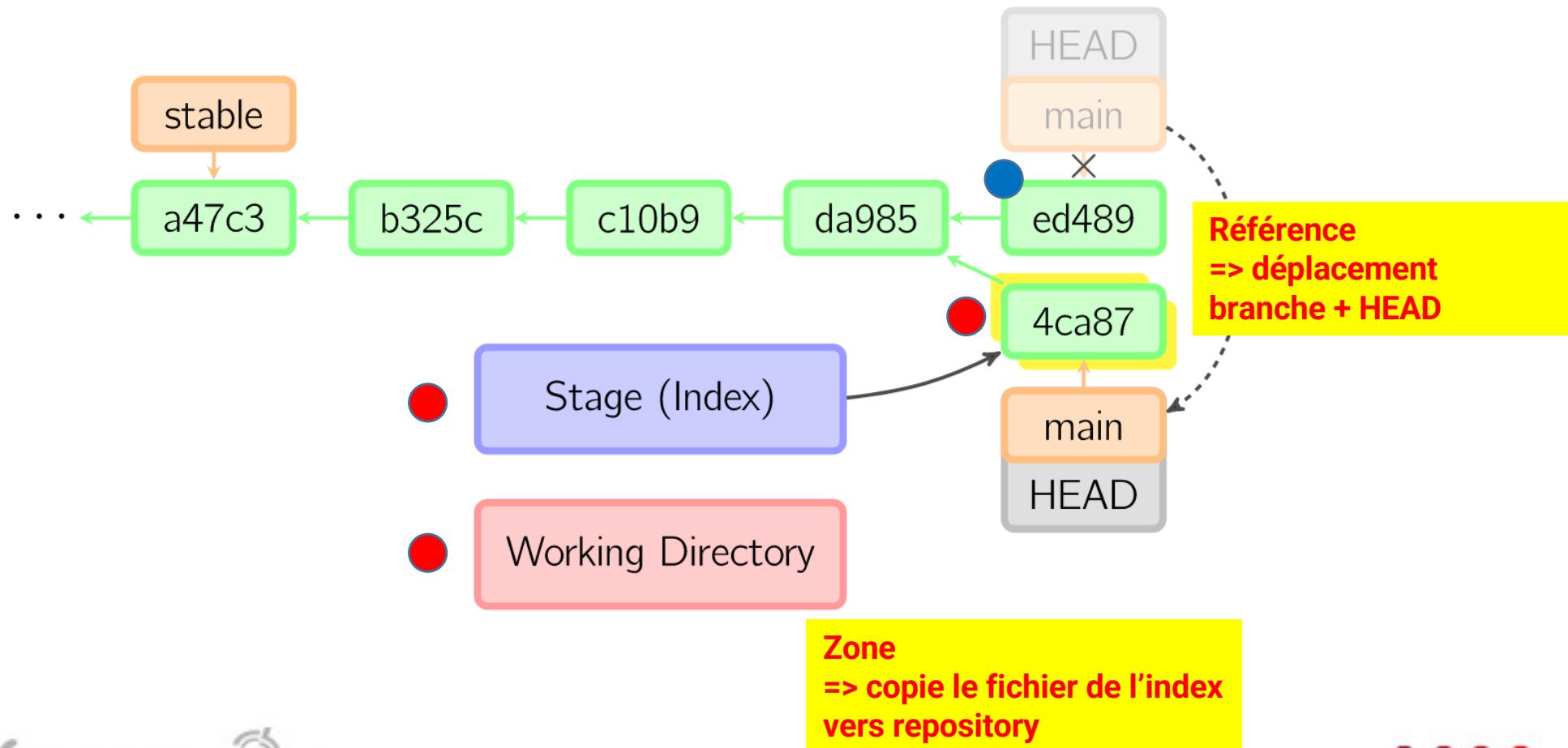




Permet de corriger (contenu et/ou message) le dernier commit.

## Commandes – Réécriture de l'historique

git commit --amend



Git conserve un journal (fichier situé : .git/logs/HEAD) de tous les déplacements de la référence HEAD, accessible grâce à la commande :

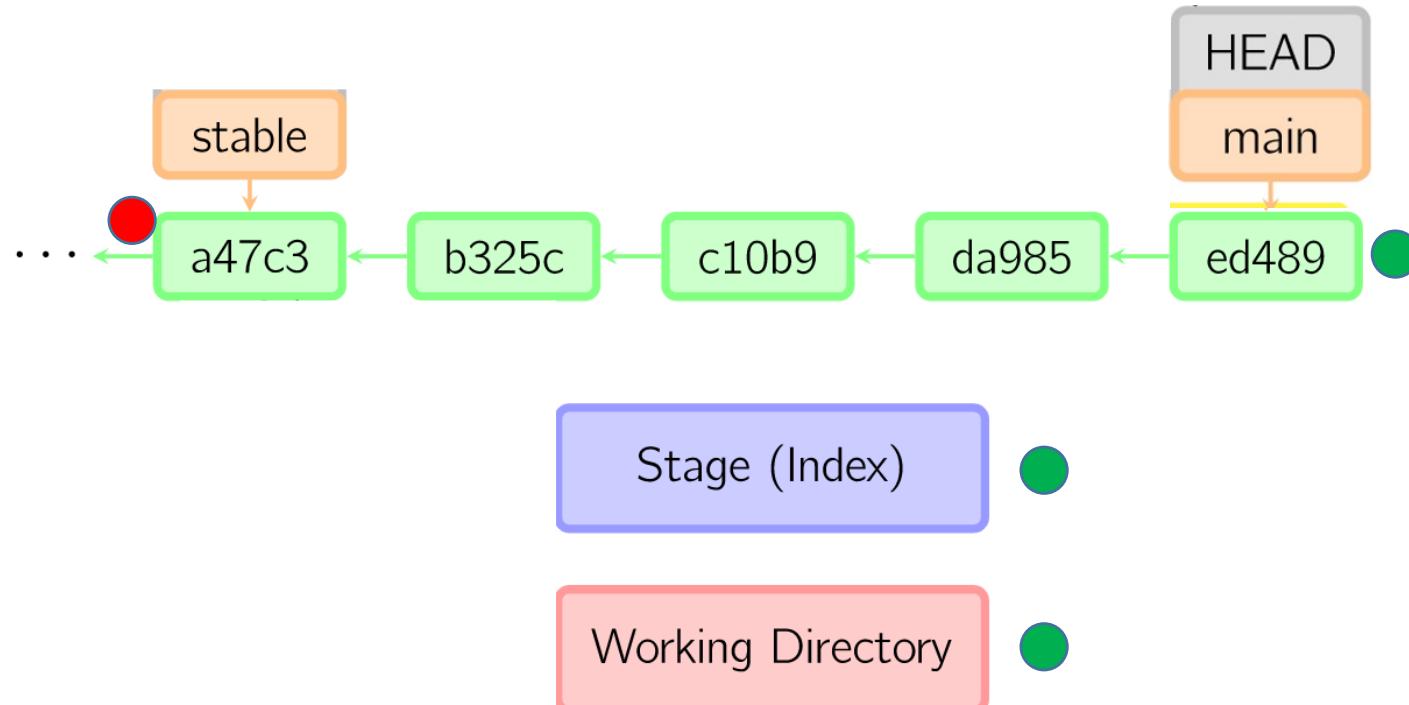
```
git reflog
```

Commande permettant de retrouver des commits orphelins.

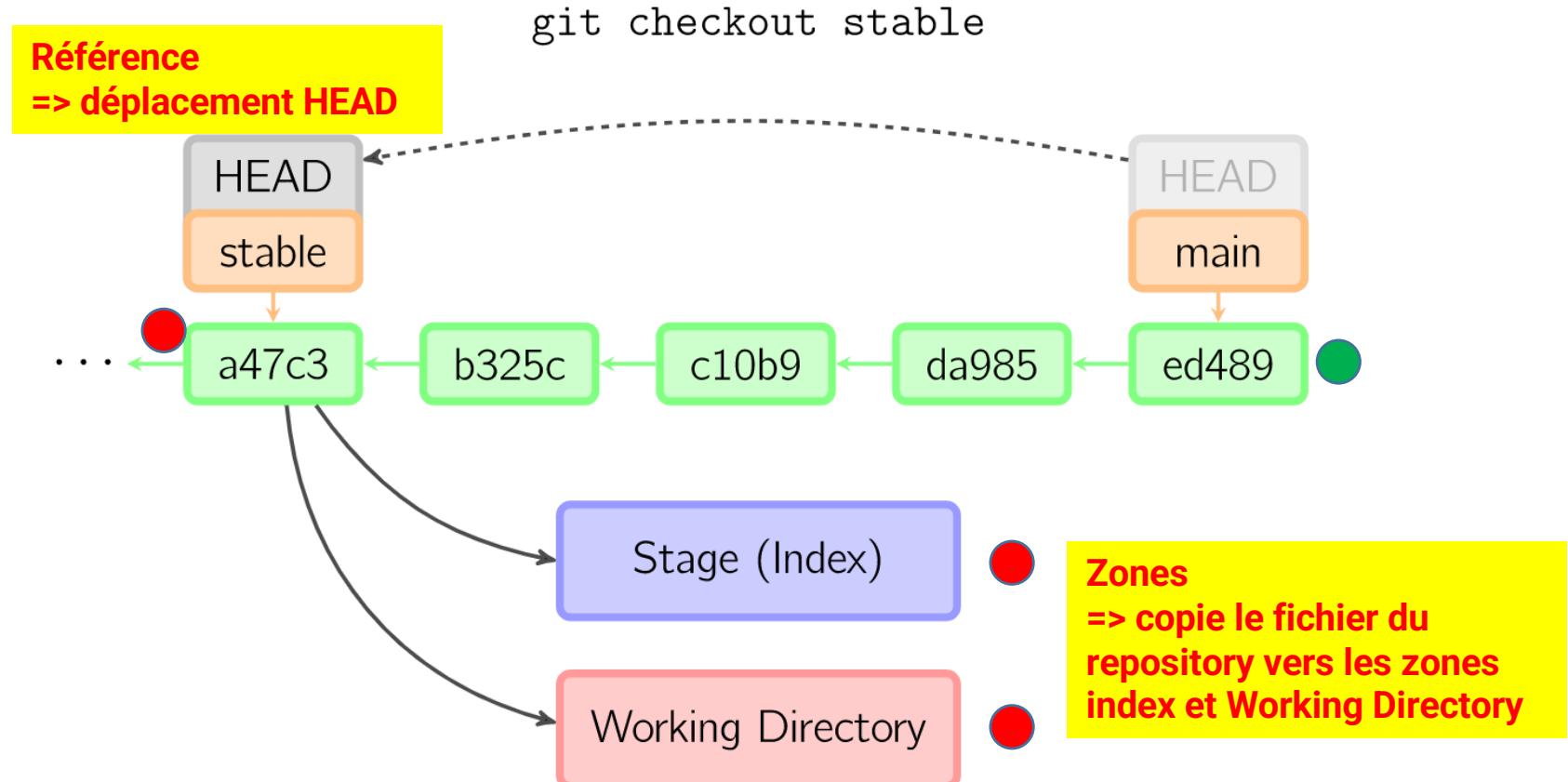
```
6cd1a37 (HEAD -> master, origin/master) HEAD@{0}: checkout: moving from 6cd1a3779636182eef50c84efd0da2
550b651d10 to master
6cd1a37 (HEAD -> master, origin/master) HEAD@{1}: checkout: moving from master to origin/master
6cd1a37 (HEAD -> master, origin/master) HEAD@{2}: merge origin/master: Fast-forward
7ea6394 HEAD@{3}: commit: ajout <p>
bd6fbff HEAD@{4}: commit (merge): fusion
b1d2975 HEAD@{5}: checkout: moving from style to master
277ece5 (style) HEAD@{6}: commit: ajout #
e38c308 HEAD@{7}: checkout: moving from master to style
b1d2975 HEAD@{8}: commit: ajout *
d37c9cb HEAD@{9}: checkout: moving from bfc5f78b89870db0663b6615ce5472034f39e005 to master
bfc5f78 HEAD@{10}: checkout: moving from master to bfc5
d37c9cb HEAD@{11}: reset: moving to d37c9cb
627023d HEAD@{12}: merge style: Merge made by the 'recursive' strategy.
d37c9cb HEAD@{13}: checkout: moving from style to master
e38c308 HEAD@{14}: checkout: moving from master to style
d37c9cb HEAD@{15}: commit: correction bug
bfc5f78 HEAD@{16}: checkout: moving from style to master
e38c308 HEAD@{17}: checkout: moving from master to style
bfc5f78 HEAD@{18}: checkout: moving from style to master
e38c308 HEAD@{19}: reset: moving to HEAD
e38c308 HEAD@{20}: checkout: moving from master to style
bfc5f78 HEAD@{21}: checkout: moving from style to master
e38c308 HEAD@{22}: checkout: moving from master to style
```

```
git checkout [tag | sha1 | branche]
```

```
git checkout stable
```

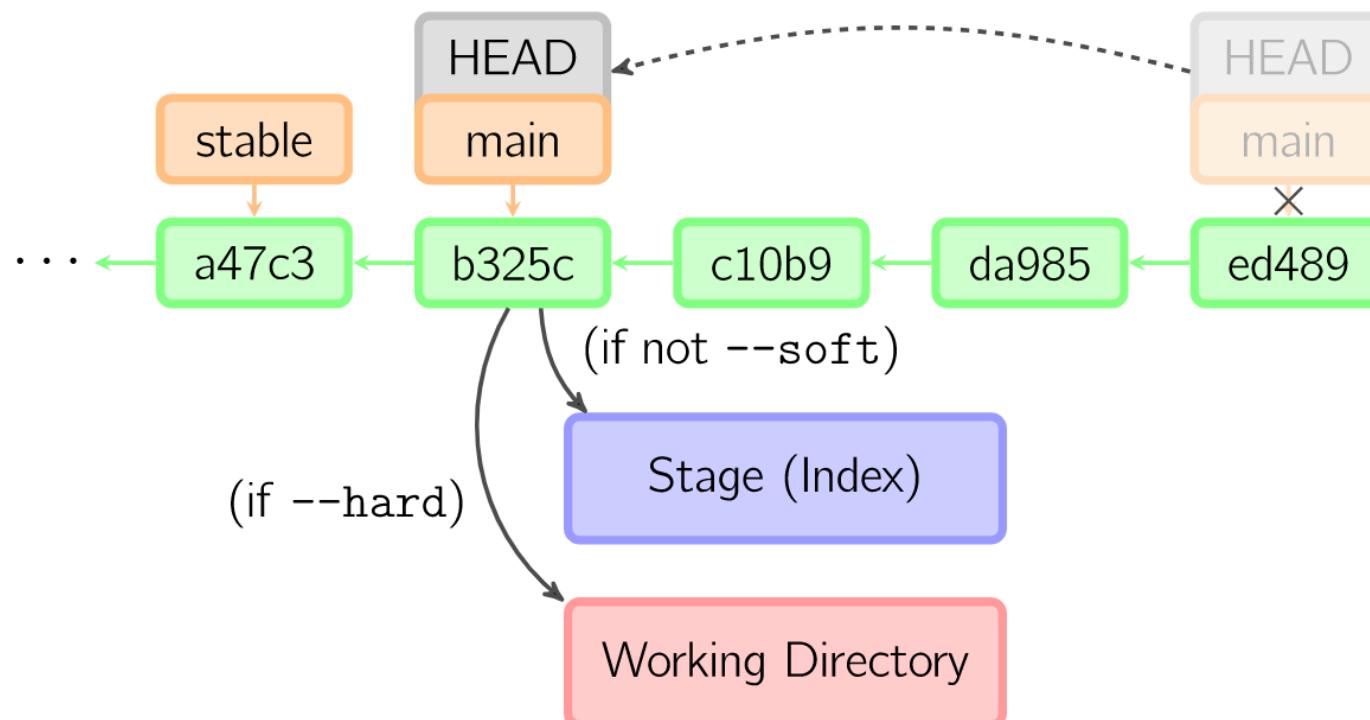


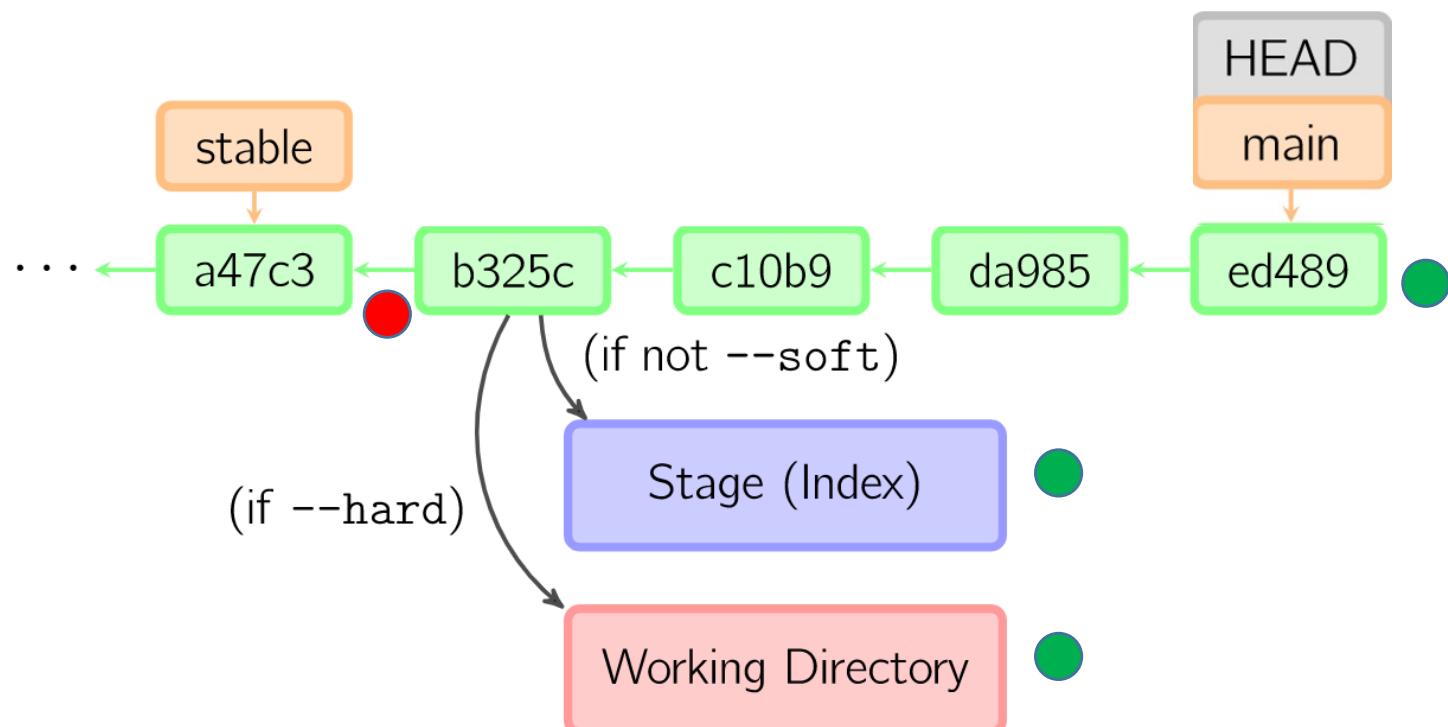
`git checkout [tag | sha1 | branche]`



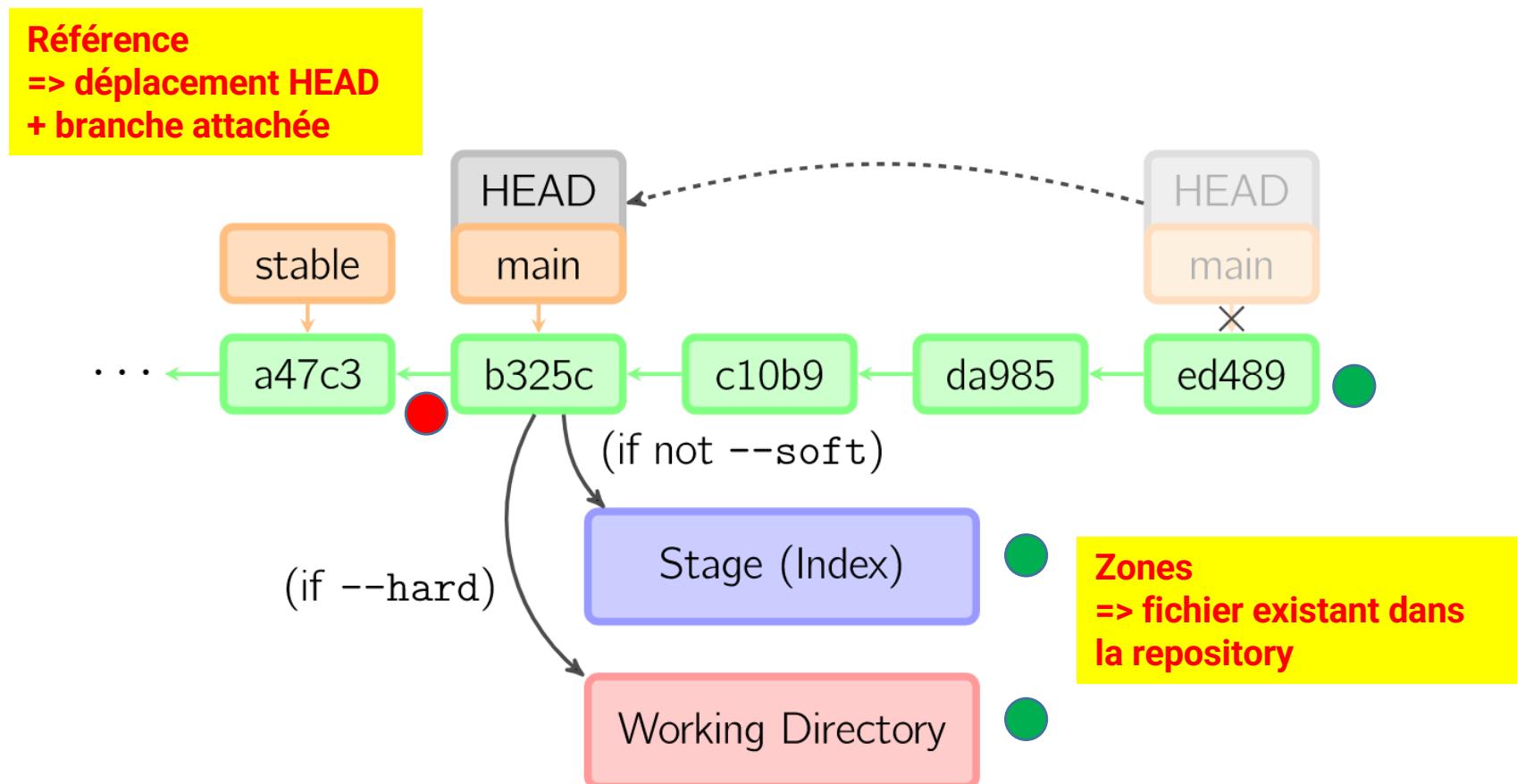
- Commit parent : HEAD^, A34D34F^, branche·, HEAD~1
- Commit grand-parent : HEAD^^ ou HEAD~2
- Commit grand-grand parent : HEAD^^^ ou HEAD~3  
=> HEAD~n (nombre de génération)

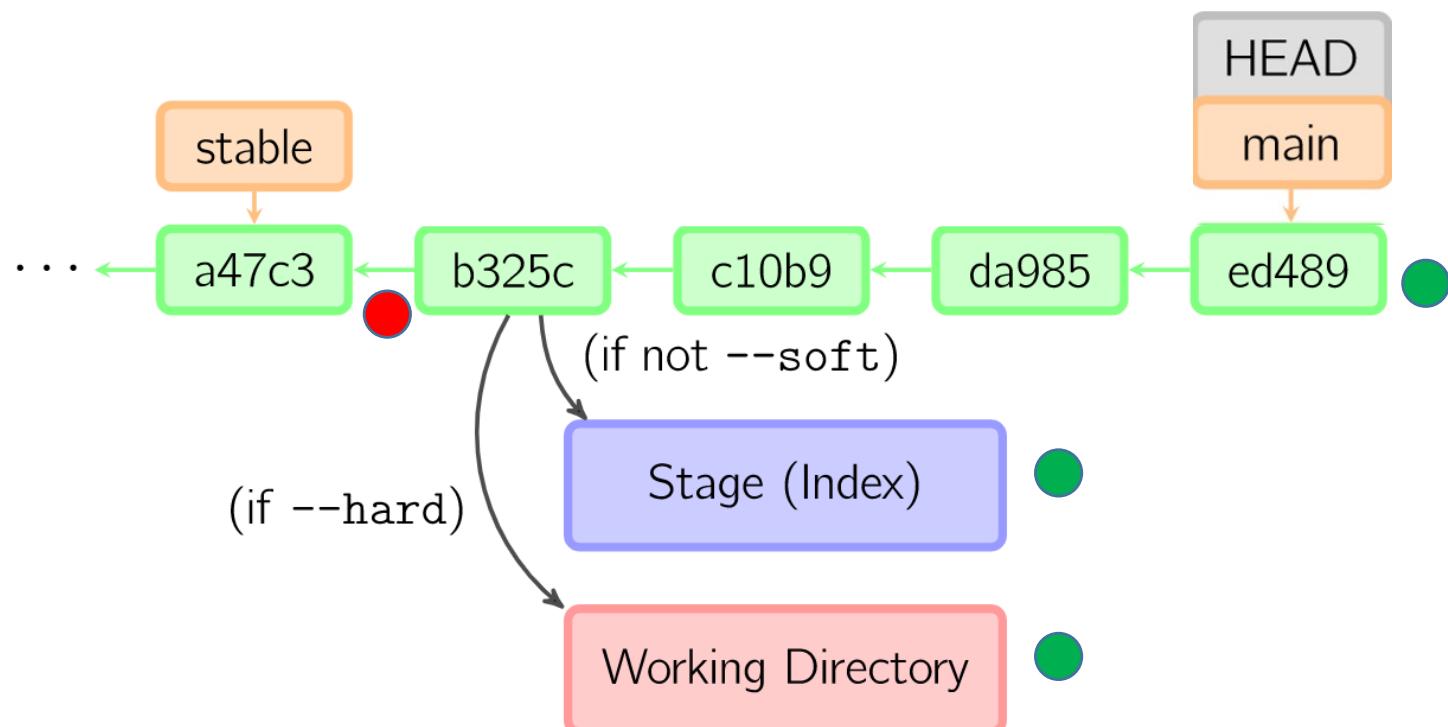
`git reset [--soft|--mixed|--hard] [tag|sha1]`





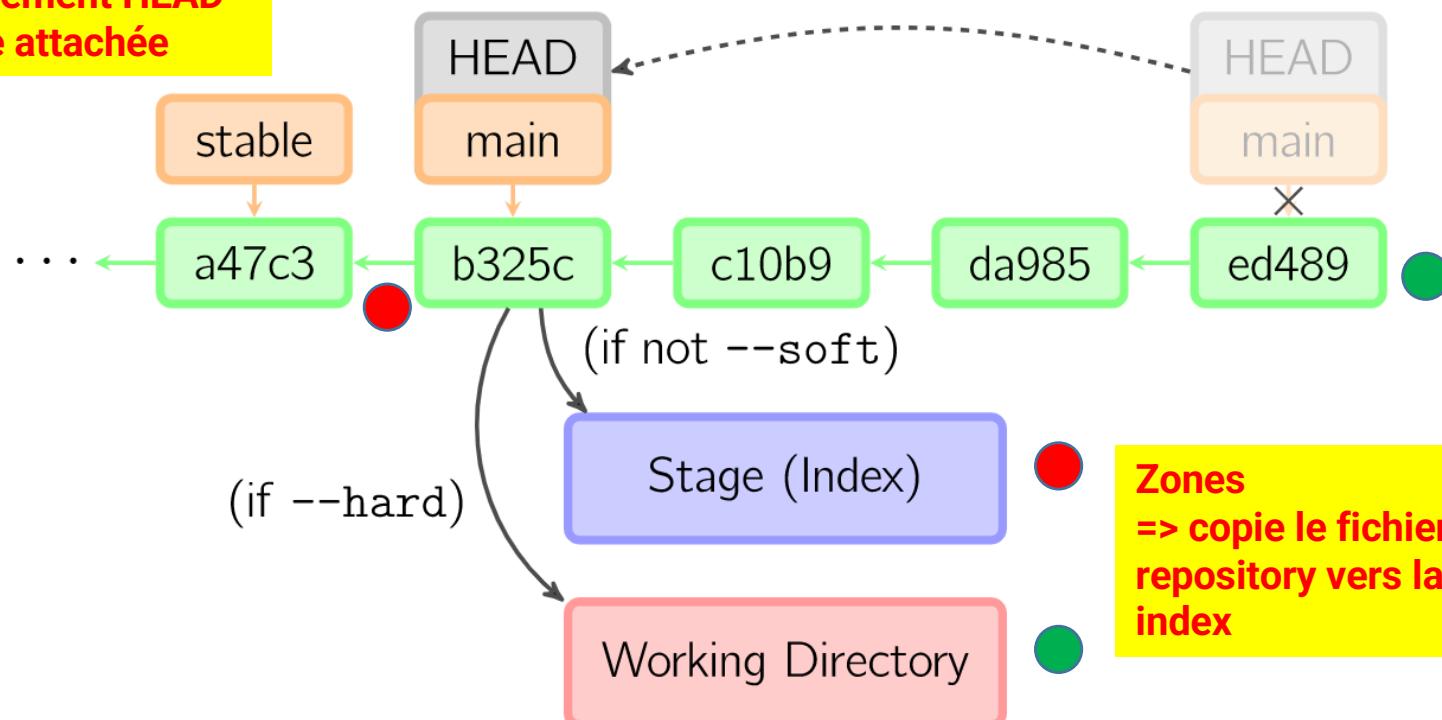
```
git reset --soft HEAD~3
```



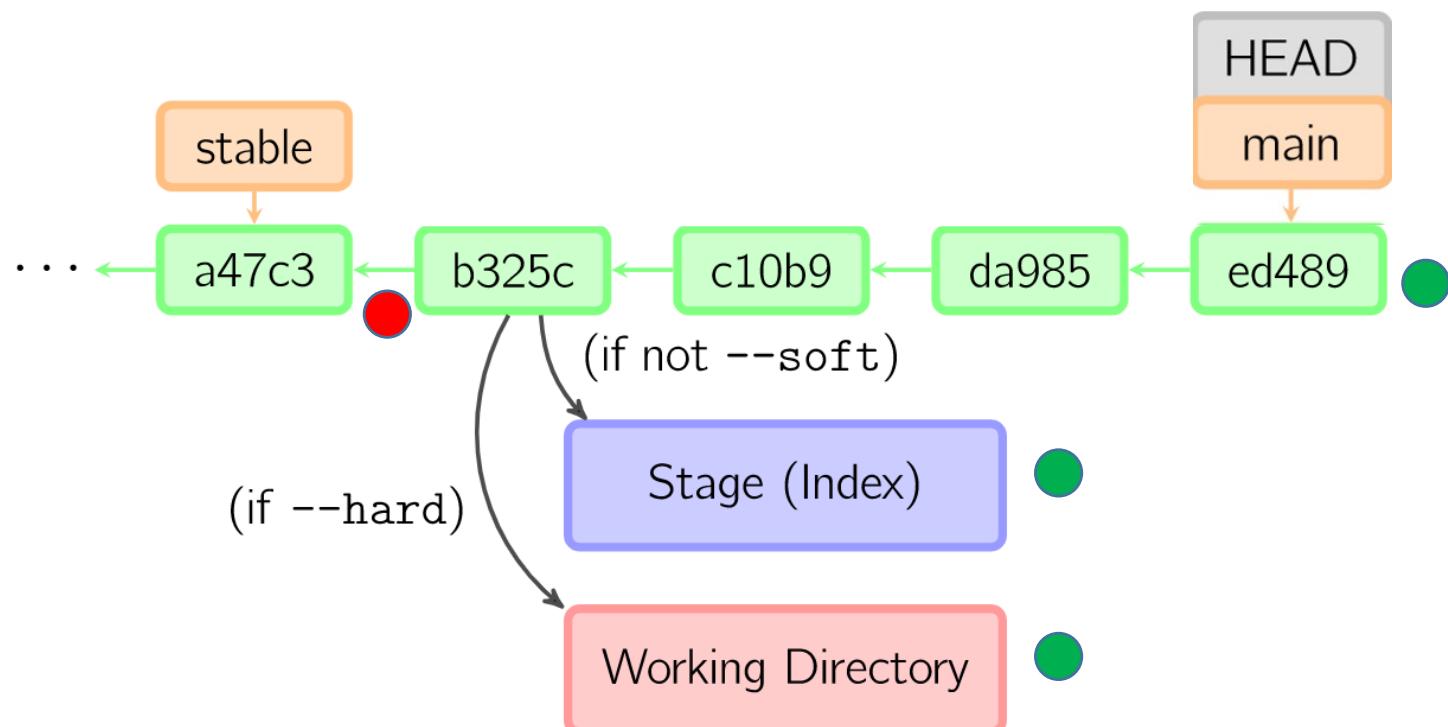


```
git reset --mixed HEAD~3
ou
git reset HEAD~3
```

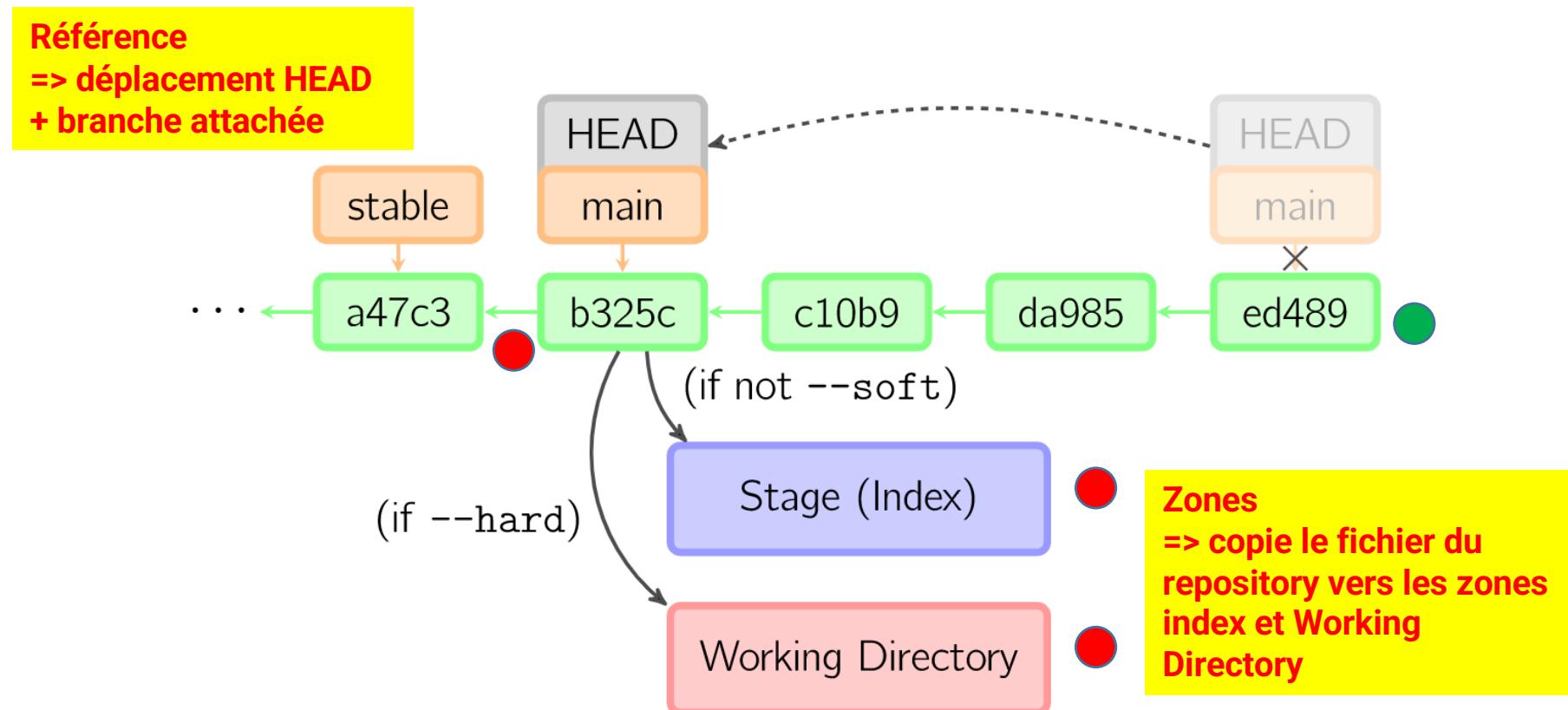
Référence  
=> déplacement HEAD  
+ branche attachée



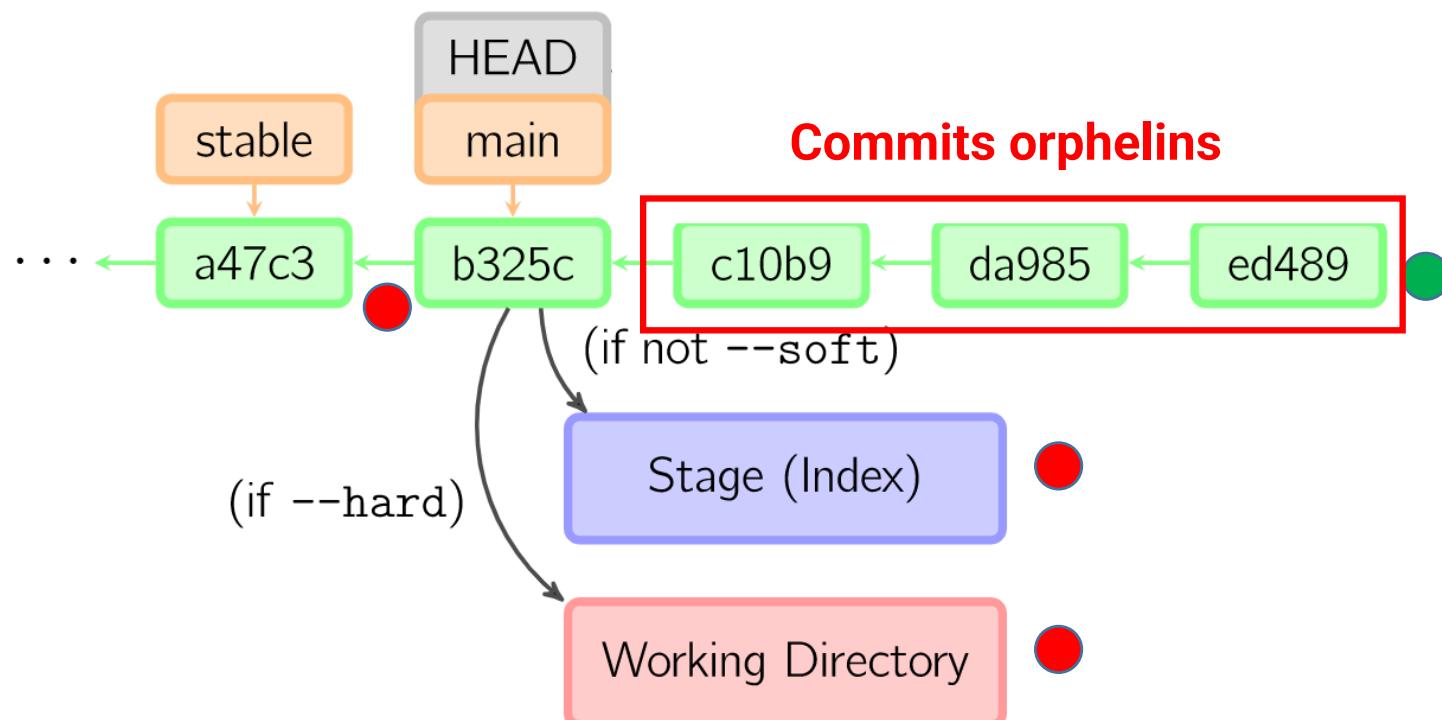
Zones  
=> copie le fichier du repository vers la zone index



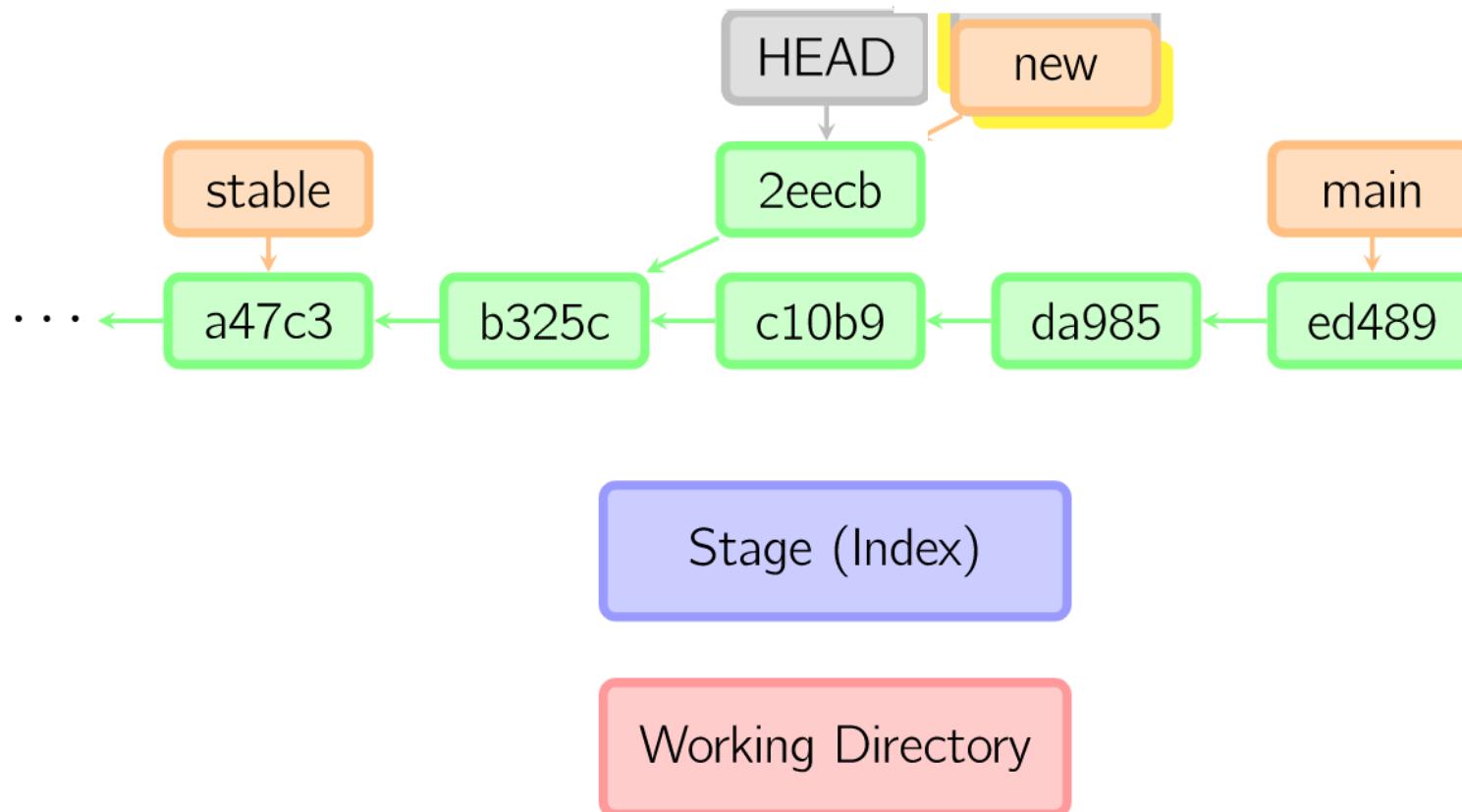
git reset --hard HEAD~3

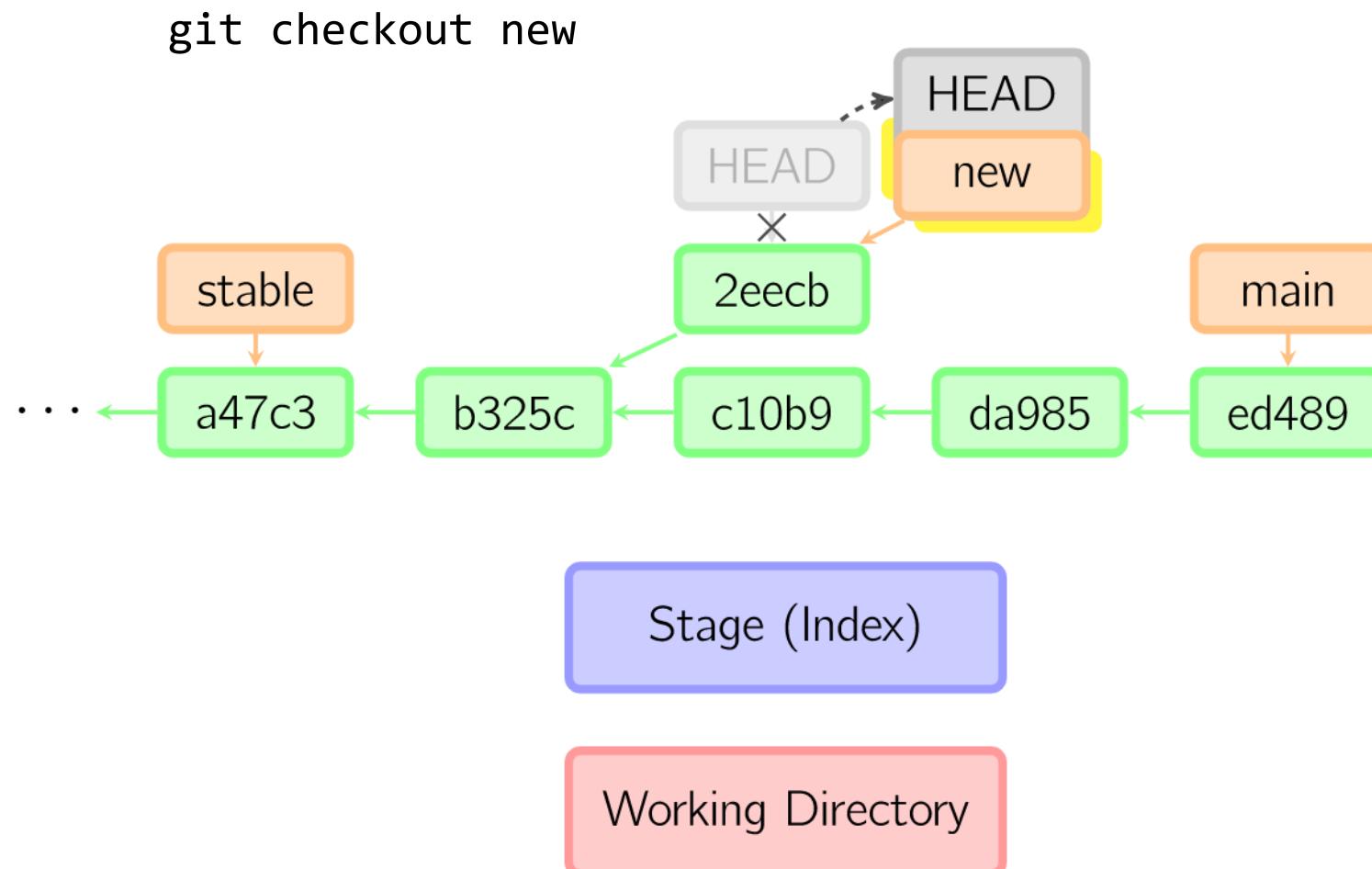


git reset --hard HEAD~3



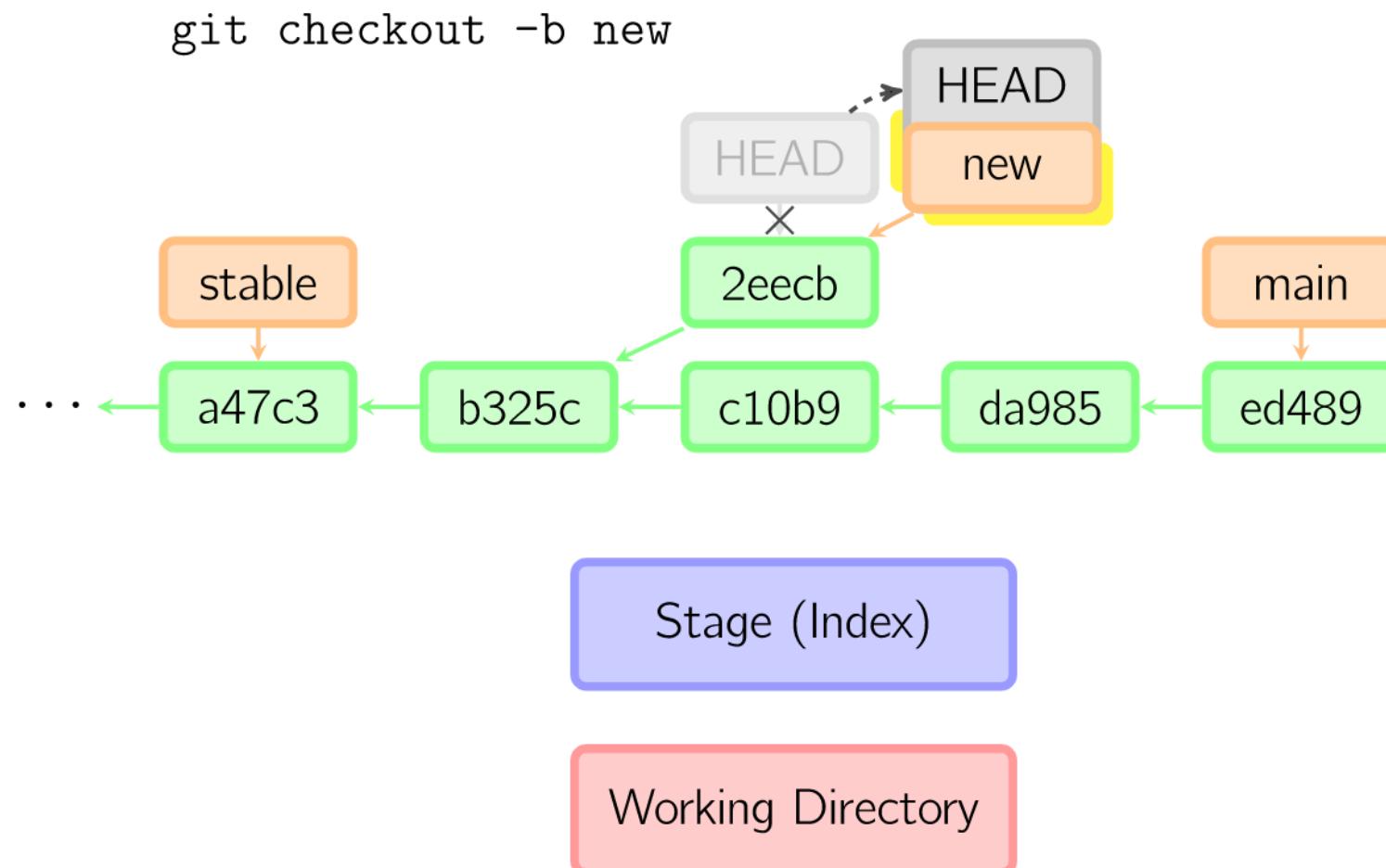
```
git branch new
```





- Lister toutes les branches locales  
`git branch`
- Master ou main : la branche par défaut
  - \* : branche courante
- Créer une branche "nomBranche"  
`git branch nomBranche`
- Basculer sur la branche "nomBranche"  
`git checkout nomBranche`
- Créer et basculer en une seule commande  
`git checkout -b nomBranche`  
= `git branch nomBranche + git checkout nomBranche`

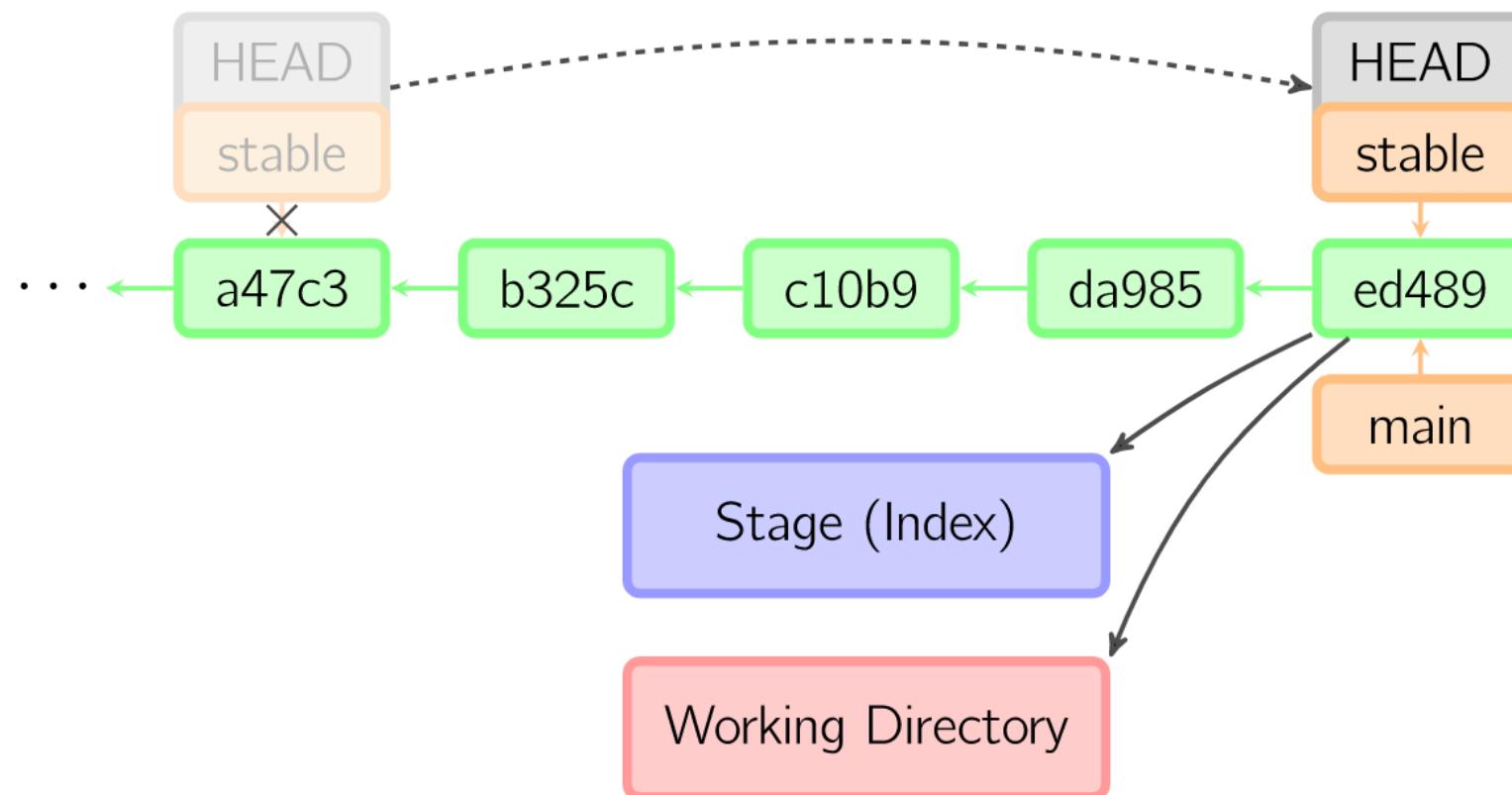
- Renommer une branche  
`git branch -m ancienne nouvelle`
- Supprimer une branche  
`git branch -d nomBranche`
- Supprimer sans vérification  
`git branch -D nomBranche`



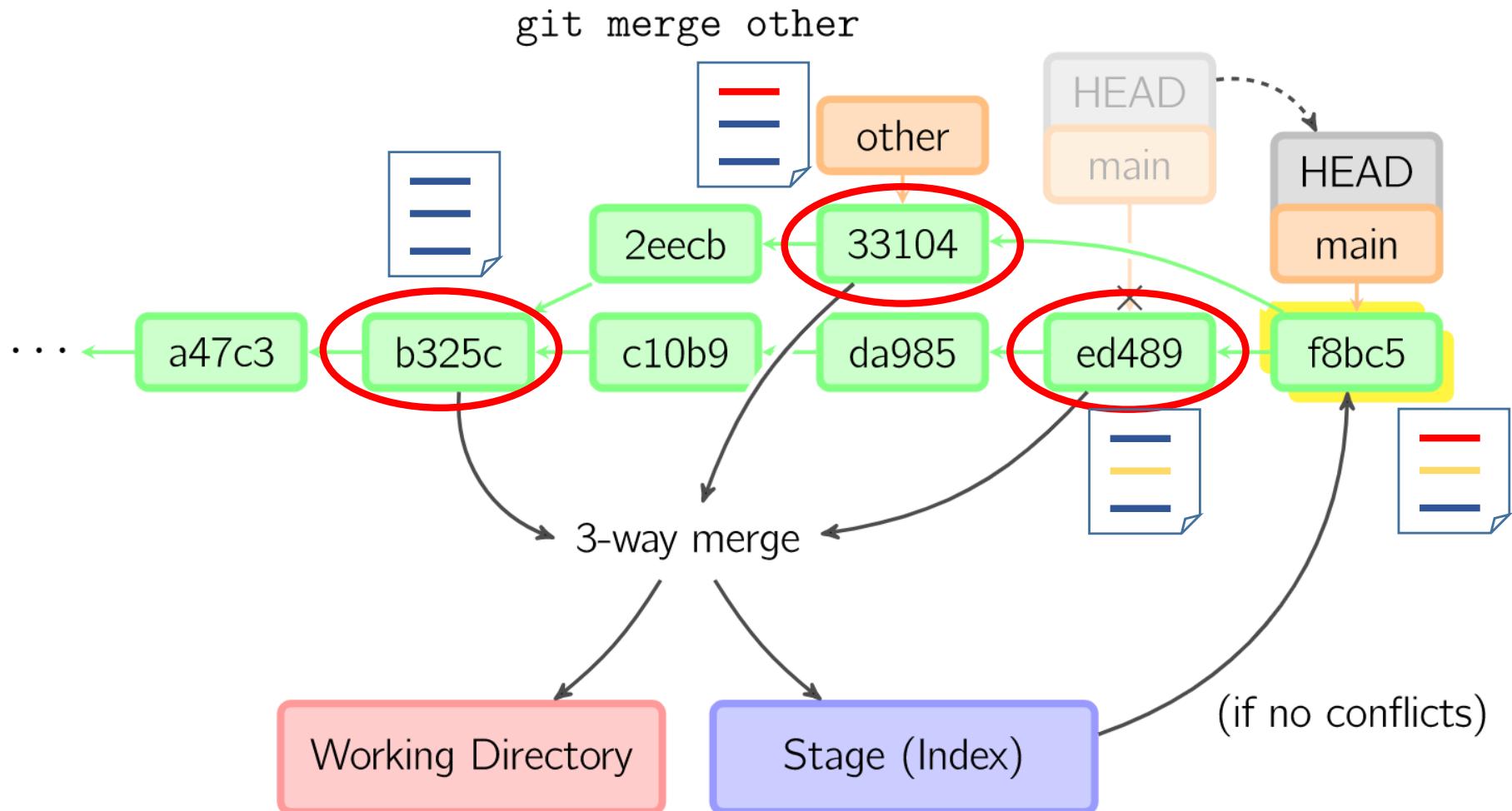
# Démo

## Commandes – fusion avance rapide–(Fast forward merge)

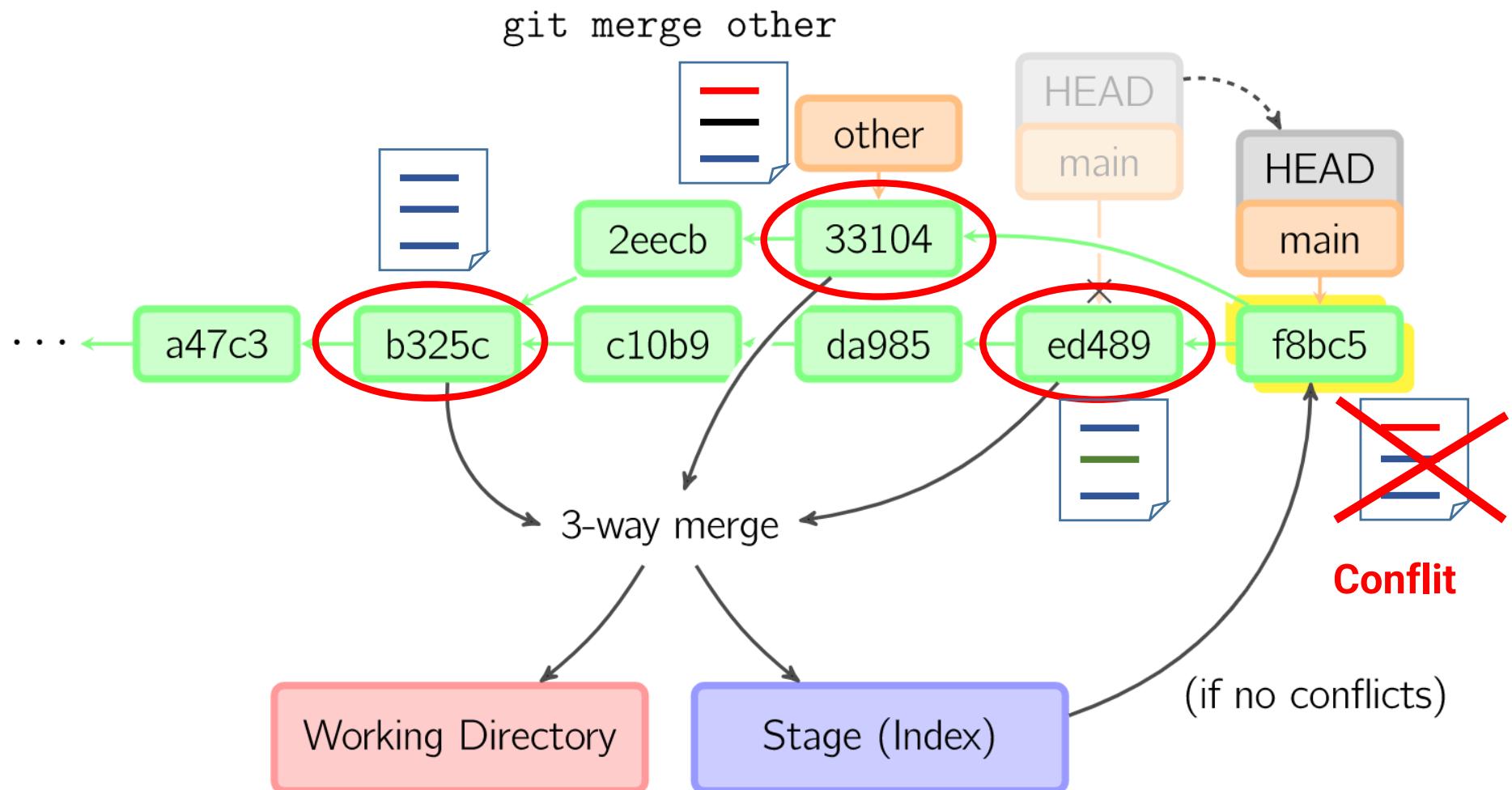
git merge main



## Commandes – fusion à trois voies – (three-way merge)



## Commandes – fusion à trois voies – (three-way merge)



Les trois possibilités de résolution d'un conflit :

- Annuler la fusion (abort)  
`git merge --abort`
- Résoudre le conflit manuellement :  
Editer le fichier confletuel /corriger le conflit / ajouter à l'index / commiter le fichier
- Utiliser un outil de fusion (p4merge, Meld, Kdiff, ...)

TRI

## Stash

Git stash

Git stash save nom

Git stash list

Git stash apply

Git stash pop

# Collaboration

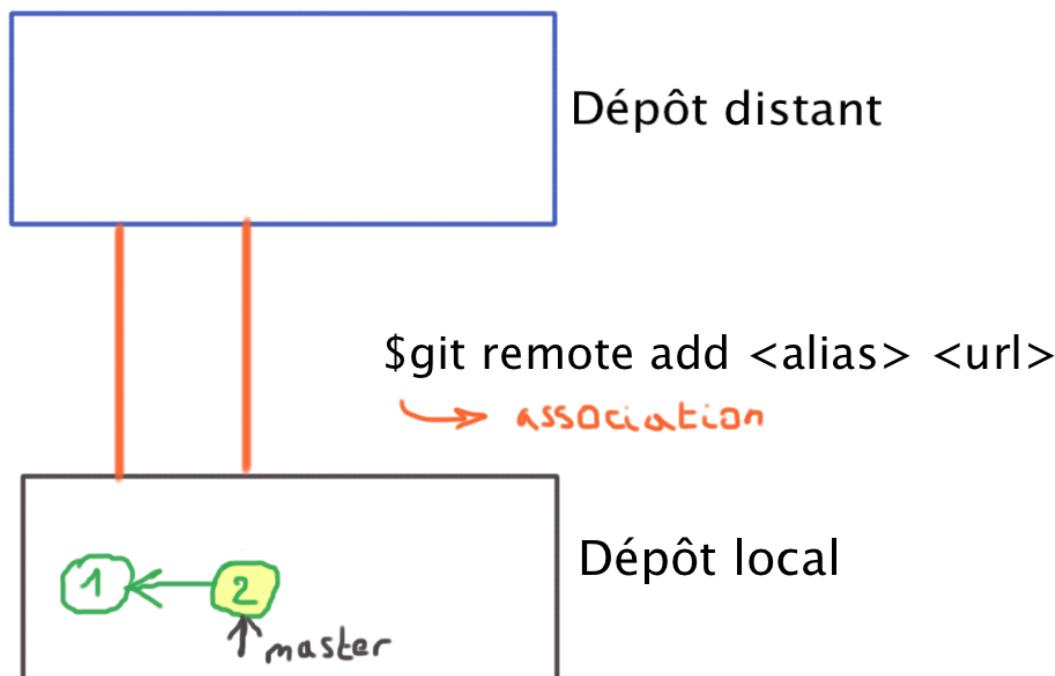
# Réseau local

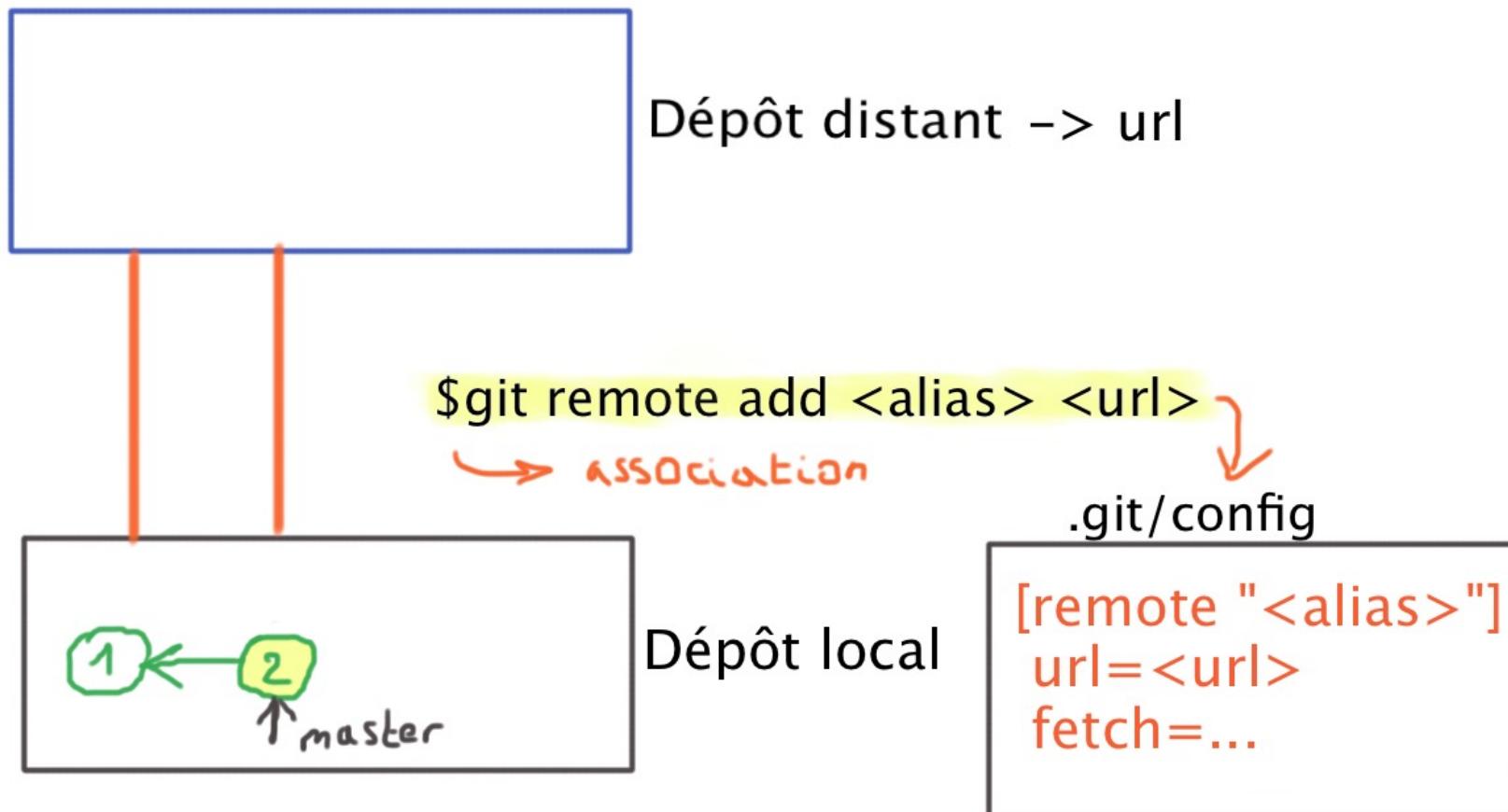
- Dépôt de type « bare »
- GitLab

Github (plateforme sur le web):

- **URL** : <https://github.com/>
- **Dépôts publics** : gratuits
- **Dépôts privés** : payants
- **Très populaires** => projets Open Source
- **Exemples** : jQuery, Symfony, Ruby on Rails. . .

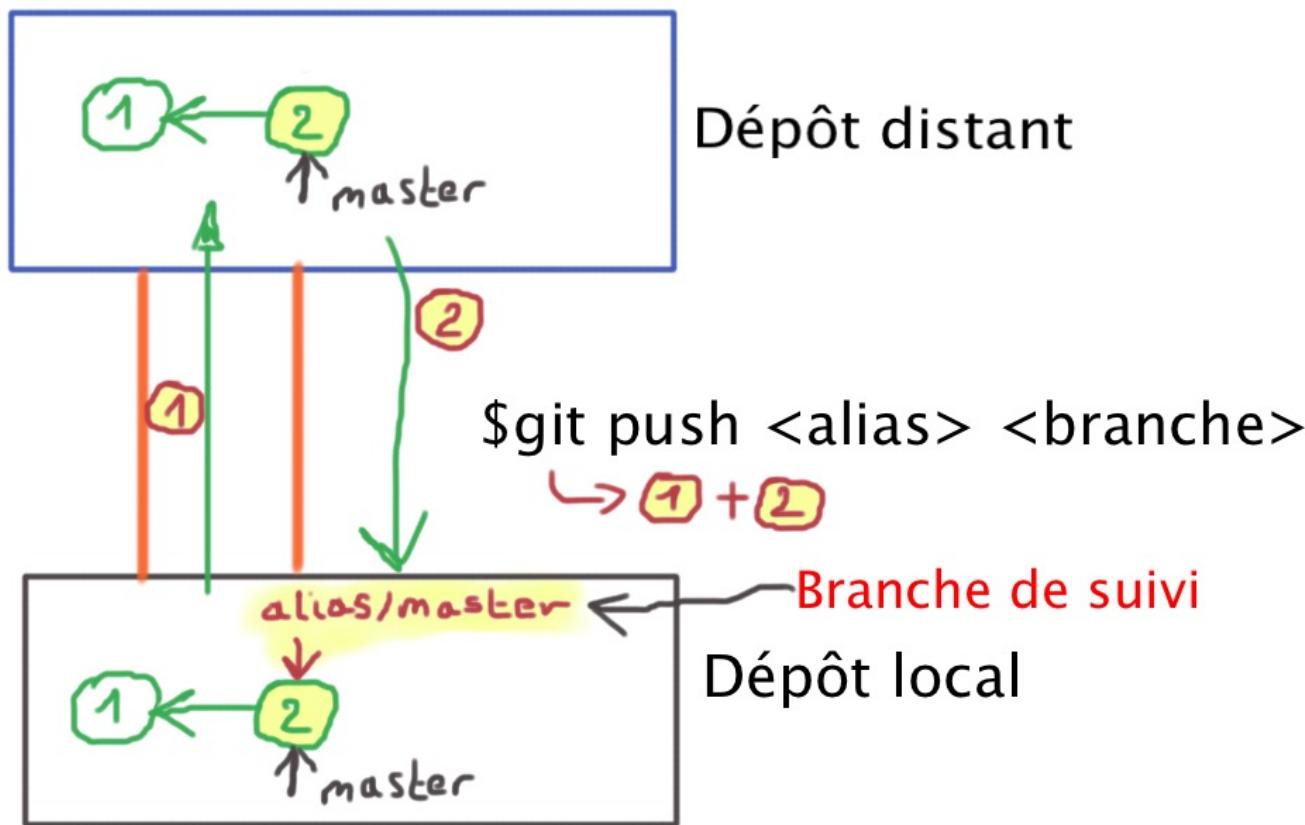
Crée une nouvelle connexion à un dépôt distant. Quand on a ajouté un dépôt distant, on peut utiliser <alias> comme raccourci pour <url> dans les autres commandes Git.





Pousse la branche spécifiée vers <distant>, avec tous les commits et objets internes nécessaires. Crée une branche locale dans le dépôt de destination.

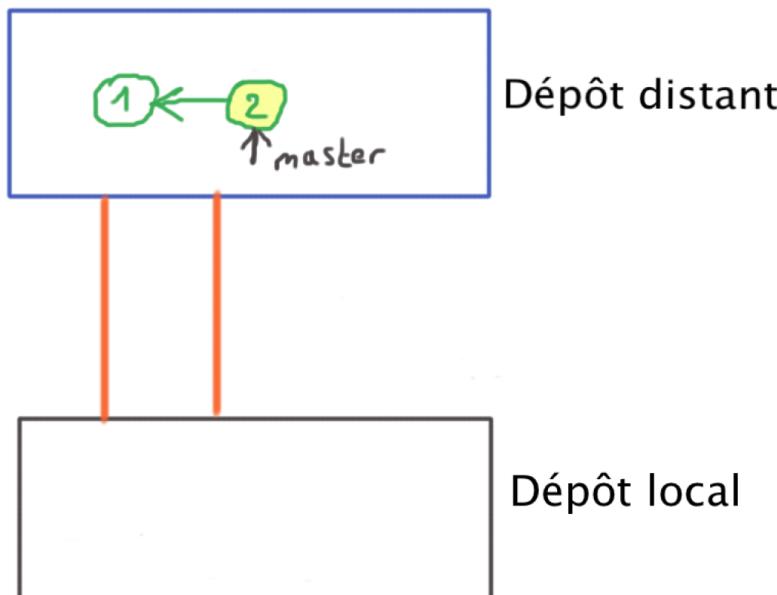
```
$ git push <distant> <branche>
```

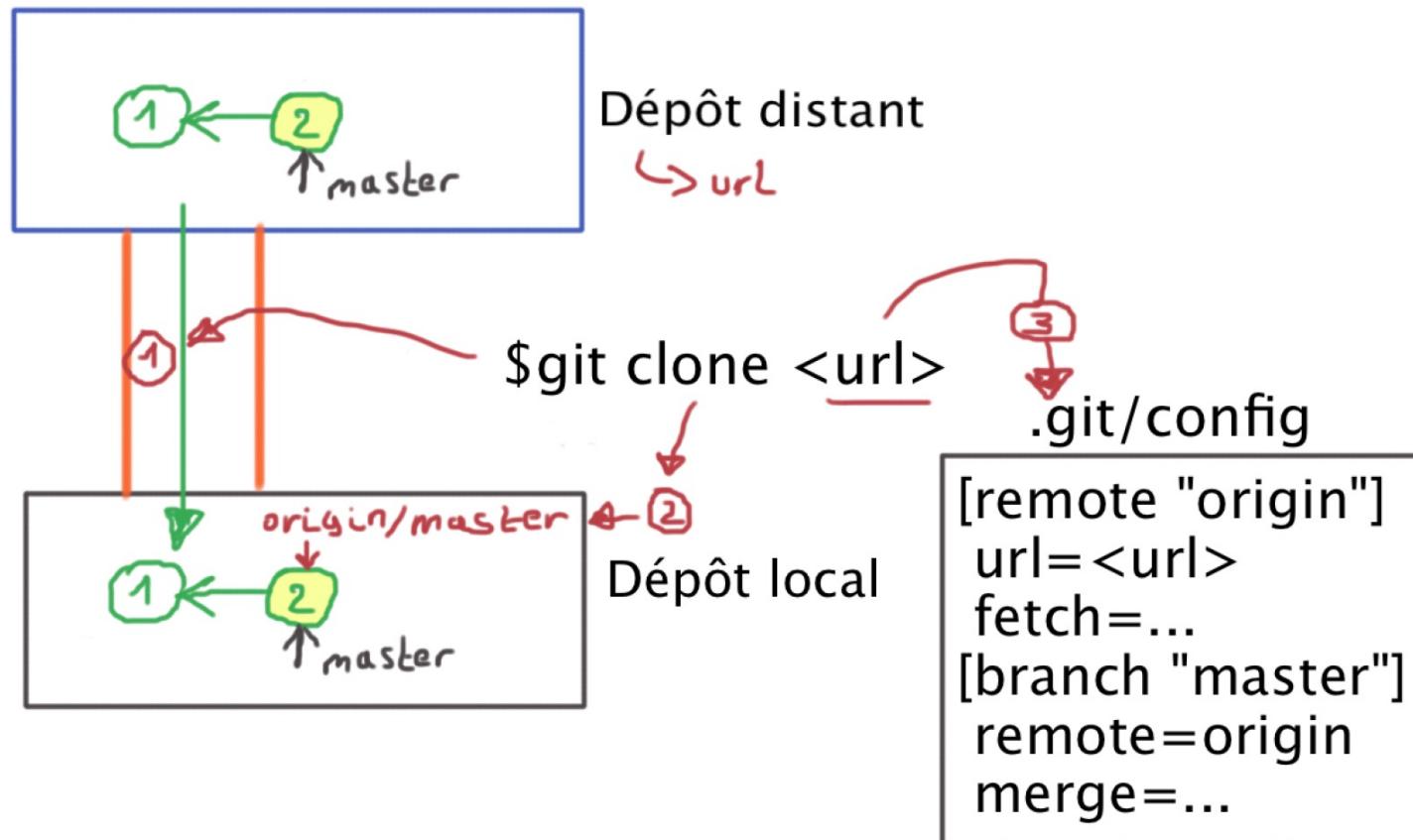


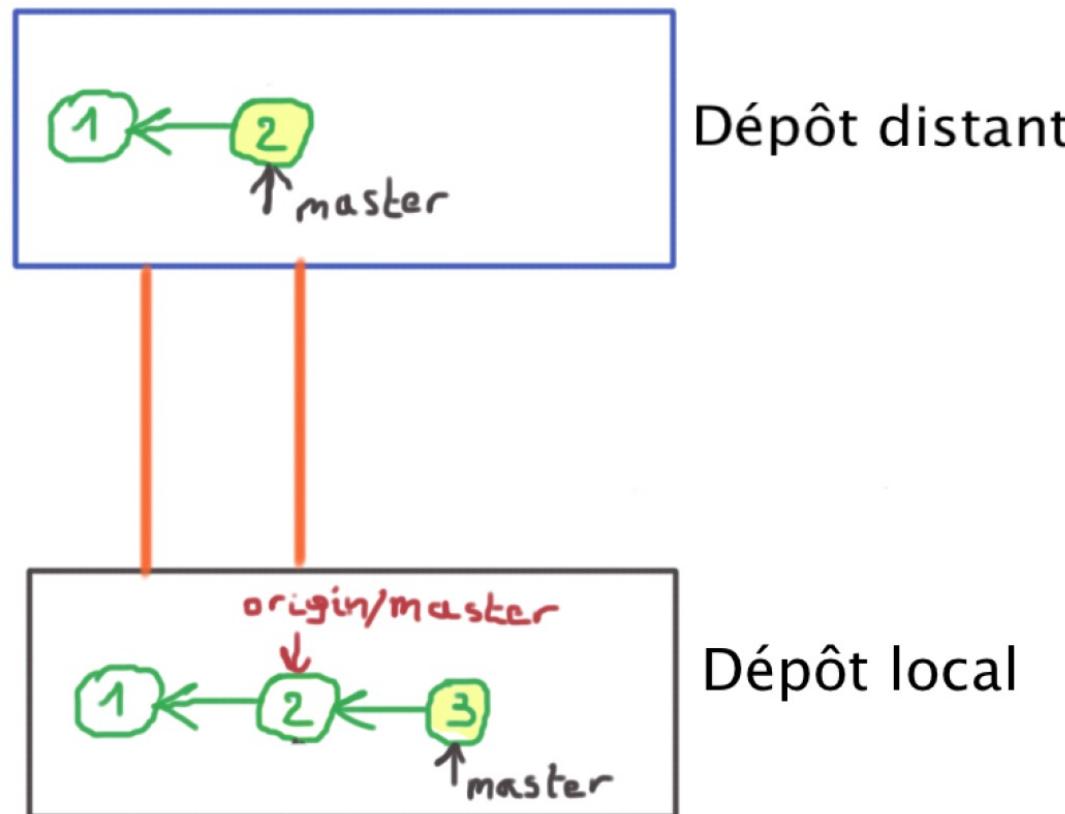
- Les branches de suivi sont des **branches locales** qui sont en relation directe avec une branche distante.
- Cette branche reflète **l'état de la branche distante** correspondante.
- Les branches distantes **ne peuvent pas être déplacées par l'utilisateur**, elles sont déplacées lors des synchronisations.

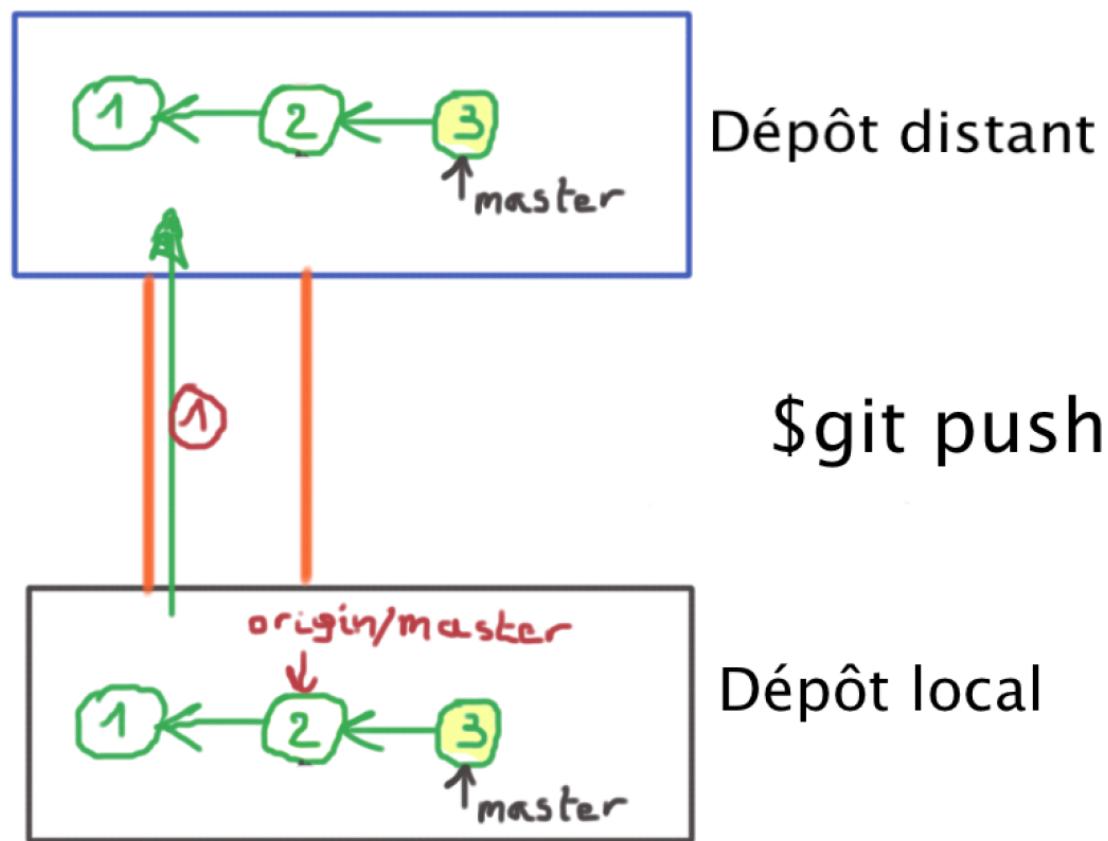
```
$ git clone <dépôt> <rédertoire>
```

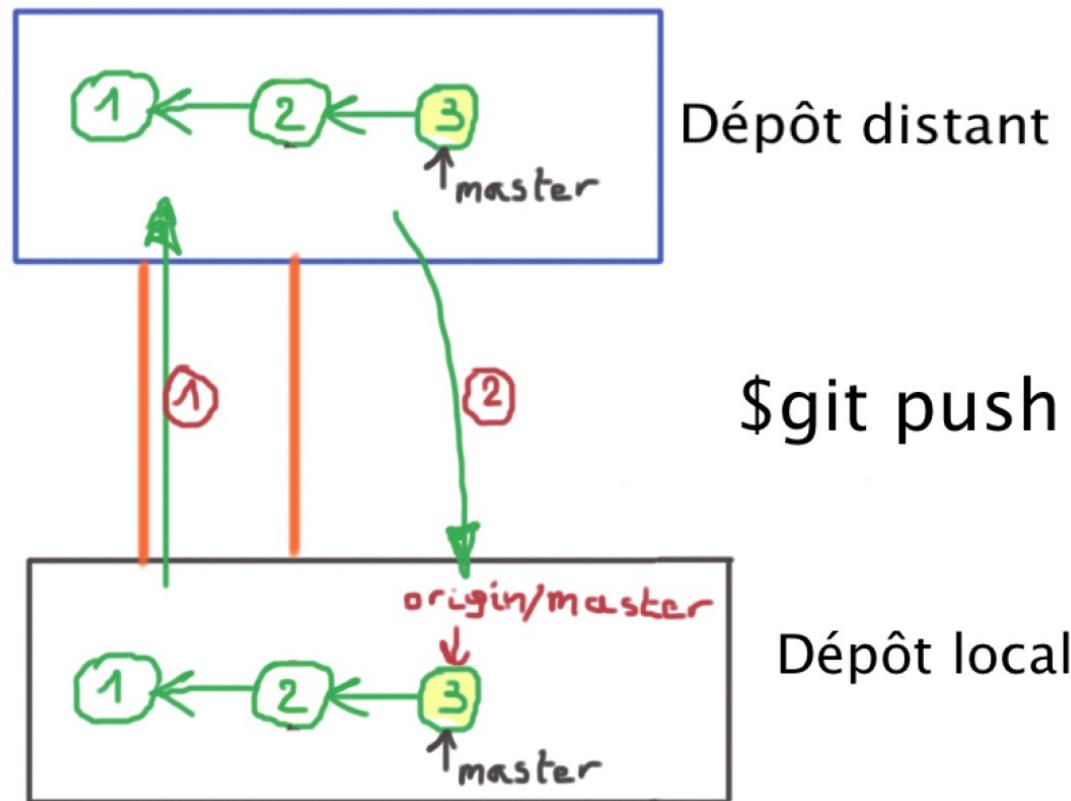
Clone le dépôt situé à l'adresse <dépôt> dans un répertoire intitulé <rédertoire> sur la machine locale.







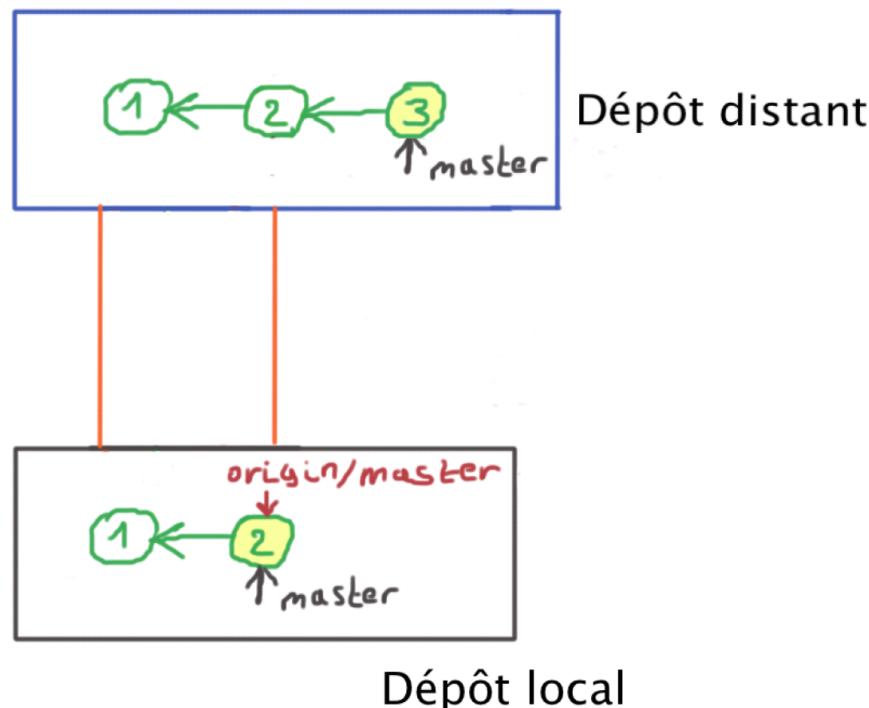




```
$ git fetch <alias> <branche>
```

Récupère la branche du dépôt <alias>. Télécharge aussi tous les commits et fichiers nécessaires depuis le dépôt. =>  
Synchronise la branche de suivi avec la branche distante ou mise à jour de la branche de suivi

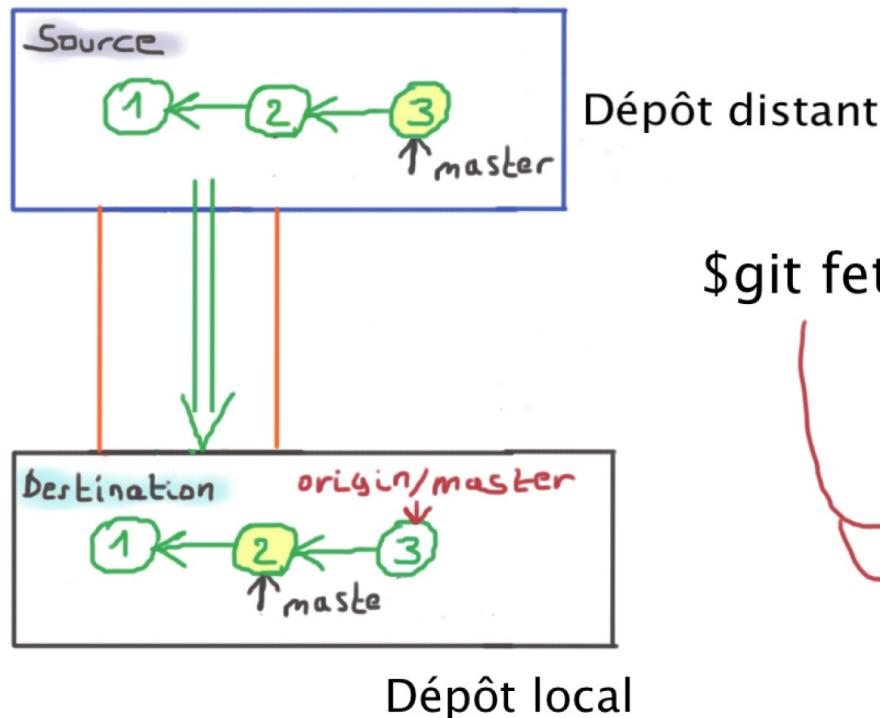
## Gestion des dépôts distants – Synchronisation avec le dépôt distant



.git/config

```
[remote "origin"]
url=<url>
fetch=+refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote=origin
merge=...
```

## Gestion des dépôts distants – Synchronisation avec le dépôt distant



Référence spécifique = Refspec  
Format -> <source>:<destination>

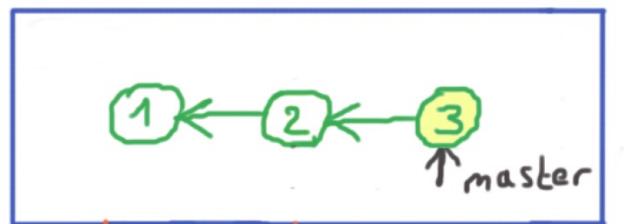
**\$git fetch**

**.git/config**

```
[remote "origin"]
url=<url>
fetch=+refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote=origin
merge=...
```

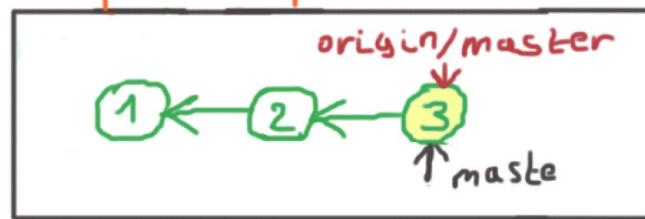
git branch -a : toutes les branches

git branch -r : toutes les branches distantes



Dépôt distant

\$git merge origin/master



Dépôt local

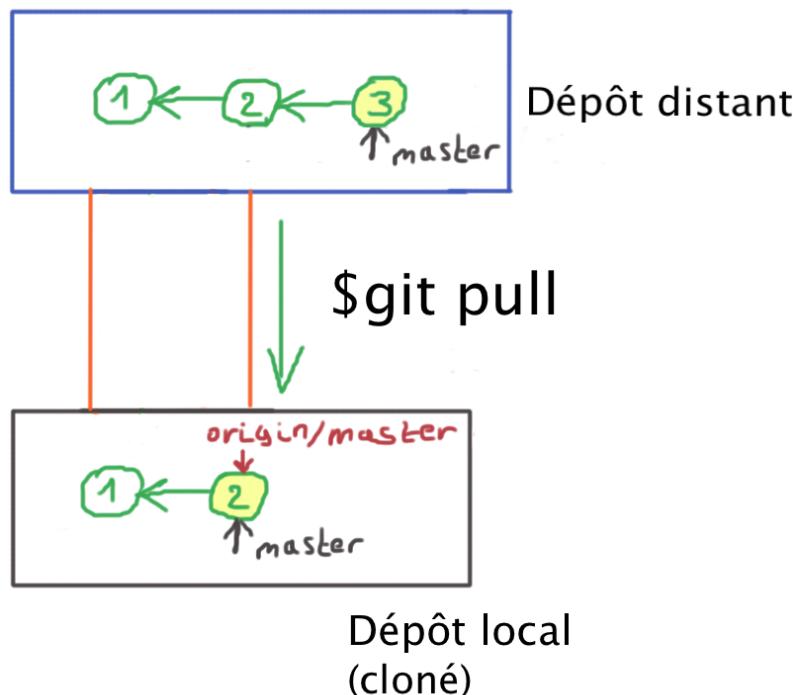
## Gestion des dépôts distants – Synchronisation avec le dépôt distant

- Avant de travailler sur votre dépôt local, effectuer une synchronisation avec le dépôt distant.
- Faire un fetch avant de pousser sur le dépôt distant.
- Faire un fetch souvent afin d'être synchro avec le dépôt.

TRI

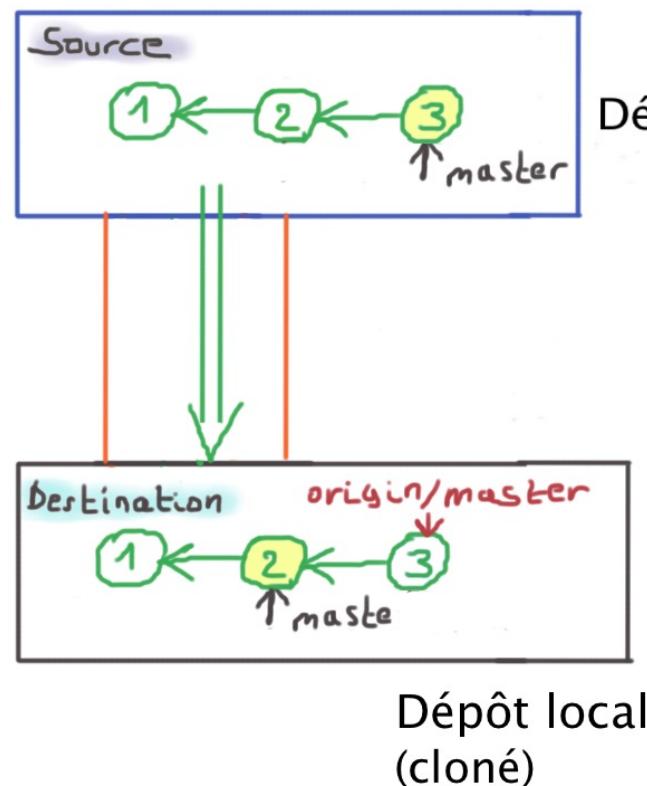
**git pull = (1)git fetch + (2)git merge**

Faire un pull sur un dépôt cloné



```
.git/config
[remote "origin"]
  url=<url>
  fetch=+refs/heads/*:refs/remotes/origin/*
[branch "master"]
  remote=origin
  merge= refs/heads/master
```

## Gestion des dépôts distants – Synchronisation avec le dépôt distant - raccourci



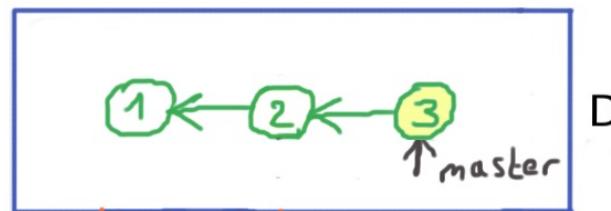
### (1) git fetch

.git/config

```
[remote "origin"]
url=<url>
fetch=+refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote=origin
merge=refs/heads/master
```



## Gestion des dépôts distants – Synchronisation avec le dépôt distant - raccourci



Dépôt distant



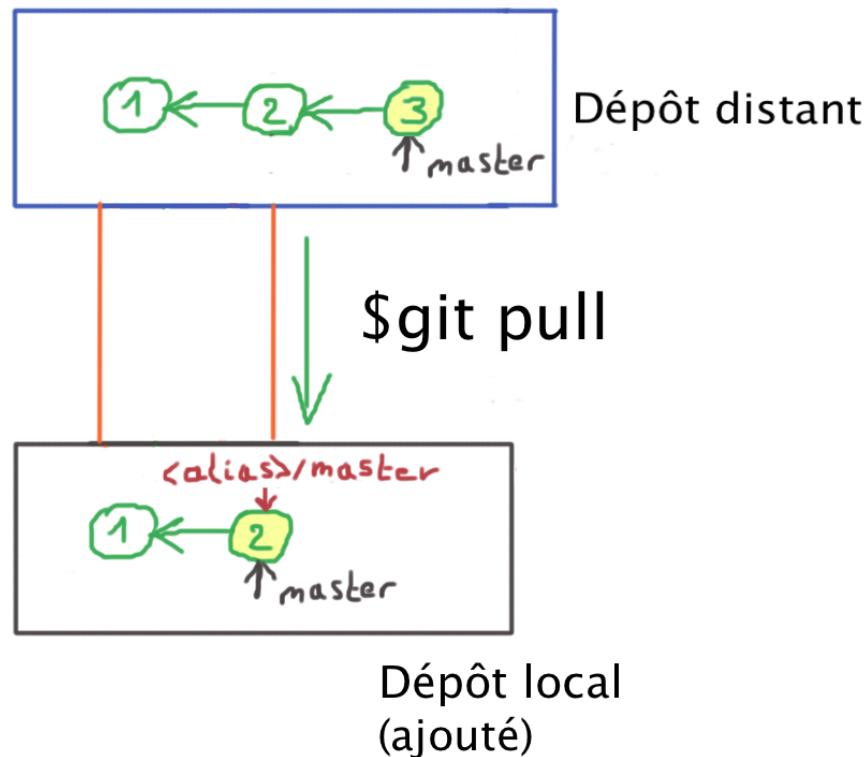
Dépôt local  
(cloné)

(2) git merge

.git/config

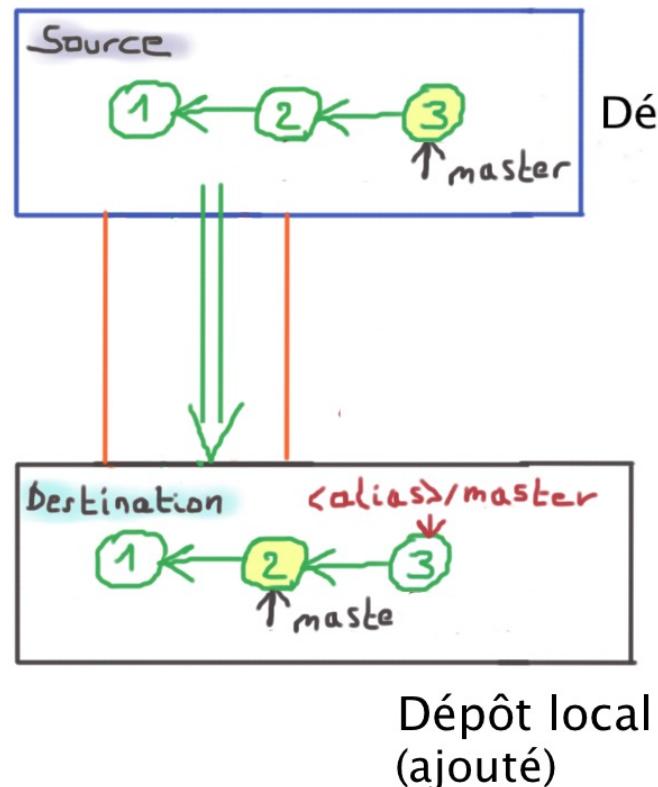
```
[remote "origin"]
url=<url>
fetch=+refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote=origin
merge=refs/heads/master
```

Faire un pull sur un dépôt ajouté



```
[remote "<alias>"]  
url=<url>  
fetch=+refs/heads/*:refs/remotes/<alias>/*
```

## Gestion des dépôts distants – Synchronisation avec le dépôt distant - raccourci



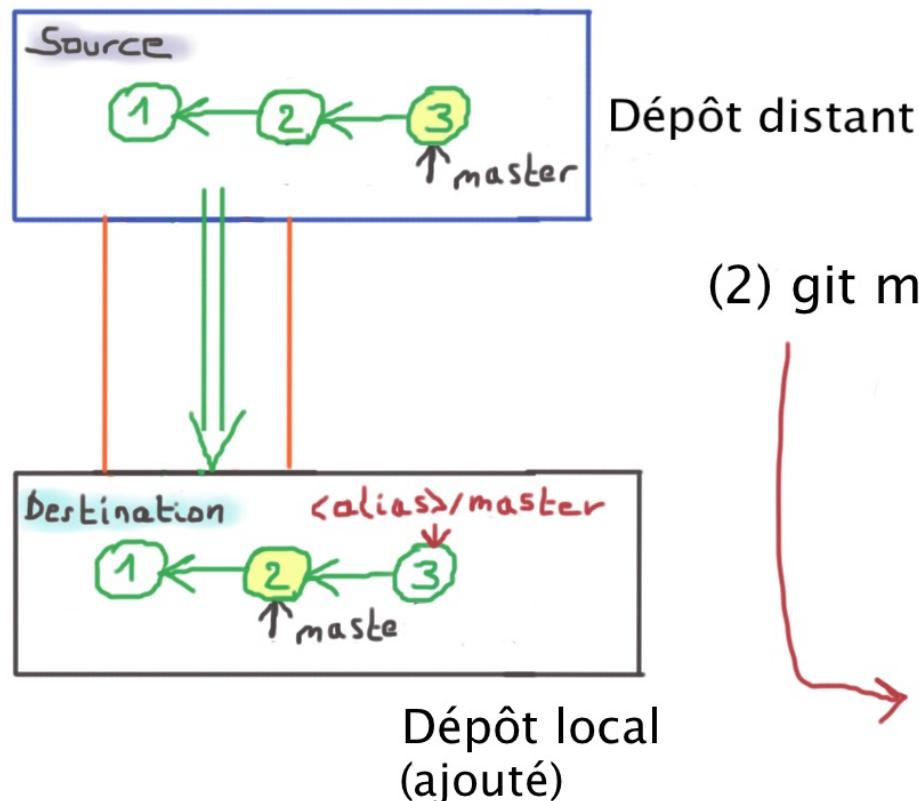
Dépôt distant

(1) git fetch

.git/config

```
[remote "<alias>"]
url=<url>
fetch=+refs/heads/*:refs/remotes/<alias>/*
```

## Gestion des dépôts distants – Synchronisation avec le dépôt distant - raccourci



(2) git merge *⇒ ne fonctionne pas !*

.git/config

```
[remote "<alias>"]
url=<url>
fetch=+refs/heads/*:refs/remotes/<alias>/*
?
```

Ajouter un dépôt

```
$ git remote add <alias> <url>
```

Deux solutions :

- `git push -u <alias> master`
- `git branch --set-upstream master <alias>/master`

- Supprime la connexion au dépôt distant intitulé <alias>.

```
$ git remote rm <alias>
```

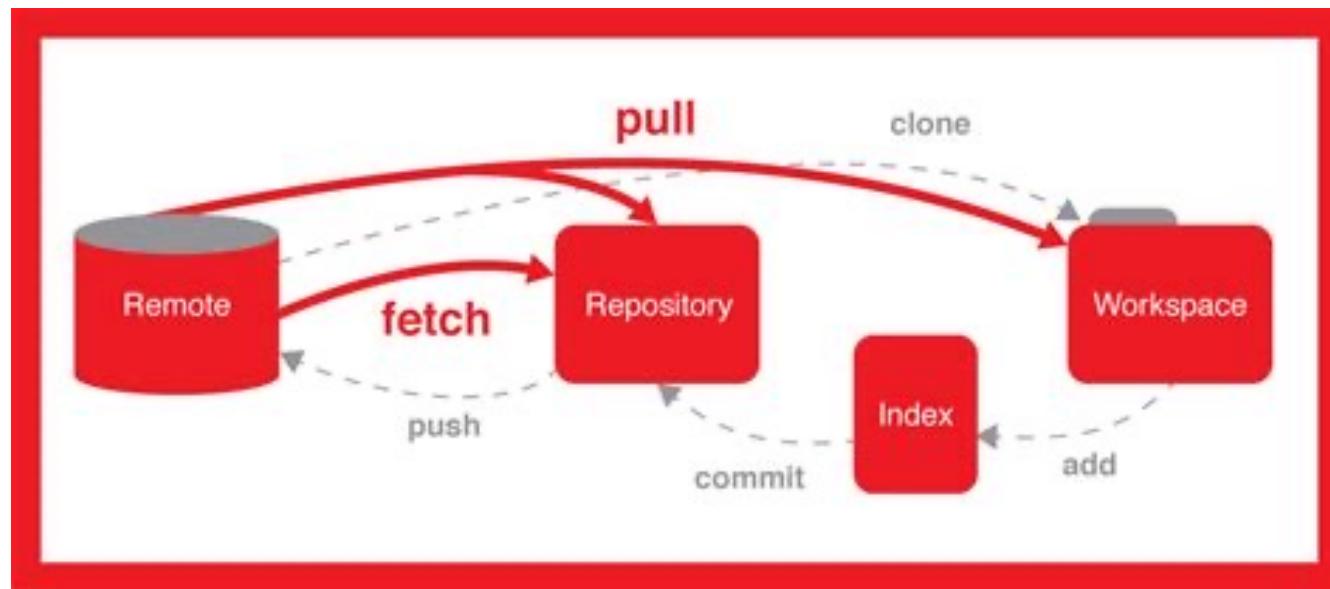
- Mettre à jour les dépôts distants.

```
$ git remote update
```

- Renomme une connexion distante <ancien-nom> en <nouveau-nom>.

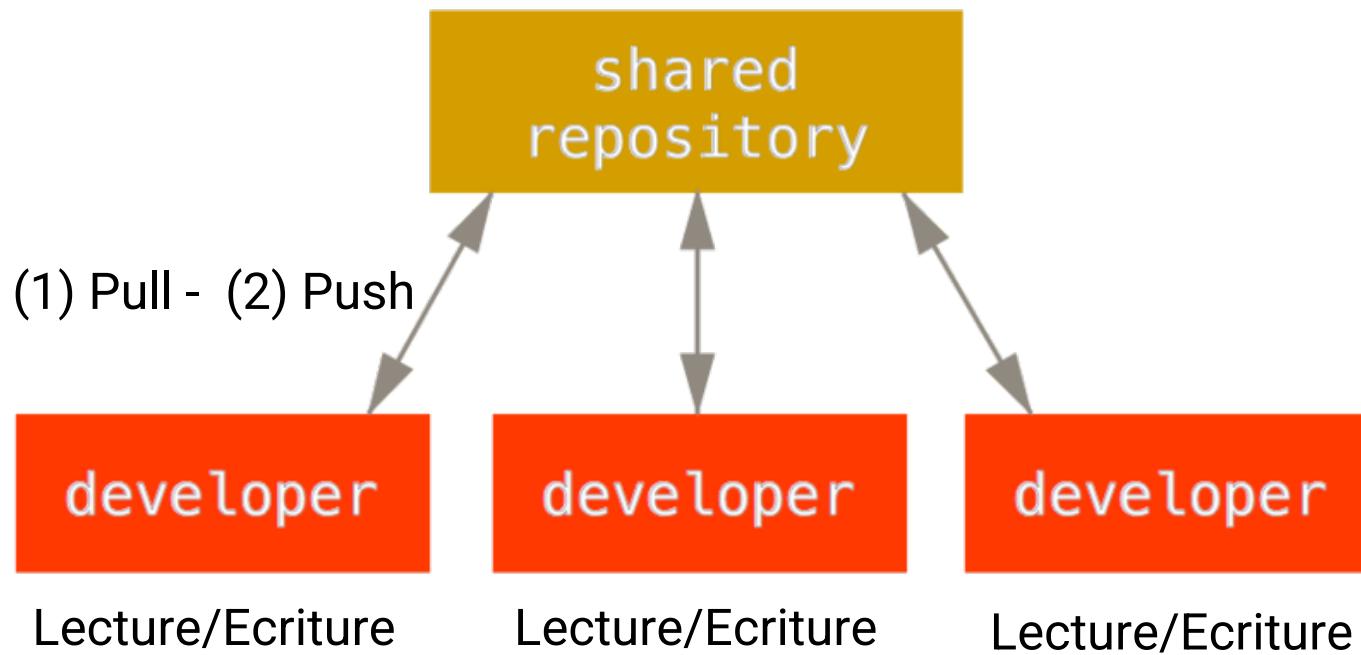
```
$ git remote rename <ancien-nom> <nouveau-nom>
```

## Gestion des dépôts distants – Commandes utiles

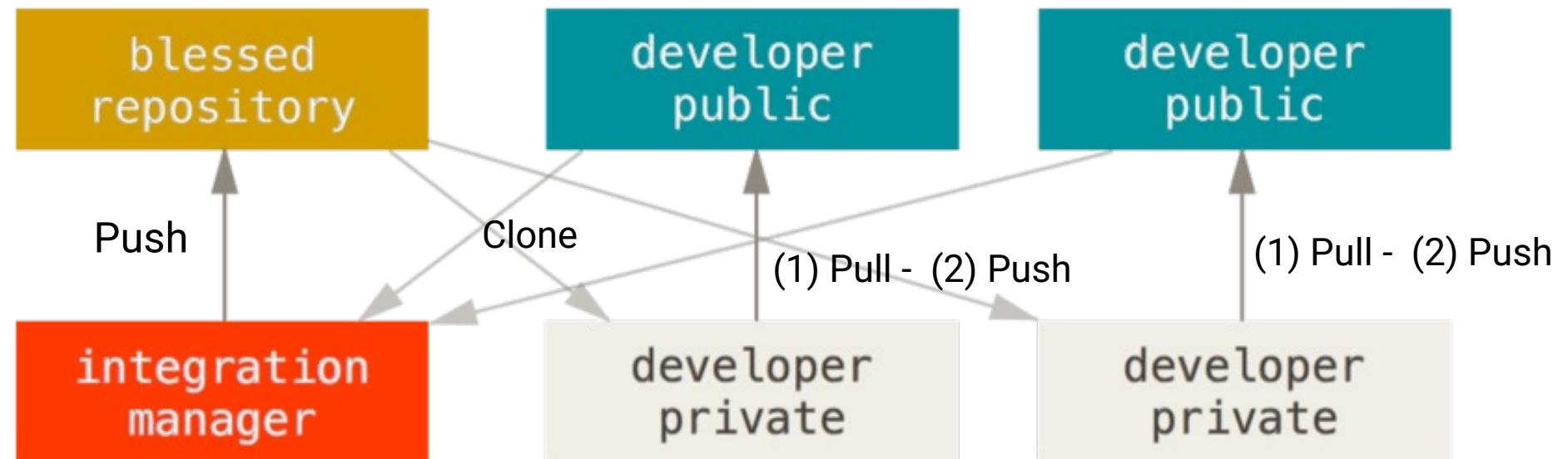


# Organisation du travail en équipe

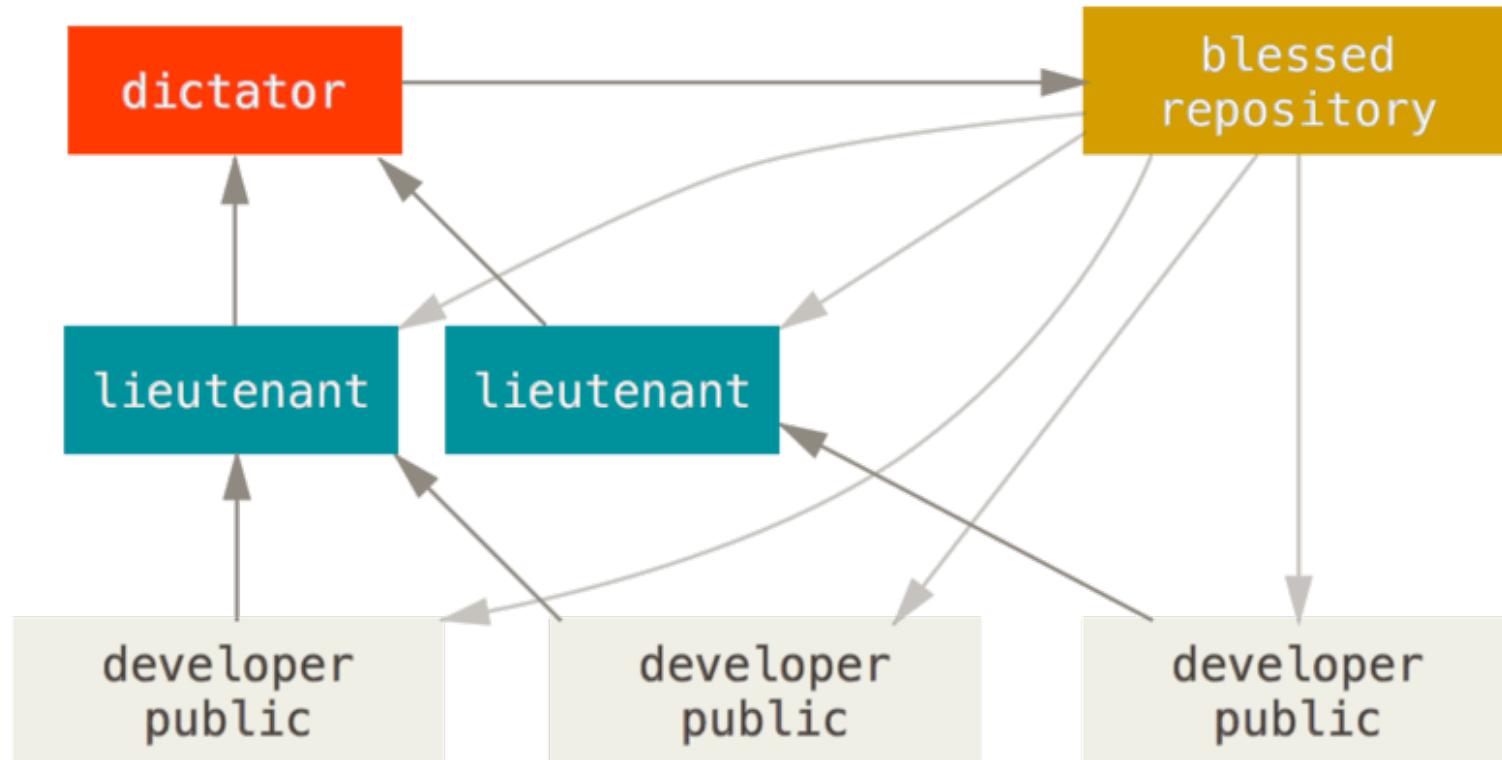
## Gestion des dépôts distants – Gestion centralisée



## Gestion des dépôts distants – Mode du gestionnaire d'intégration

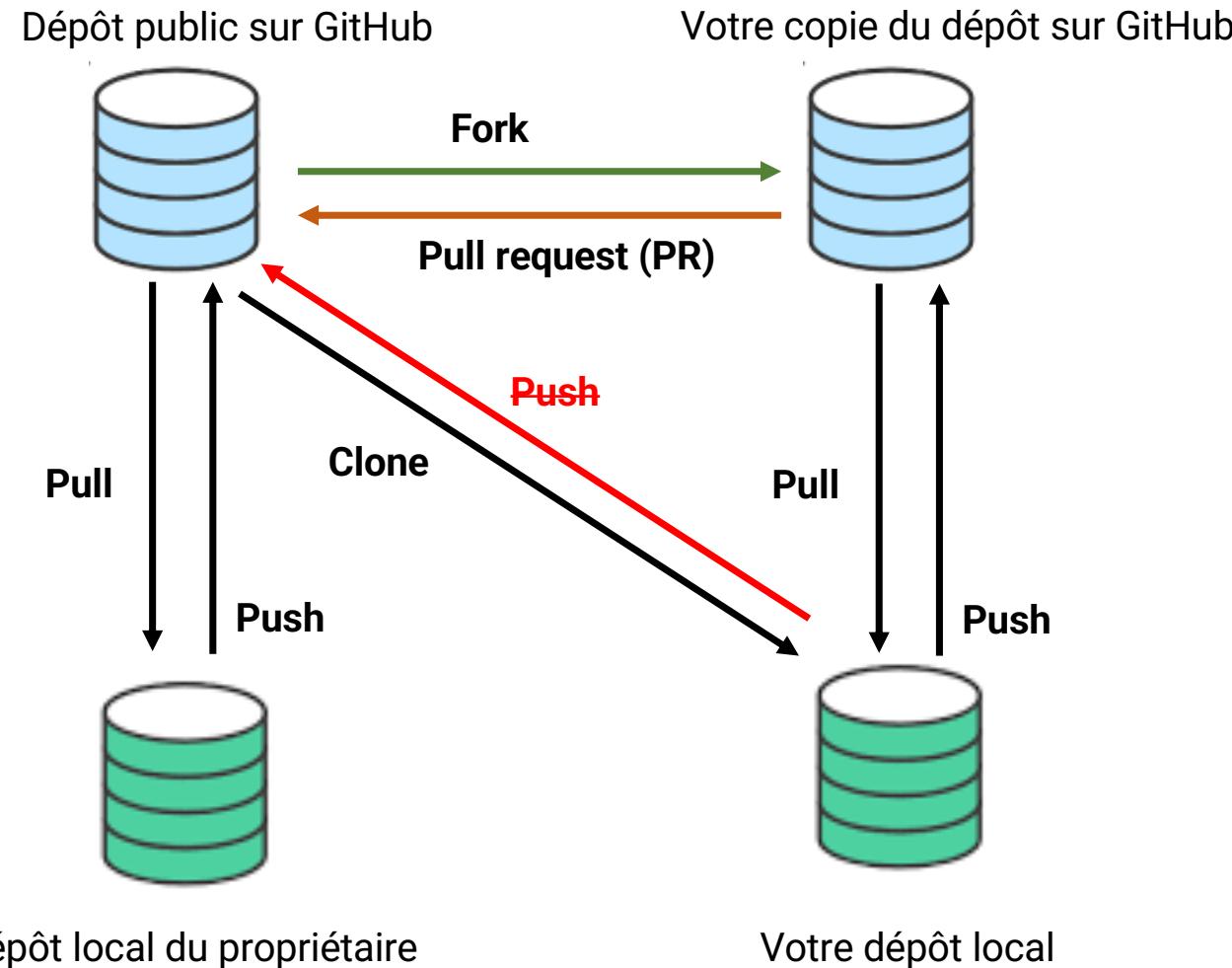


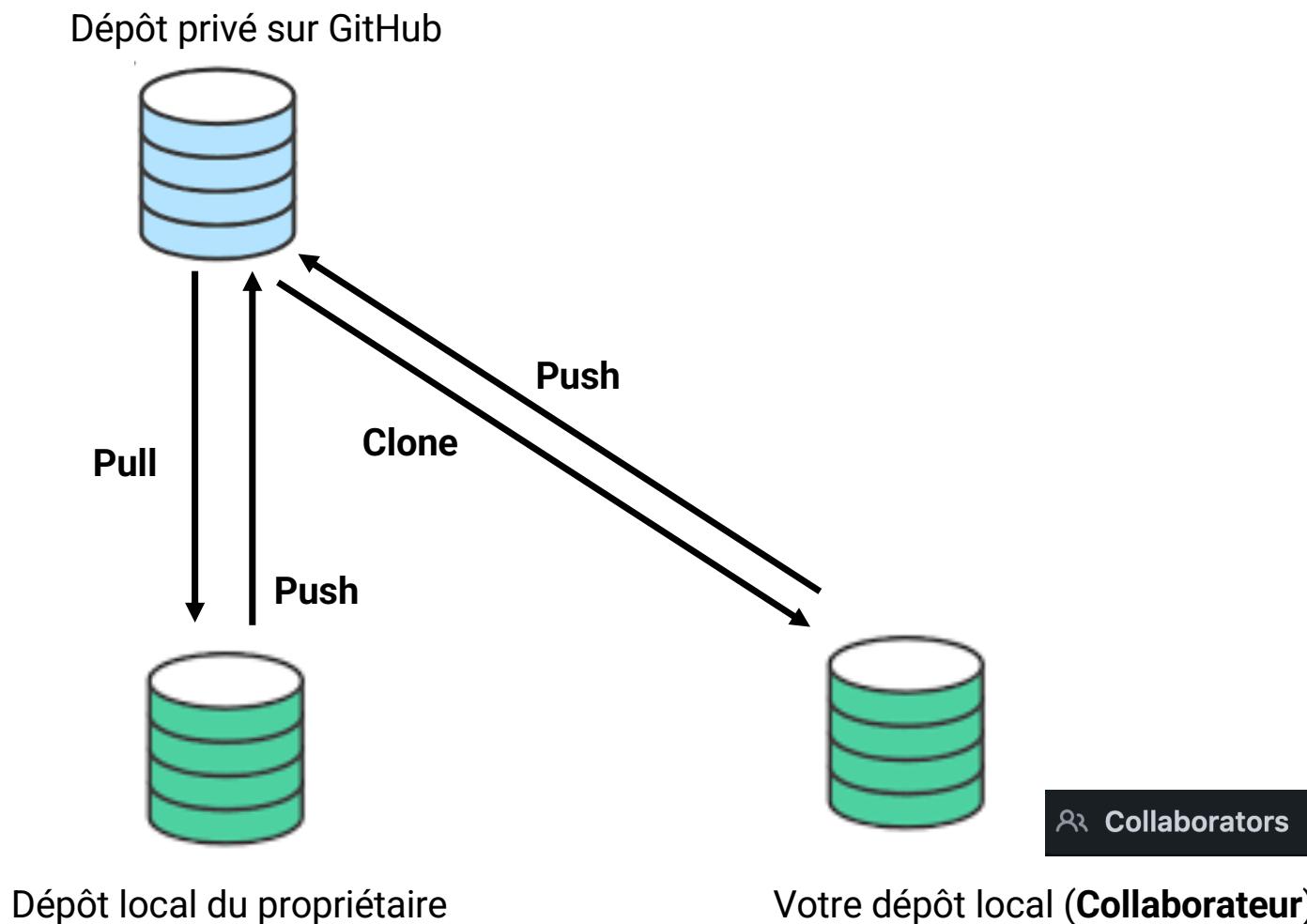
## Gestion des dépôts distants – Mode dictateur et ses lieutenants



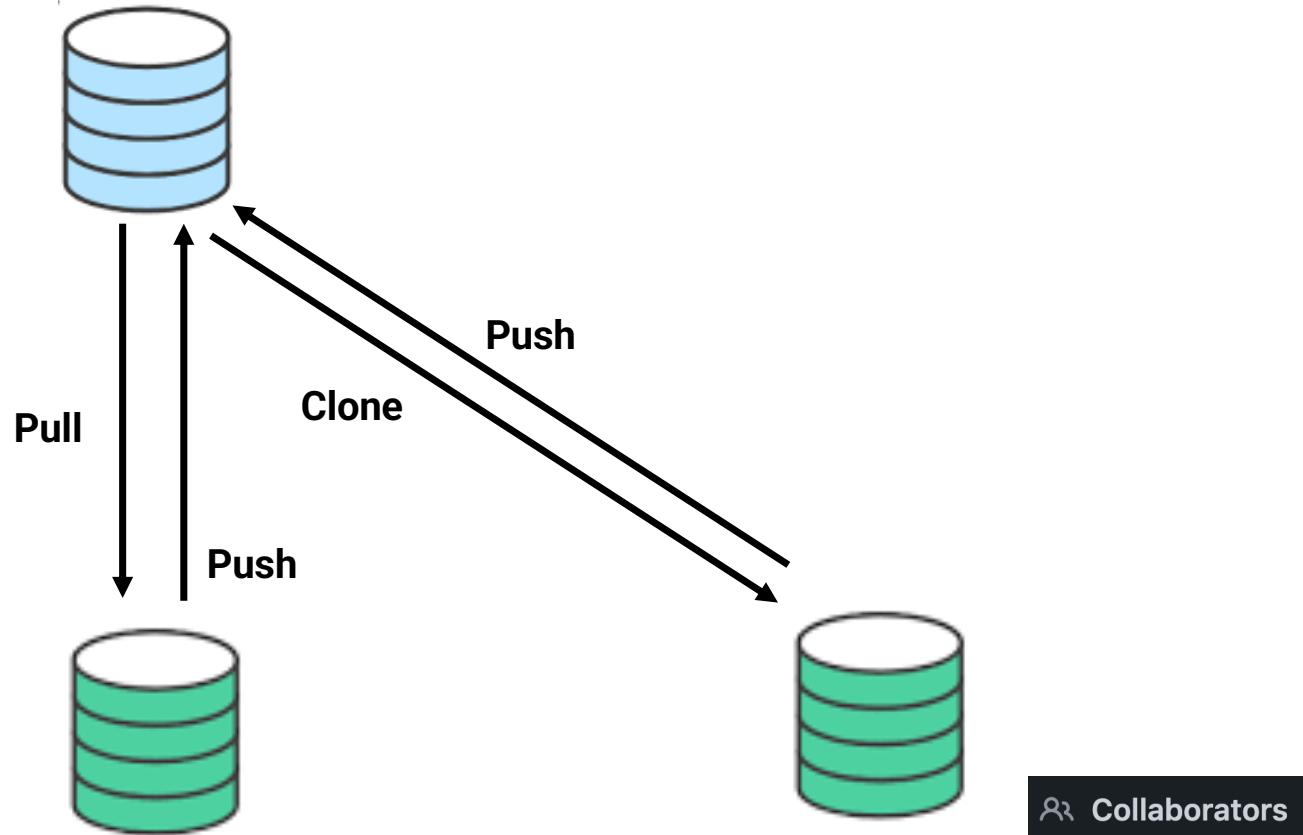
Mode utilisé lors du développement du noyau Linux (+100 collaborateurs)

- Création d'un compte
- Accès dépôt distant
  - HTTPS (nom d'utilisateur + mot de passe)
  - SSH
    - Génération d'une paire de clé (Publique et privée)
      - Keygen (Linux ou Mac)
      - PuTTY (Windows)
    - Copier la clé publique dans GitHub (contenu du fichier `~/.ssh/id_rsa.pub`)





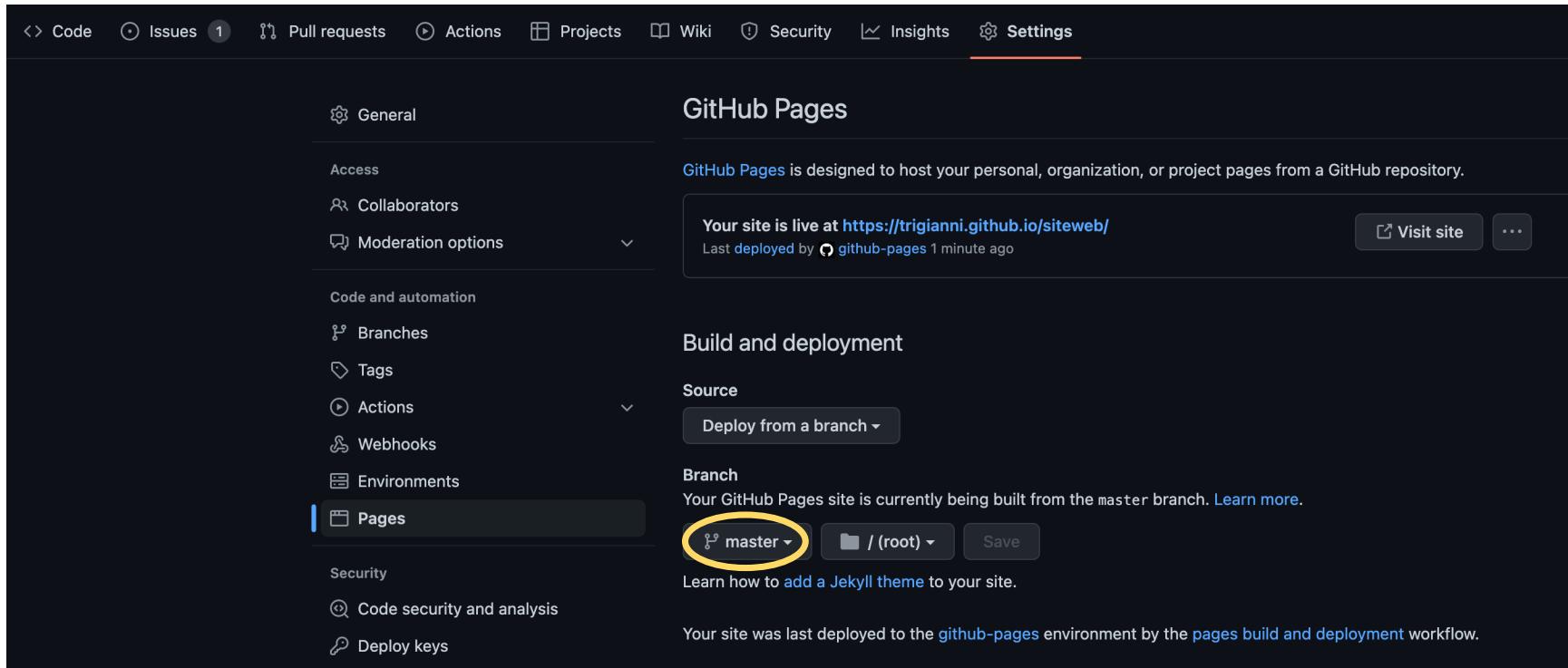
Dépôt public sur GitHub



Dépôt local du propriétaire

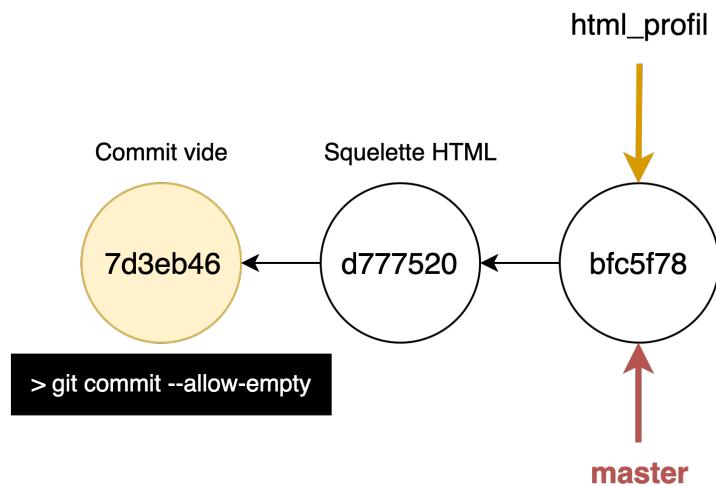
Votre dépôt local (**Collaborateur**)

## Mettre en ligne son CV à l'aide de GitHub Pages



The screenshot shows the GitHub Pages settings page for a repository. The top navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings (which is currently selected). The left sidebar lists General, Access, Collaborators, Moderation options, Code and automation (Branches, Tags, Actions, Webhooks, Environments), and Pages (which is highlighted with a blue bar). The main content area is titled "GitHub Pages" and states: "GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository." It displays a message: "Your site is live at <https://trigianni.github.io/siteweb/>" and "Last deployed by [github-pages](#) 1 minute ago". Below this is a "Visit site" button and a three-dot menu. The "Build and deployment" section shows "Source" set to "Deploy from a branch" with a dropdown menu showing "master" (which is circled in yellow) and "/ (root)". There is a "Save" button. A note says, "Your GitHub Pages site is currently being built from the master branch. Learn more." Another note says, "Learn how to [add a Jekyll theme](#) to your site." At the bottom, it says, "Your site was last deployed to the [github-pages](#) environment by the [pages](#) build and deployment workflow."

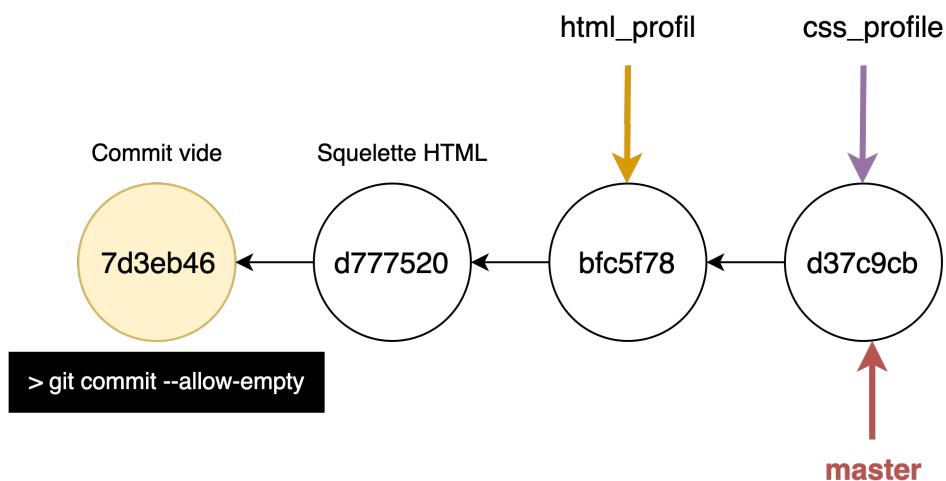
## Dépôt local



Master : branche de production

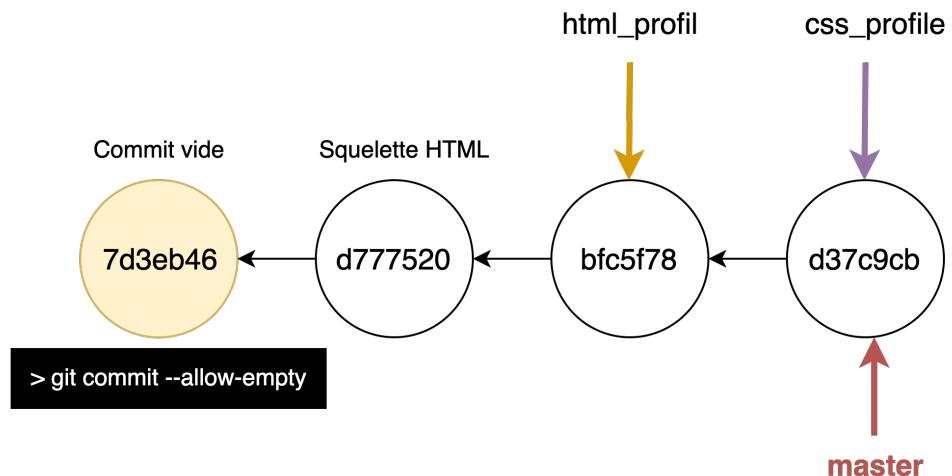
html\_profil : branche de travail

## Dépôt local



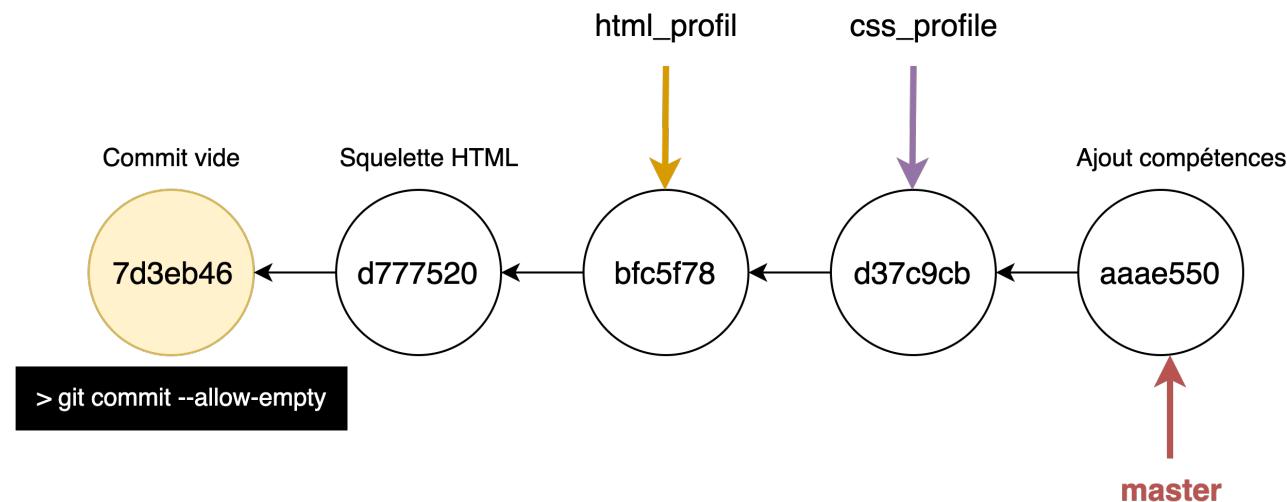
## Dépôt distant - GitHub (origin)

Pousser les commits sur le dépôt distant

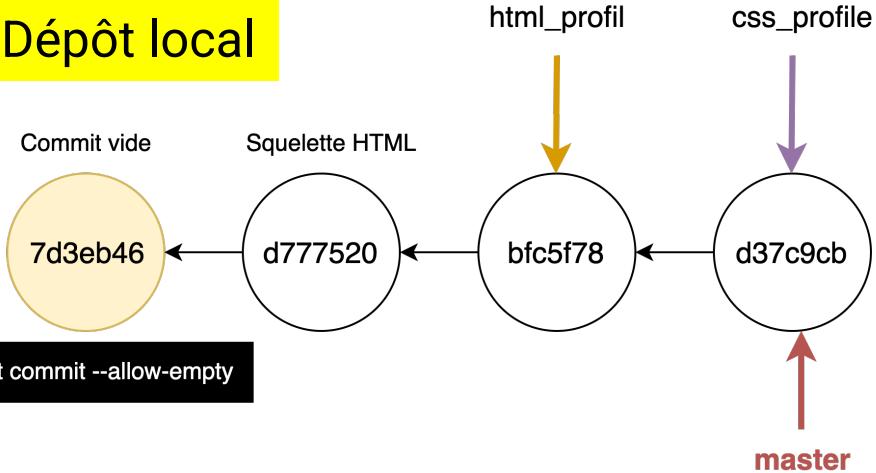


## Dépôt distant - GitHub (origin)

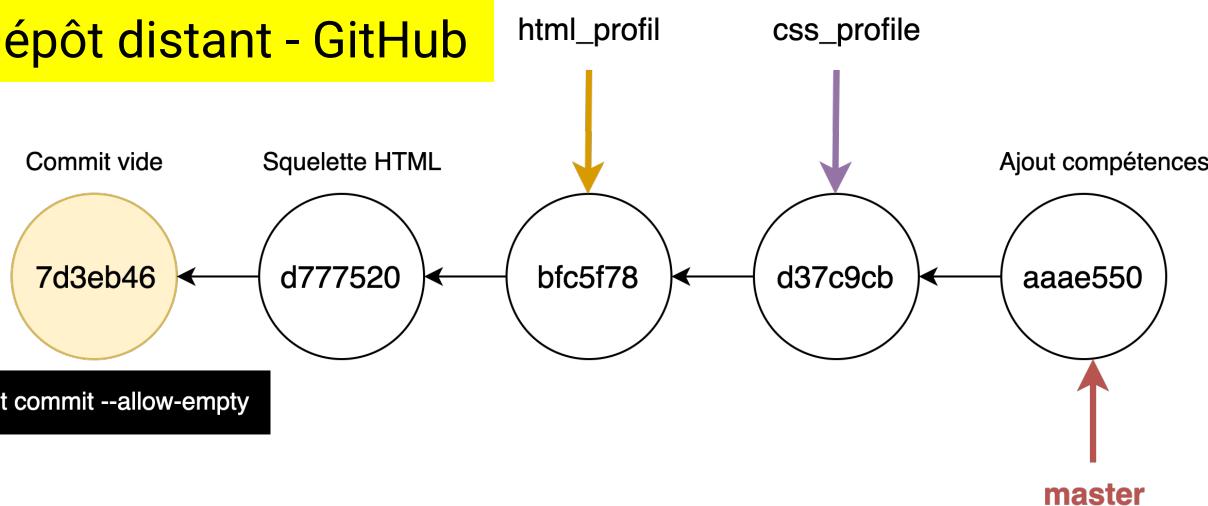
### Création d'un commit avec GitHub



## Dépôt local



## Dépôt distant - GitHub



Dépôt local

## Récupération du commit distant

