Le versioning avec Git

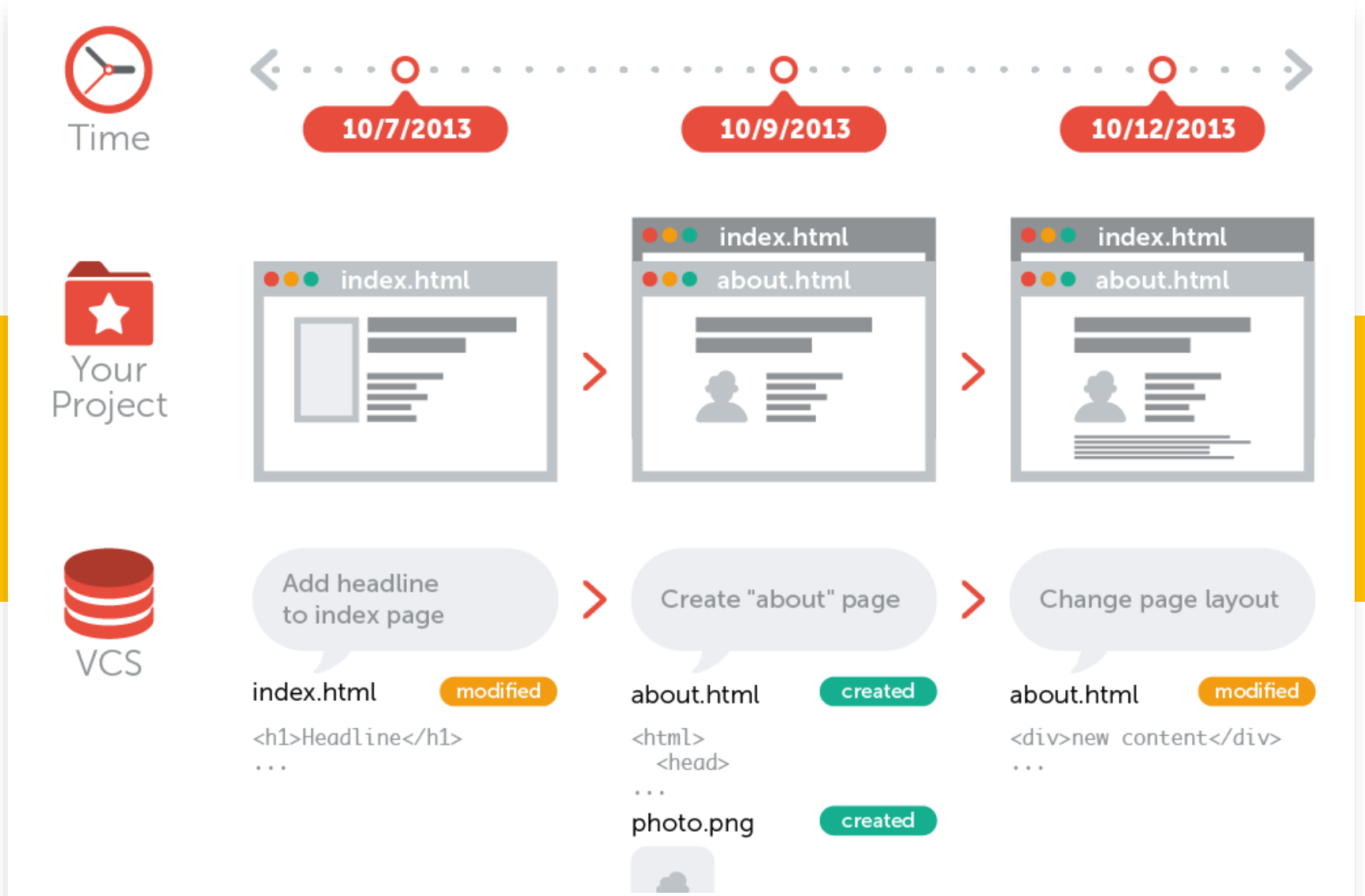
Elies Jebri

Elies Jebri : elies.jebri@gmail.com

Définition

- Git est un système de gestion de version distribué (DVCS).
- Un gestionnaire de version est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment.
- Ce n’est pas un outil de *backup*.

Elies Jebri : elies.jebri@gmail.com



Version Control System

Principales utilités

- Nombreuses fonctionnalités utiles le long de l'évolution d'un projet informatique :
  - Ils retiennent qui a effectué chaque modification de chaque fichier et *pourquoi*.
  - Si plusieurs personnes travaillent simultanément sur un même fichier, ils sont capables d'assembler (de fusionner) leurs modifications et d'éviter que le travail d'une de ces personnes ne soit écrasé.

Elies Jebri : elies.jebri@gmail.com

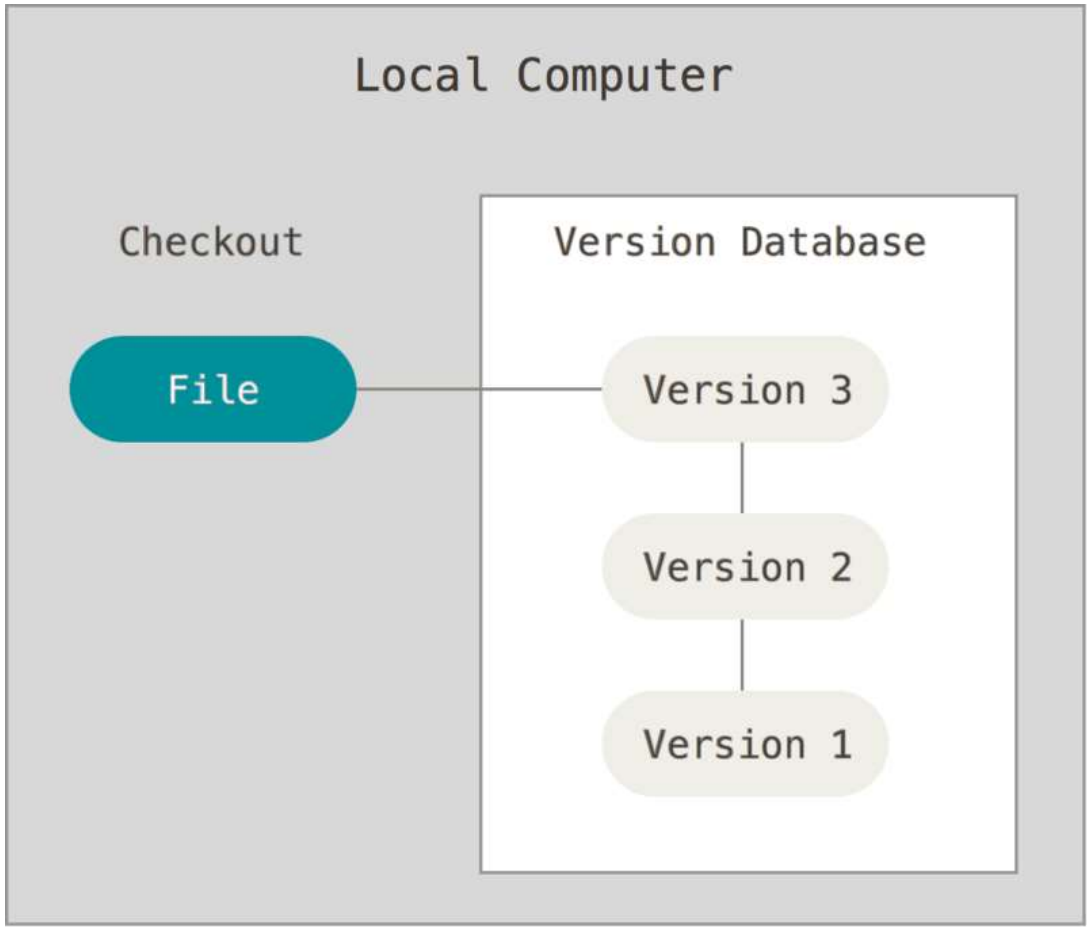
Logiciels Locaux, centralisés et distribués

Les systèmes de gestion de version locaux

Cette méthode est la plus courante parce que c'est la plus simple pour la gestion de version.

Il s'agit généralement de recopier les fichiers dans un autre répertoire (avec des métadonnées tq un nom une date, …)

Mais c'est aussi la moins fiable.

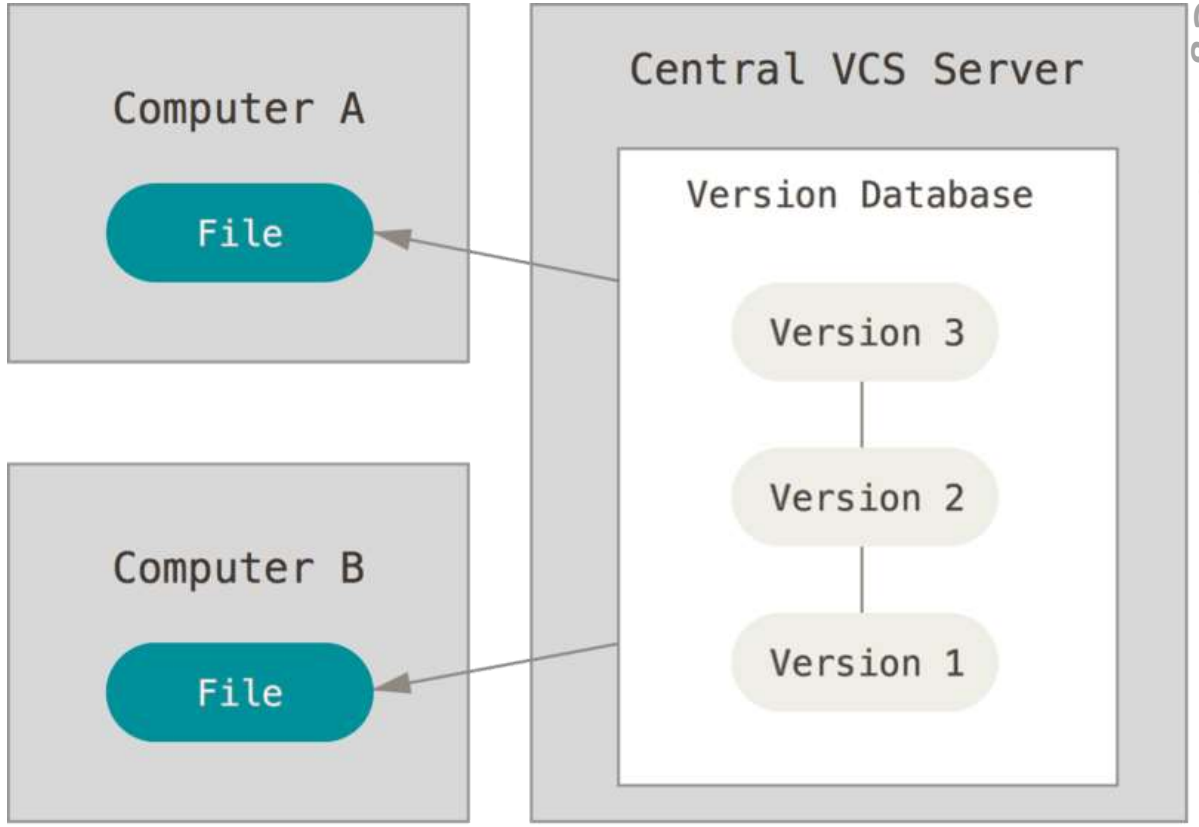


Elies Jebri : elies.jebri@gmail.com

Logiciels centralisés et distribués

Les systèmes de gestion de version centralisés

- Ces systèmes mettent en place un serveur central qui contient tous les fichiers sous gestion de version, et des clients qui peuvent extraire les fichiers de ce dépôt central.

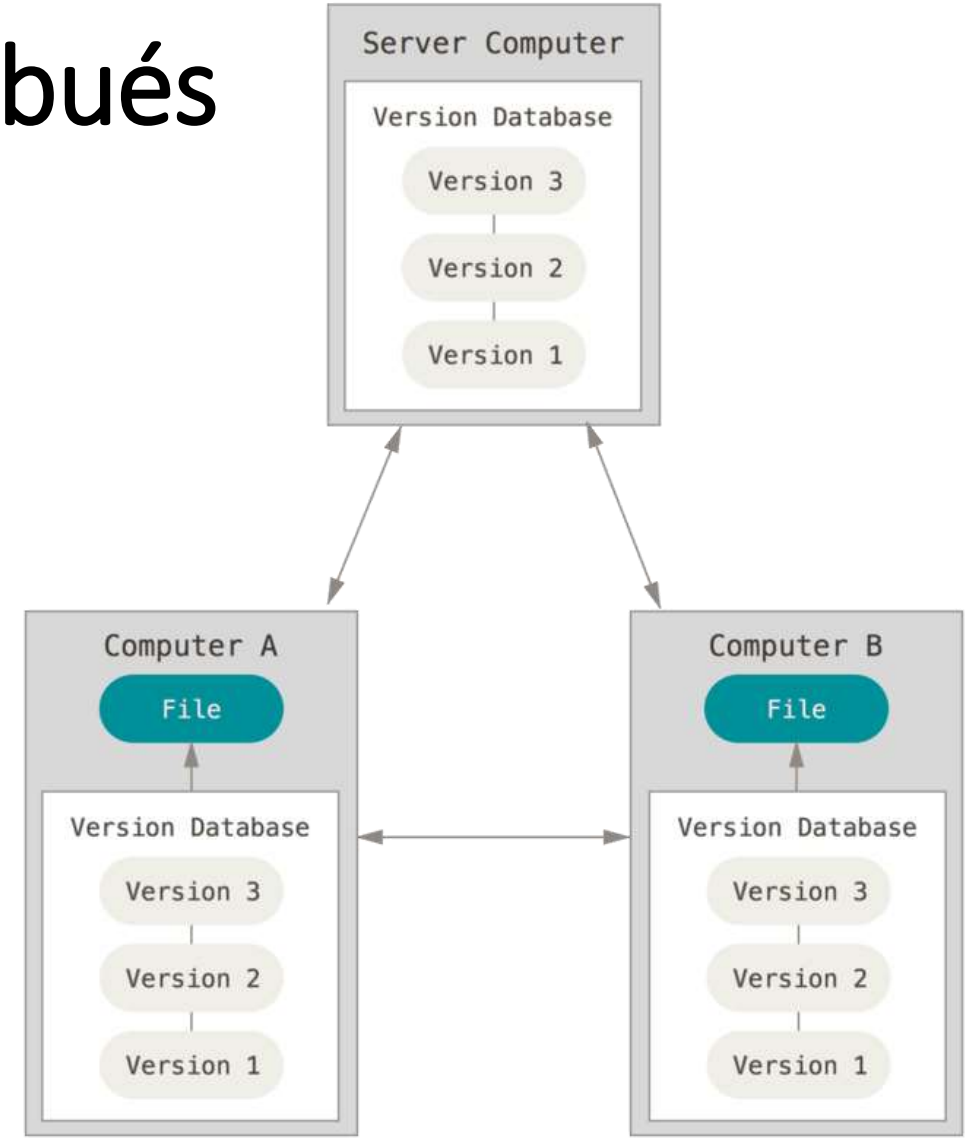


Elies Jebri : elies.jebri@gmail.com

Logiciels centralisés et distribués

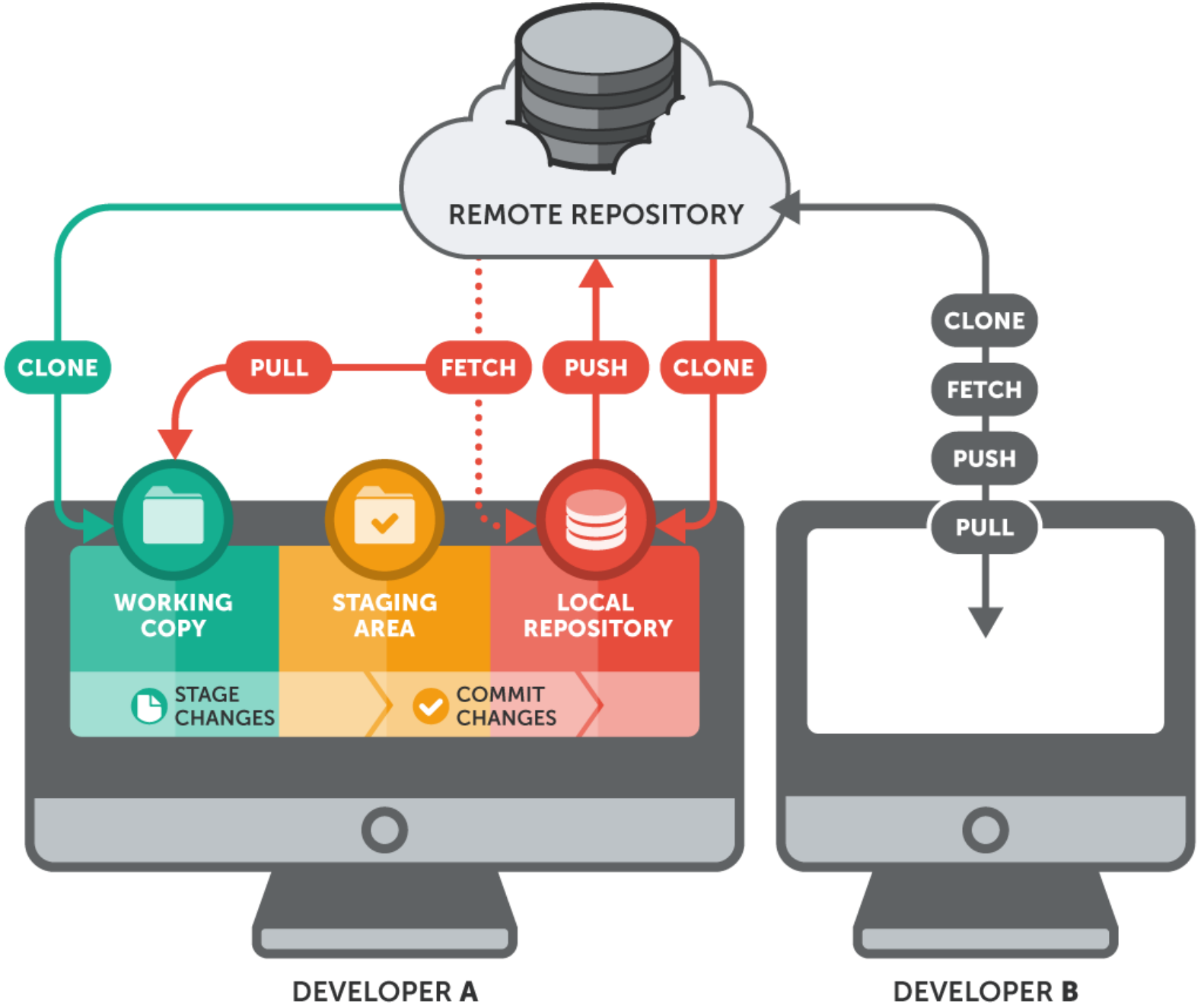
Les systèmes de gestion de version distribués (DVCS)

- Les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt.
- Ainsi, si le serveur disparaît et si les systèmes collaboraient via ce serveur, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer.
- Chaque extraction devient une sauvegarde complète de toutes les données.

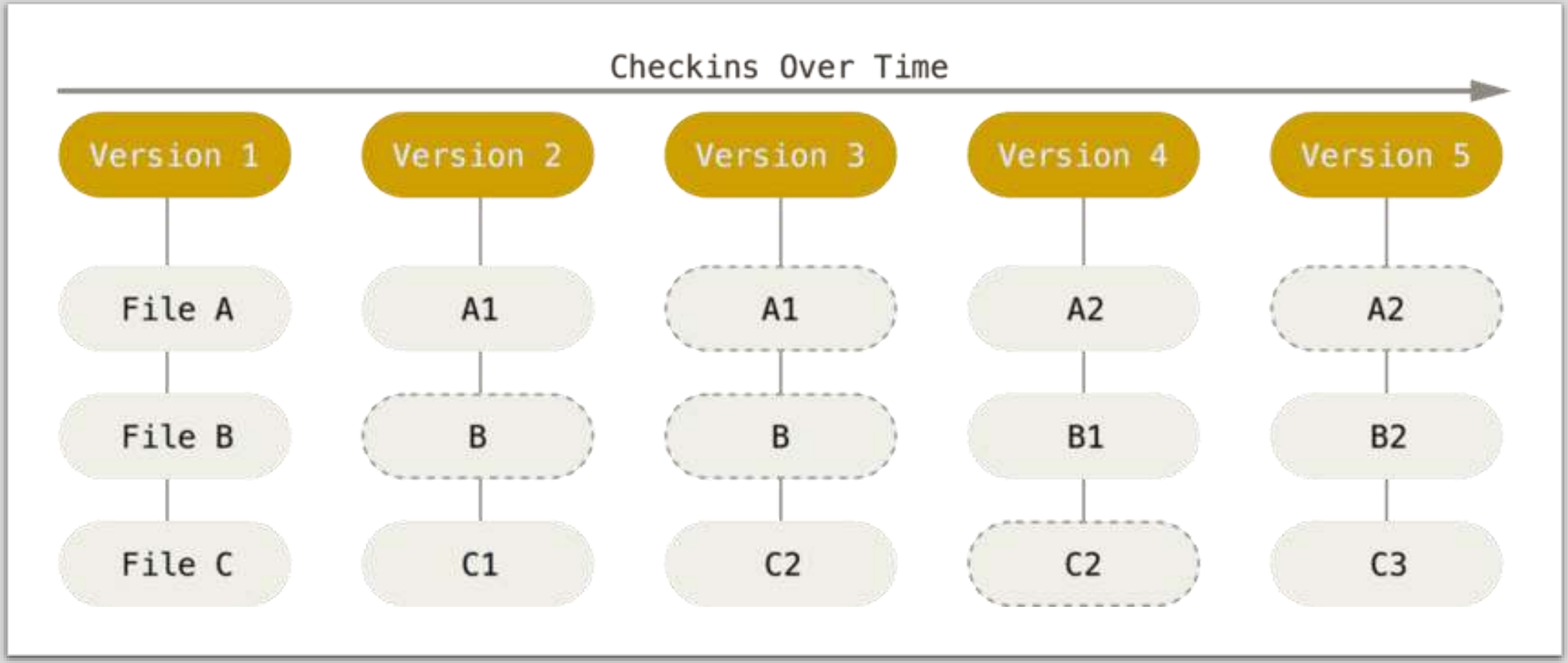


Elies Jebri : elies.jebri@gmail.com

Git



Elies Jebri : elies.jebri@gmail.com



Les snapshots de Git

Rudiments de Git

- Presque toutes les opérations sont locales
- Git gère l'intégrité (SHA-1)
- Généralement, Git ne fait qu'ajouter des données
- Les trois états dans lesquels les fichiers peuvent se trouver : modifié, indexé, validé (modified, staged, committed)
  - vous modifiez des fichiers dans votre répertoire de travail ;
  - vous indexez les fichiers modifiés, ce qui ajoute des instantanés de ces fichiers dans la zone d'index ;
  - vous validez, ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la base de données du répertoire Git.

Elies Jebri : elies.jebri@gmail.com

Working Copy

Your Project's Files

Git watches tracked files for new local modifications...

Staging Area

Changes included in the Next Commit

Local Repository

The ".git" Folder

Tracked (and modified)

M — If a file was modified since it was last committed, you can stage & commit these changes

stage

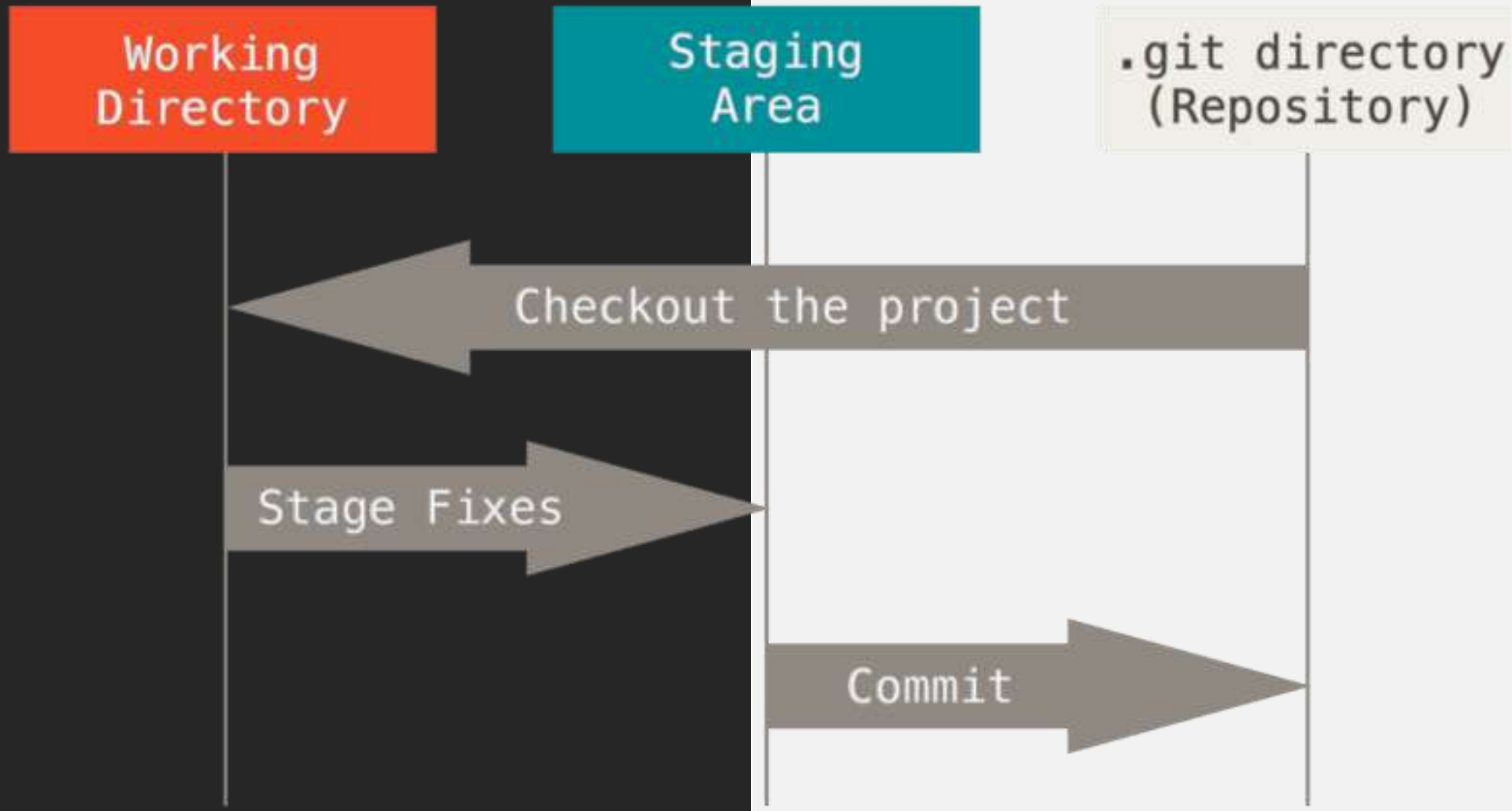
M — Changes that were added to the Staging Area will be included in the next commit

commit

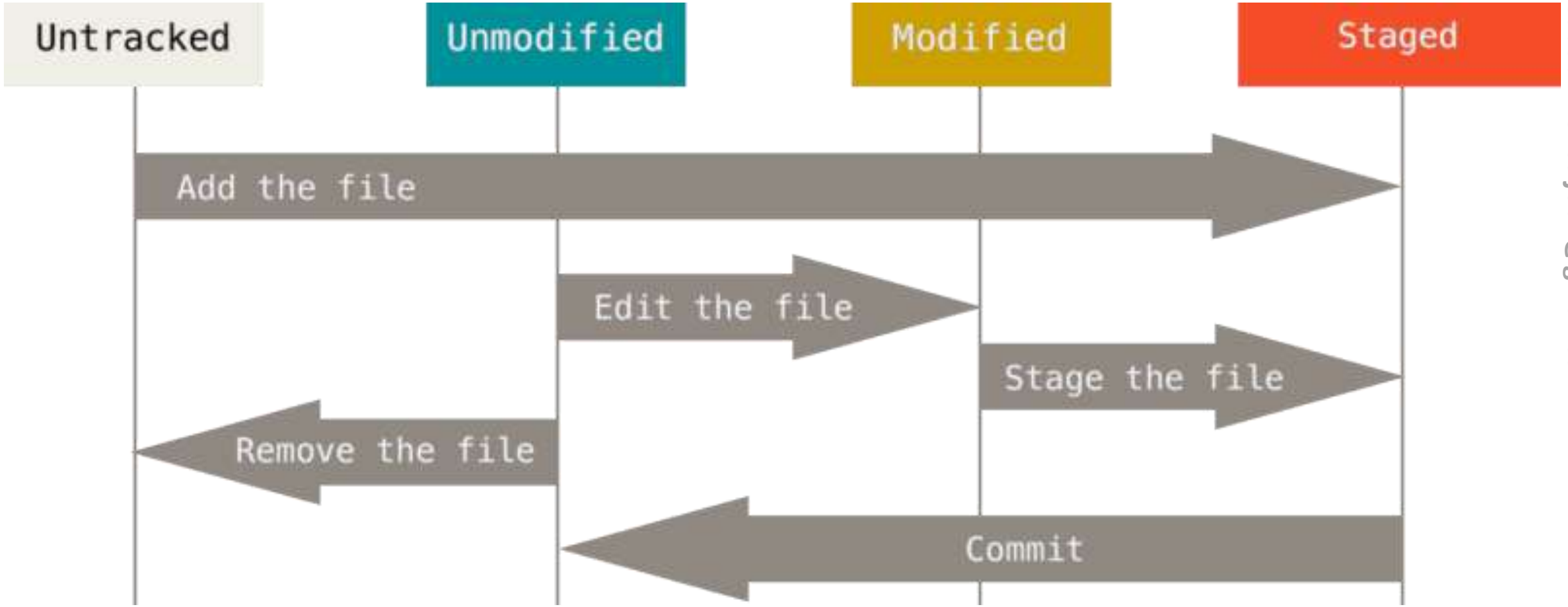
All changes contained in a commit are saved in the local repository as a new revision

M — Changes that are **not staged** will not be committed & remain as local changes until you stage &

Les trois états



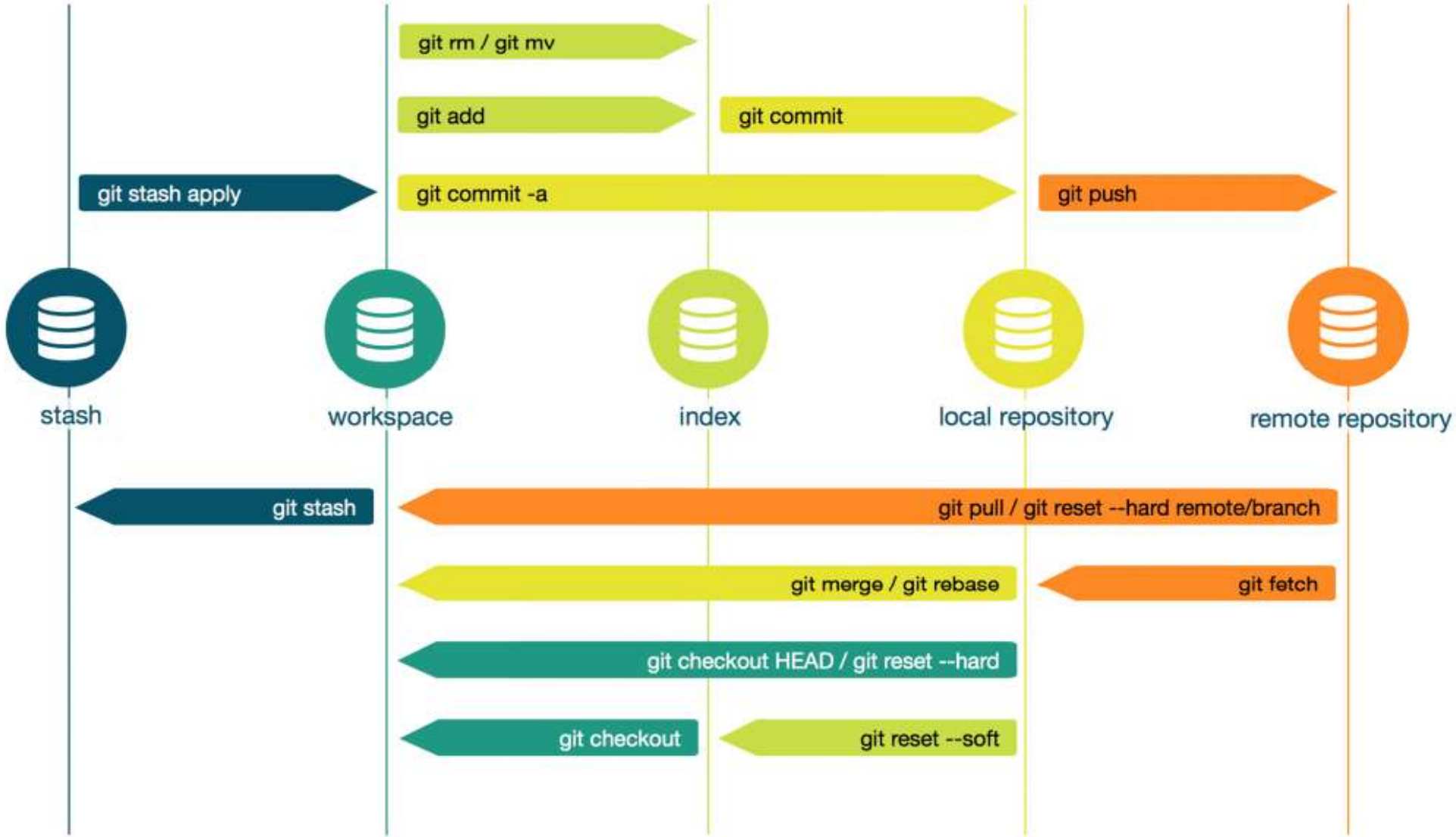
Workflow des trois états.



Le cycle de vie des états des fichiers.

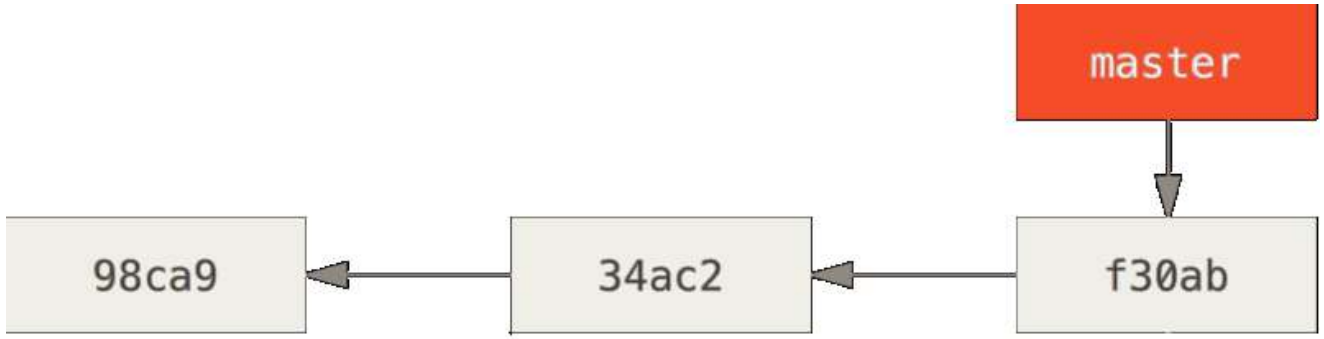
Elies Jebri : elies.jebri@gmail.com

Git cheatsheet



Les branches en bref

- L’indexation calcule un checksum SHA-1 pour chaque fichier, stocke cette version du fichier dans le dépôt Git (Git les nomme **blobs**) et ajoute cette empreinte à la staging area.
- Lors d’un commit, cet objet contient noms, prénoms et message que vous avez renseigné ainsi que des pointeurs vers le ou les commits qui précèdent directement ce commit.
  - Aucun parent pour le commit initial,
  - Un parent pour un commit normal
  - Multiples parents pour un commit qui résulte de la fusion d’une ou plusieurs branches.



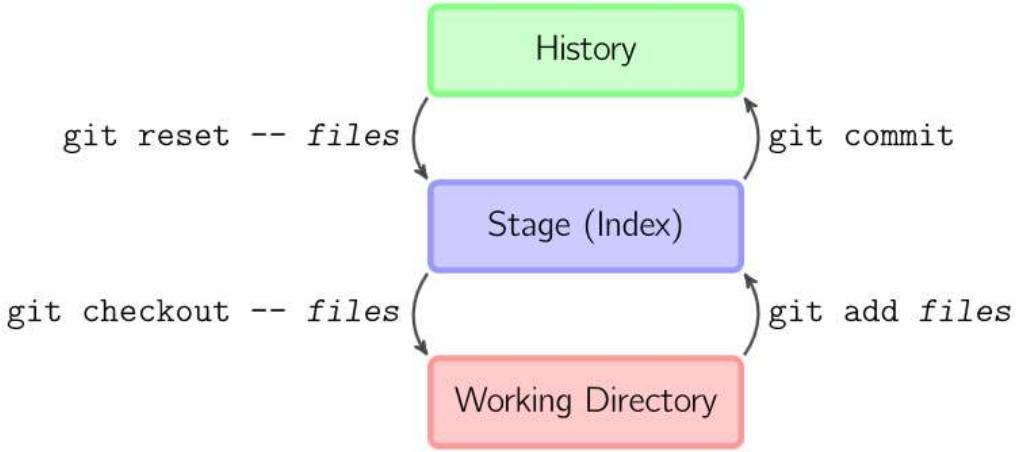
Pointeur HEAD
Branche master

- Git conserve un pointeur spécial appelé HEAD.
- Il s’agit d’un pointeur sur la branche locale où vous vous trouvez.
- Dans ce cas, vous vous trouvez sur la branche master.

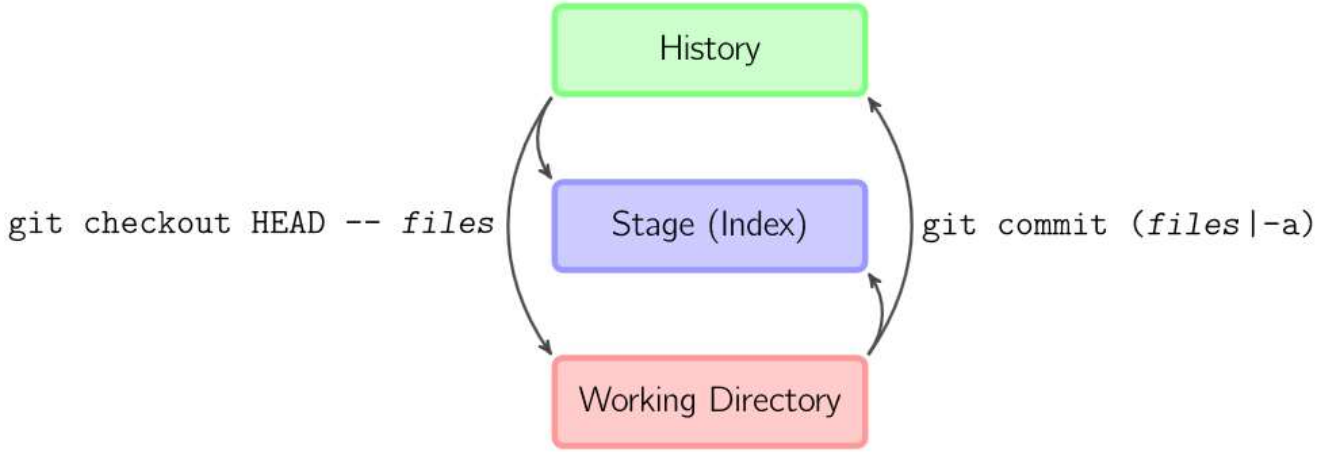
Les commandes en détail

Opérations de base

Copie des fichiers entre la working copy, le stage et l'histoire

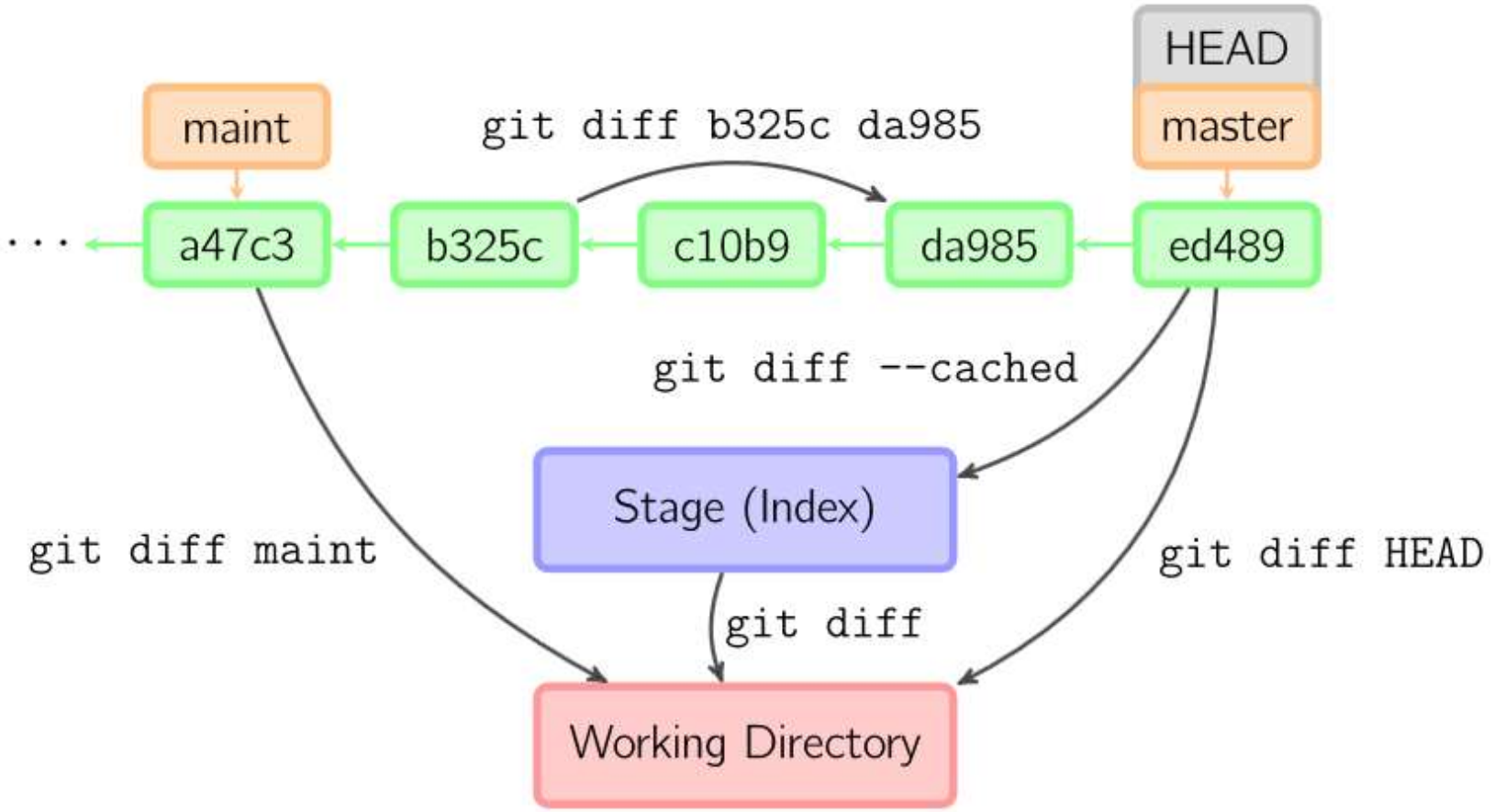


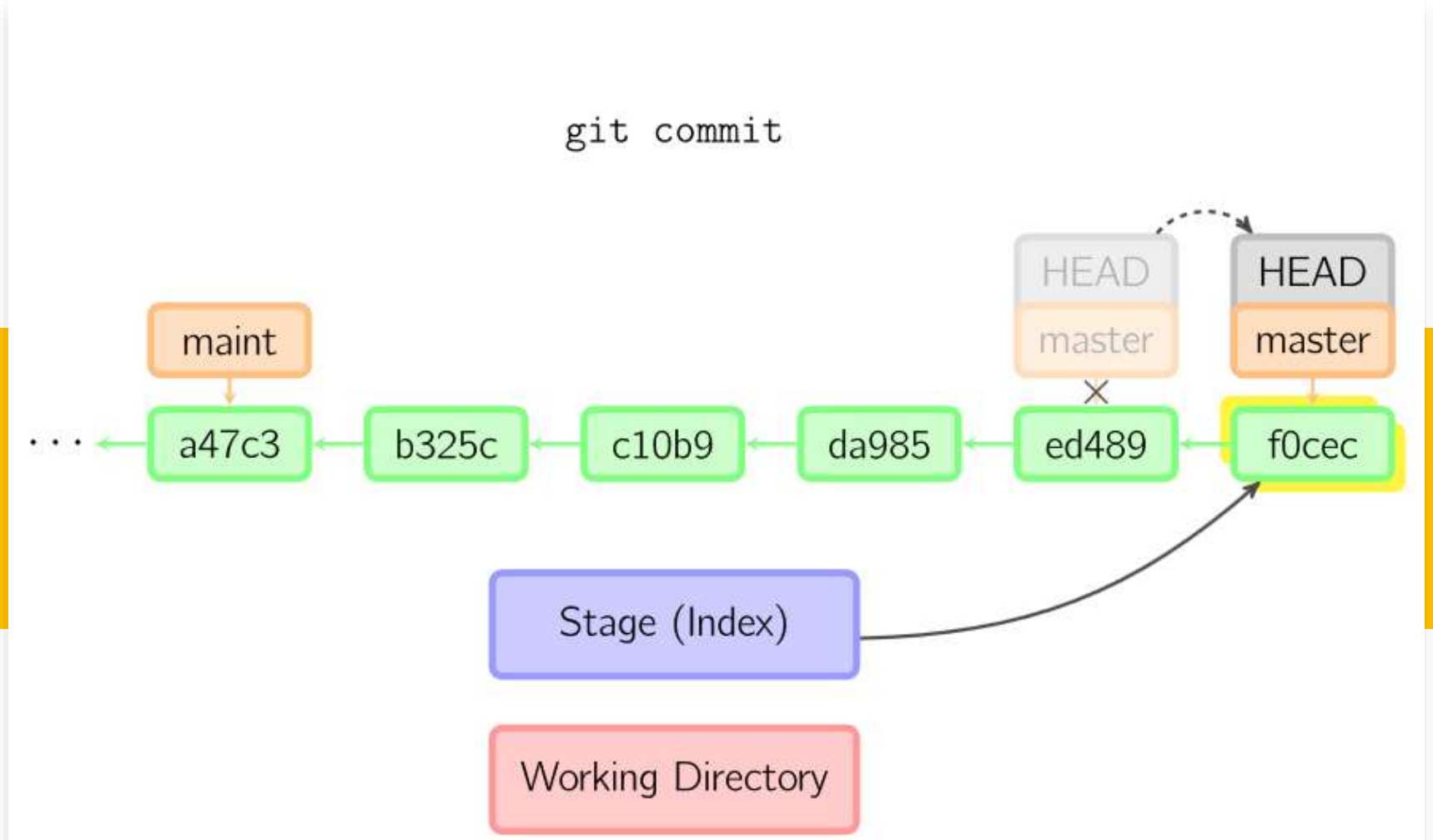
Contourner le stage et sortir (check out) les fichiers directement de l'histoire, ou commiter les fichiers sans passer par le stage.



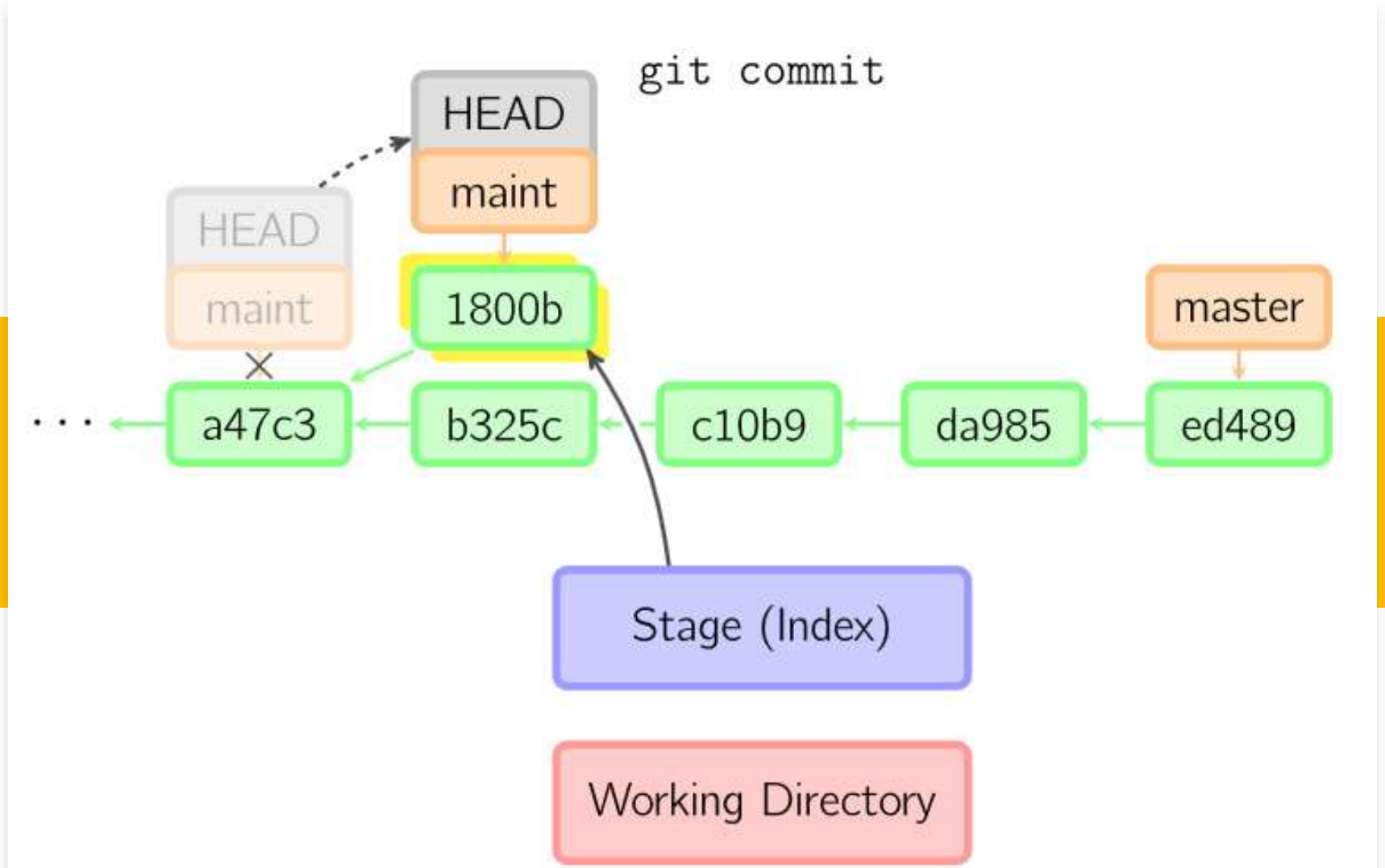
Elies Jebri : elies.jebri@gmail.com

diff

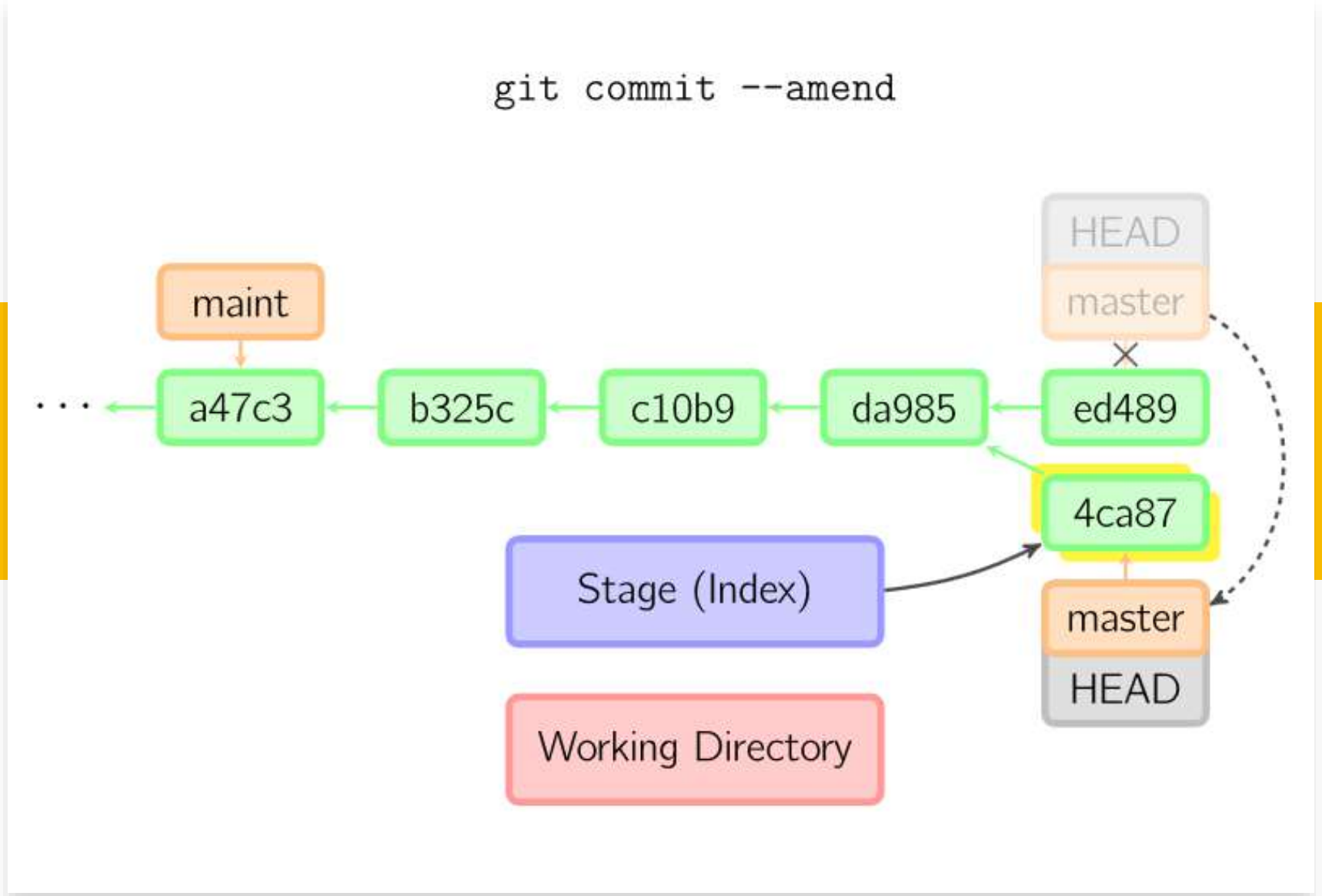

HEAD
maint
git diff b325c da985
master
...
a47c3
b325c
c10b9
da985
ed489
git diff --cached
Stage (Index)
git diff maint
git diff HEAD
git diff
Working Directory


git commit
HEAD
master
HEAD
master
maint
...
a47c3
b325c
c10b9
da985
ed489
f0cec
Stage (Index)
Working Directory

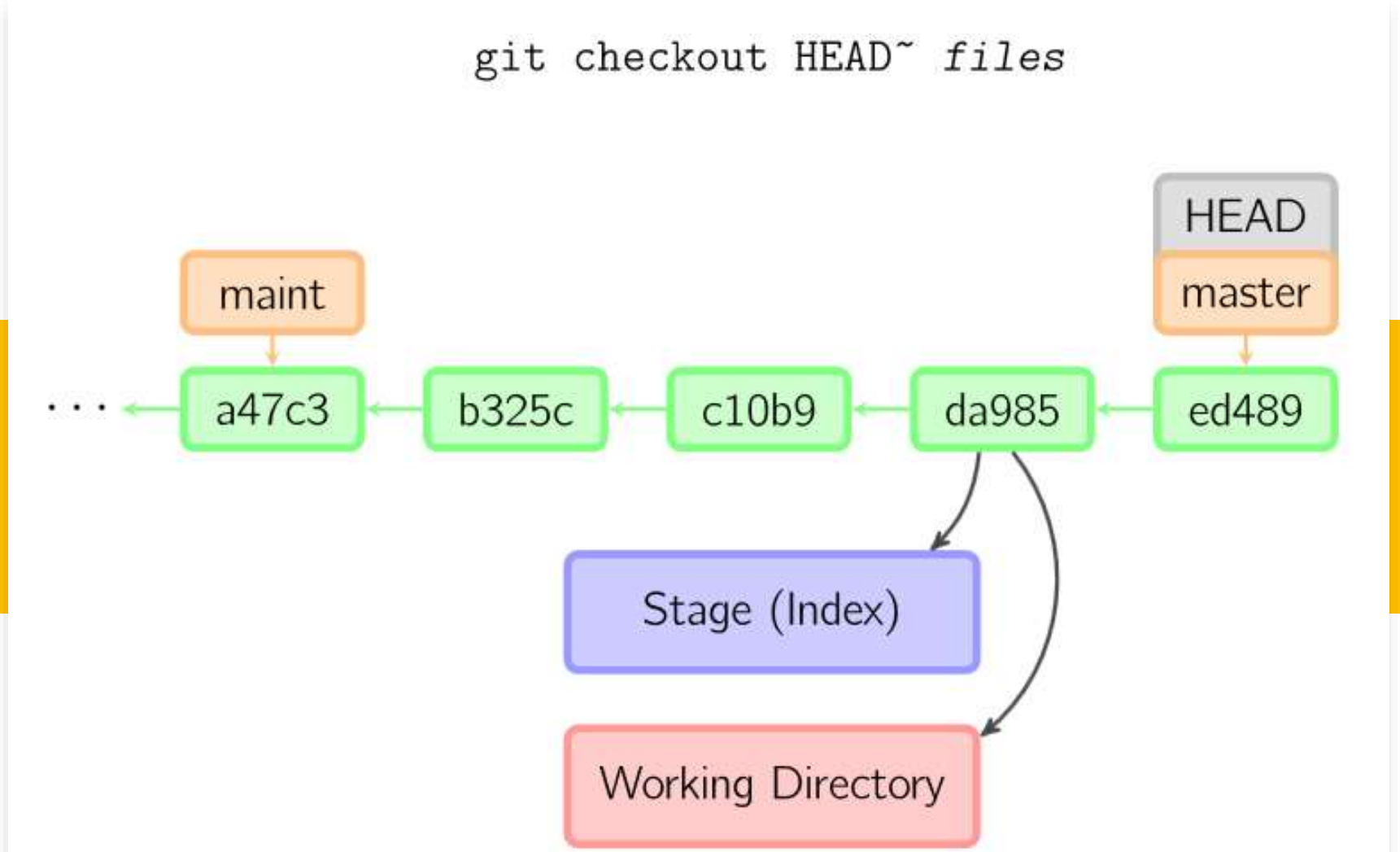
commit


git commit
HEAD
maint
HEAD
maint
1800b
master
...
a47c3
b325c
c10b9
da985
ed489
Stage (Index)
Working Directory

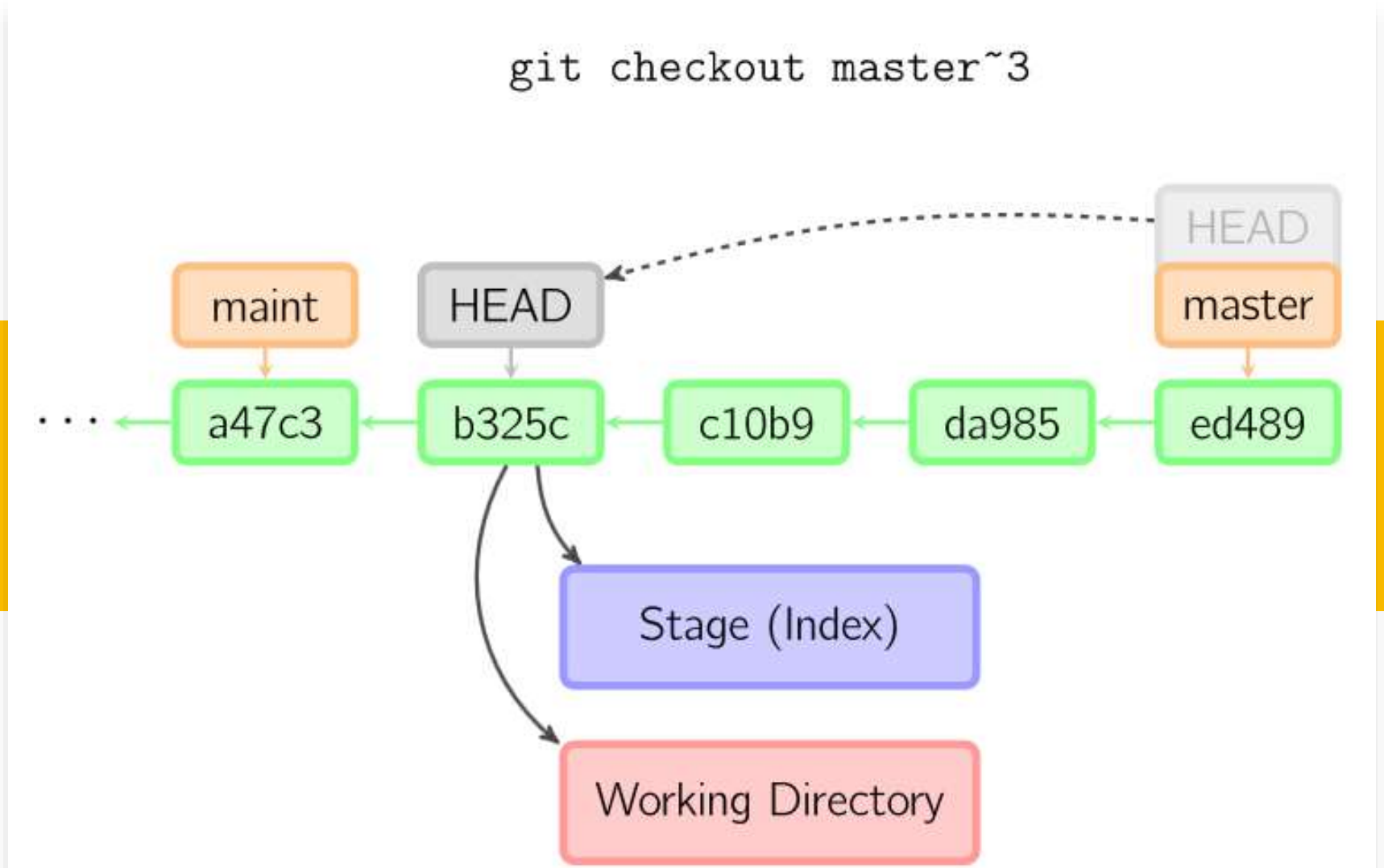
commit



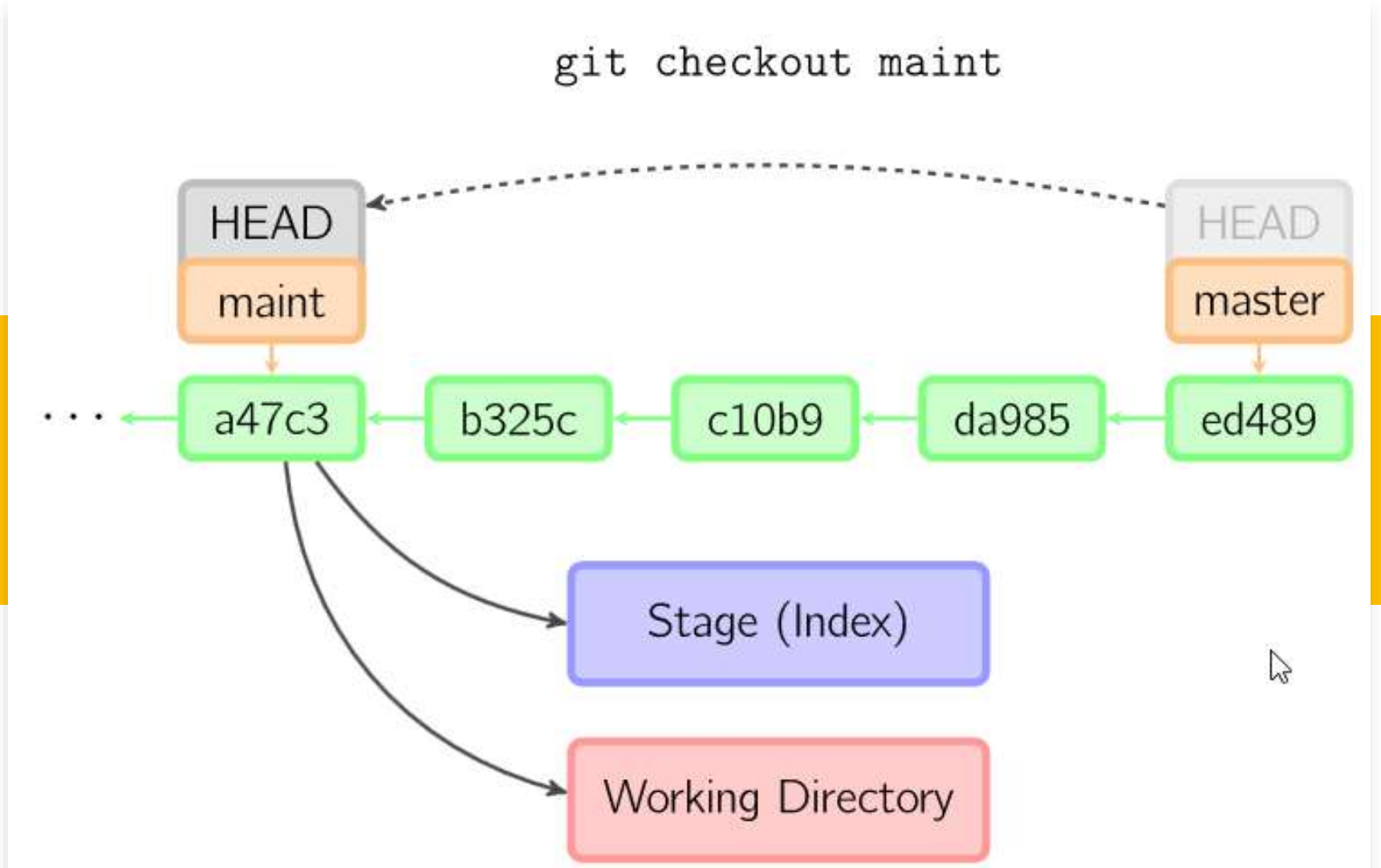
commit
--amend


git checkout HEAD~ files
HEAD
maint
master
...
a47c3
b325c
c10b9
da985
ed489
Stage (Index)
Working Directory

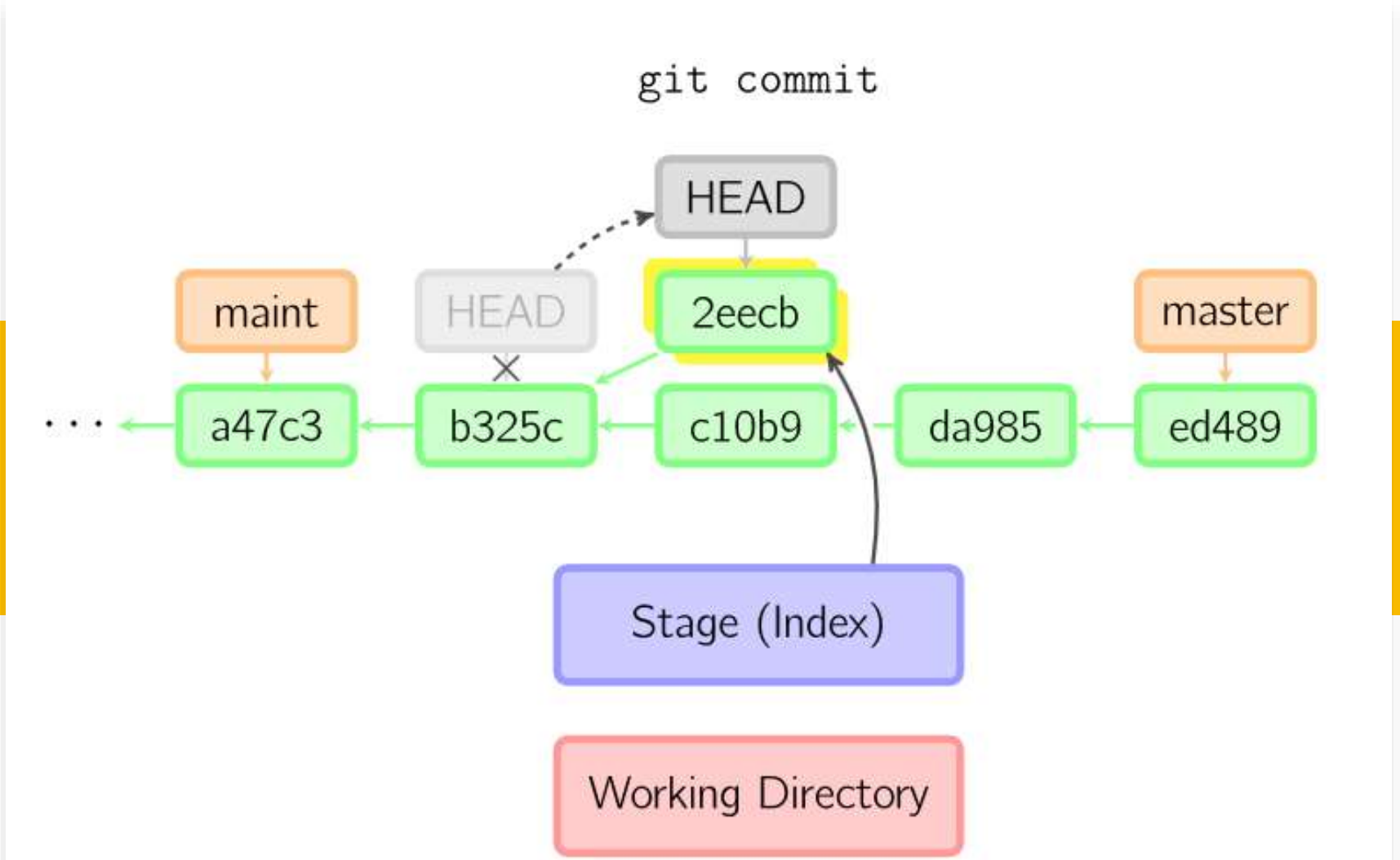
checkout



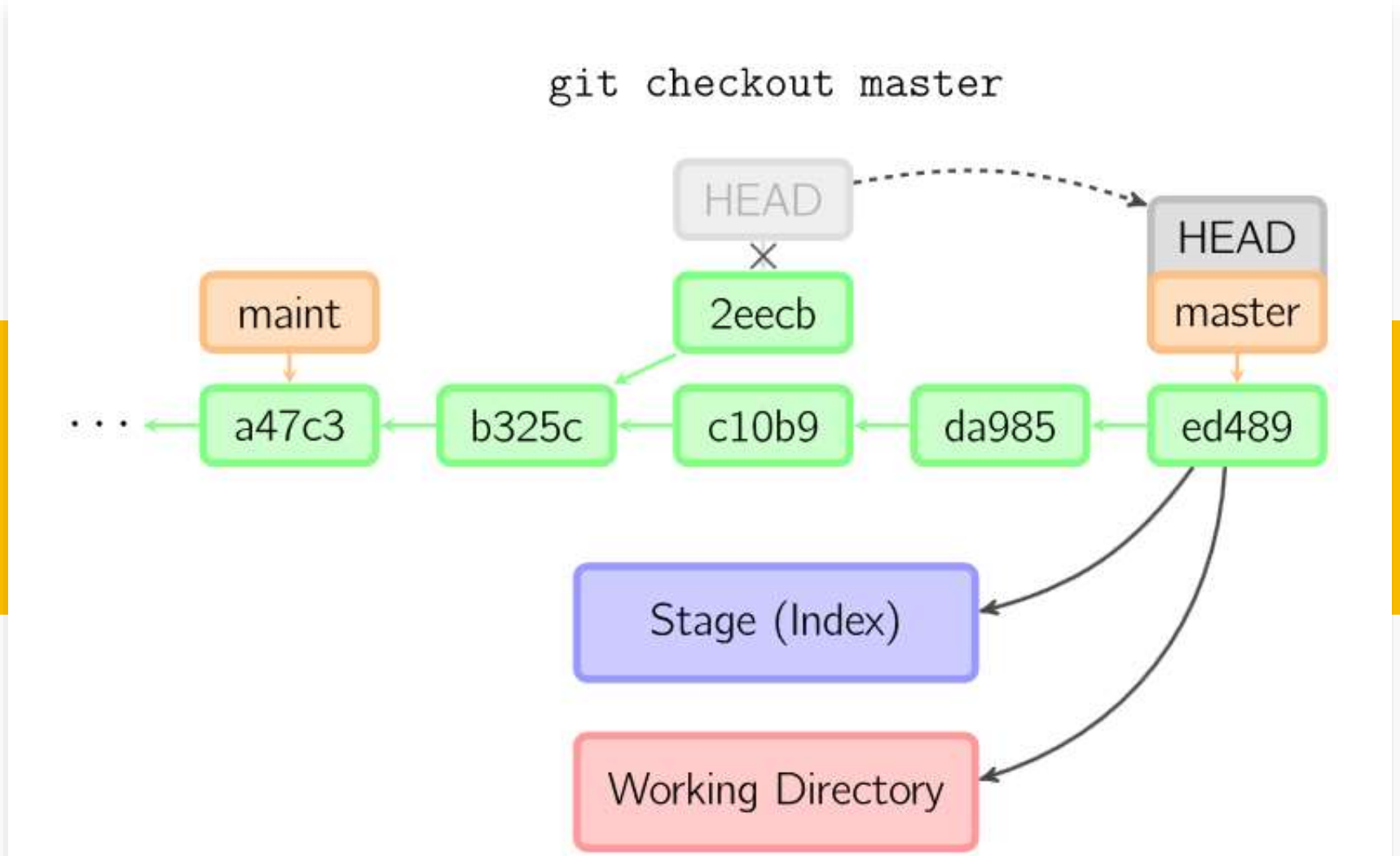
checkout (branche anonyme ou detached HEAD)



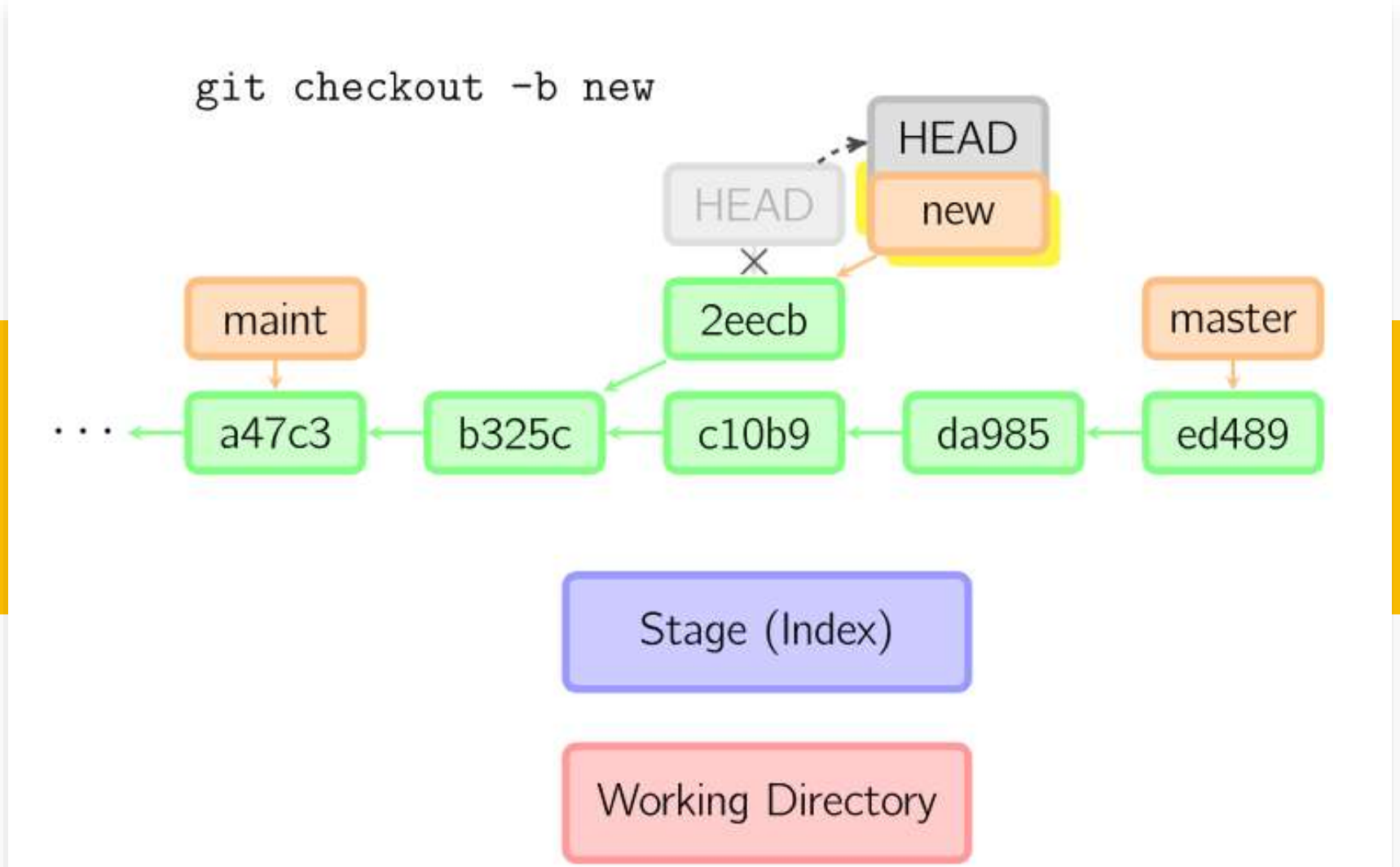
checkout branche



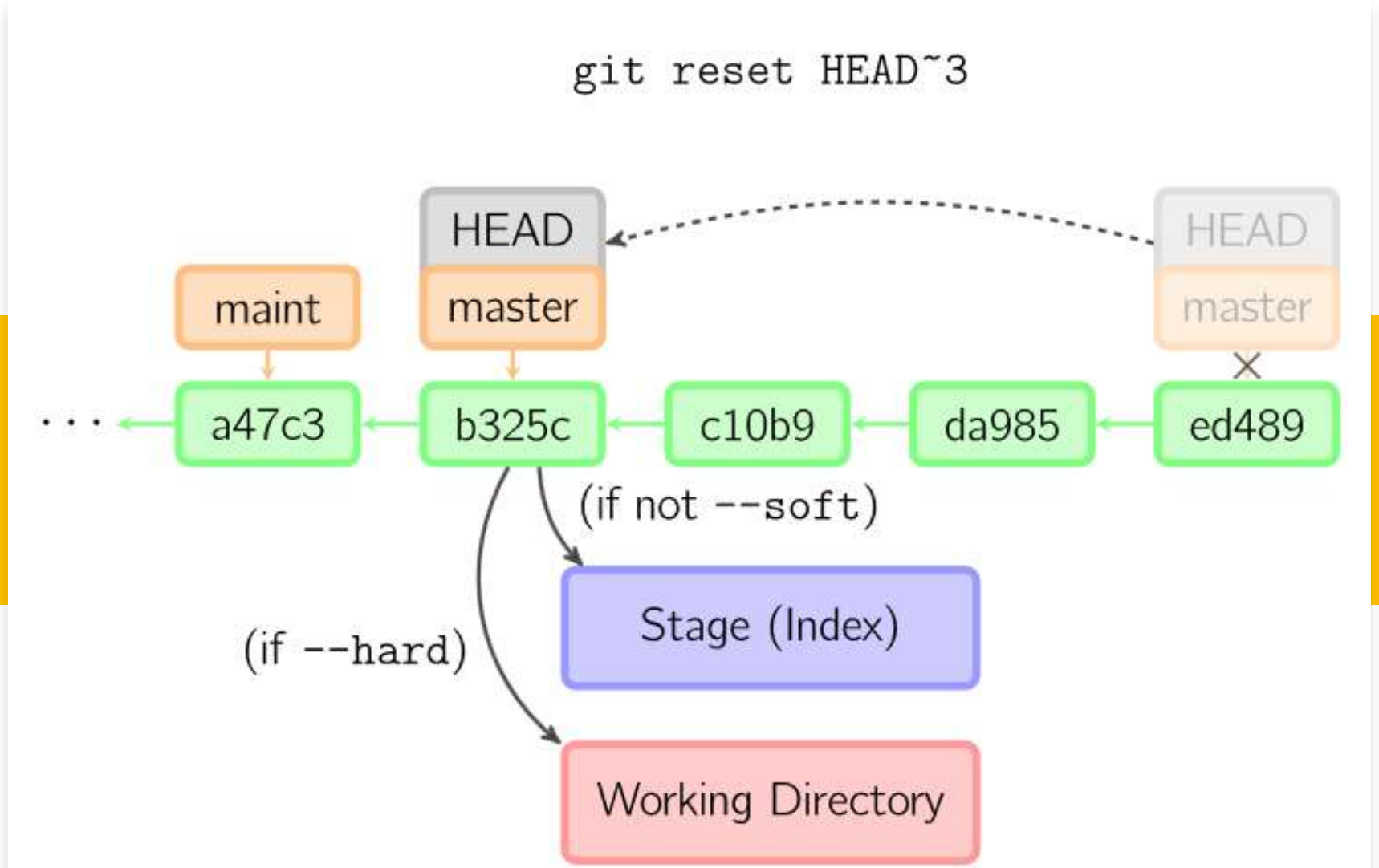
Commiter avec une "Detached HEAD"



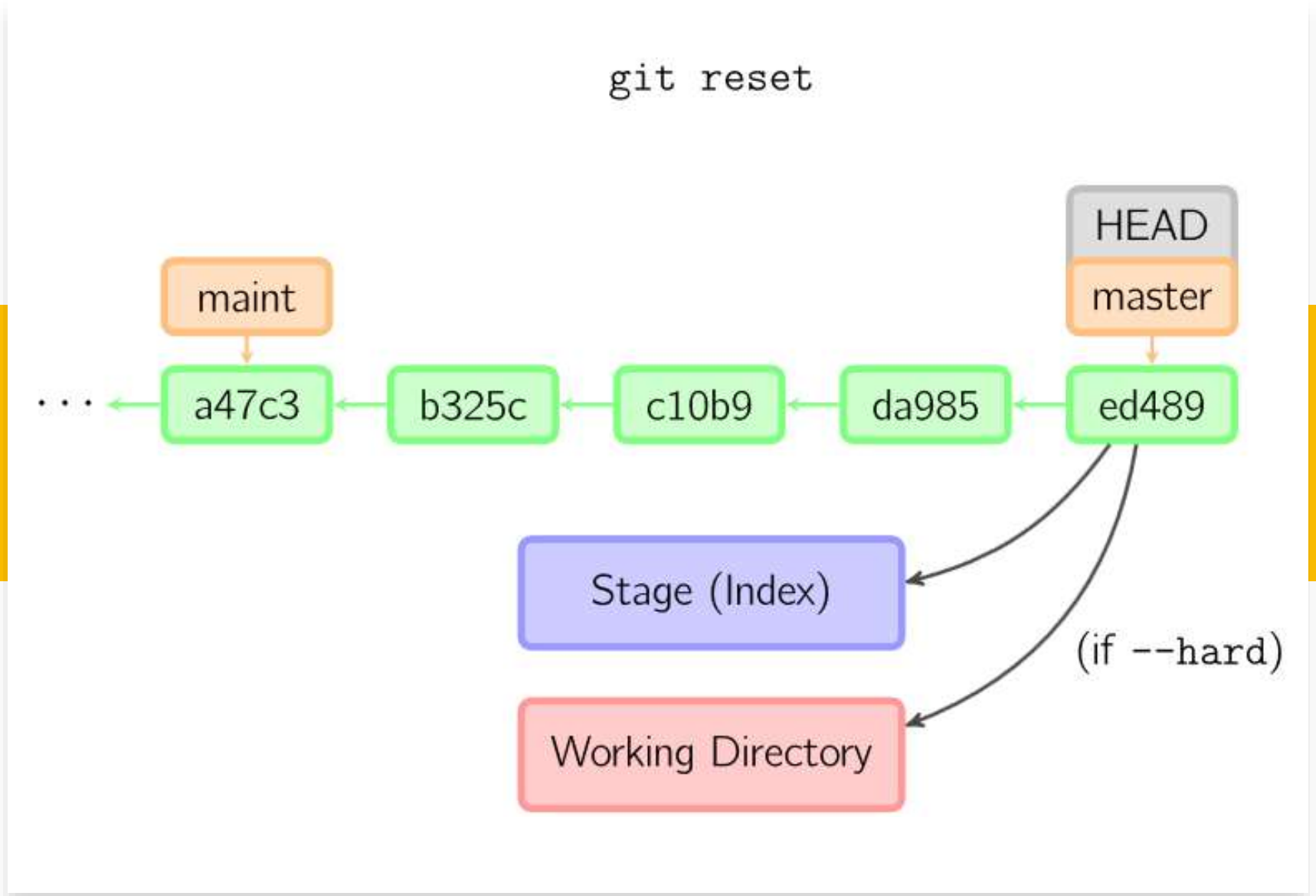
Commit non référencé et perdu



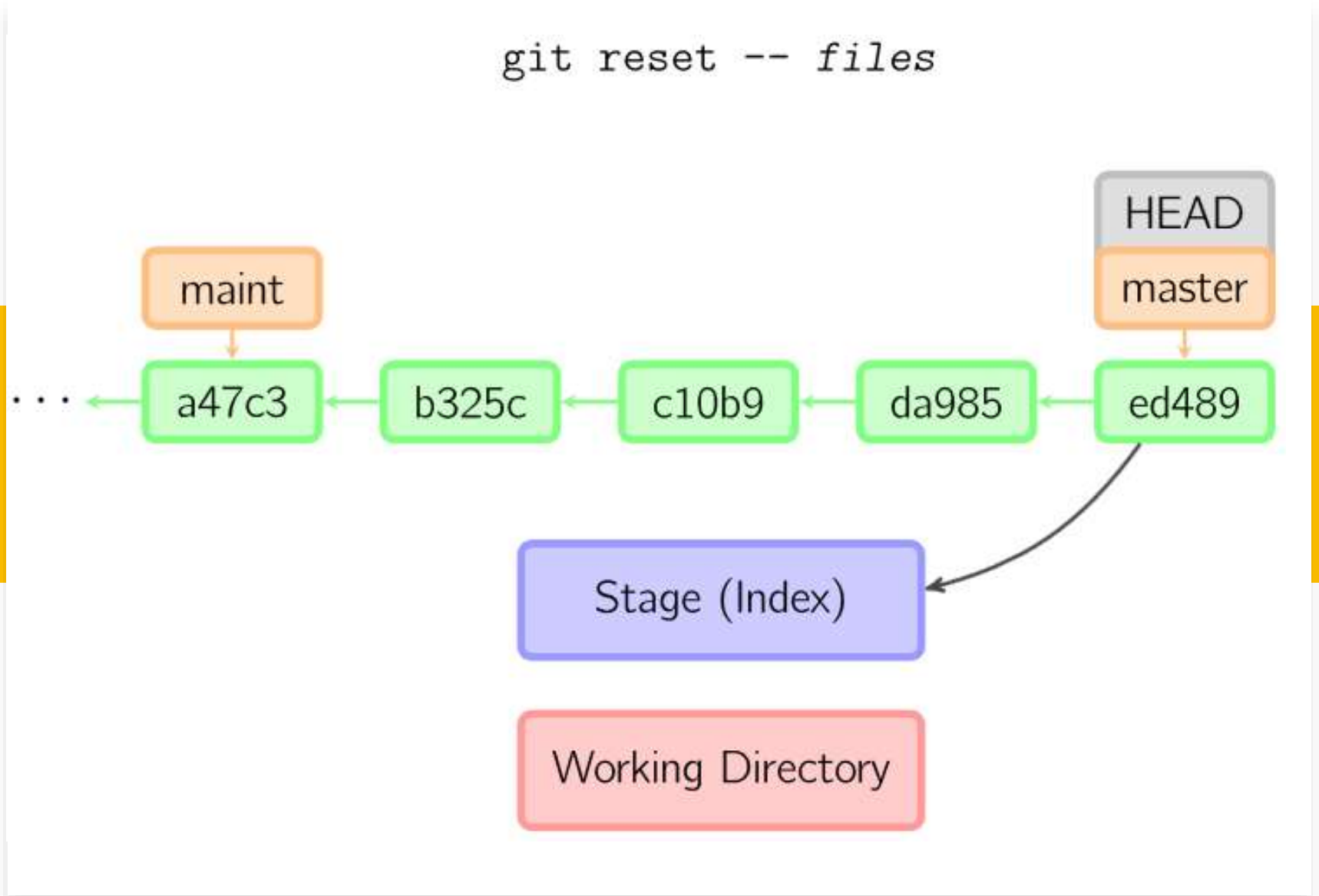
Conserver un detached HEAD



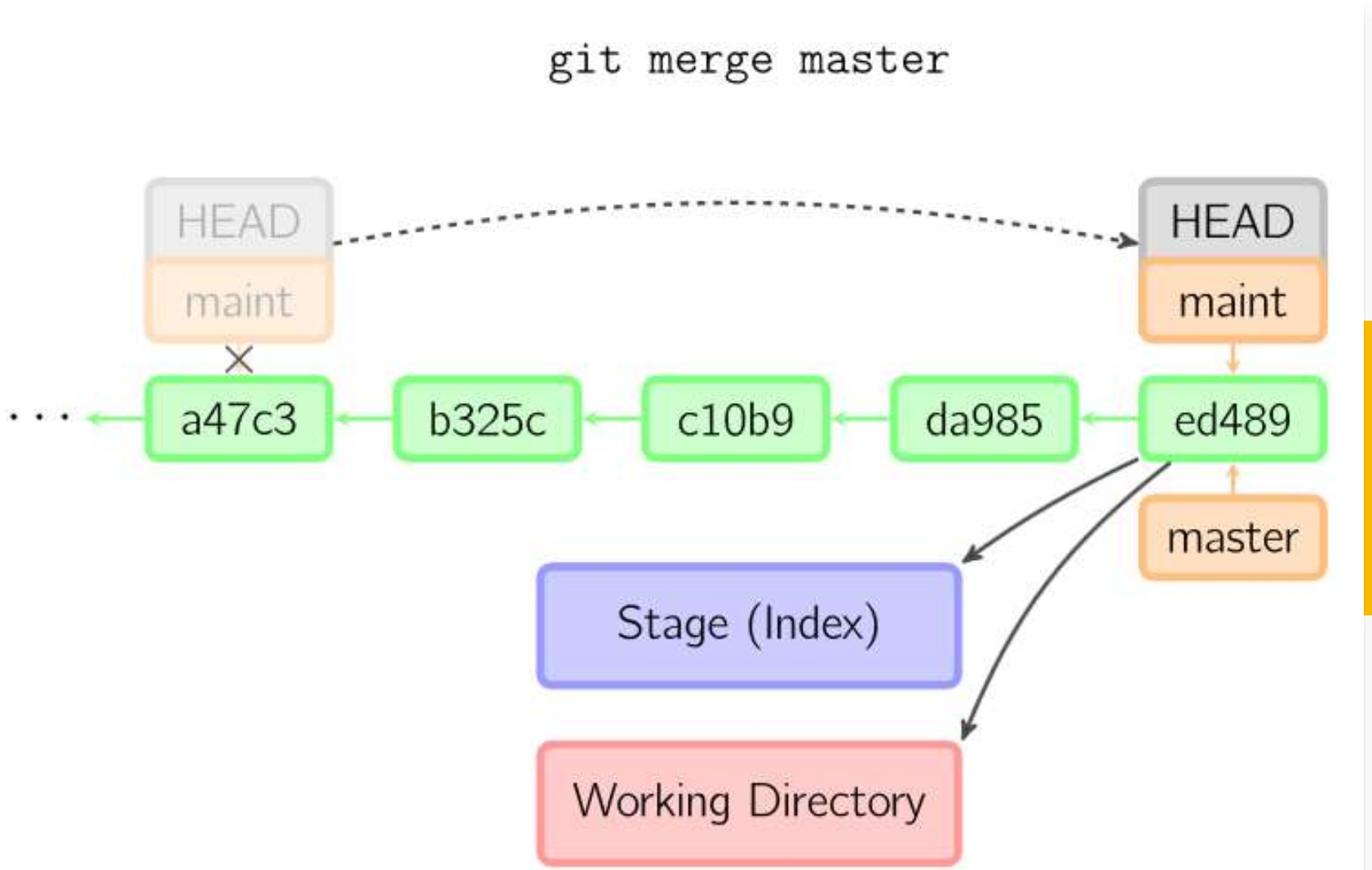
reset destination


git reset
HEAD
maint
master
...
a47c3
b325c
c10b9
da985
ed489
Stage (Index)
(if --hard)
Working Directory

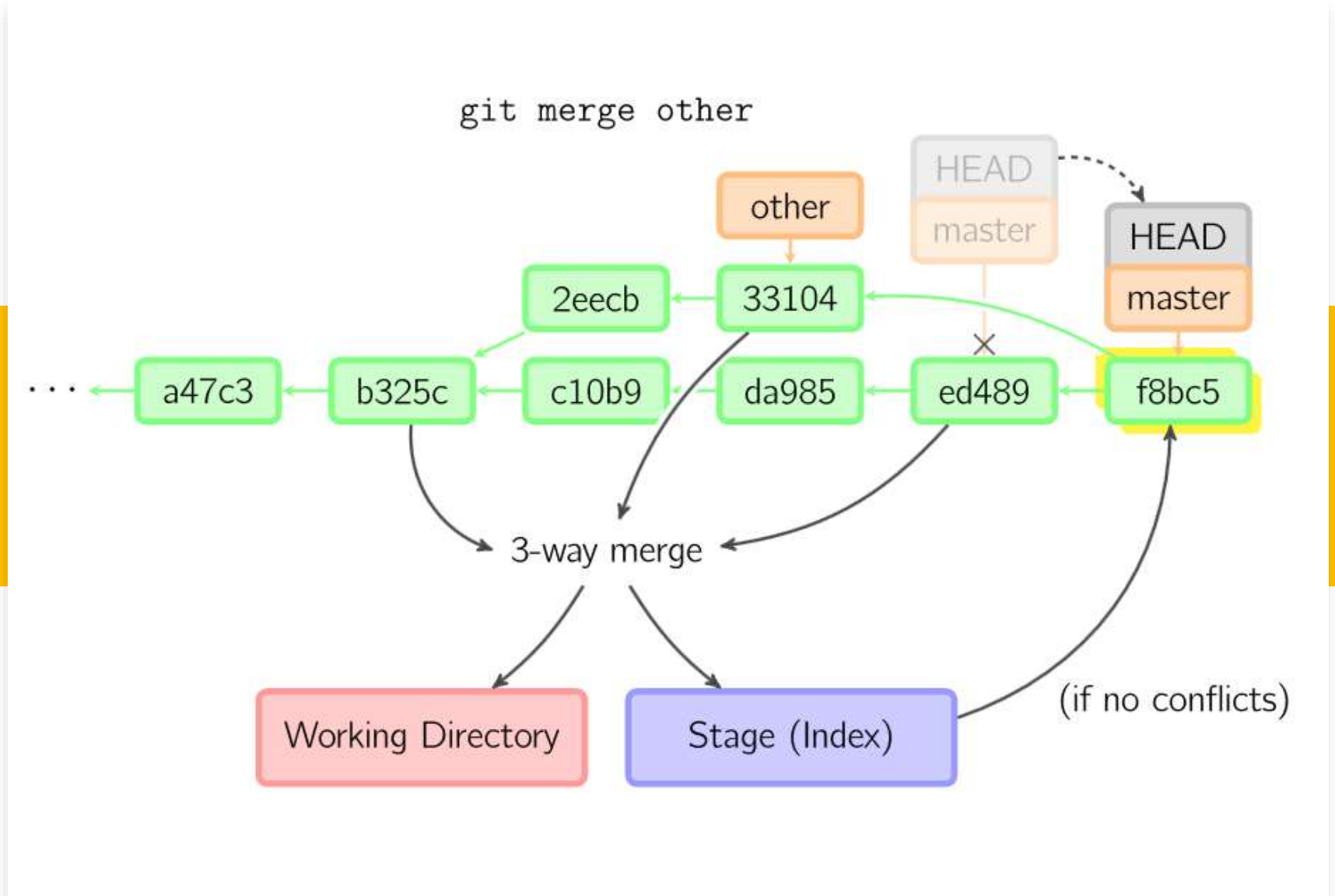
reset



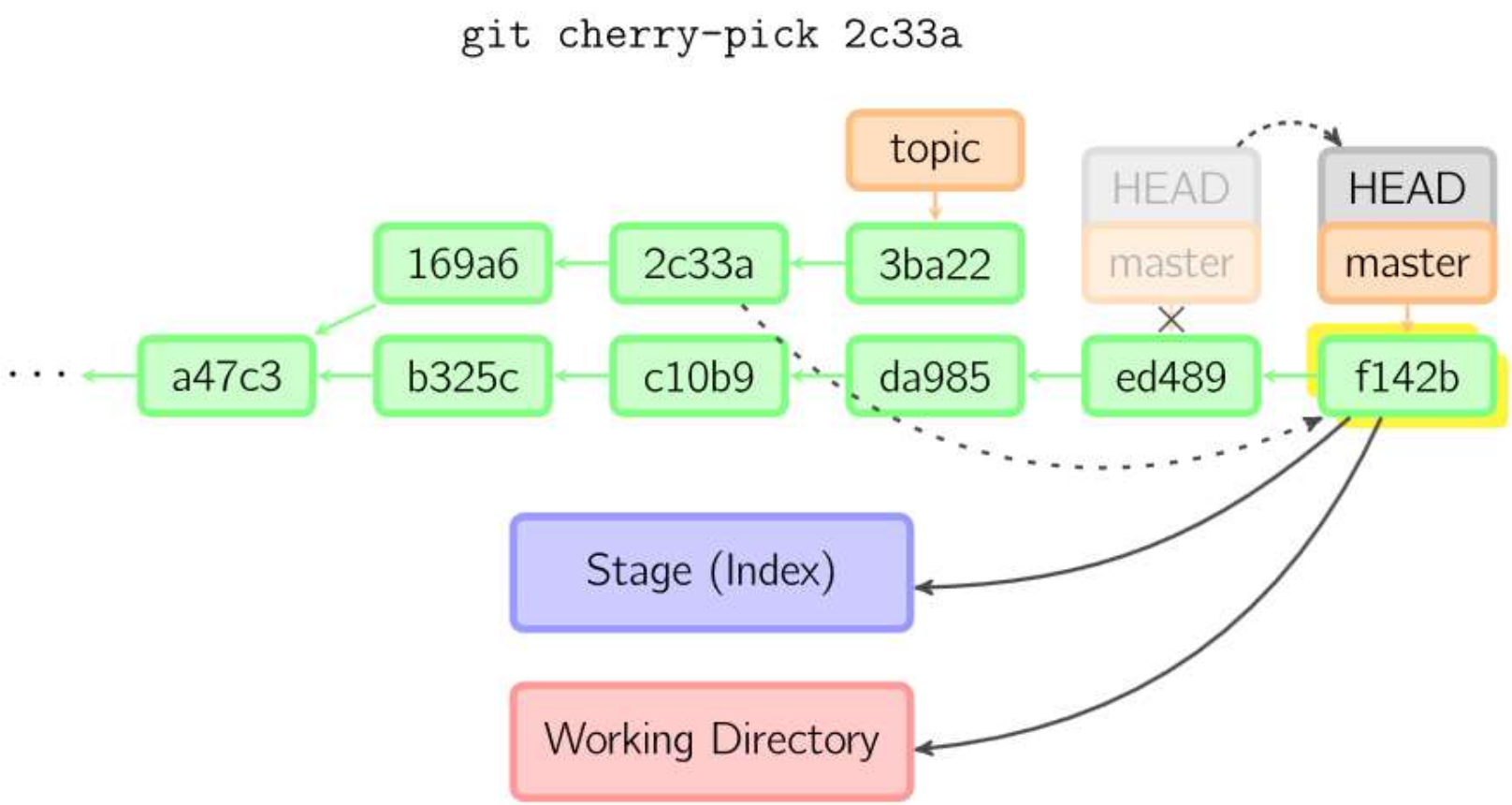
reset fichier


git merge master
HEAD
maint
HEAD
maint
a47c3
b325c
c10b9
da985
ed489
master
Stage (Index)
Working Directory

merge

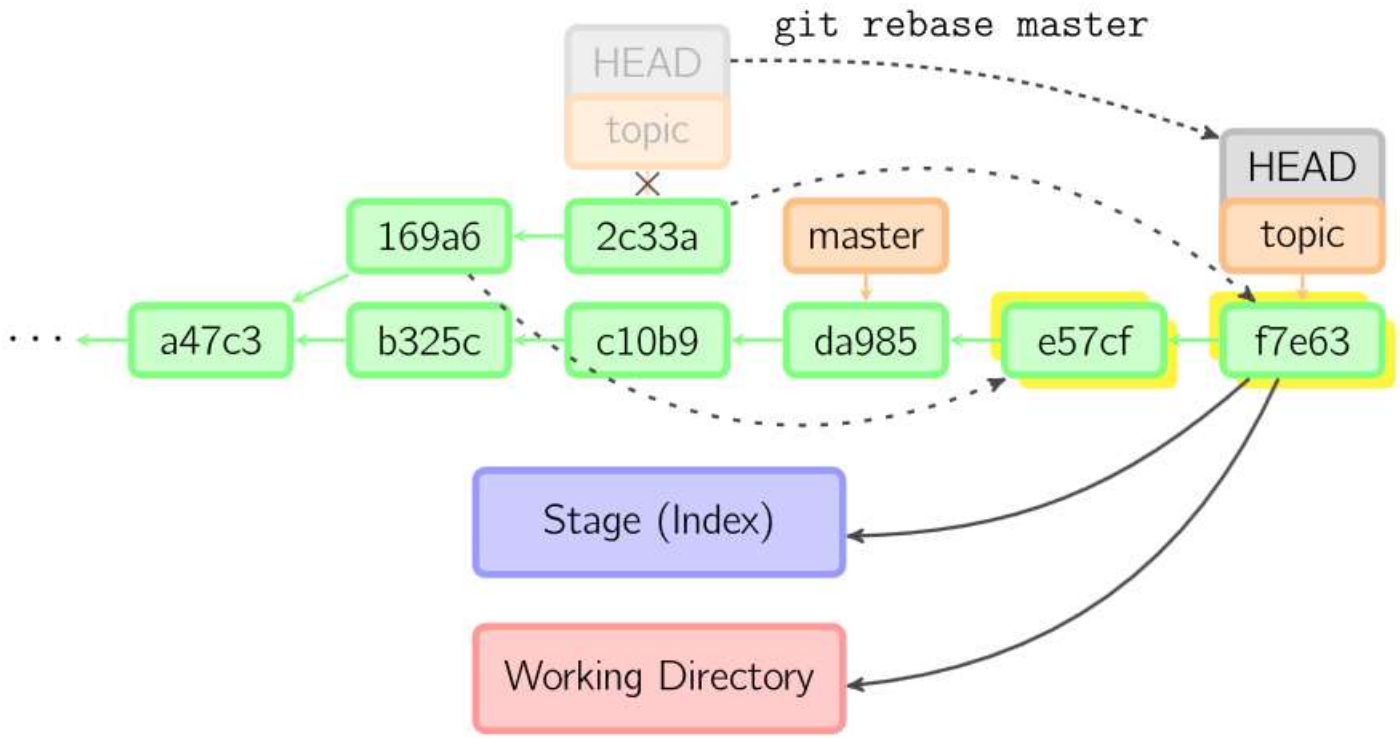


merge (3 way merge)


git cherry-pick 2c33a
topic
HEAD
master
HEAD
master
169a6
2c33a
3ba22
a47c3
b325c
c10b9
da985
ed489
f142b
Stage (Index)
Working Directory

cherry-pick

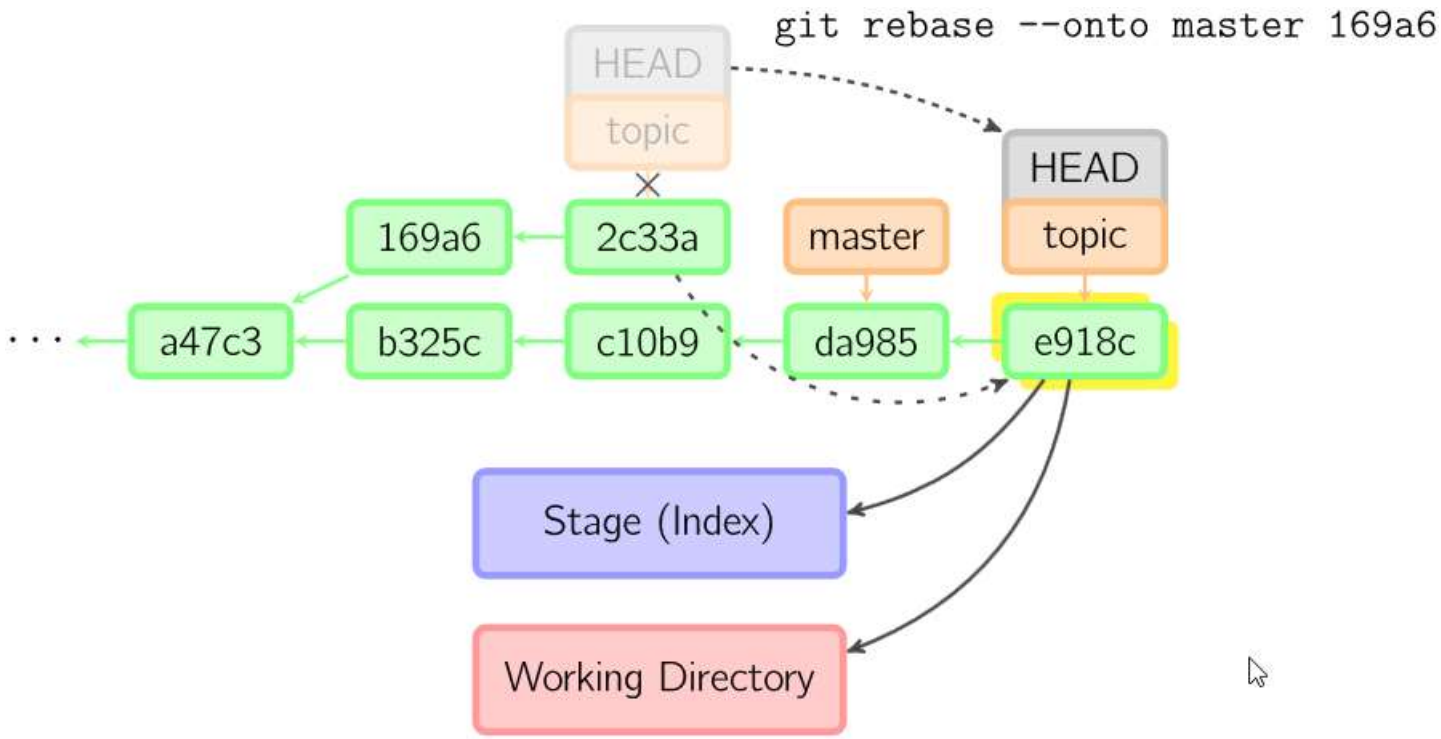
rebase



The golden rule of rebase
“No one shall rebase a shared branch”

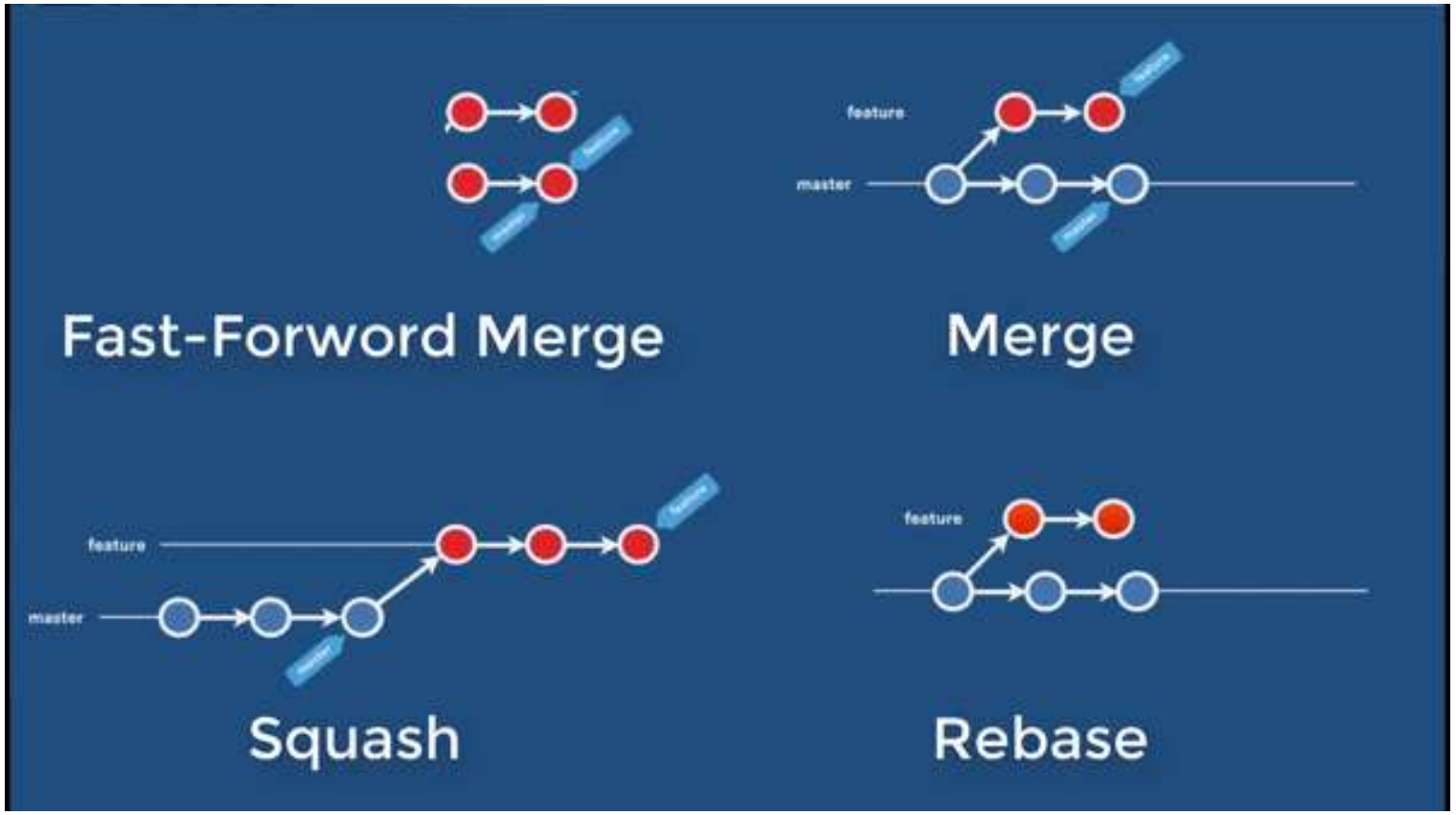
Elies Jebri : elies.jebri@gmail.com

rebase --onto



Elies Jebri : elies.jebri@gmail.com

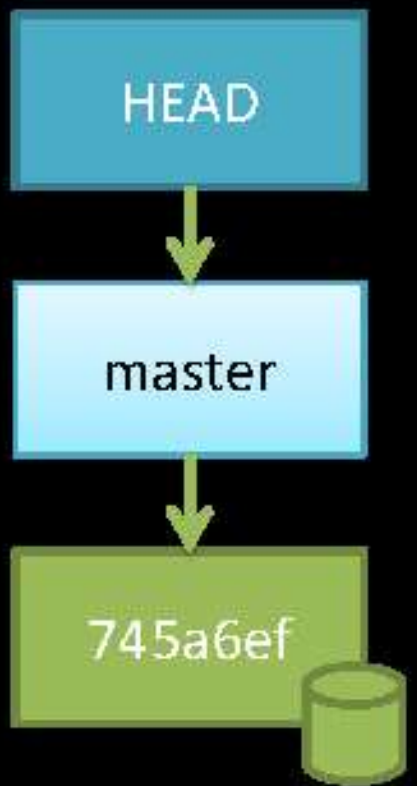
Fusion des branches



Step by Step

Git

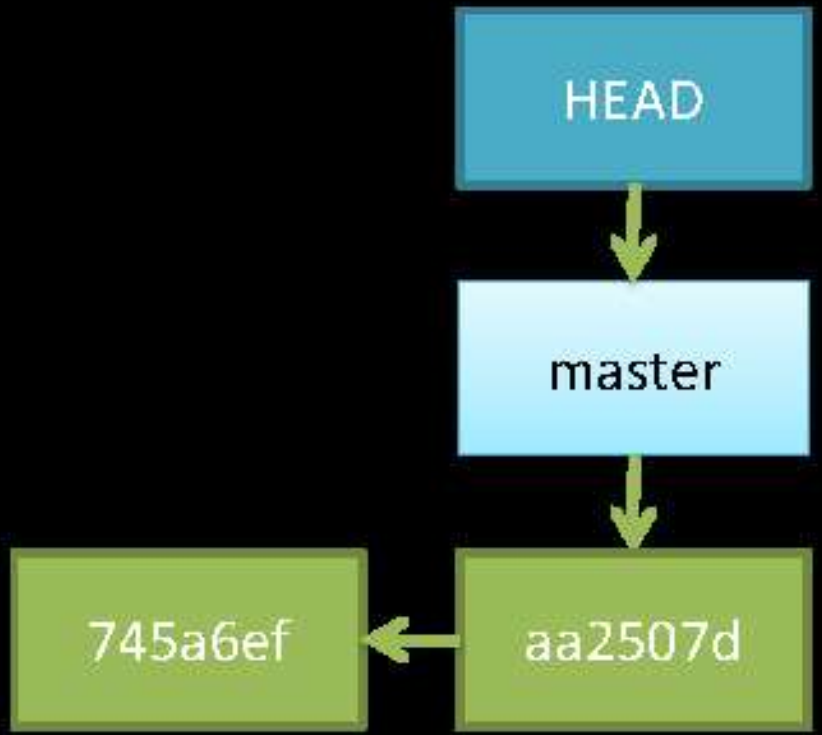
```
$ git init
$ git add file.txt
$ git commit -m "Initial Commit"
[master (root-commit) 745a6ef] Initial Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
```



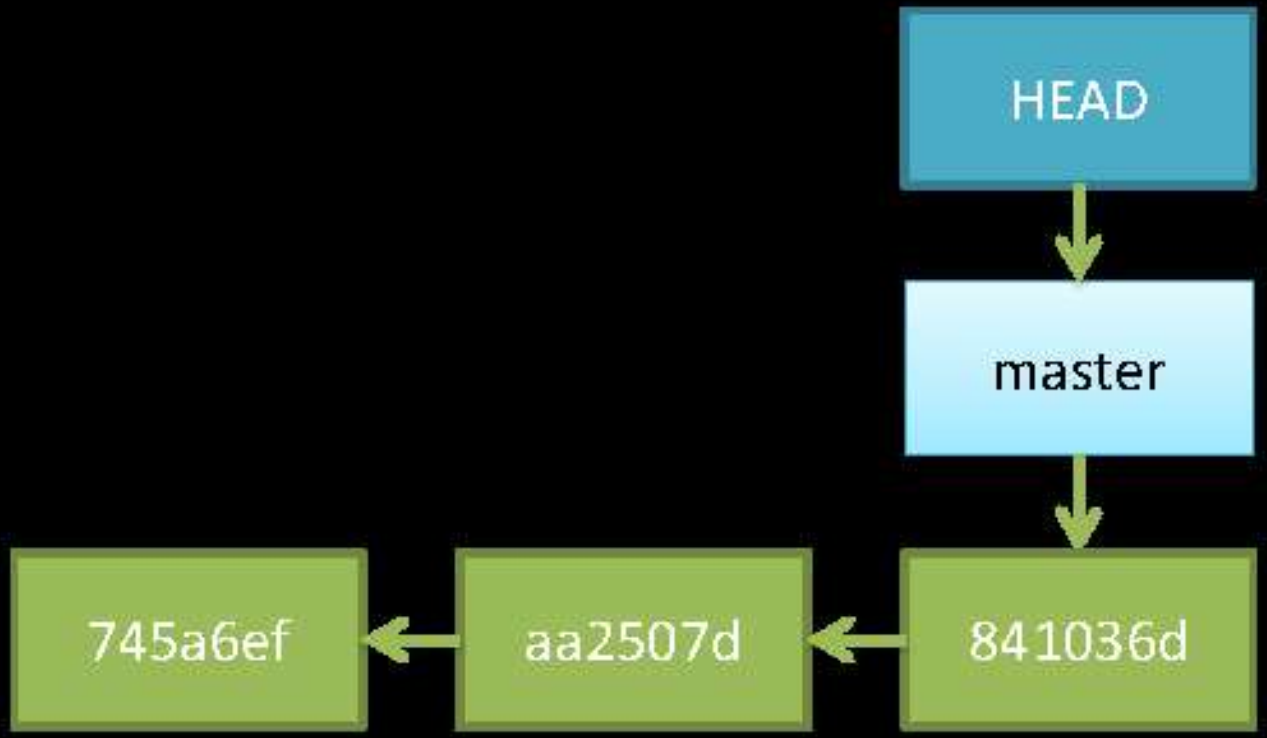
Pointeur de working directory

Branche

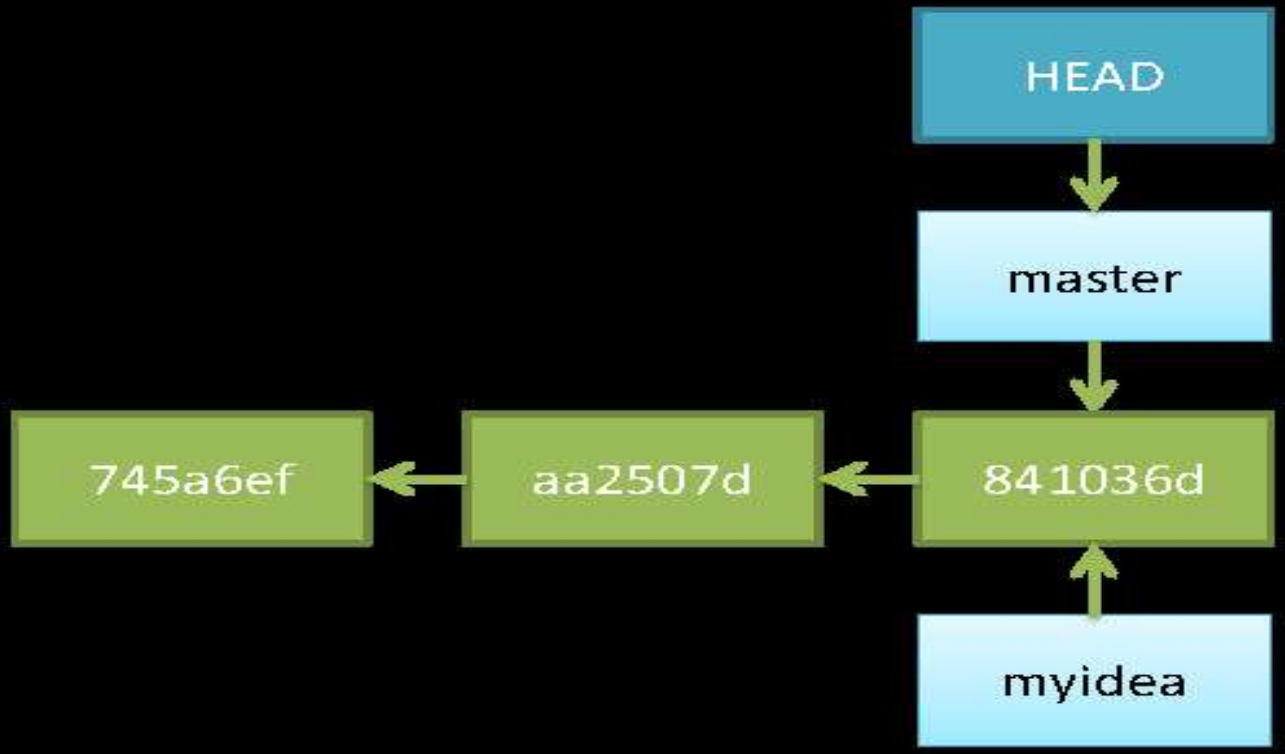
```
$ git add file2.txt file3.txt
$ git commit -m "Added file2 file3"
[master aa2507d] Added file2 file3
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt
 create mode 100644 file3.txt
```



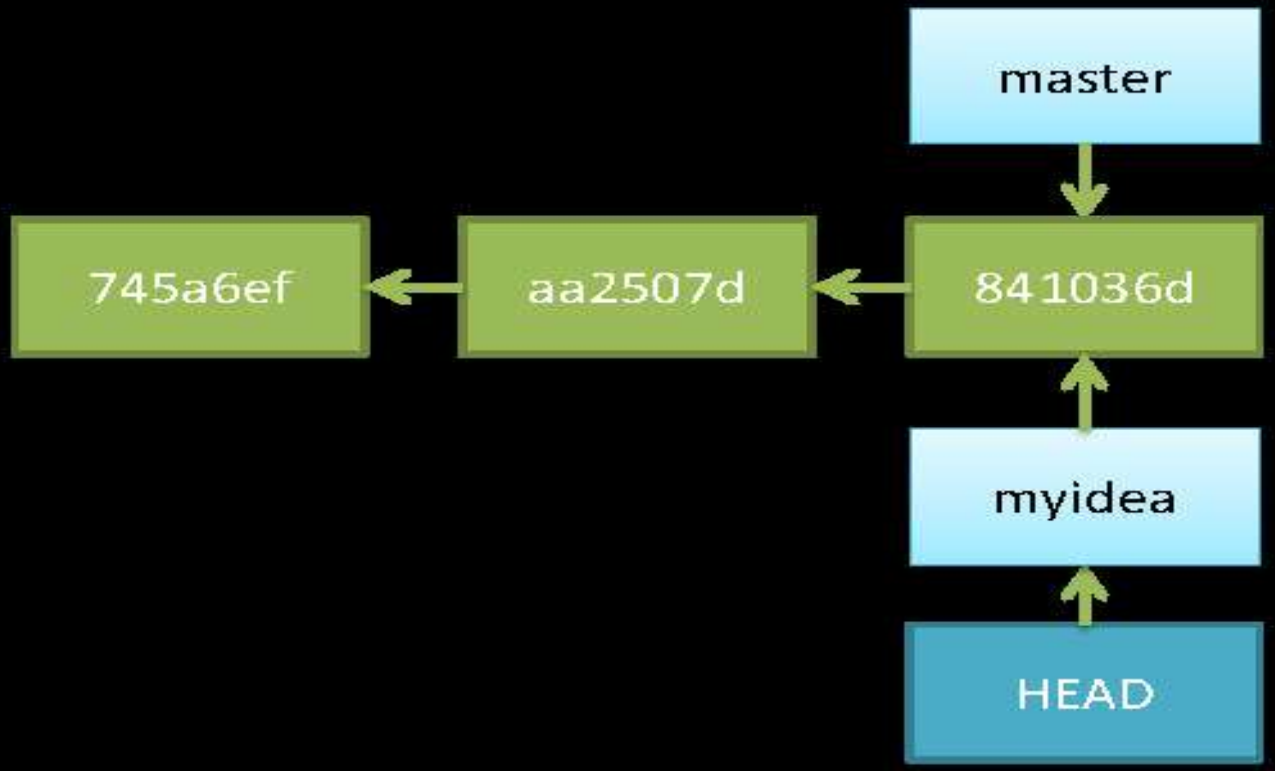
```
$ notepad file2.txt
$ git commit -a -m "Changed file2"
[master 841036d] Changed file2
 1 files changed, 2 insertions(+), 1 deletions(-)
```



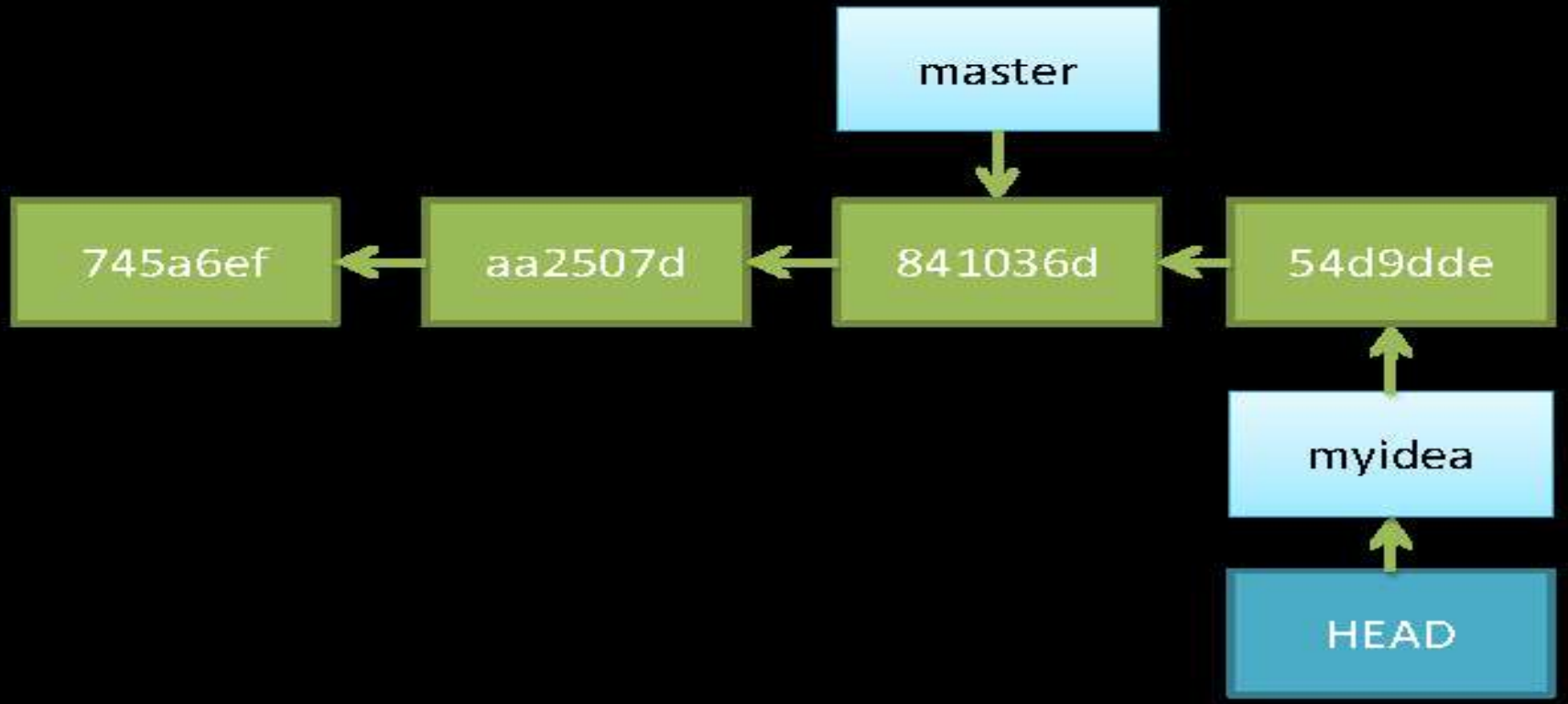
```
$ git branch myidea
```



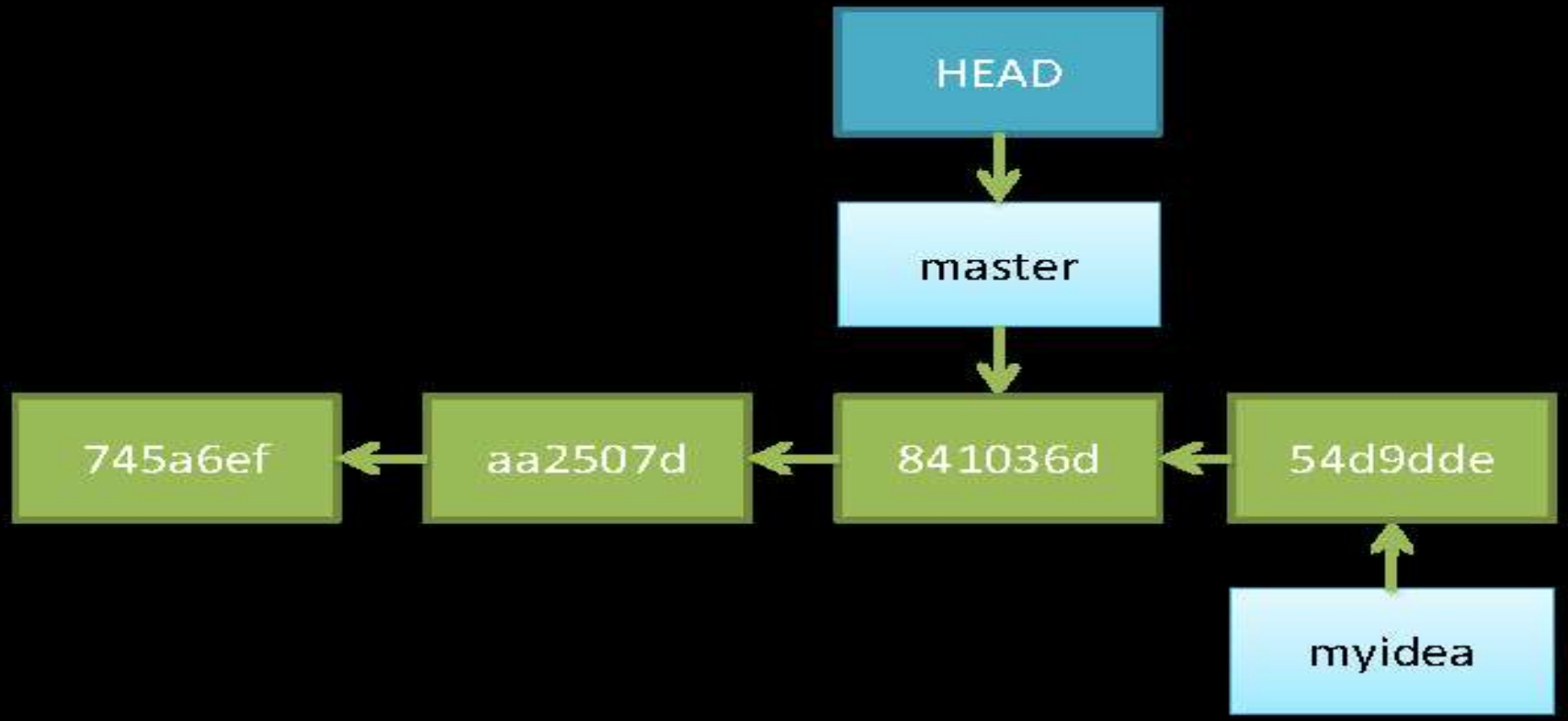
```
$ git checkout myidea
Switched to branch 'myidea'
```



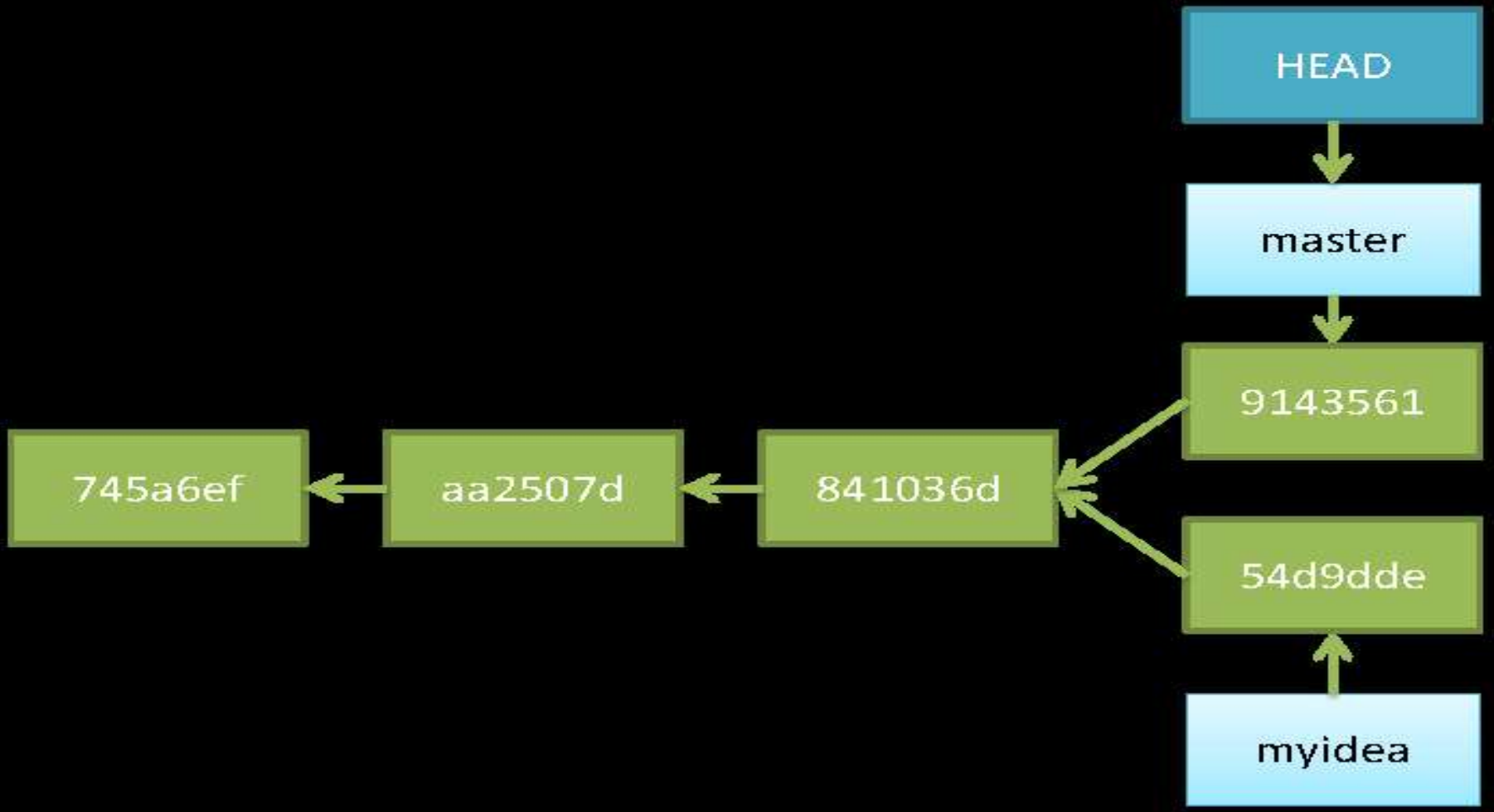
```
$ notepad file3.txt
$ git commit -a -m "Changed file3"
[myidea 54d9dde] Changed file3
 1 files changed, 2 insertions(+), 1 deletions(-)
```



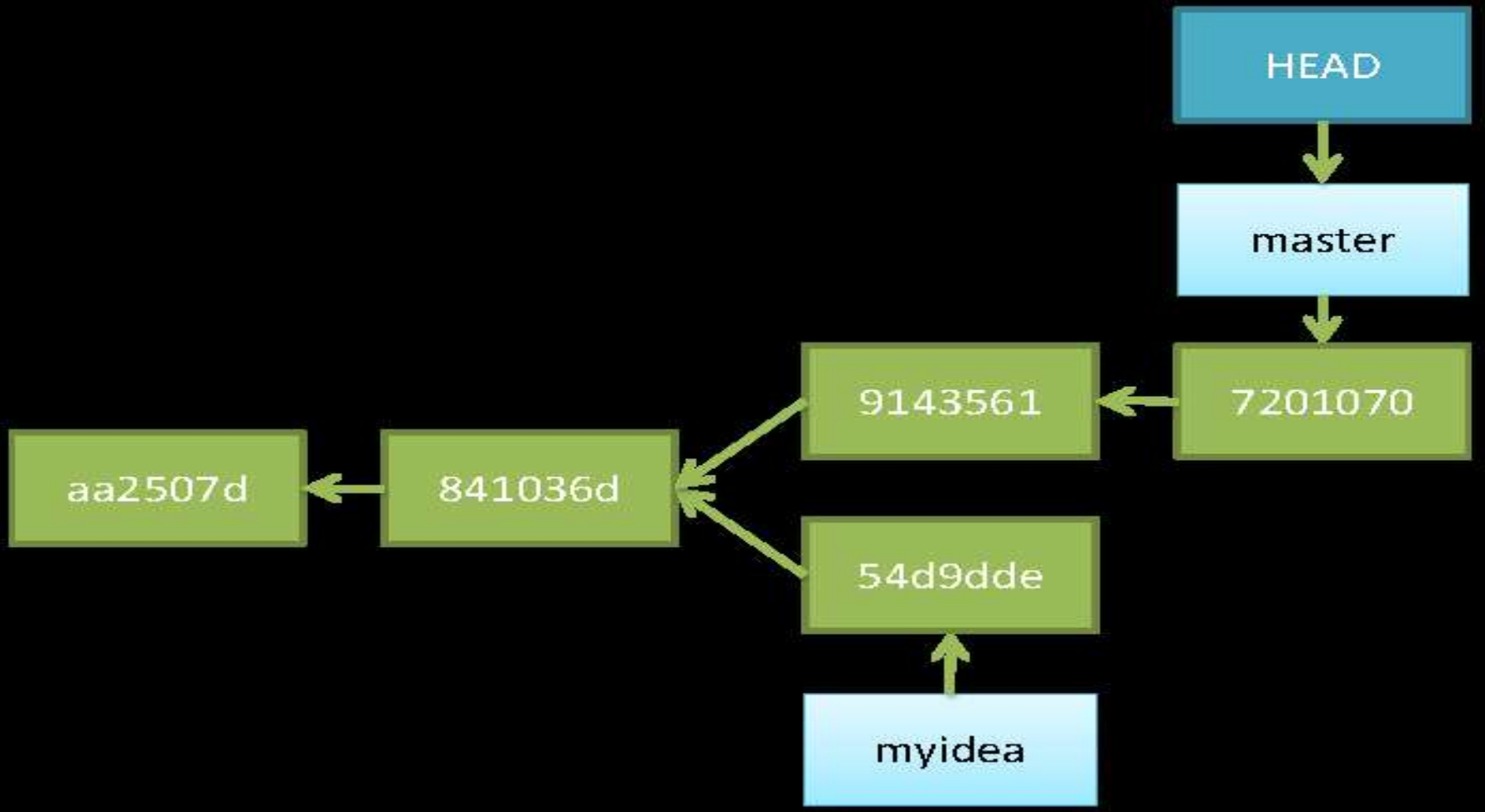
```
$ git checkout master
Switched to branch 'master'
```



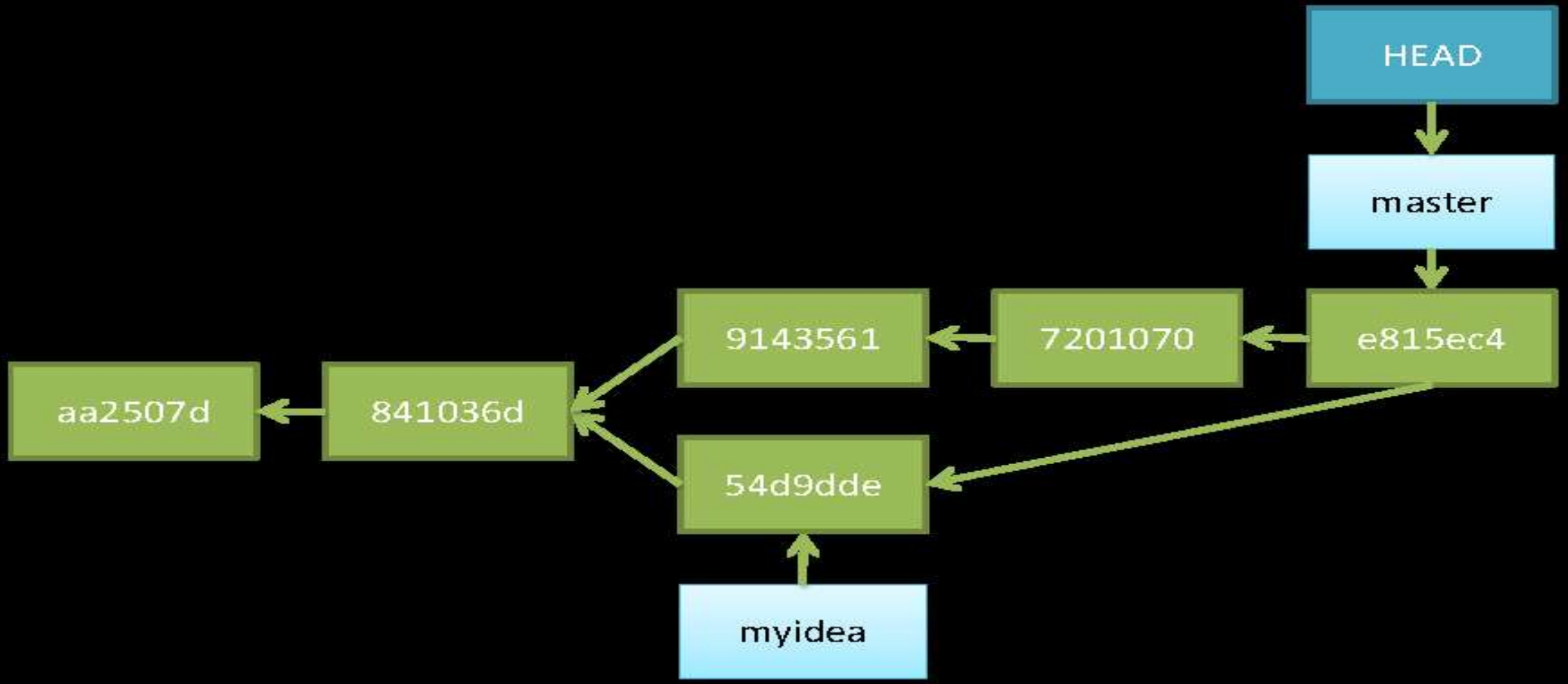
```
$ notepad file2.txt
$ git commit -a -m "Changed file2"
[master 9143561] Changed file2
 1 files changed, 2 insertions(+), 1 deletions(-)
```



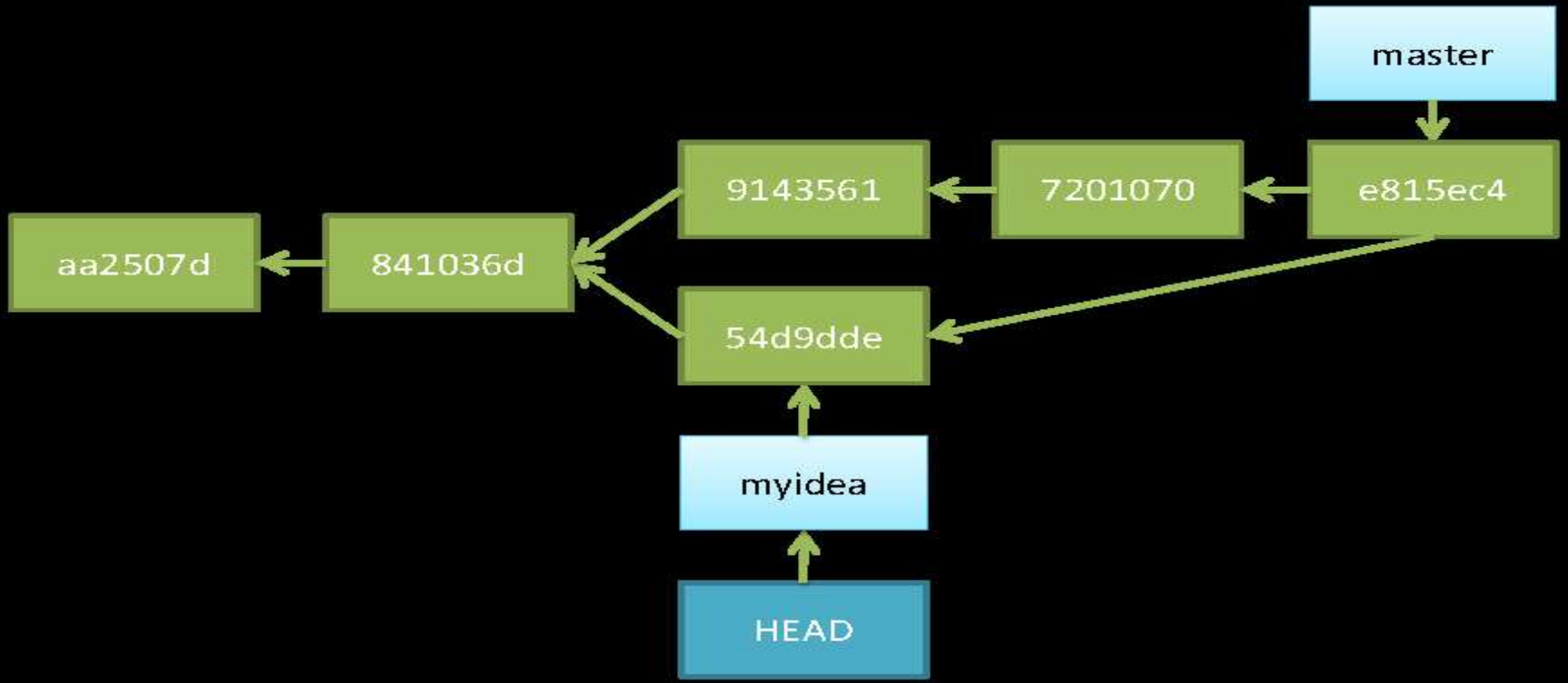
```
$ git add file4.txt
$ git commit -m "Added file4"
[master 7201070] Added file4
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 file4.txt
```



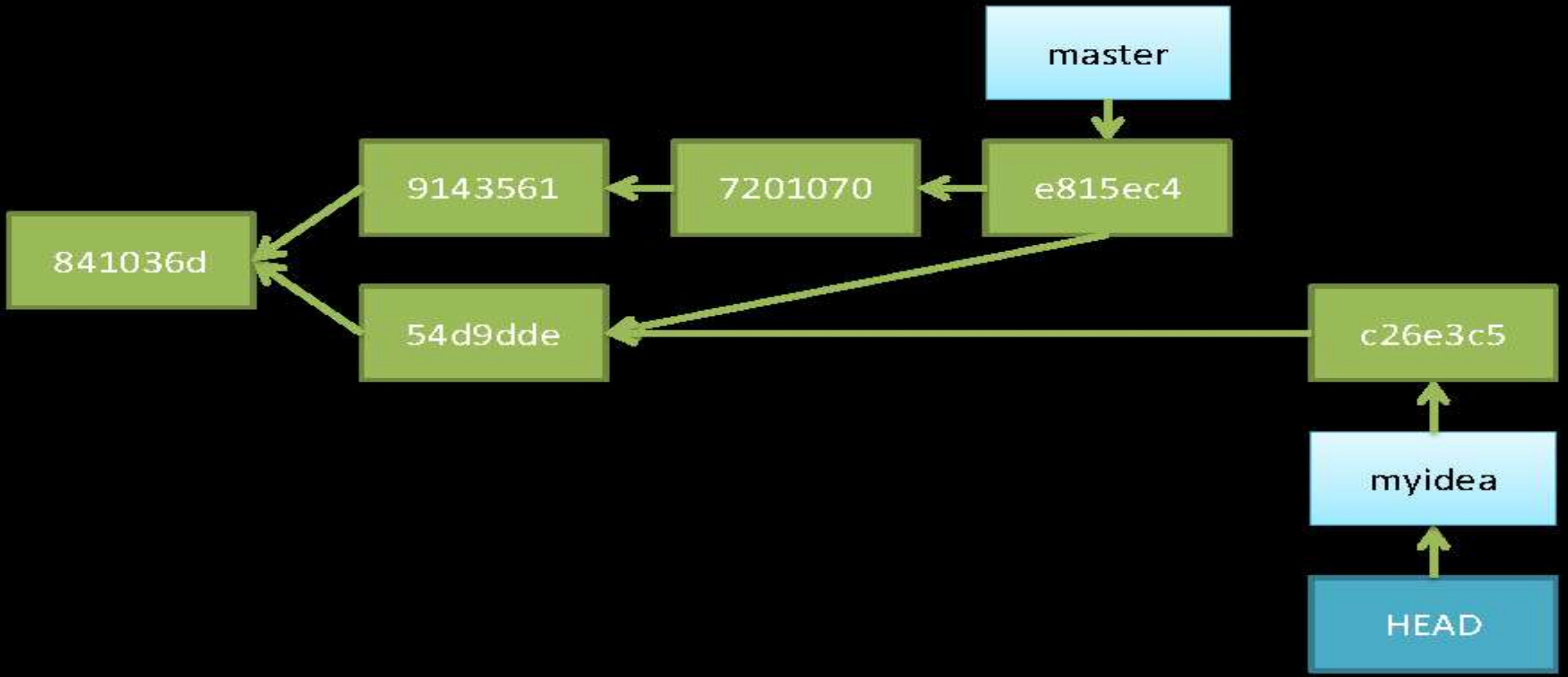
```
$ git merge myidea
Merge made by recursive.
 file3.txt |    3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)
```



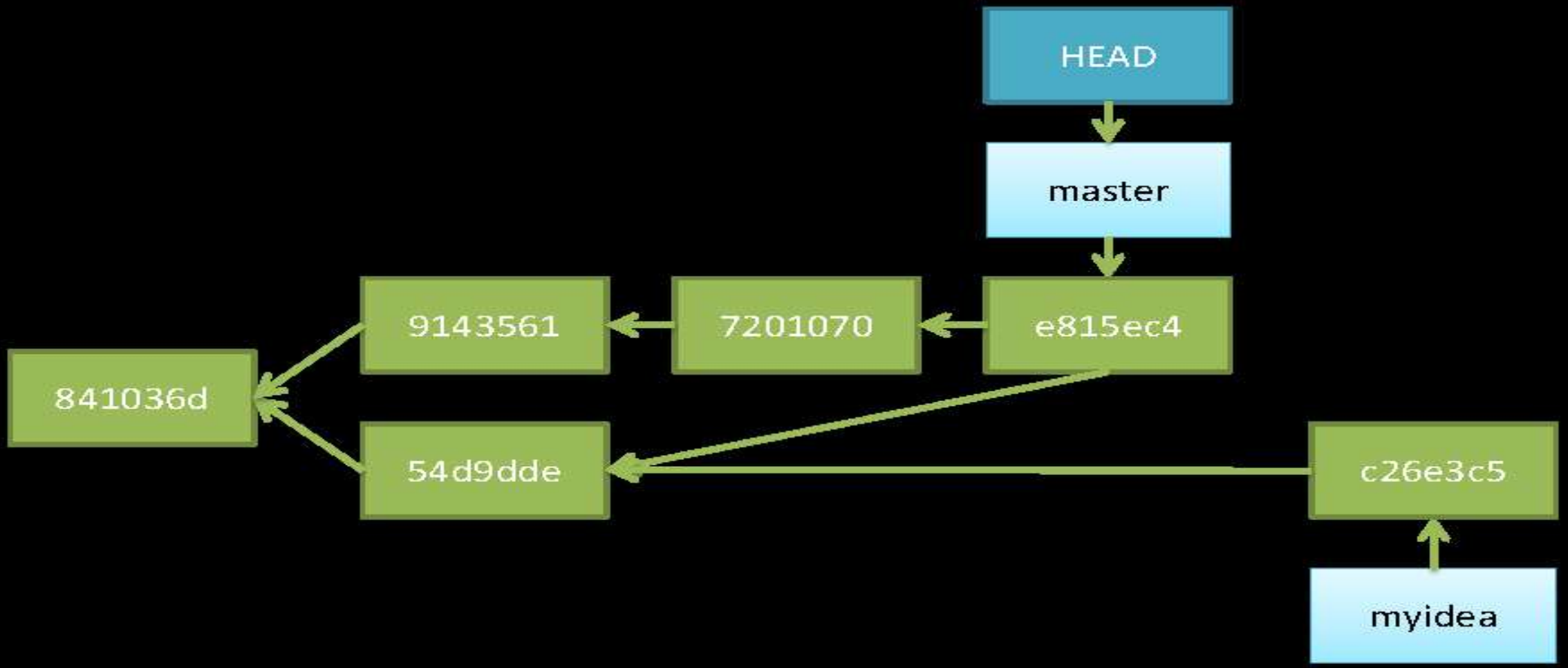
```
$ git checkout myidea
Switched to branch 'myidea'
```



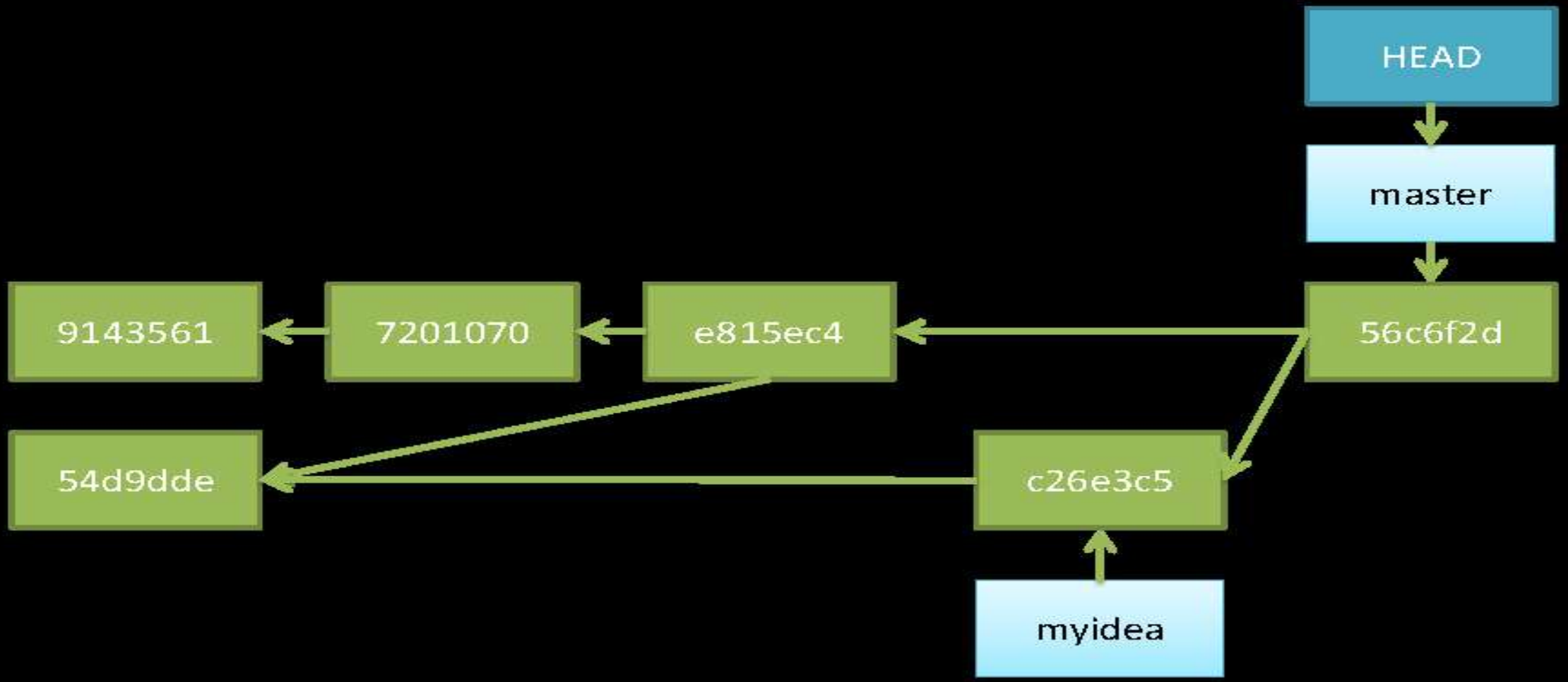
```
$ notepad file.txt
$ git commit -a -m "Changed file.txt«
[myidea c26e3c5] Changed file.txt
 1 files changed, 2 insertions(+), 1 deletions(-)
```



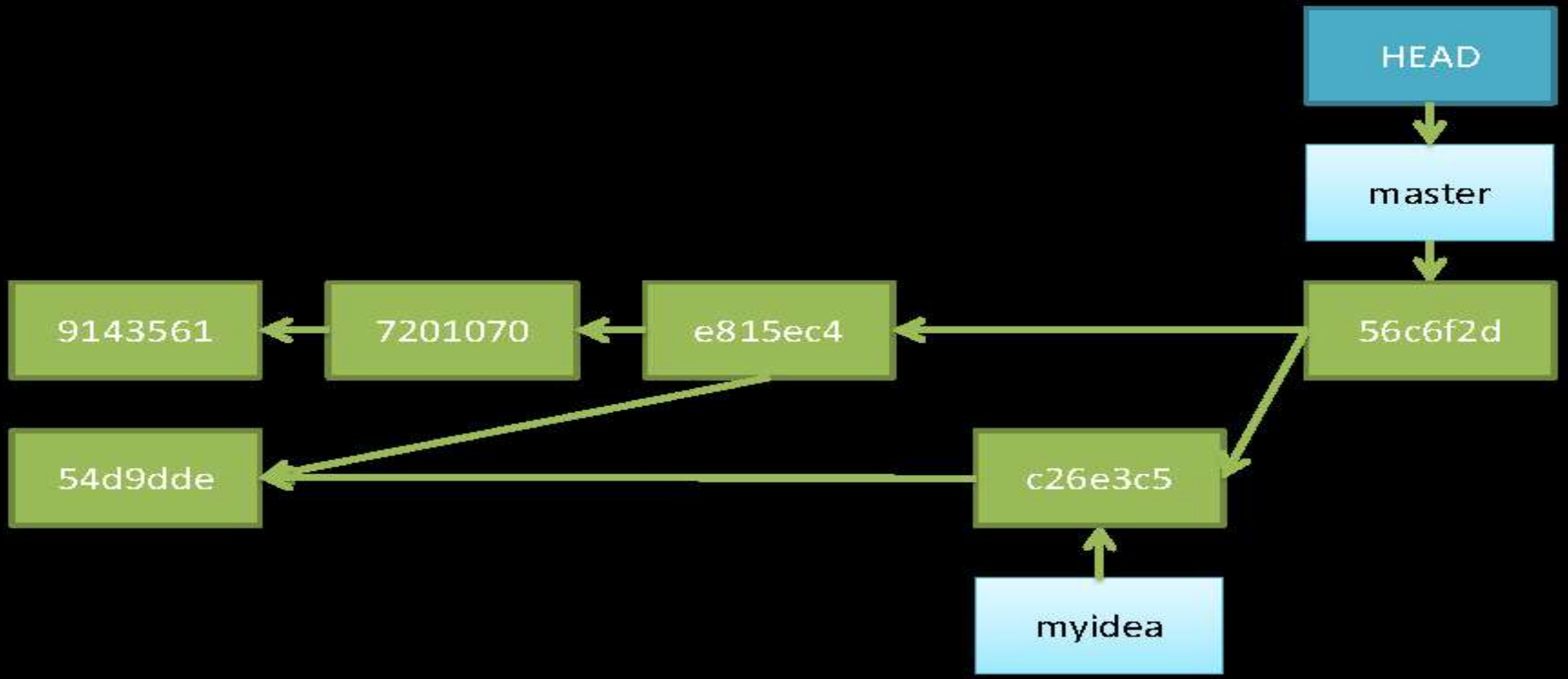
```
$ git checkout master
Switched to branch 'master'
```



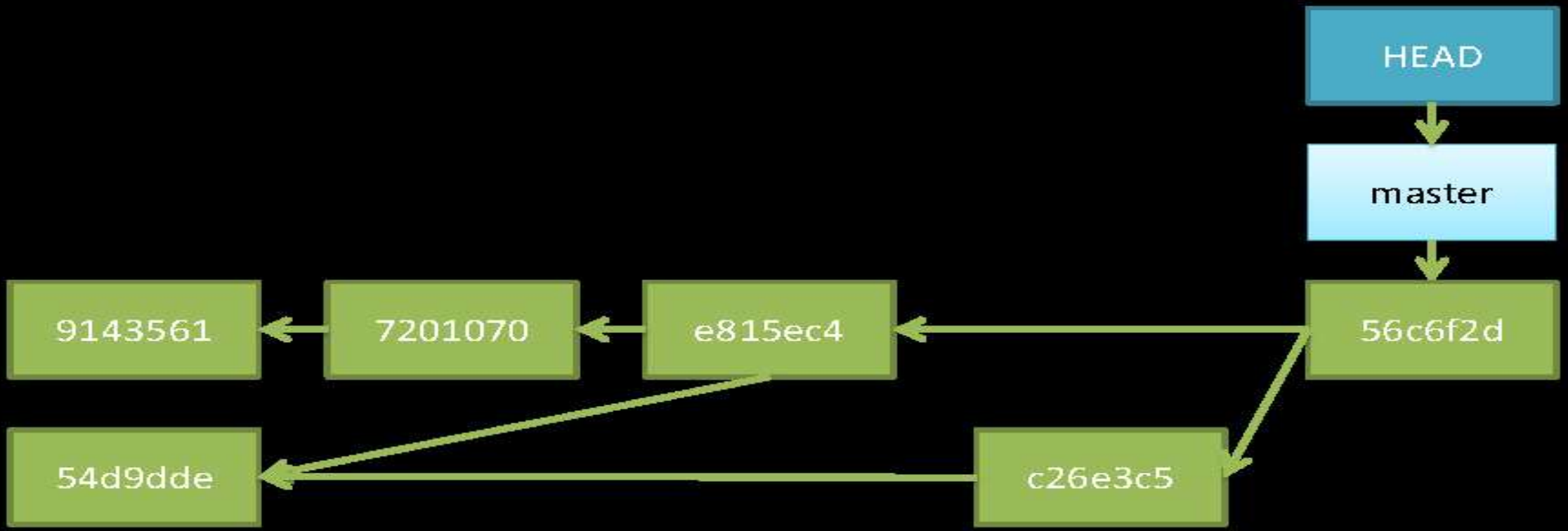
```
$ git merge myidea
Merge made by recursive.
 file.txt |    3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)
```

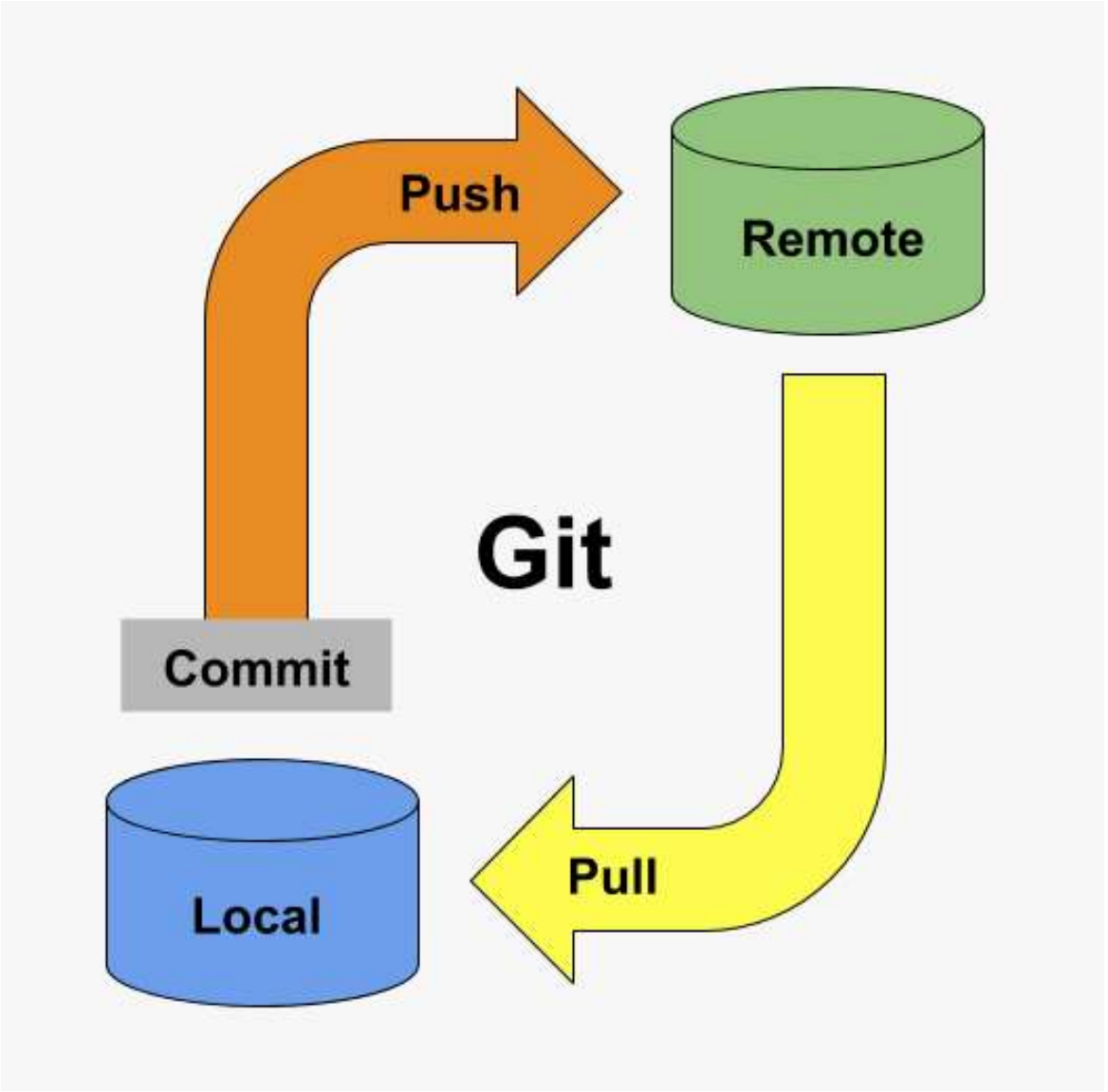


```
$ git merge myidea
Already up-to-date.
$ git merge myidea
Already up-to-date.
```



```
$ git branch -d myidea
Deleted branch myidea (was c26e3c5).
```





Branches de suivi à distance

Styles de développement

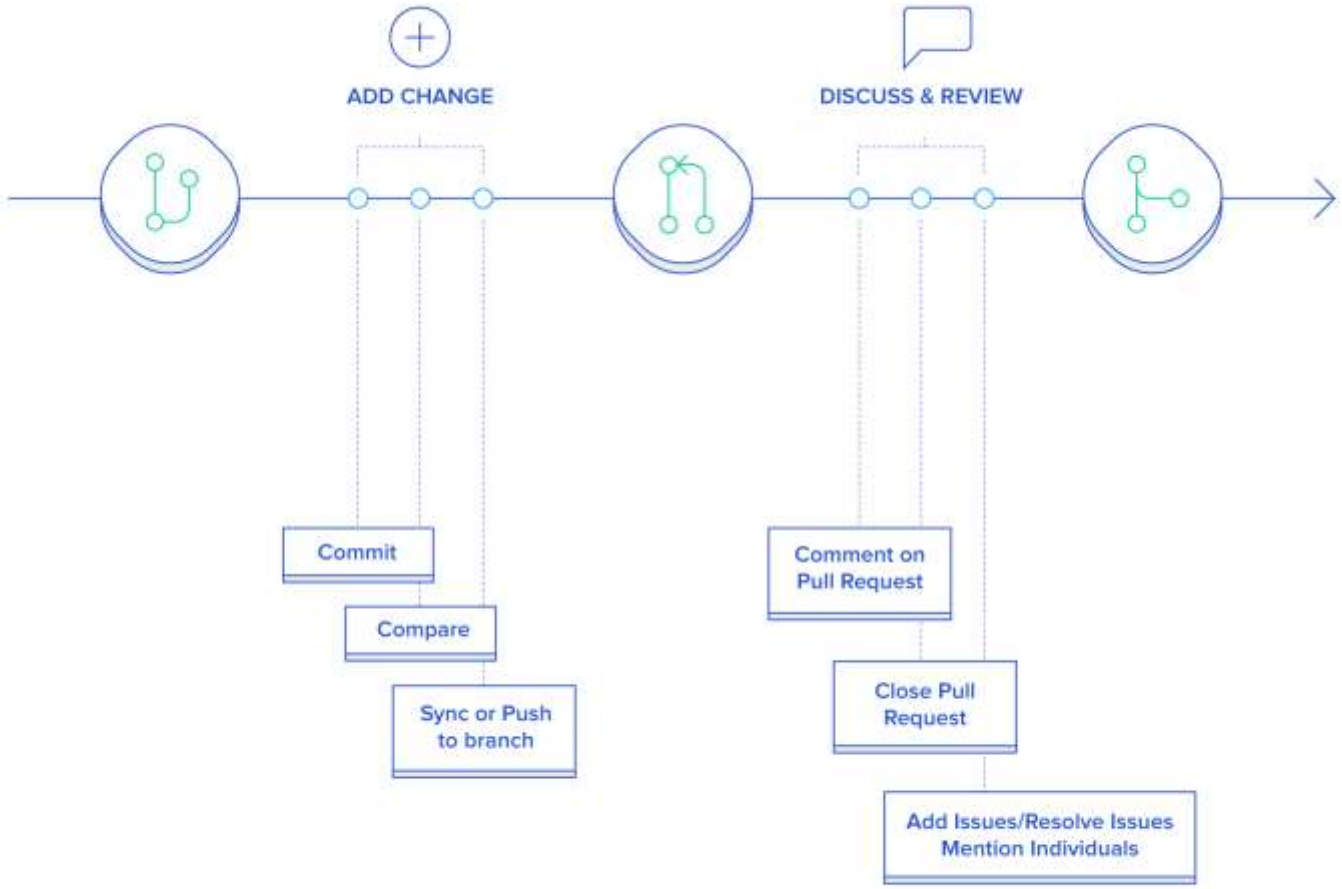
- En général, un développeur effectue des demandes d'extraction (Pull Request) pour combiner les modifications qu'il a créées avec le projet principal.
- Un processus d'examen de ces changements est lancé par les réviseurs qui peuvent insérer des commentaires sur chaque élément qu'ils pensent pouvoir être amélioré ou considérer comme inutile.

- Après avoir reçu des commentaires, le créateur peut y répondre, créer une discussion, ou simplement la suivre et modifier son code en conséquence.

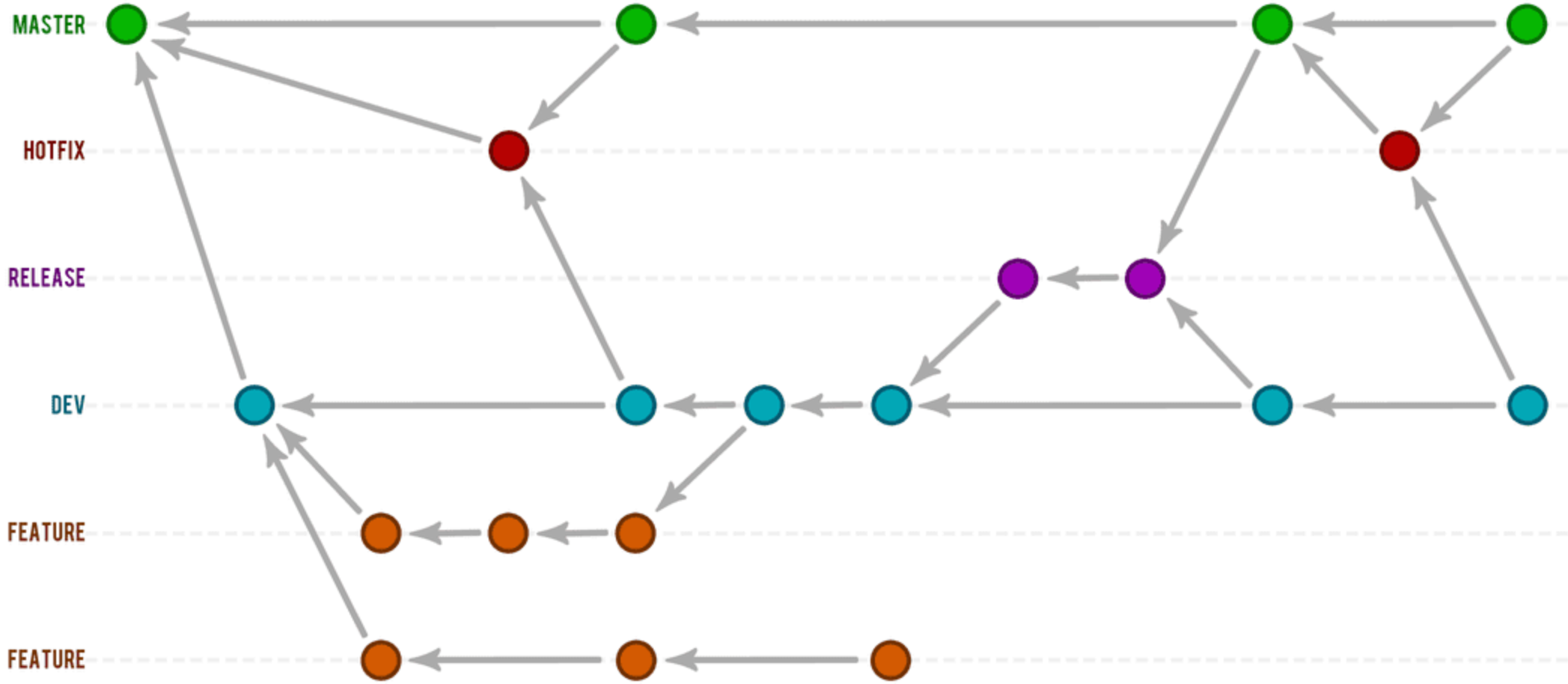
Elies Jebri : elies.jebri@gmail.com

Styles de développement

Actuellement, les deux styles de développement les plus populaires que vous pouvez rencontrer sont Git flow et le Trunk-based Development.

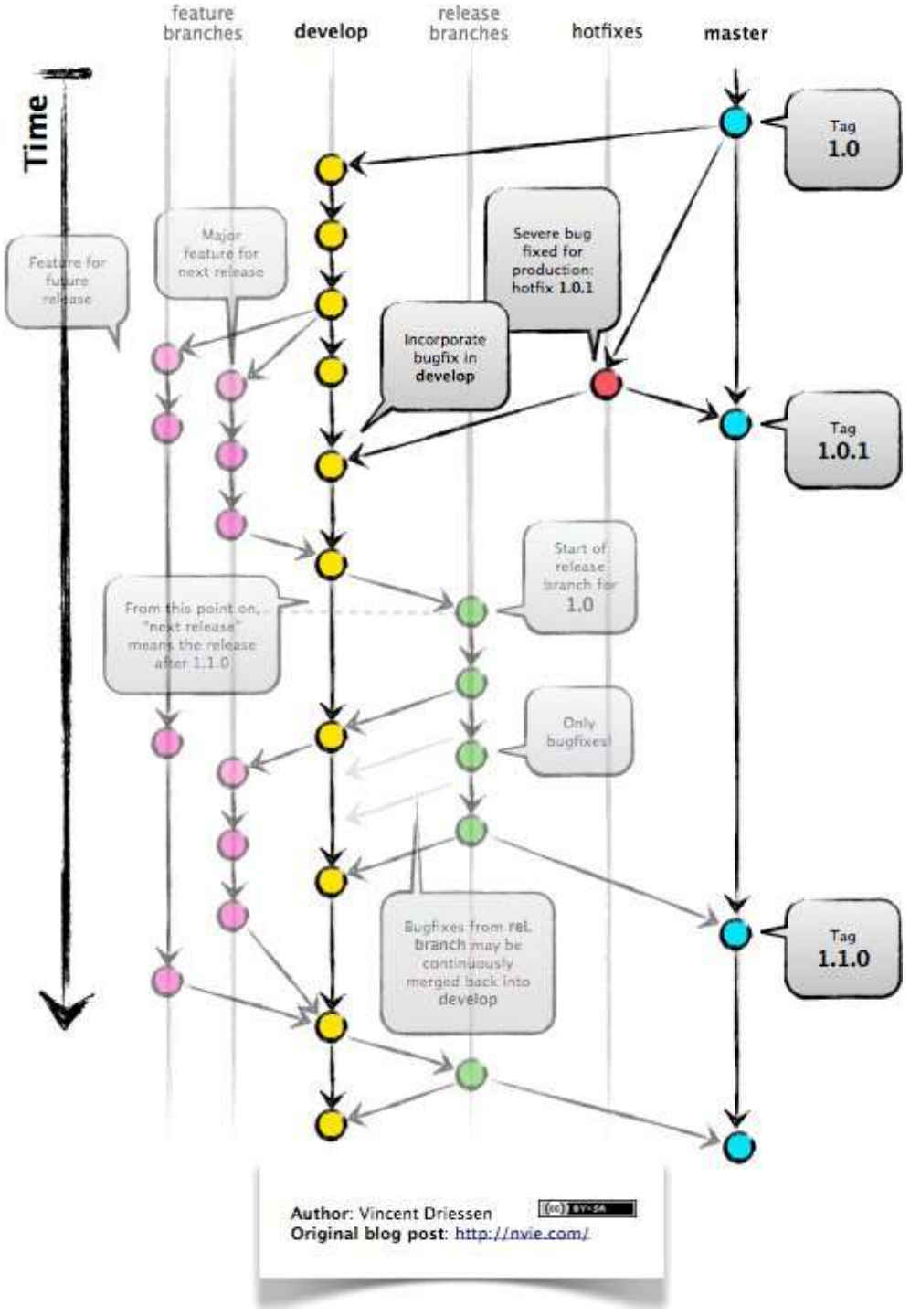


Le modèle GitFlow



Elies Jebri : elies.jebri@gmail.com

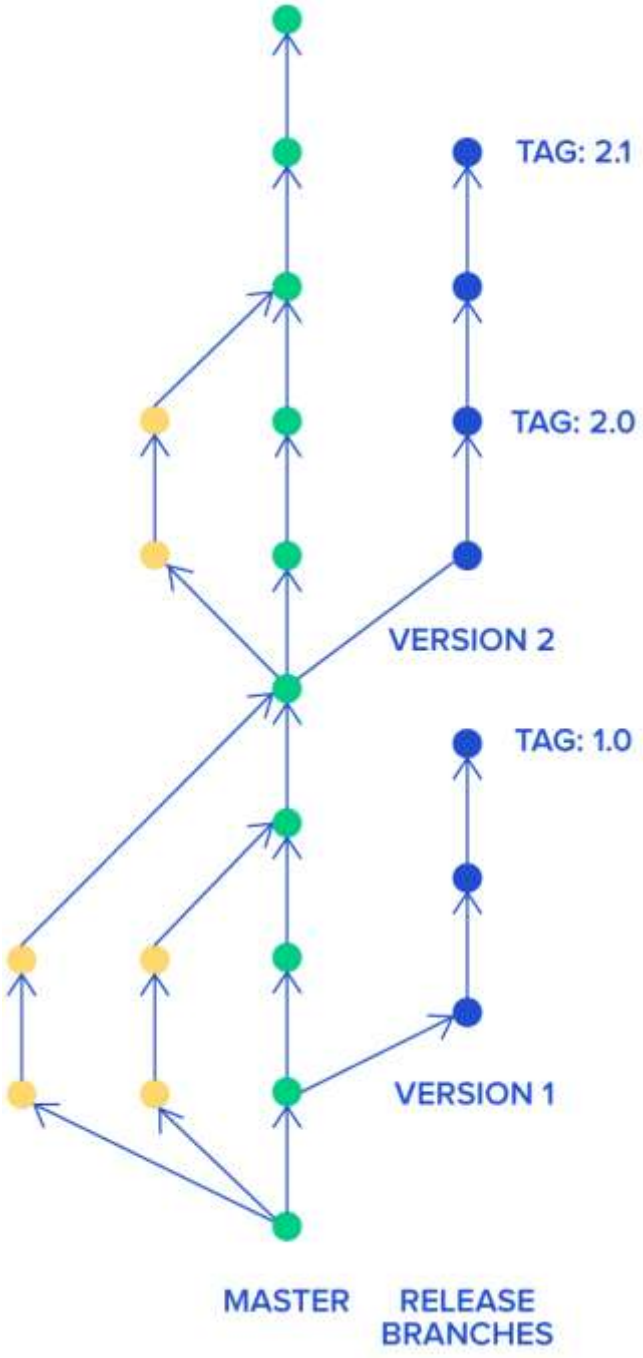
GitFlow exemple



Elies Jebri : elies.jebri@gmail.com

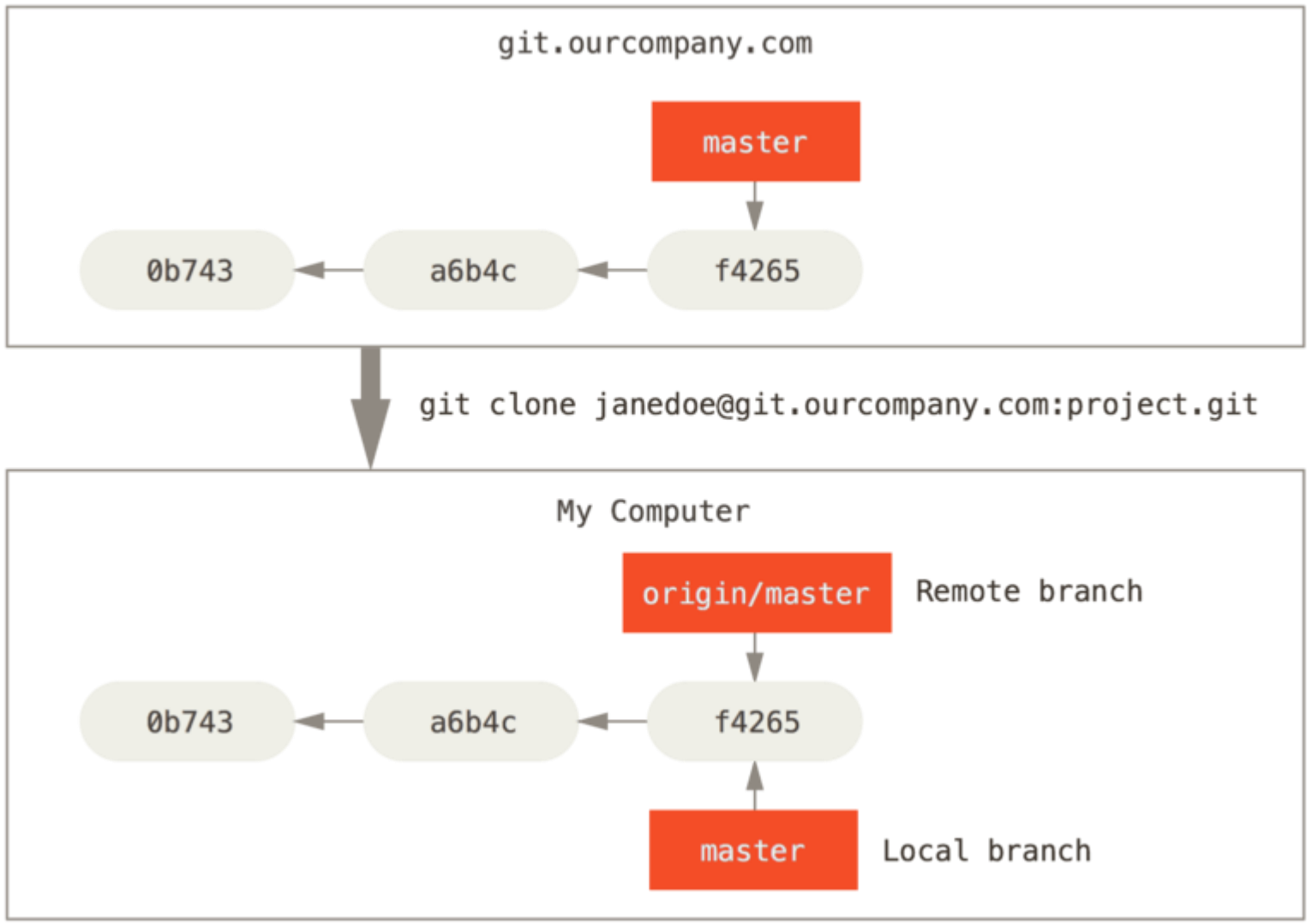
Trunk-based Development Workflow

- tous les développeurs travaillent sur une seule branche avec un accès ouvert à celle-ci. Souvent, c'est simplement la masterbranche. Ils y commettent du code et l'exécutent. C'est ultra simple.

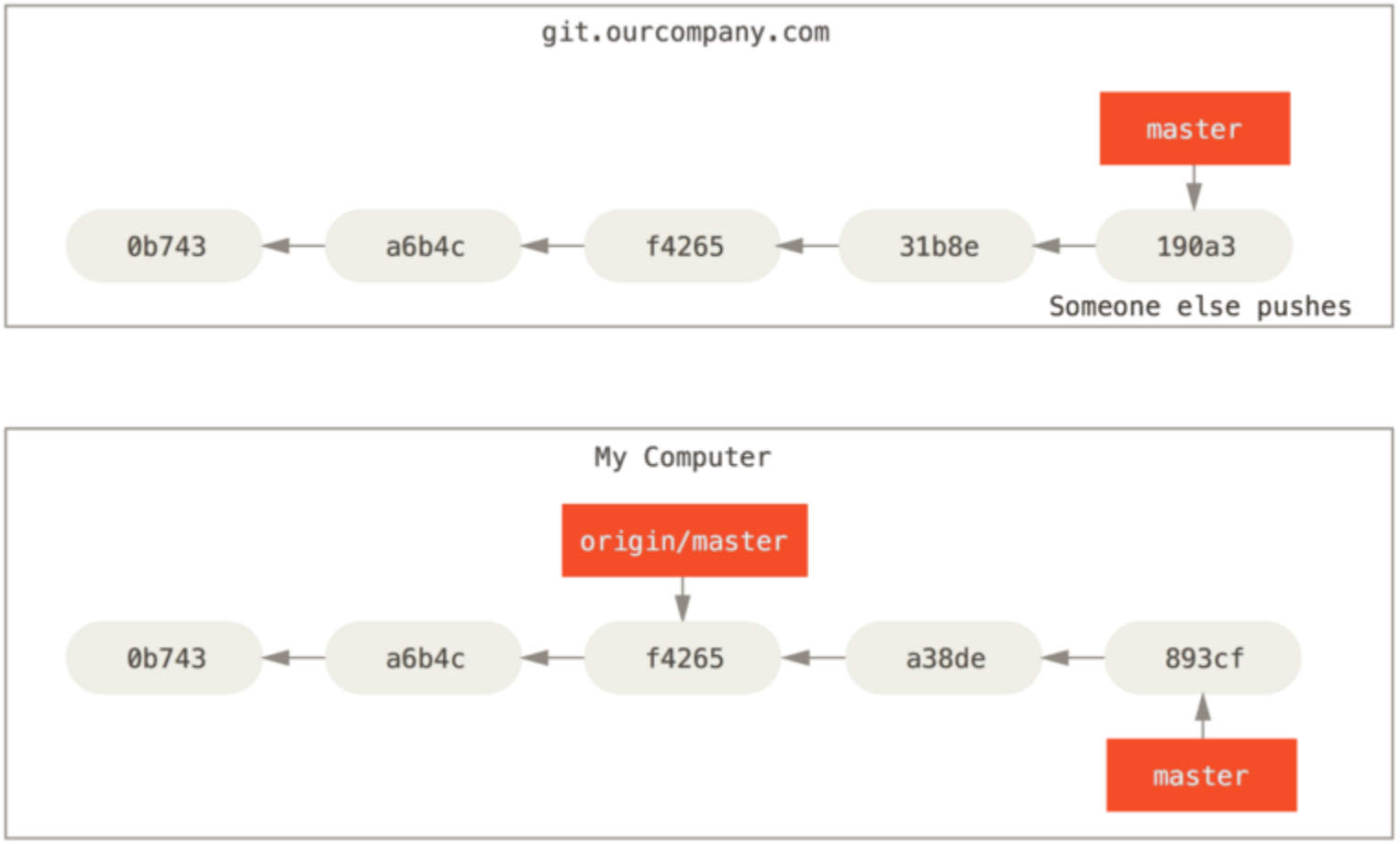


Les commandes en détail

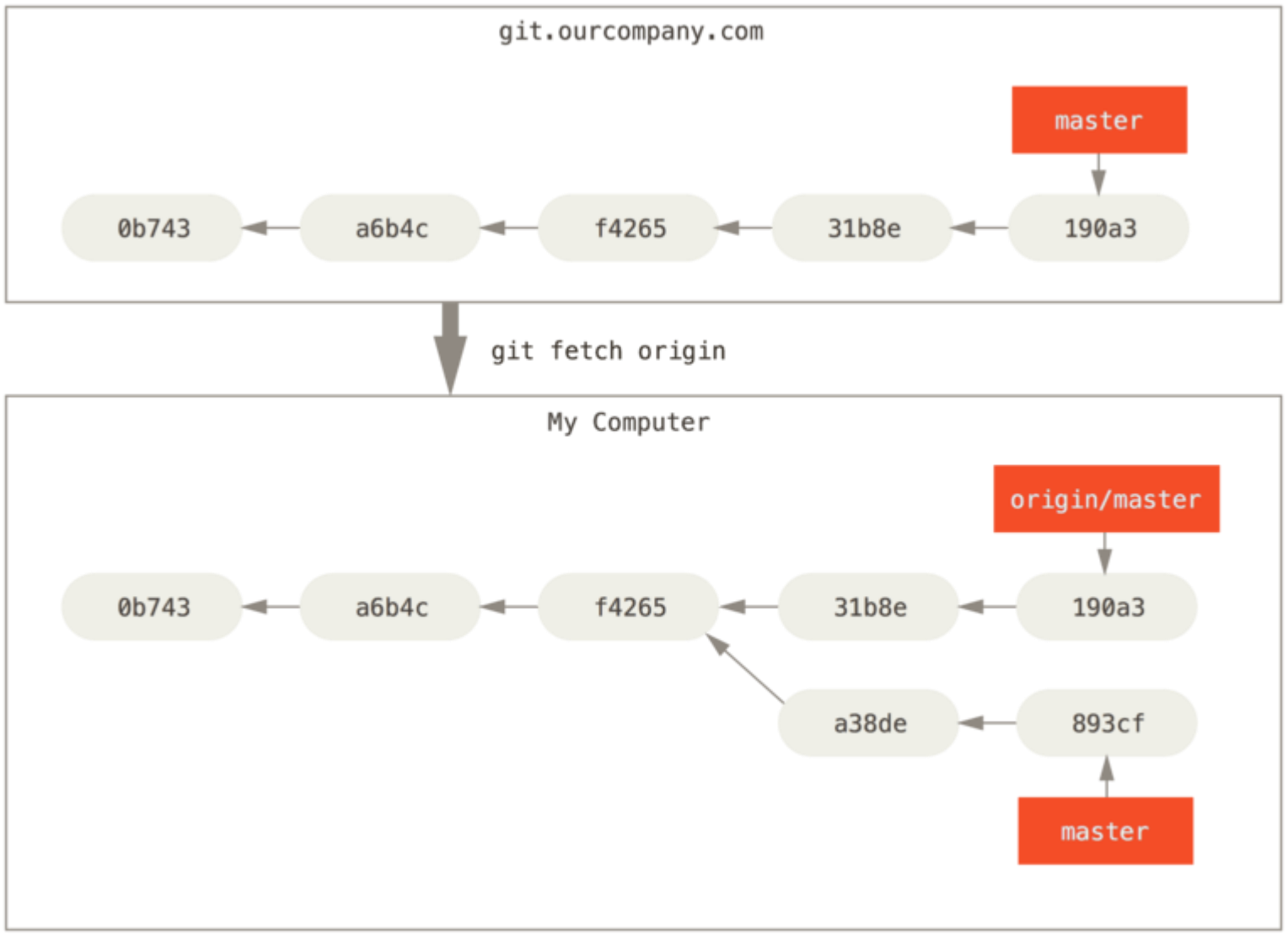
Les remote



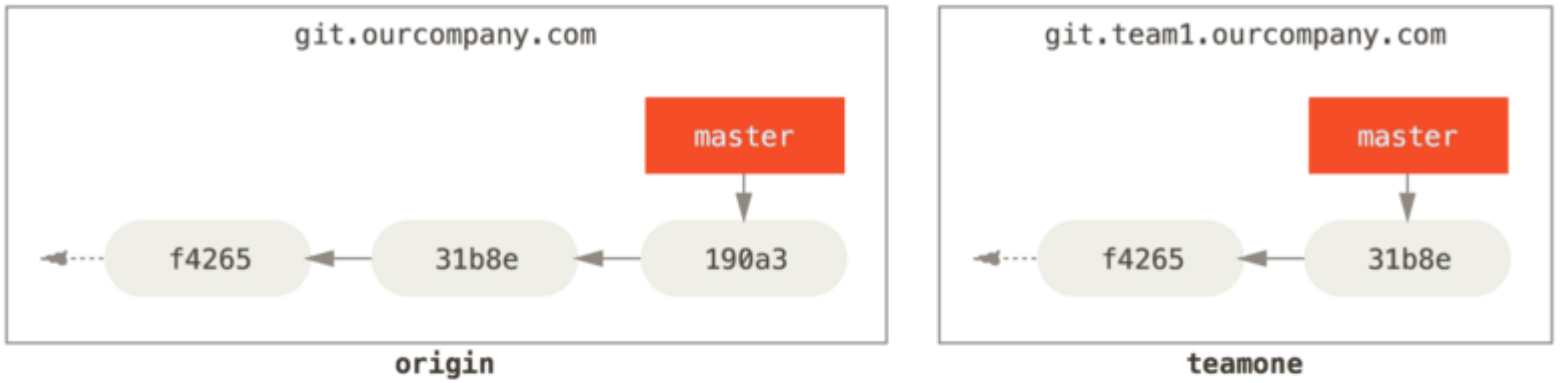
Dépôts distant et local après un clone

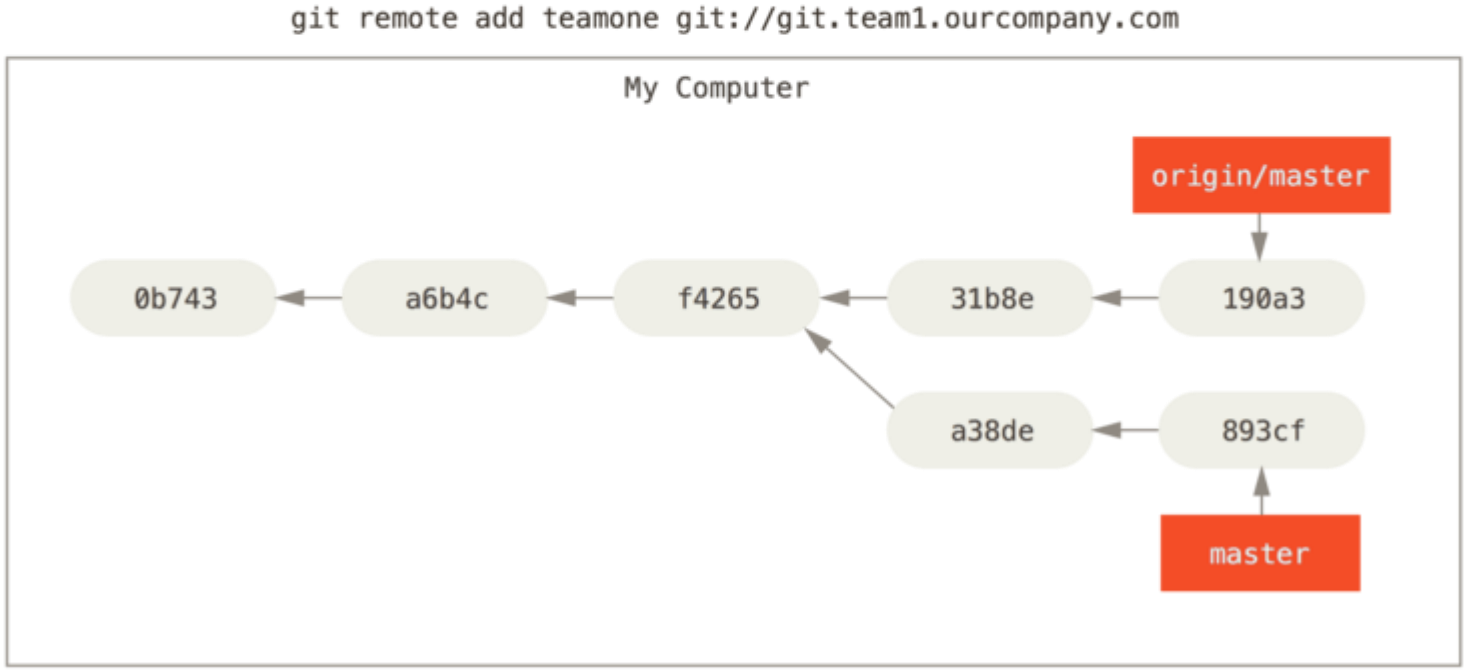


Les travaux locaux et distants peuvent diverger

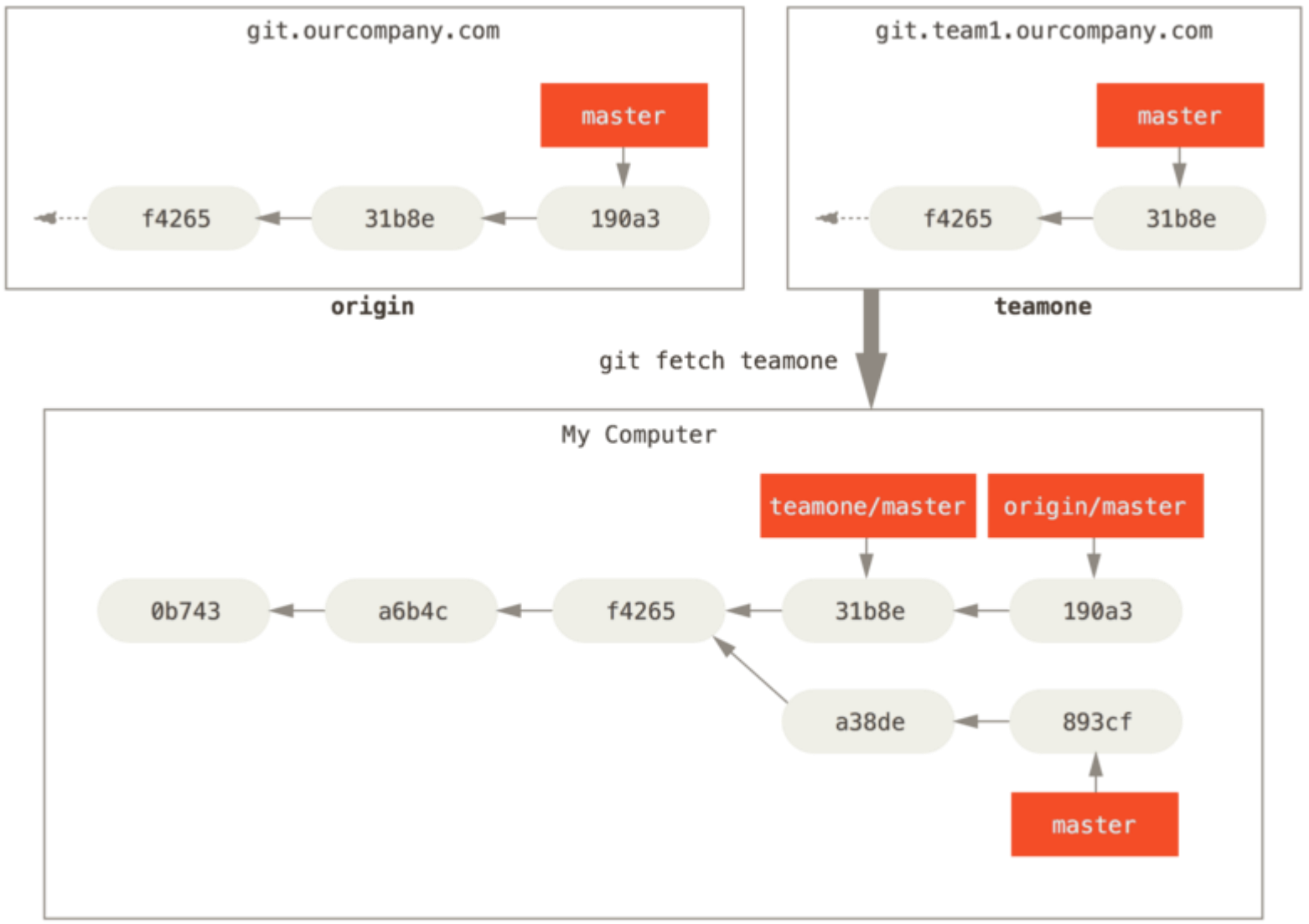


git fetch met à jour vos branches de suivi à distance



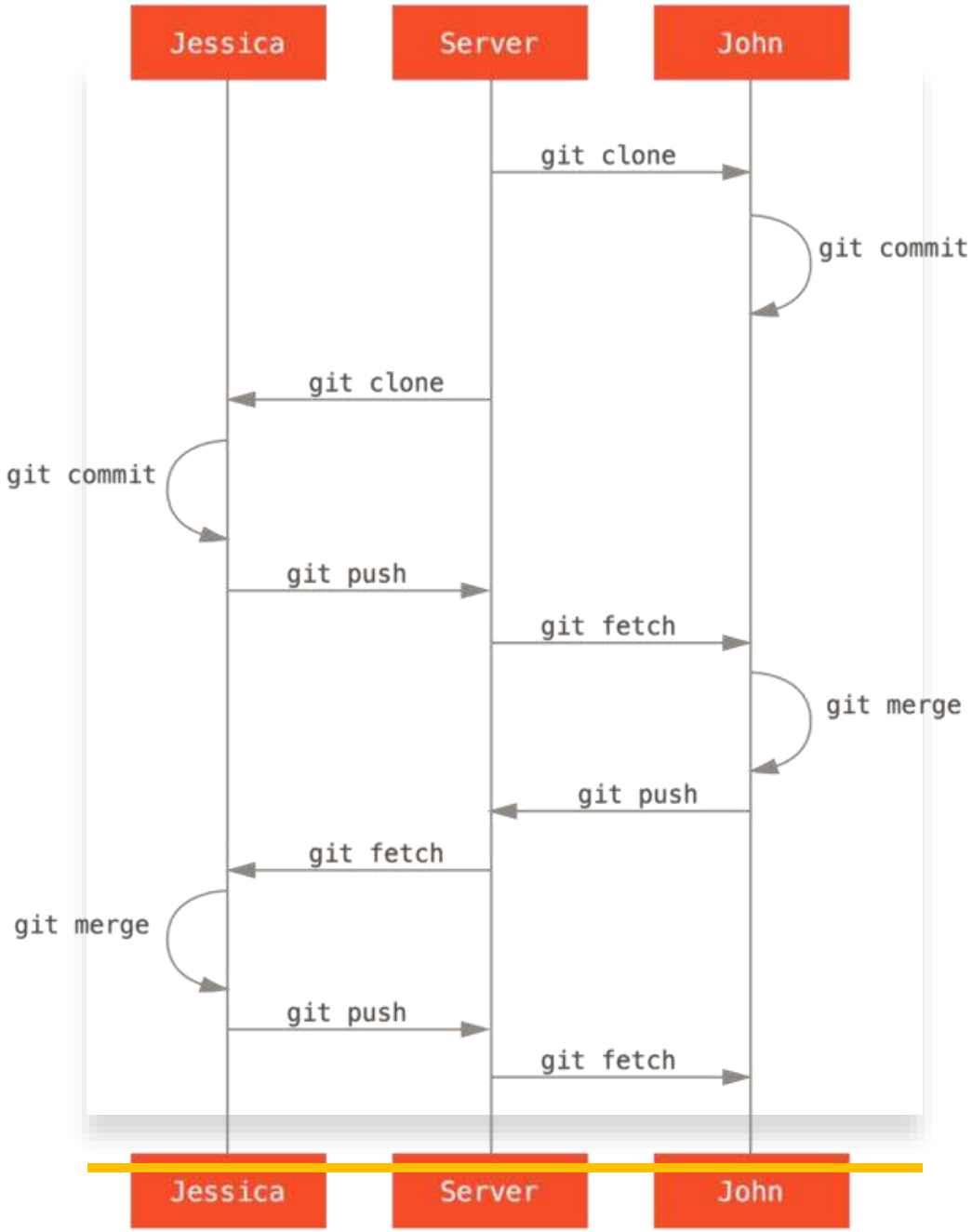


Ajout d'un nouveau serveur en tant que référence distante



Branche de suivi à distance teamone/master

Séquence générale des événements pour une utilisation simple multi-développeur de Git.



Flux de travail typique sur la machine d'un développeur

#1. Clone the project repository

developer$ git clone https://gitlab.com/git_divya/structuralStrategy.git

#2. Checkout 'dev'(or 'uat') branch and switch to the branch with the "-b" flag

developer:[release] $ git checkout -b dev

 # Add code changes on dev branch

developer:[dev] $git add . && git commit -m "Adding navigation code"

#3. Create 'feature1' branch from an older project snapshot; say commit id-"62fc03f"

developer:[dev] $git checkout -b feature1 62fc03f

#Develop the code for feature1 changes, add to index and commit the changes

developer:[feature1] $git add . && git commit -m "Adding feature1 code functions"

#4. Create 'bugfix_940' branch from an initial project snapshot with the tagid-"r1.0"

developer:[feature1] $git checkout -b bugfix_940 r1.0

#Write and test the coce and commit to this branch

developer:[bugfix_940] $git add . && git commit -m "fixed the bug#940"

#5. Checkout 'dev' branch

developer:[bugfix_940] $git checkout dev

 # Merge 'feature1' and 'bugfix_940' branches into dev resolving the conflicts if any

developer:[dev] $git merge feature1

developer: [dev] $git merge bugfix_940

#6. Update local 'dev' branch with remote 'dev' branch using 'git pull' command to be in synch

developer:[dev] $git pull

#7. Push local 'dev' branch merged changes to the Project remote repo

Developer:[dev] $git push

Elies Jebri : elies.jebri@gmail.com

Flux de travail typique sur la machine d'un Mainteneur

#1. Clone the project repository

Maintainer:~ $git clone https://gitlab.com/git_divya/structuralStrategy.git

#2. Checkout 'release' branch

Maintainer: [dev] $git checkout release

#3. Merge 'dev'(or 'uat') branch codes, resolve conflicts if any

Maintainer: [release] $git commit -m "Merge dev changes"

#4. Tag the latest release code

Maintainer:[release] $git tag -a r1.6 -m "Tag latest dev changes" HEAD

#5. Send to 'QA' team for approval, merge approved commits into 'release'

Maintainer: [release] $git merge QA

#6. Update 'dev' branch with latest commits on 'release' branch

Maintainer: [release] $git checkout dev

Maintainer: [dev] $git merge release

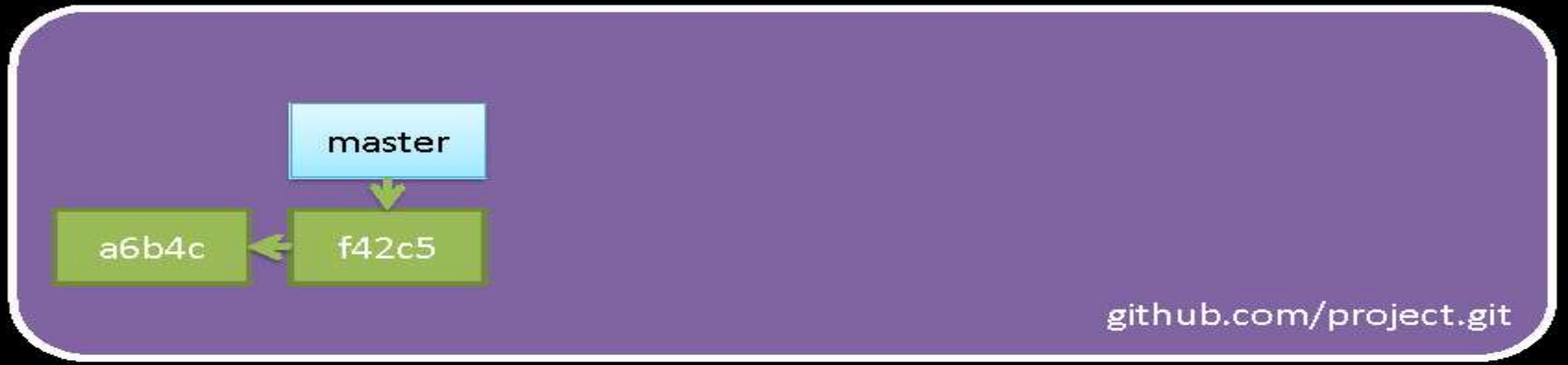
#7. Maintainer push all new commits to the repository

Maintainer: [release] $git push --all

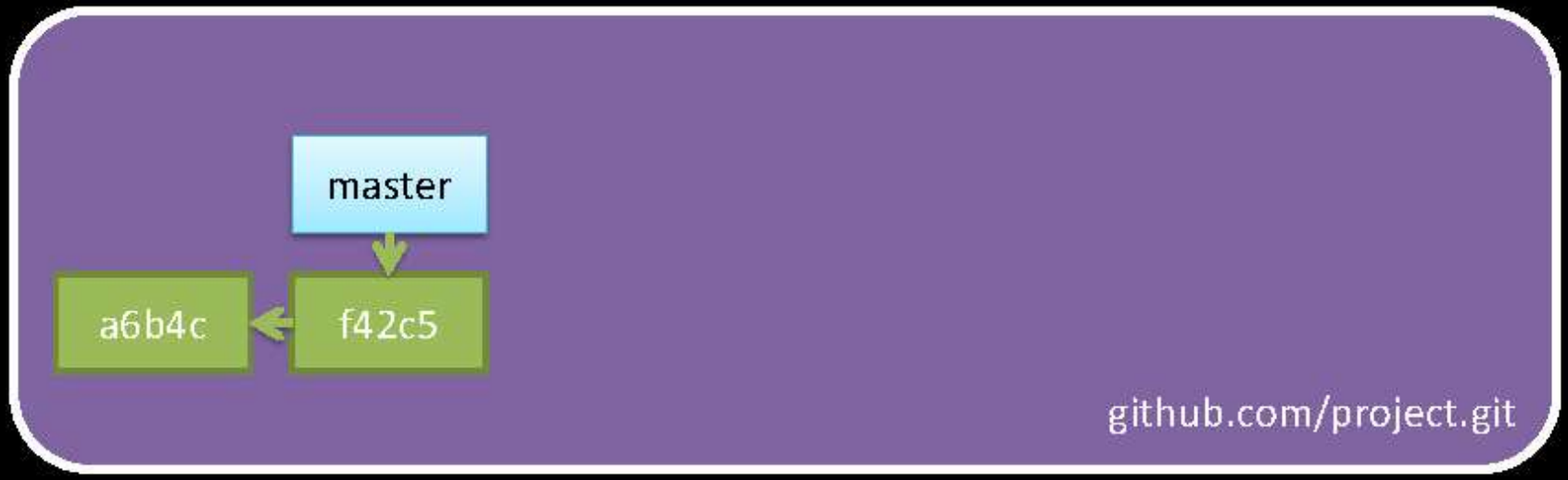
Elies Jebri : elies.jebri@gmail.com

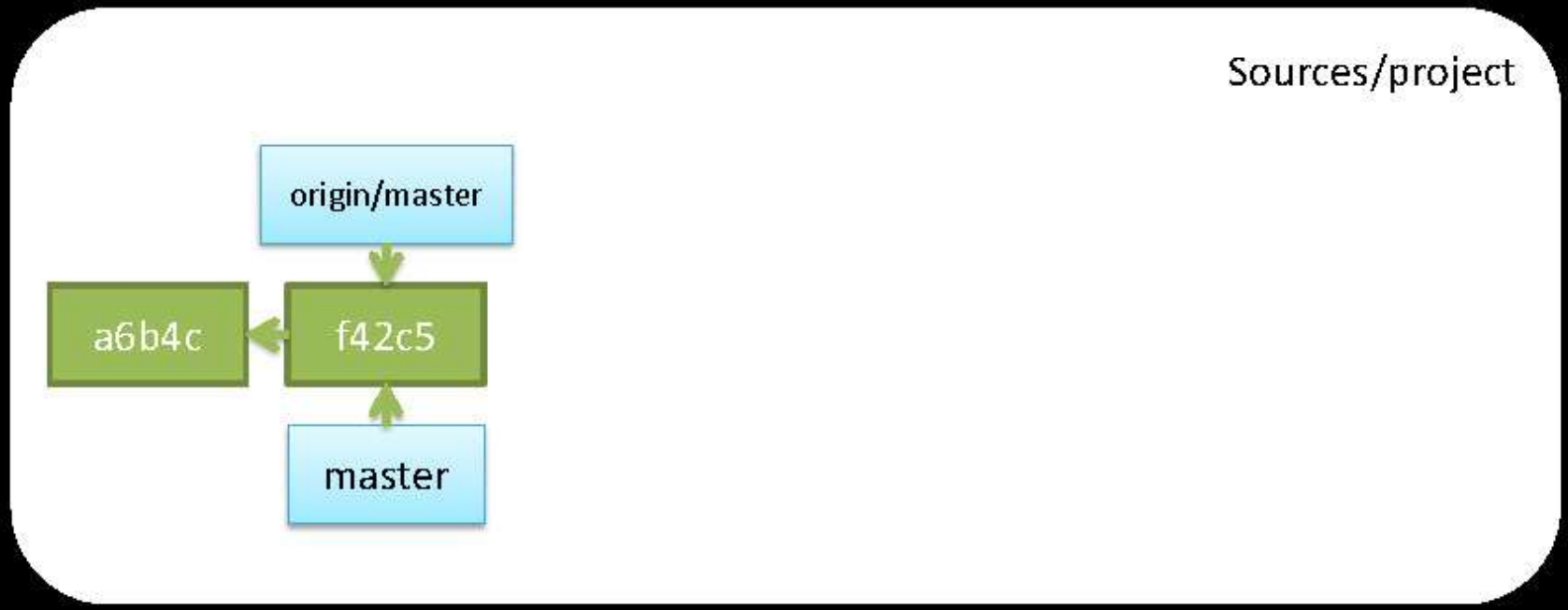
Step by Step

Git branching

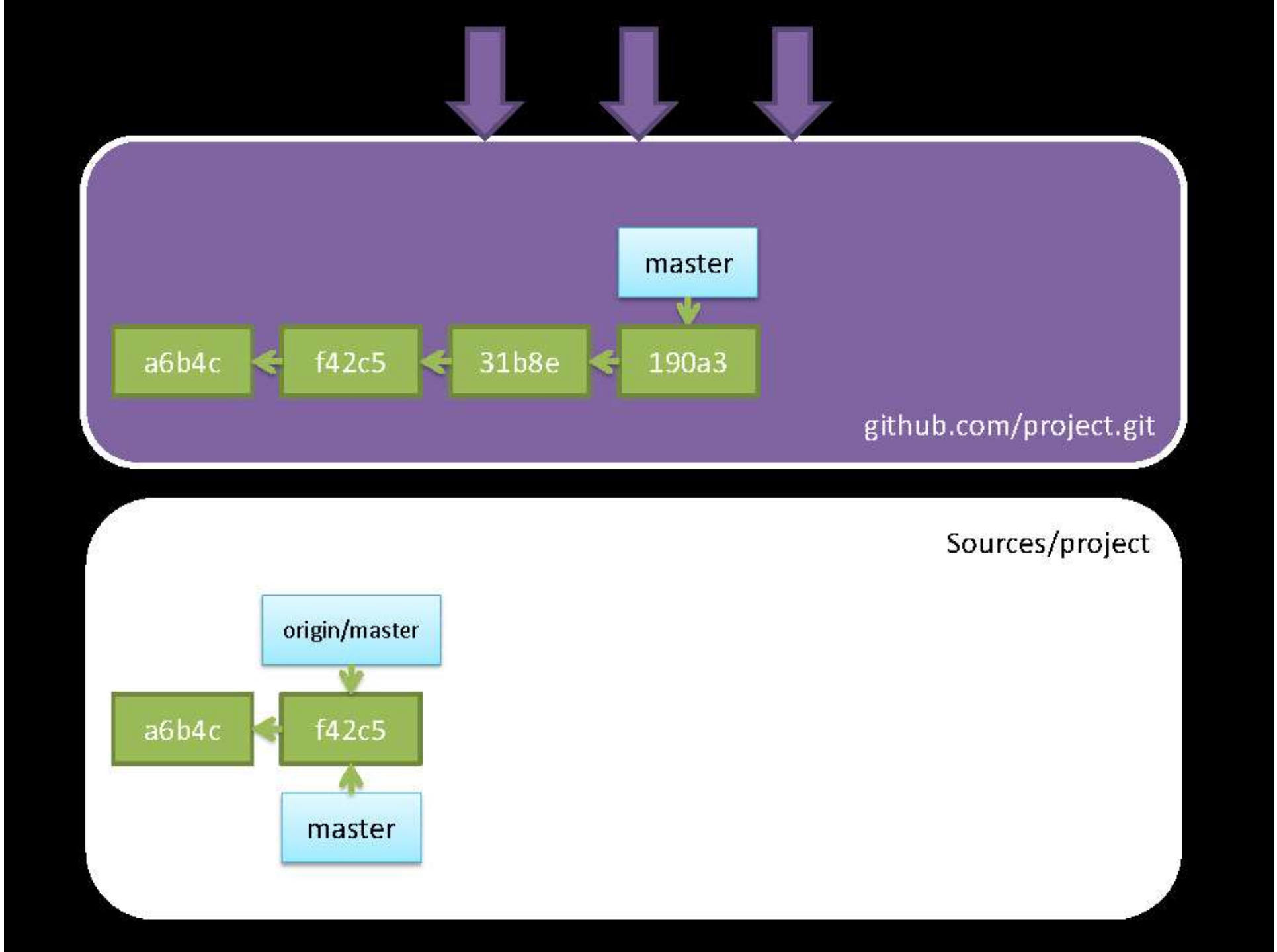

master
a6b4c
f42c5
github.com/project.git

```
$ cd Sources
$ git clone git@github.com/project.git
```



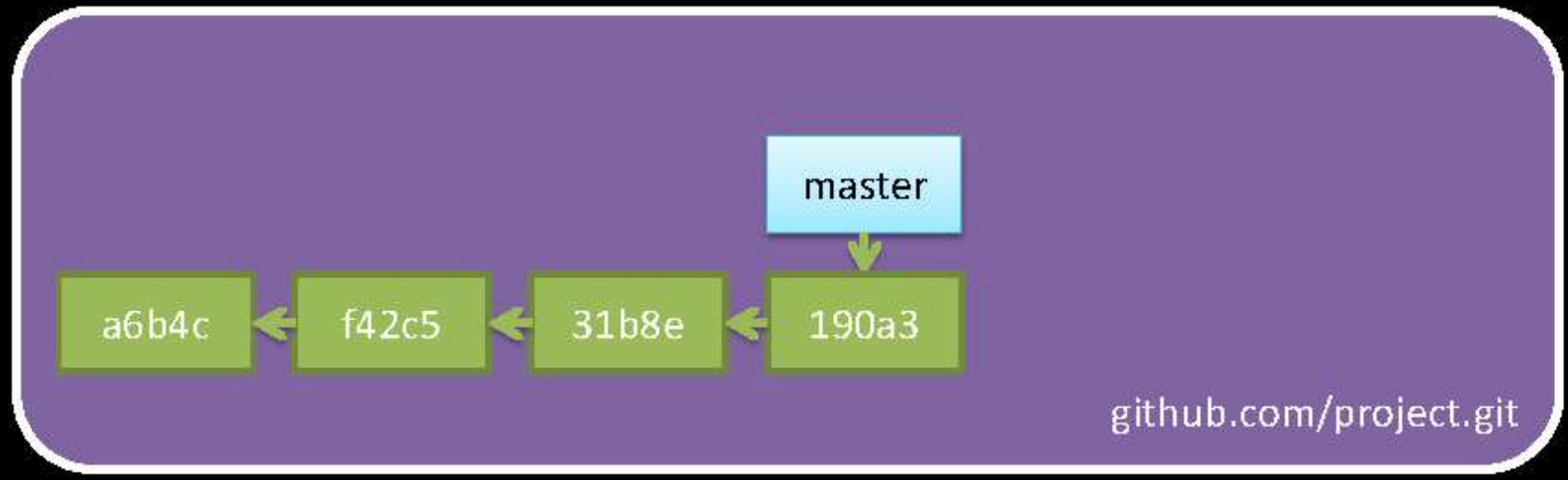


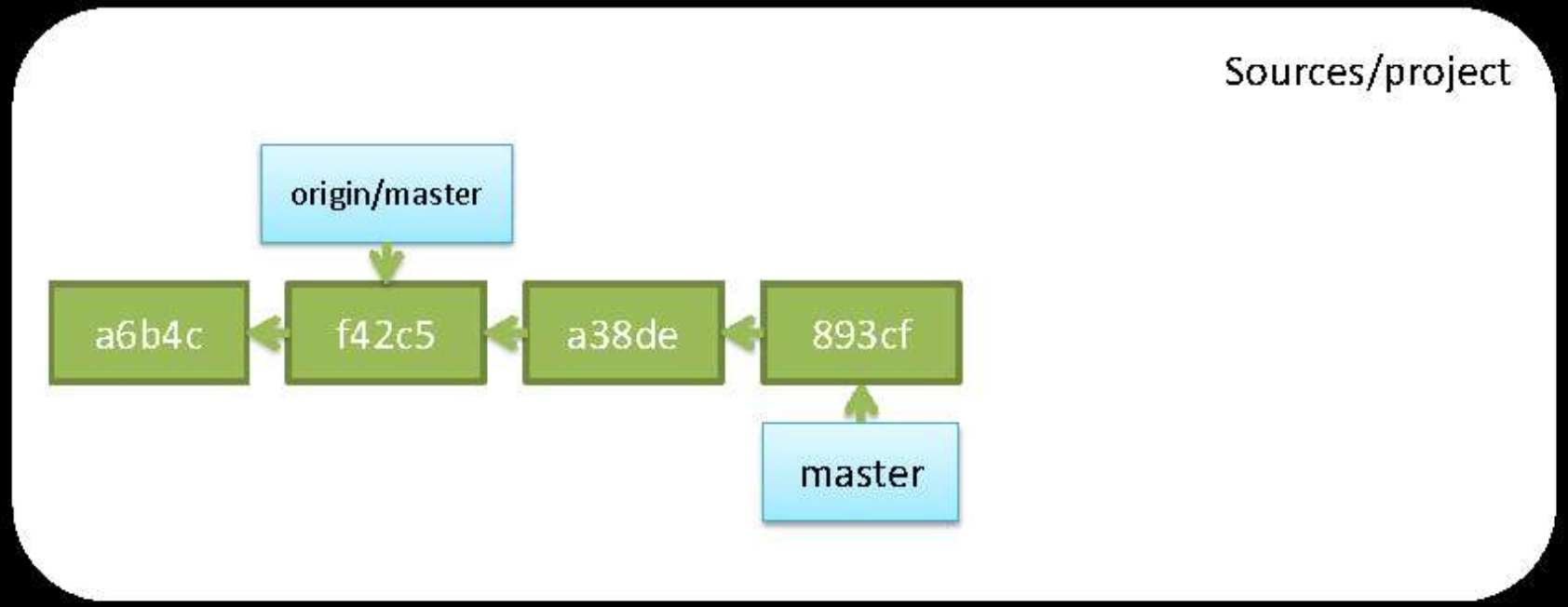
Elies Jebri : elies.jebri@gmail.com



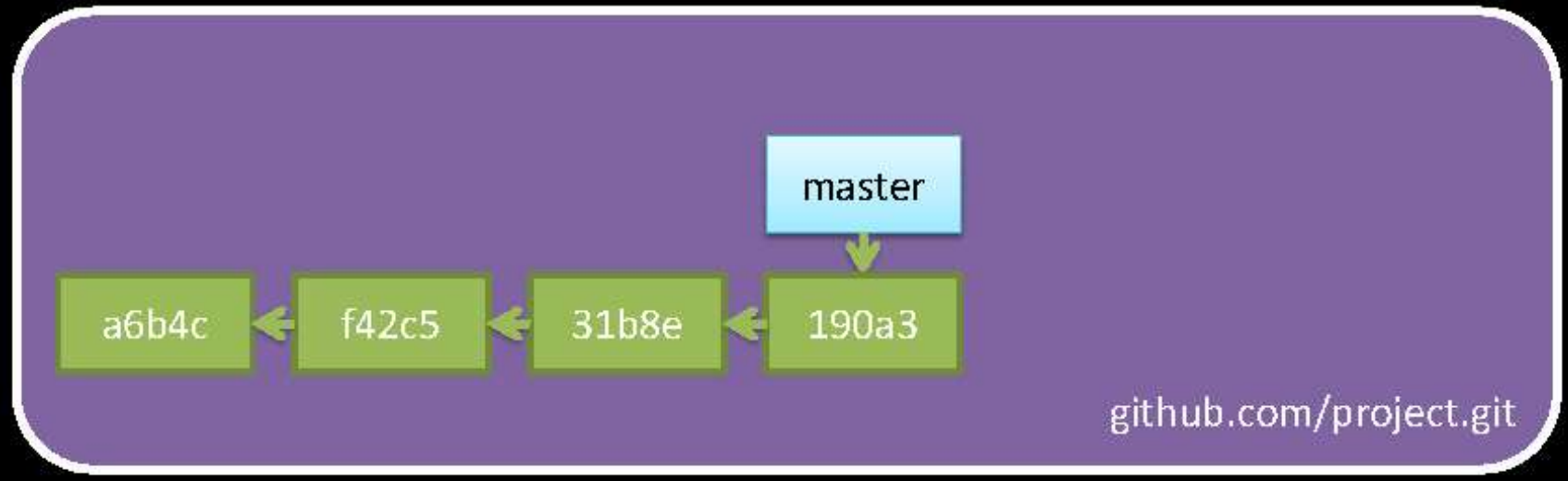
Elies Jebri : elies.jebri@gmail.com

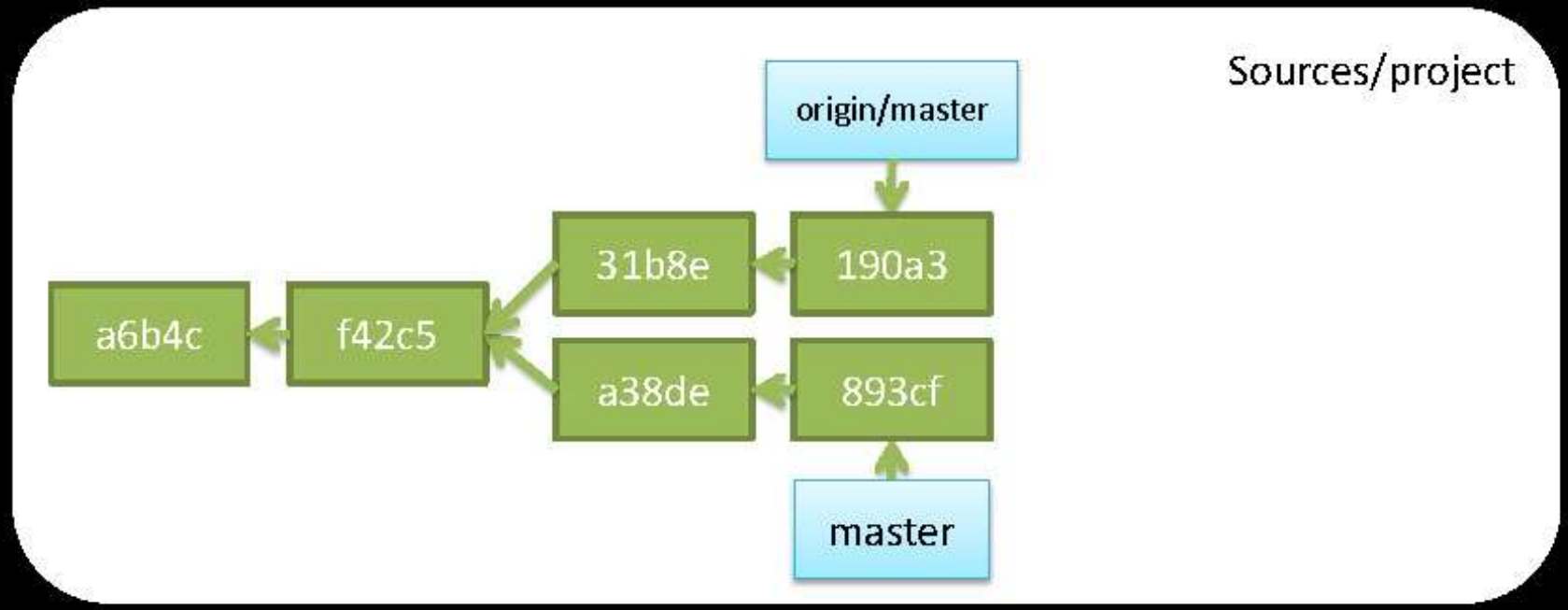
```
$ ...
```





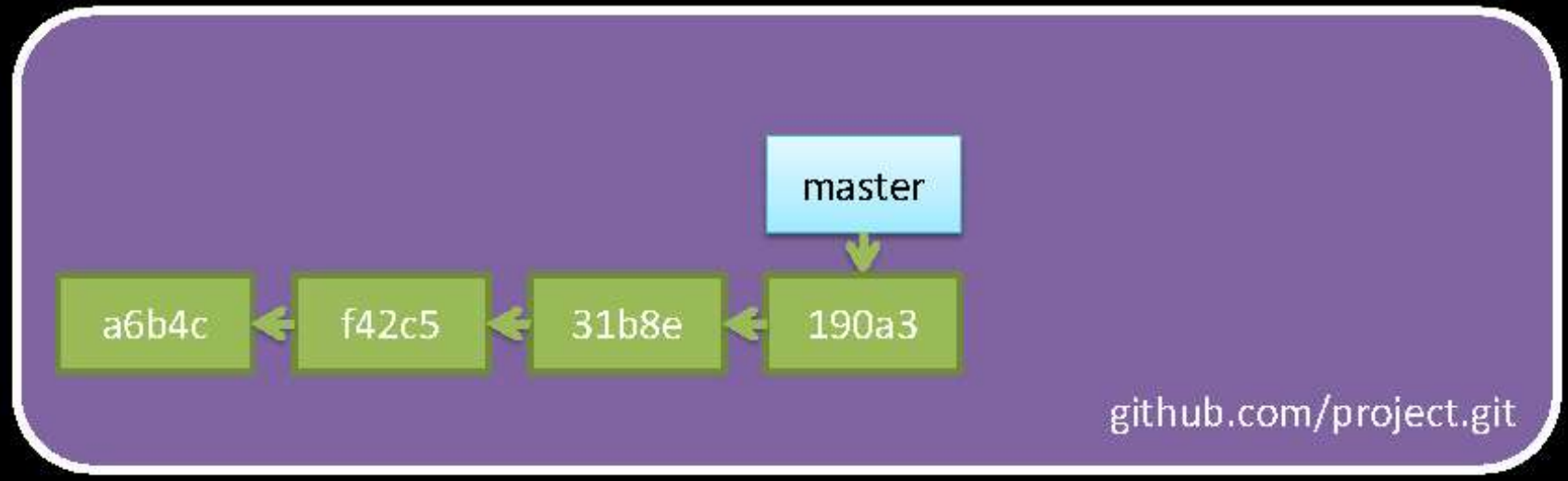
```
$ git fetch origin
```

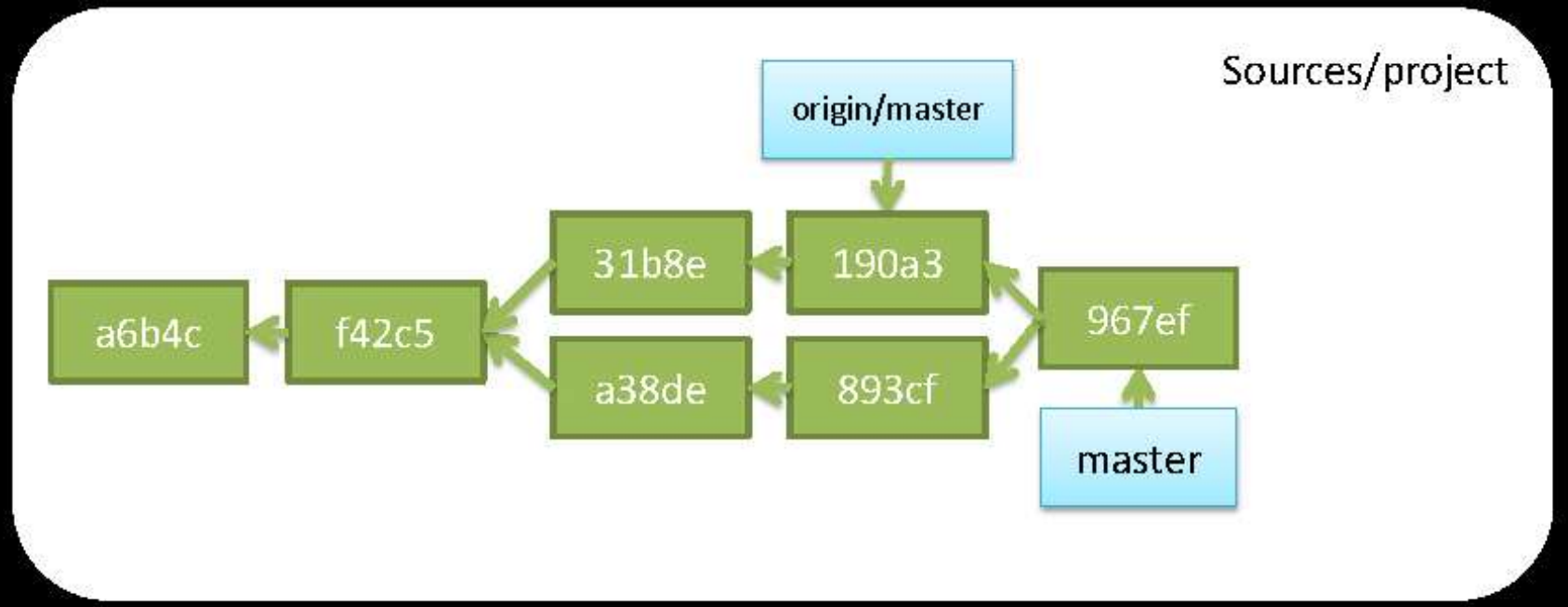




Elies Jebri : elies.jebri@gmail.com

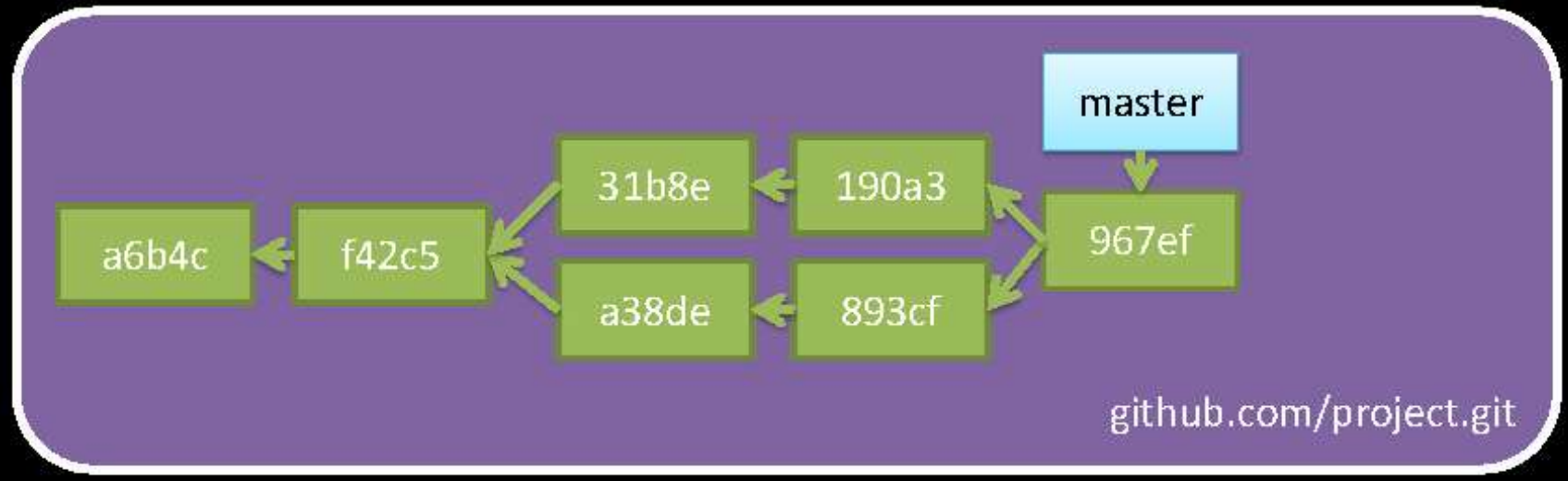
```
$ git merge origin
```

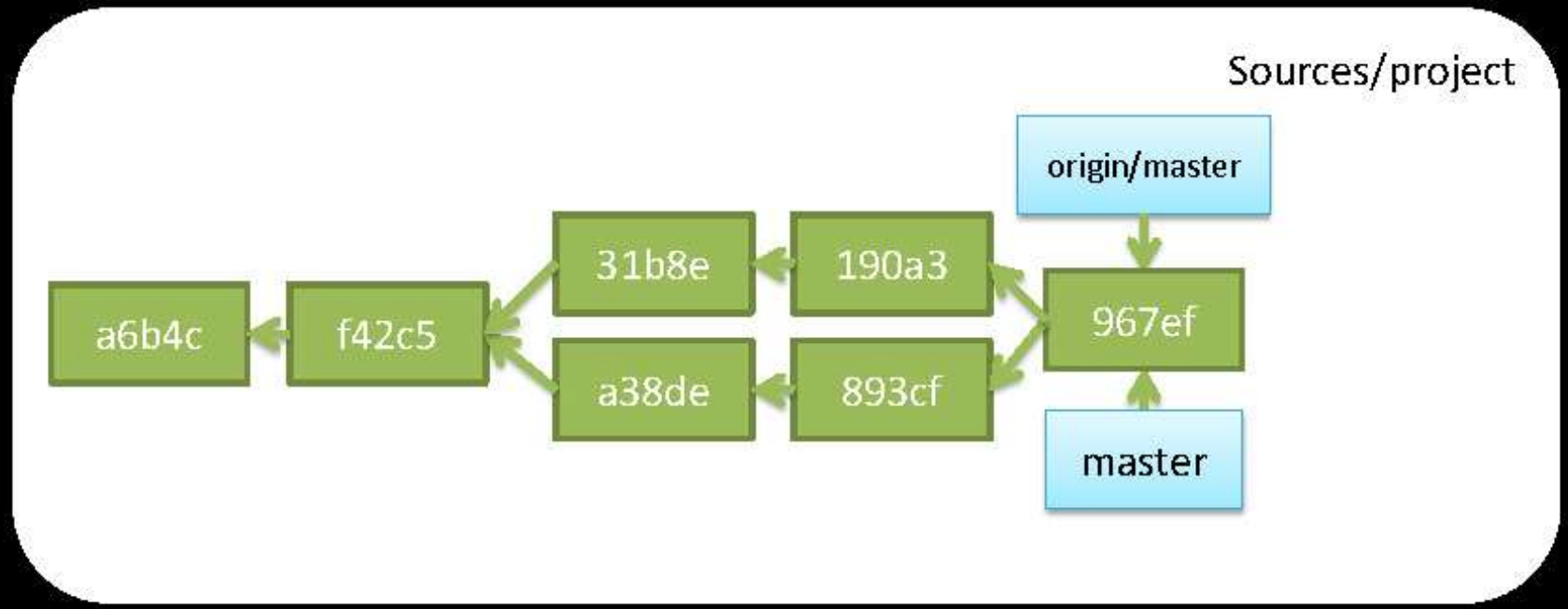




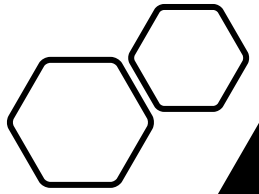
Elies Jebri : elies.jebri@gmail.com

```
$ git push origin
```





Elies Jebri : elies.jebri@gmail.com



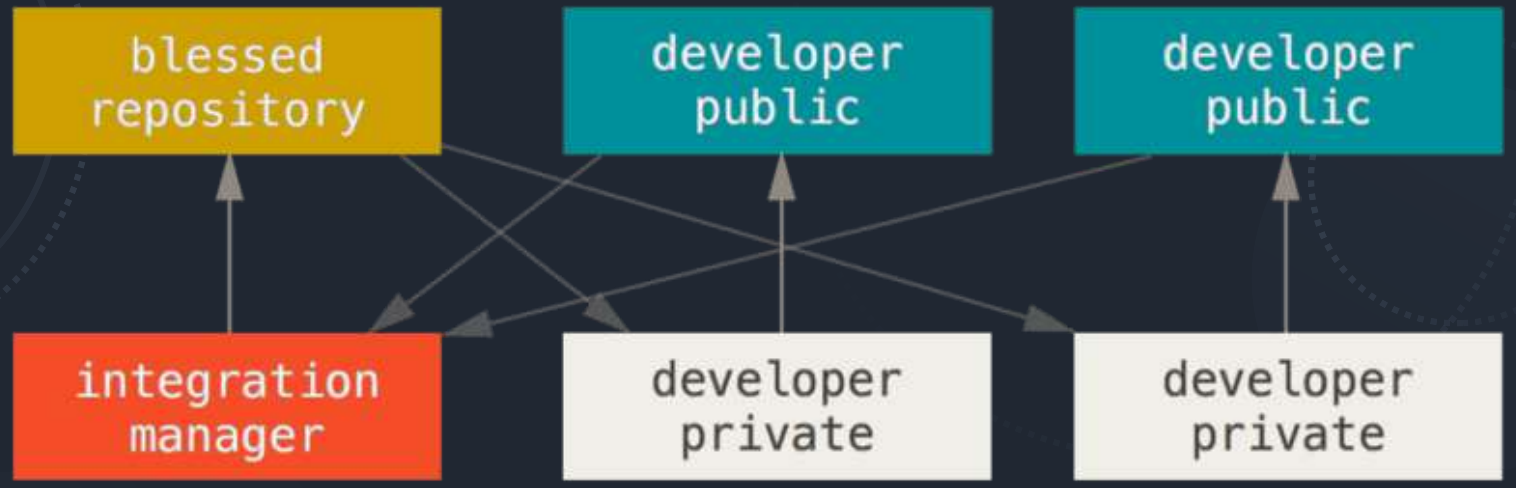
Développements distribués

Git distribué

Qui commande?

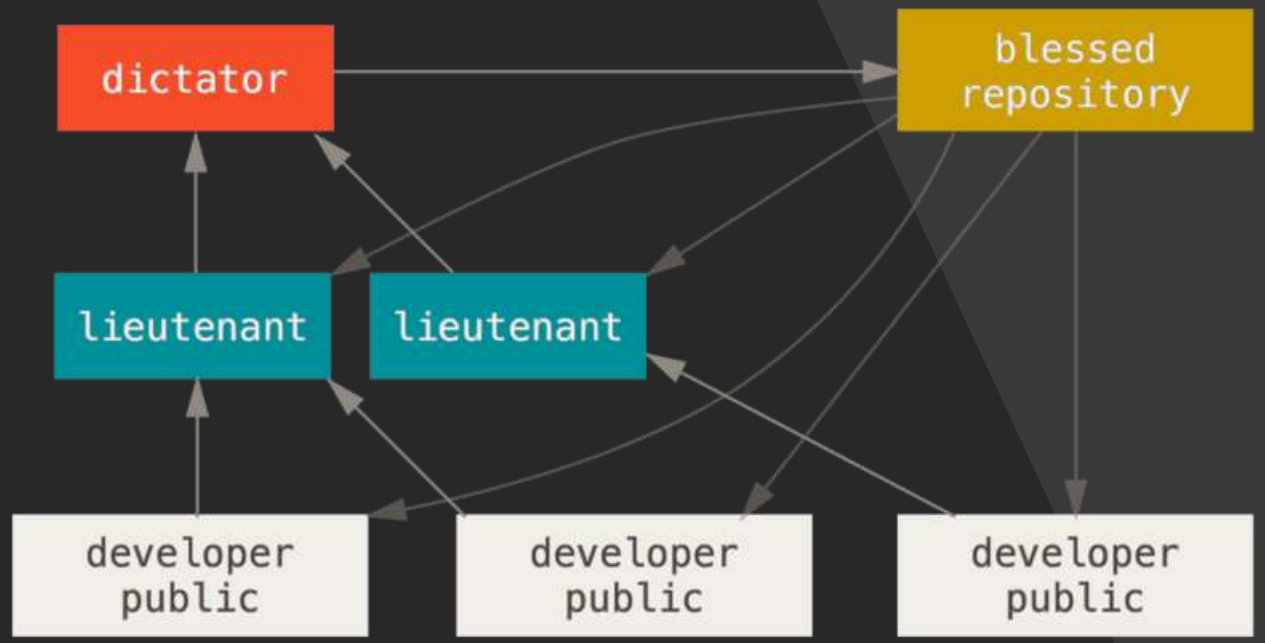
Mode du gestionnaire d’intégration

1. Le mainteneur du projet pousse vers son dépôt public.
2. Un contributeur clone ce dépôt et introduit des modifications.
3. Le contributeur pousse son travail sur son dépôt public.
4. Le contributeur envoie au mainteneur un e-mail de demande pour tirer ses modifications depuis son dépôt.
5. Le mainteneur ajoute le dépôt du contributeur comme dépôt distant et fusionne les modifications localement.
6. Le mainteneur pousse les modifications fusionnées sur le dépôt principal.



Mode dictateur et ses lieutenants

- Les simples développeurs travaillent sur la branche thématique et rebasent leur travail sur master. La branche master est celle du dictateur.
- Les lieutenants fusionnent les branches thématiques des développeurs dans leur propre branche master.
- Le dictateur fusionne les branches master de ses lieutenants dans sa propre branche master.
- Le dictateur pousse sa branche master sur le dépôt de référence pour que les développeurs se rebasent dessus.



Elies Jebri : elies.jebri@gmail.com

Références

- https://git-scm.com/book/fr/v2/
- https://rogerdudler.github.io/git-guide/
- https://www.daolf.com/posts/git-series-part-1/
- https://www.jquery-az.com/git-github-tutorials/
- https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git