PSIA : Git, un gestionnaire de versions décentralisé.

Version 3.01

Julien Sopena[1]

[1]julien.sopena@lip6.fr
Équipe REGAL - INRIA Rocquencourt
LIP6 - Université Pierre et Marie Curie

Master SAR 2ème année - PSIA - 2010/2011

Grandes lignes du cours

Outline

Principe de base des gestionnaires de version

Un gestionnaire de versions (**VCS**) doit permettre de :

- conserver toutes les versions de tous les fichiers ;
- conserver toutes les arborescences de fichiers ;
- permettre d'identifier une arborescence de versions de fichiers ;
- fournir des outils pour gérer le tout.

Un **DVCS** (*"Distributed Version Control"*) offre les mêmes services sur une architecture décentralisée.

Tous deux reposent sur deux mécanismes :

1. un mécanisme permettant de calculer la différence entre deux versions : **diff**/**patch** ;
2. un gestionnaire d'**historique** des diff.

diff & patch

diff : Comparaison de fichiers ligne par ligne

- indique les lignes ajoutées ou supprimées ;
- peut ignorer les casses, les tabulations, les espaces ;
- option -u pour créer des patchs unifiés, avec plus d'informations.

patch : Utilise la différence entre deux fichiers pour passer d'une version à l'autre.

Exemple

```
$ diff toto.c toto-orig.c > correction.patch
$ bzip2 correction.patch
$ bzcat correction.patch.bz2 | patch -p 0 toto.c
```

diff & patch

Les diff peuvent comparer des hiérarchies de fichiers (option -r) :

Exemple

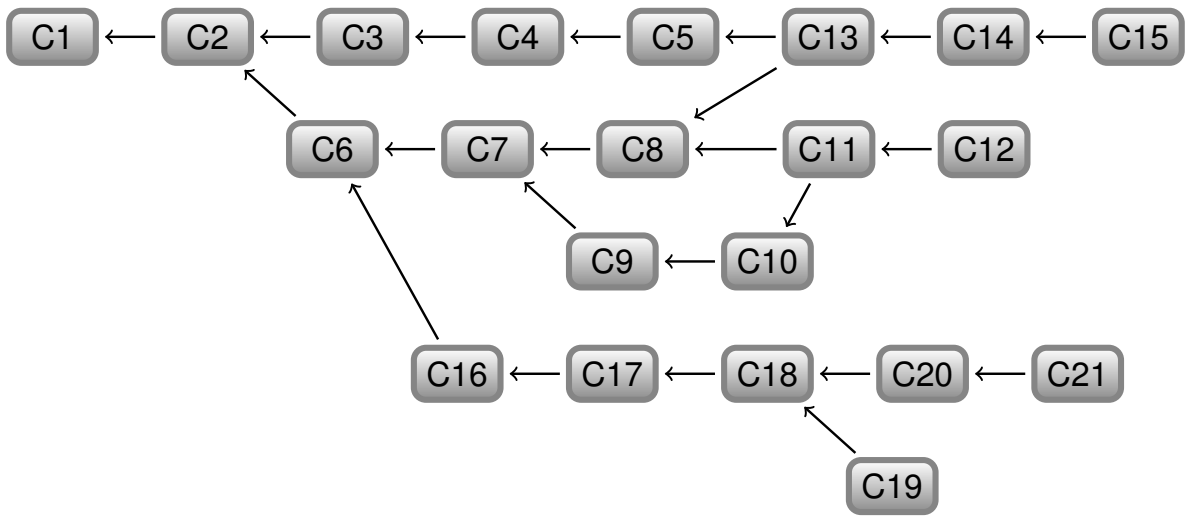
```
$ cp -r linux-2.6.17 linux-2.6.17-orig
$ cd linux-2.6.17
$ vim [...]
$ cd ..
$ diff -r -u linux-2.6.17-orig linux-2.6.17 > \
              network-driver-b44.patch
$ gzip -9 network-driver-b44.patch

$ cd linux-2.6.17
$ zcat network-driver-b44.patch.gz | patch -p 1
```

L'option -p permet de laisser les chemins du patch au complet 0, en supprimant le premier dossier 1, etc.

La notion d'historique

Définition

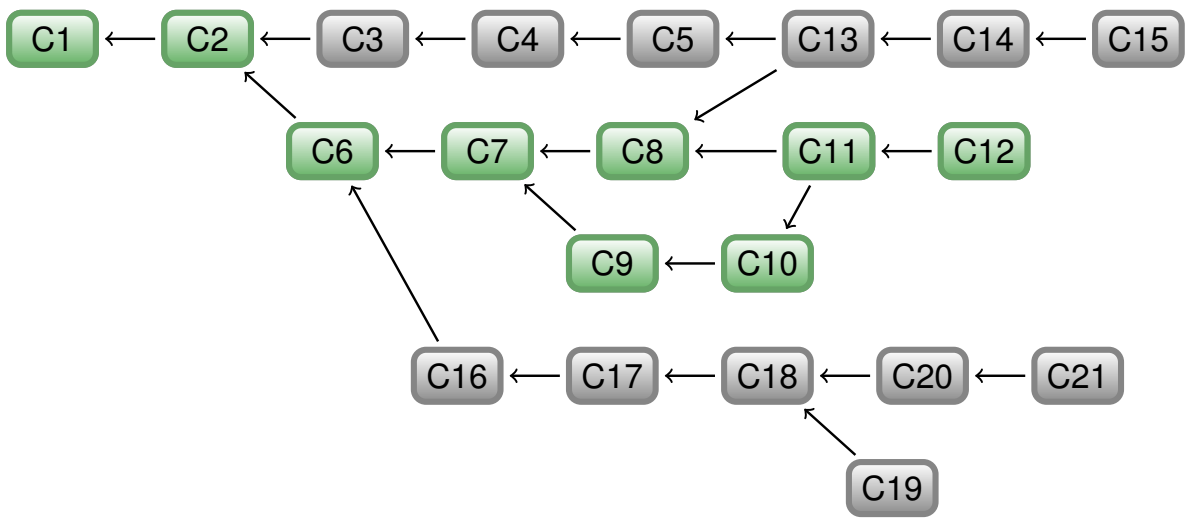
On appelle ***historique*** *un graphe orienté acyclique composé d'un ensemble de versions pouvant être recalculées à partir des versions adjacentes en appliquant les patchs modélisés par les arcs sortants.*



Historique : Les branches

Définition

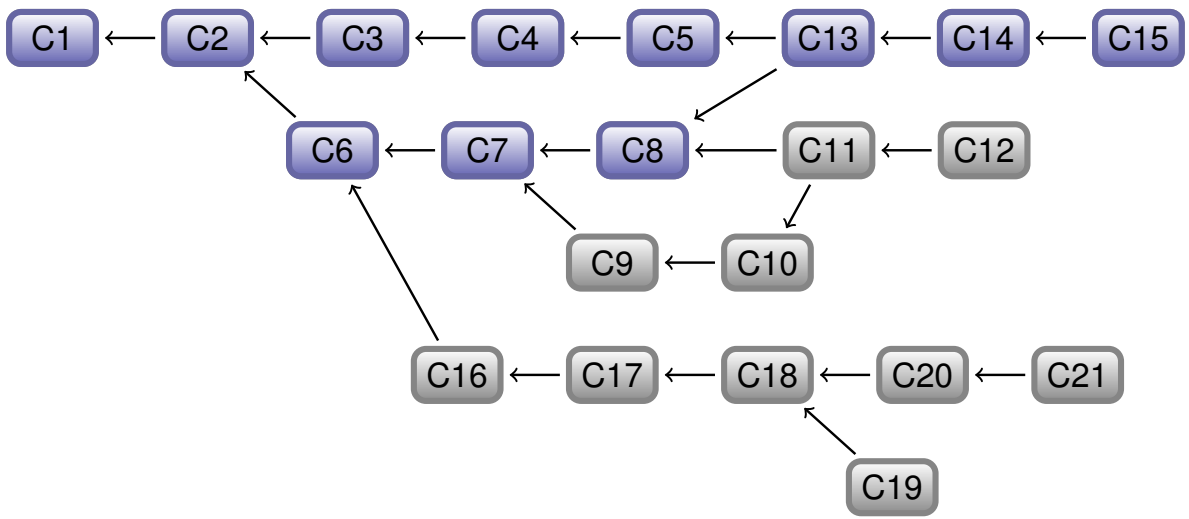
*La **branche** de la version v_i d'un historique est le sous graphe composé de l'ensemble des versions accessibles depuis v_i dans le graphe de l'historique.*



Historique : Le tronc

Définition

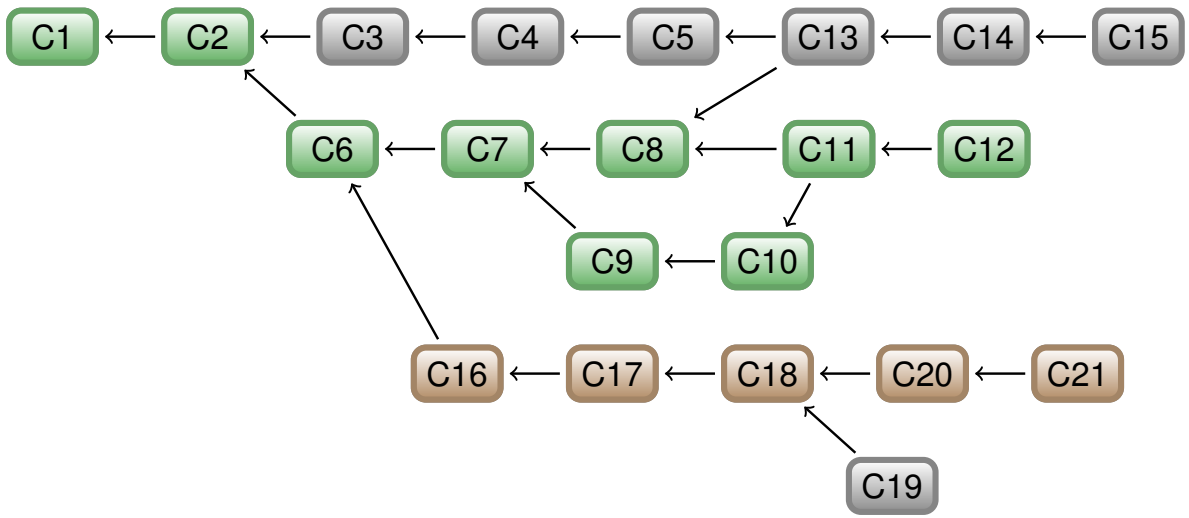
*Le **tronc** ou **branche principale** de l'historique est la branche issue de la dernière version stable.*



Historique : Sous branches

Définition

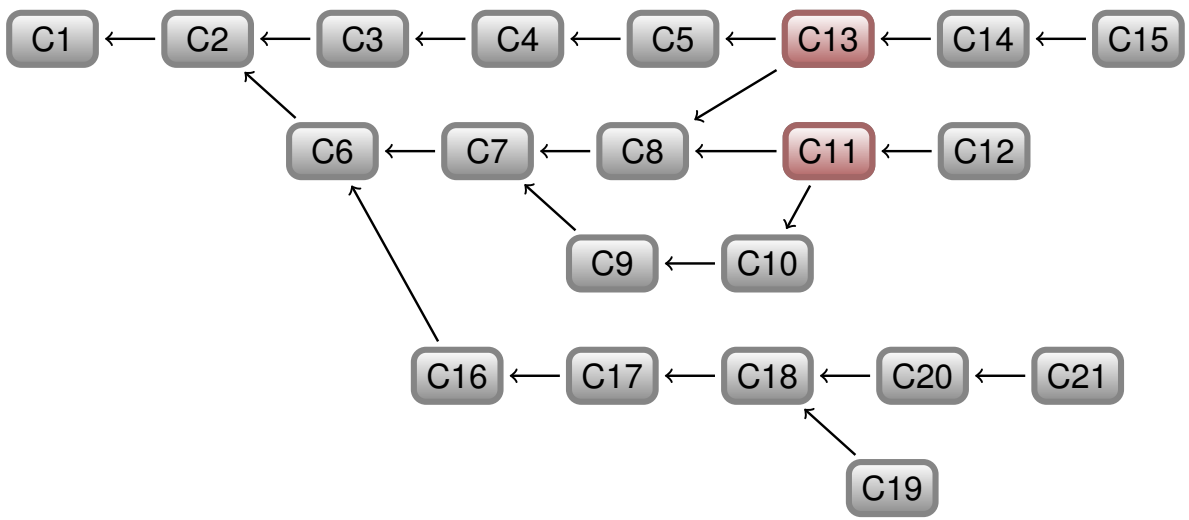
*Une **sous-branche** SB_1 d'une branche B_2 est le sous graphe composé de l'ensemble des versions d'une branche B_1 n'appartenant pas à la branche B_2, l'intersection de B_1 et de B_2 devant être non vide.*



Historique : Les merges.

Définition

*On appelle **merge** toute version ayant un degré sortant strictement supérieur à 1. Cette version correspond alors à la fusion des **patchs** de plusieurs branches.*



Historique

2001 : Linux est développé sur **CVS**.

2002 à 2005 : Linux est développé sur **Bitkeeper** :

- *Bitkeeper* est décentralisé.
- Plus *Bitkeeper* progresse et plus le développement de Linux devient efficace.

6 Avril 2005 : *Bitkeeper* quitte le libre :

- Création de Git par Linus Thorvalds.

18 Avril 2005 : Git peut faire un *Merge*

16 Juin 2005 : Linux est développé officiellement sur **Git**.

14 Février 2007 : Sortie de la version 1.5.0

- Git devient vraiment utilisable par tous !

Outline

Les 4 types d'objets

Git utilise quatre types d'objets :

Blob : Stocke le contenu des fichiers.

Tree : Stocke l'arborescence.

Commit : Stocke les versions du dépôt.

Tag : Identifie certaines versions du dépôt.

Attention

Les objets de types Blob, Tree ou Commit ne sont pas mutables et ne doivent normalement pas être supprimés.

Les références d'objets

Git a été conçu comme un : **système de fichiers versionnés**.

Comme dans le noyau, on retrouve ici la notion de pointeur ou référence sous-forme de clés calculées à partir d'une fonction de hachage : **SHA-1**.

Les objets sont organisés sous formes d'un **graphe orienté acyclique**.

SHA-1 ?

Secure Hash Algorithm

- SHA-1 est une fonction de hachage cryptographique.
- SHA-1 a été conçu par la NSA.
- fonction de hachage $\rightarrow$ grand ensemble de données en un ensemble plus petit et unique (à quelques 2^{80} clés différentes).
- Génère un *hash* de 160 bits.

Exemple

```
$ echo a > toto
$ sha1sum toto
    3f786850e387550fdab836ed7e6dc881de23001b  toto
$ echo a >> toto
$ sha1sum toto
    d7c8127a20a396cff08af086a1c695b0636f0c29  toto
```

Outline

Les "Blobs" : définitions

Définition

*On appelle **Blob**, l'élément de base qui permet de stocker le contenu d'un fichier :*

- *Chaque Blob est identifié de manière unique par une clé SHA1 calculée à partir de son contenu.*
- *À chaque révision du fichier correspond un nouveau Blob.*
- *Le Blob ne dépend pas du nom ou de l'emplacement :*
  - *Si un fichier est renommé, pas de nouveau Blob*
  - *Si un fichier est déplacé, pas de nouveau Blob*

Les "Blobs" : exemple

Outline

Les "Trees" : définitions

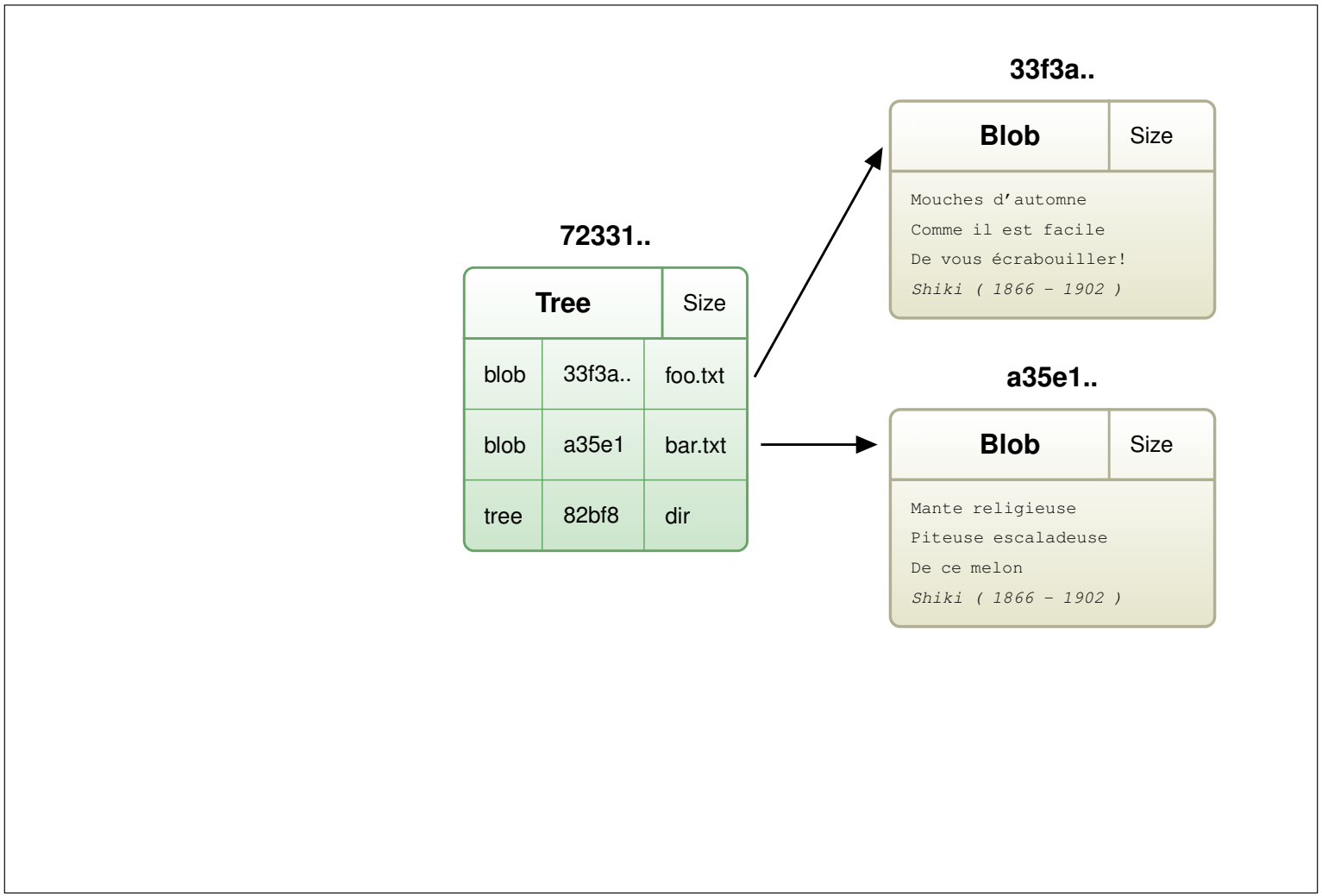
Définition

*Un **Tree** stocke la liste des fichiers d'un répertoire :*

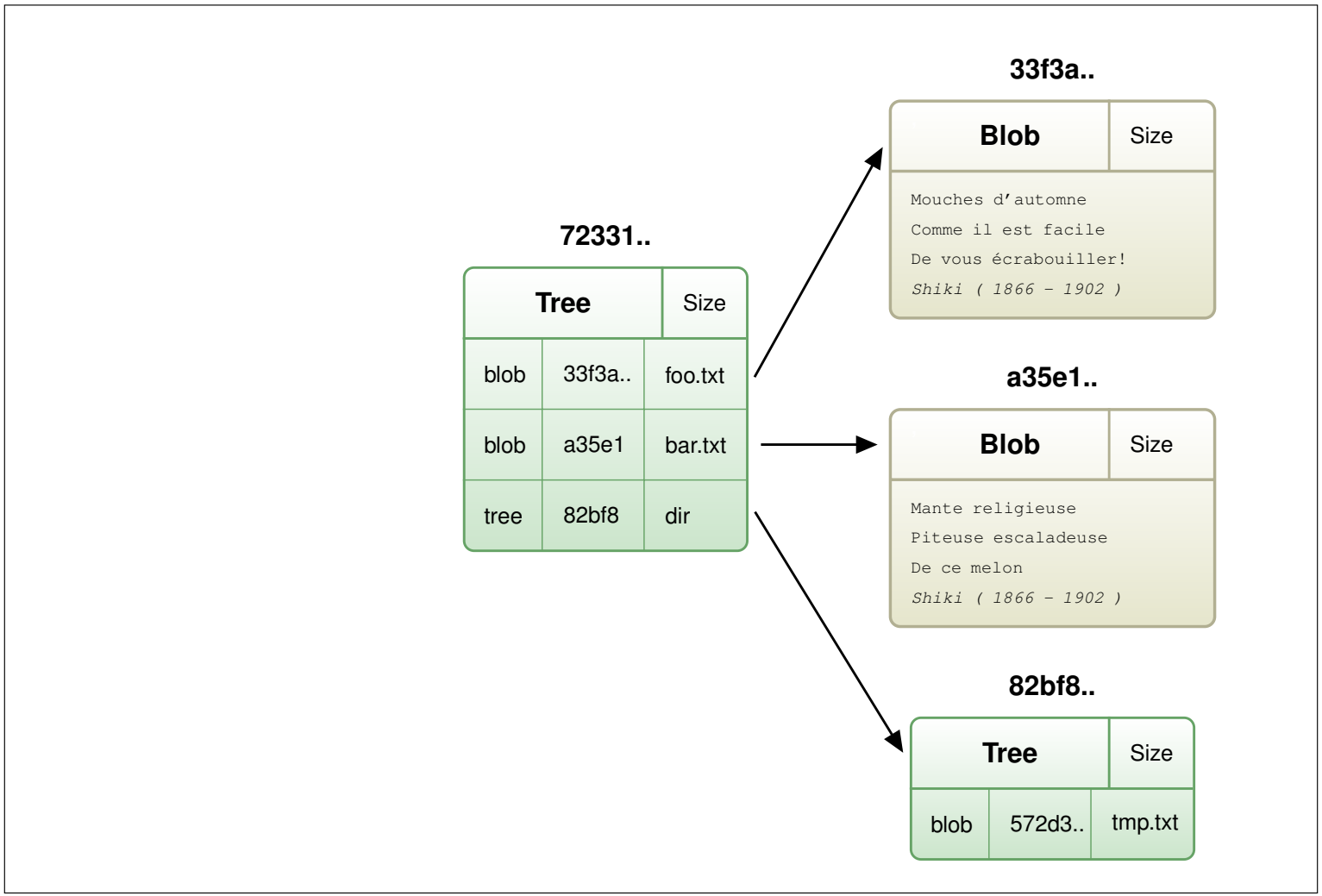
- *Un **Tree** est un ensemble de pointeurs vers des **Blobs** et d'autres **Trees**.*
- *Un **Tree** associe un nom de fichier (resp. repertoire) à chacun des pointeurs de **Blobs** (resp. **Trees**).*
- *Un ensemble de **Tree** permet de décrire l'état d'une hiérarchie de dossiers à un moment donné.*

Les "Tree" : exemple

Les "Tree" : exemple



Les "Tree" : exemple



Outline

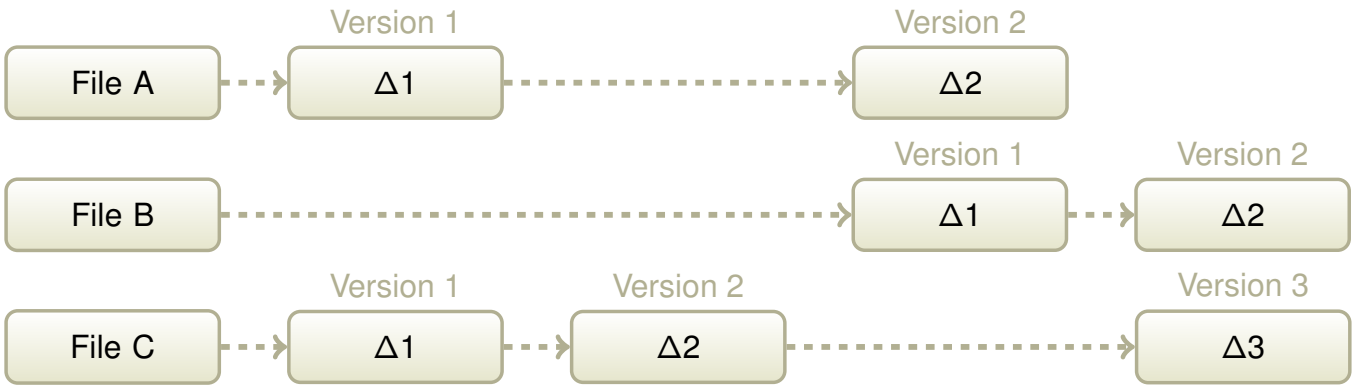
Commit : définition.

Définition

Commiter *un fichier signifie : enregistrer une version de ce dernier (le plus souvent la version actuelle, mais pas toujours) dans un gestionnaire de versions. Par extension, si le fichier est déjà versionné, on dit que l'on* ***commit*** *une modification du fichier.*

La numérotation des commits dans *CVS*.

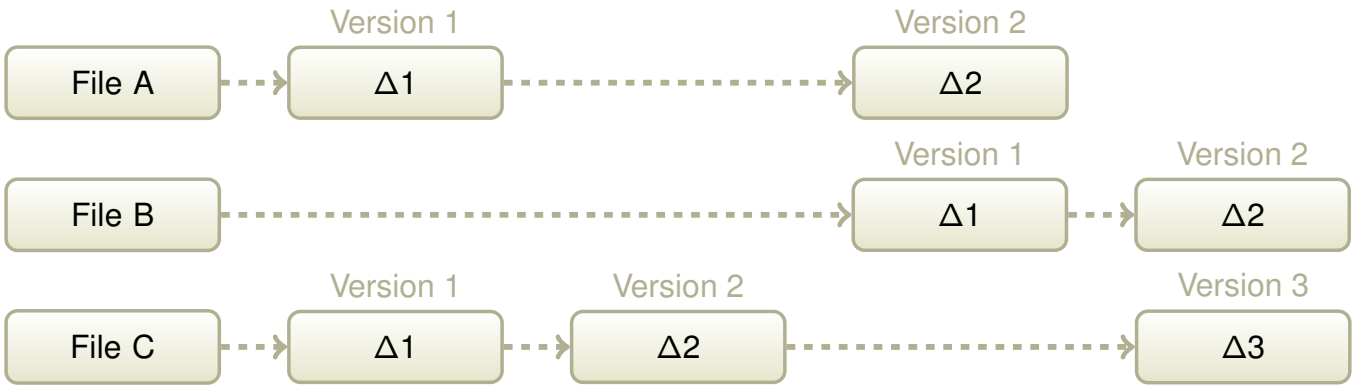
CVS utilise une numérotation sur des patchs, fichier par fichier :



La numérotation des commits dans *CVS*.

CVS utilise une numérotation sur des patchs, fichier par fichier :

=> difficile de trouver un état cohérent du système (ajout de tag).

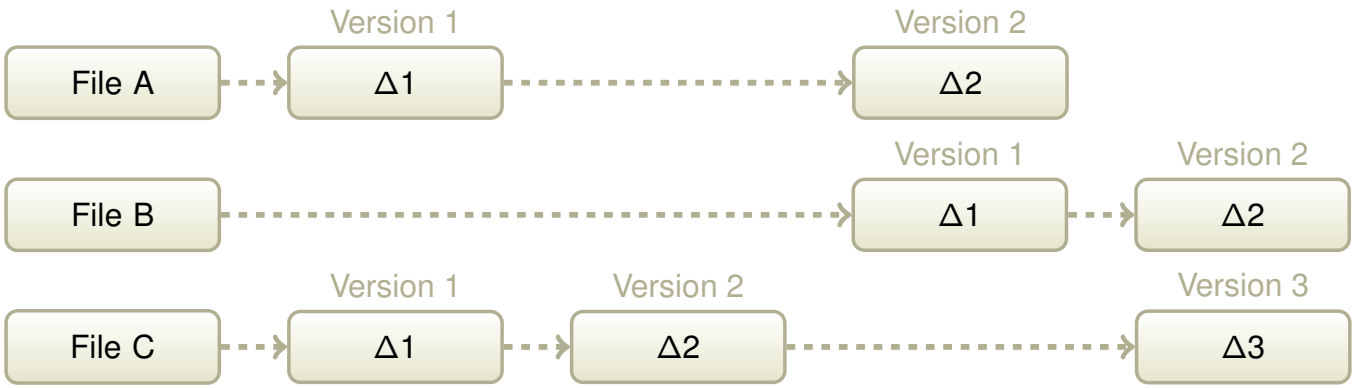


La numérotation des commits dans *CVS*.

CVS utilise une numérotation sur des patchs, fichier par fichier :

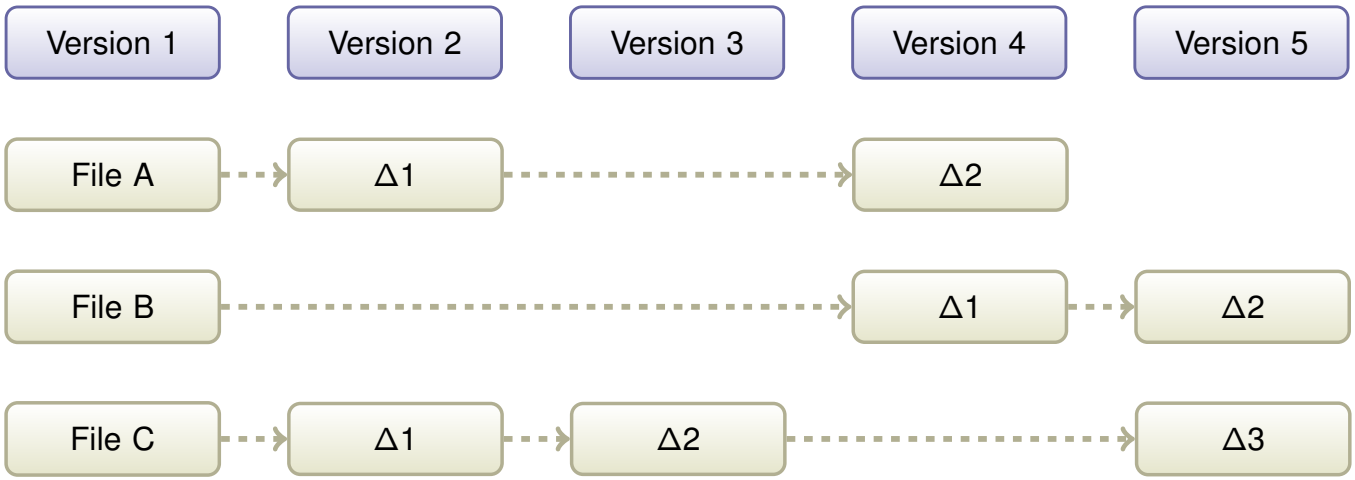
=> difficile de trouver un état cohérent du système (ajout de tag).

=> l'accès à une version nécessite de ré-appliquer les patchs.



La numérotation des commits dans *SVN*.

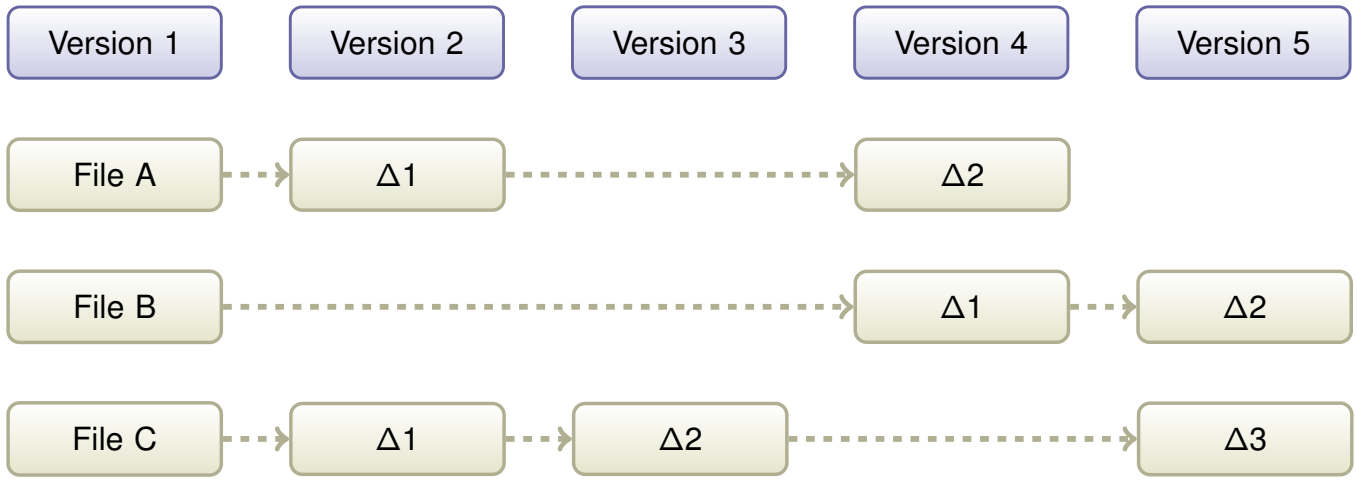
SVN utilise une numérotation globale sur l'ensemble des patchs :



La numérotation des commits dans *SVN*.

SVN utilise une numérotation globale sur l'ensemble des patchs :

=> chaque numéro de version correspond à un état cohérent ;

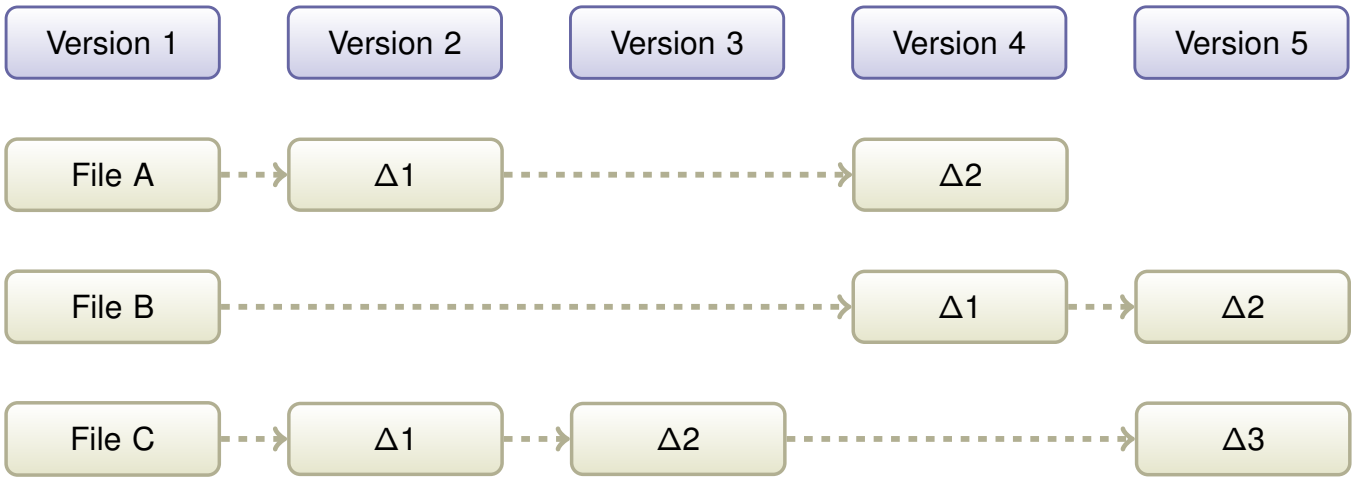


La numérotation des commits dans *SVN*.

SVN utilise une numérotation globale sur l'ensemble des patchs :

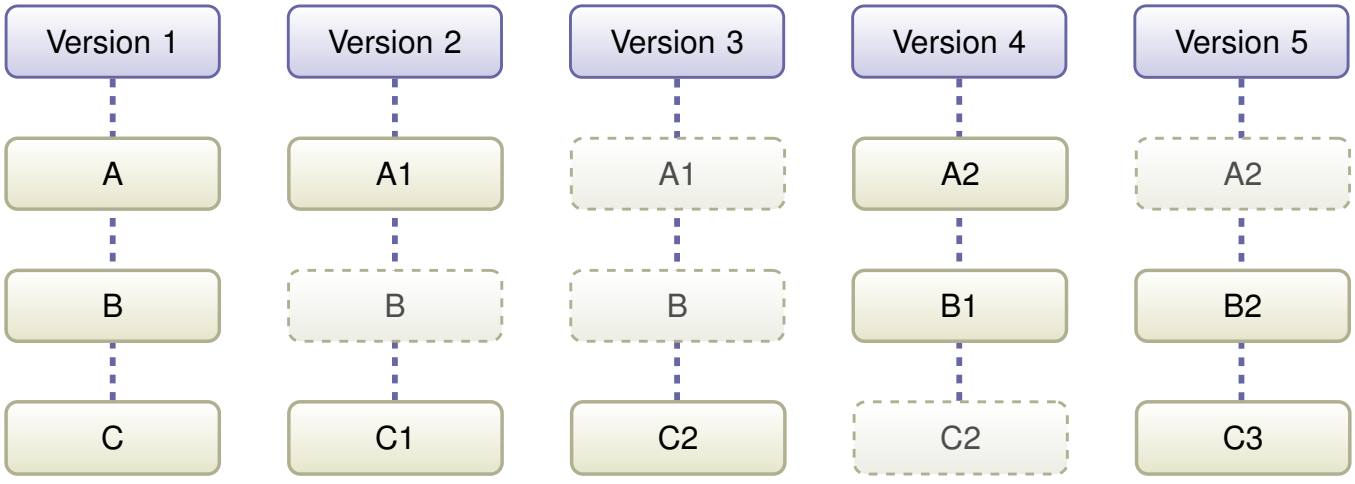
=> chaque numéro de version correspond à un état cohérent ;

=> l'accès à une version nécessite de ré-appliquer les patchs.



La numérotation des commits dans *Git*.

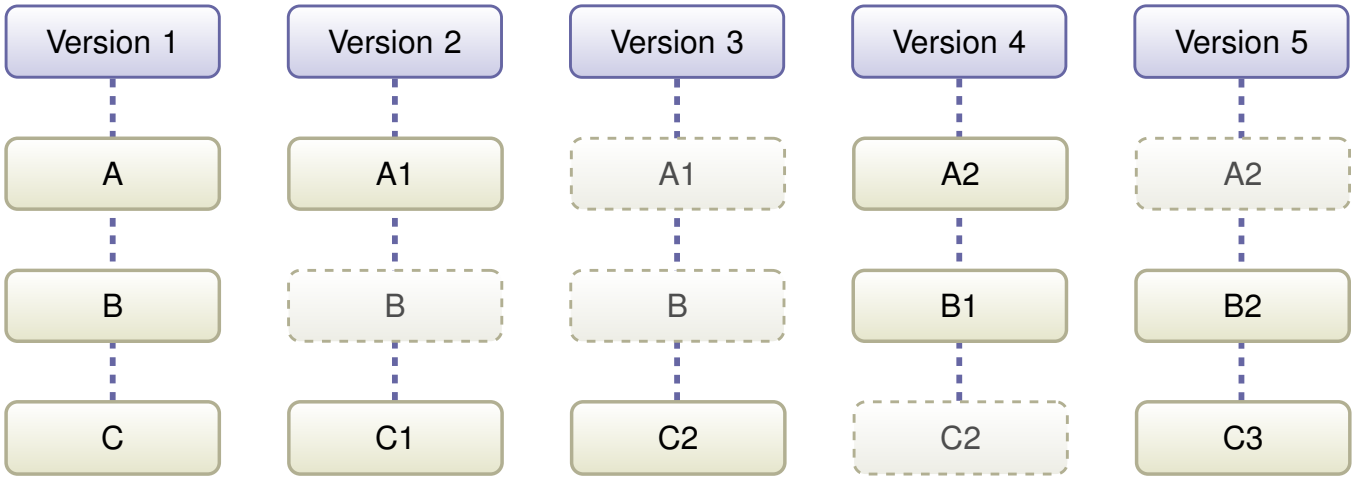
Git utilise une numérotation globale des versions des fichiers :



La numérotation des commits dans *Git*.

Git utilise une numérotation globale des versions des fichiers :

=> chaque numéro de version correspond à un état cohérent ;

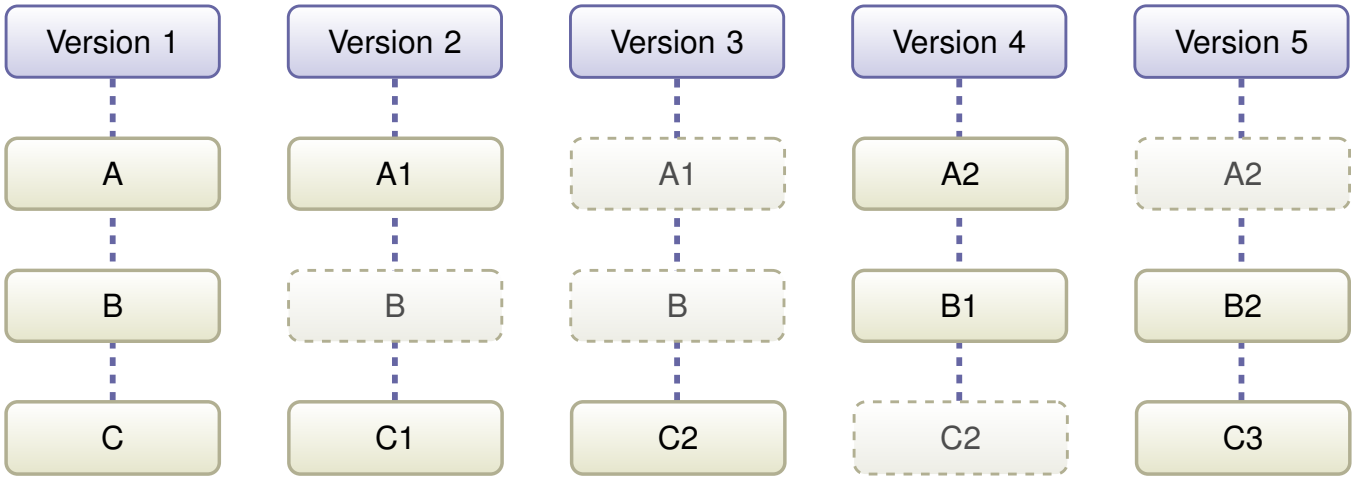


La numérotation des commits dans *Git*.

Git utilise une numérotation globale des versions des fichiers :

=> chaque numéro de version correspond à un état cohérent ;

=> accès direct aux versions du système.



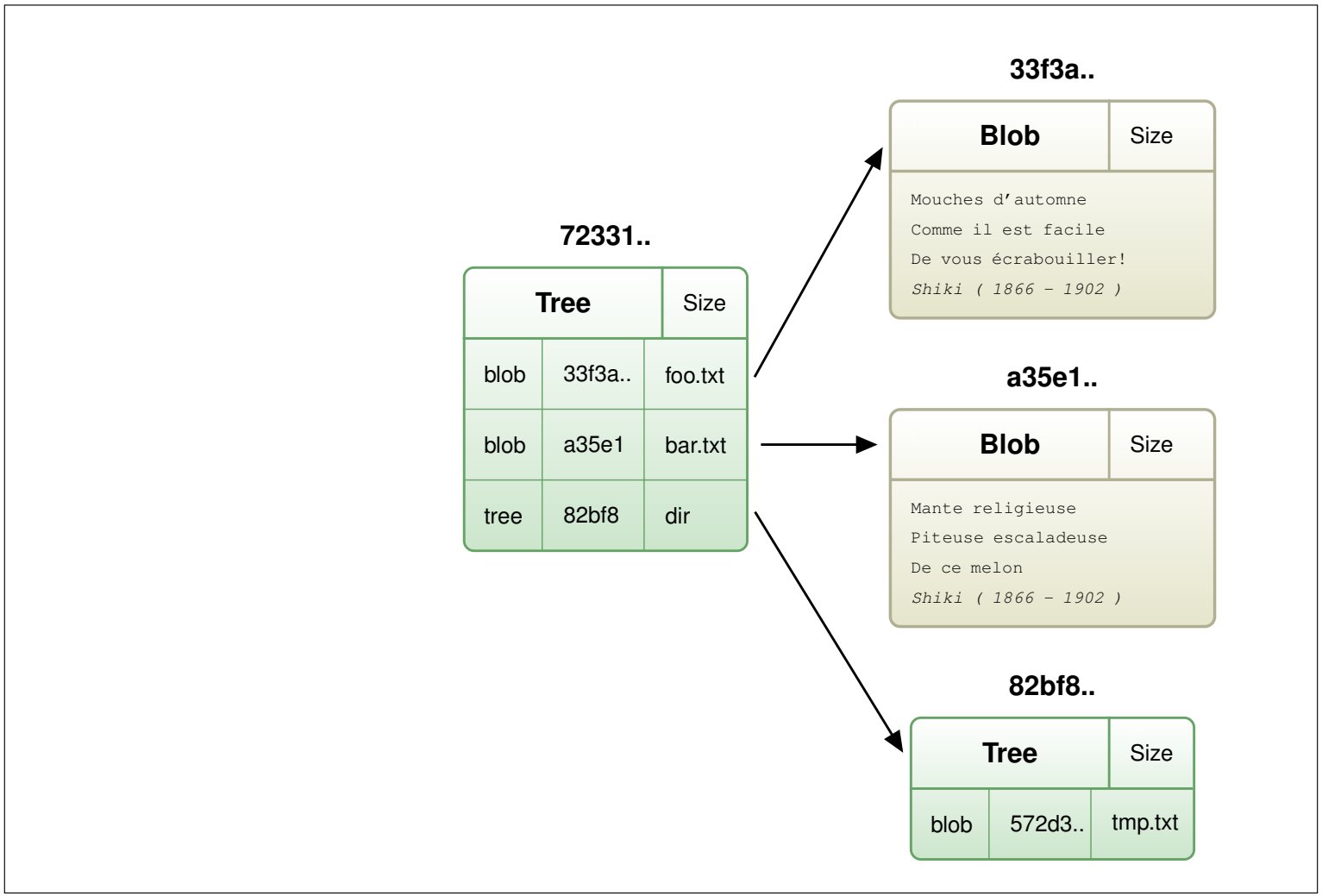
Les "Commits" : définitions

Définition

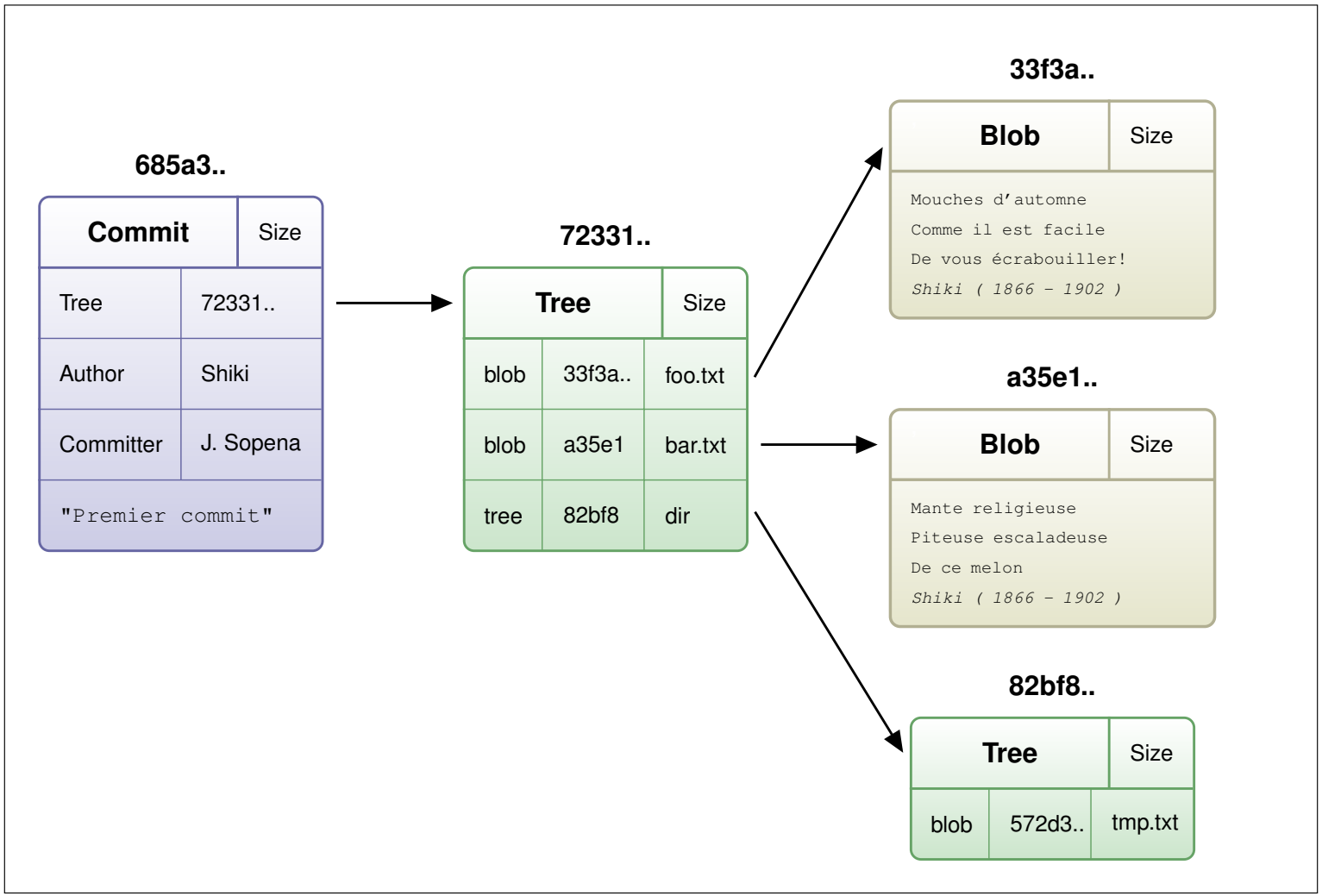
*Un **Commit** stocke l'état d'une partie du dépôt à un instant donné. Il contient :*

- *un pointeur vers un **Tree** dont on souhaite sauver l'état.*
- *un pointeur vers un ou plusieurs autres **Commits** pour constituer un historique.*
- *le nom d'un **auteur** et d'un **commiteur**.*
- *une description sous forme d'une chaîne de caractères.*

Les "Commits" : exemple



Les "Commits" : exemple



Outline

Les "Tags" : définitions

Définition

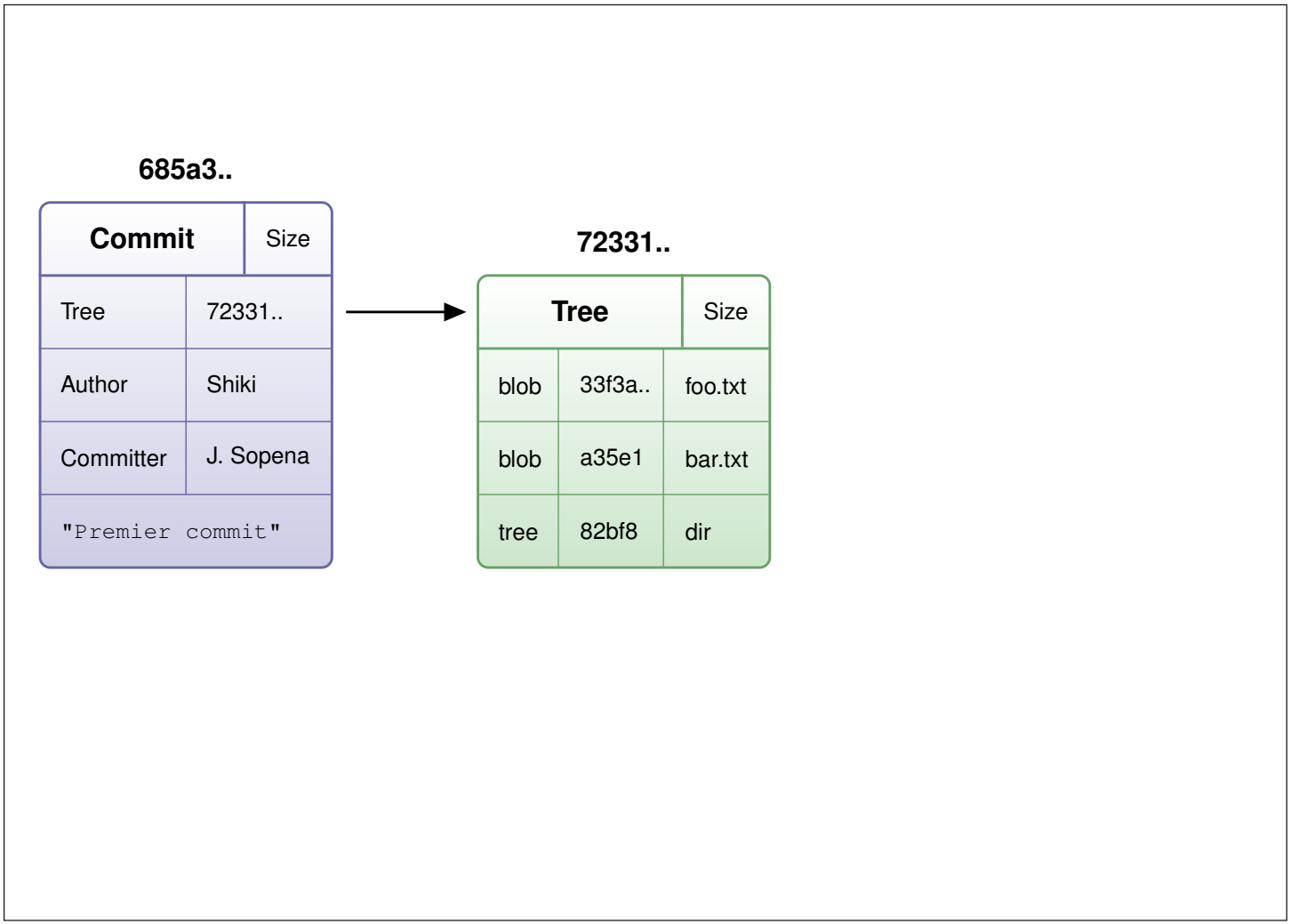
*Un **tag** permet d'identifier un des objets précédents à l'aide d'un nom. Il contient :*

- *un pointeur vers un **Blob**, un **Tree** ou un **Commit**.*
- *une signature.*

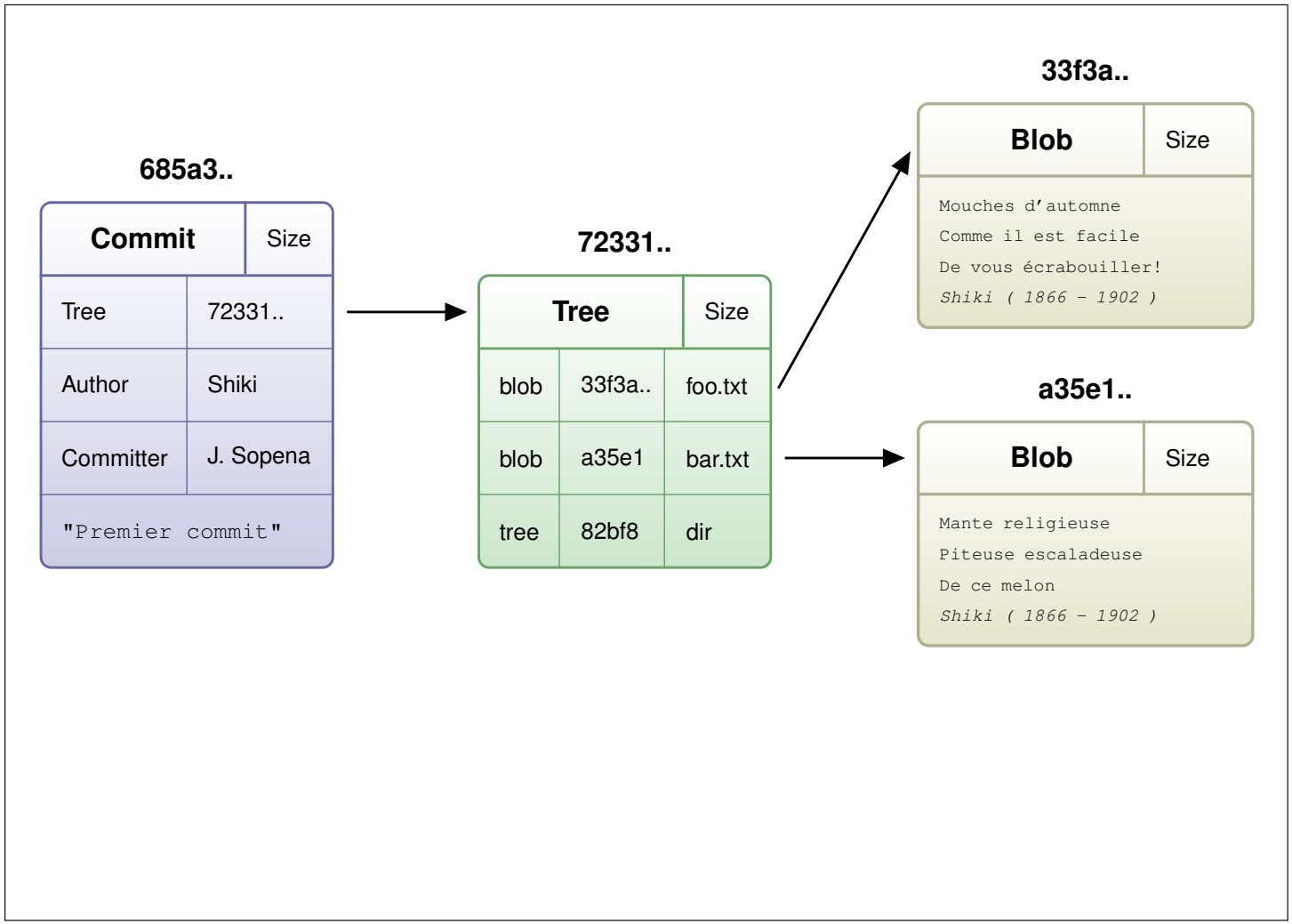
Outline

Graphe acyclique des objets.

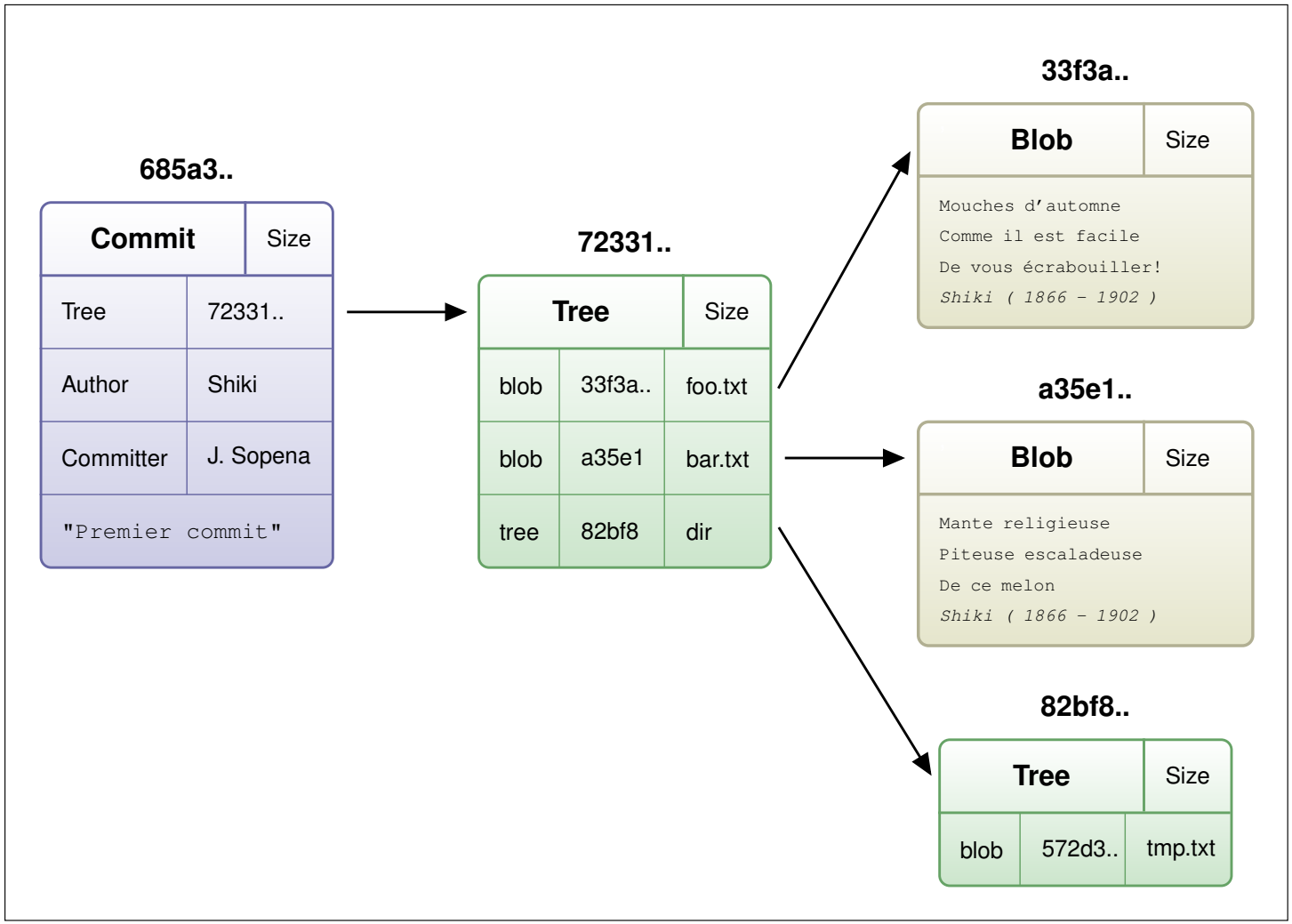
Graphe acyclique des objets.



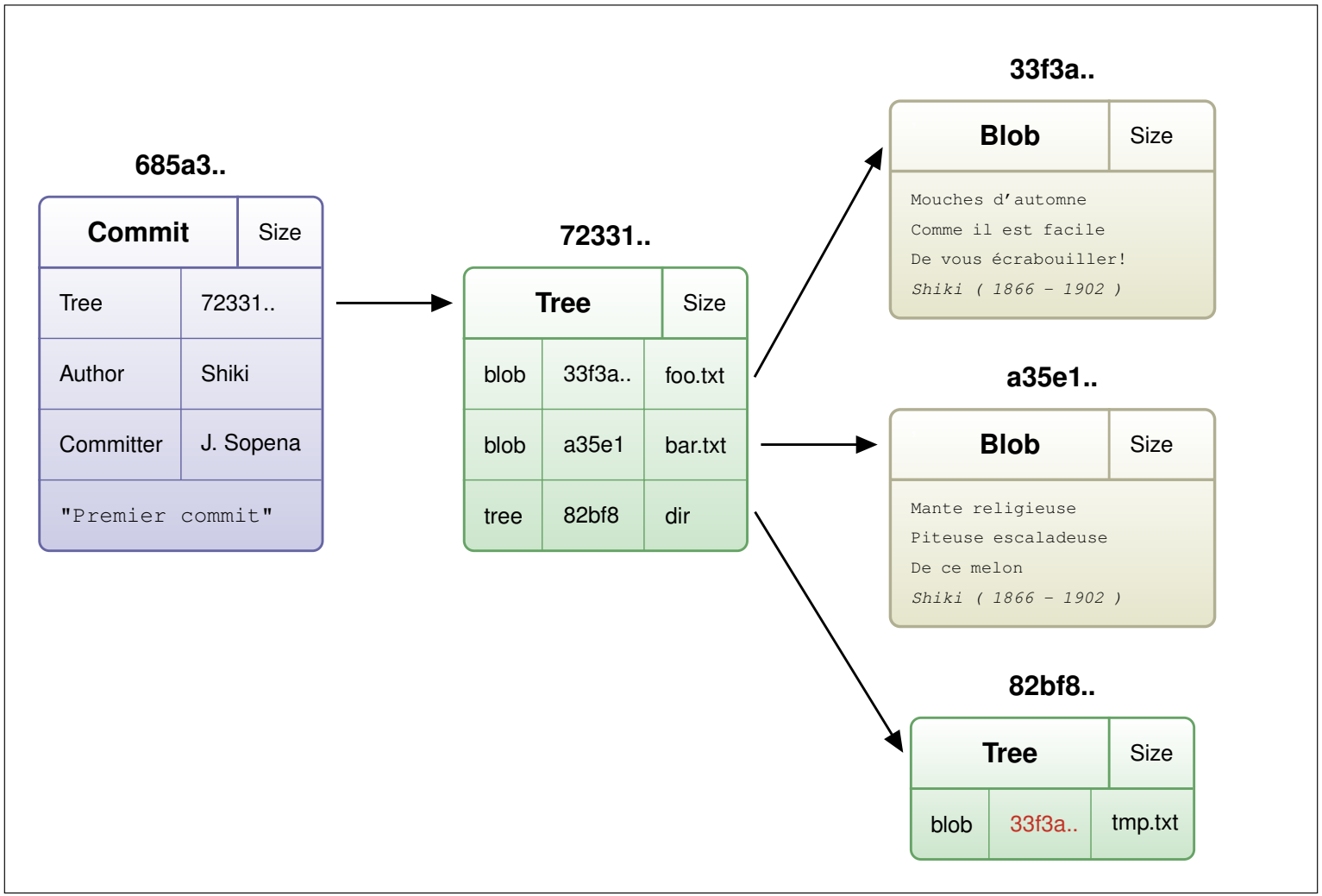
Graphe acyclique des objets.



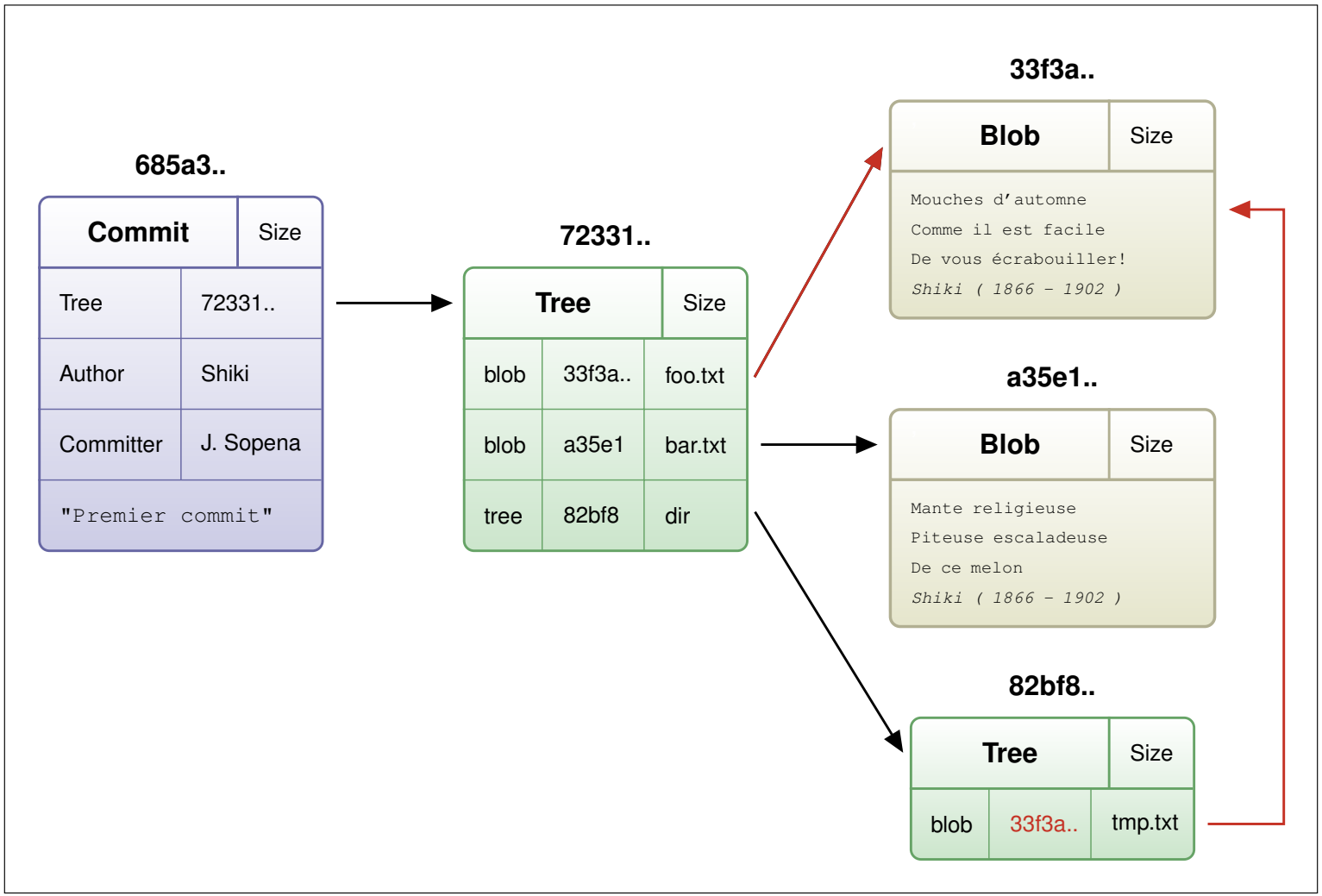
Graphe acyclique des objets.



Déduplication implicite



Déduplication implicite



Outline

Les commandes

Définition

Git *est un ensemble de commandes indépendantes permettant d'archiver, de rechercher et de publier des ensembles d'objets représentant l'état global de l'espace de travail versionné à un instant donné.*

Toutes ces commandes :

- portent un nom du type **git-<nom_de_la_commande>**.
- ont un équivalent sous la forme d'une option de la commnade **git <nom_de_la_commande>**.

Exemple

`git-add monFichier.txt` $\Longleftrightarrow$ `git add monFichier.txt`

Git c'est un ensemble de 145 commandes.

```
add                 describe            instaweb            peek-remote         show-branch
add-interactive     diff                log                 prune               show-index
am                  diff-files          lost-found          prune-packed        show-ref
annotate            diff-index          ls-files            pull                stage
apply               diff-tree           ls-remote           push                stash
archimport          difftool            ls-tree             quiltimport         status
archive             difftool-helper     mailinfo            read-tree           stripspace
bisect              fast-export         mailsplit           rebase              submodule
bisect-helper       fast-import         merge               rebase-interactive  svn
blame               fetch               merge-base          receive-pack        symbolic-ref
branch              fetch-tool          merge-file          reflog              tag
bundle              fetch-pack          merge-index         relink              tar-tree
cat-file            filter-branch       merge-octopus       remote              unpack-file
check-attr          fmt-merge-msg       merge-one-file      remote-curl         unpack-objects
check-ref-format    for-each-ref        merge-ours          repack              update-index
checkout            format-patch        merge-recursive     replace             update-ref
checkout-index      fsck                merge-resolve       repo-config         update-server-info
cherry              fsck-objects        merge-subtree       request-pull        upload-archive
cherry-pick         gc                  merge-tree          rerere              upload-pack
citool              get-tar-commit-id   mergetool           reset               var
clean               grep                mergetool-lib       rev-list            verify-pack
clone               gui (et gitk)       mktag               rev-parse           verify-tag
commit              gui-askpass         mktree              revert              web-browse
commit-tree         hash-object         mv                  rm                  whatchanged
config              help                name-rev            send-email          write-tree
count-objects       http-fetch          pack-objects        send-pack
cvsexportcommit     http-push           pack-redundant      sh-setup
cvsimport           imap-send           pack-refs           shell
cvsserver           index-pack          parse-remote        shortlog
daemon              init                patch-id            show
```

Git c'est un ensemble de 145 commandes.

Les commandes permettant de gérer un dépôt local

`add`

`branch`

`checkout`

`commit`

`config`

`diff`

`grep`

`help`

`init`

`log`

`mv`

`prune`

`rebase`

`reset`

`revert`

`rm`

`stash`

`status`

`tag`

Git c'est un ensemble de 145 commandes.

Les commandes permettant gérer des dépôts distants

add

am

branch

checkout

clone
commit

config

diff

fetch

grep

help

init

log

merge

mv

prune

pull
push

rebase

reset

revert
rm

stash
status

tag

Git c'est un ensemble de 145 commandes.

git-gui et gitk pour l'interface graphique.

add

am

branch

checkout

clone

commit

config

diff

fetch

grep

gui (et gitk)

help

init

log

merge

mv

prune

pull

push

rebase

reset

revert

rm

stash

status

tag

Git c'est un ensemble de 145 commandes.

Les commandes pour une utilisation avancée

add

am

bisect

blame
branch

checkout

clone
commit

config

diff

fetch

gc

grep
gui (et gitk)

help

init

log

merge

mv

prune

pull
push

rebase

repack

rerere
reset

revert
rm

stash
status

tag

Git c'est un ensemble de 145 commandes.

Les autres commandes au cas par cas

```
add                 daemon              index-pack          parse-remote        shell
 add-interactive     describe            init                patch-id            shortlog
 am                  diff                instaweb            peek-remote         show
 annotate            diff-files          log                 prune               show-branch
 apply               diff-index          lost-found          prune-packed        show-index
 archimport          diff-tree           ls-files            pull                show-ref
 archive             difftool            ls-remote           push                stage
 bisect              difftool-helper     ls-tree             quiltimport         stash
 bisect-helper       fast-export         mailinfo            read-tree           status
 blame               fast-import         mailsplit           rebase              stripspace
 branch              fetch               merge                                   submodule
 bundle              fetch-tool          merge-base          rebase-interactive  svn
 cat-file            fetch-pack          merge-file          receive-pack        symbolic-ref
 check-attr          filter-branch       merge-index         reflog              tag
                     fmt-merge-msg       merge-octopus       relink              tar-tree
check-ref-format     for-each-ref        merge-one-file      remote              unpack-file
 checkout            format-patch        merge-ours          remote-curl         unpack-objects
 checkout-index      fsck                merge-recursive     repack              update-index
 cherry              fsck-objects        merge-resolve       replace             update-ref
 cherry-pick         gc                  merge-subtree       repo-config
 citool                                  merge-tree          request-pull        update-server-info
 clean               get-tar-commit-id   mergetool           rerere               upload-archive
 clone               grep                mergetool-lib       reset               upload-pack
 commit              gui (et gitk)       mktag               rev-list            var
 commit-tree         gui-askpass         mktree              rev-parse           verify-pack
 config              hash-object         mv                  revert              verify-tag
 count-objects       help                name-rev            rm                  web-browse
 cvsexportcommit     http-fetch          pack-objects        send-email          whatchanged
 cvsimport           http-push           pack-redundant      send-pack           write-tree
 cvsserver           imap-send           pack-refs           sh-setup
```

Outline

Commandes basiques du dépôt

init : initialisation d'un dépôt.

clone : copie d'un dépôt existant (local ou distant).

fsck-object : pour valider un dépôt.

repack : fait des paquets de blobs pour l'efficacité.

prune : supprime les blobs uniques mais existants dans un paquet.

Création d'un nouveau dépôt

La création d'un dépôt, une commande simple :

Exemple

```
$ mkdir monprojet
$ cd monprojet
$ git init
defaulting to local storage area
```

Un petit exemple pour la création d'un dépôt Git sur un projet existant.

Exemple

```
$ cd monprojet
$ git init
$ git add .
$ git commit -a
defaulting to local storage area
```

Création d'un nouveau dépôt "serveur"

La création d'un dépôt "serveur" se fait avec la commande :

git clone –bare <URL>

Ce dépôt n'a :

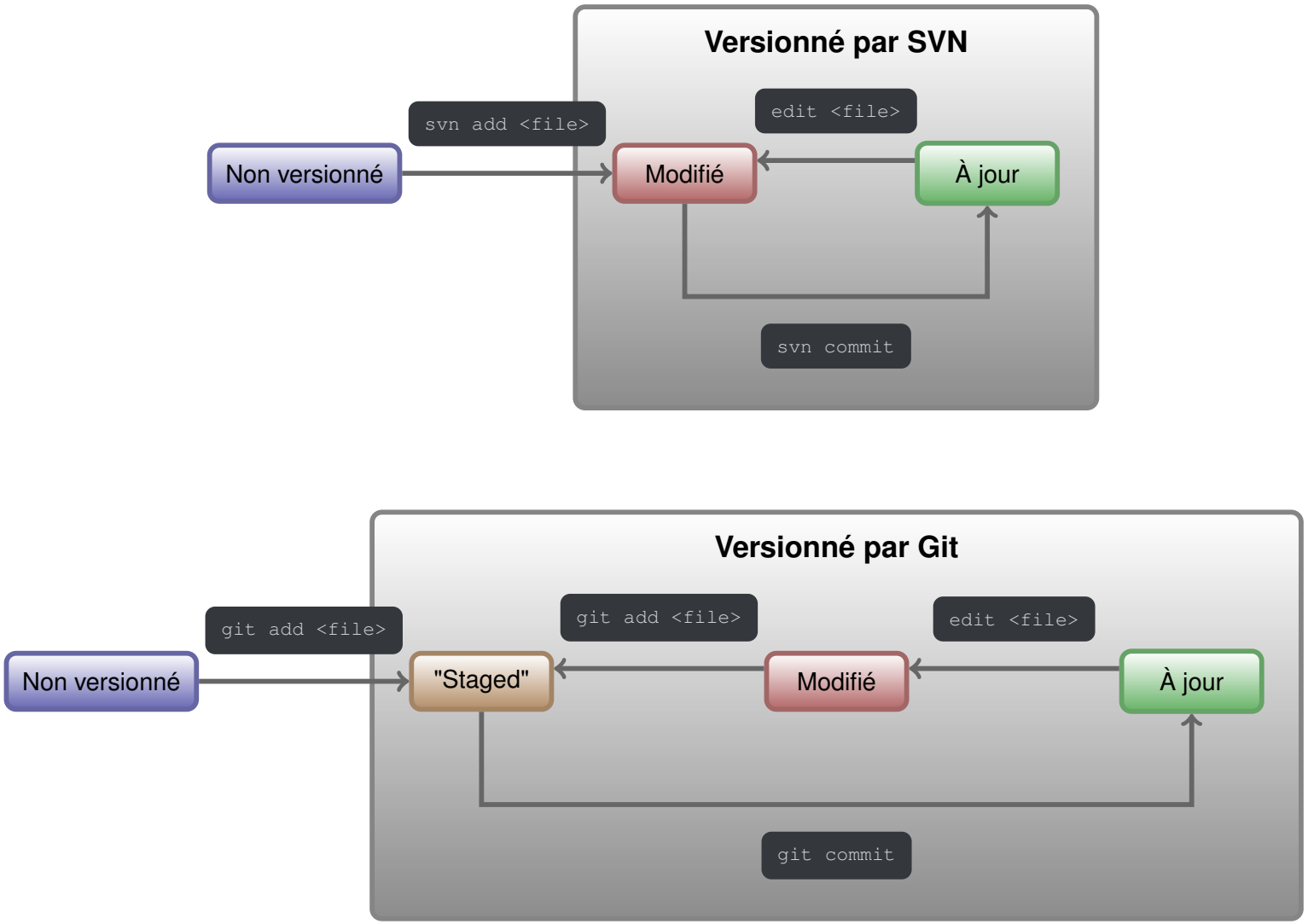
- pas de fichier juste l'historique
- pas de répertoire xxx/.git mais directement un xxx.git/
- Pas de "remote" (origin)

Exemple

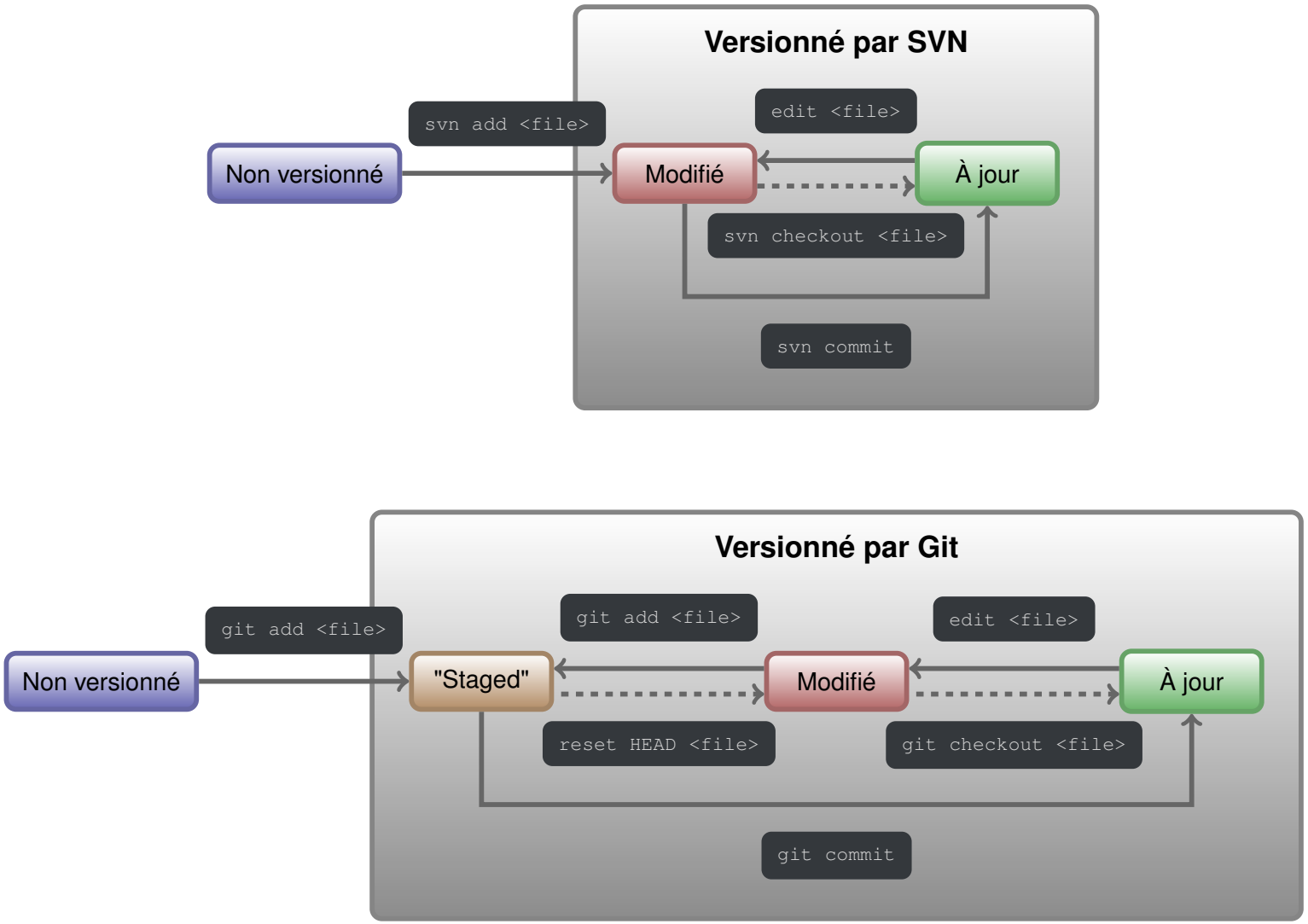
```
$ git clone --bare <URL>
$ cat xxx.git/config
      [core]
          ....
          bare = true
          ....
```

Outline

Commit : différence avec cvs/svn.



Commit : différence avec cvs/svn.



Commit : les commandes.

add : ajoute dans l'index un fichier à commiter dans son état actuel.

commit : enregistre dans le dépôt local les modifications qui ont été ajoutées dans l'index par une commande **add**.

reset HEAD : supprime la référence d'un fichier de l'index ajouté par une commande **add**.

Exemple

```
$ echo "Premier fichier" > foo.txt
$ git add foo.txt
$ git commit -m "Description de ce commit"
```

Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init
```

Étude des objets générés par un exemple simple.

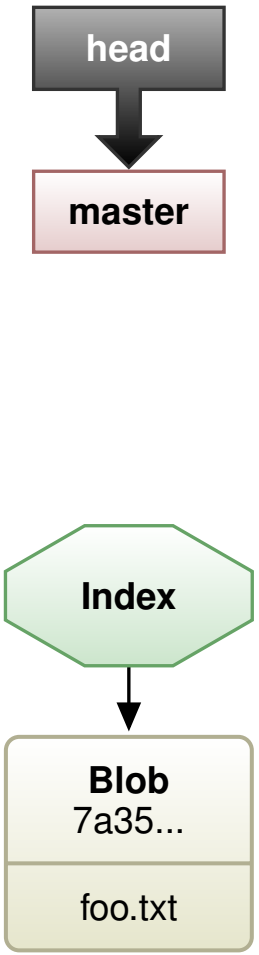
```
mkdir project
cd project
git init
```

Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt
```

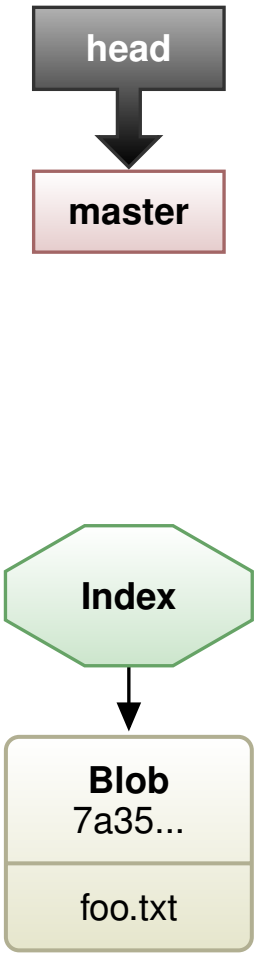
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt
```



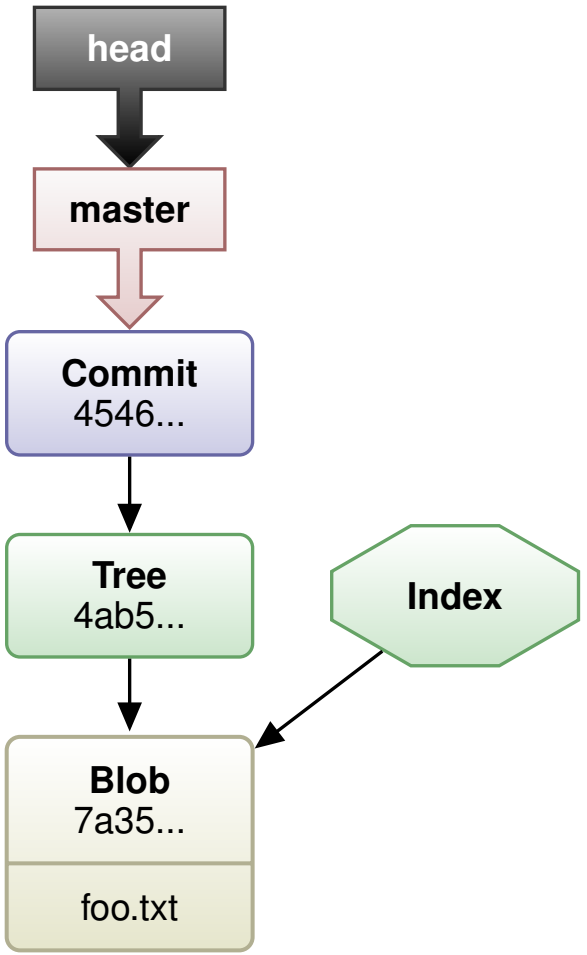
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"
```



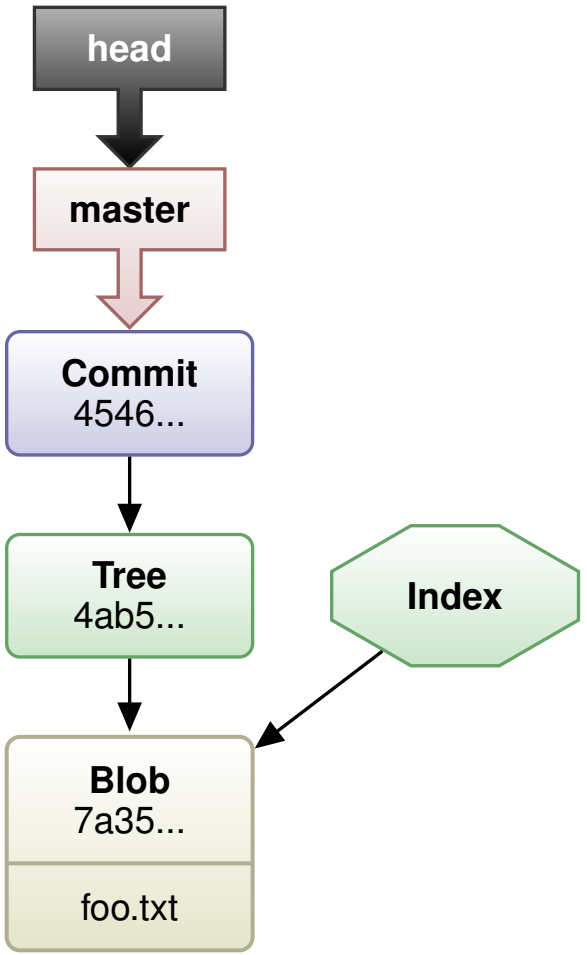
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"
```



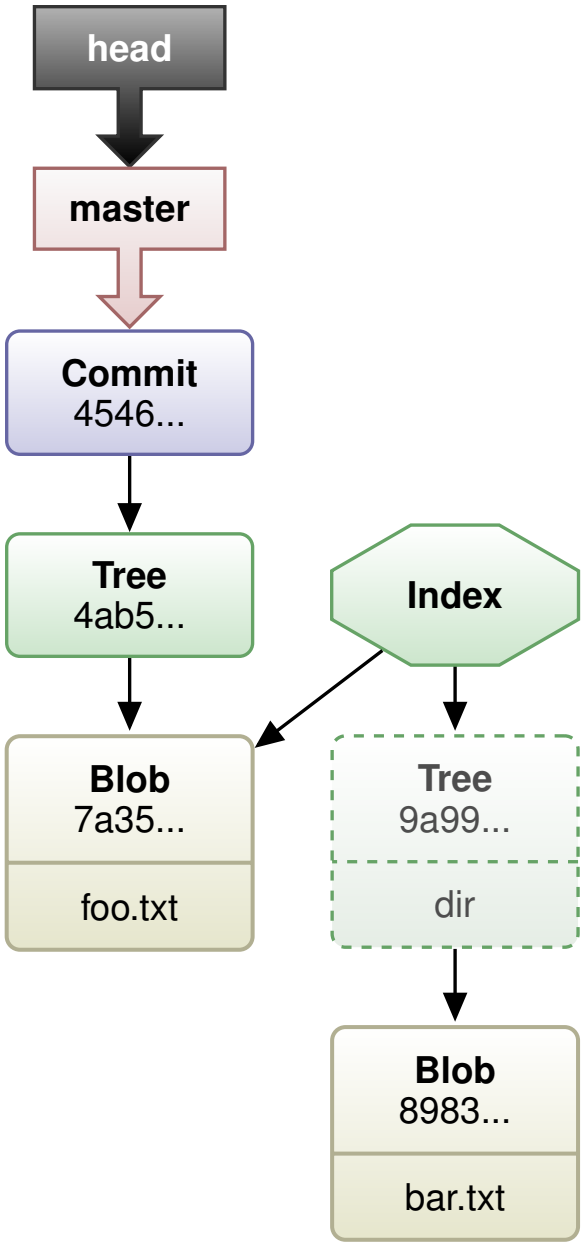
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt
```



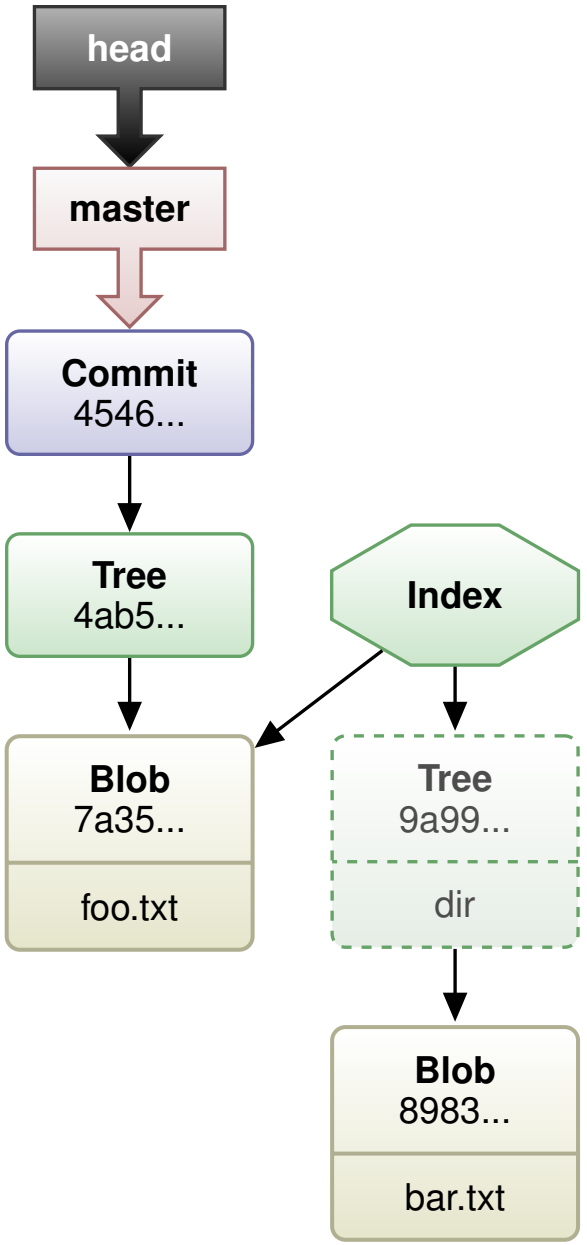
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt
```



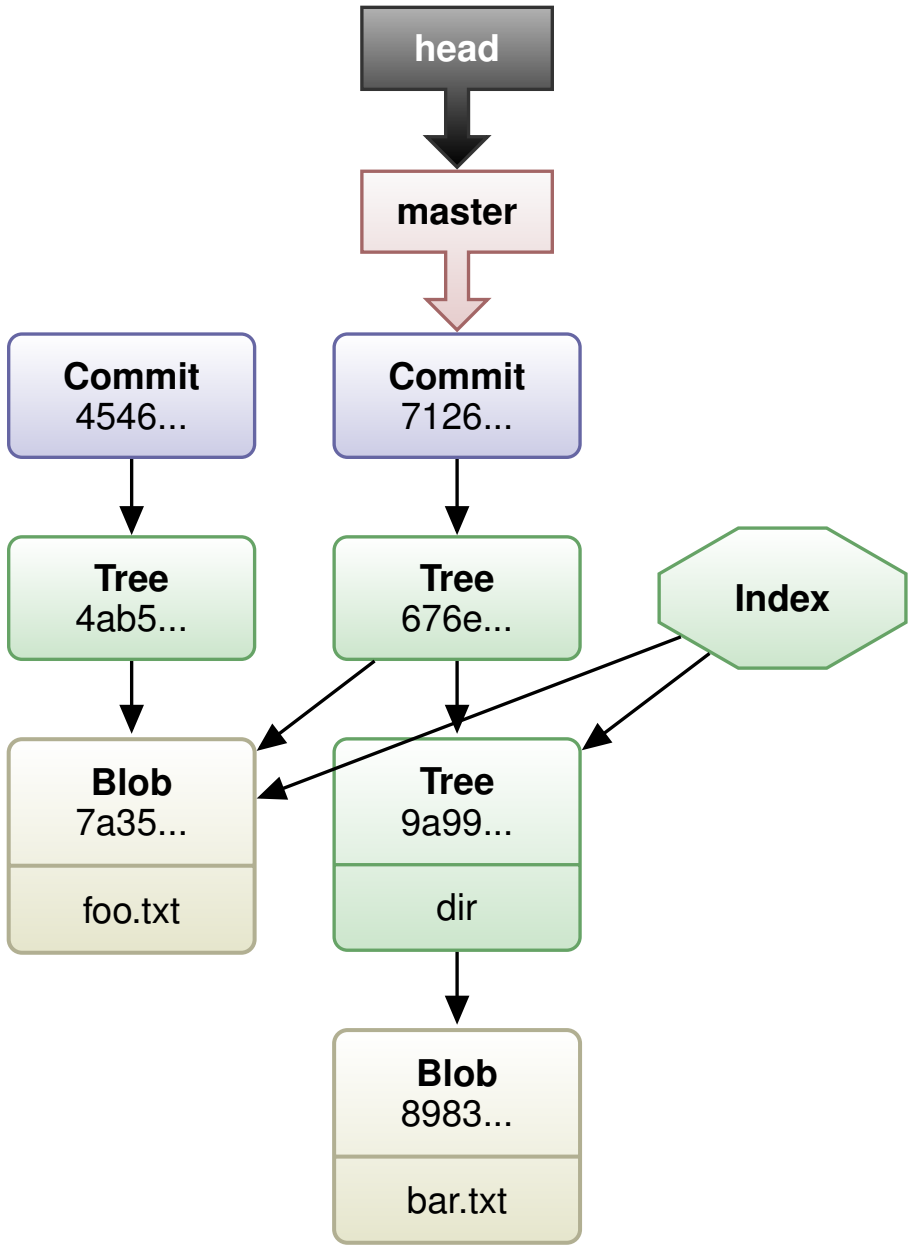
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"
```



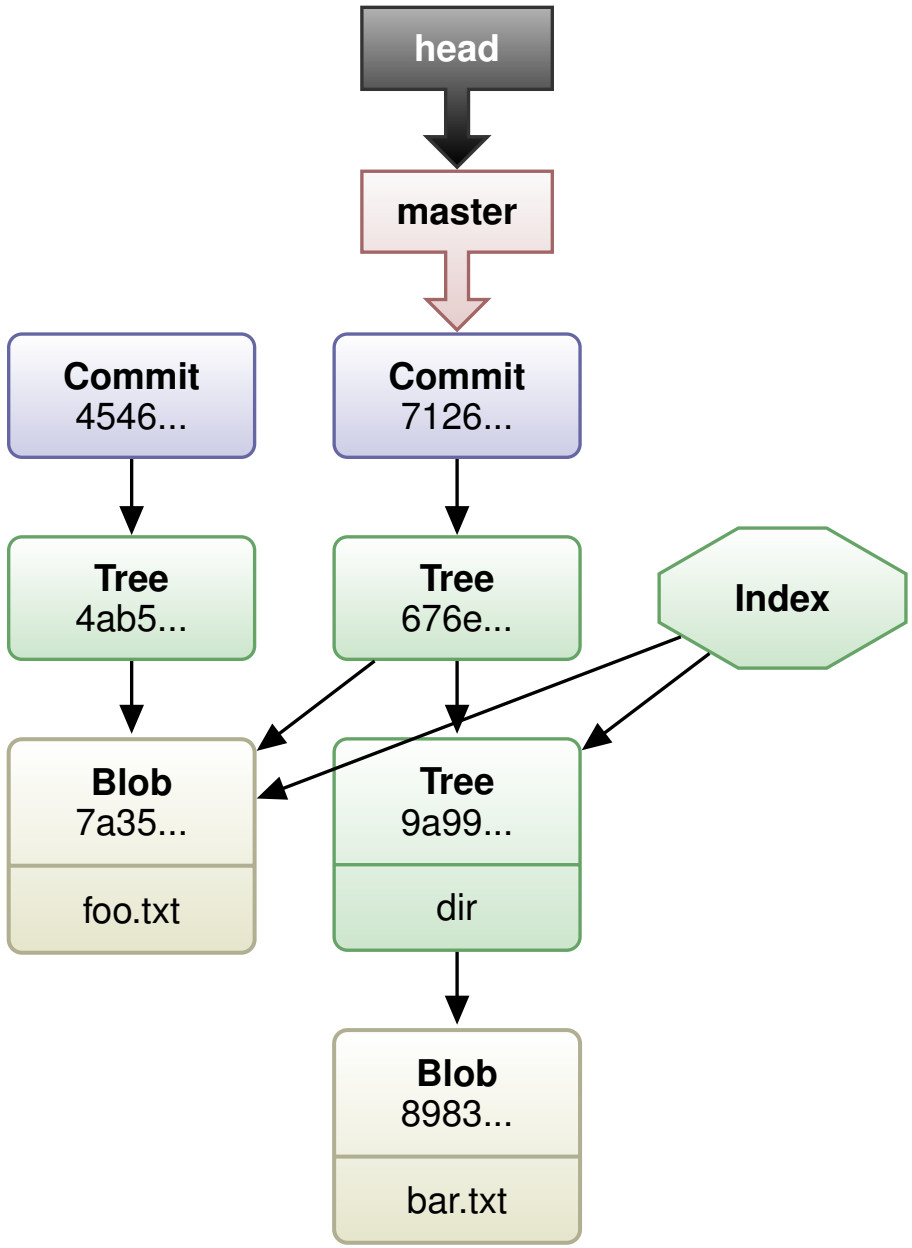
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"
```



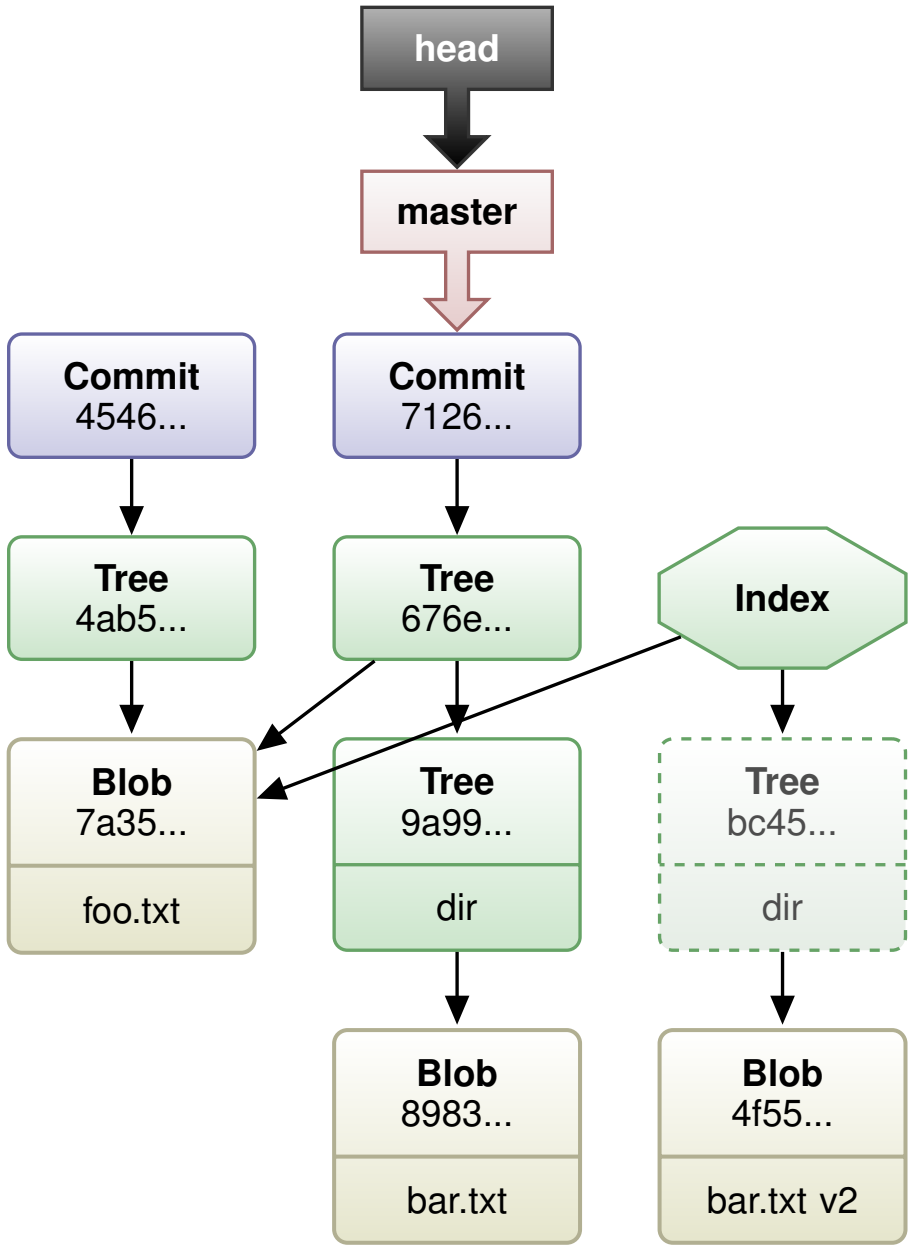
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt
```



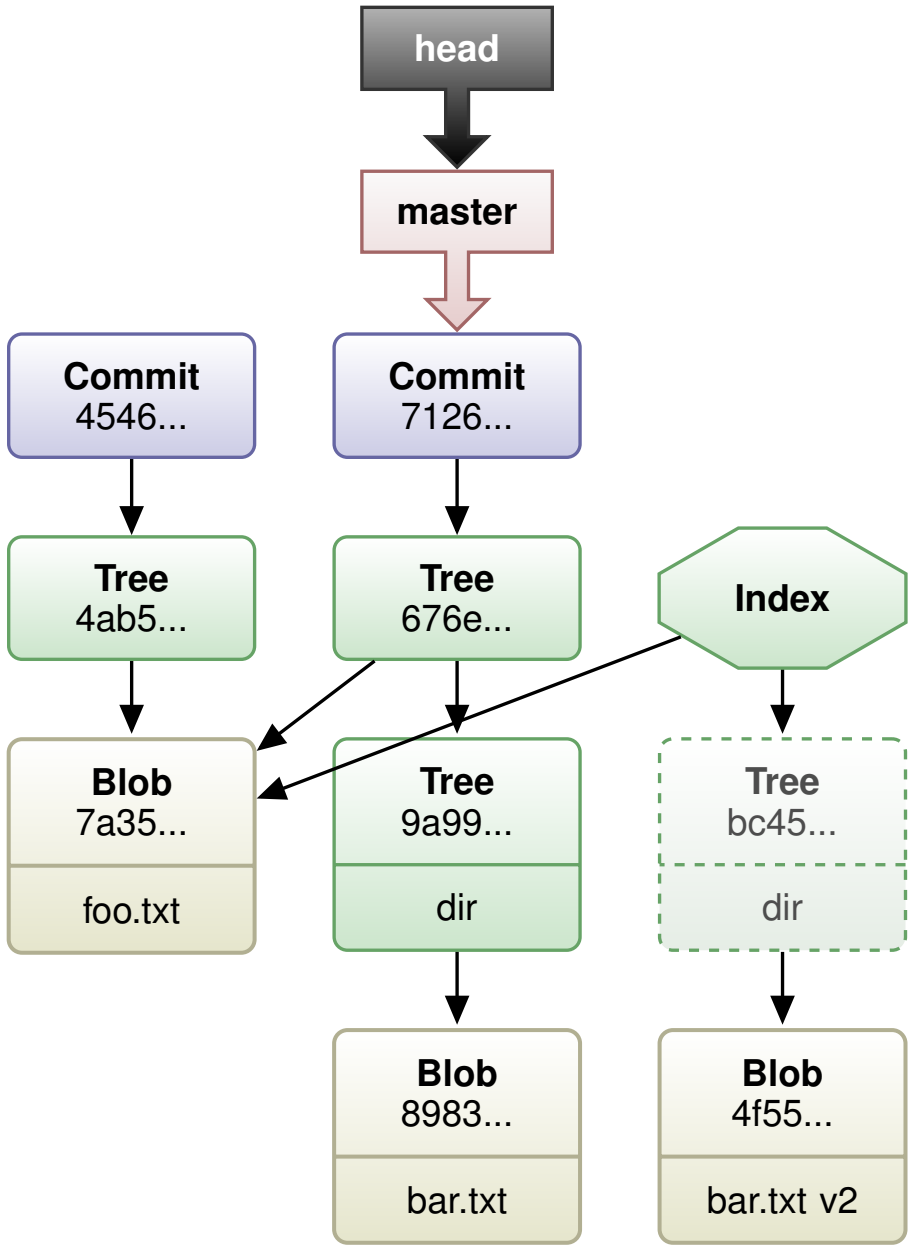
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt
```



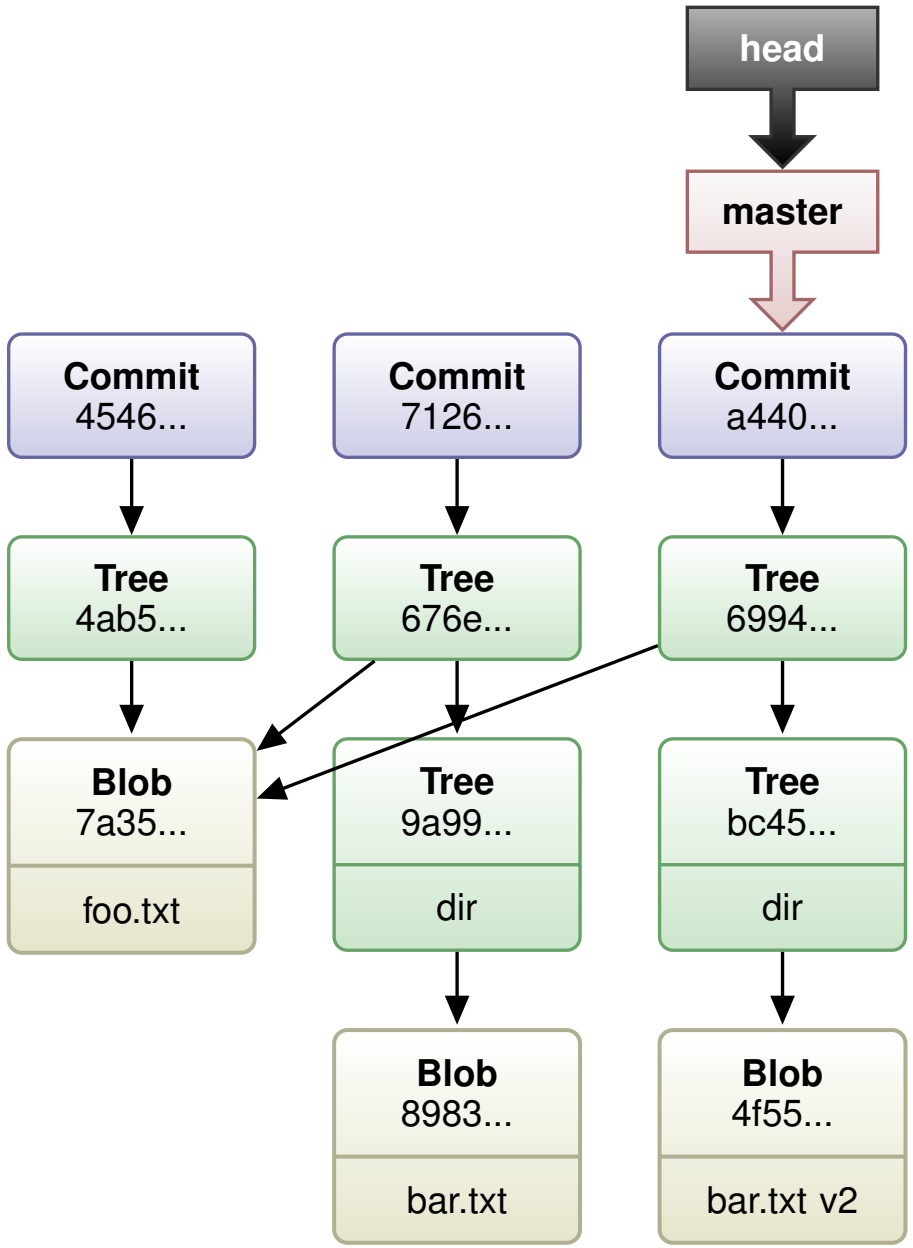
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



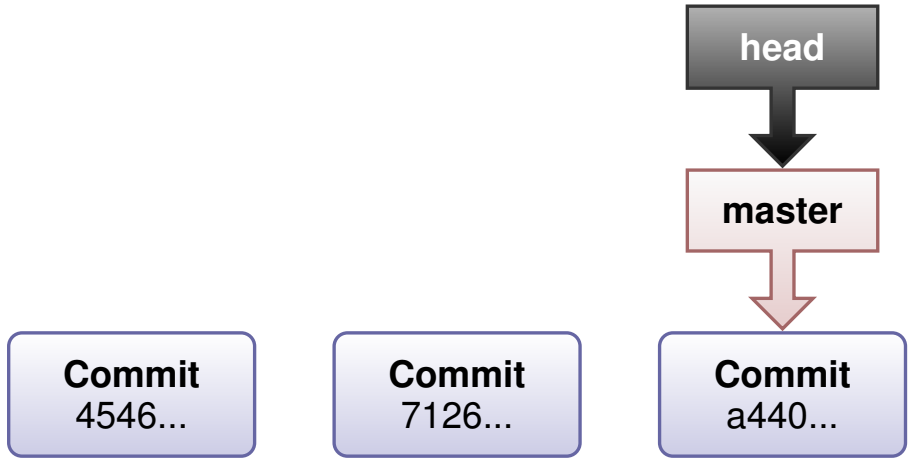
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



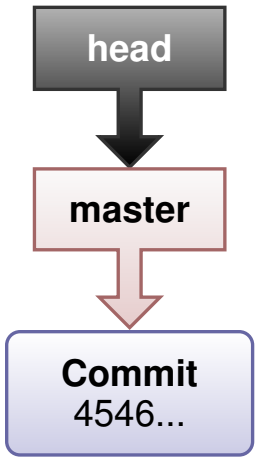
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



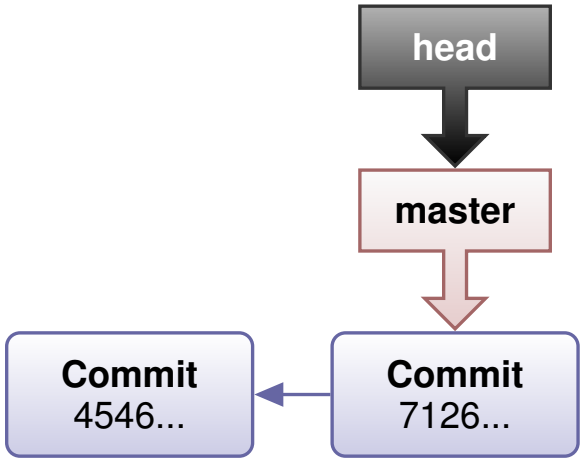
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



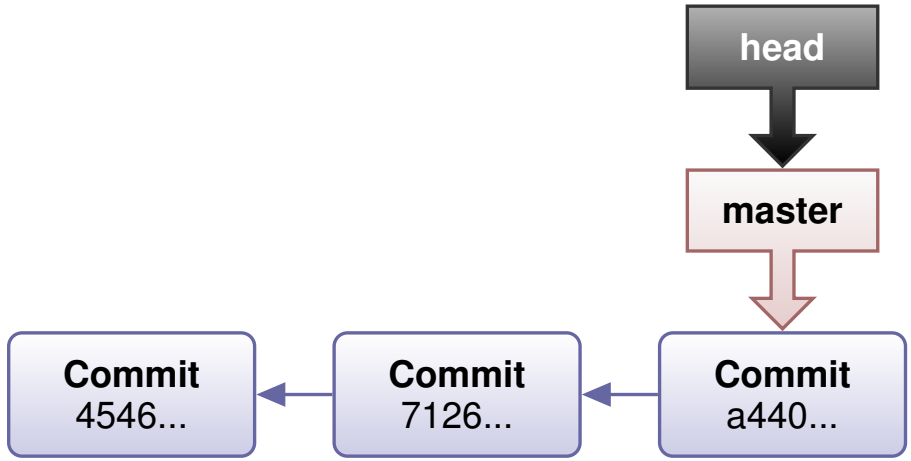
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



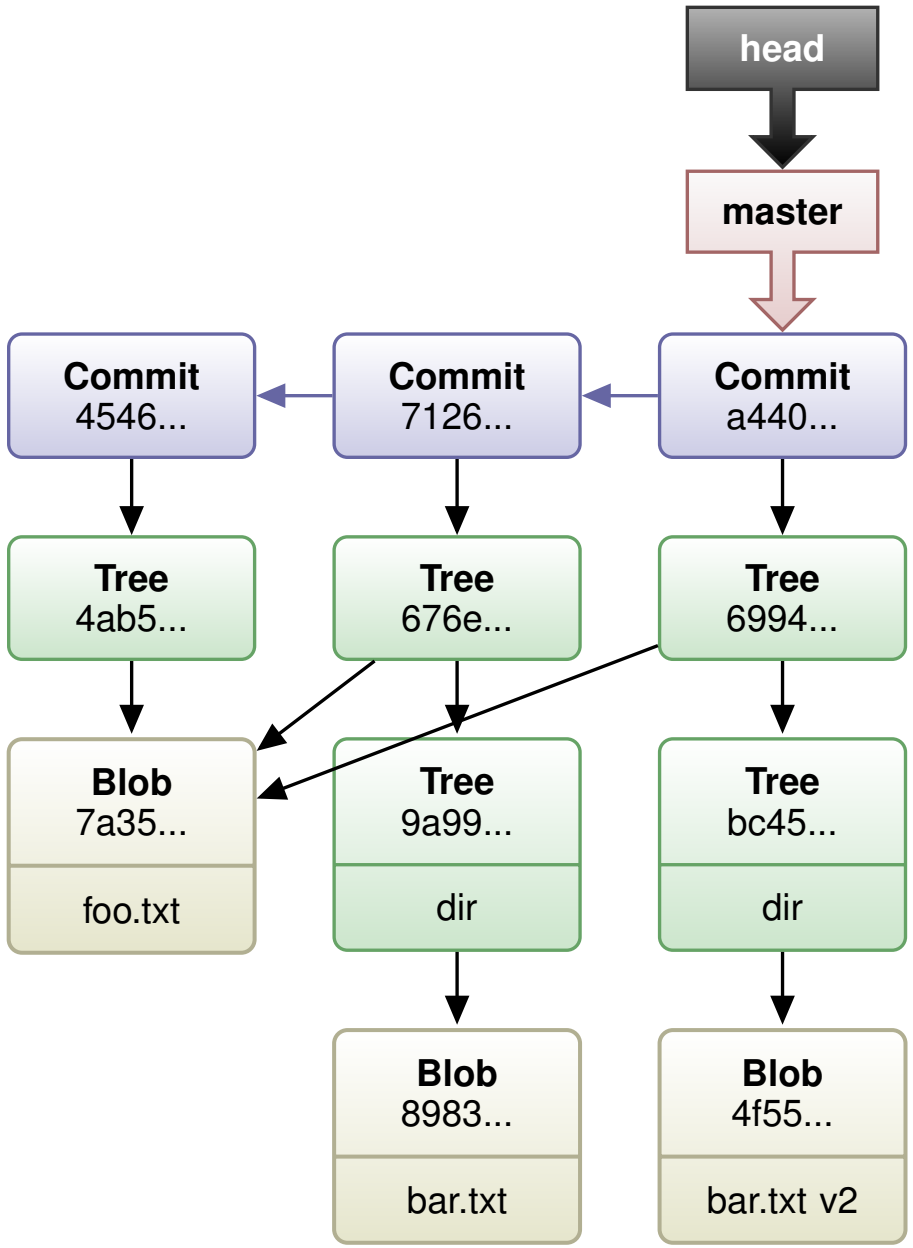
Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



Étude des objets générés par un exemple simple.

```
mkdir project
cd project
git init

echo "toto" > foo.txt
git add foo.txt

git commit -m "Add foo.txt"

mkdir dir
echo "titi" > dir/bar.txt
git add dir/bar.txt

git commit -m "Add dir/bar.txt"

echo "tutu" » dir/bar.txt
git add dir/bar.txt

git commit -m "Modif dir/bar.txt"
```



Outline

Branche : les commandes.

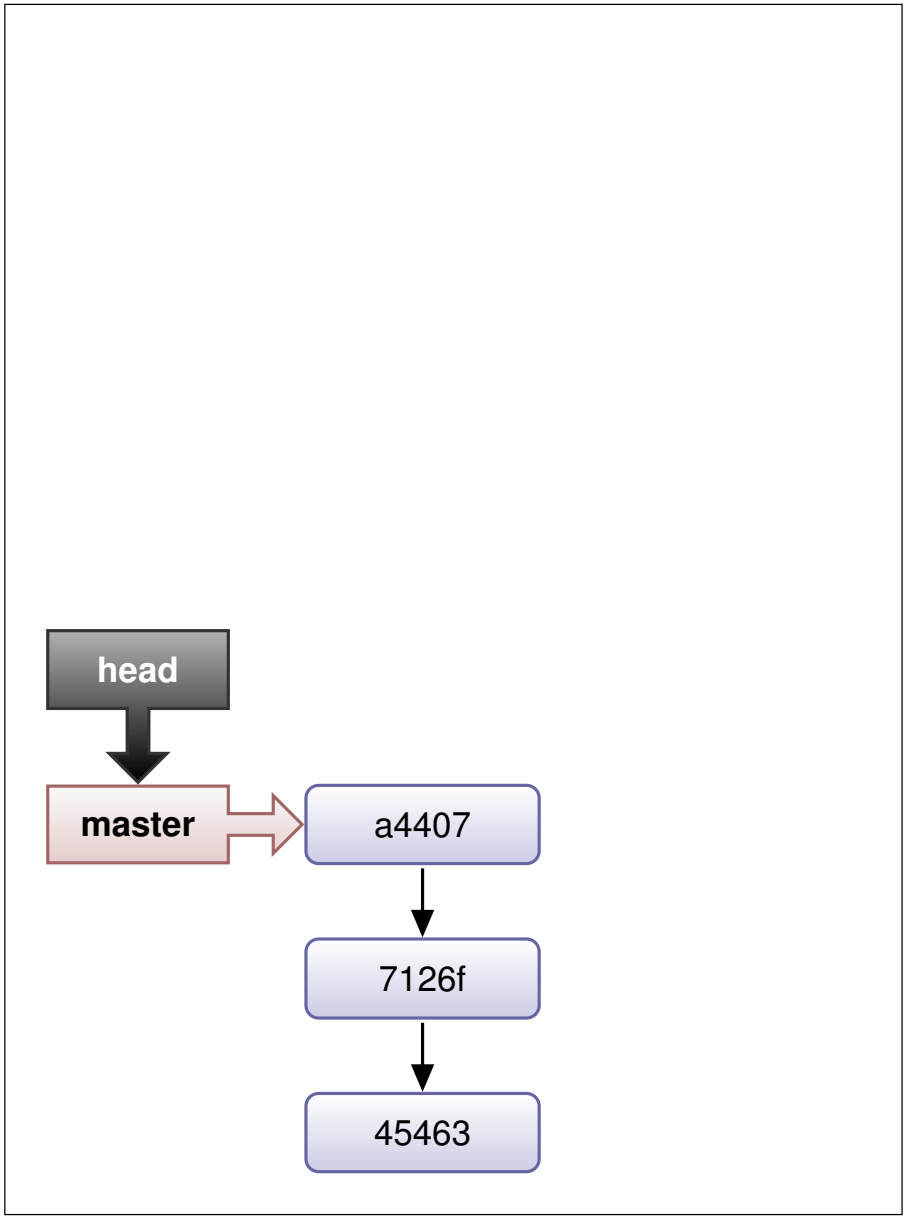
branch : liste les branches avec une * pour la branche active.
branch <nom> : crée une nouvelle branche <nom>.
branch -m : permet de renommer une branche.
branch -d : permet de supprimer une branche.
checkout : change (ou/et crée) de branche active.
show-branch : affiche les branches et leurs commits.

Exemple

```
$ git branch
    * master
$ git branch maBranche
$ git branch
      maBranche
    * master
$ git checkout maBranche
$ git branch
    * maBranche
      master
```

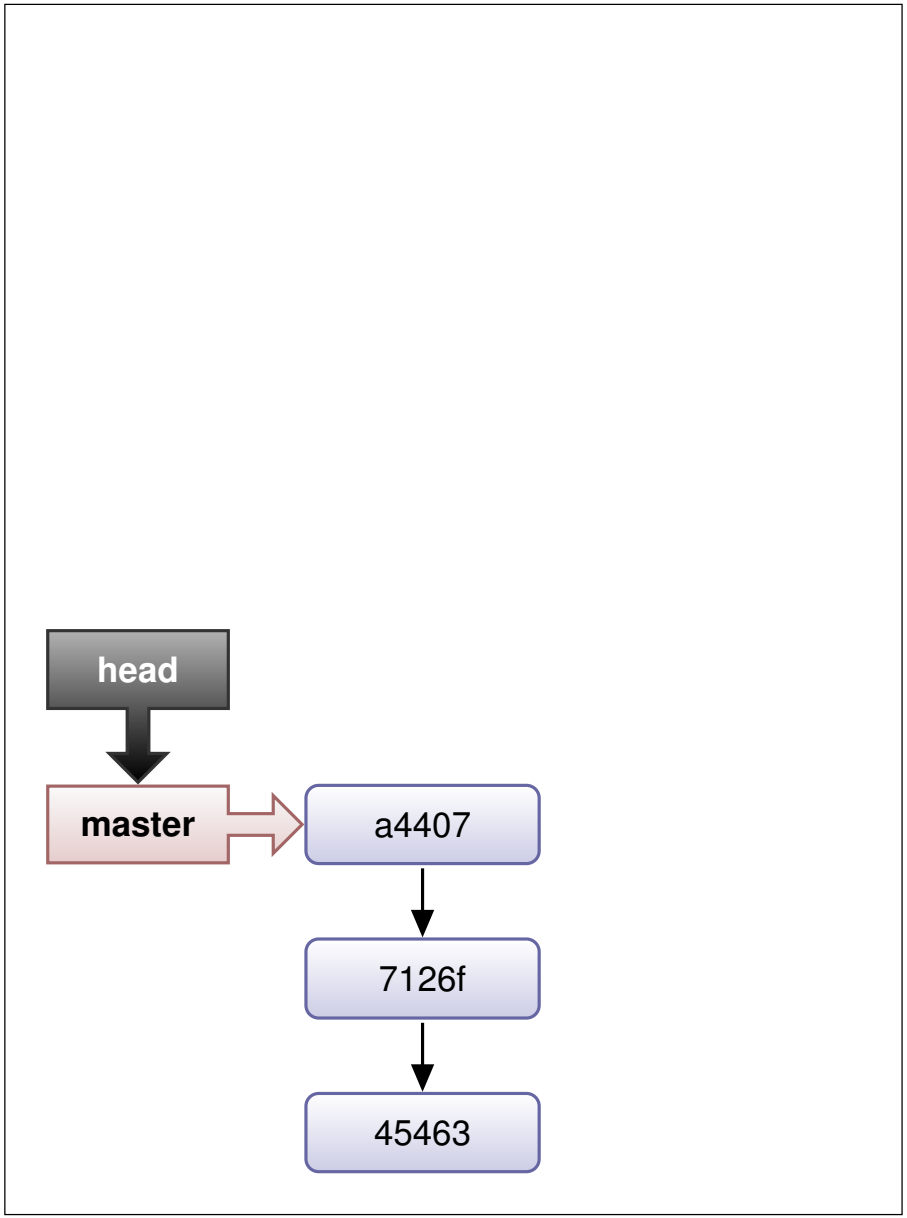
Branche : structure interne des commits.

```
ls
 foo.txt dir
```



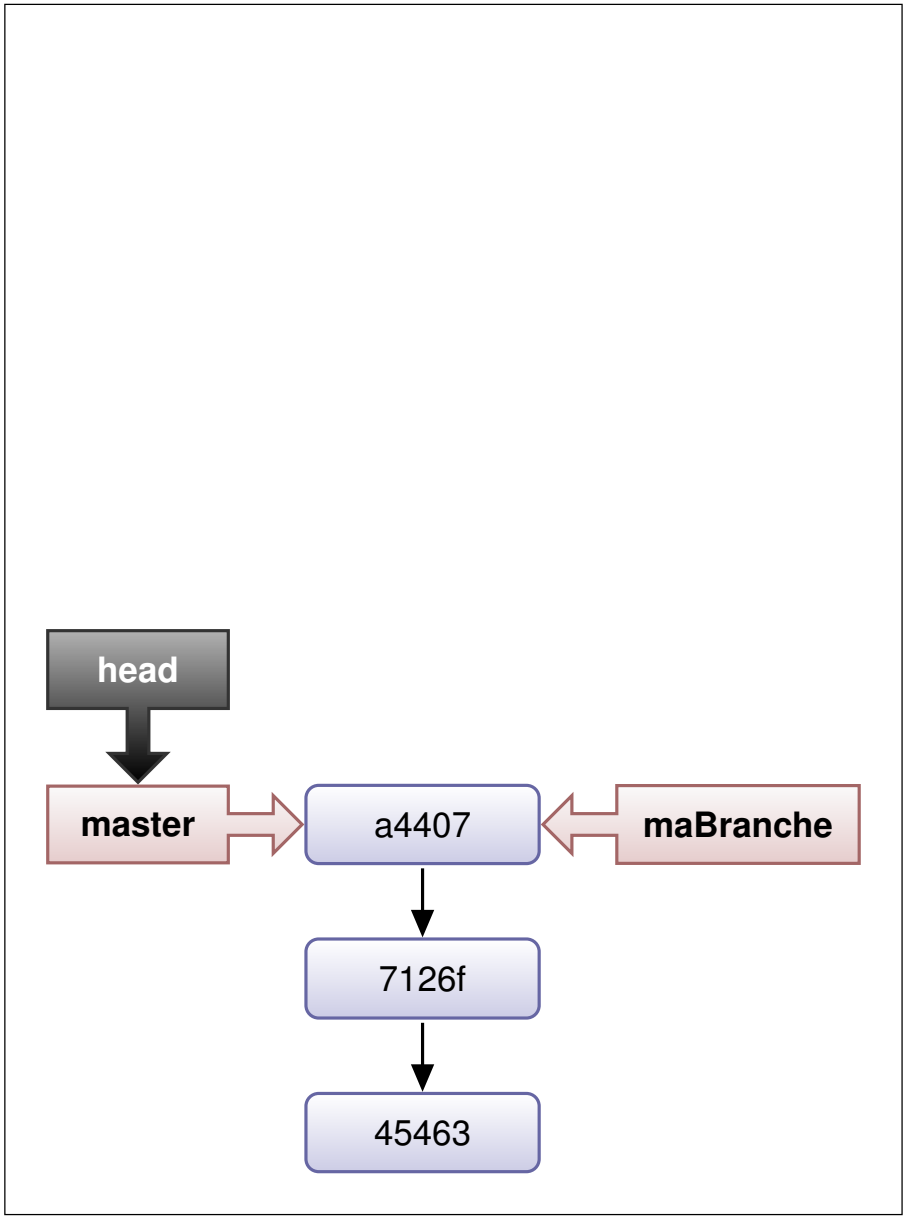
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche
```



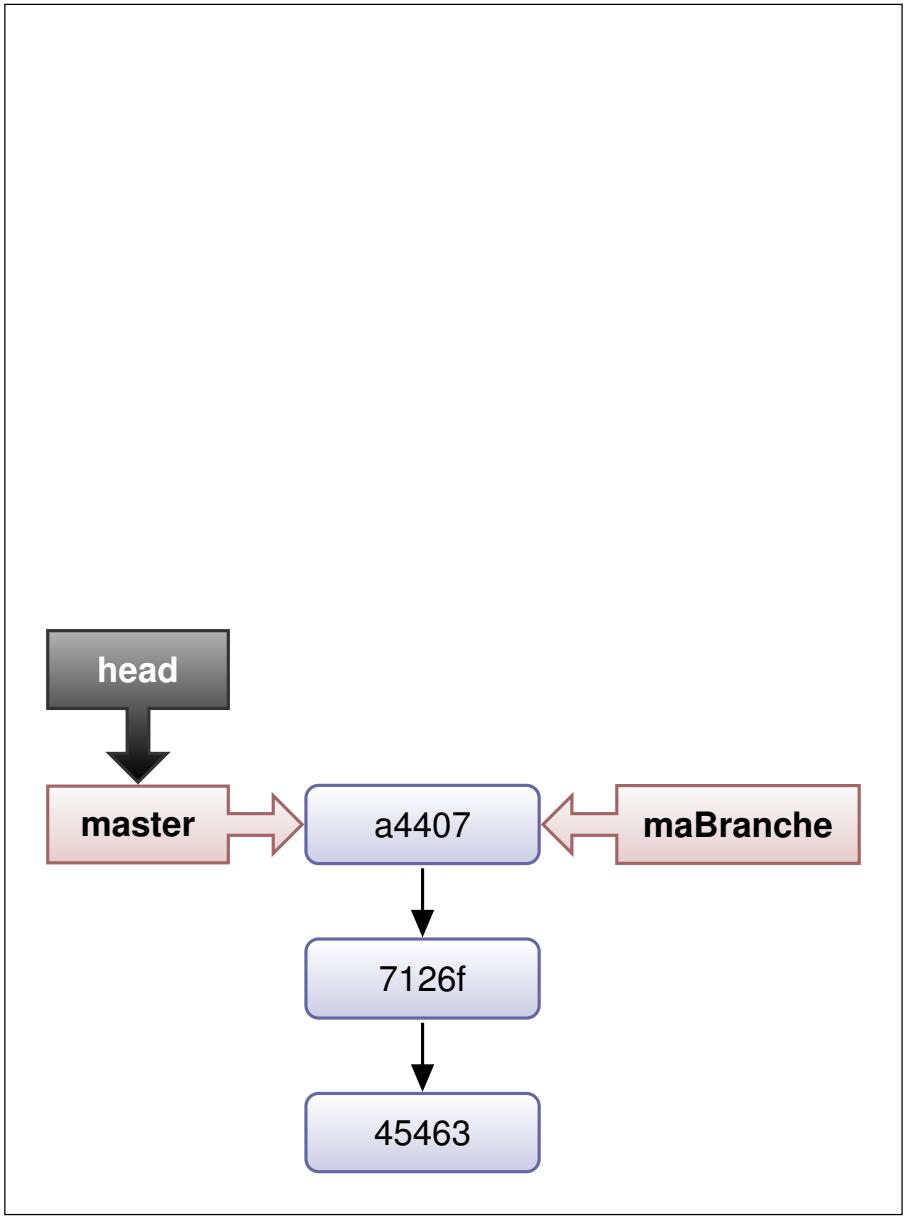
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche
```



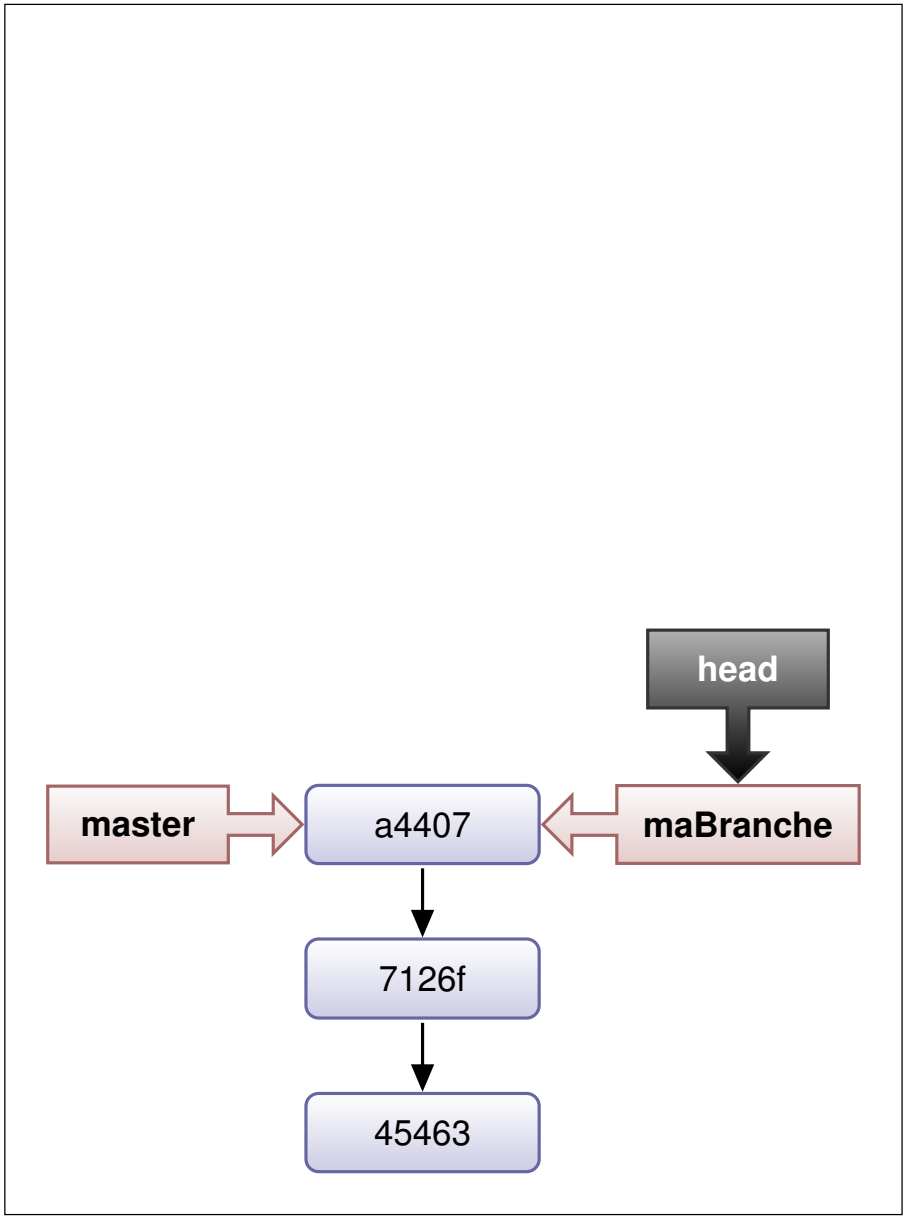
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



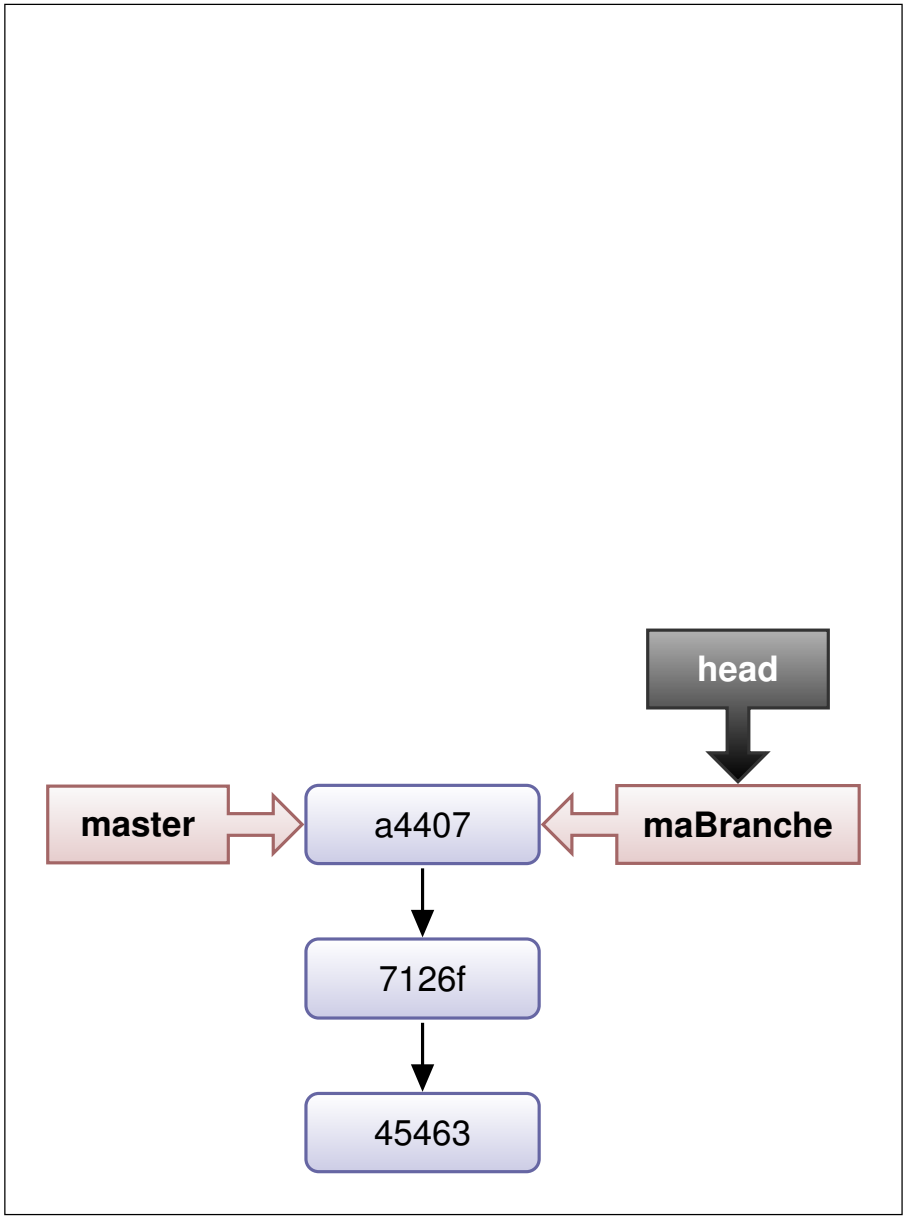
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



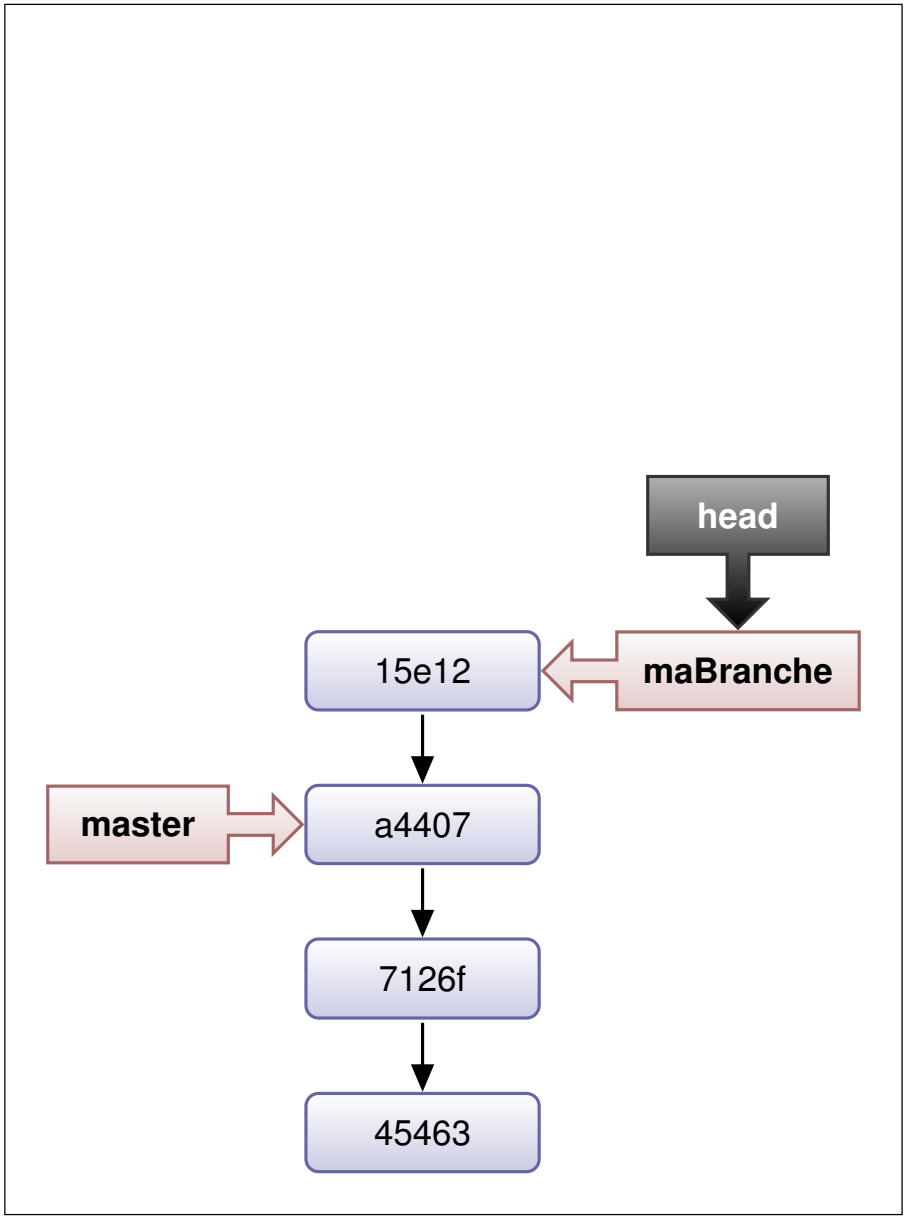
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



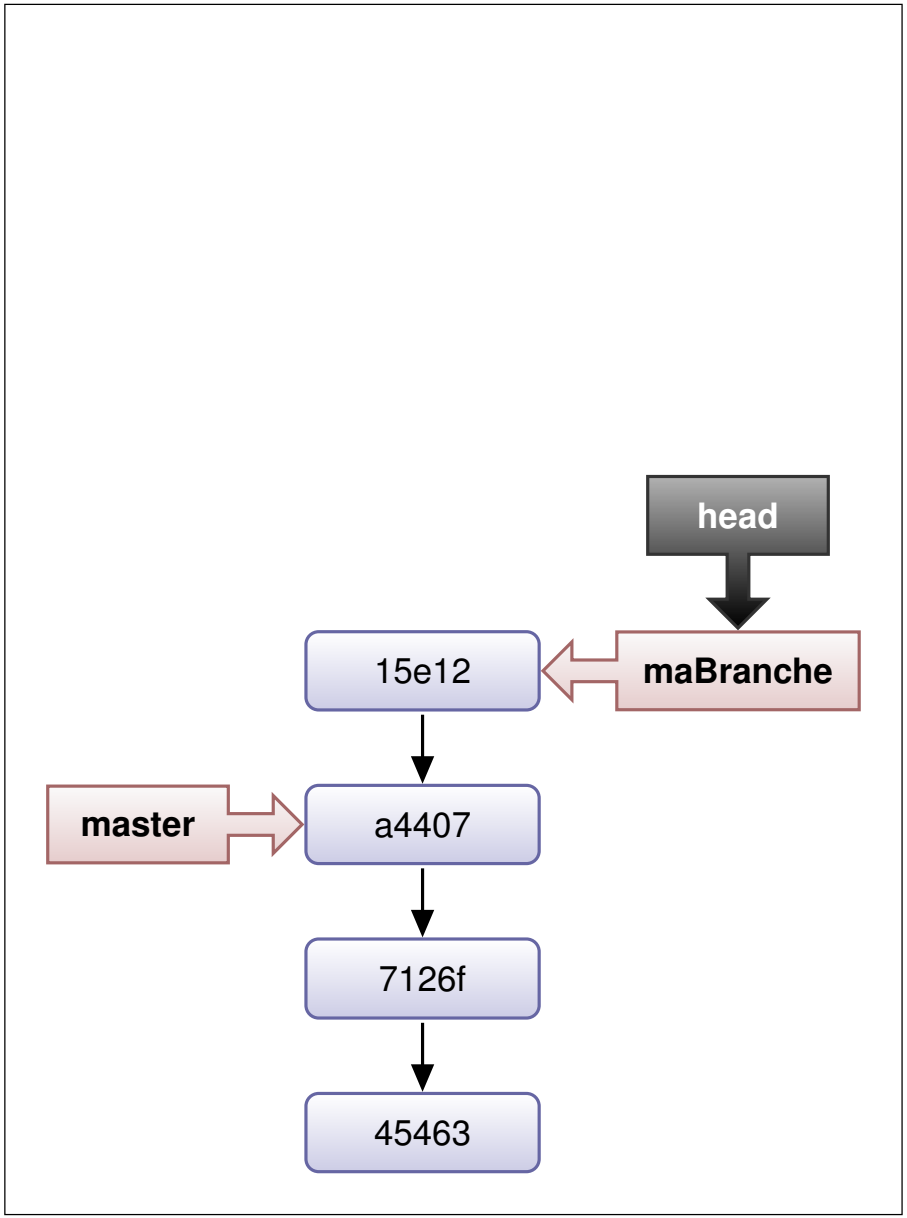
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



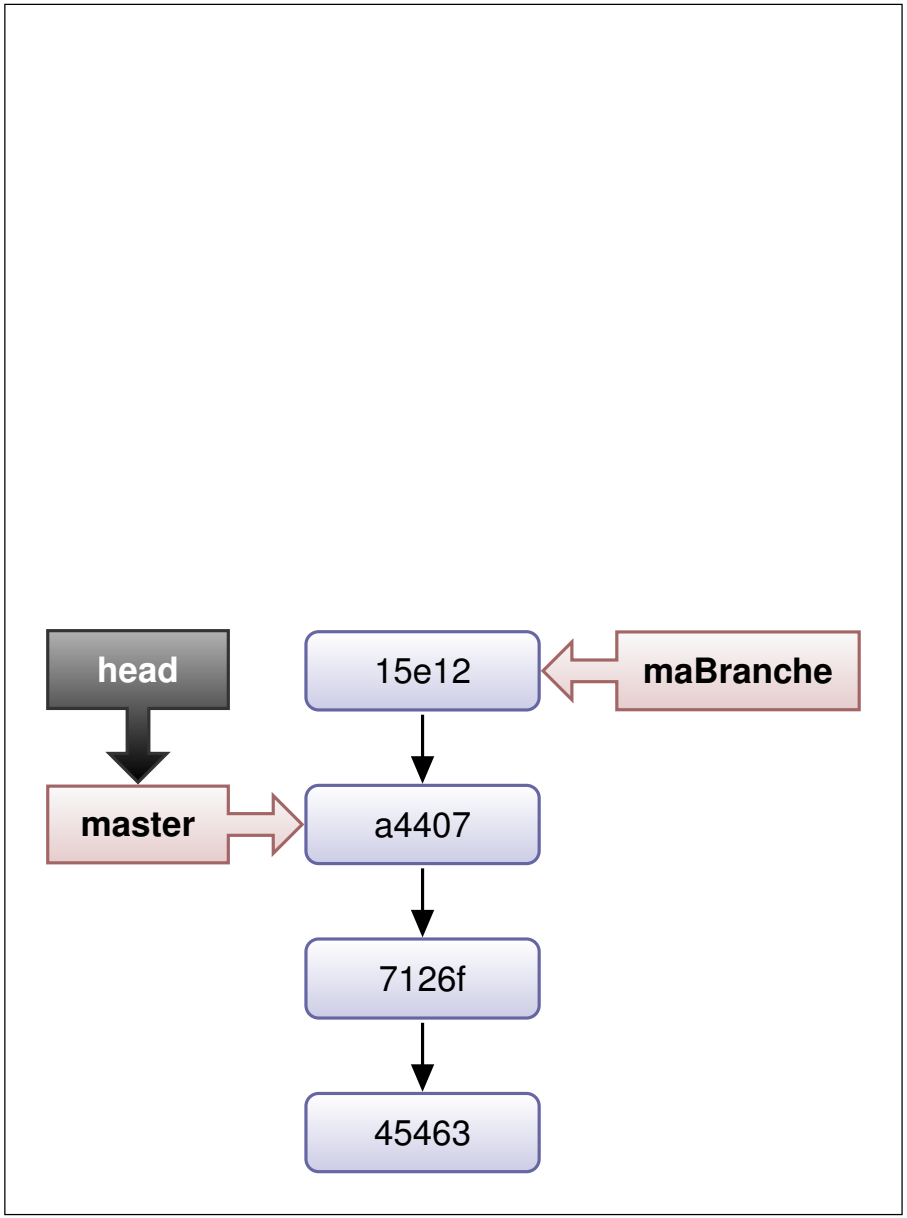
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt
```



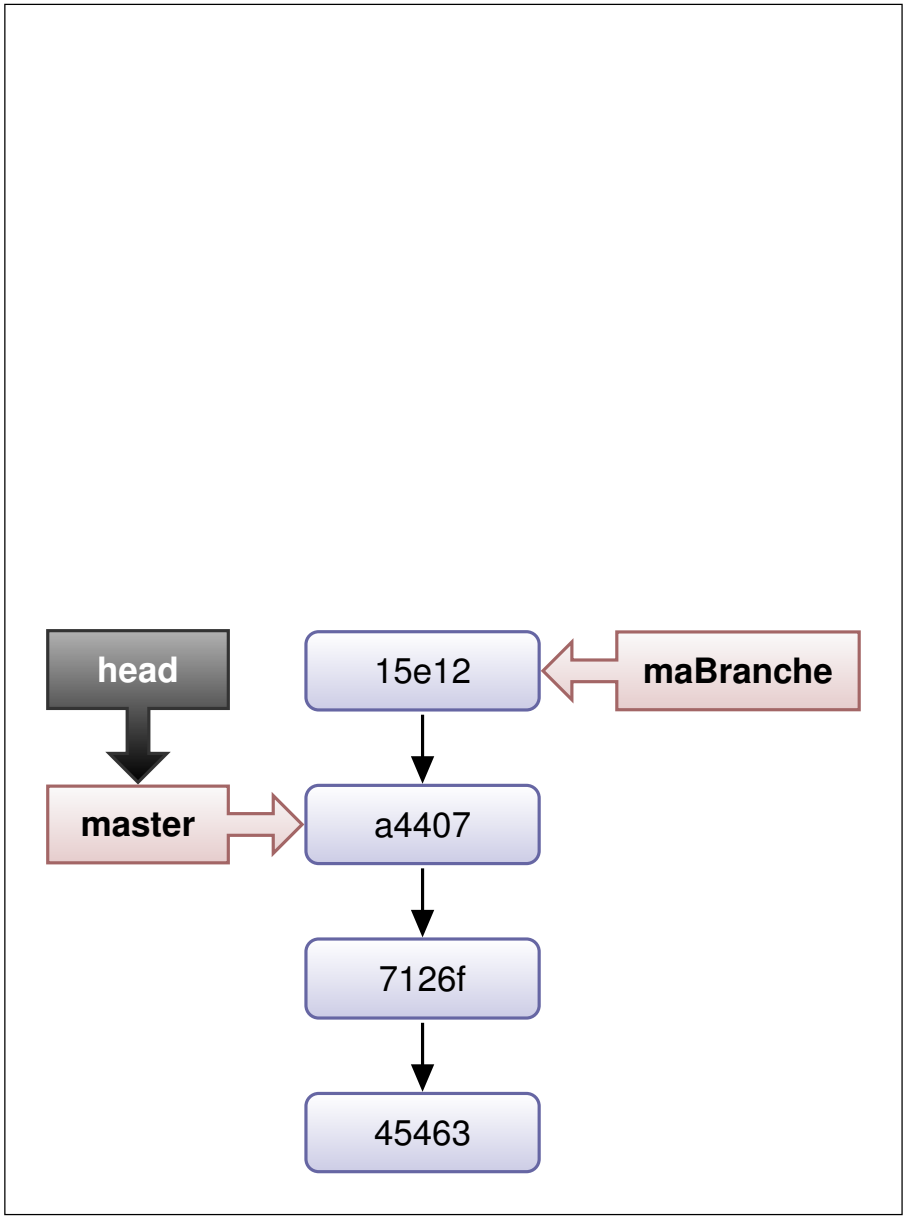
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt
```



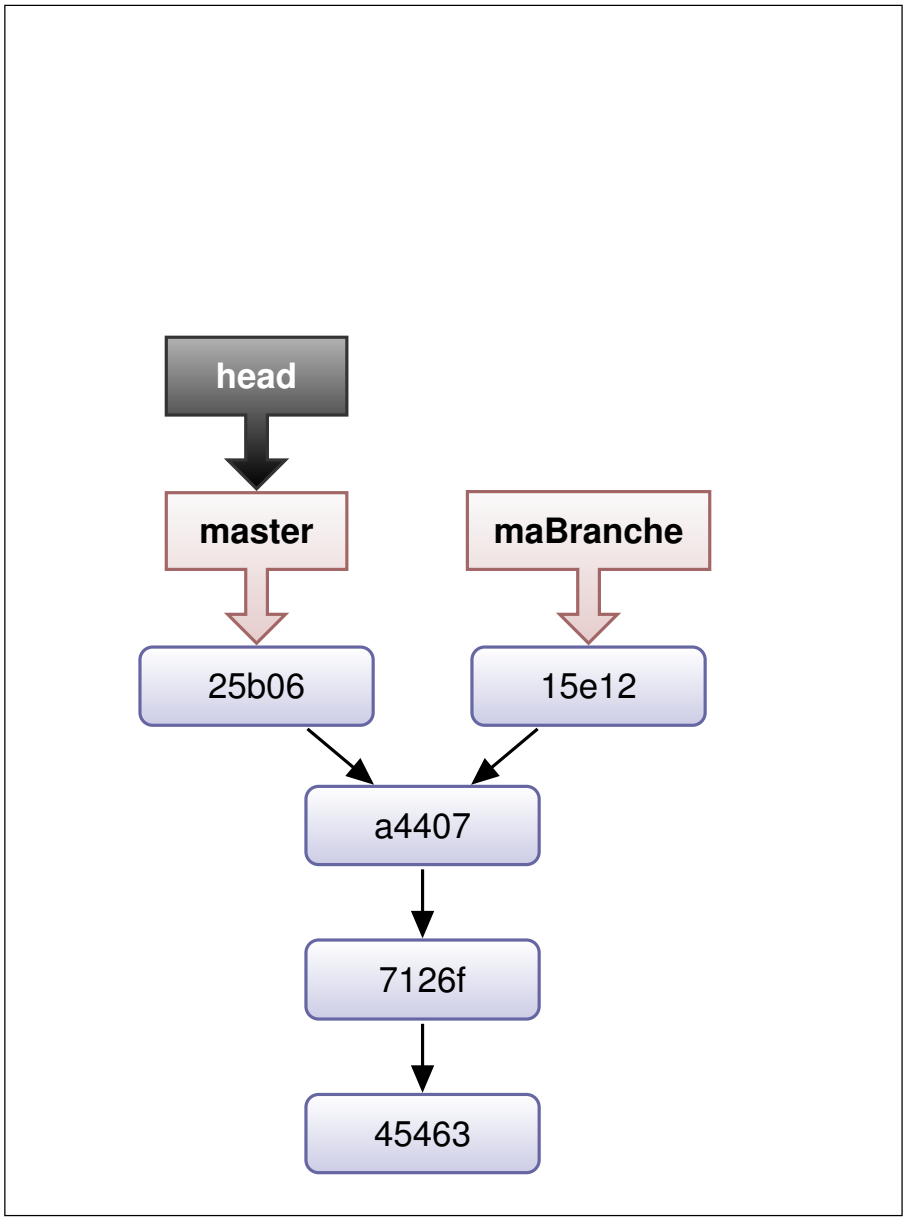
Branche : structure interne des commits.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt

touch fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"
```



Branche : structure interne des commits.

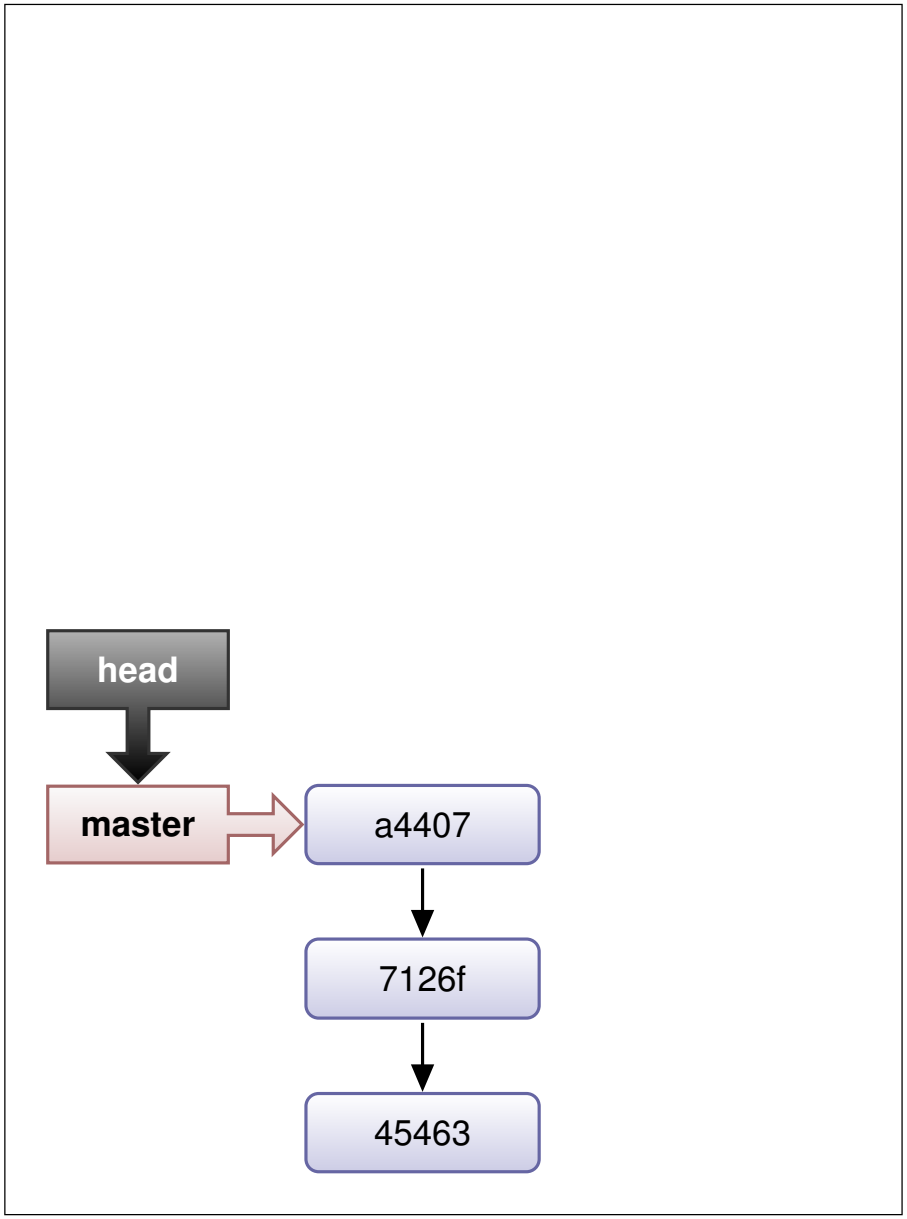
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt

touch fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"
```



Outline

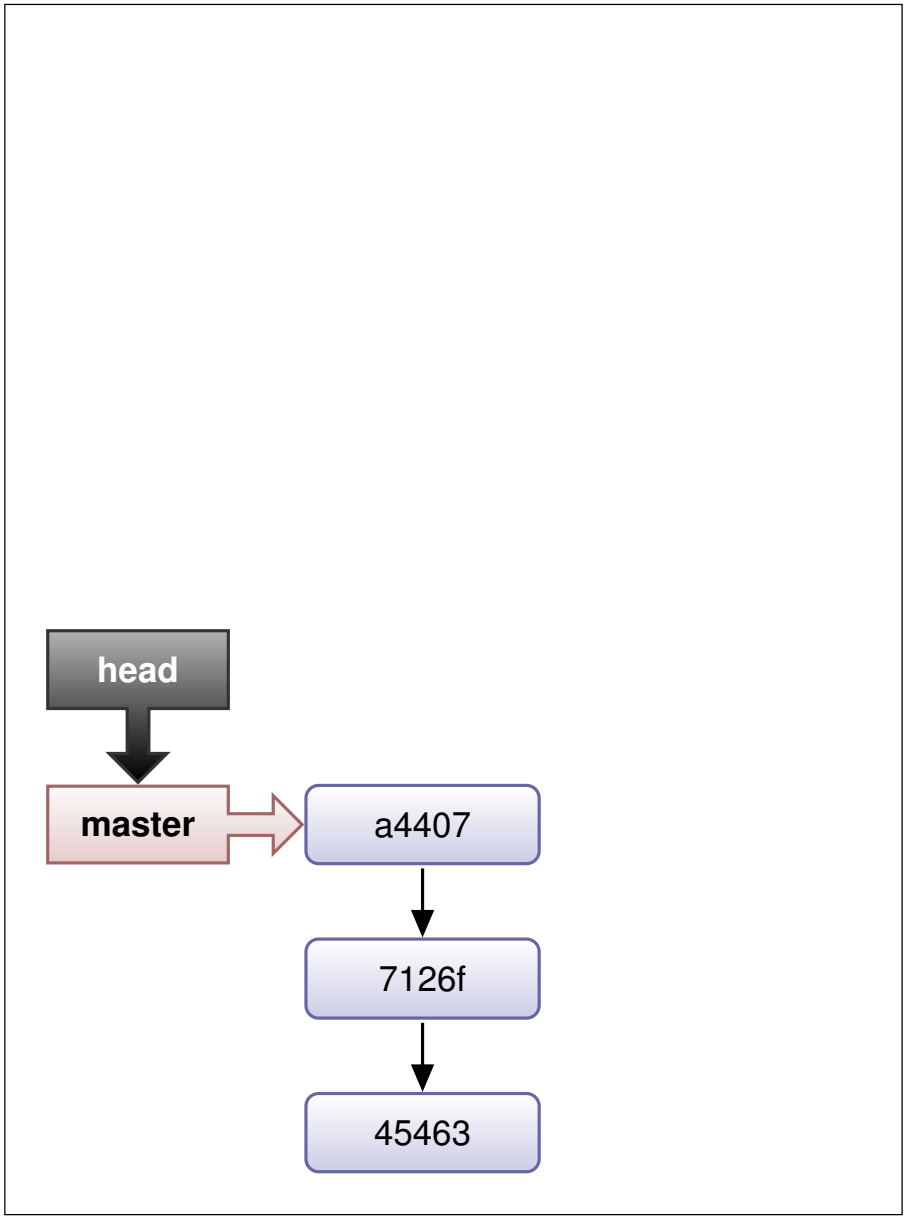
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir
```



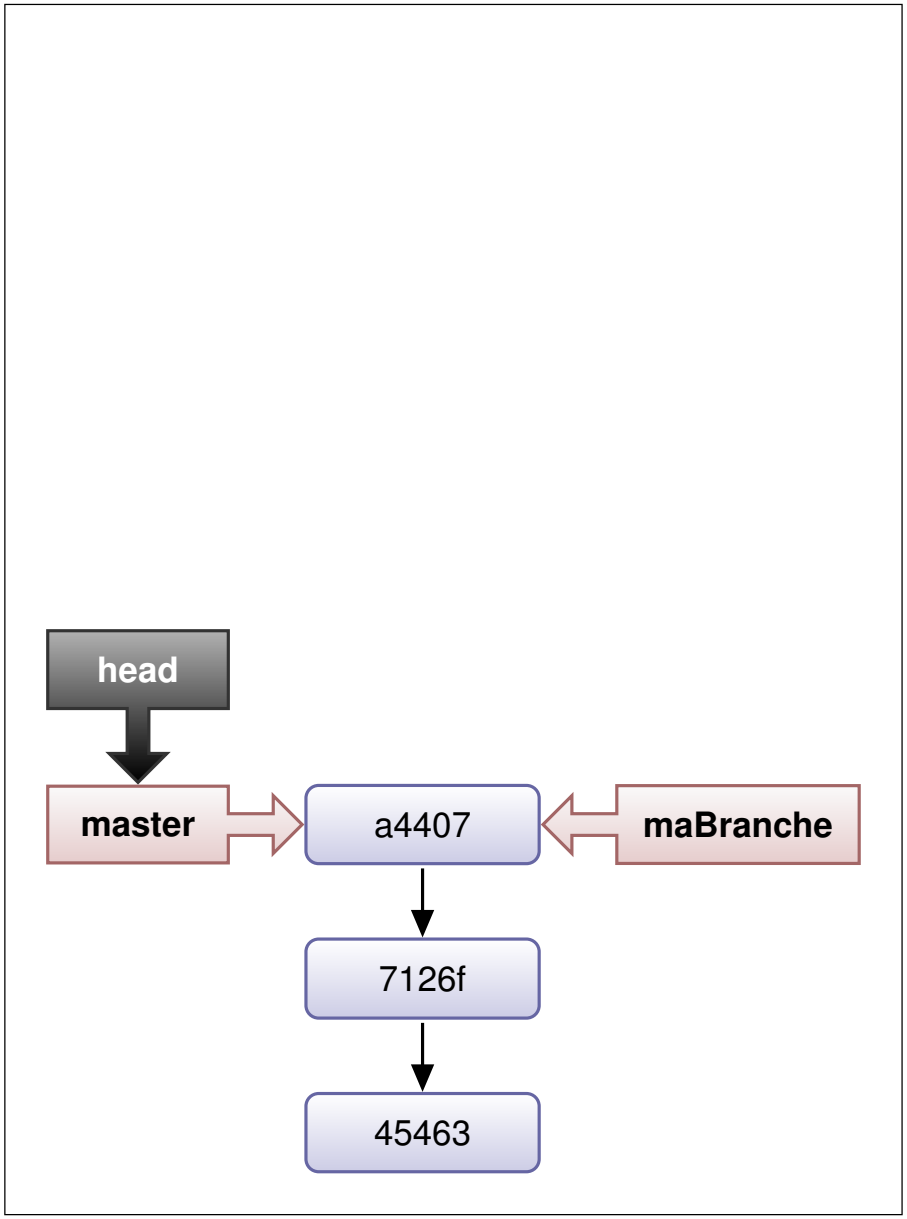
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche
```



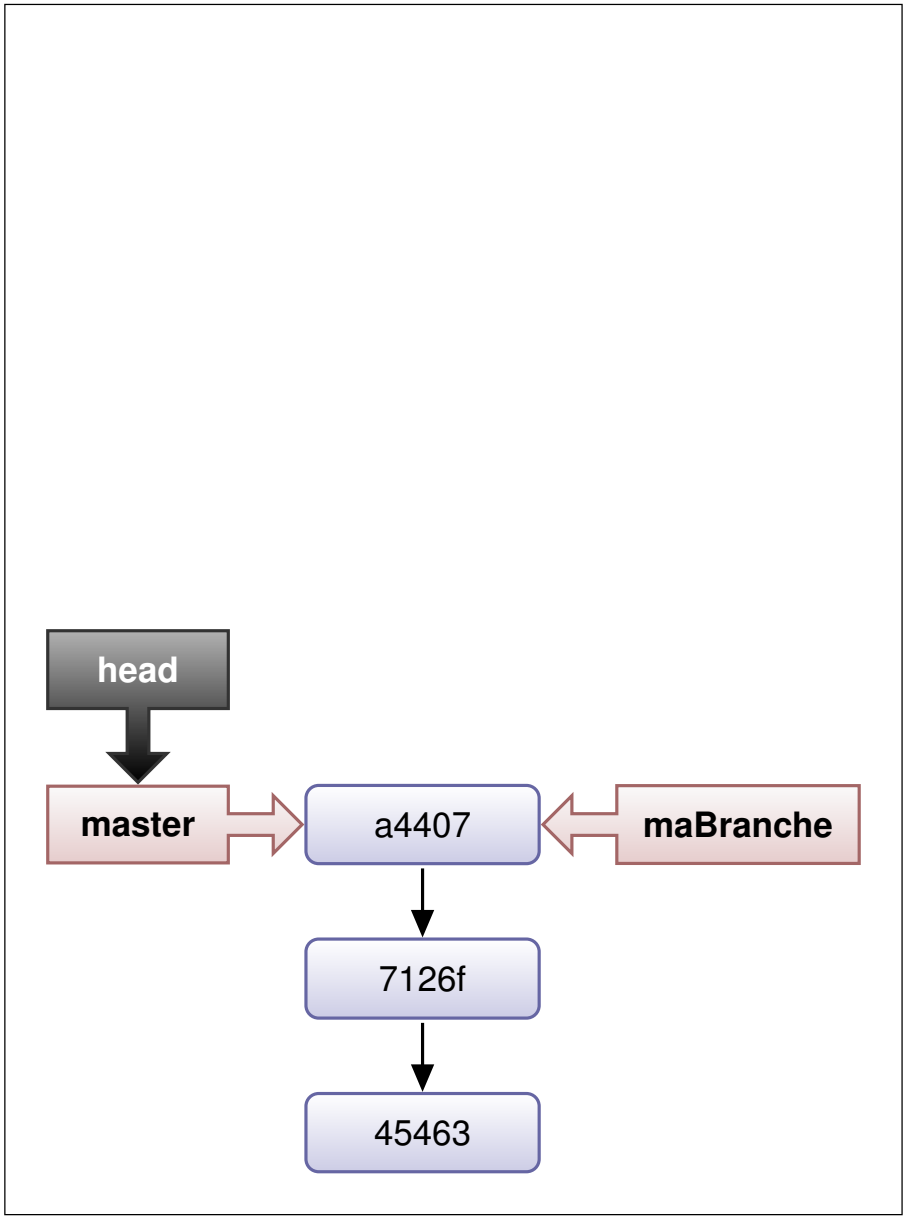
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche
```



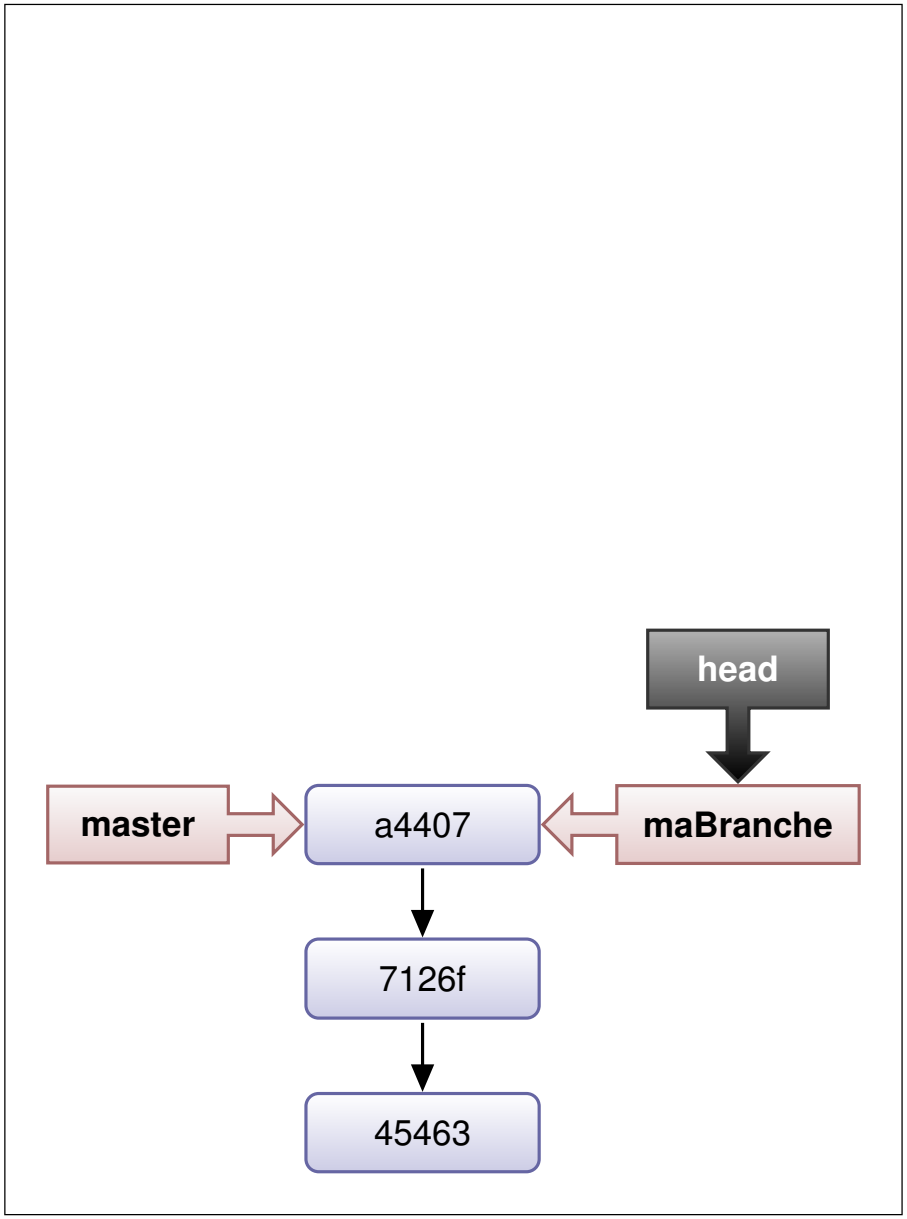
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



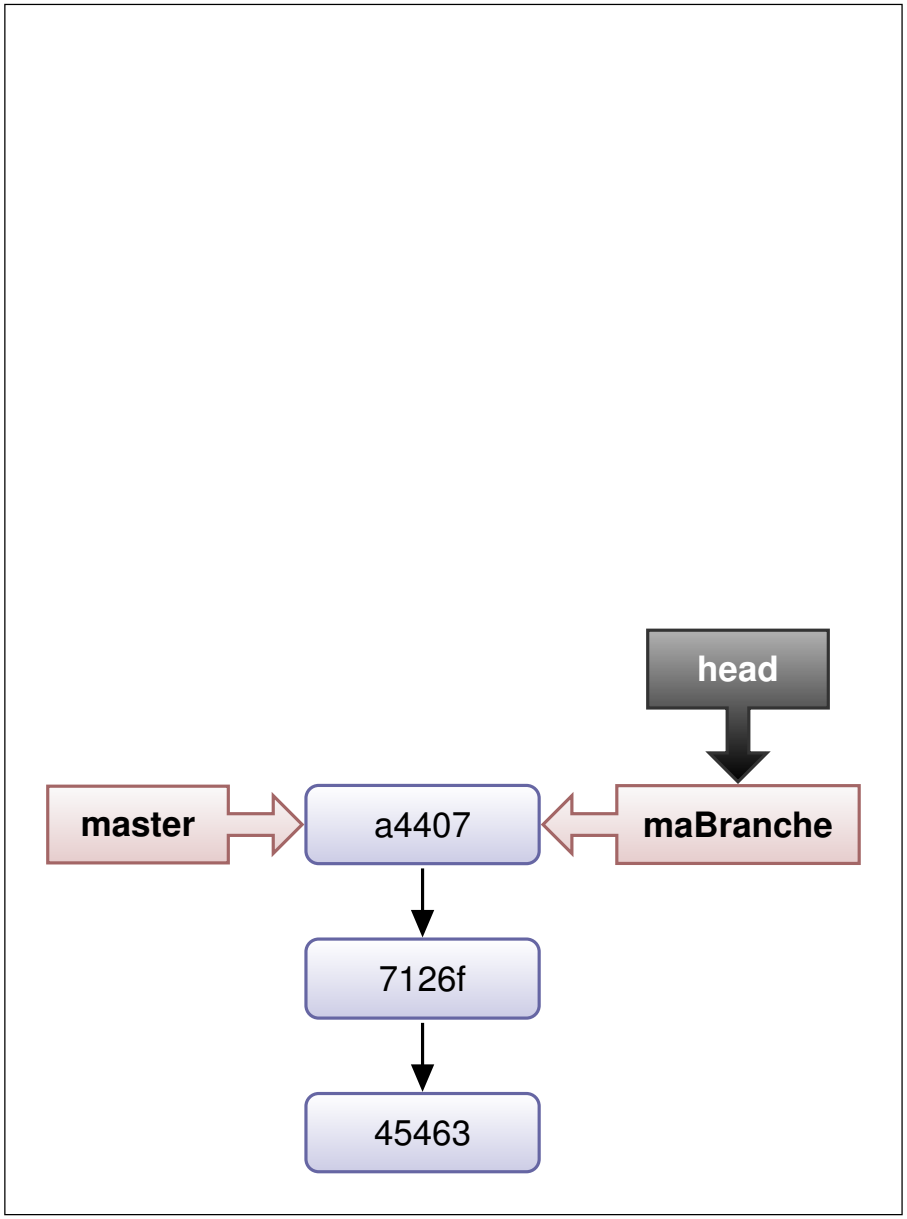
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



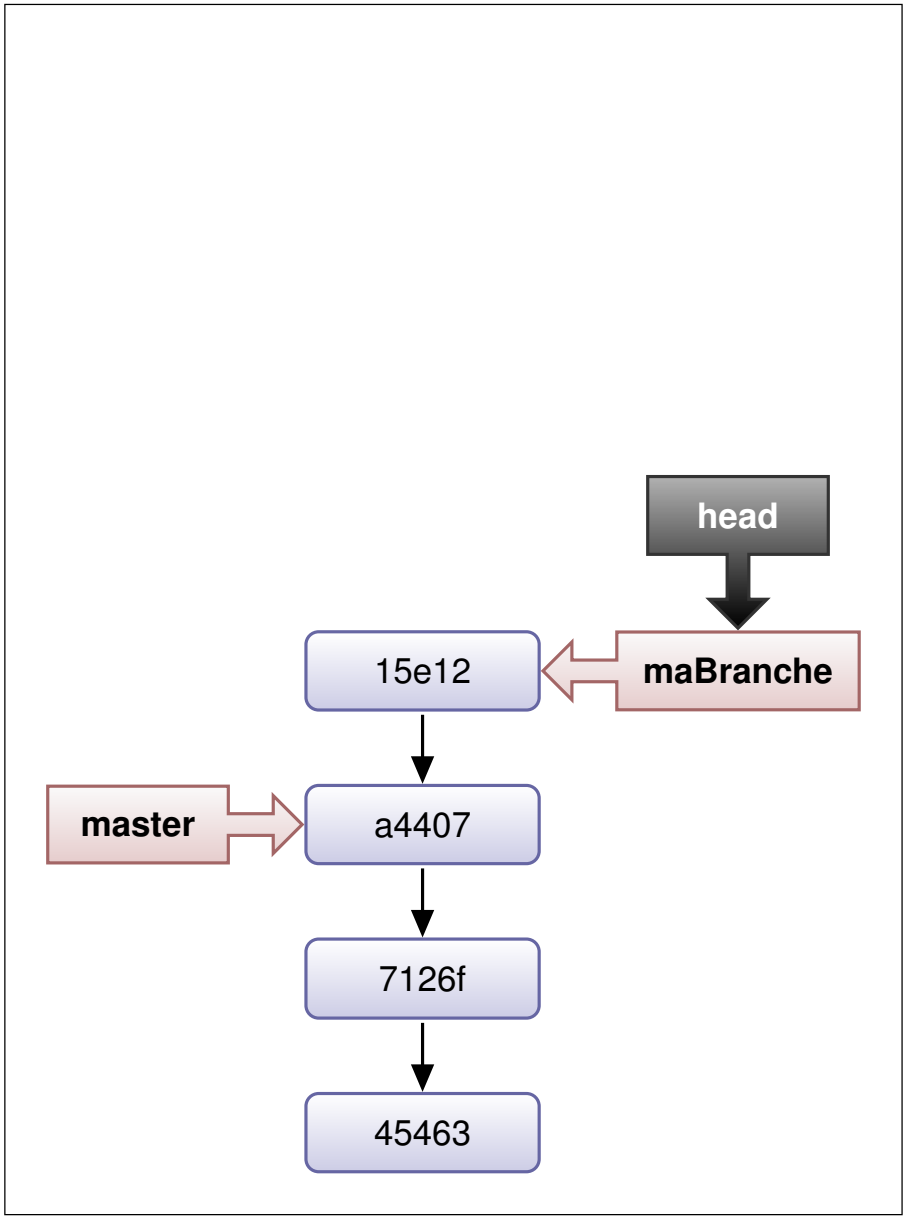
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



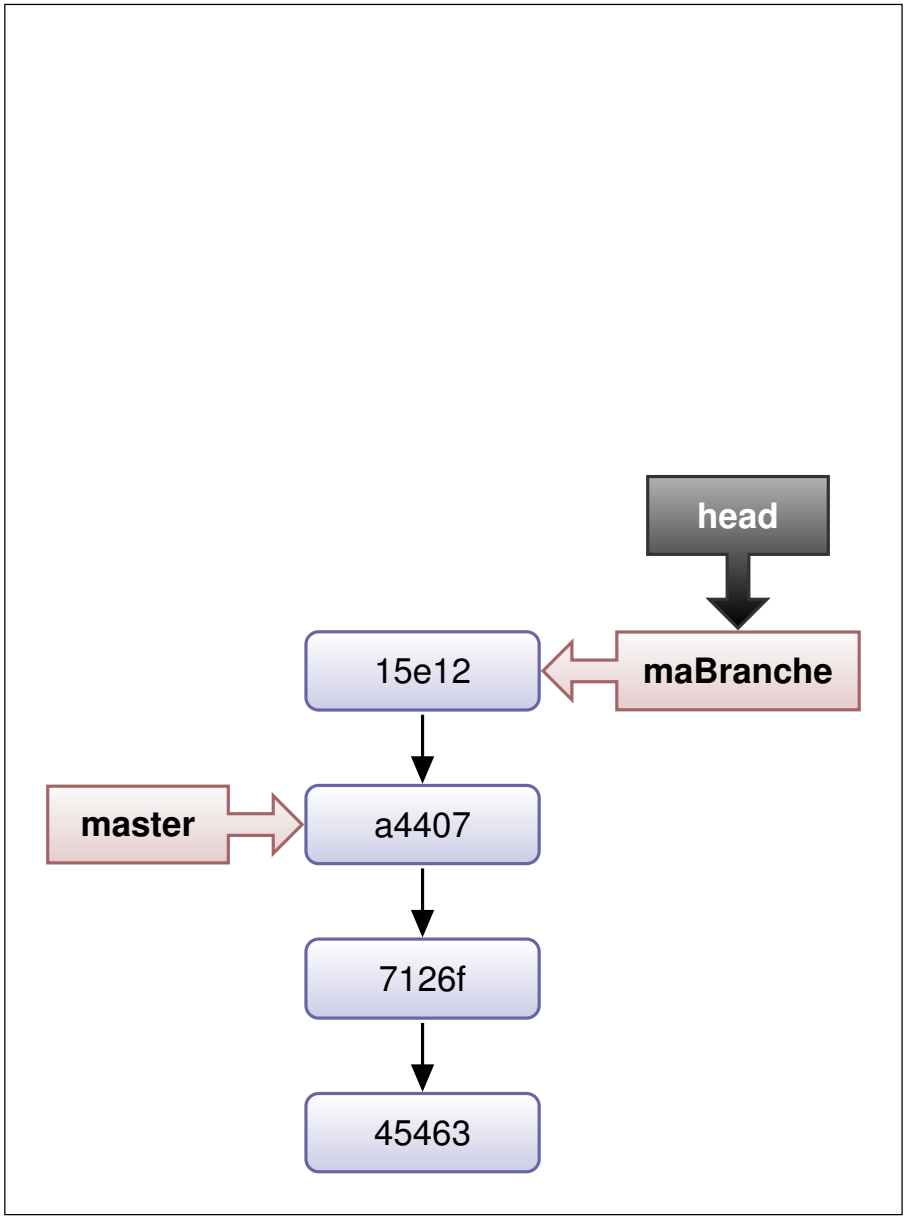
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



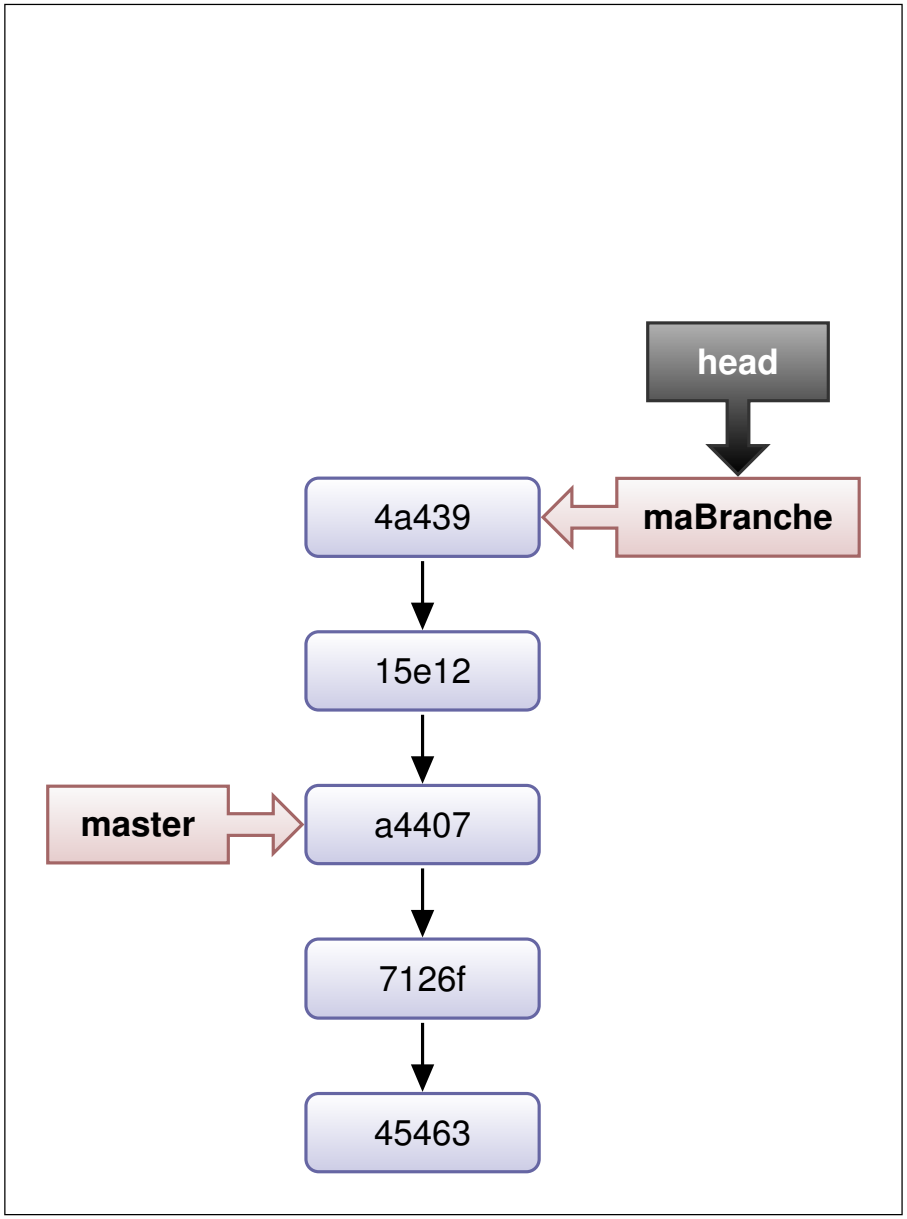
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"
```



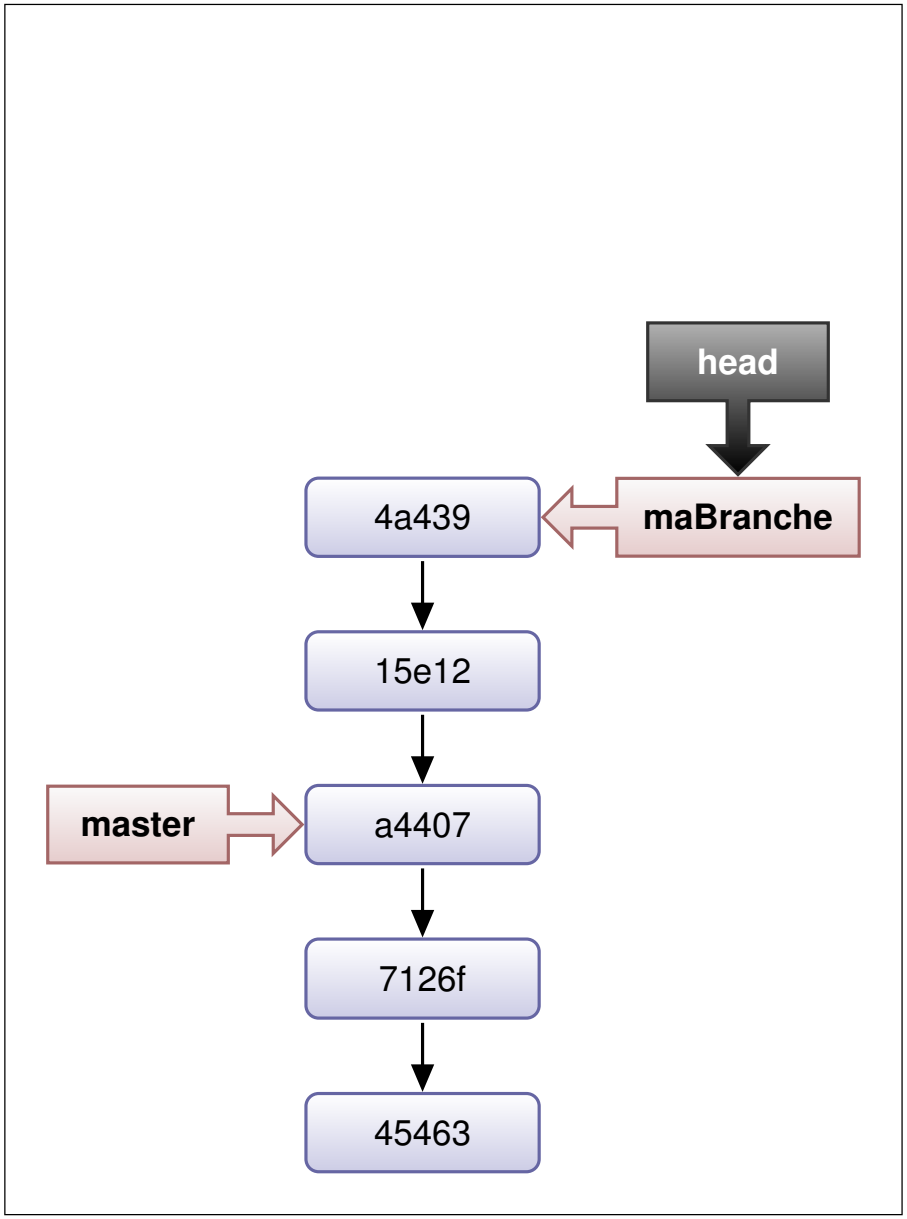
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"
```



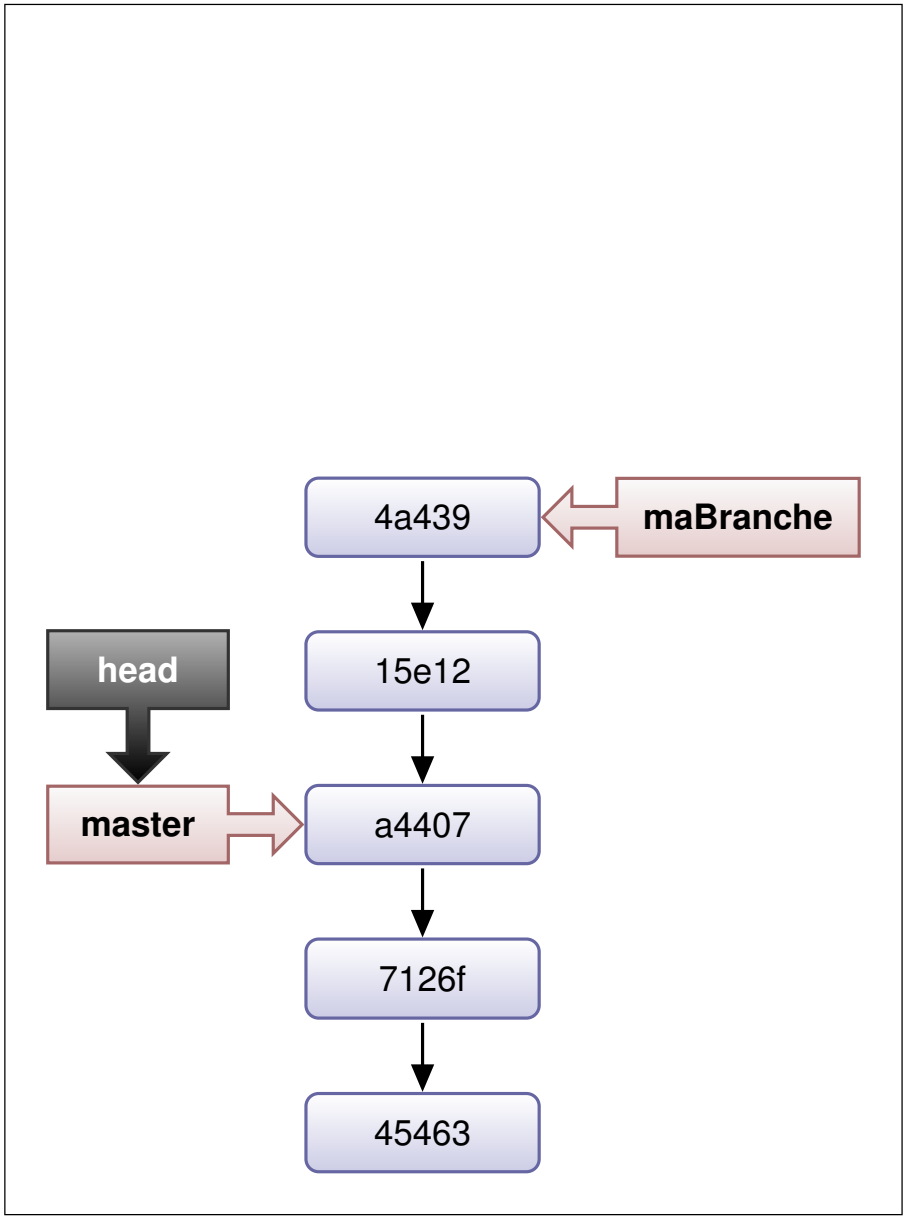
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt
```



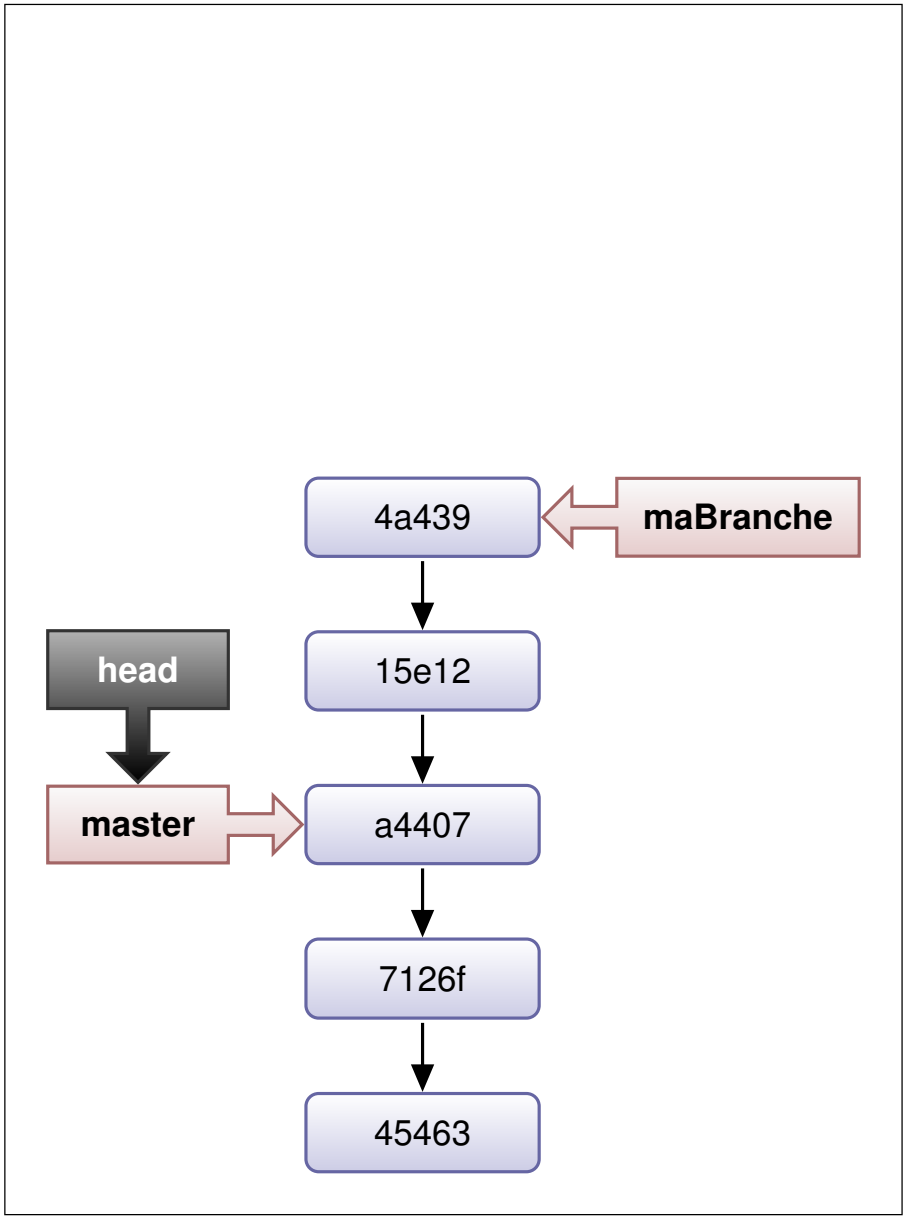
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt
```



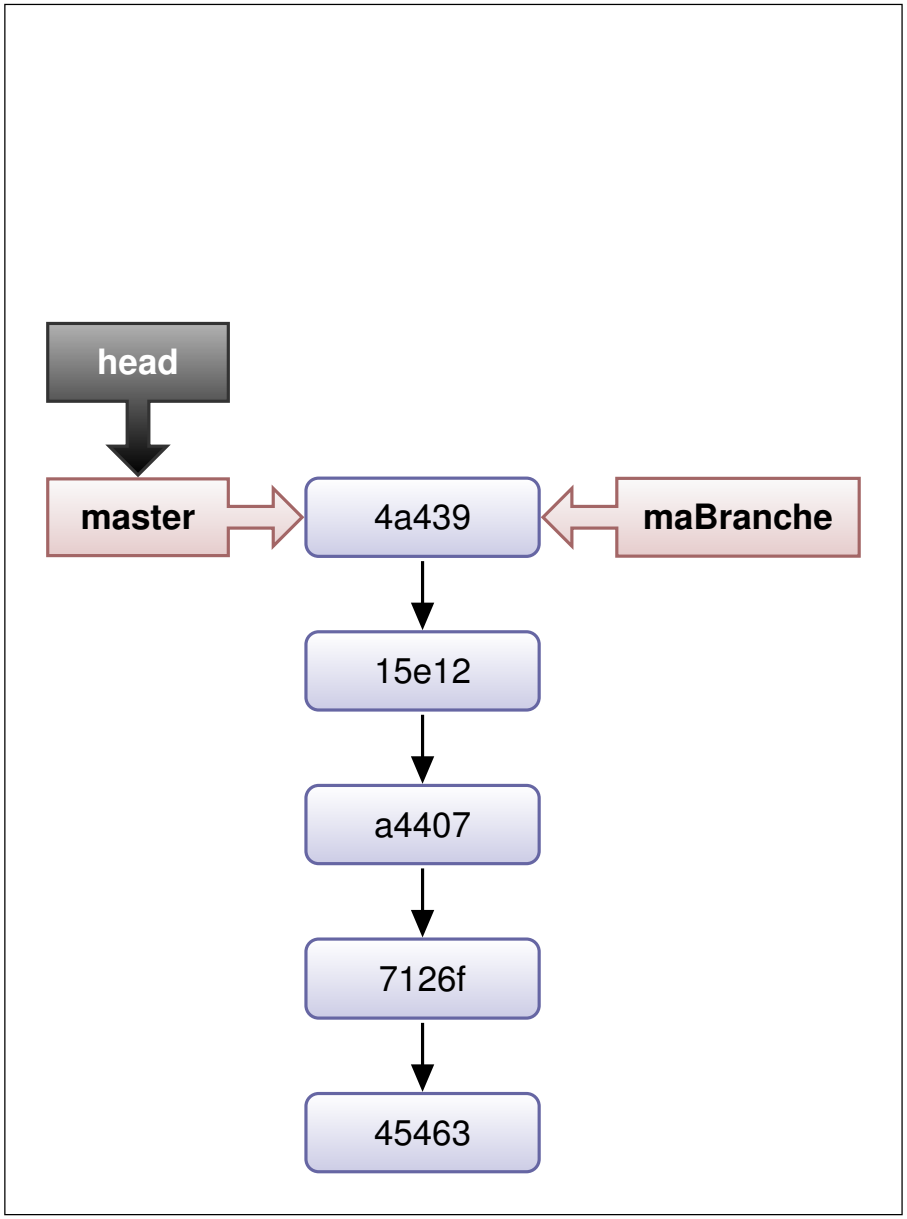
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt

git merge maBranche
cat fichier1.txt
 titi
```



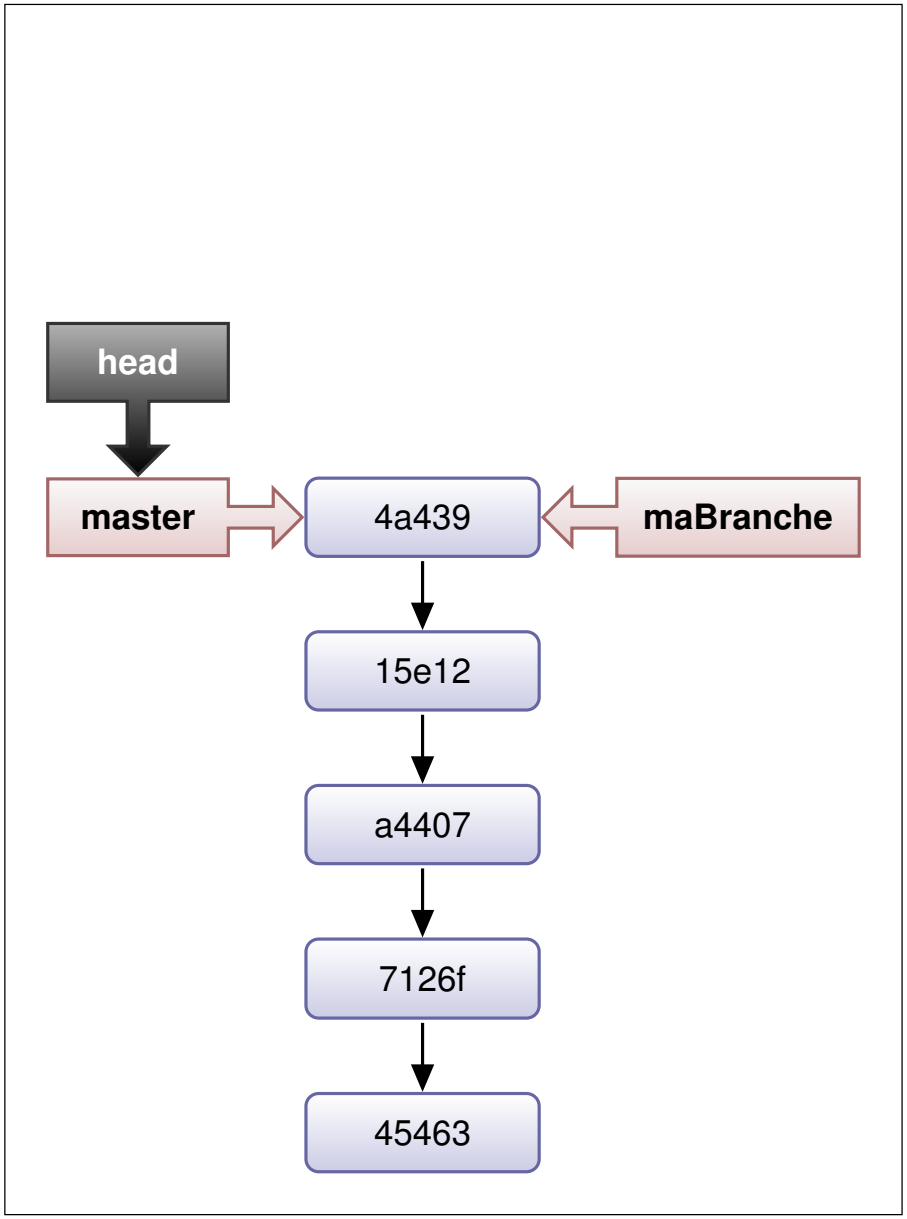
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt

git merge maBranche
cat fichier1.txt
 titi
```



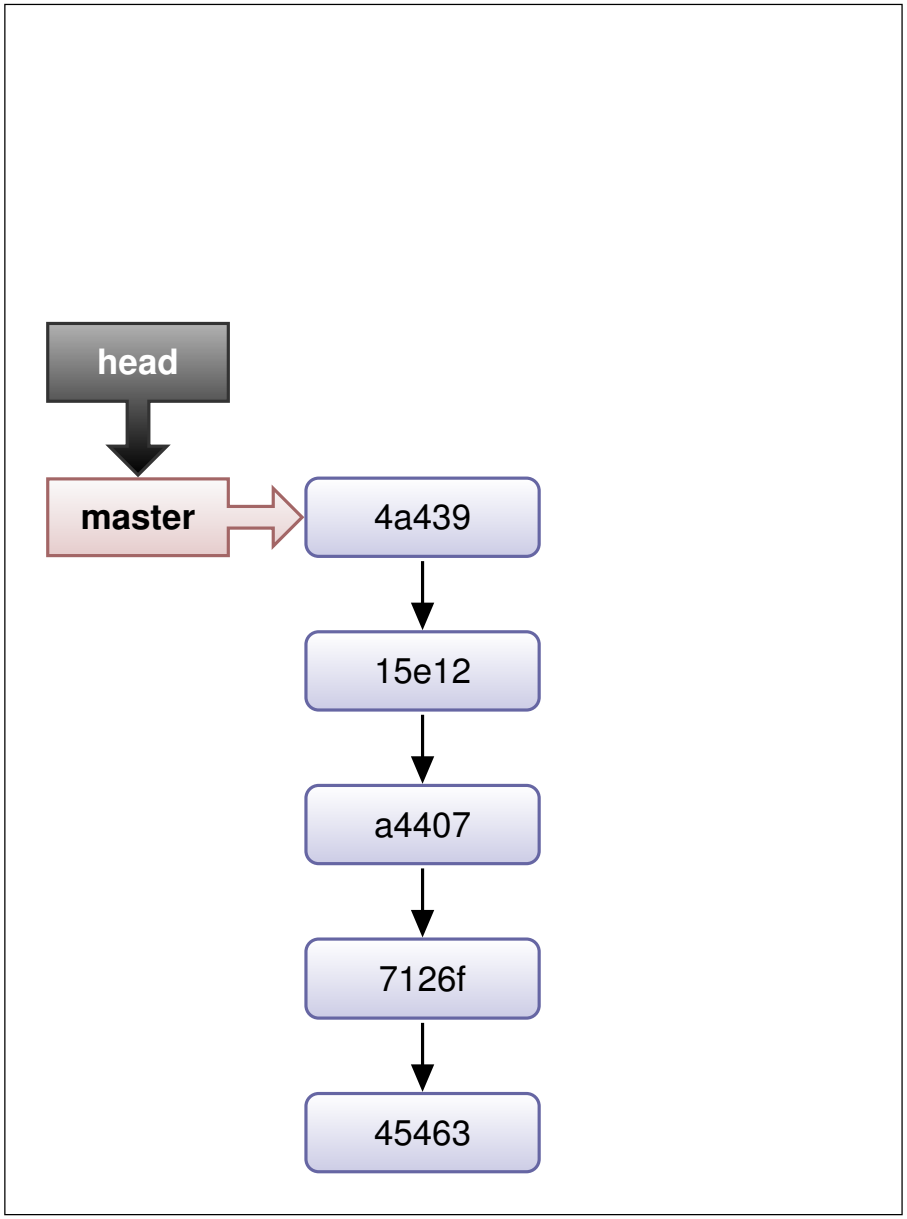
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt

git merge maBranche
cat fichier1.txt
 titi

git branch -d maBranche
```



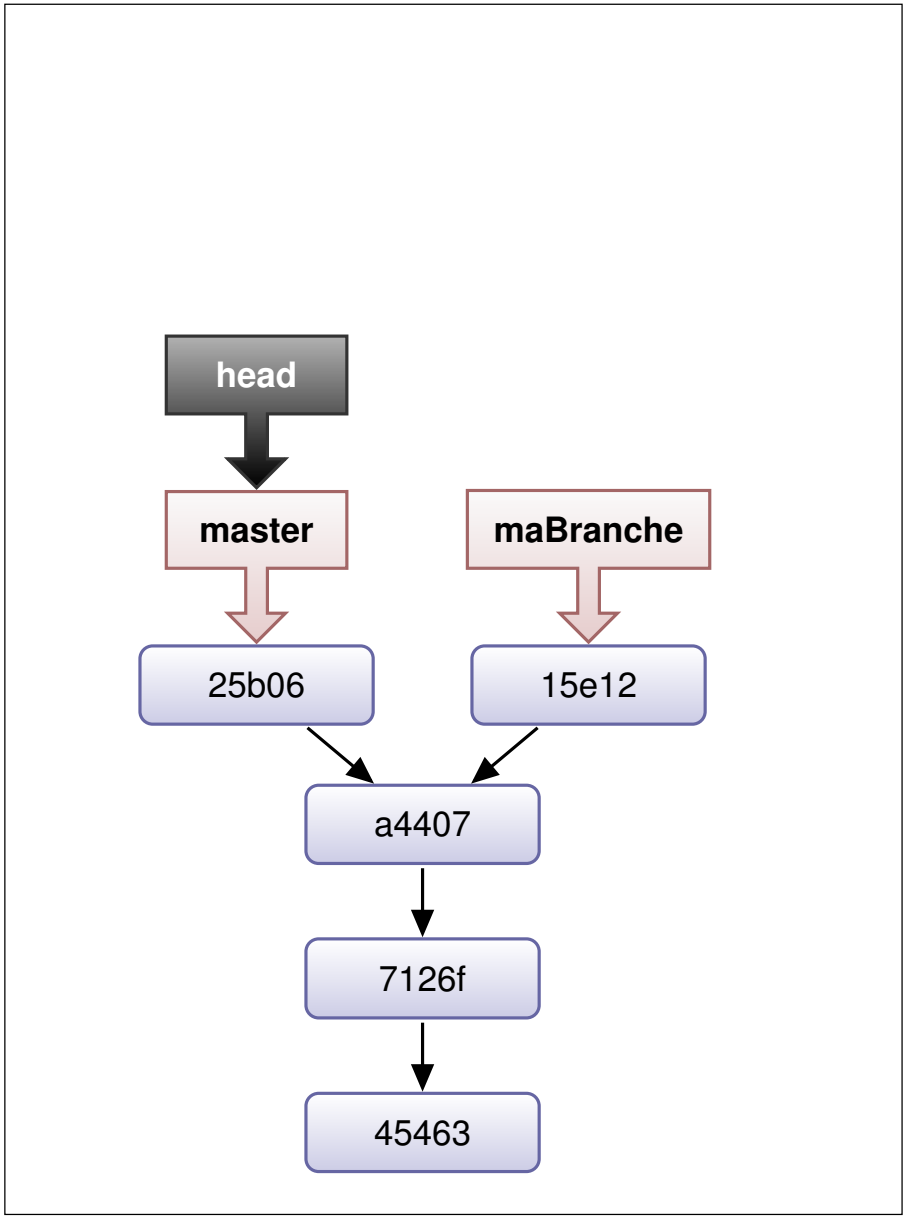
Merge : exemple sur un historique unique.

```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

echo "toto" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
ls
 dir foo.txt

git merge maBranche
cat fichier1.txt
 titi

git branch -d maBranche
```



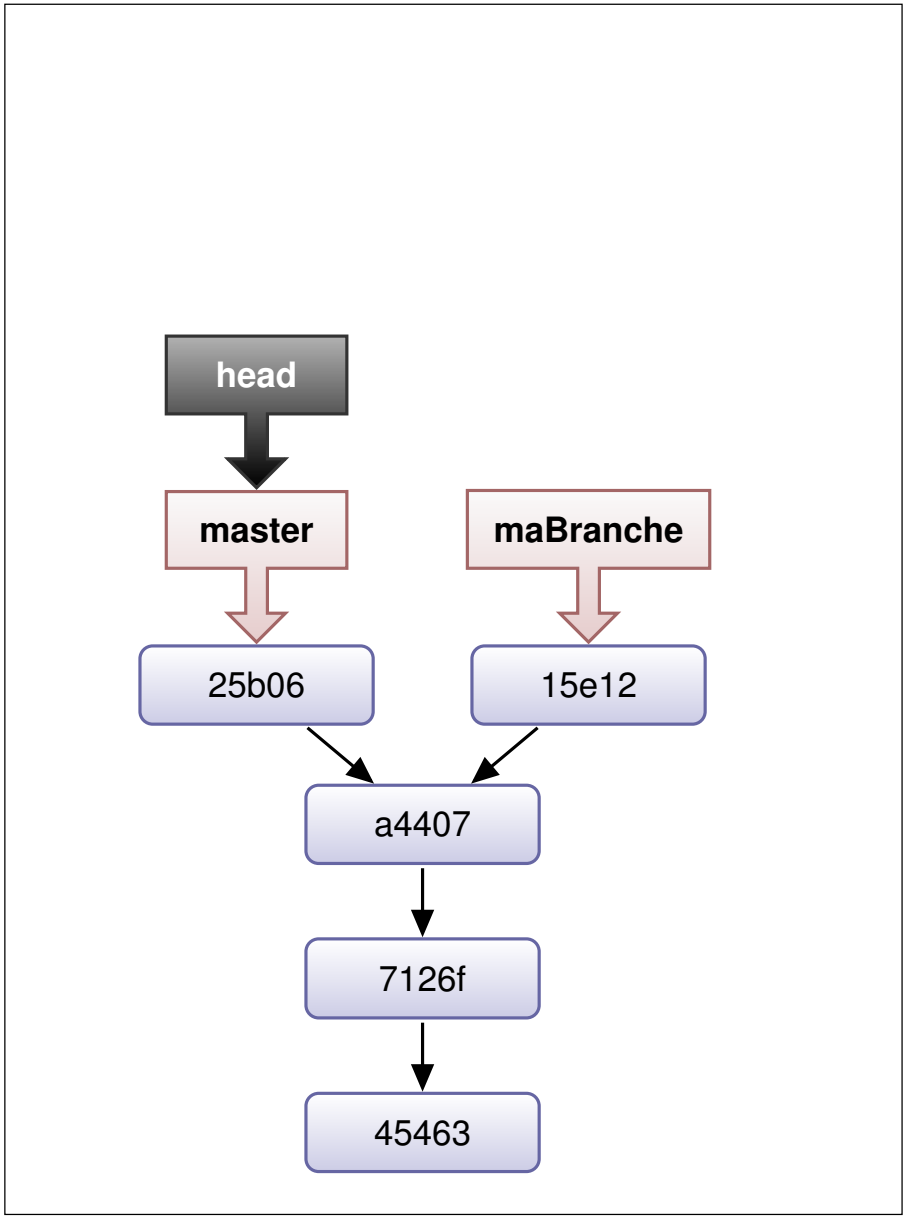
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt
```



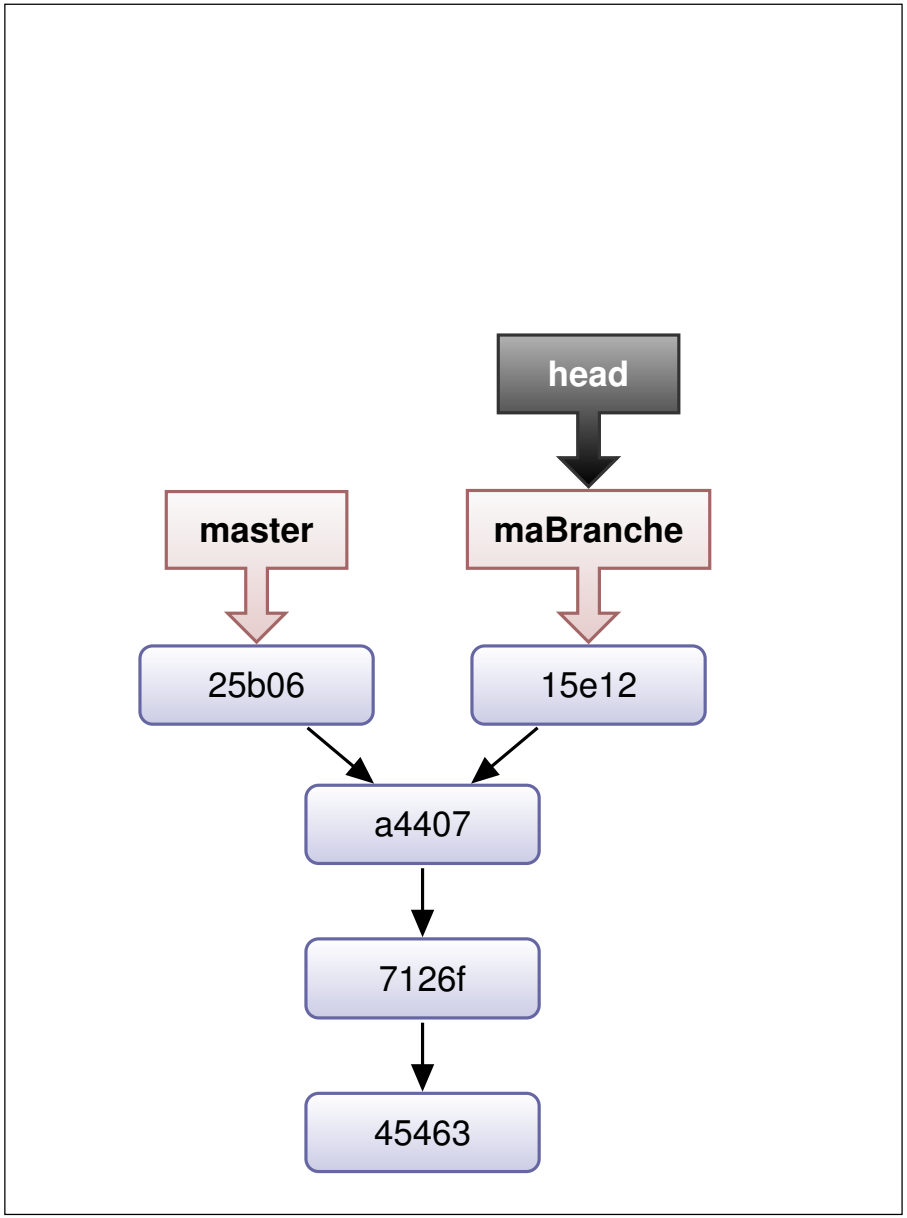
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt
```



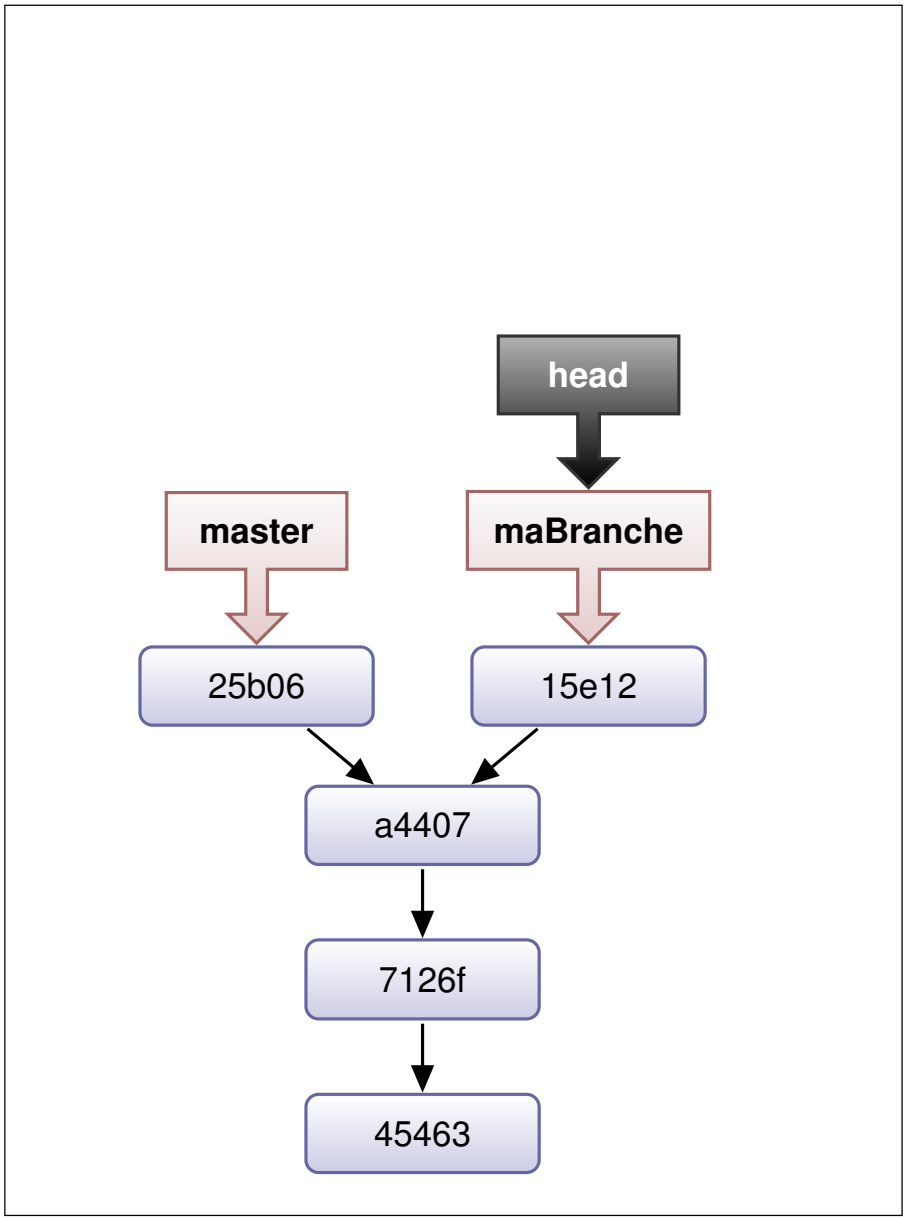
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt
```



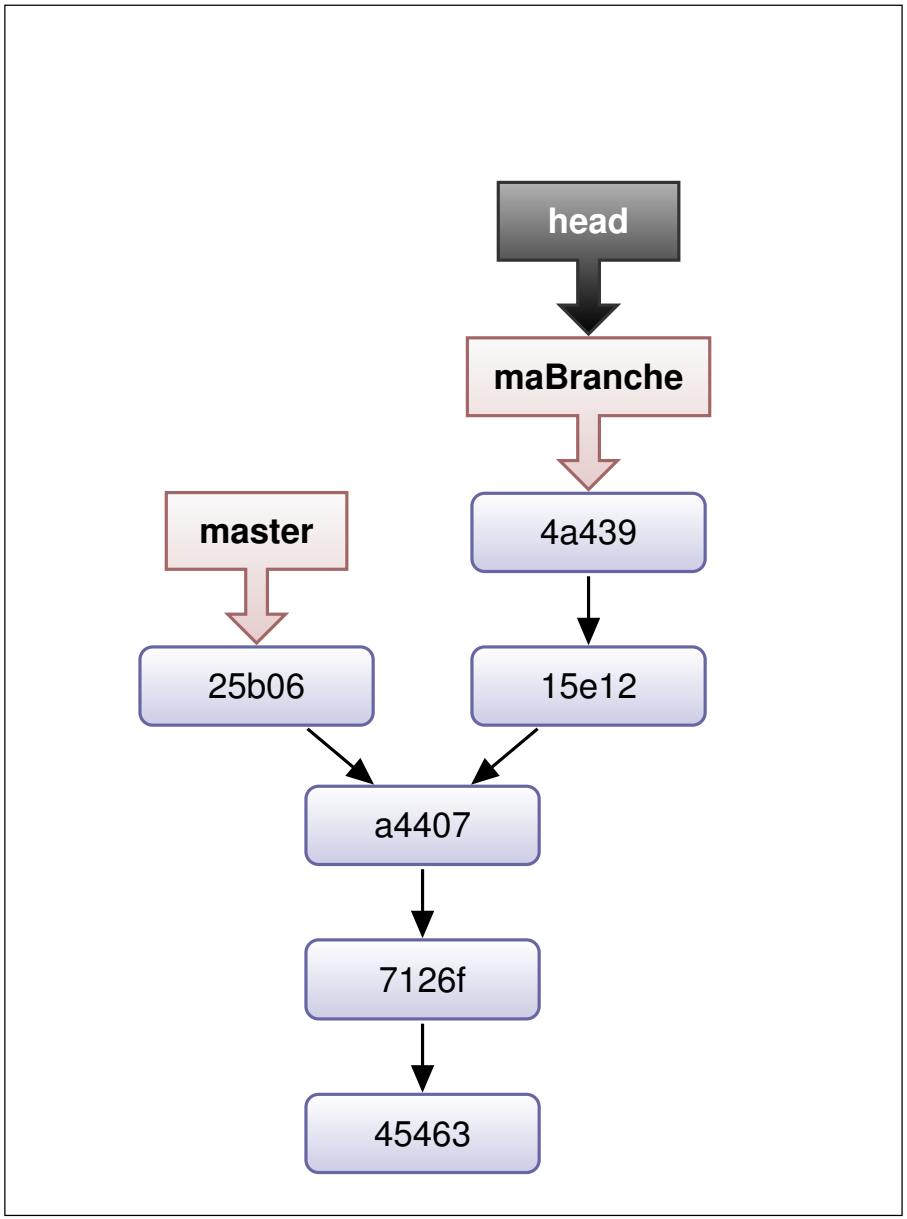
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"
```



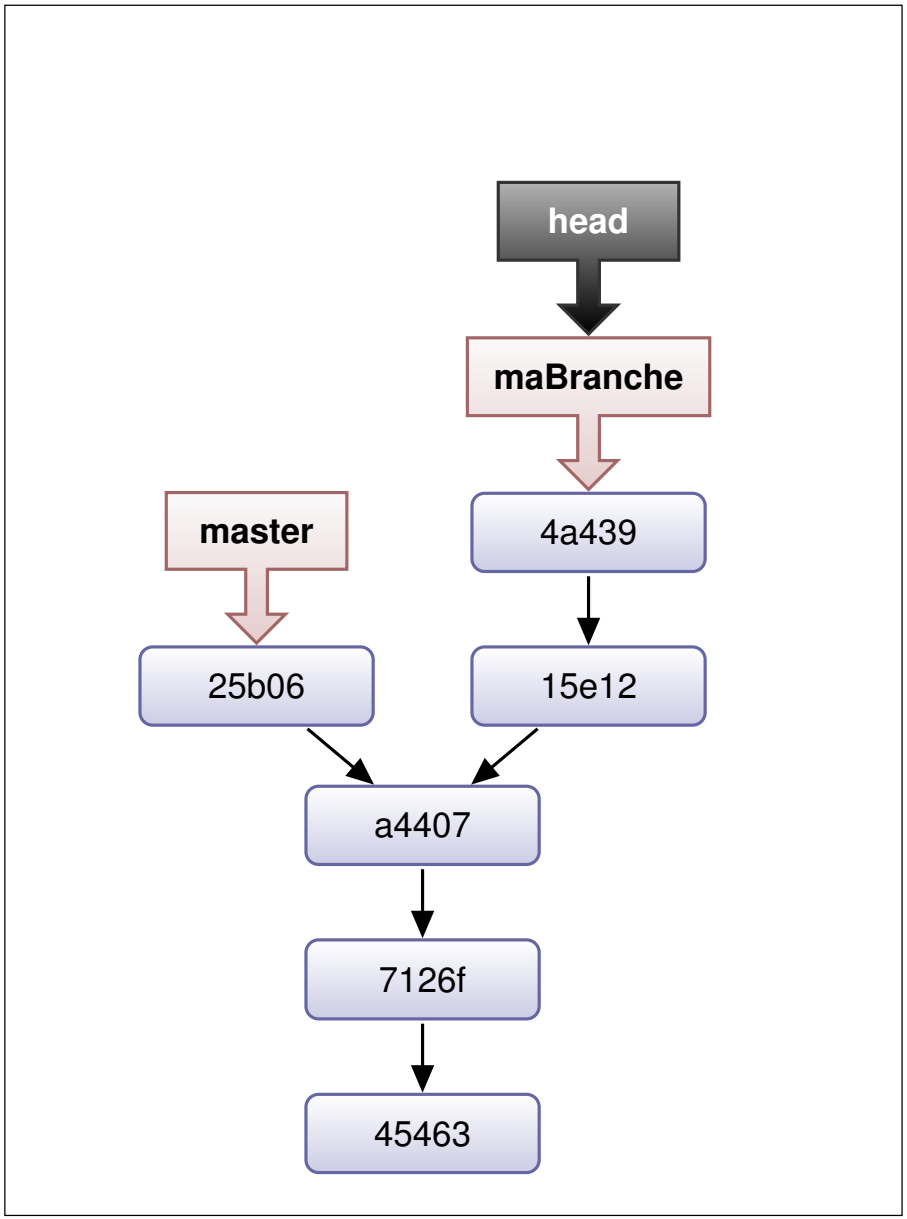
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"
```



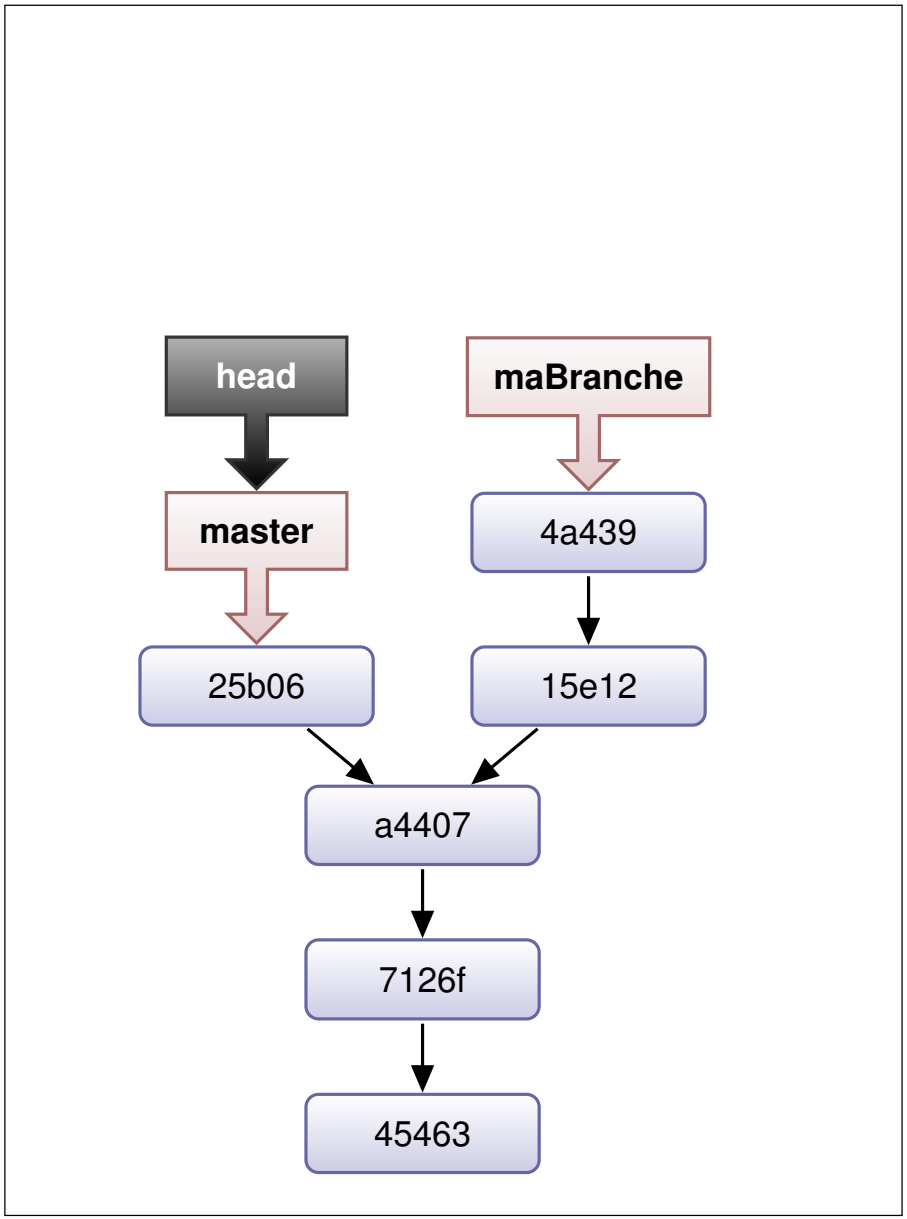
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
```



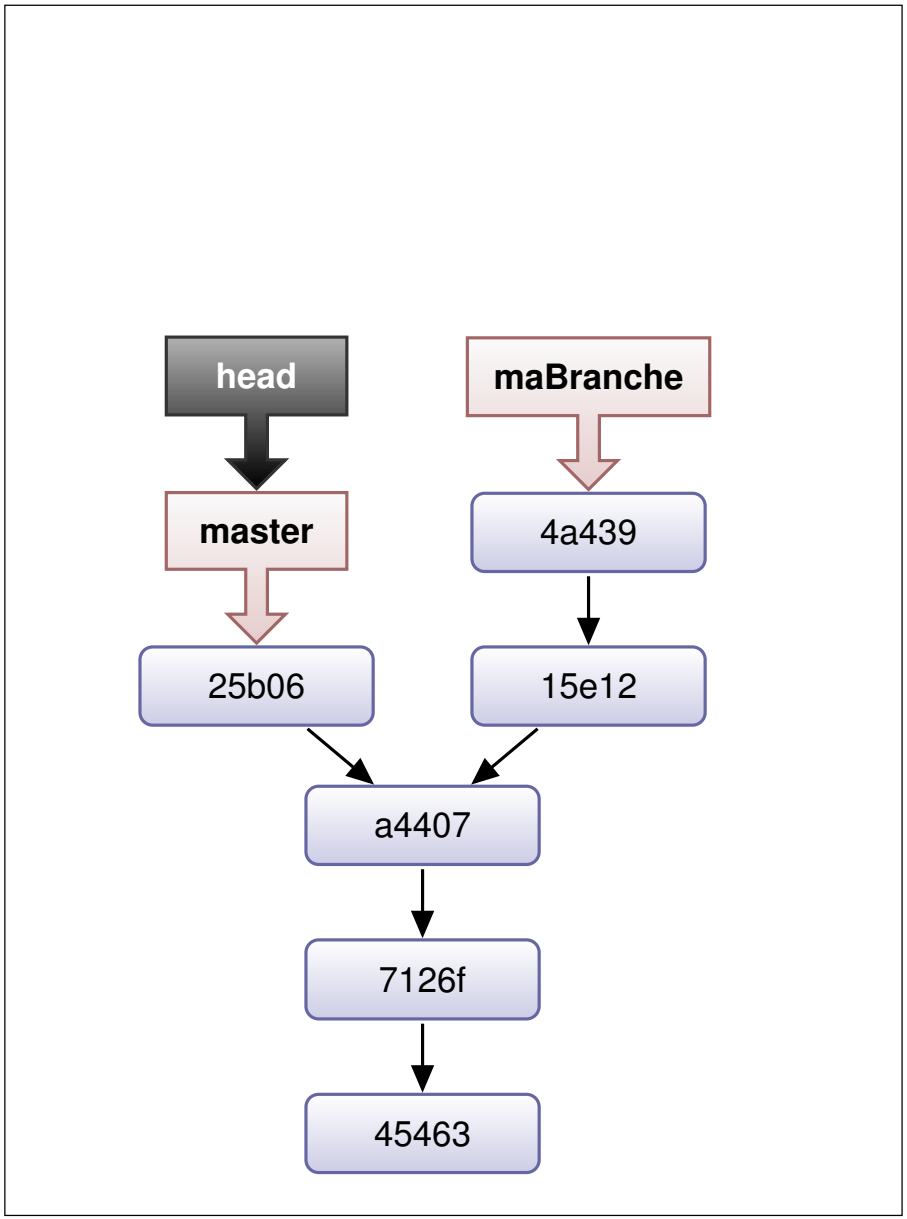
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
```



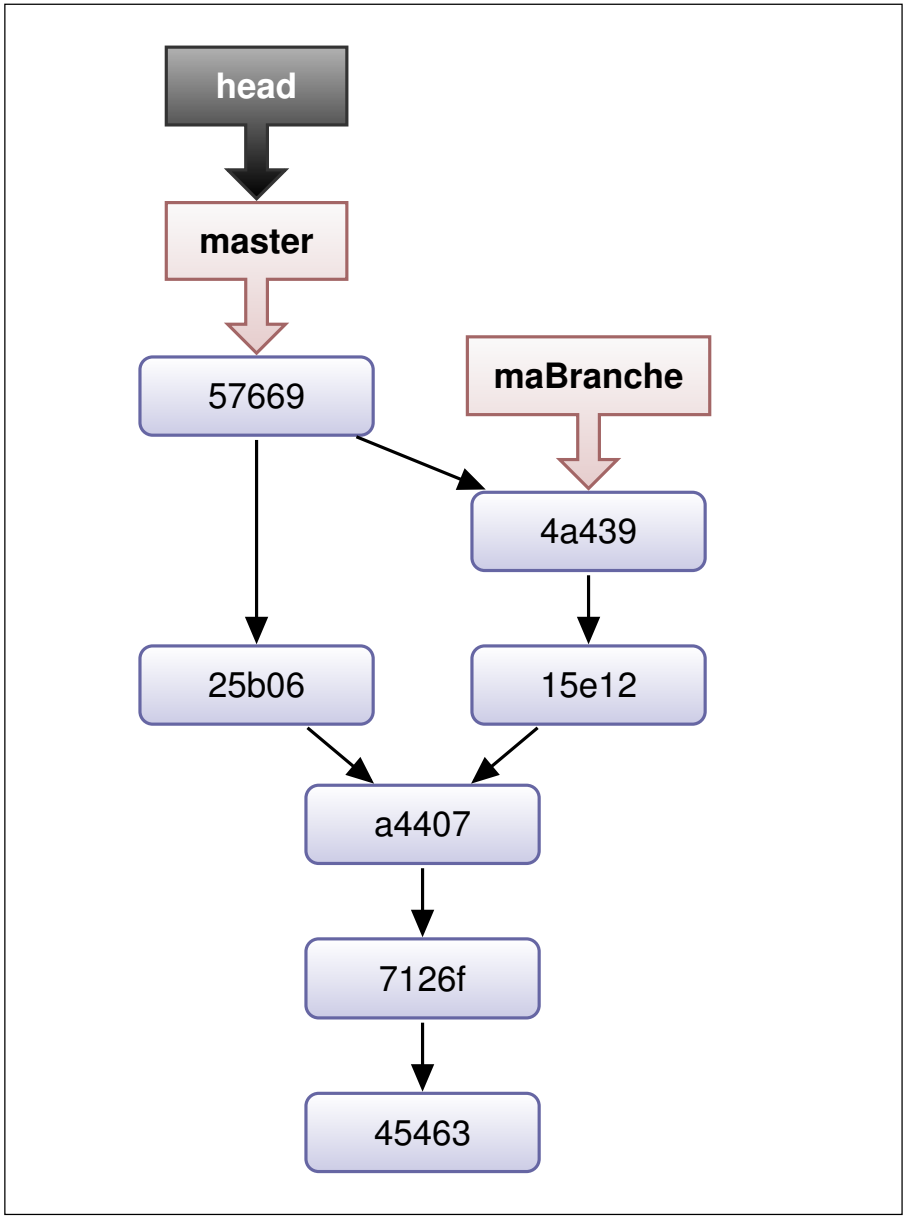
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt
```



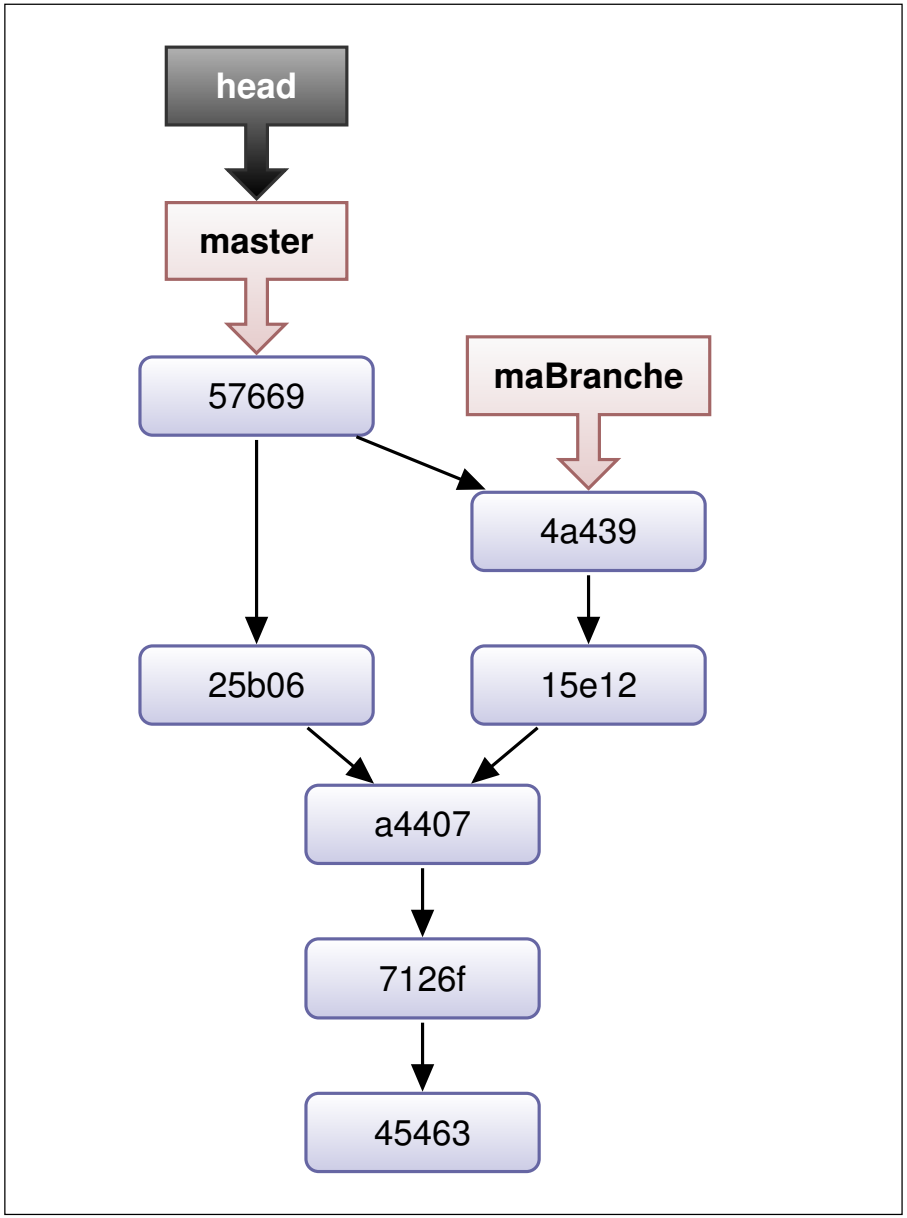
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt
```



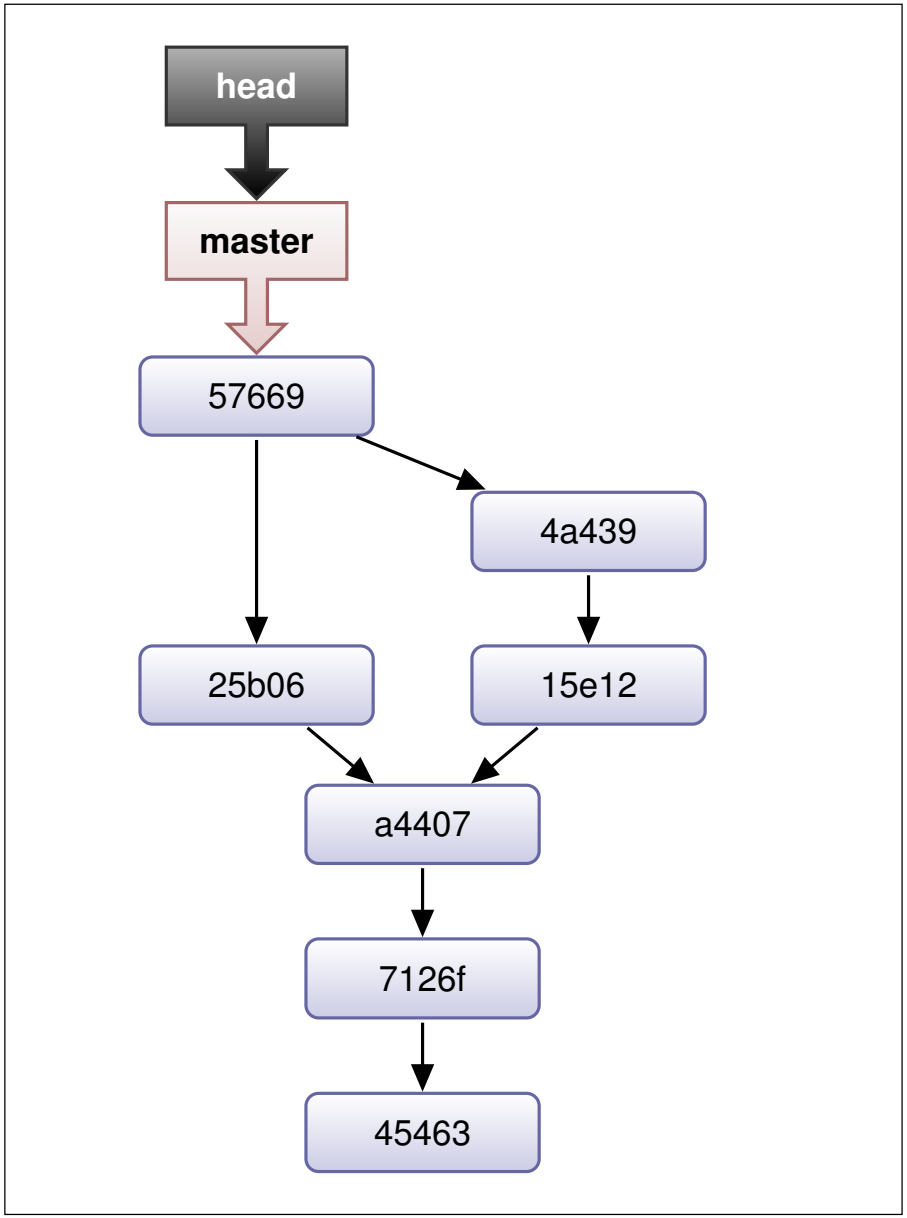
Merge : exemple sur deux branches distinctes.

```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt

git branch -d maBranche
```



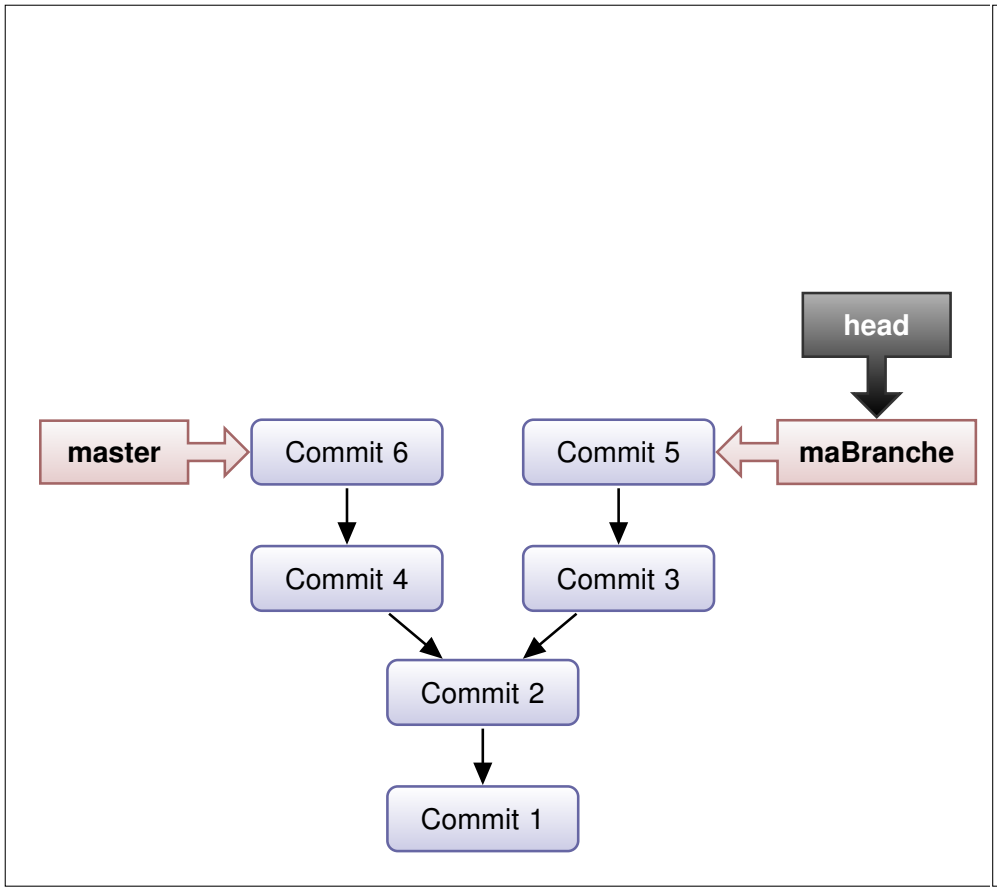
Merge : exemple sur deux branches distinctes.

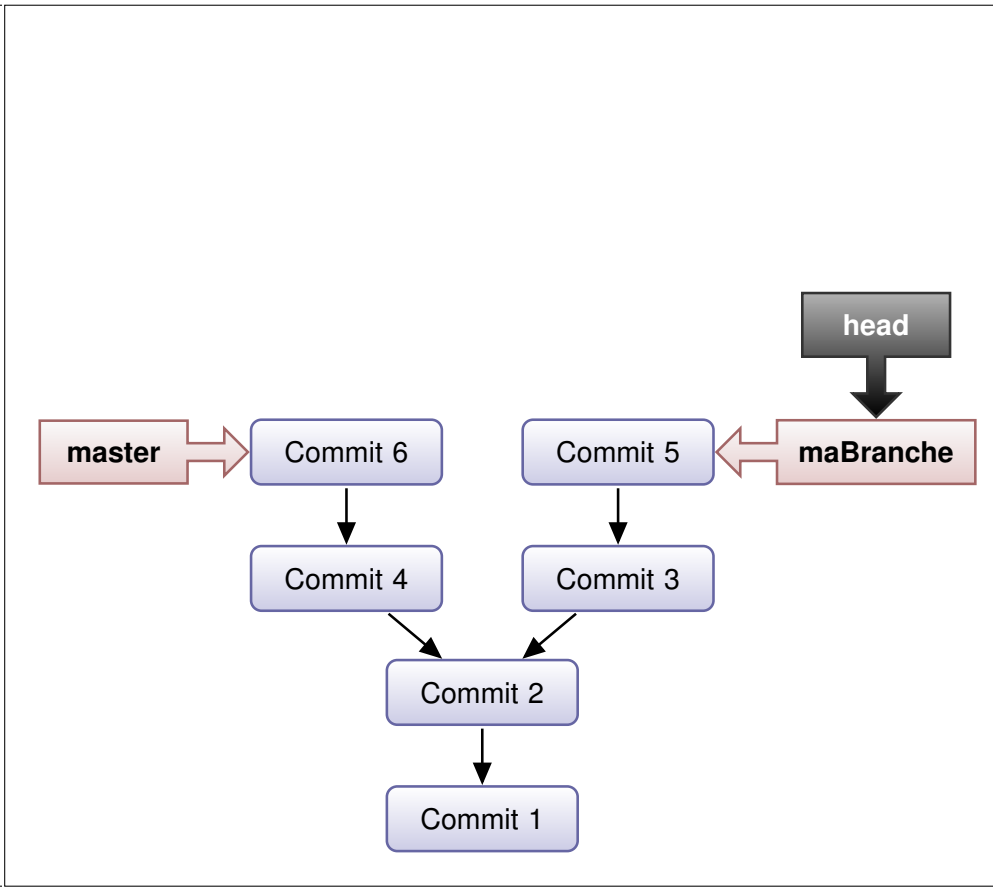
```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt

git branch -d maBranche
```



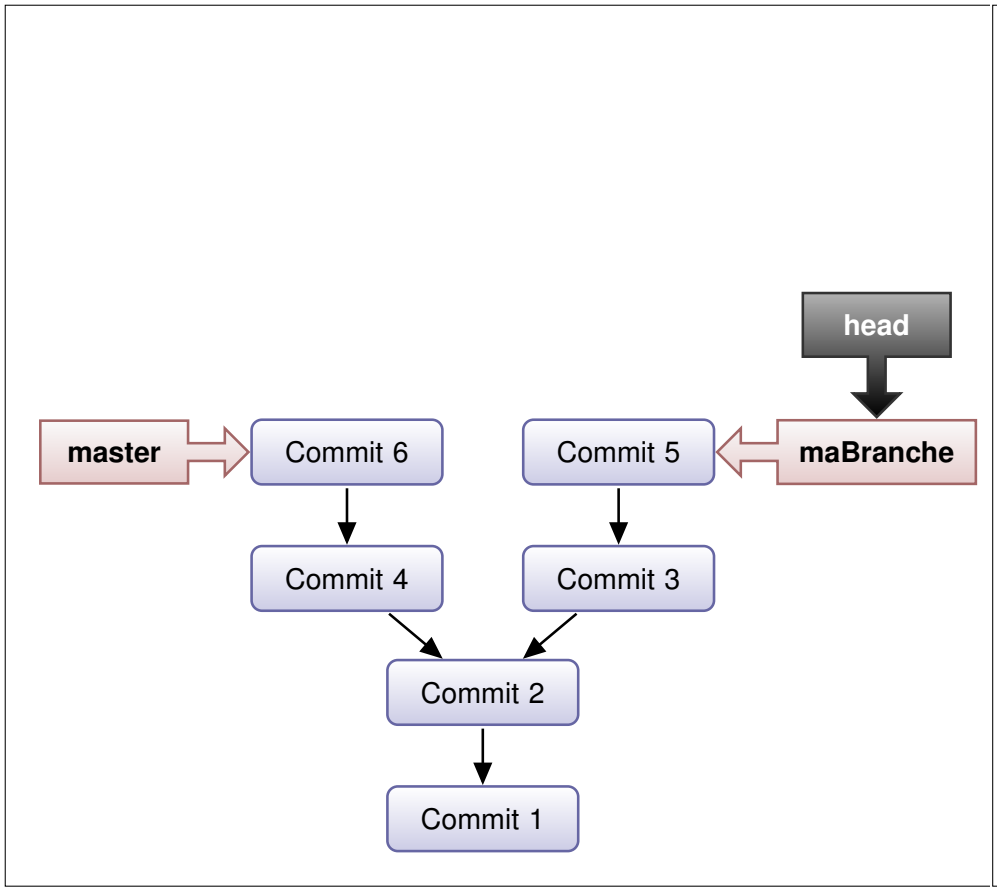
Outline

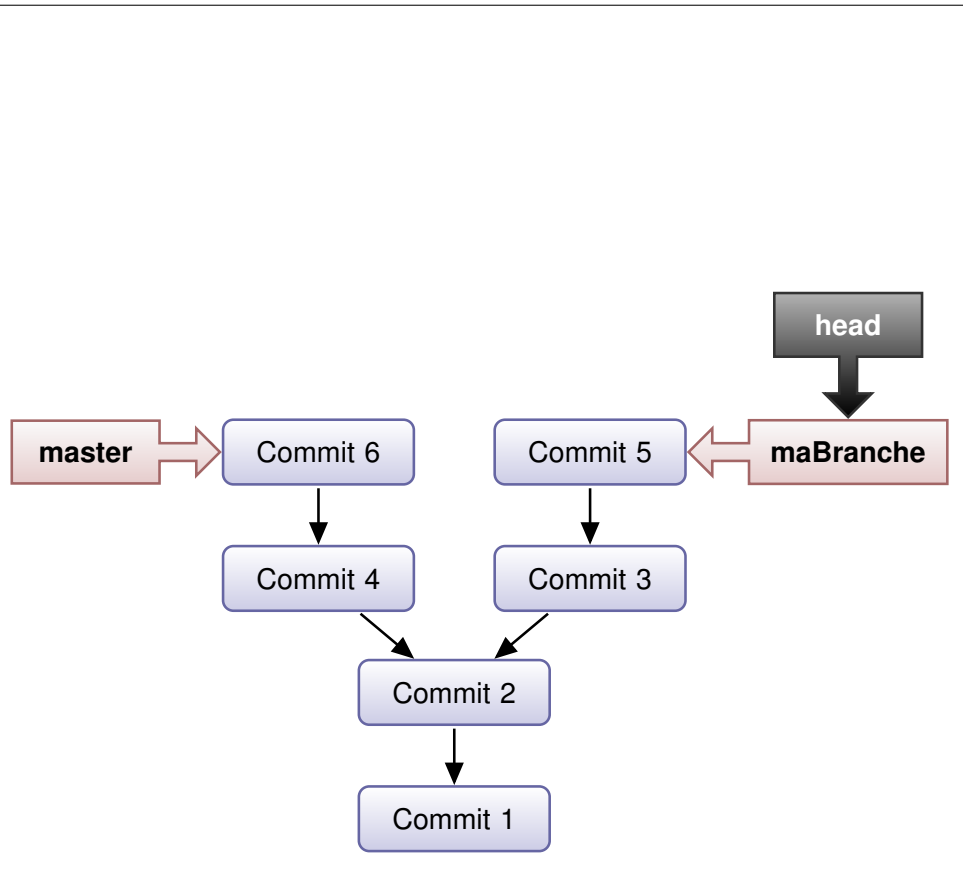
Rebase vs Merge.





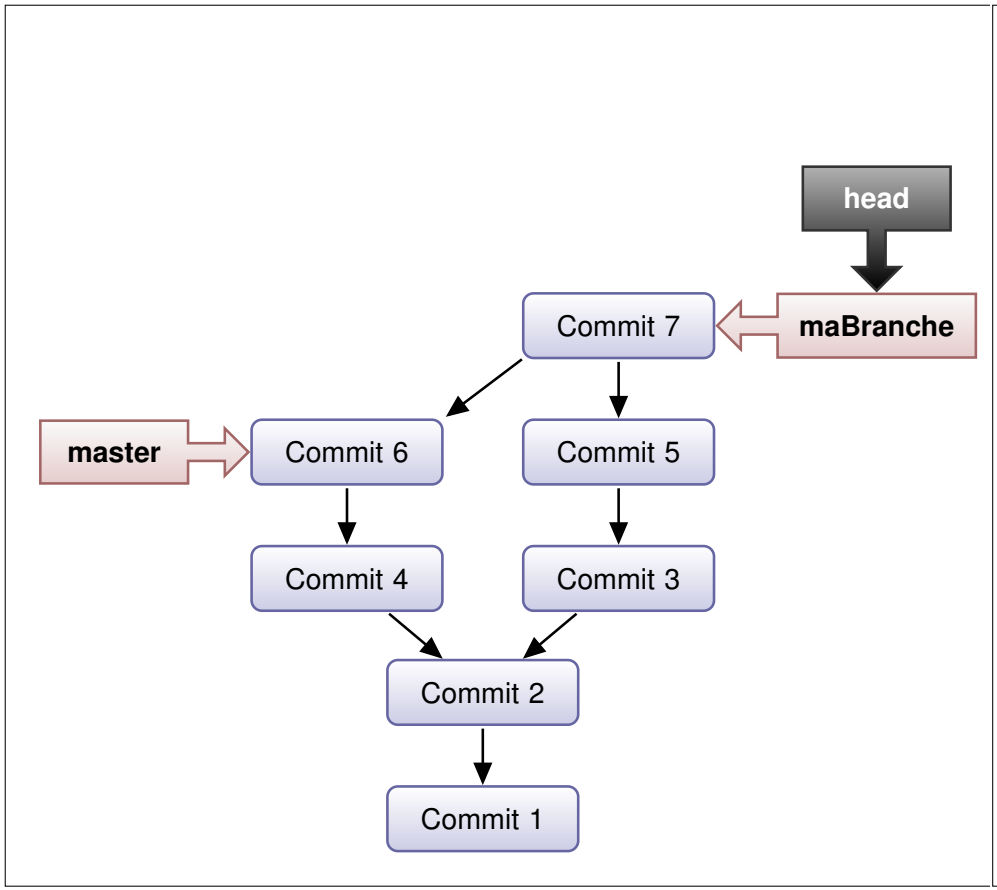
Rebase vs Merge.

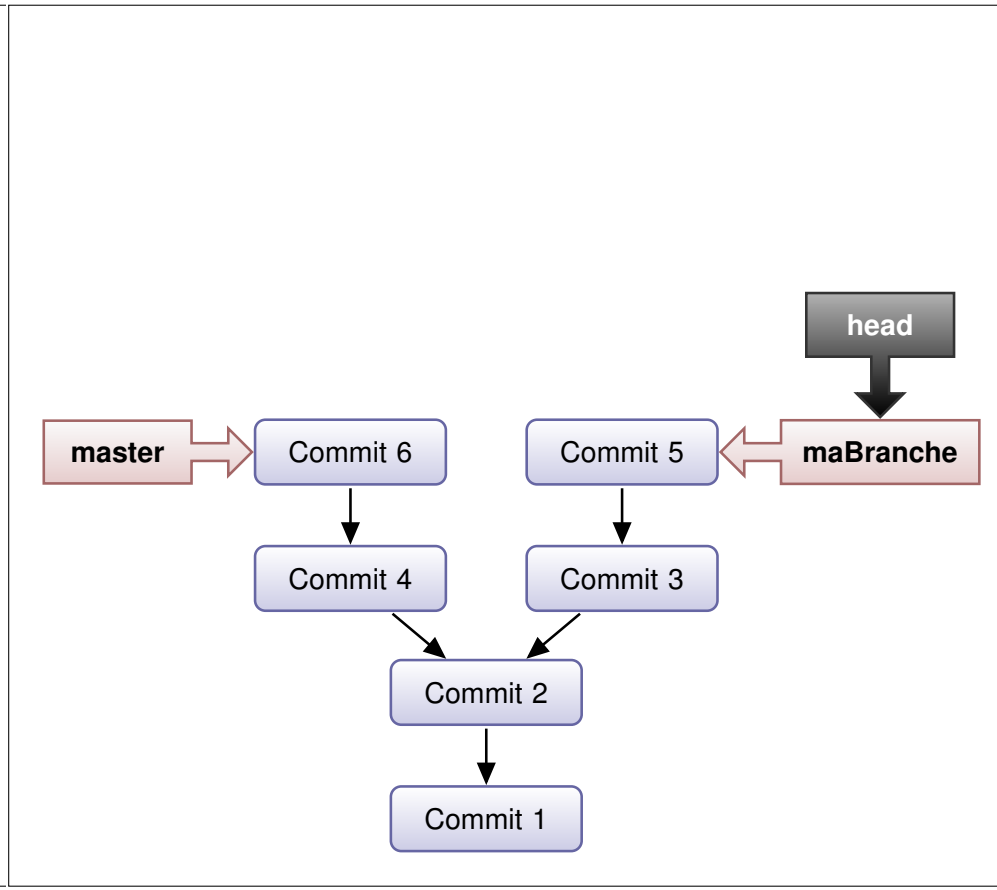





```
git checkout maBranche
git merge master
```

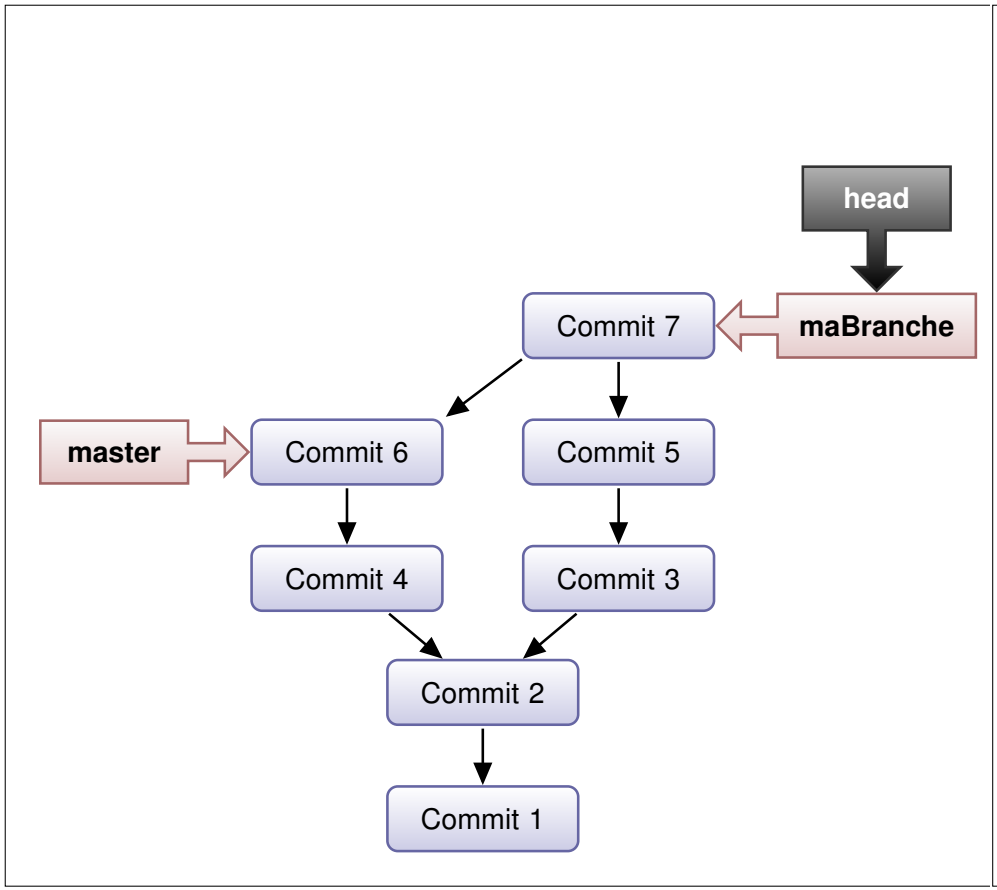
Rebase vs Merge.

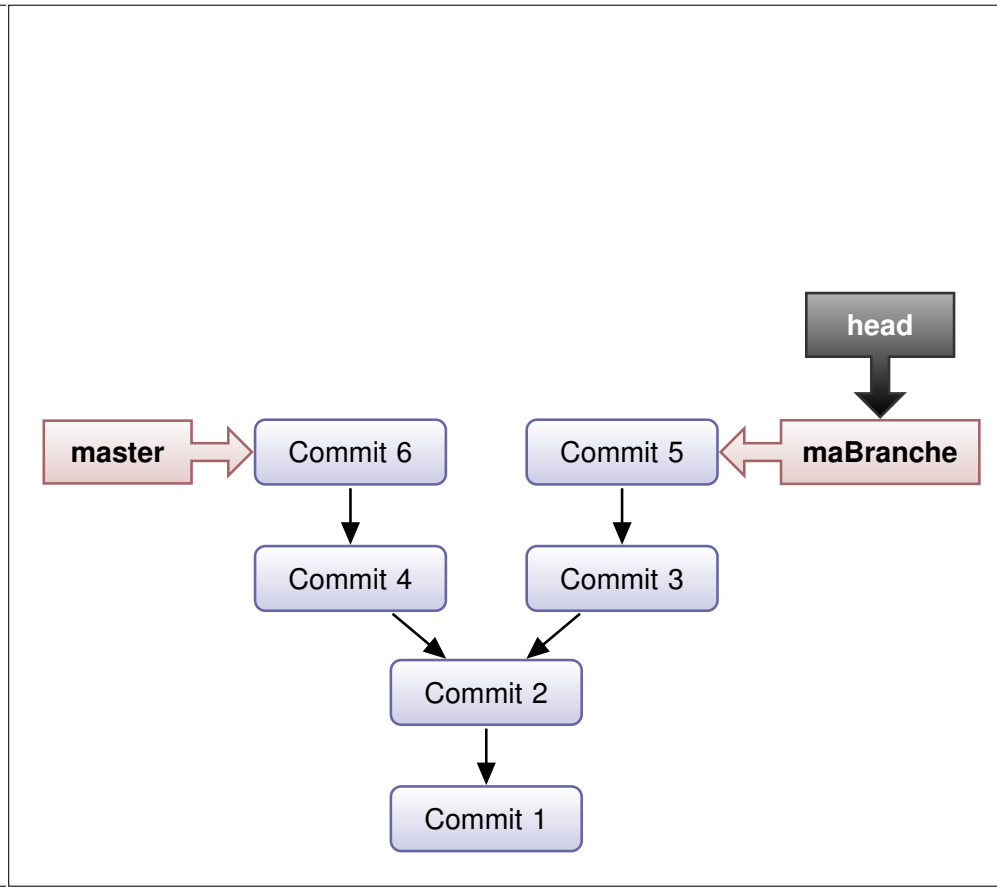





```
git checkout maBranche
git merge master
```

Rebase vs Merge.

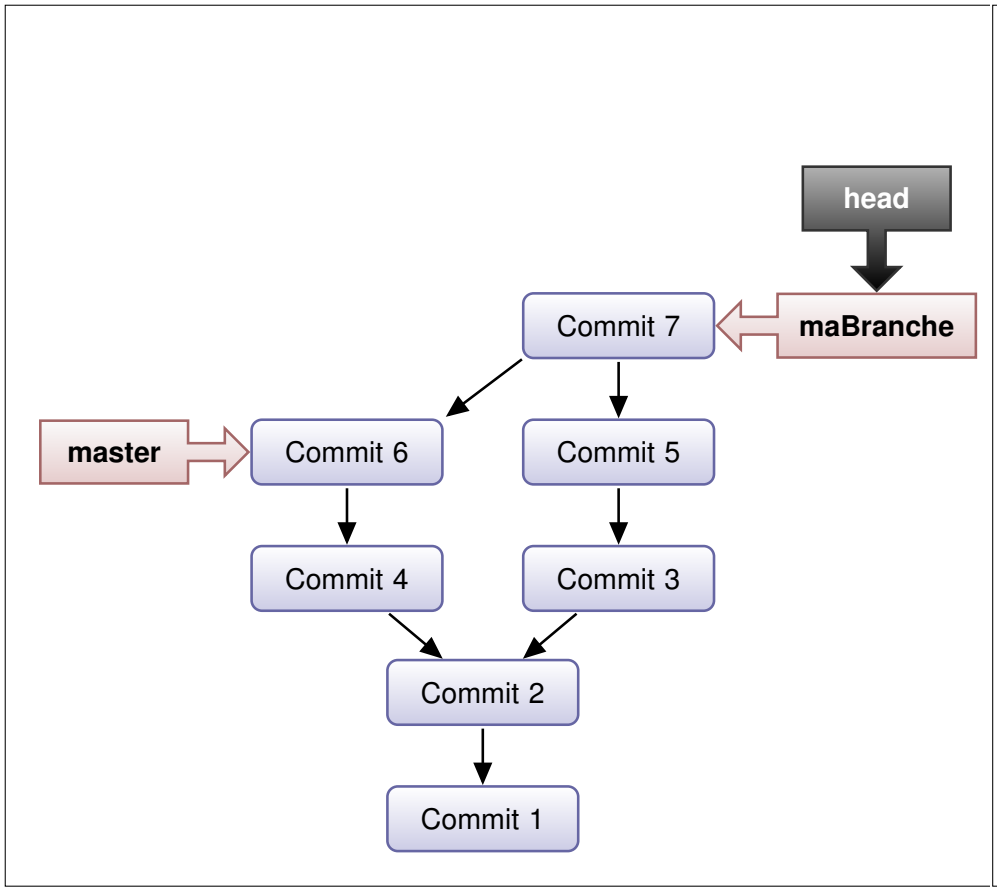


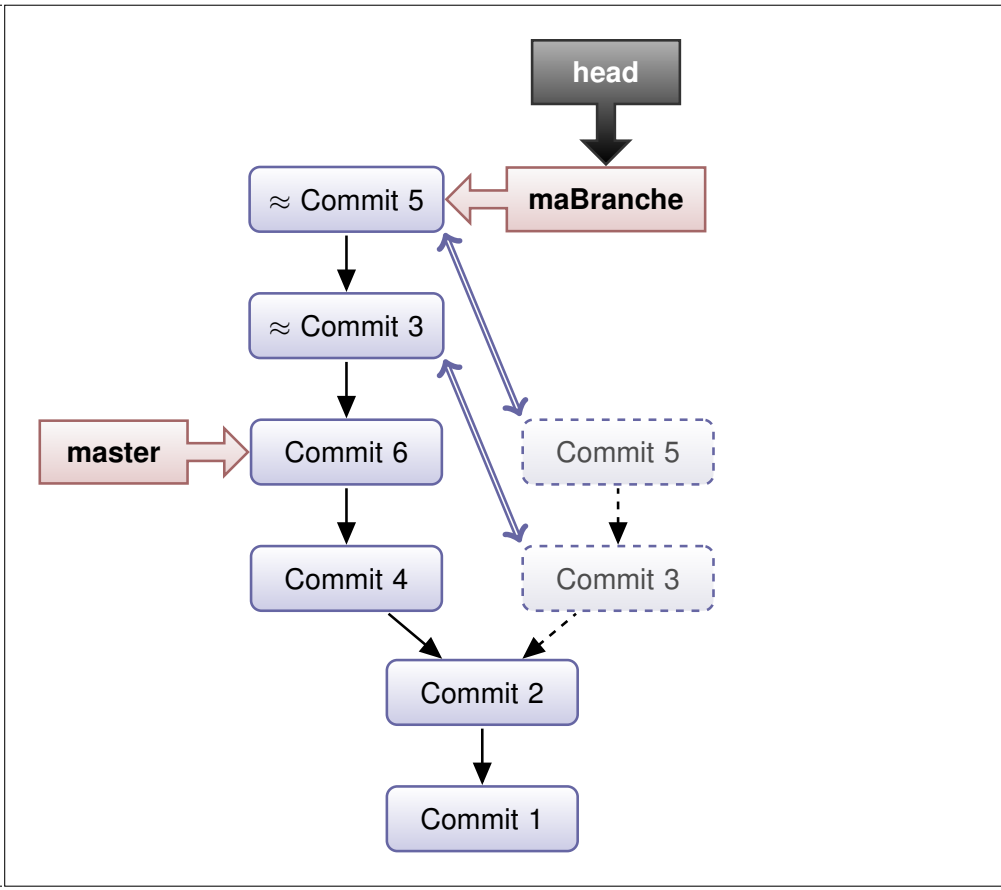



```
git checkout maBranche
git merge master
```

```
git checkout maBranche
git rebase master
```

Rebase vs Merge.

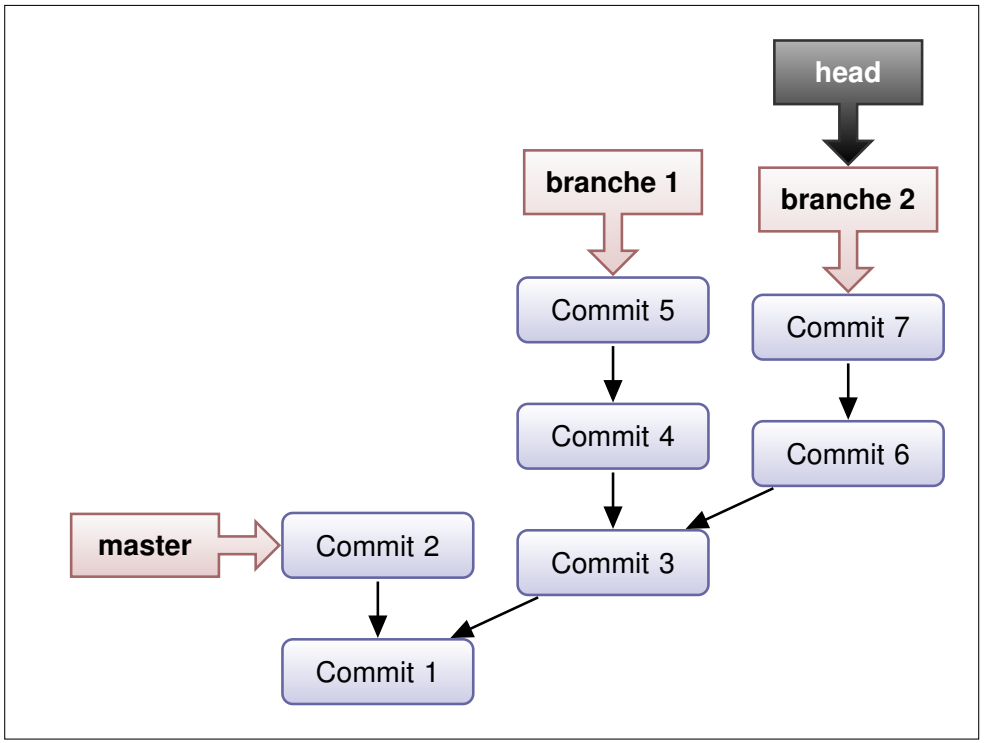




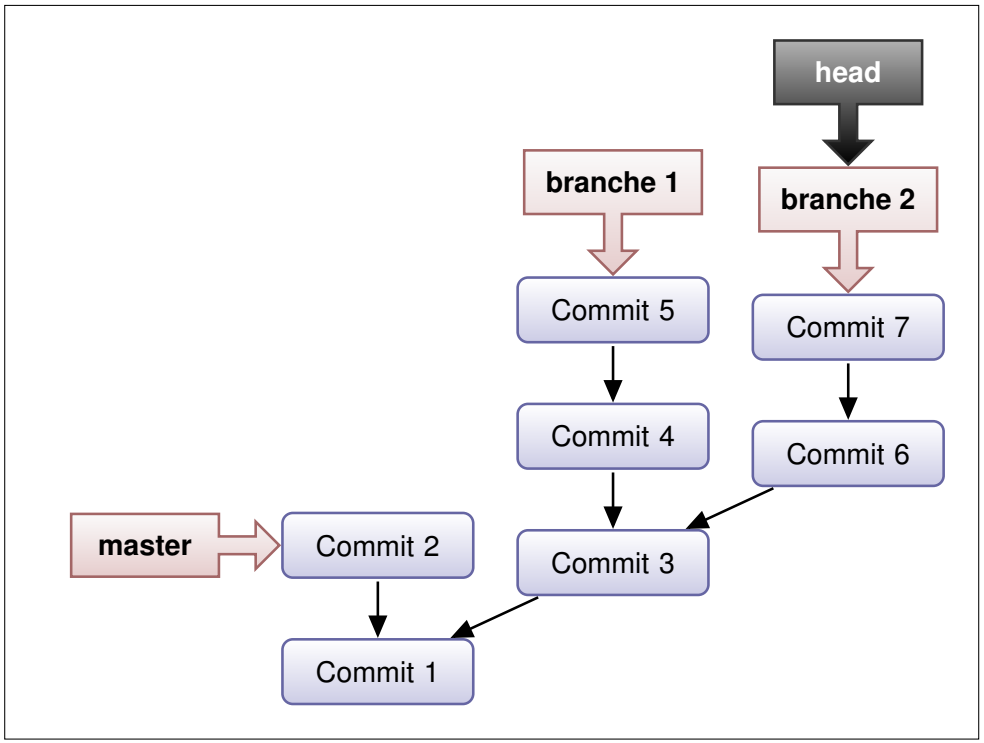

```
git checkout maBranche
git merge master
```

```
git checkout maBranche
git rebase master
```

Rebase vs Merge.

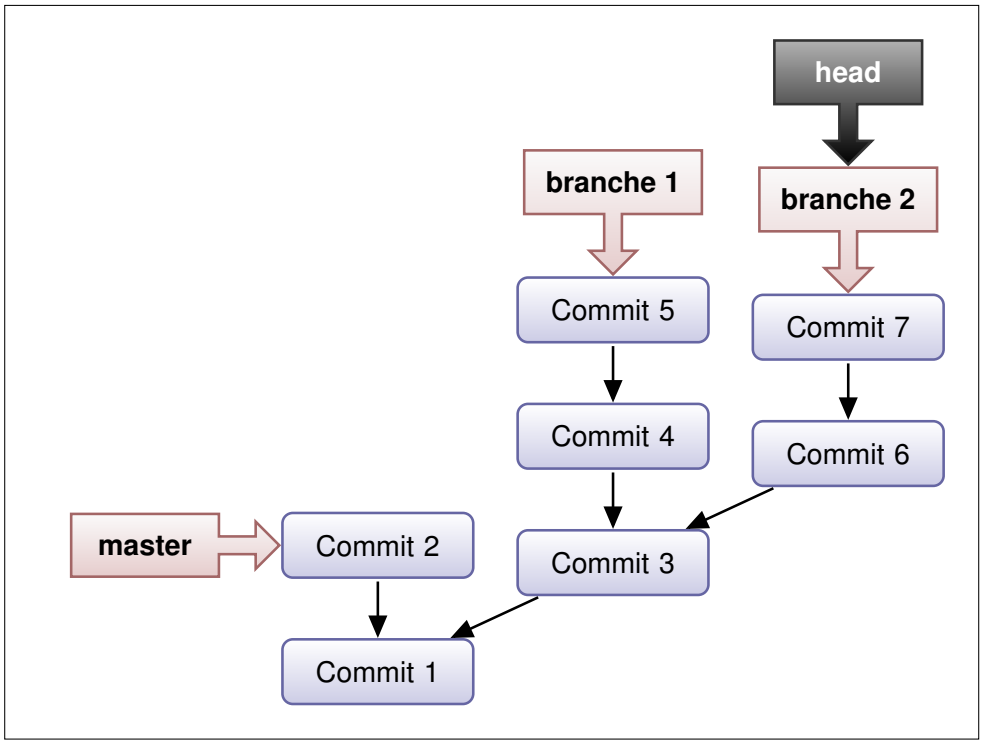


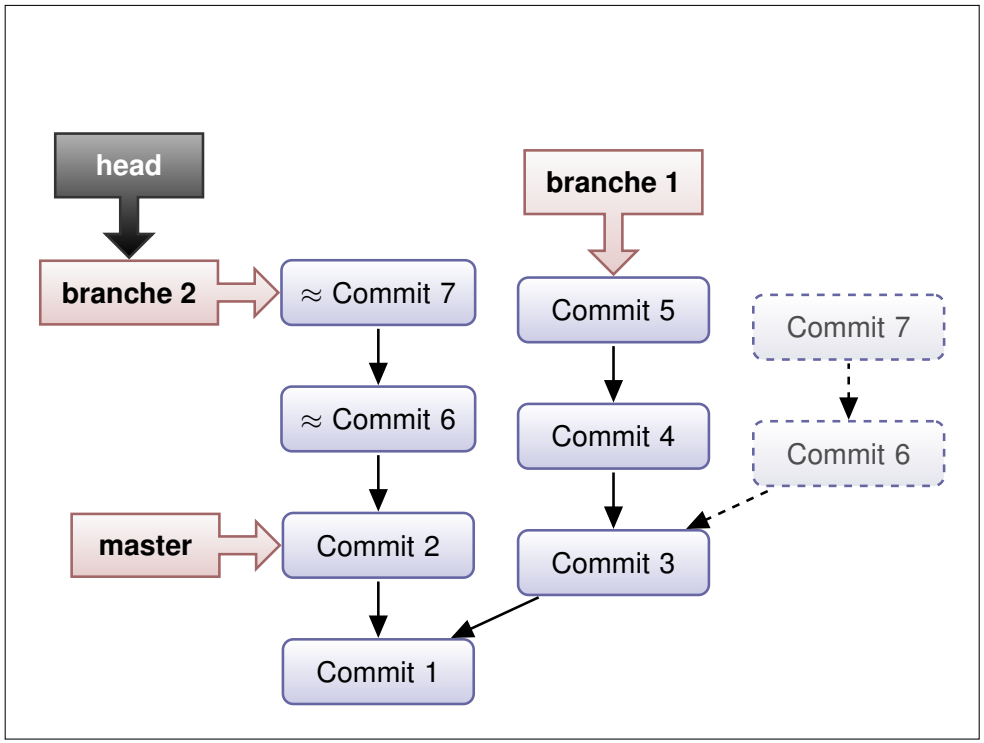
Rebase vs Merge.




```
git checkout branche2
git rebase -onto master branche1 branche2
```

Rebase vs Merge.






```
git checkout branche2
git rebase -onto master branche1 branche2
```

Outline

Les remords

En cas d'erreur sur un commit, git propose 3 types de correction :

revert : pour annuler un commit par un autre commit.

amend : modifier le dernier commit.

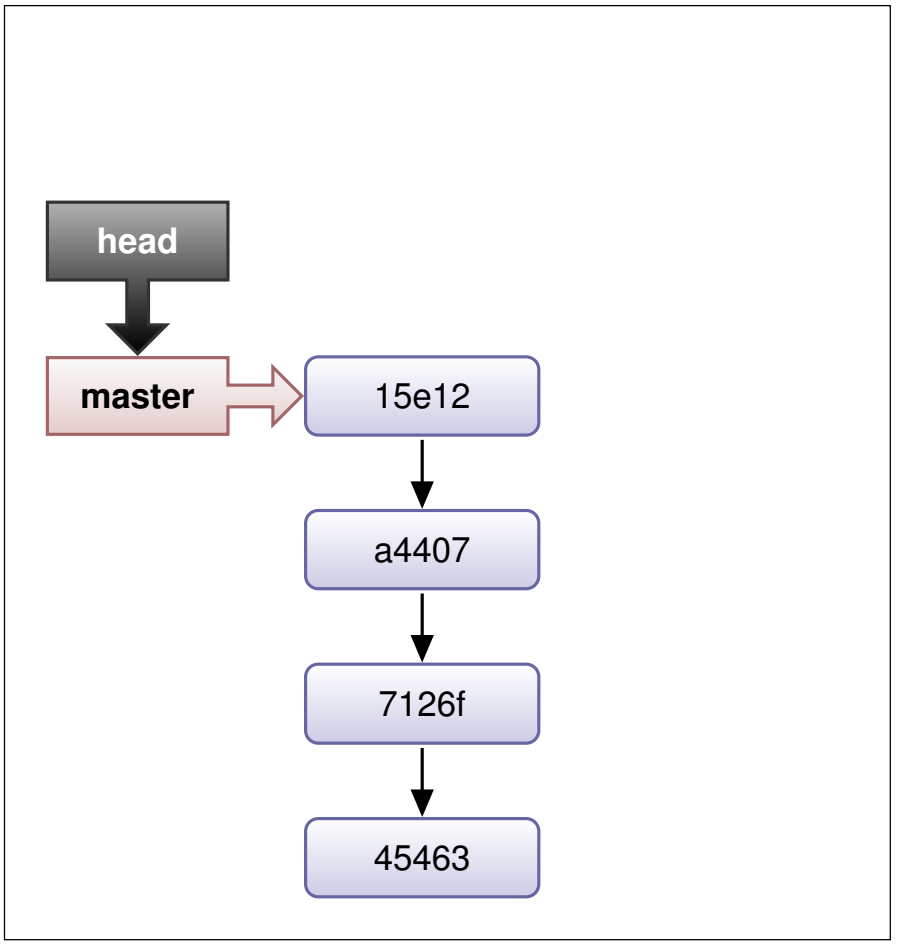
reset : pour rétablir la situation d'un ancien commit.

Attention

Si l'erreur a déjà été rendue publique, la seule bonne pratique est le **revert**. Les autres solutions peuvent conduire à des incohérences.

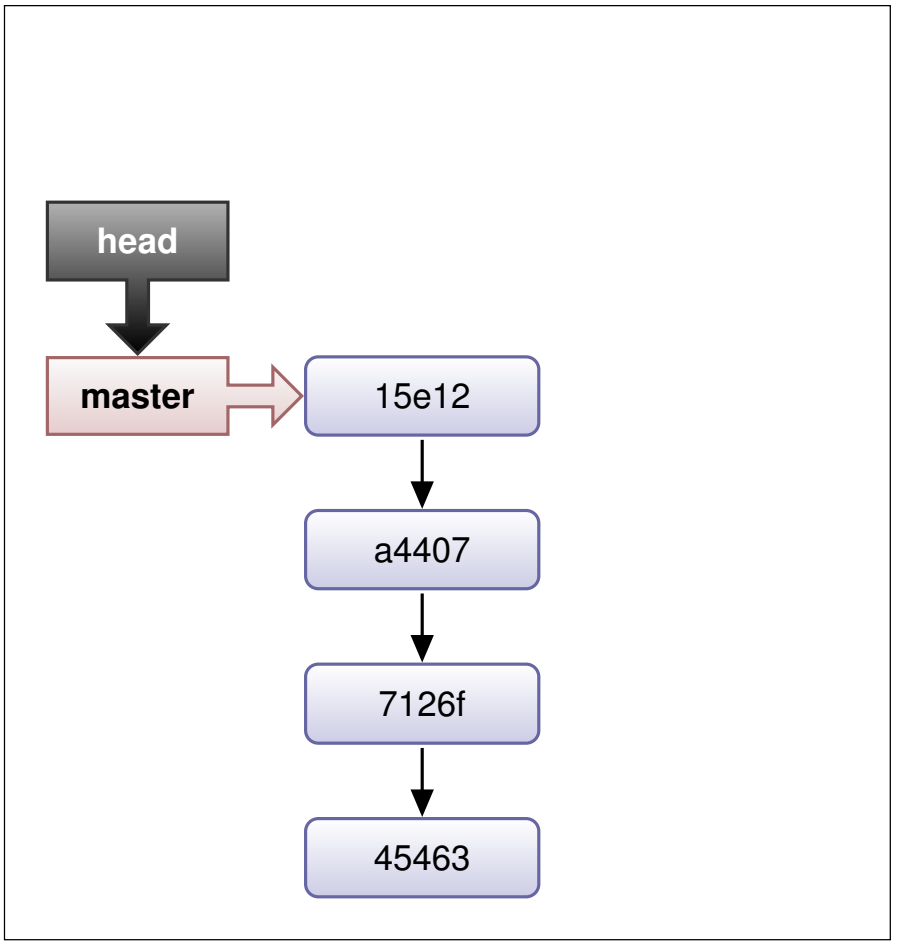
Amend : modification du dernier commit.

```
ls
 foo.txt dir
```



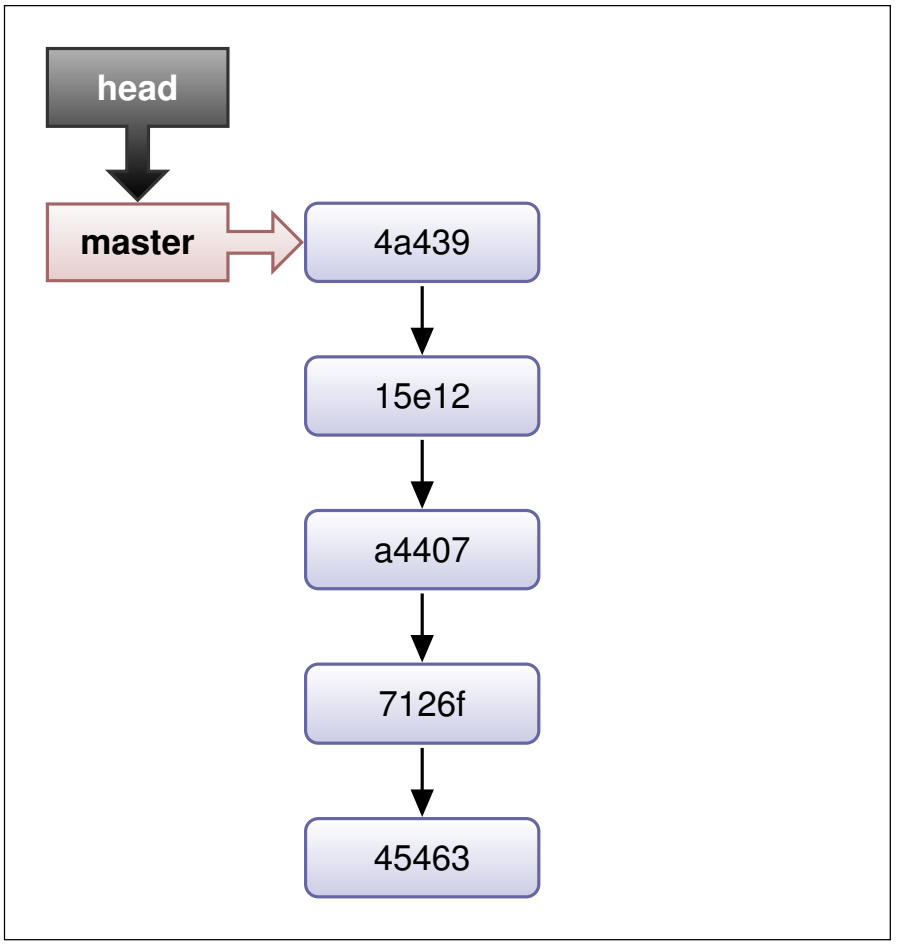
Amend : modification du dernier commit.

```
ls
 foo.txt dir

touch bar.txt
git commit -m "Ajou d'un fichier."
```



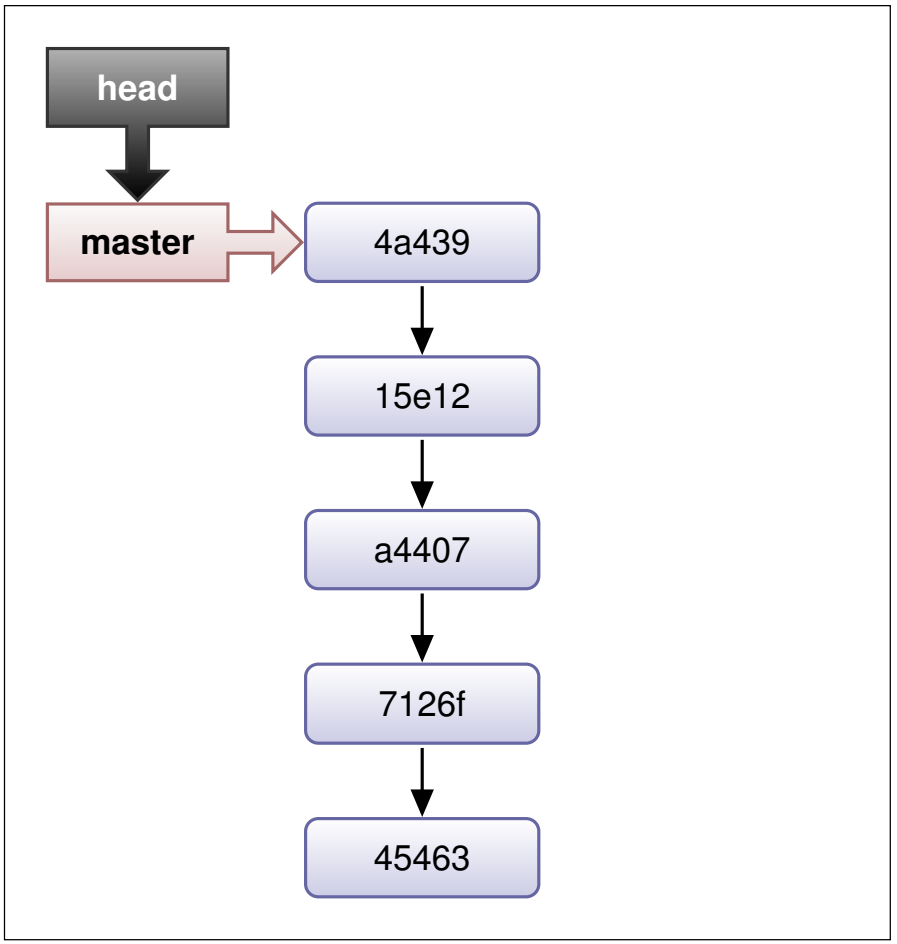
Amend : modification du dernier commit.

```
ls
 foo.txt dir

touch bar.txt
git commit -m "Ajou d'un fichier."
```



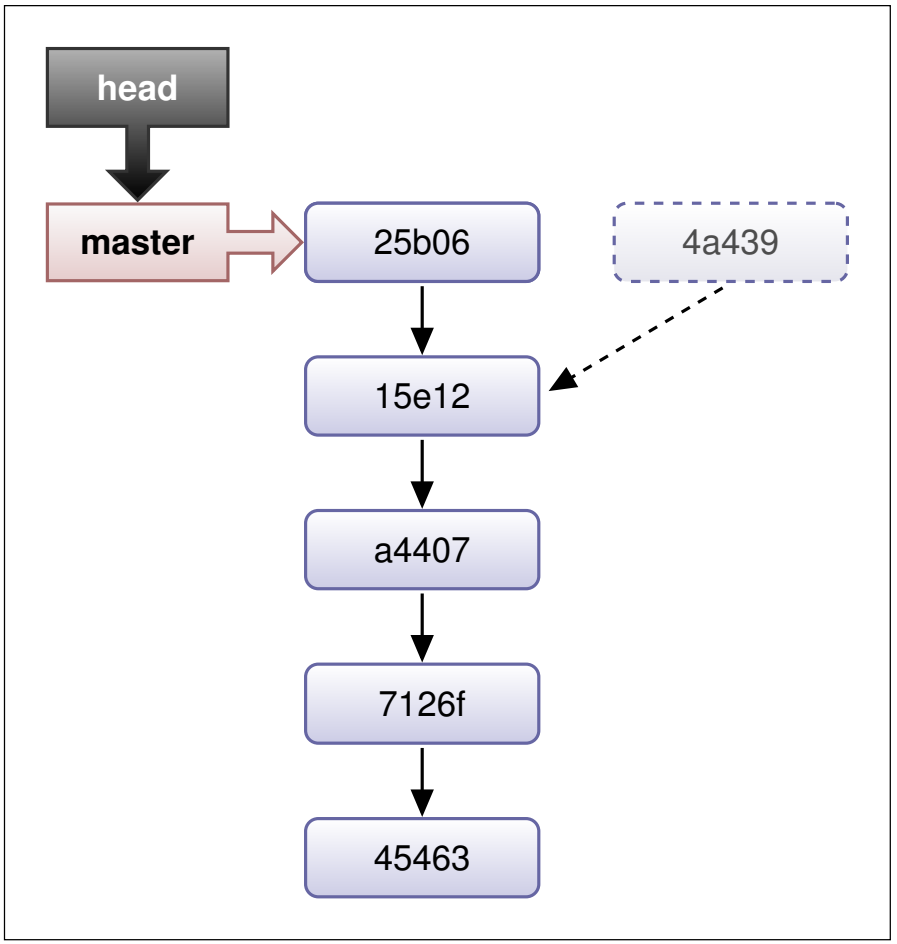
Amend : modification du dernier commit.

```
ls
 foo.txt dir

touch bar.txt
git commit -m "Ajou d'un fichier."

git add bar.txt

git commit --amend -m "Ajout d'un
fichier."
```



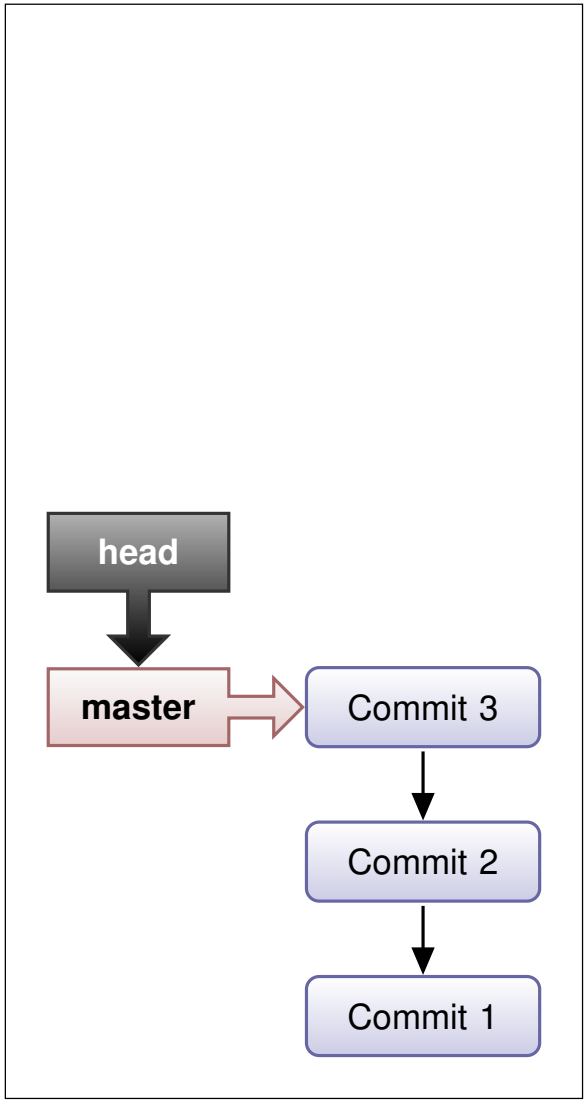
Amend : modification du dernier commit.

```
ls
 foo.txt dir

touch bar.txt
git commit -m "Ajou d'un fichier."

git add bar.txt

git commit --amend -m "Ajout d'un
fichier."
```



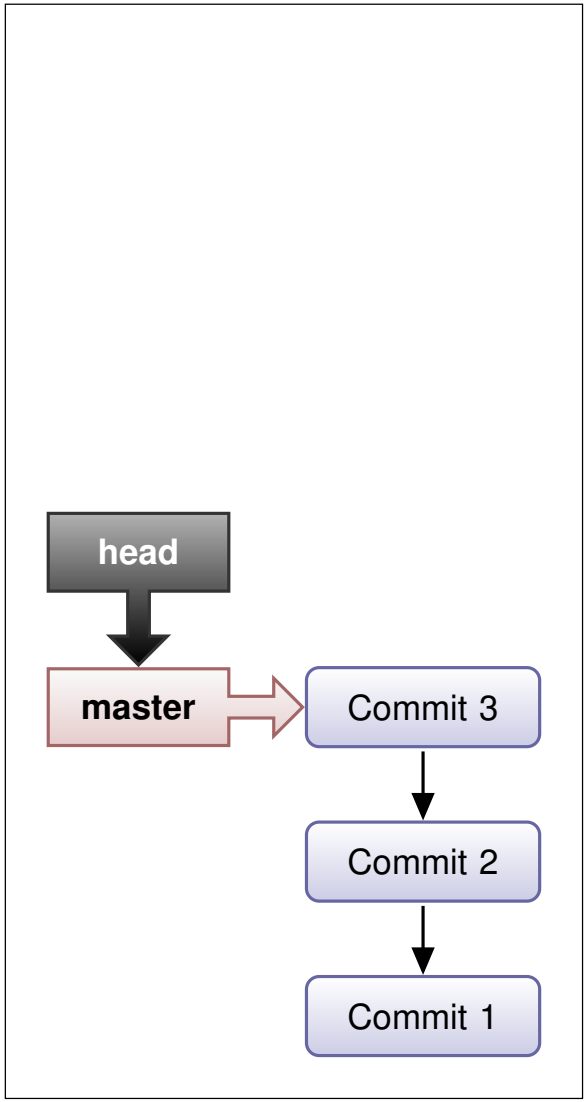
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2
```



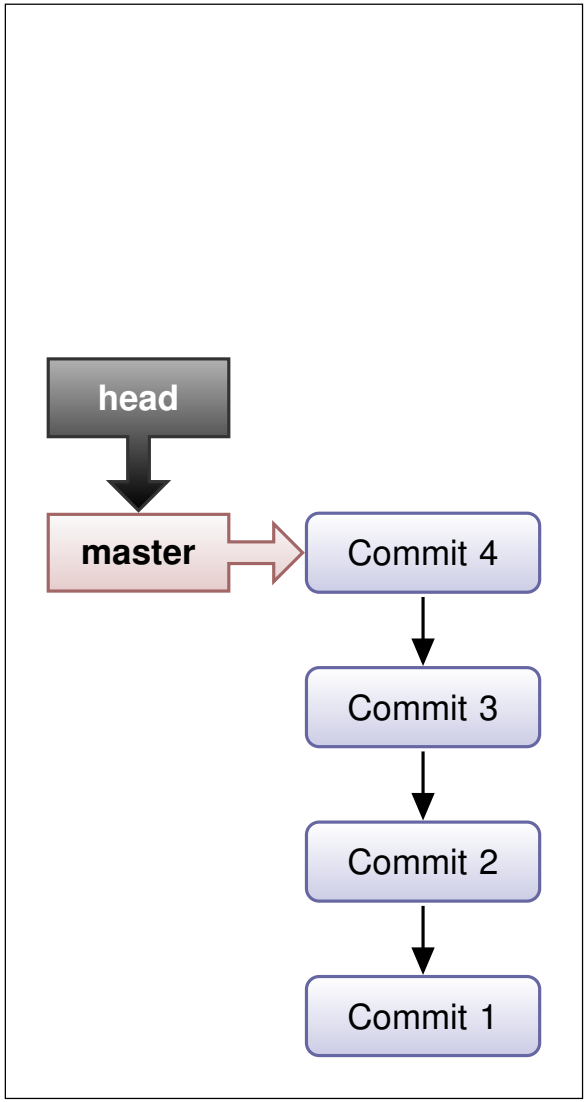
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



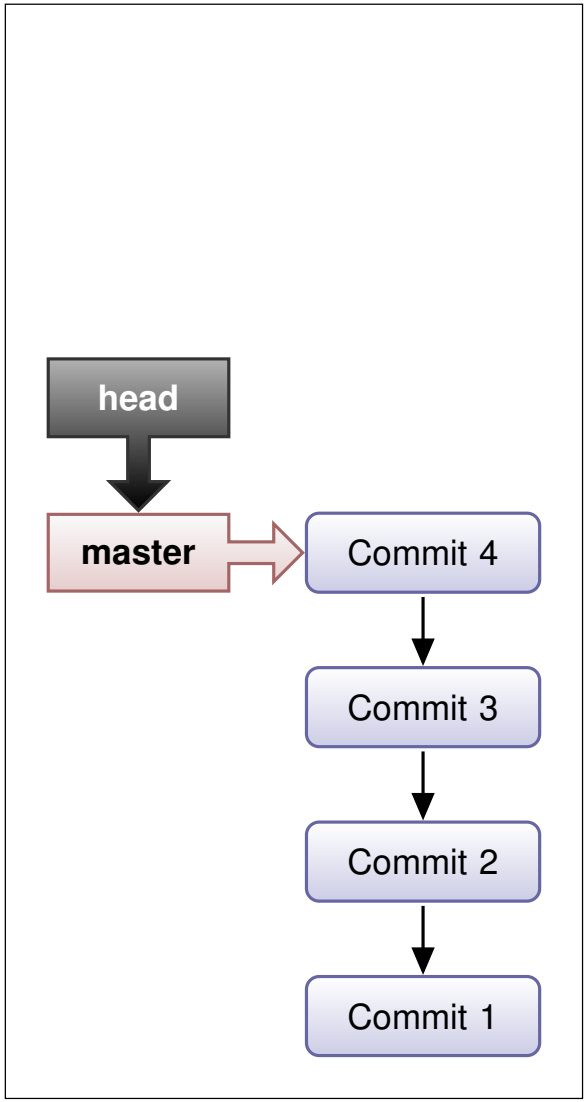
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



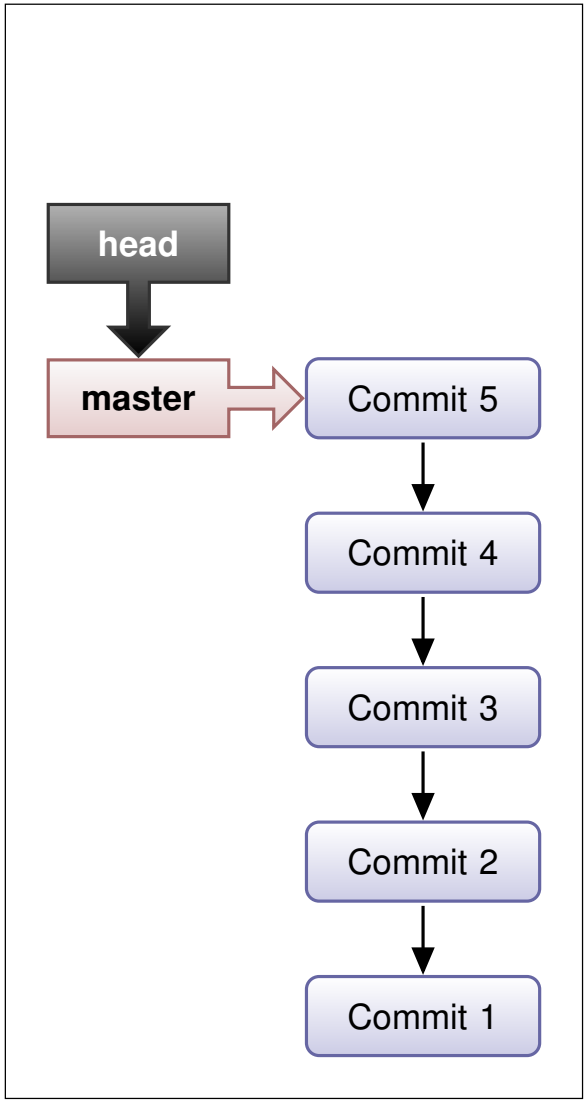
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "Deuxieme version de F2" > fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"
```



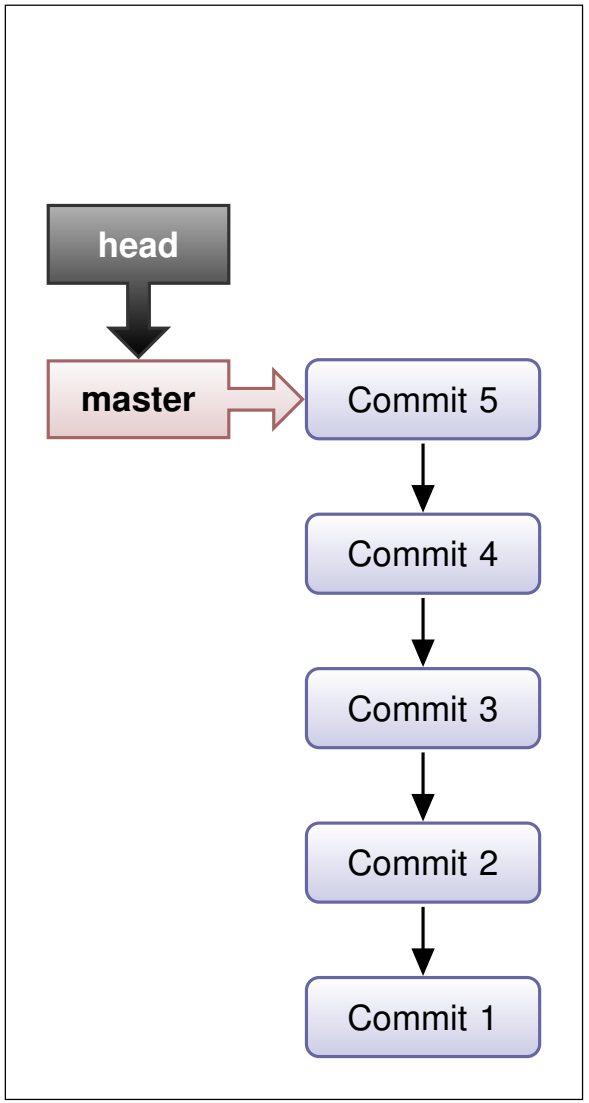
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "Deuxieme version de F2" > fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"
```



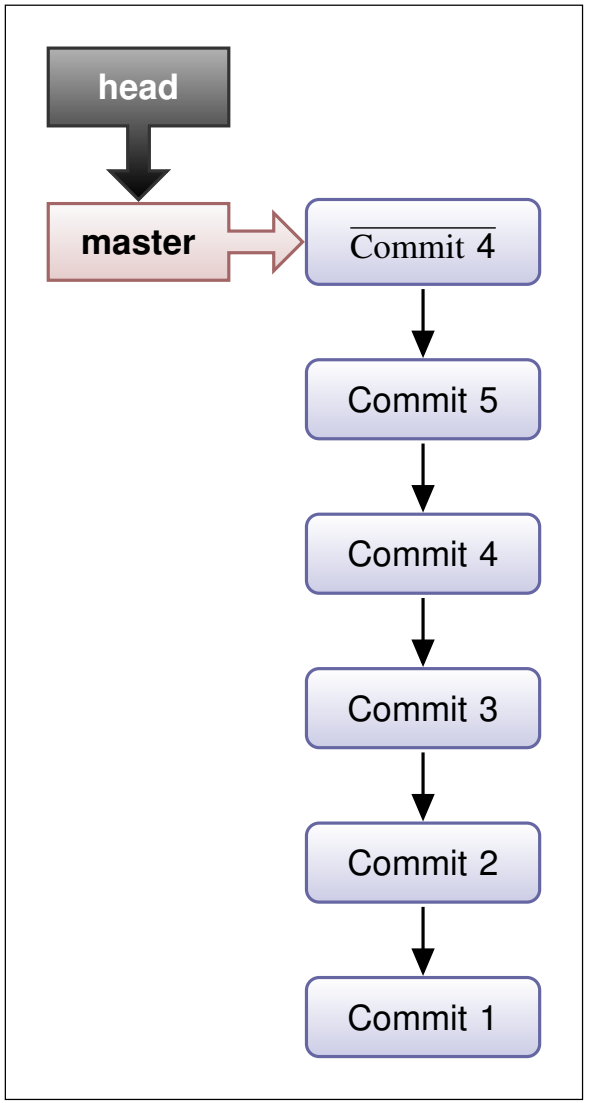
Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "Deuxieme version de F2" > fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"

git revert HEAD^
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Deuxieme version de F2
```



Git revert : annulation par commit.

```
git branch master
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Premiere version de F2

echo "Deuxieme version de F1" > fichier1.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

echo "Deuxieme version de F2" > fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"

git revert HEAD^
cat fichier1.txt
 Premiere version de F1
cat fichier2.txt
 Deuxieme version de F2
```



Git reset : "suppression" de commit.

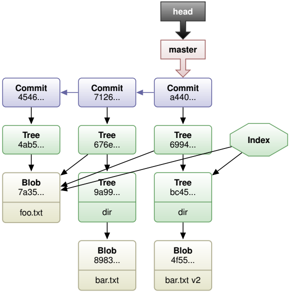
git reset permet de "supprimer" un ou plusieurs commits.

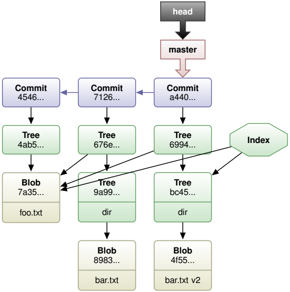
Dans les faits, les objets liés à ces commits ne seront vraiment effacés qu'après un appel à **git gc** et s'ils sont suffisamment vieux (15 jours par défaut).

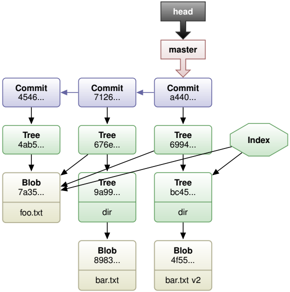
Il existe trois types de reset :

1. **git reset --hard** :
   - restore la référence du commit (de la branche active)
   - restore l'index
   - restore les données
2. **git reset** :
   - restore la référence du commit (de la branche active)
   - restore l'index
3. **git reset --soft** :
   - restore la référence du commit (de la branche active)

Les différents type de reset.





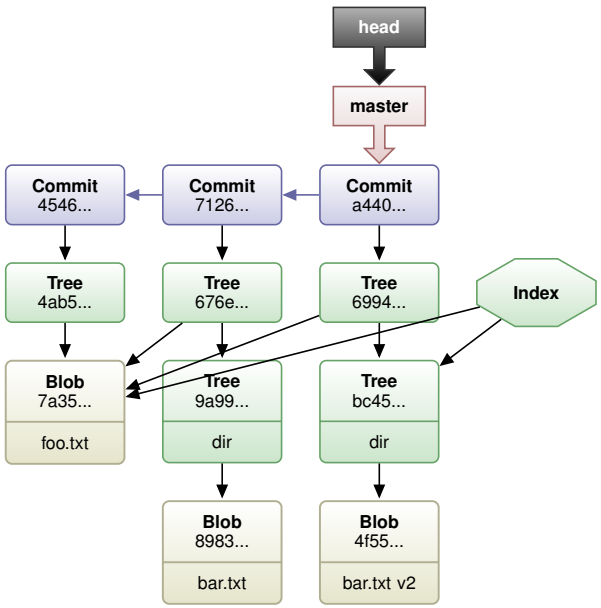



```
ls -R
 .: foo.txt
 dir: bar.txt
```

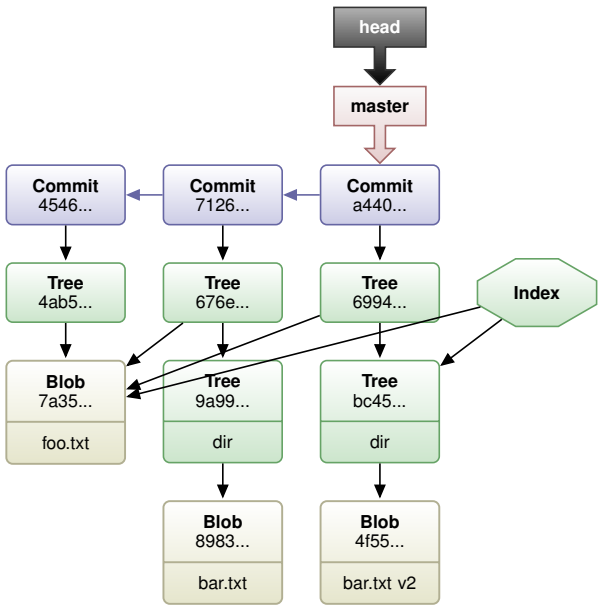
```
ls -R
 .: foo.txt
 dir: bar.txt
```

```
ls -R
 .: foo.txt
 dir: bar.txt
```

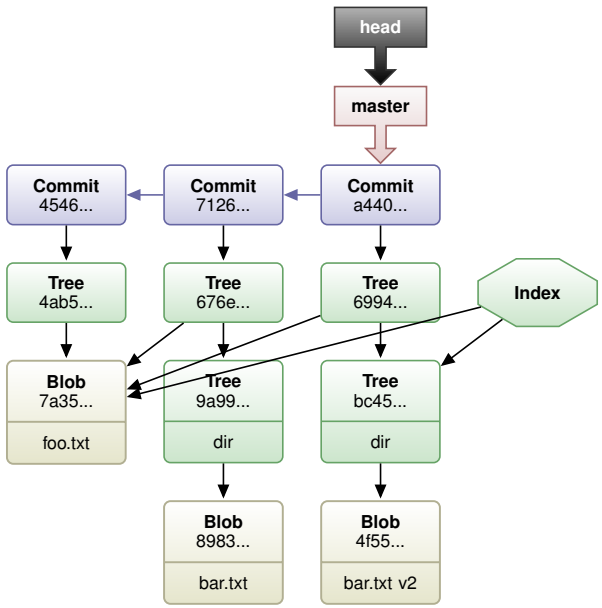
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
```

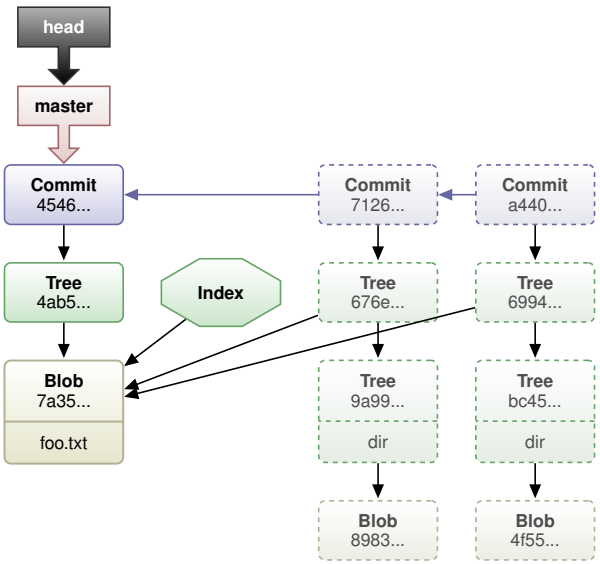



```
ls -R
  .: foo.txt
  dir: bar.txt
```

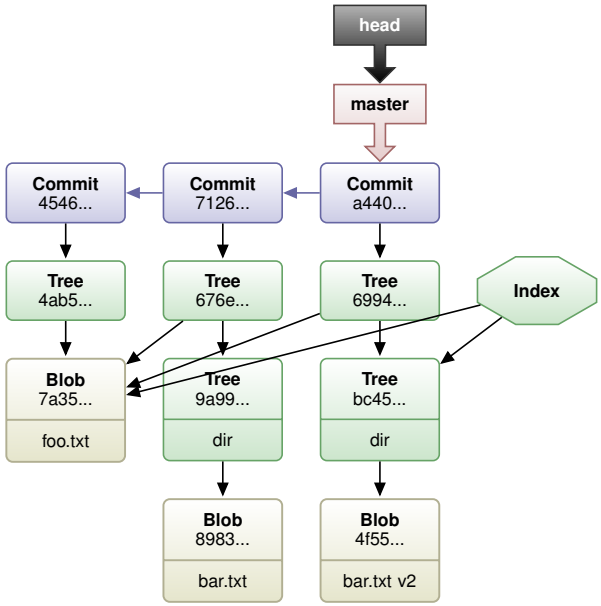



```
ls -R
  .: foo.txt
  dir: bar.txt
```

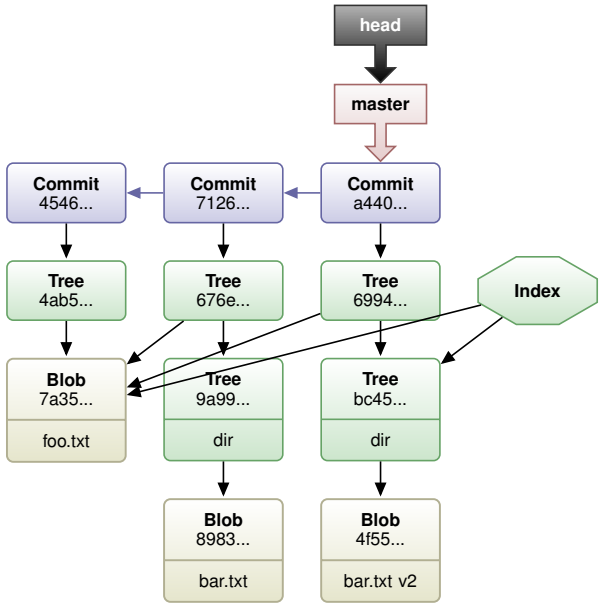
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
```

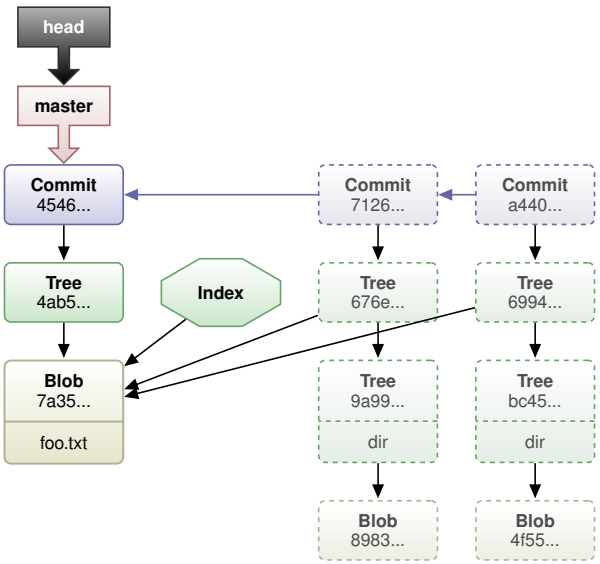



```
ls -R
  .: foo.txt
  dir: bar.txt
```

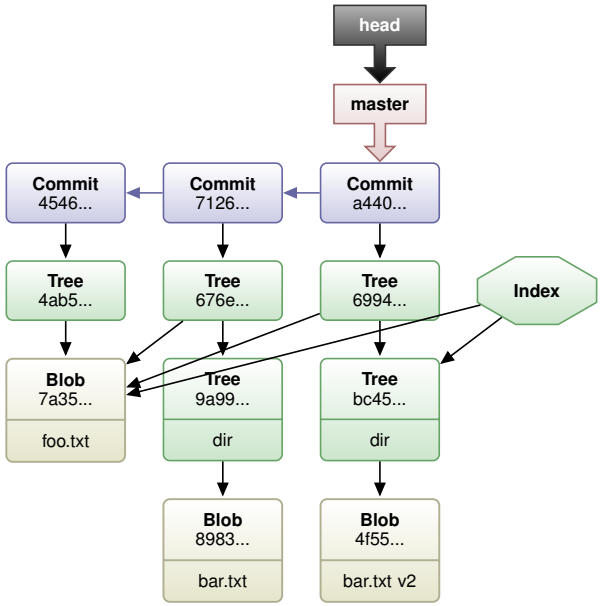



```
ls -R
  .: foo.txt
  dir: bar.txt
```

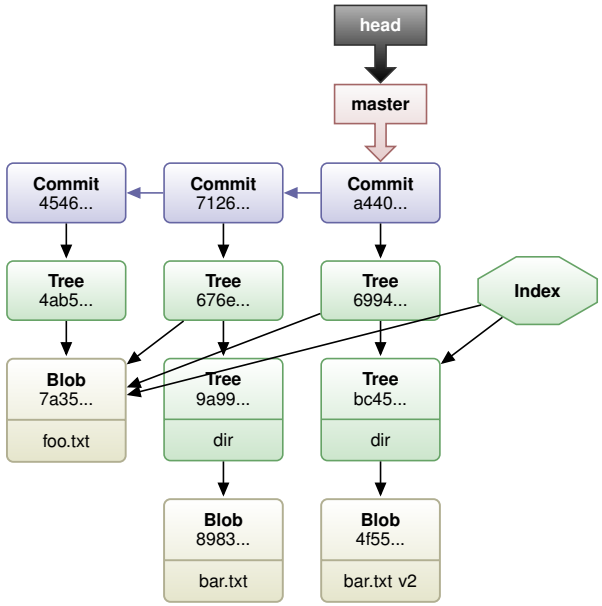
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

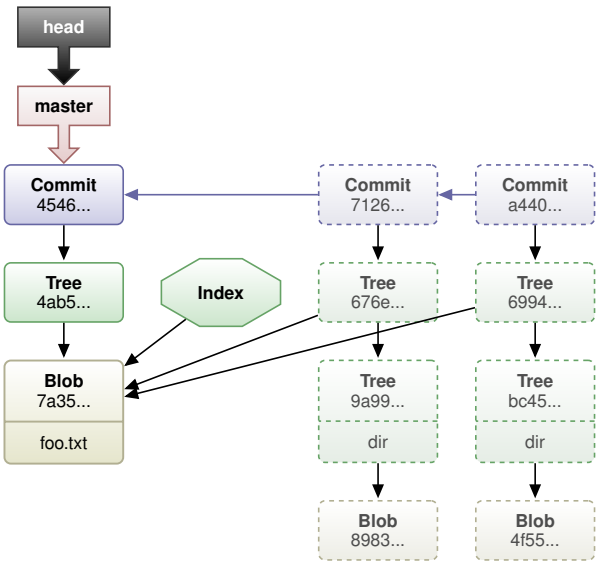



```
ls -R
  .: foo.txt
  dir: bar.txt
```

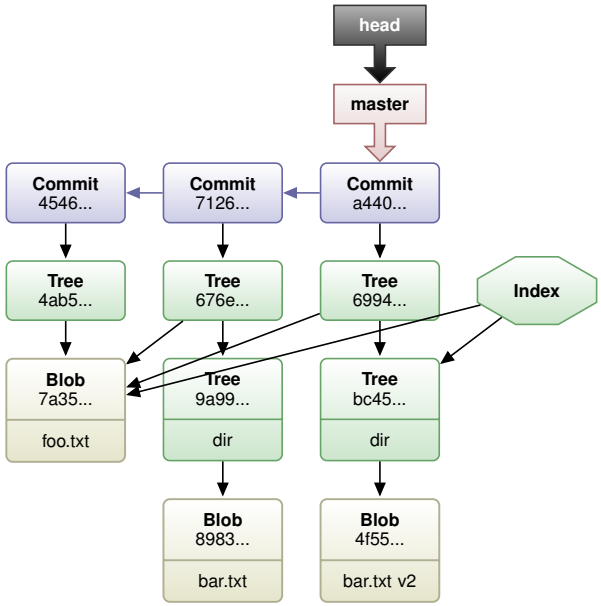



```
ls -R
  .: foo.txt
  dir: bar.txt
```

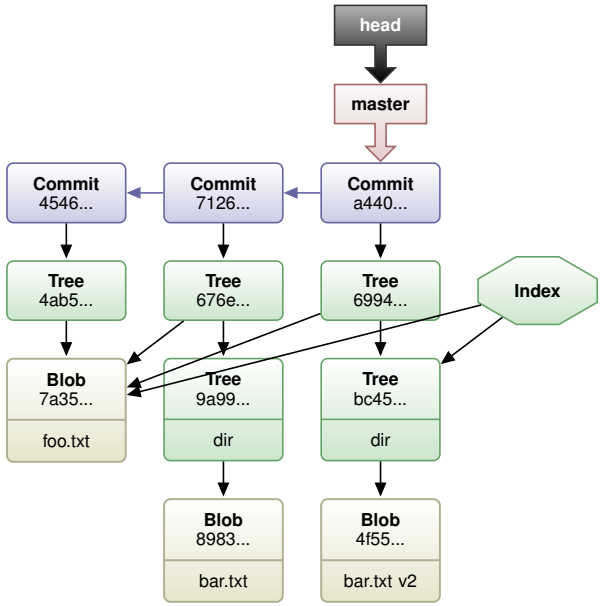
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

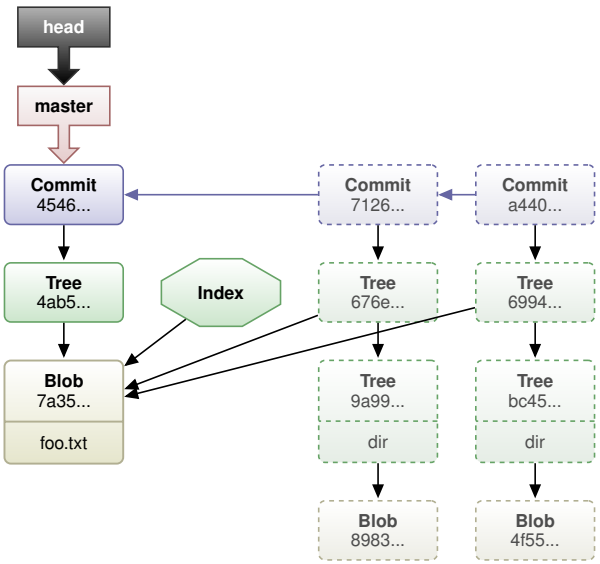



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
```

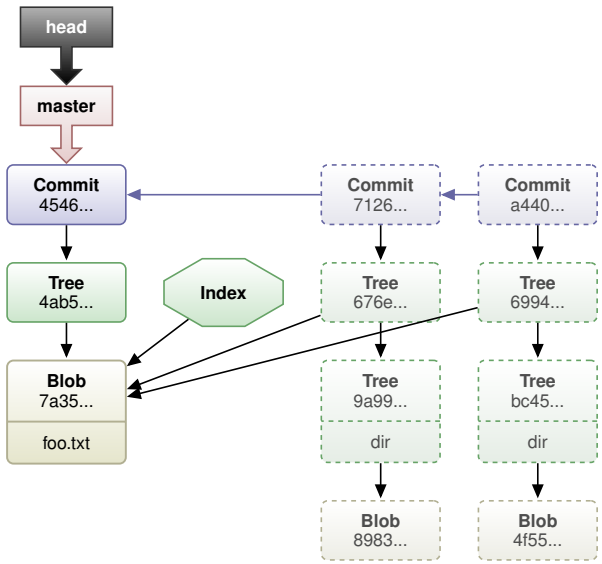



```
ls -R
  .: foo.txt
  dir: bar.txt
```

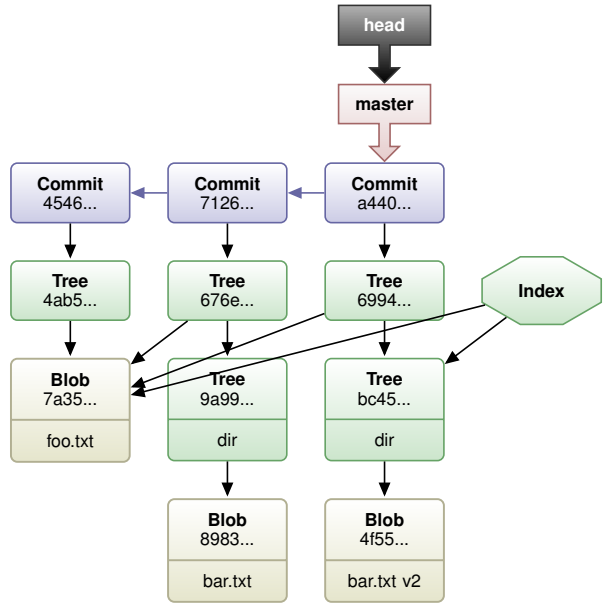
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

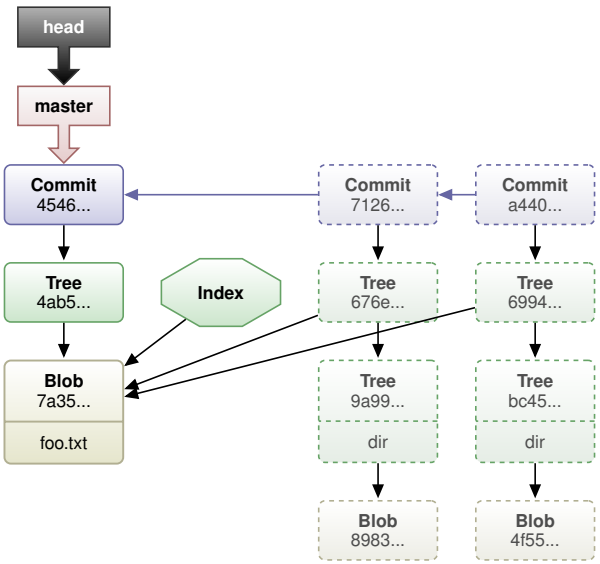



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
```

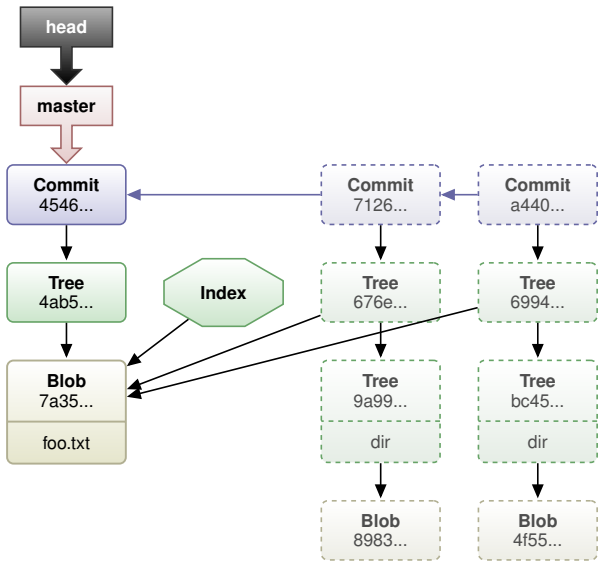



```
ls -R
  .: foo.txt
  dir: bar.txt
```

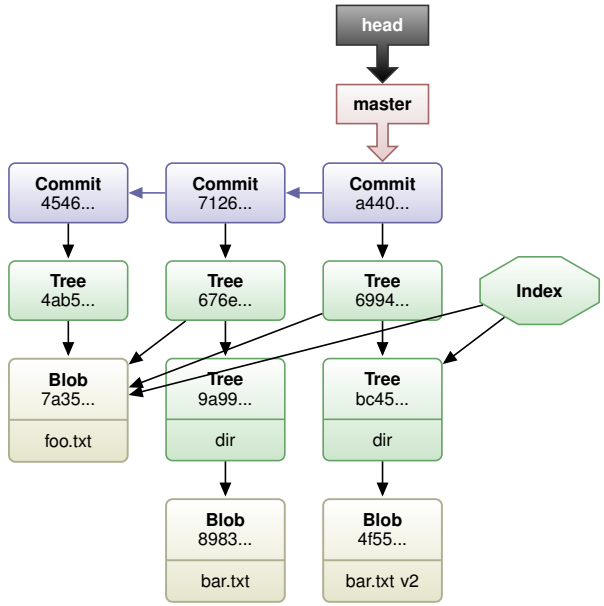
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

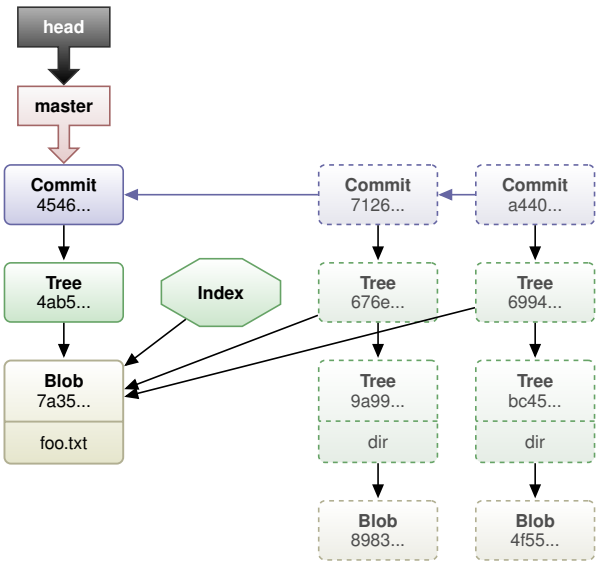



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
ls -R
  .: foo.txt
  dir: bar.txt
```

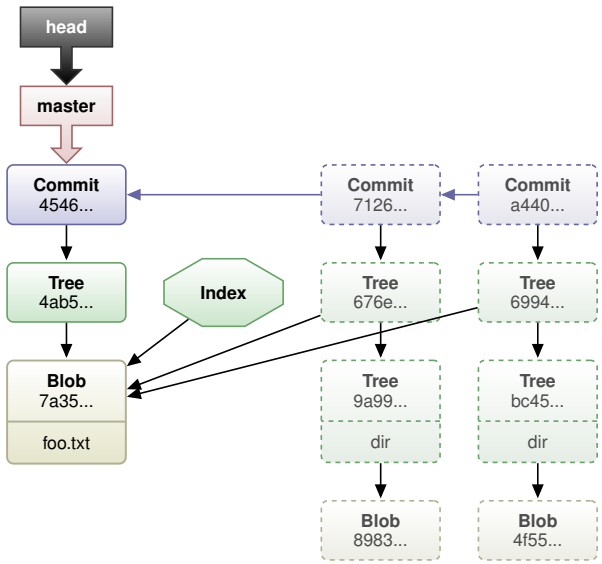



```
ls -R
  .: foo.txt
  dir: bar.txt
```

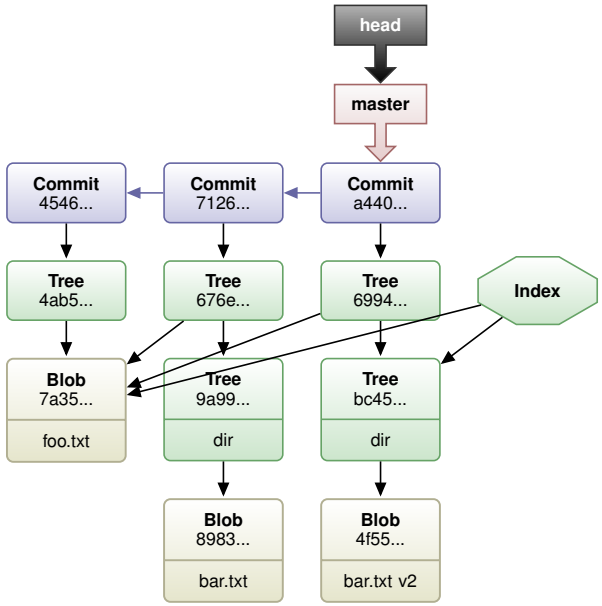
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

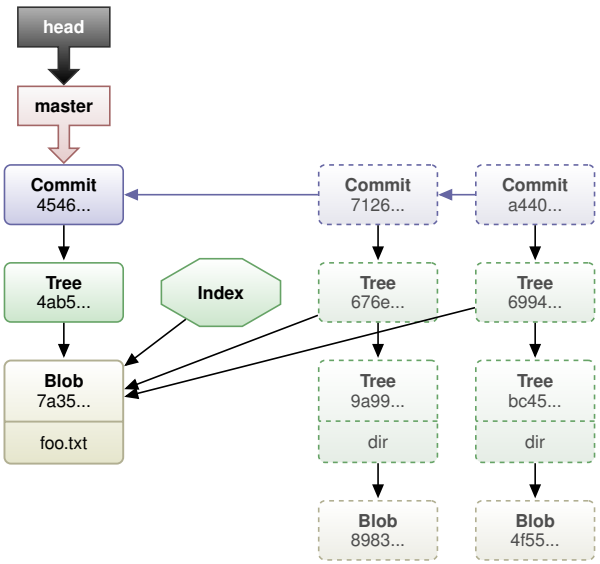



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
ls -R
  .: foo.txt
  dir: bar.txt
```

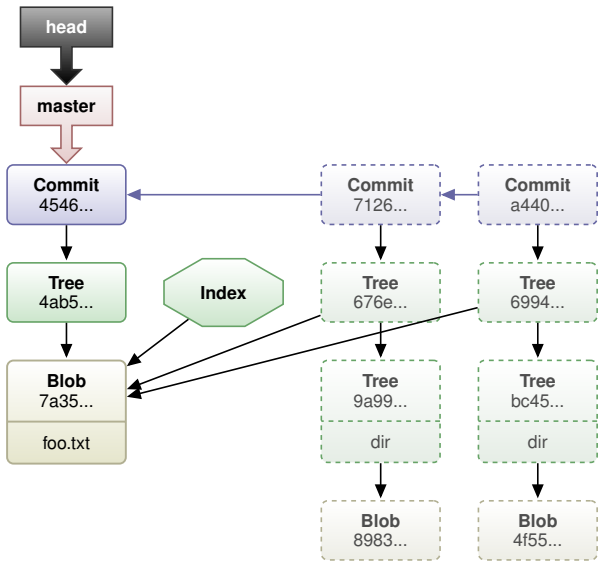



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --soft 4546
```

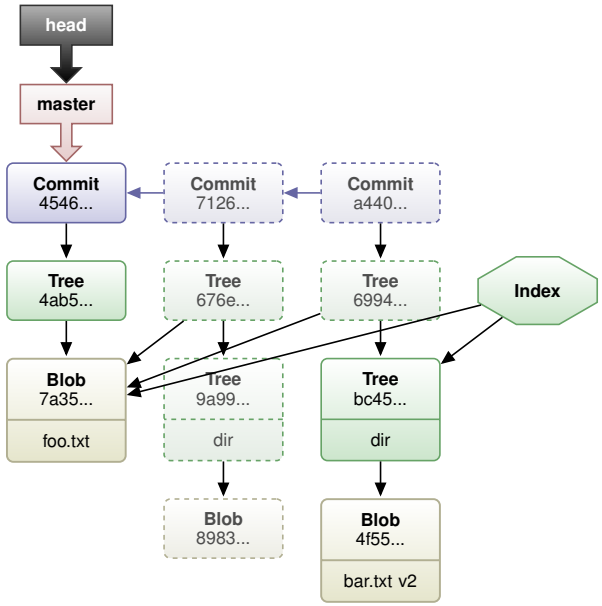
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```

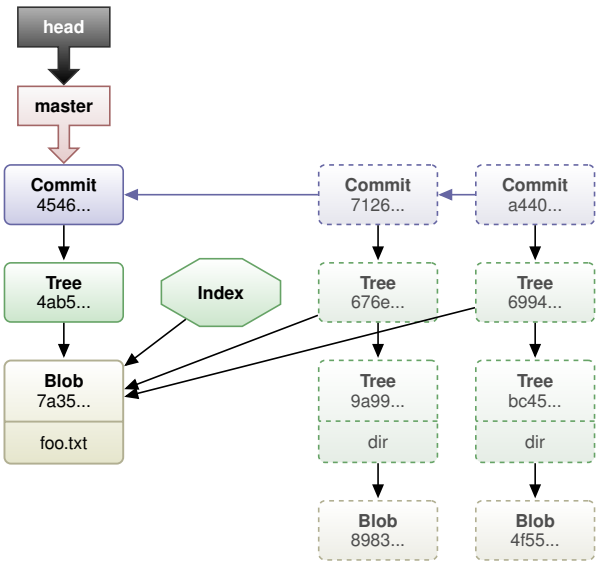



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
ls -R
  .: foo.txt
  dir: bar.txt
```

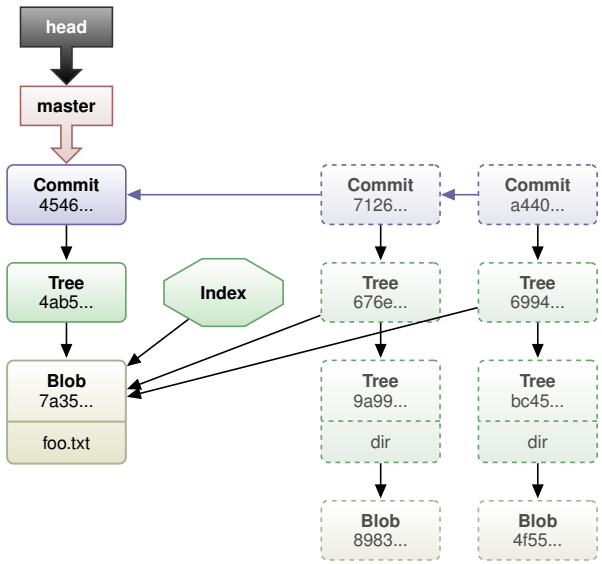



```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --soft 4546
```

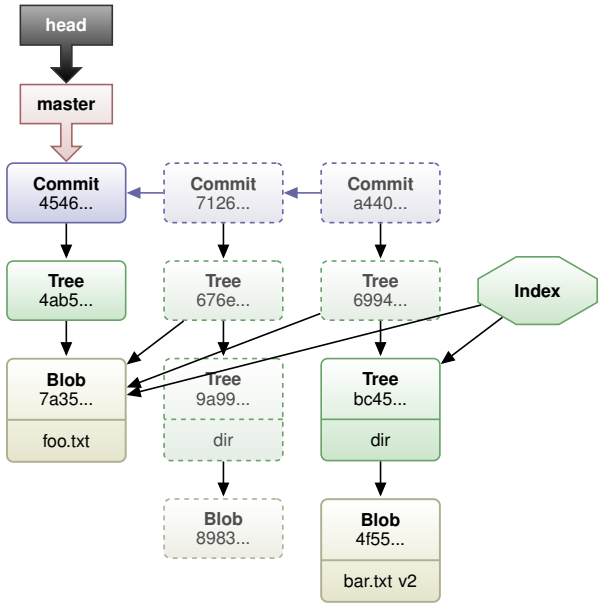
Les différents type de reset.




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --hard 4546
ls -R
  .: foo.txt
```




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset 4546
ls -R
  .: foo.txt
  dir: bar.txt
```




```
ls -R
  .: foo.txt
  dir: bar.txt
git reset --soft 4546
ls -R
  .: foo.txt
  dir: bar.txt
```

Outline

Comparaison : git diff

- Différences entre le répertoire de travail et l'index :

```
$ git diff
```

- Différences entre HEAD et l'index :

```
$ git diff --staged
```

- Différences entre répertoire de travail et HEAD :

```
$ git diff HEAD
```

- Différences entre répertoire de travail et un autre commit :

```
$ git diff <commit_1>
```

- Différences entre deux commit :

```
$ git diff <commit_1> <commit_2>
```

Information sur un commit : git show

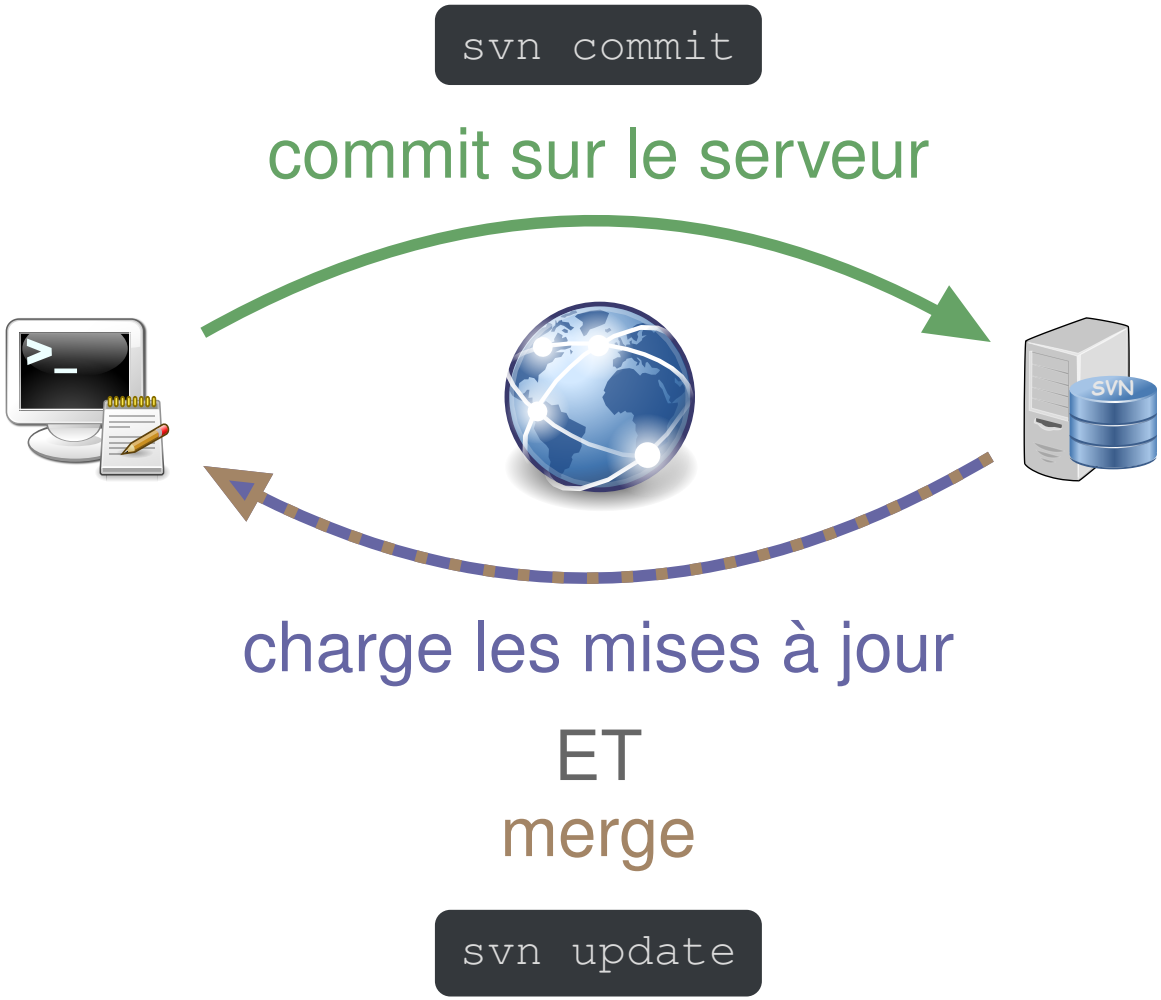
```
$ git show
commit 7a618a3f3c23262ad281c9afe69e145ef867d43b
Author: Julien SOPENA <julien.sopena@lip6.fr>
Date: Mon Oct 25 03:55:27 2010 +0200

Ceci est un petit exemple de commit.

diff -git a/test b/test
index 808a2c4..99810fa 100644
-- a/test
+++ b/test
@@ -1,3 +1,3 @@
Ligne de texte non modifié par ce commit.
-Ligne de texte supprimée.
+Nouvelle ligne de texte
Suite du texte non modifié.
```

Outline

Outline

Dépôts centralisés

CVS ou Subversion



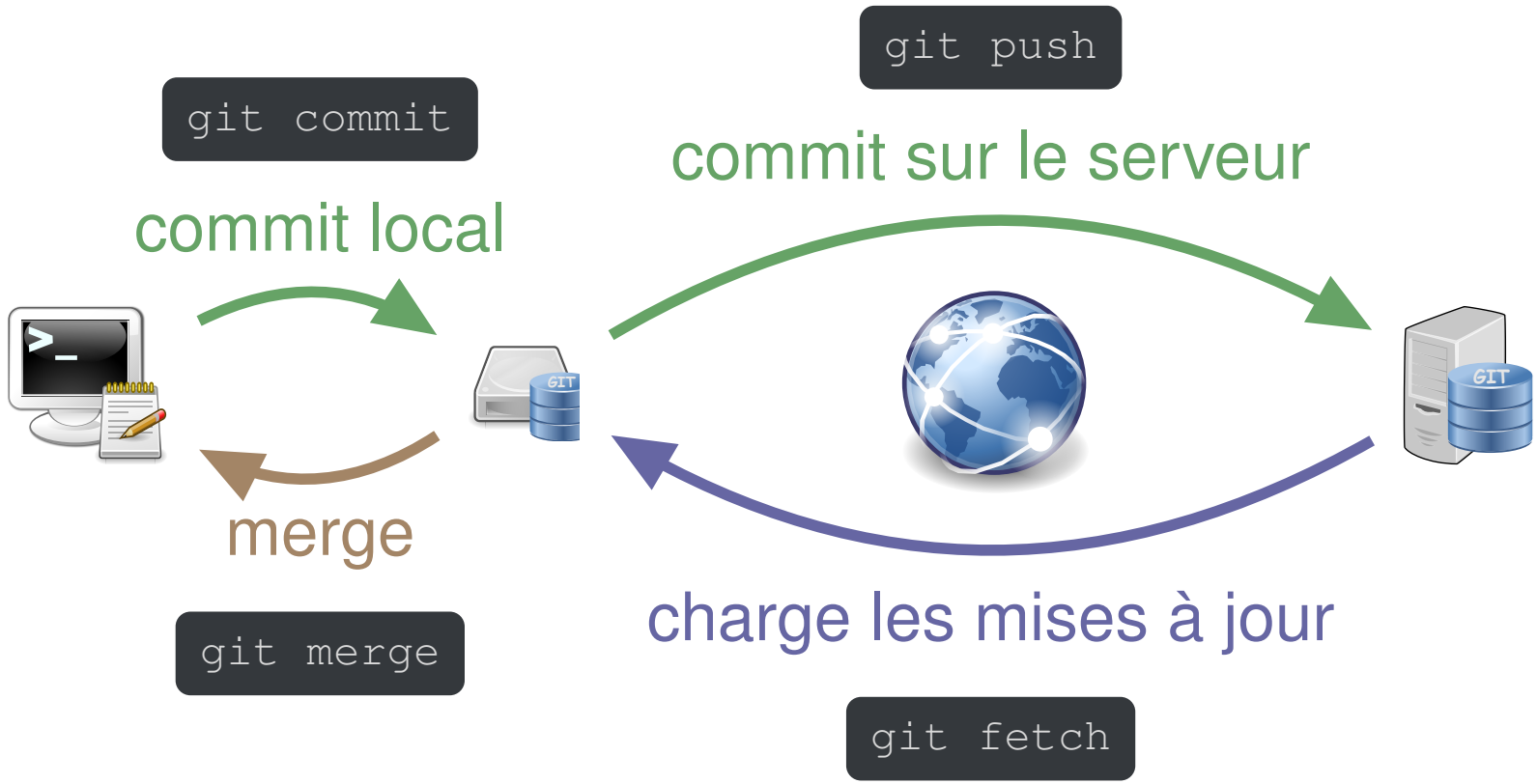
Dépôts centralisés

CVS ou Subversion

- Toutes les versions des fichiers sont entreposées dans un unique dépôt accessible aux clients autorisés
- Les clients ne travaillent que sur une partie des données, en général, une simple branche
- Chaque changement de branche nécessite un téléchargement de l'ensemble des données de la branche.
- Impossibilité de versionner *Off-line*
- Inclure un contributeur nécessite d'ouvrir le dépôt en écriture.

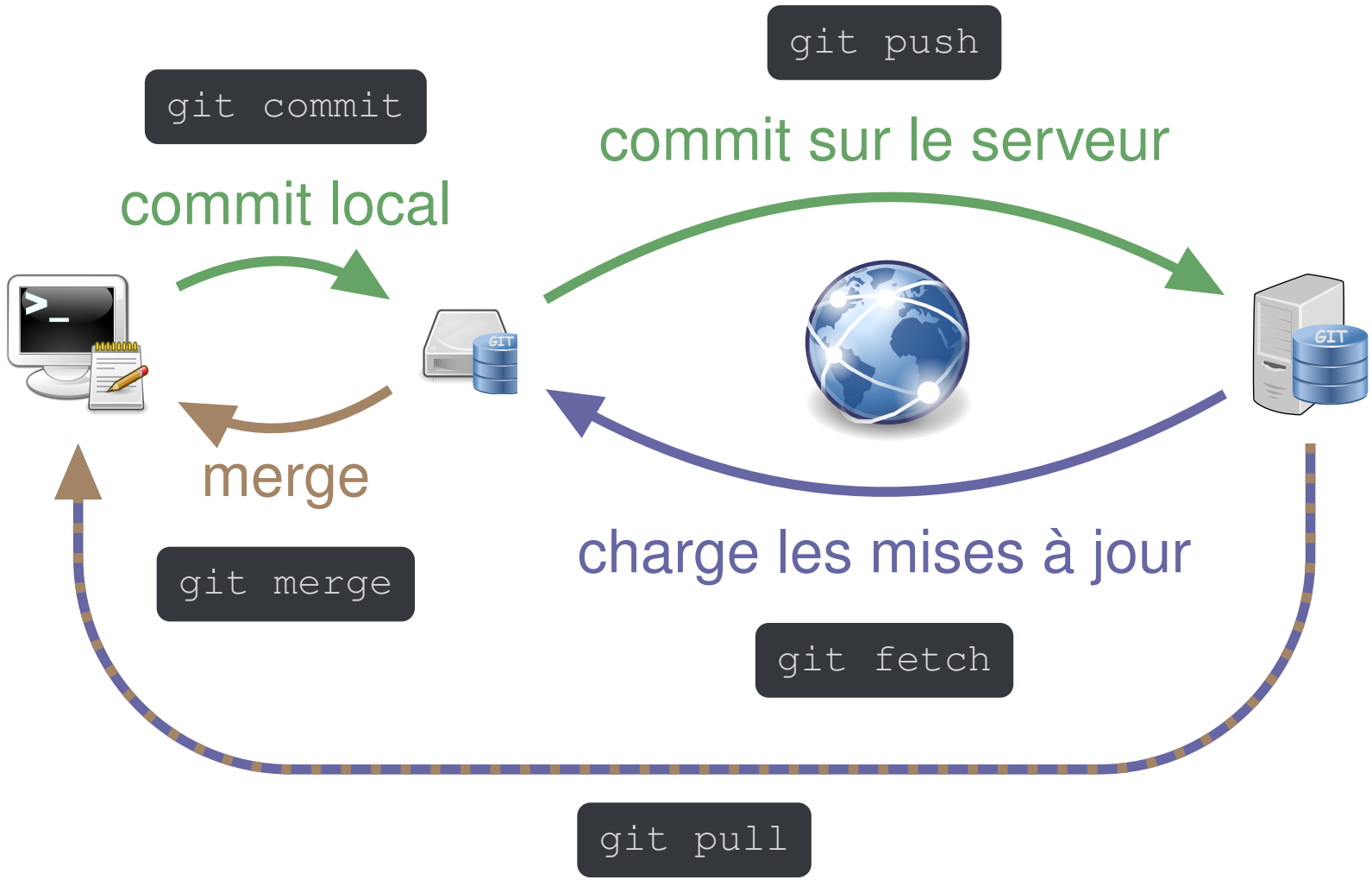
Dépôts décentralisés

Arch, Bazar-Ng, Git, monotone etc.



Dépôts décentralisés

Arch, Bazar-Ng, Git, monotone etc.



Dépôts décentralisés

Arch, Bazar-Ng, Git, monotone etc.

- Chaque client a l'ensemble des fichiers dans son dépôt local :
  - On peut travailler *Off-Line* (à la plage, à la montagne,...)
  - Le changement de branche est rapide et est donc, une des méthodes de développement : utiliser les branches
- Les seules actions du client nécessitant un accès au dépôt distant sont :
  - la mise à jour du dépôt local depuis l'extérieur
  - l'envoi d'information.
- Le client peut versionner en local !

Conflits

Définition

*On appelle **conflits**, toute modification d'un fichier dans un dépôt qui n'a pas été élaborée à partir de la version actuelle du fichier sur ce dépôt, i.e.,lorsqu'il y a une modification de ce fichier sur le dépôt pendant son édition.*

Apparition sur CVS/Subversion :

- au téléchargement des modifications : cvs update ;
- au commit avec un refus d'enregistrer sur le dépôt.

Apparition sur Git et les autres :

- Uniquement lors des fusions de branches :
  locale/locale, locale/distante ou distante/locale.

Outline

Branches distantes et branches locales

Un projet décentralisé possède deux types de branches :

Définition

*On appelle **branche distante**, une branche qui pointe sur des dépôts distants en lecture et/ou écriture. Ces dépôts distants peuvent être référencés par une ou plusieurs personnes.*

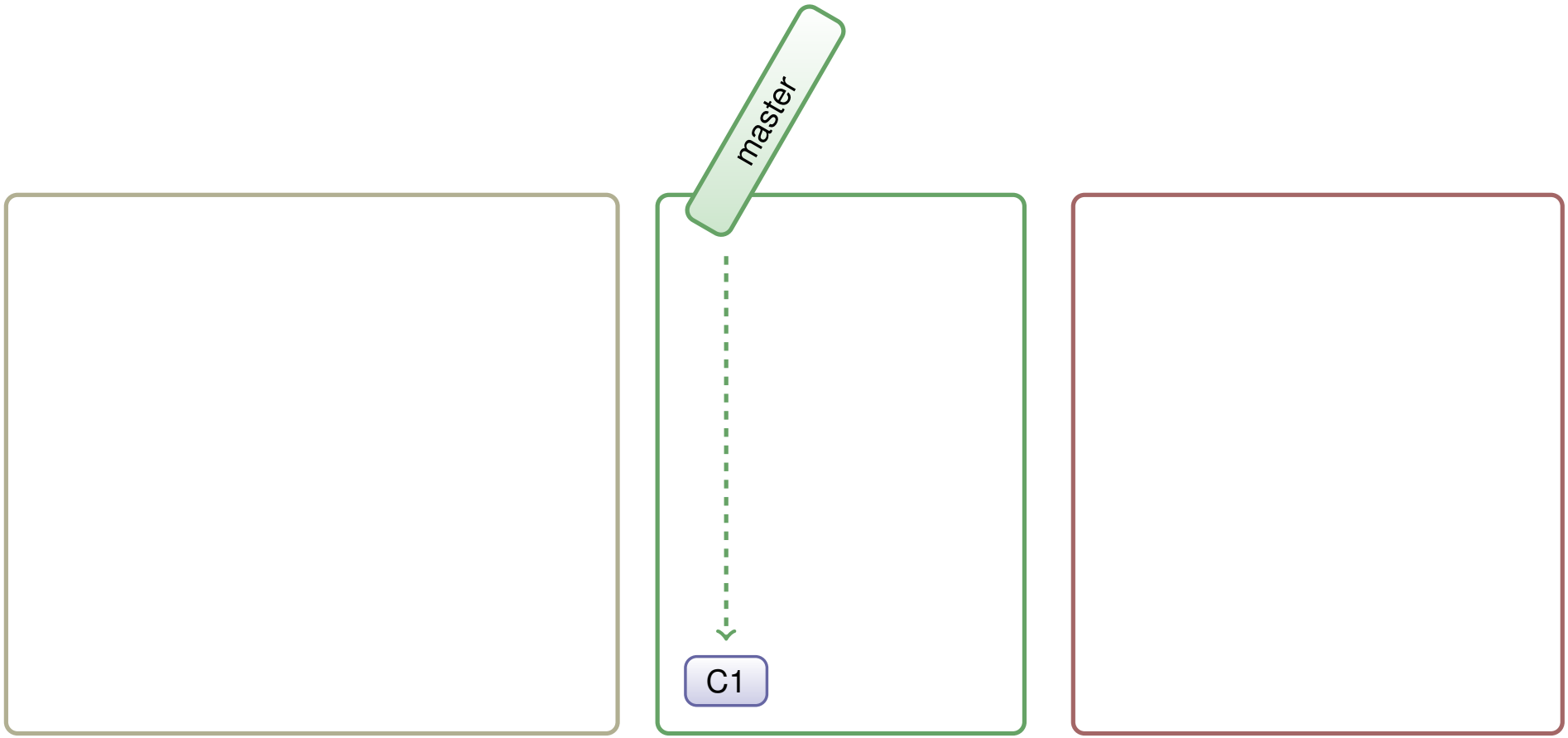
Définition

*On appelle **branche locale**, une branche propre au dépôt local. Pour être envoyées, les données d'une telle branche doivent être fusionnées avec une branche distante.*

Dépôts distants et gestion de la concurrence.

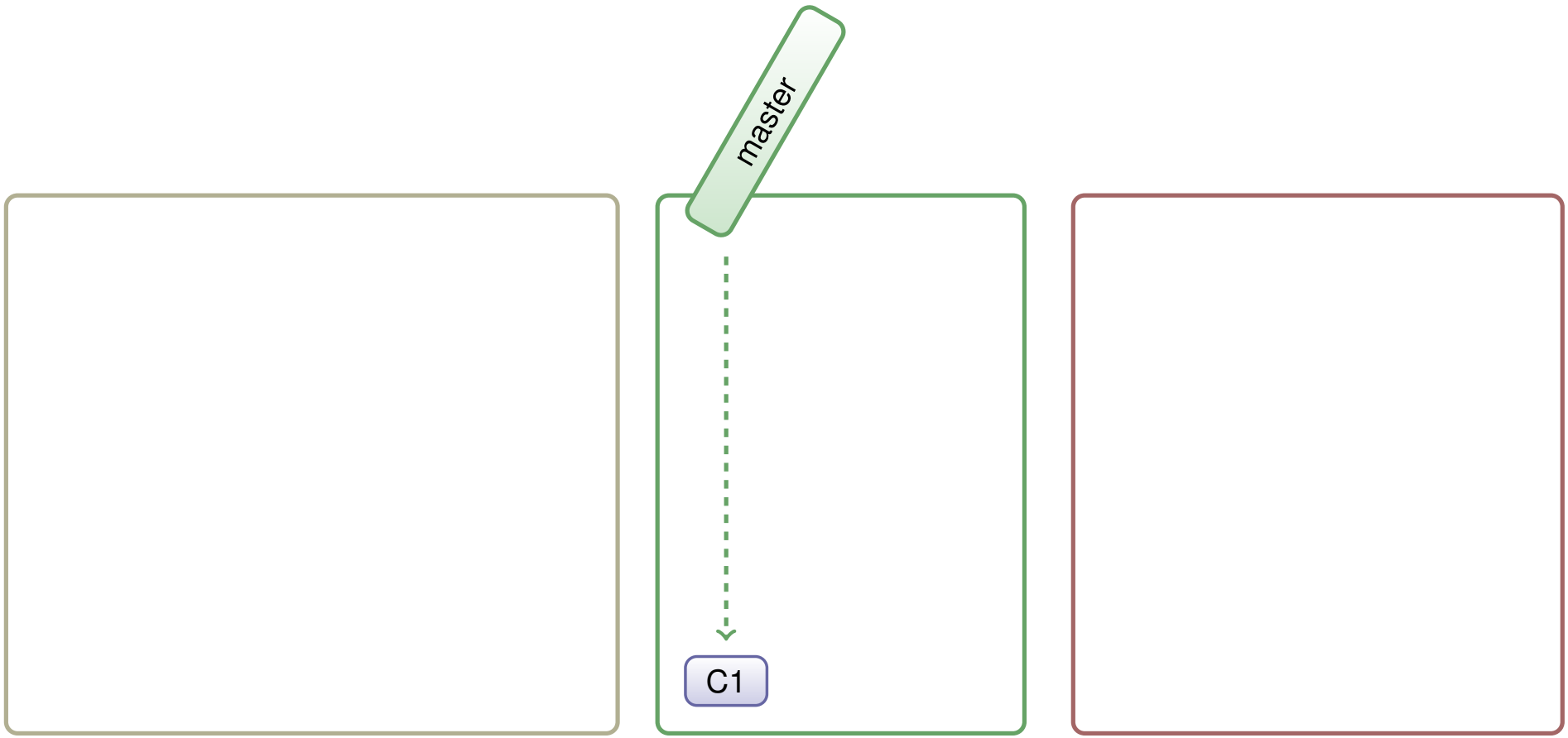
```
git init
git add ...
git commit
```

Dépôts distants et gestion de la concurrence.



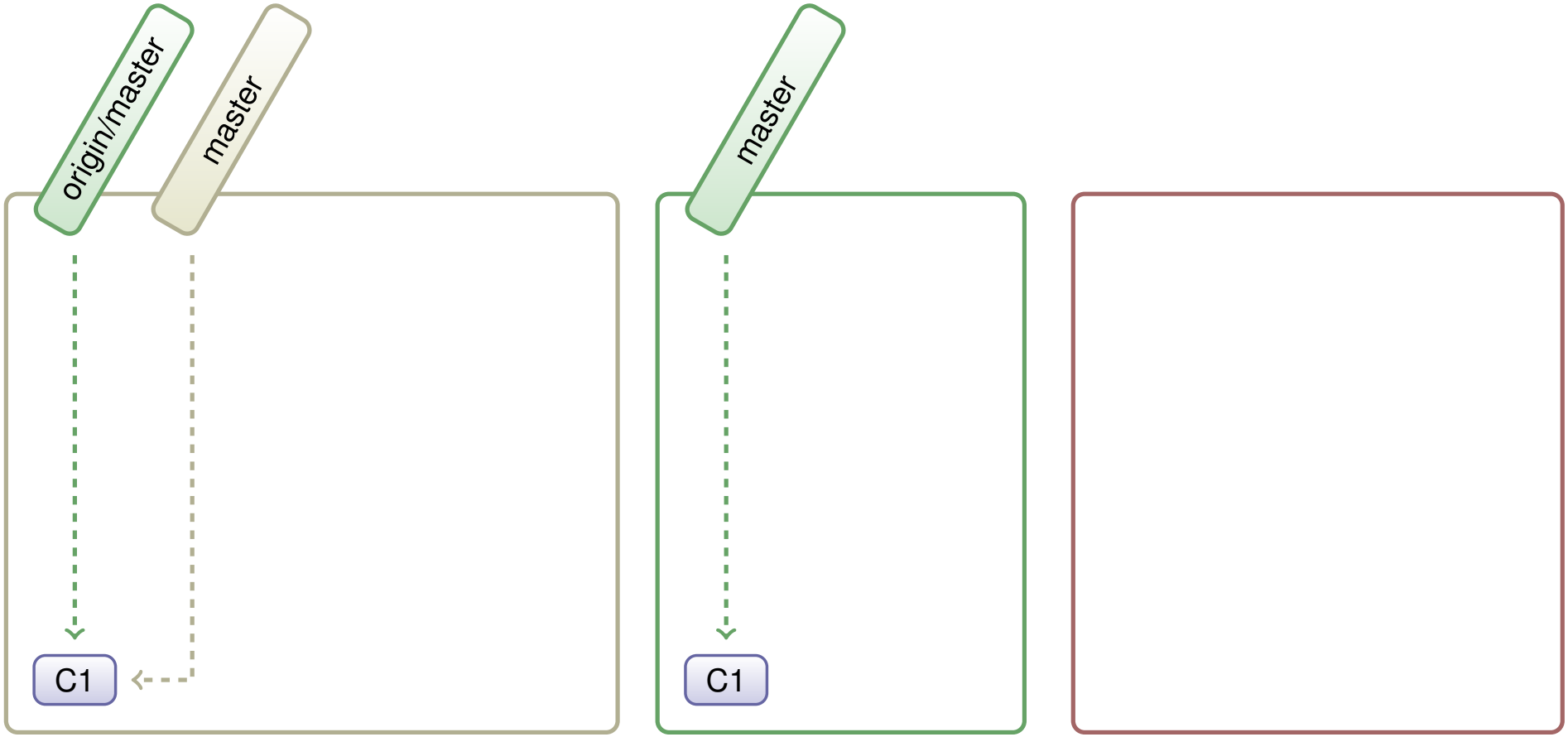

```
git init
git add ...
git commit
```

Dépôts distants et gestion de la concurrence.



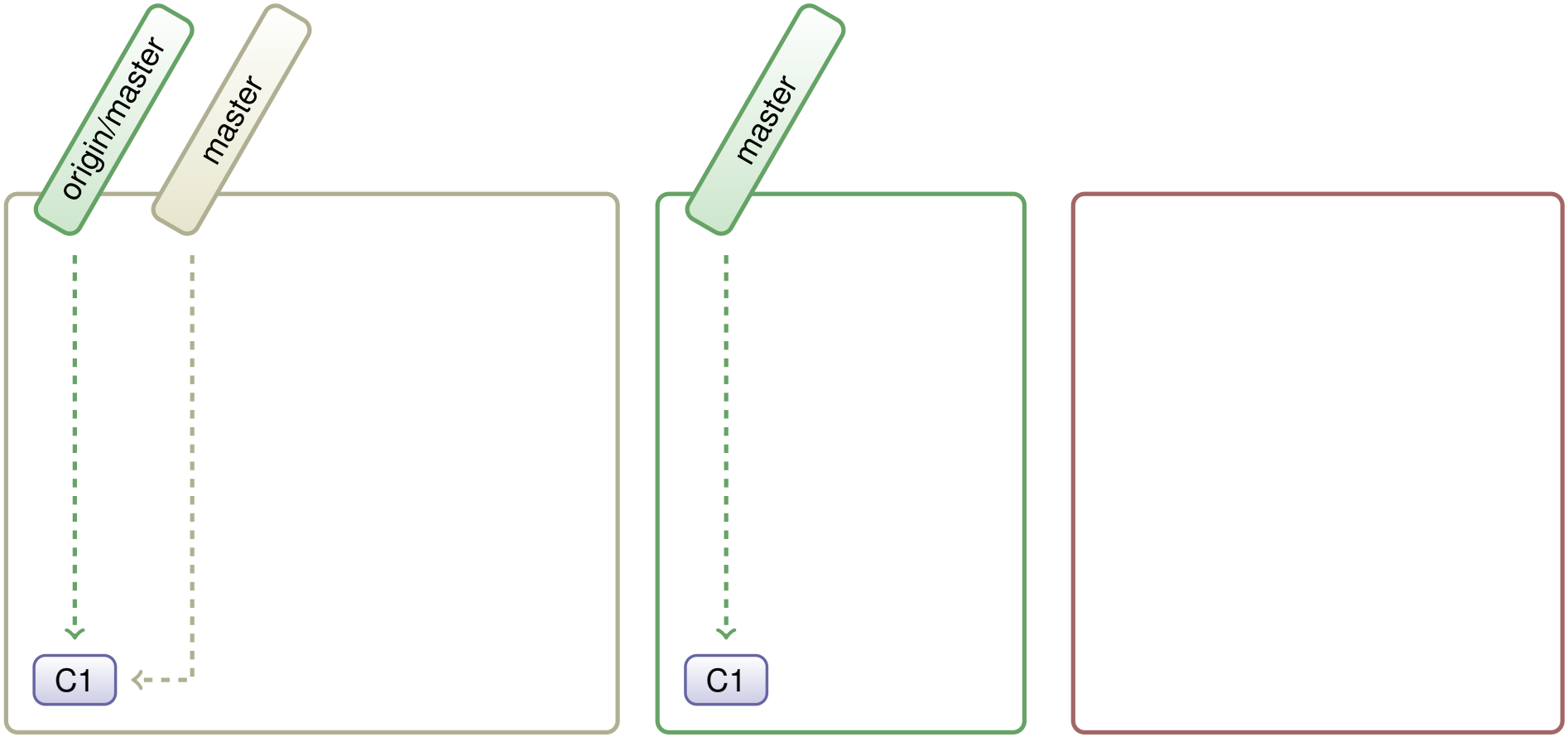

```
git clone <@serveur>
```

Dépôts distants et gestion de la concurrence.



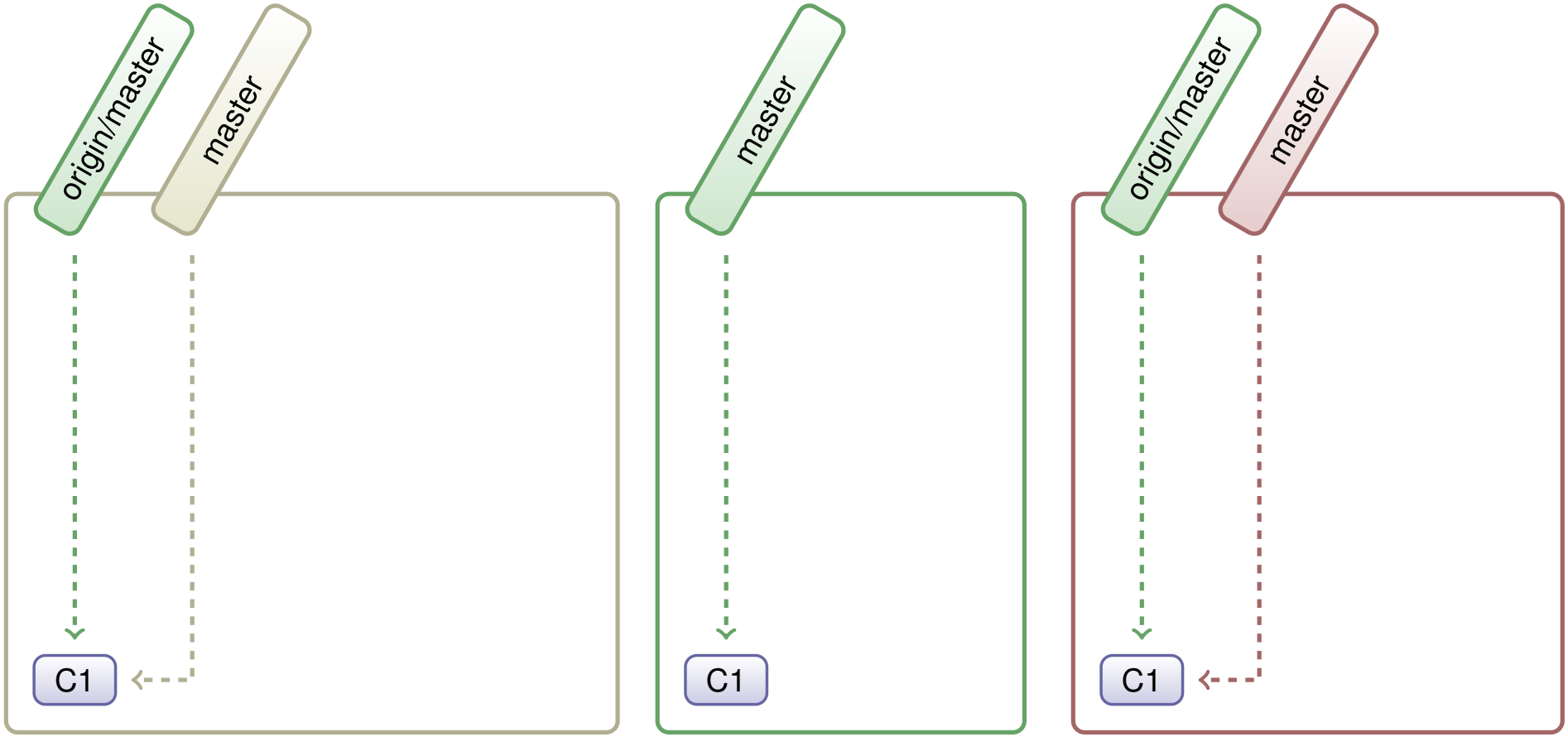

```
git clone <@serveur>
```

Dépôts distants et gestion de la concurrence.



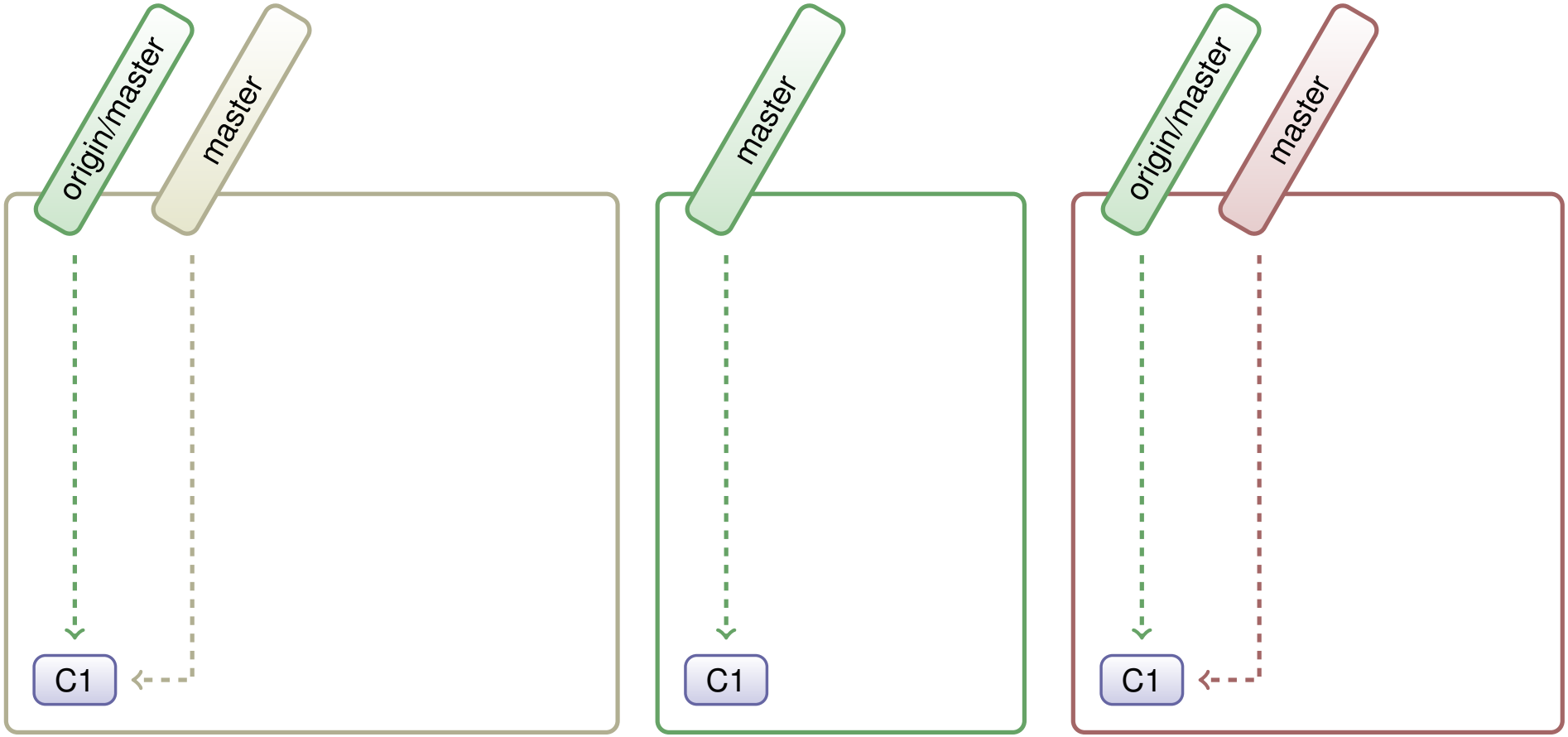

```
git clone <@serveur>
```

Dépôts distants et gestion de la concurrence.



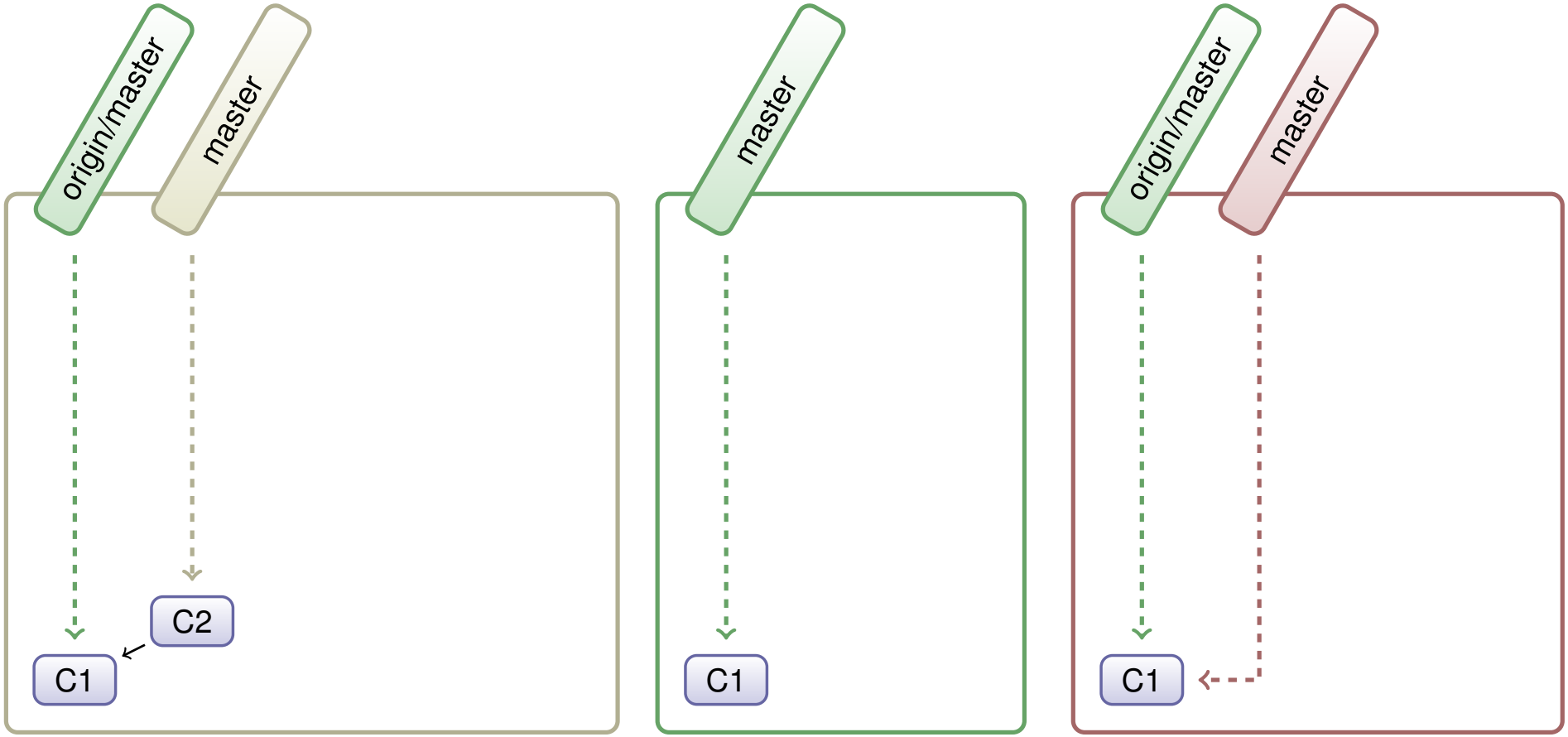

```
git clone <@serveur>
```

Dépôts distants et gestion de la concurrence.



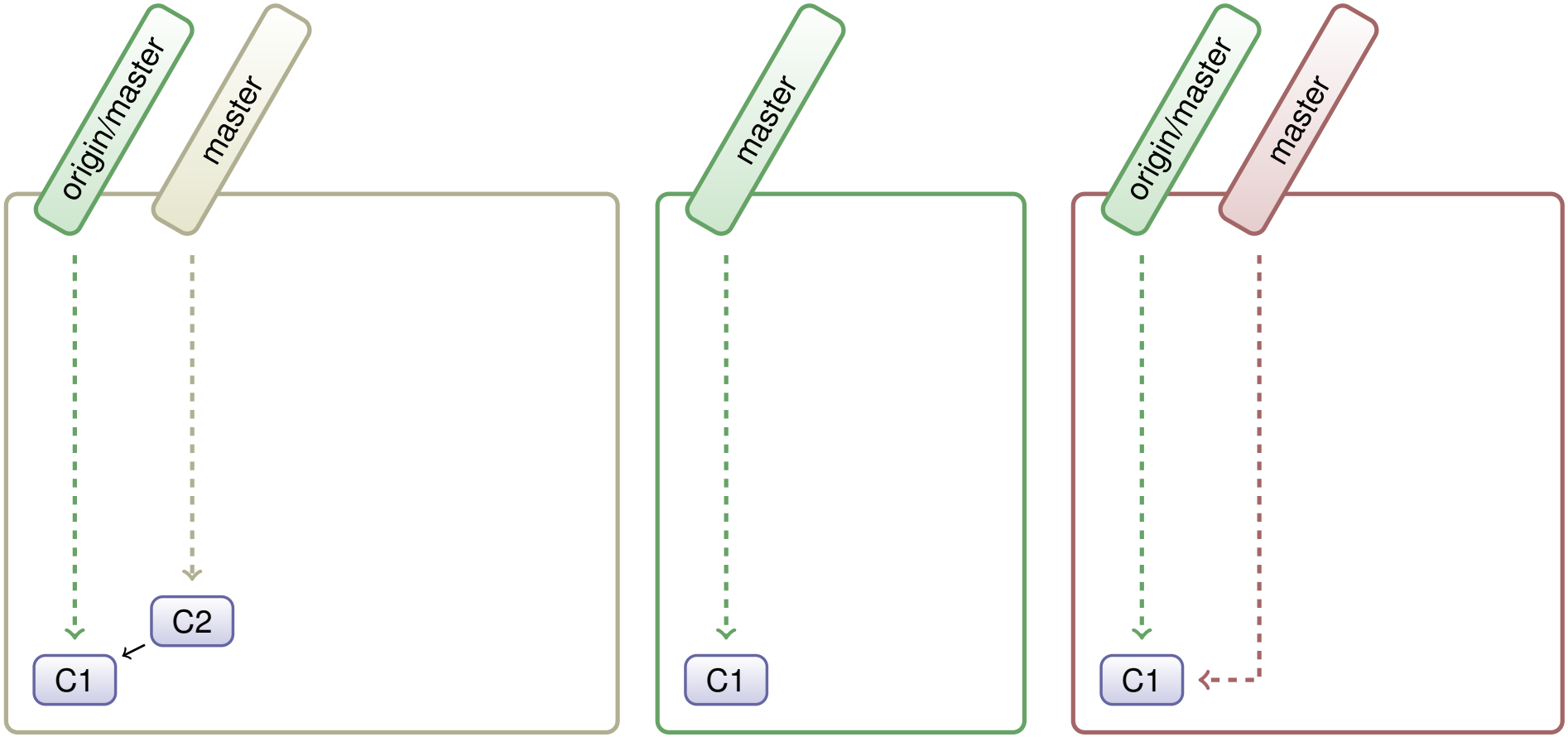

```
git commit
```

Dépôts distants et gestion de la concurrence.



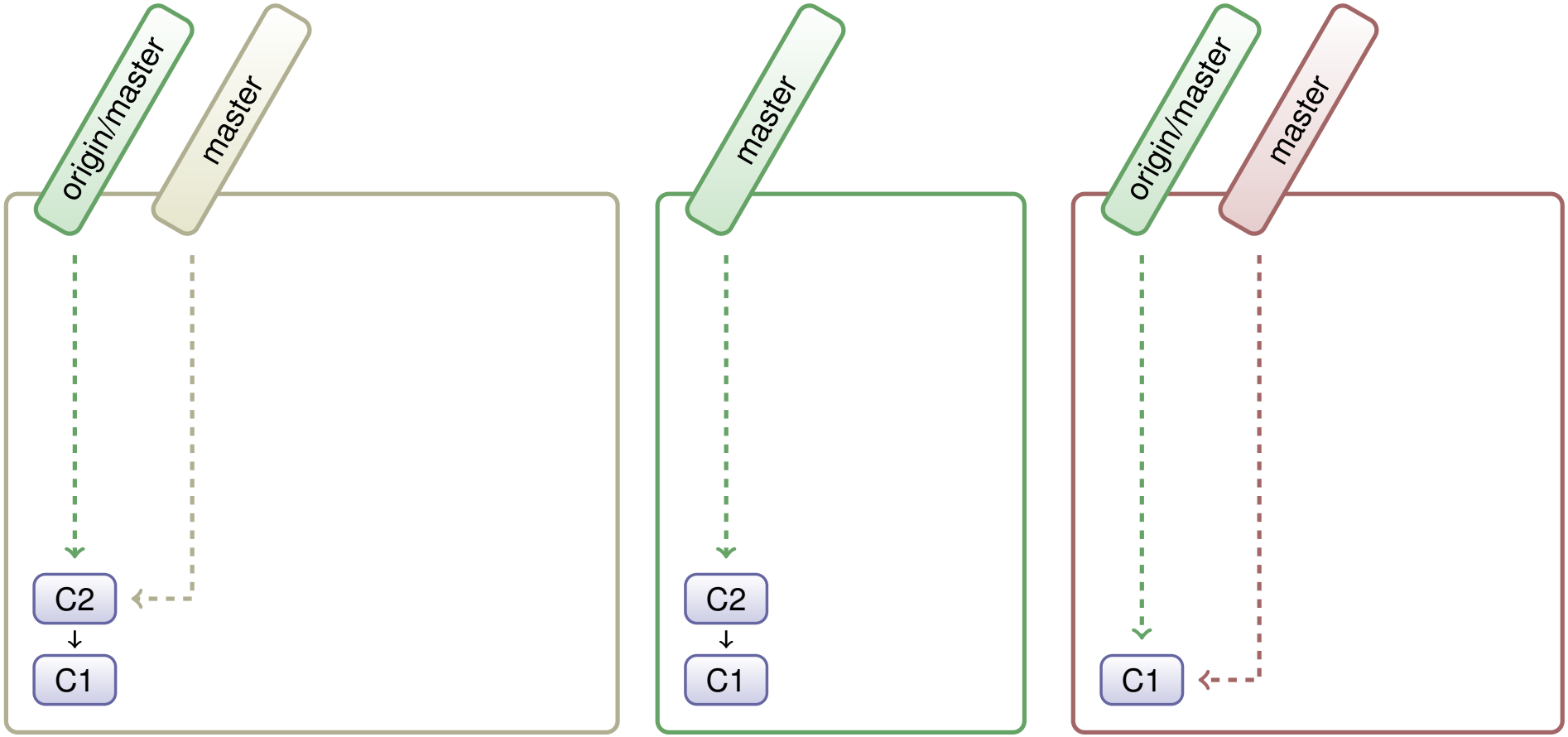

```
git commit
```

Dépôts distants et gestion de la concurrence.



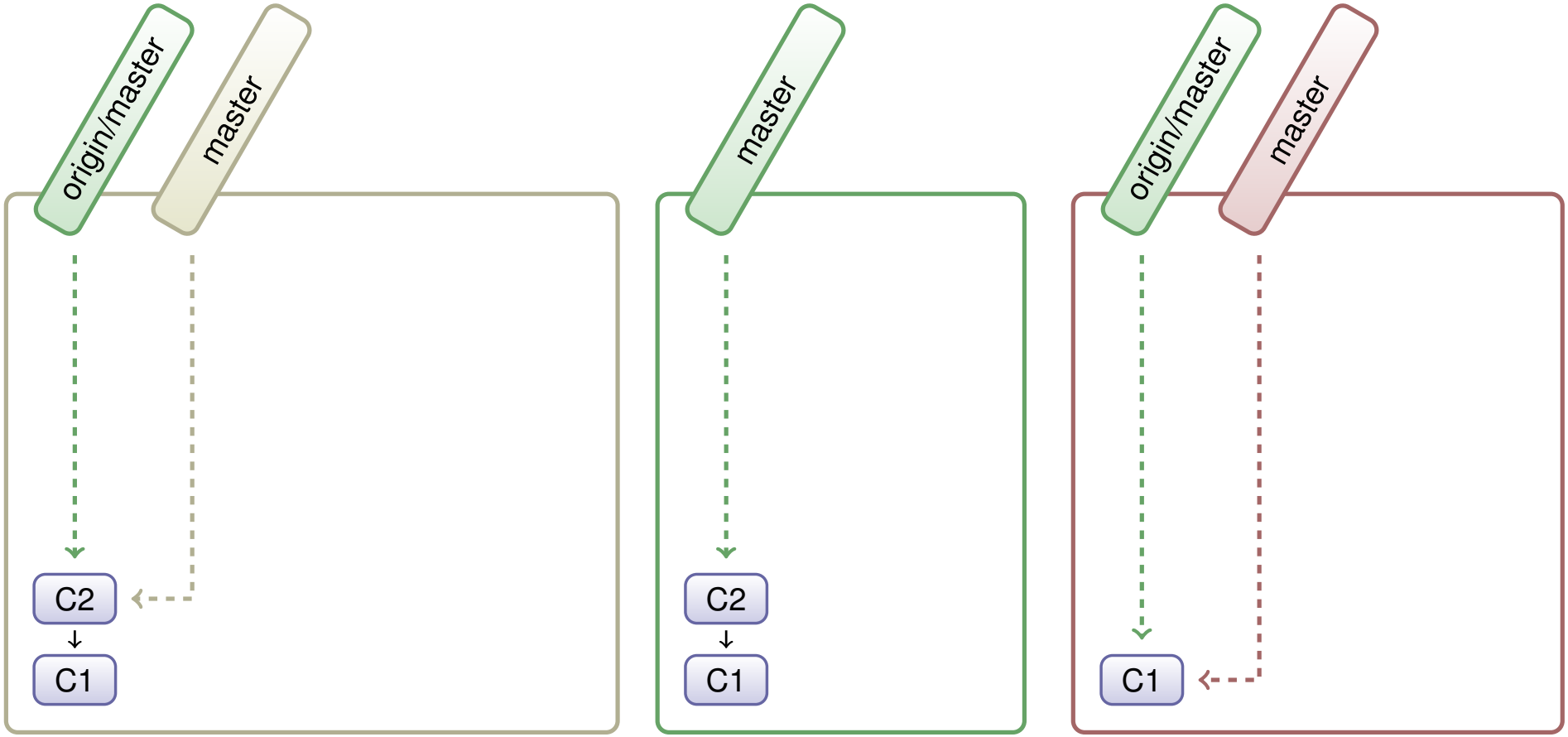

```
git push origin
```

Dépôts distants et gestion de la concurrence.



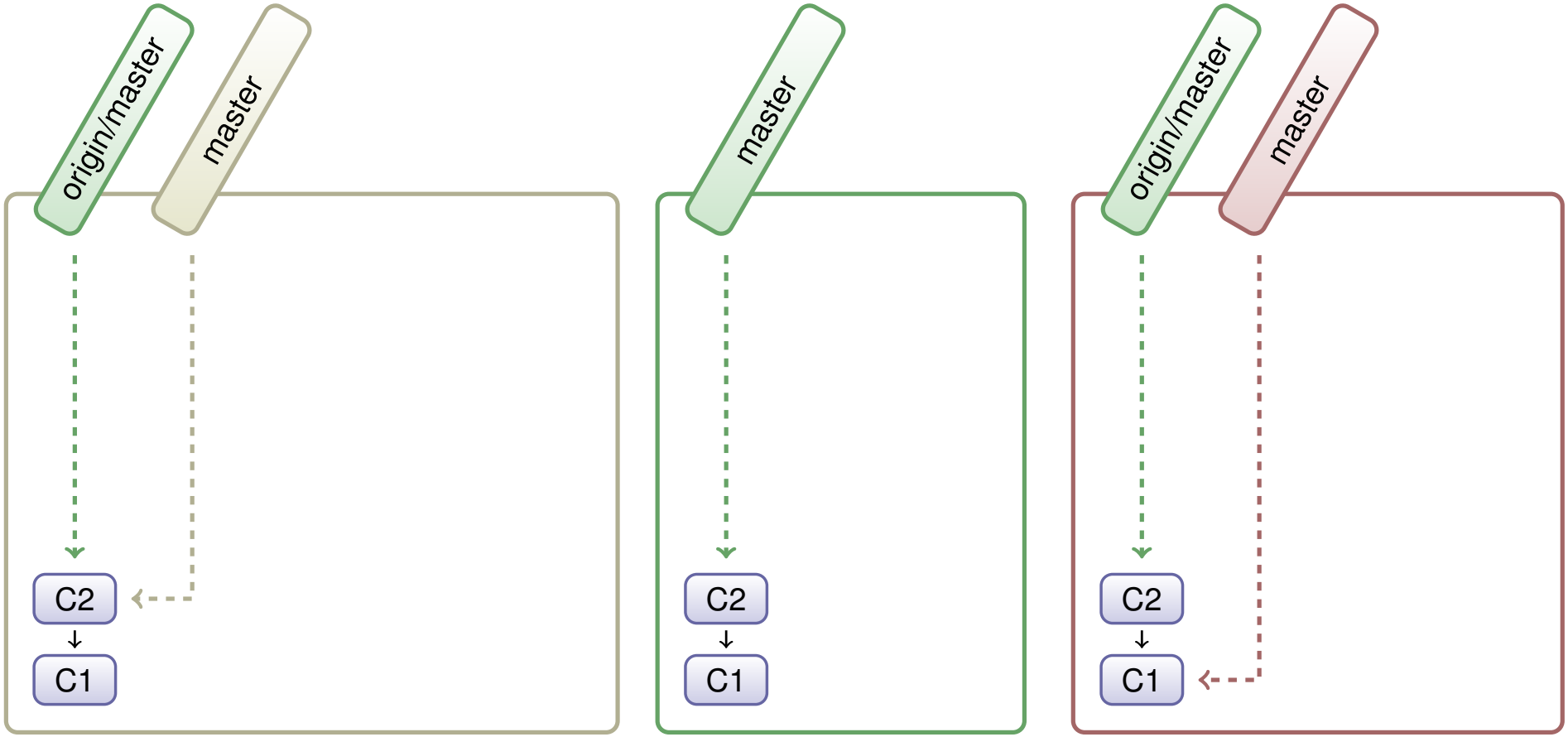

```
git push origin
```

Dépôts distants et gestion de la concurrence.



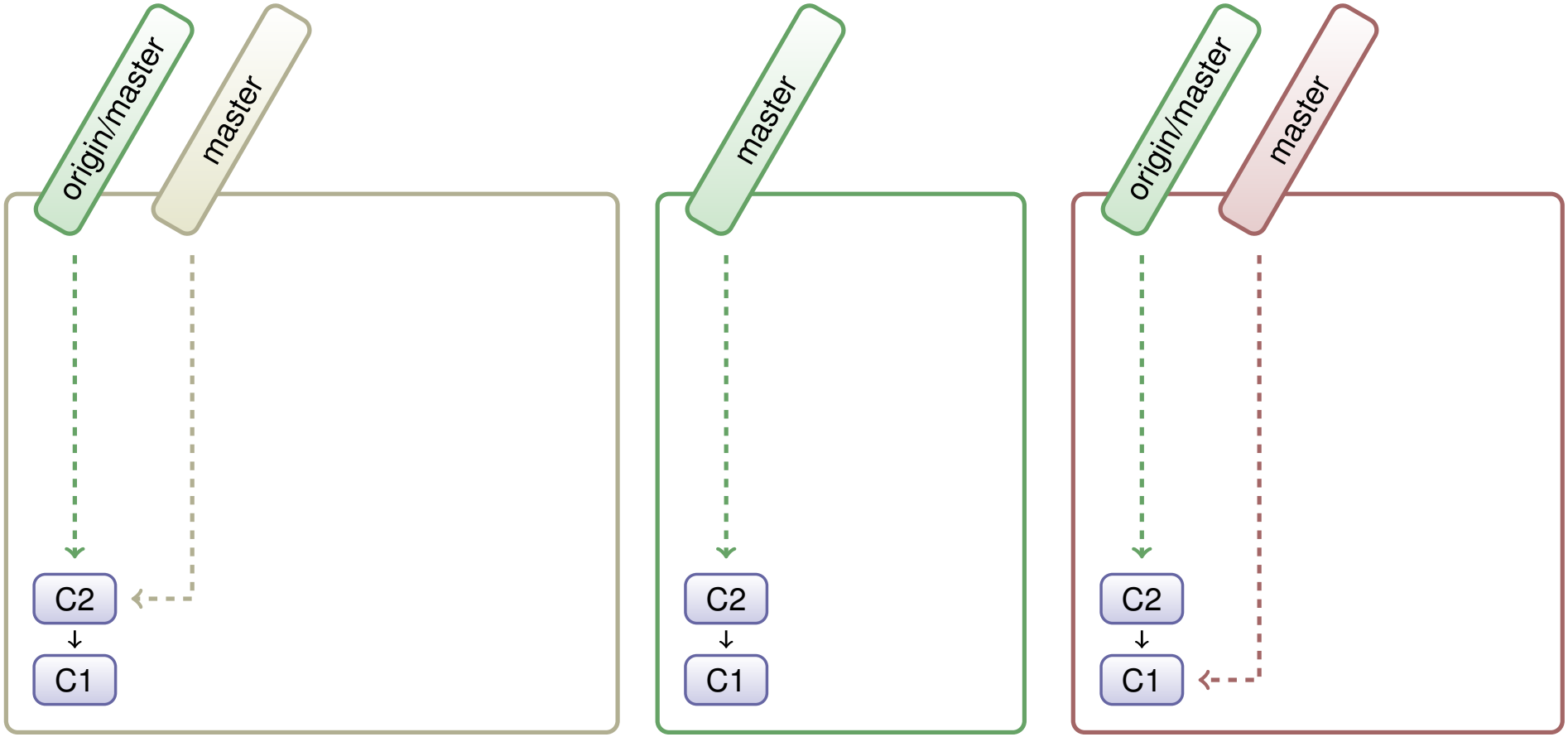

```
git fetch origin
```

Dépôts distants et gestion de la concurrence.



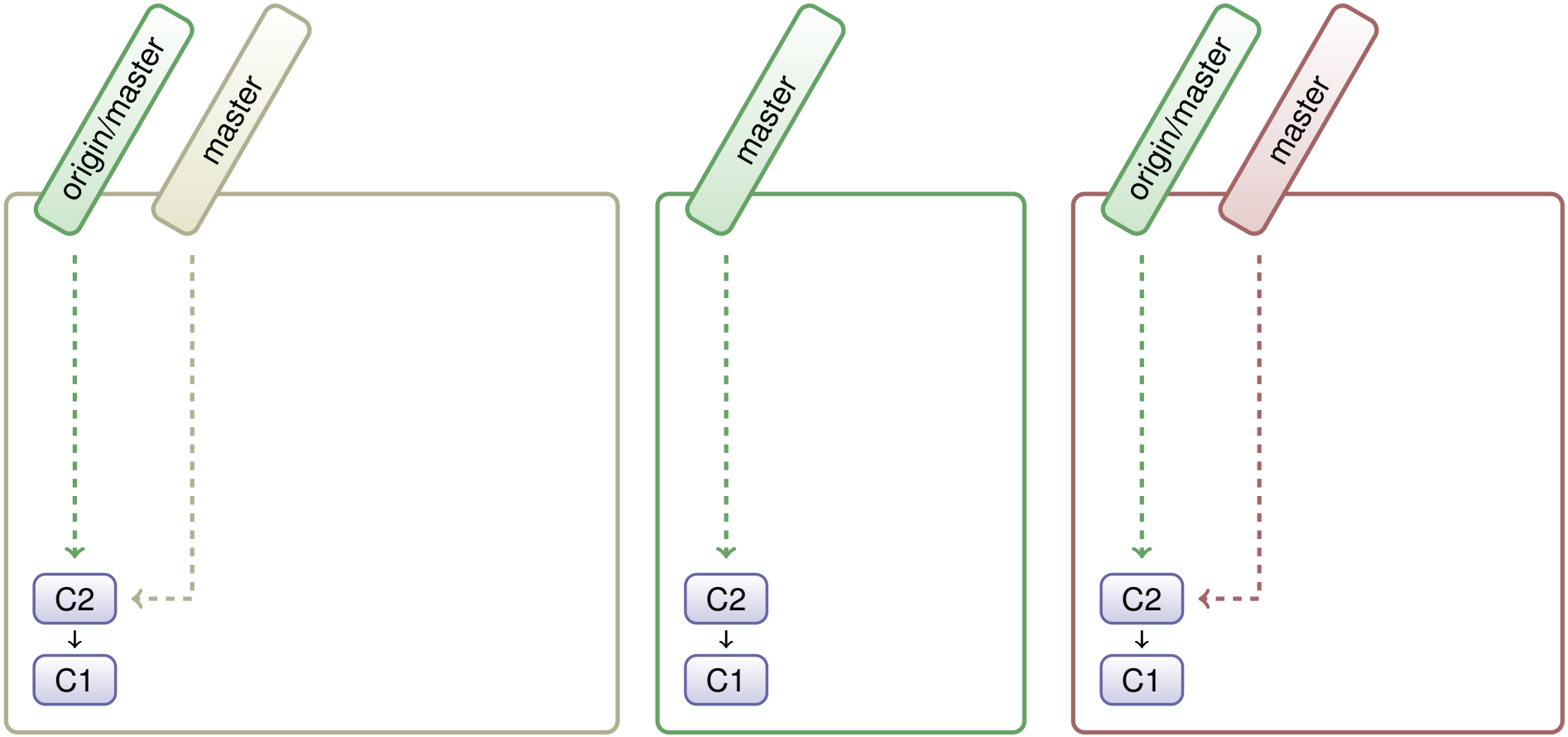

```
git fetch origin
```

Dépôts distants et gestion de la concurrence.



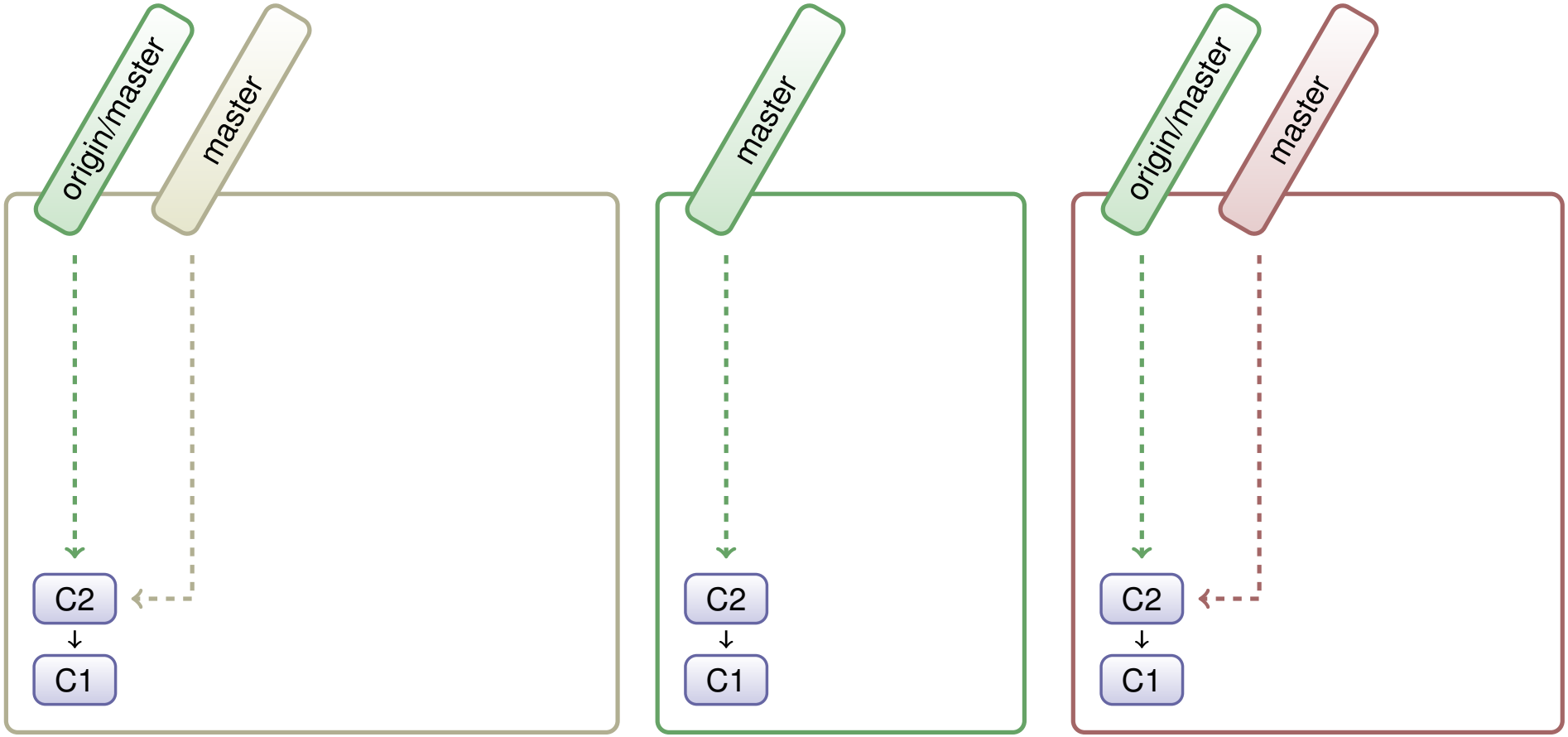

```
git merge
```

Dépôts distants et gestion de la concurrence.



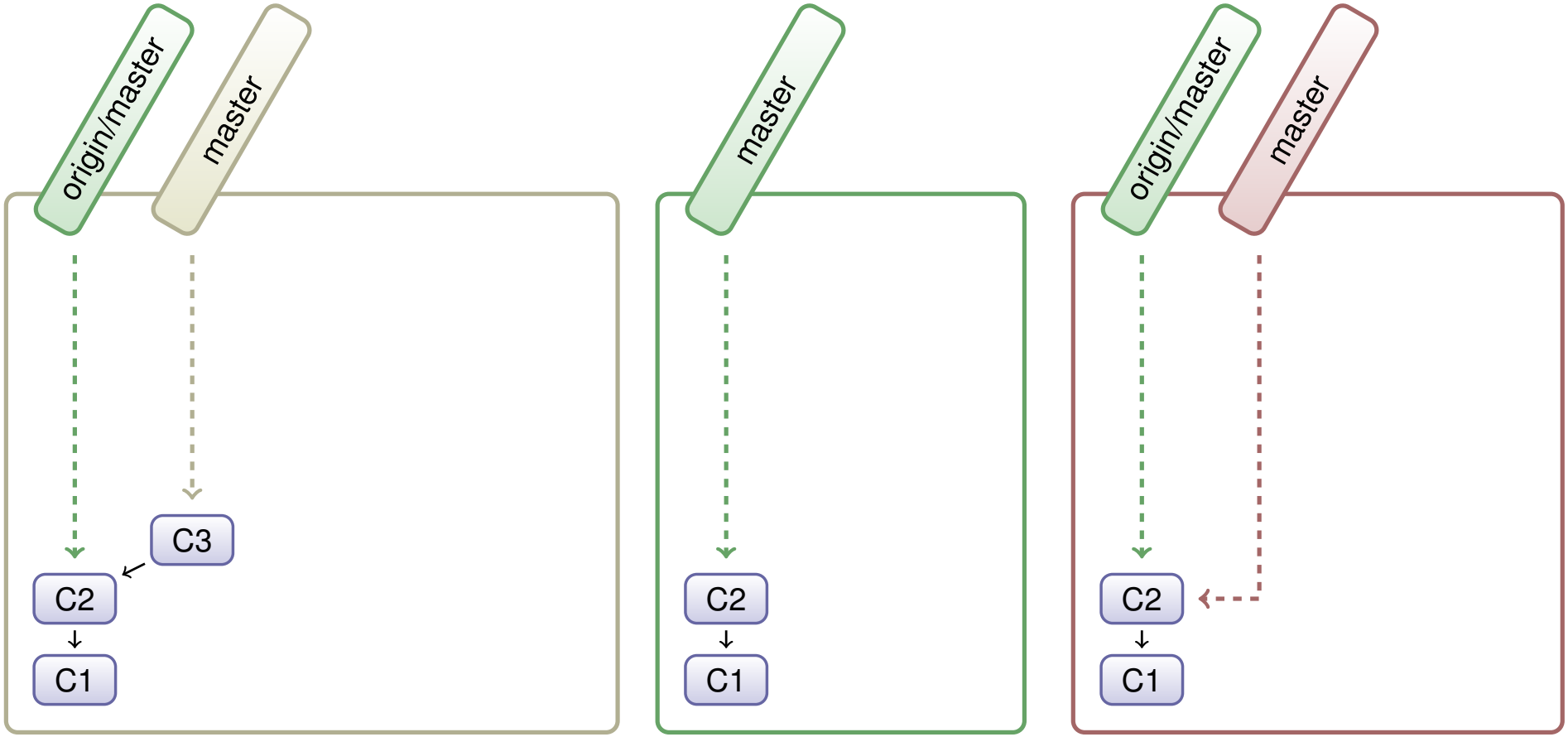

```
git merge
```

Dépôts distants et gestion de la concurrence.



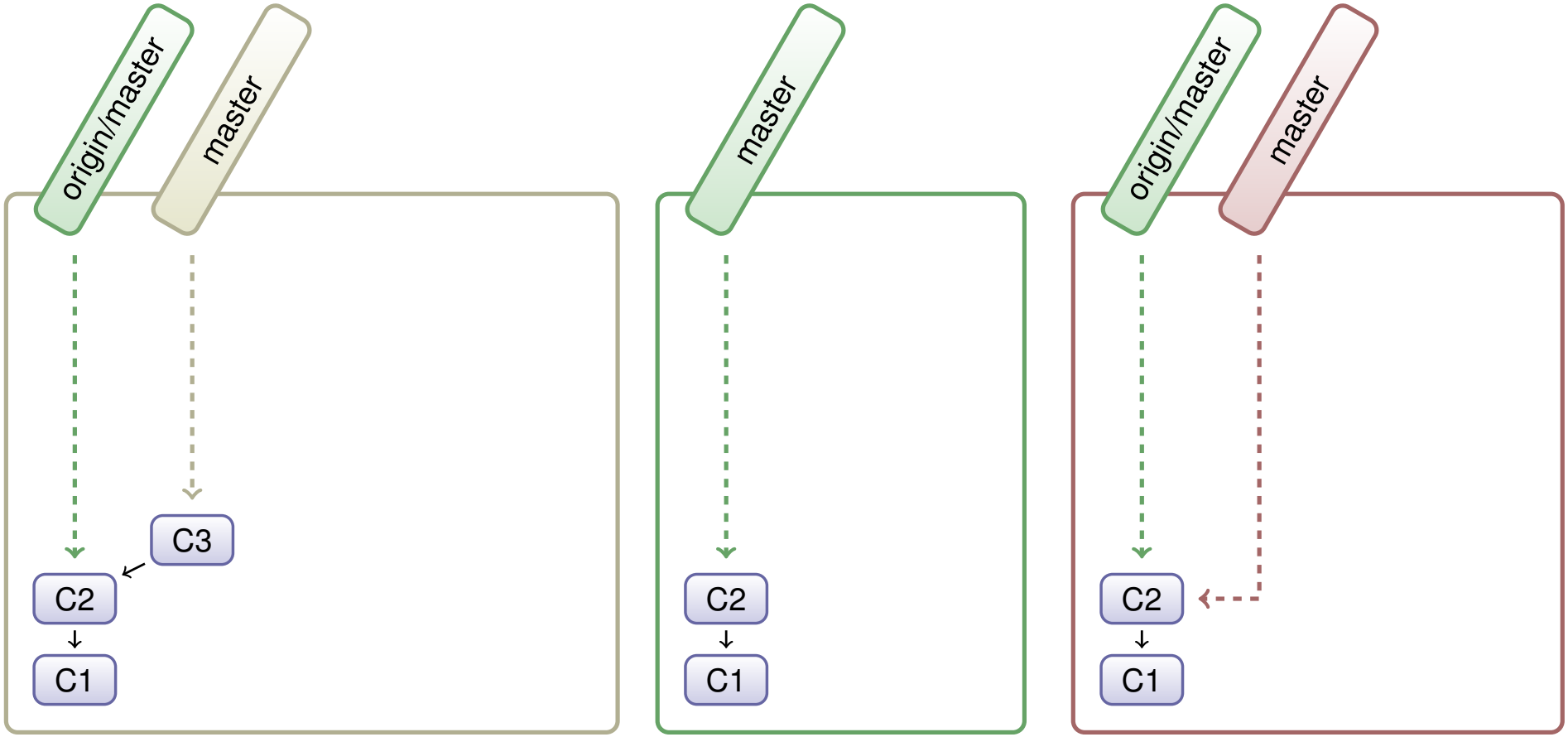

```
git commit
```

Dépôts distants et gestion de la concurrence.



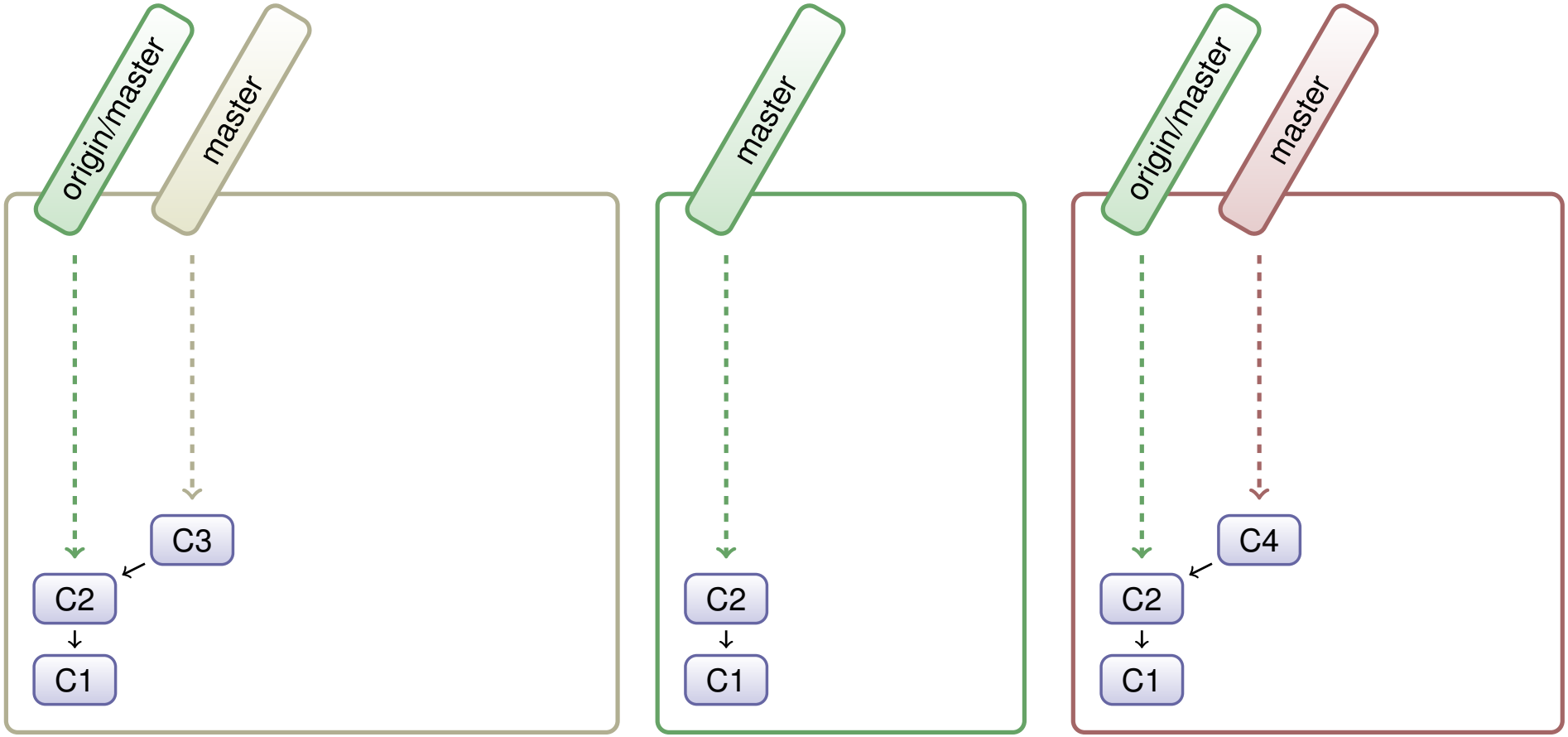

```
git commit
```

Dépôts distants et gestion de la concurrence.



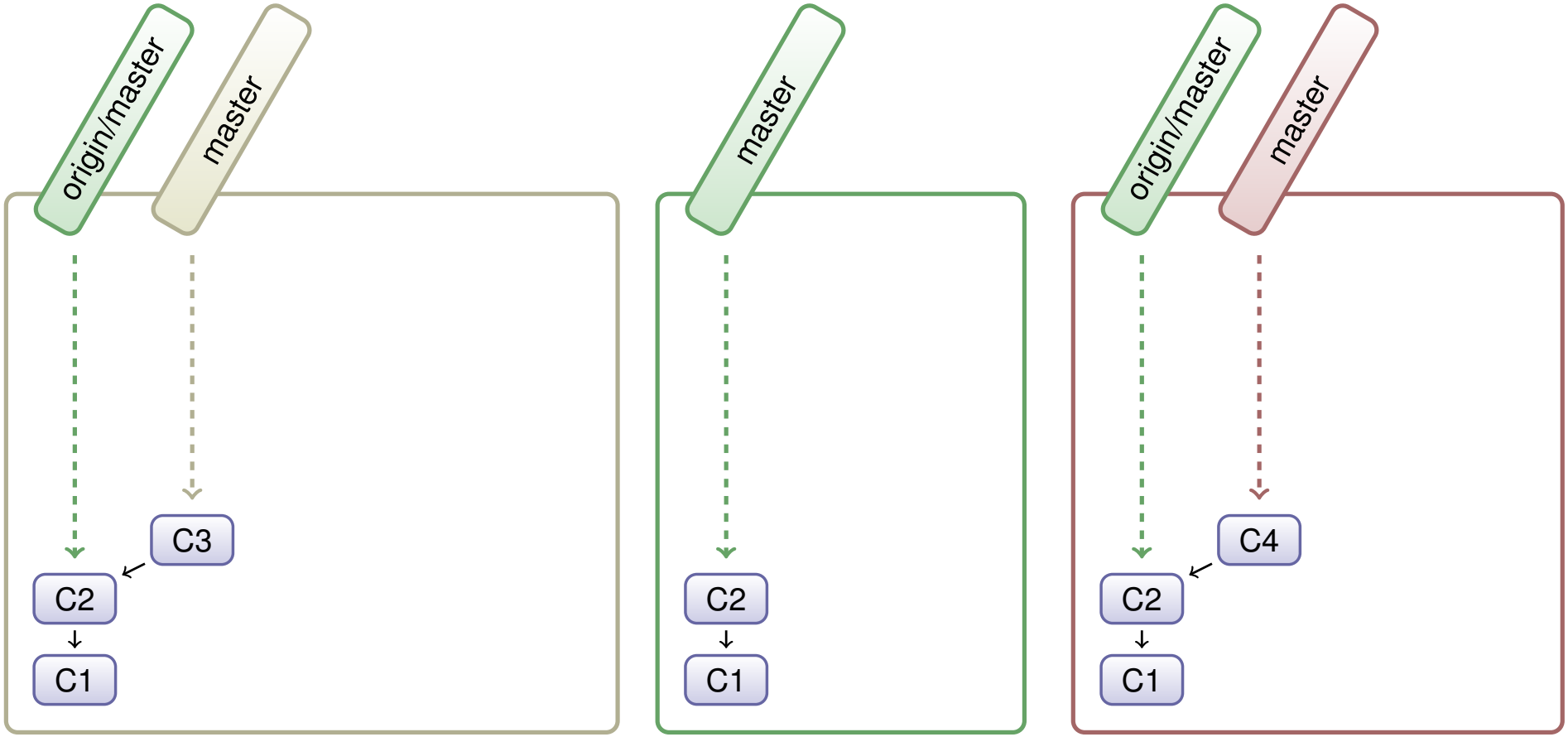

```
git commit
```

Dépôts distants et gestion de la concurrence.



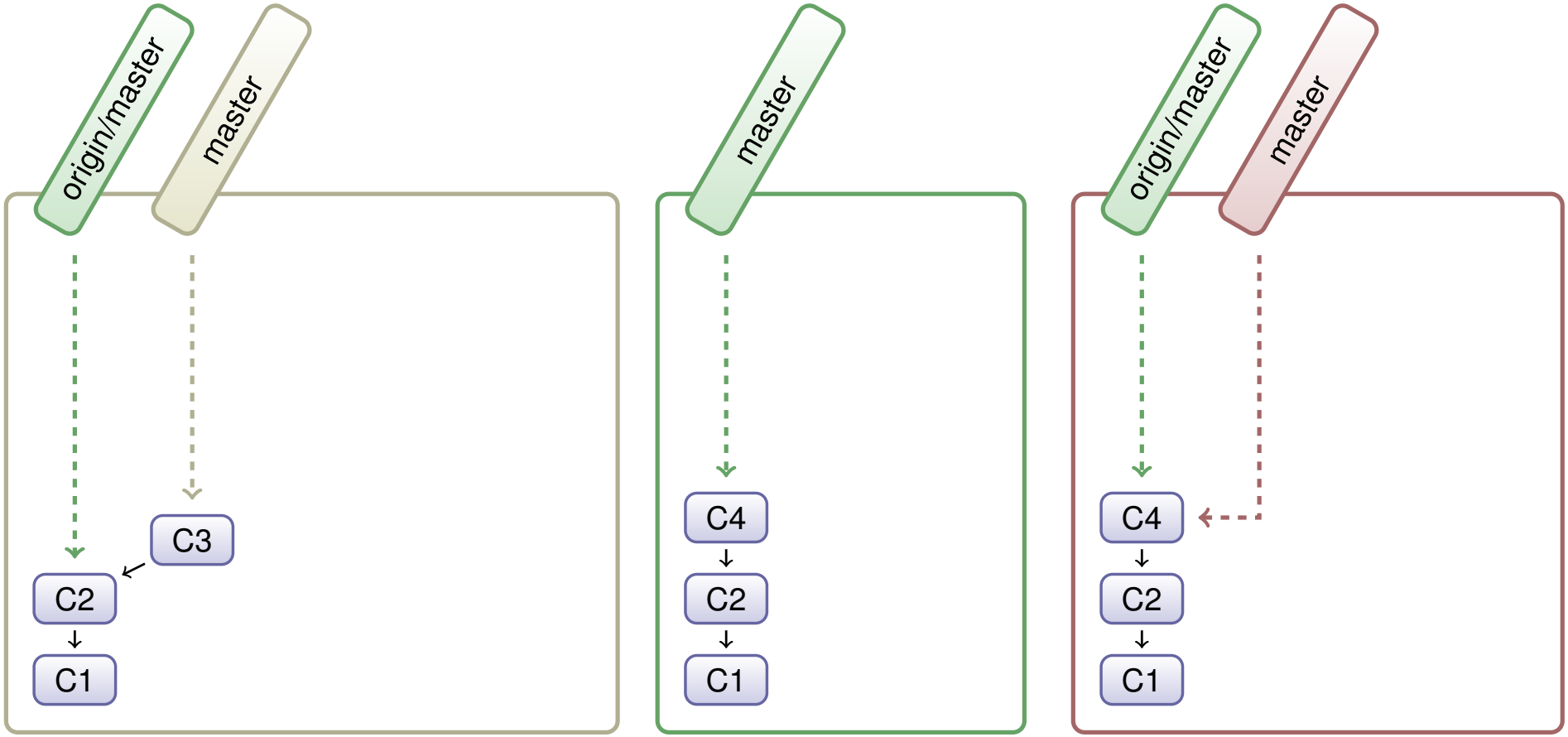

```
git commit
```

Dépôts distants et gestion de la concurrence.



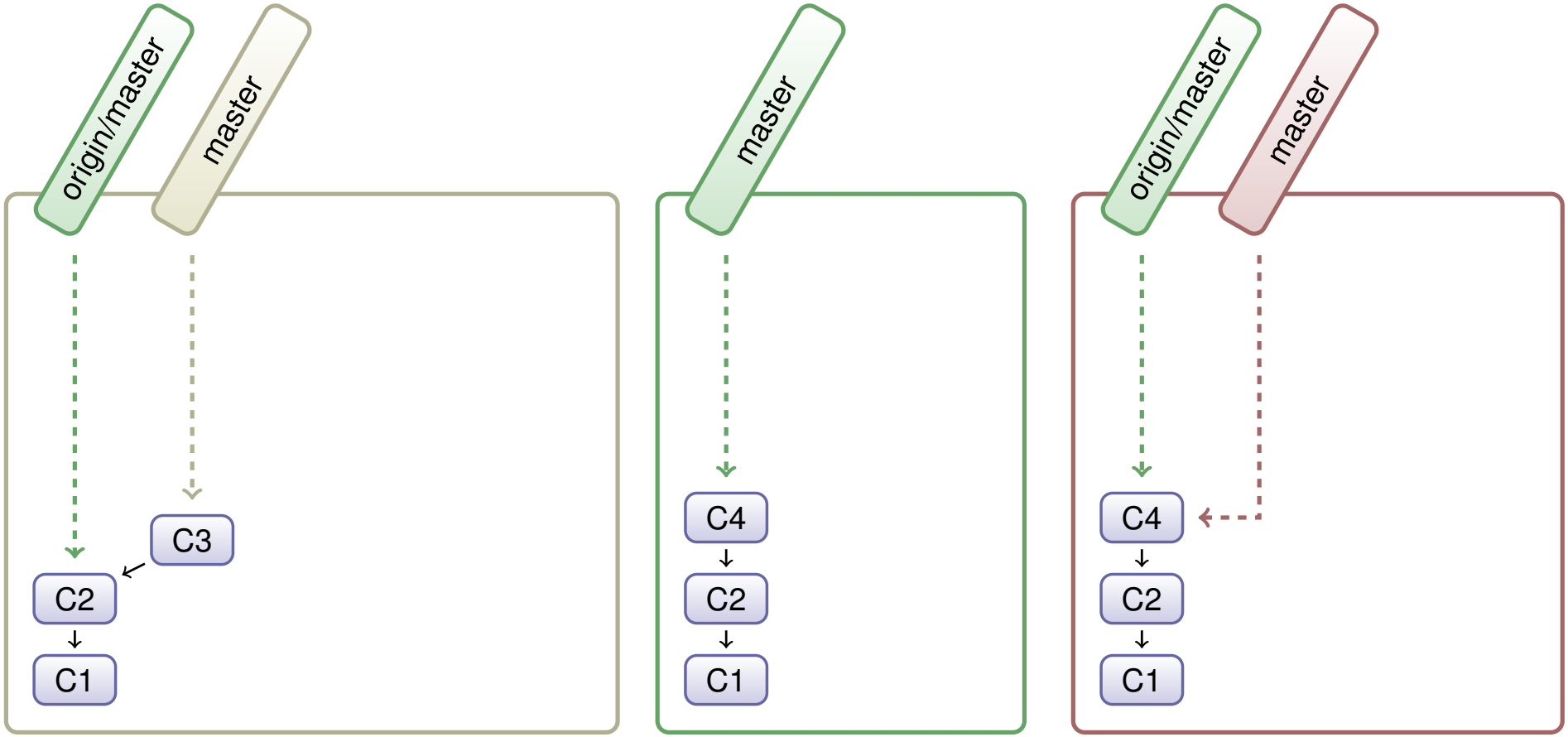

```
git push origin
```

Dépôts distants et gestion de la concurrence.



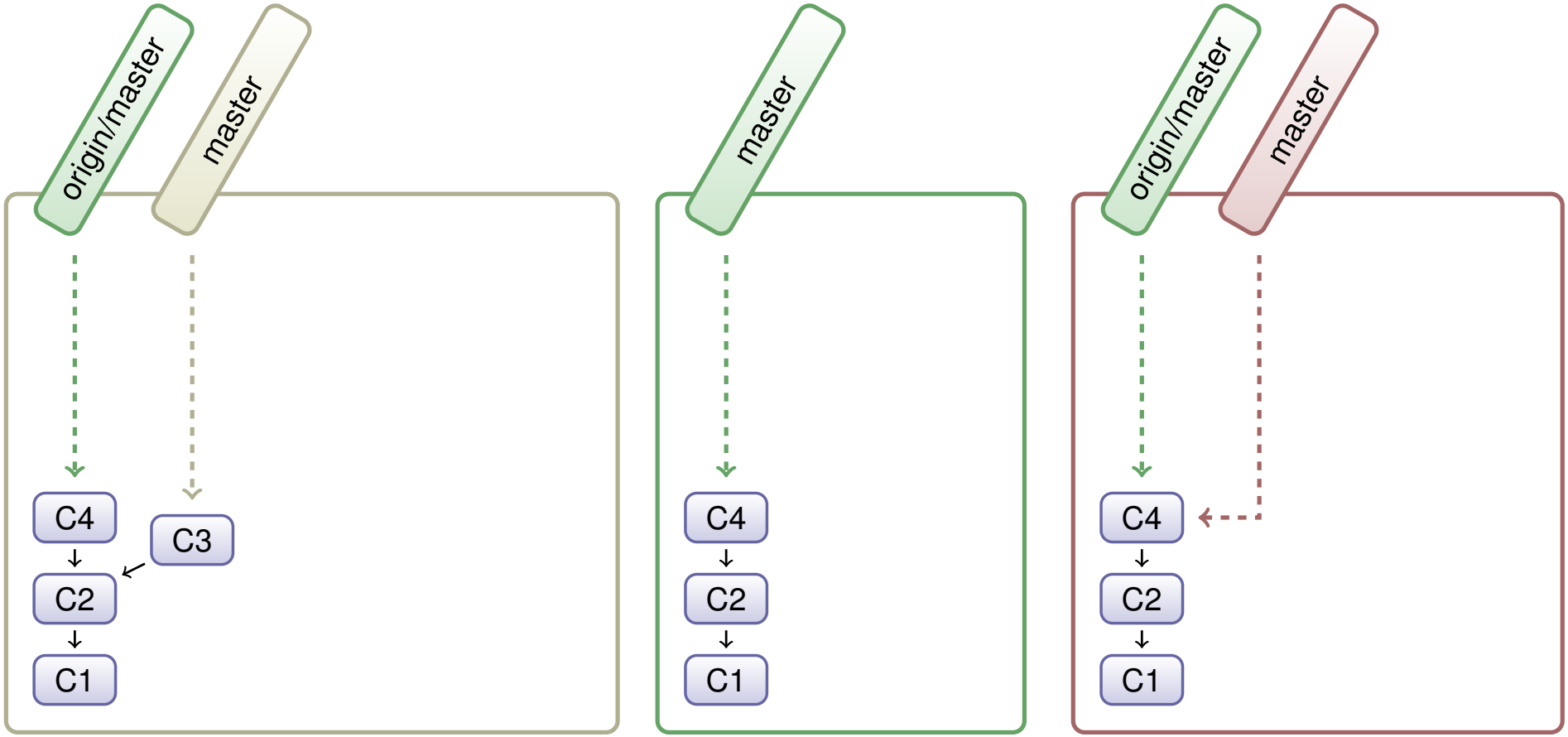

```
git push origin
```

Dépôts distants et gestion de la concurrence.



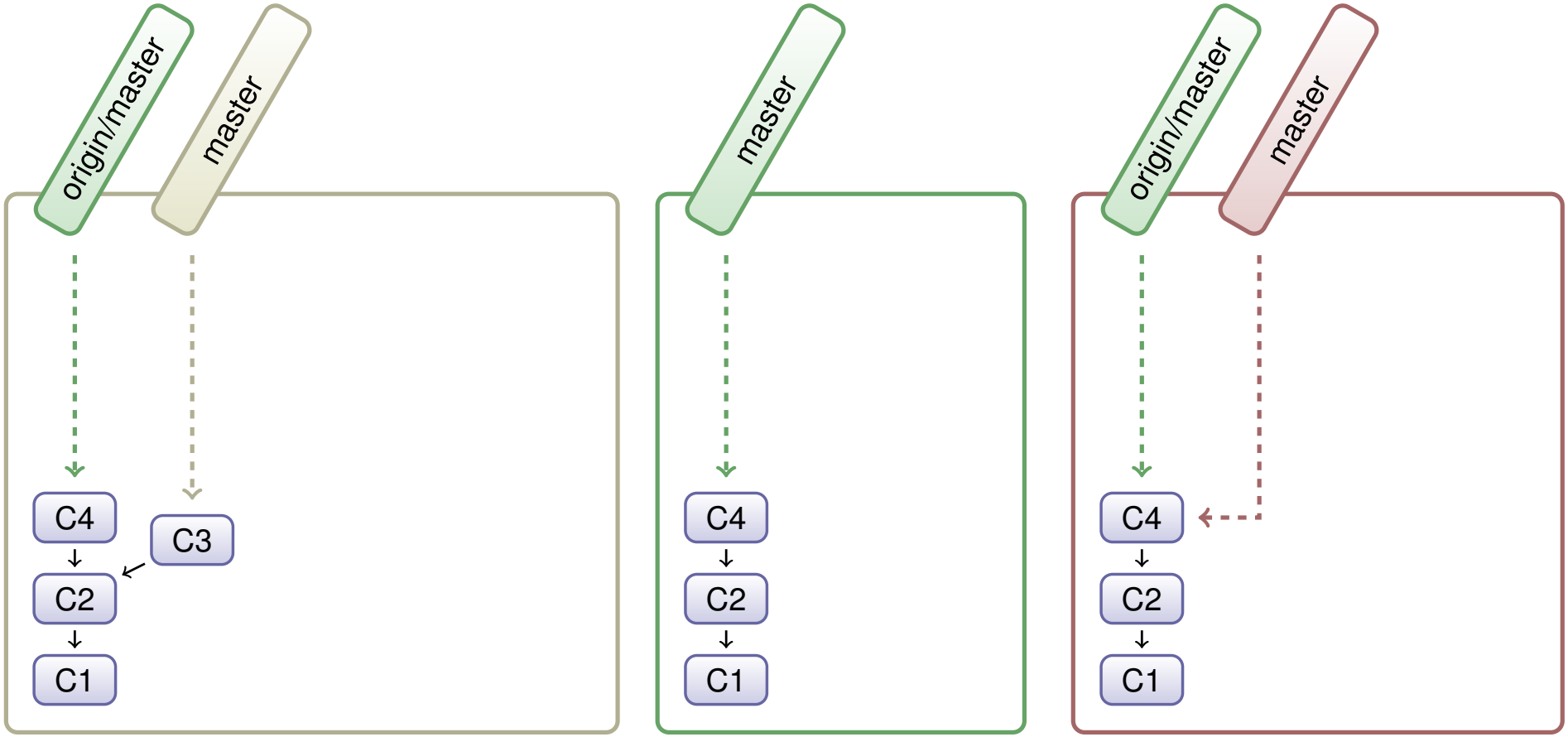

```
git fetch origin
```

Dépôts distants et gestion de la concurrence.



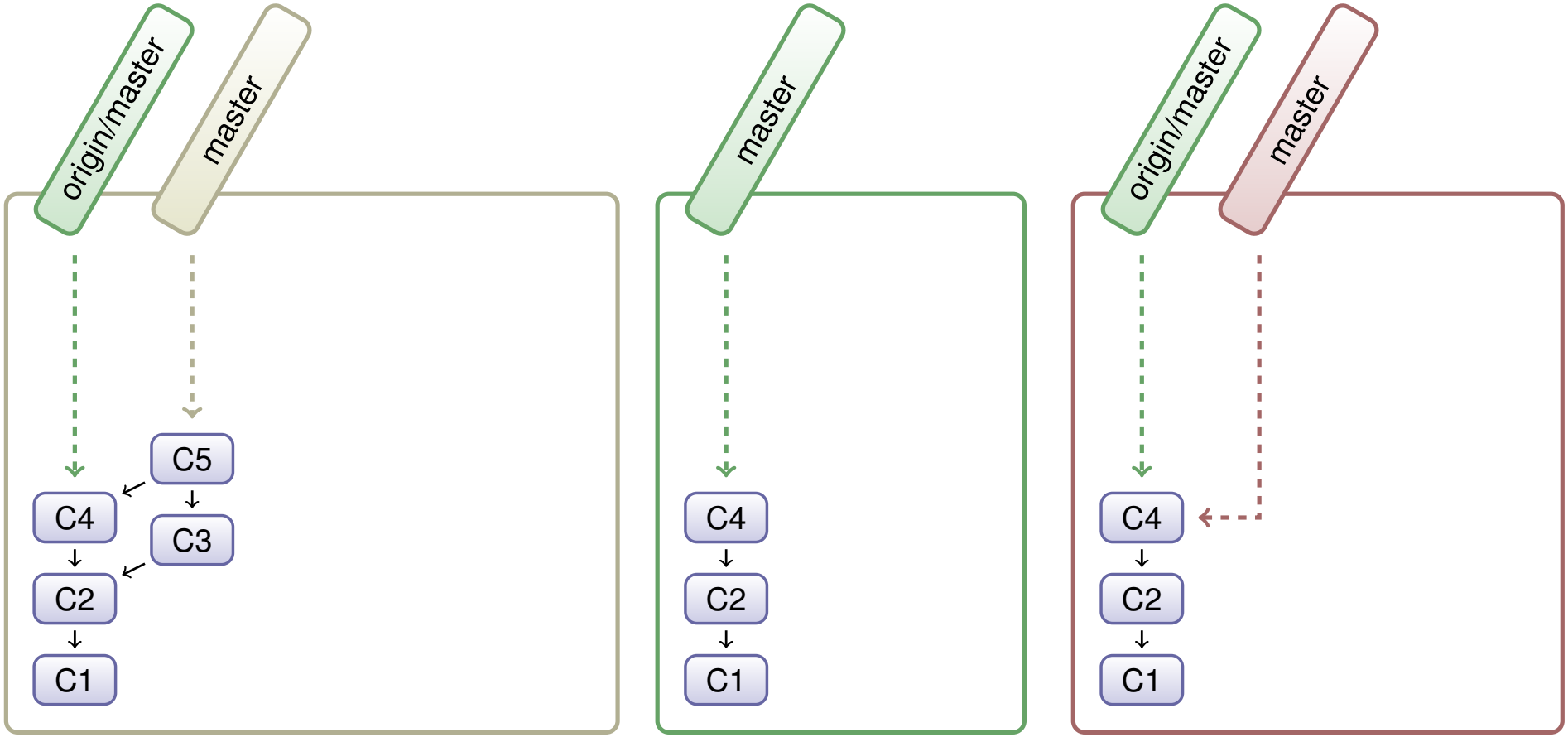

```
git fetch origin
```

Dépôts distants et gestion de la concurrence.



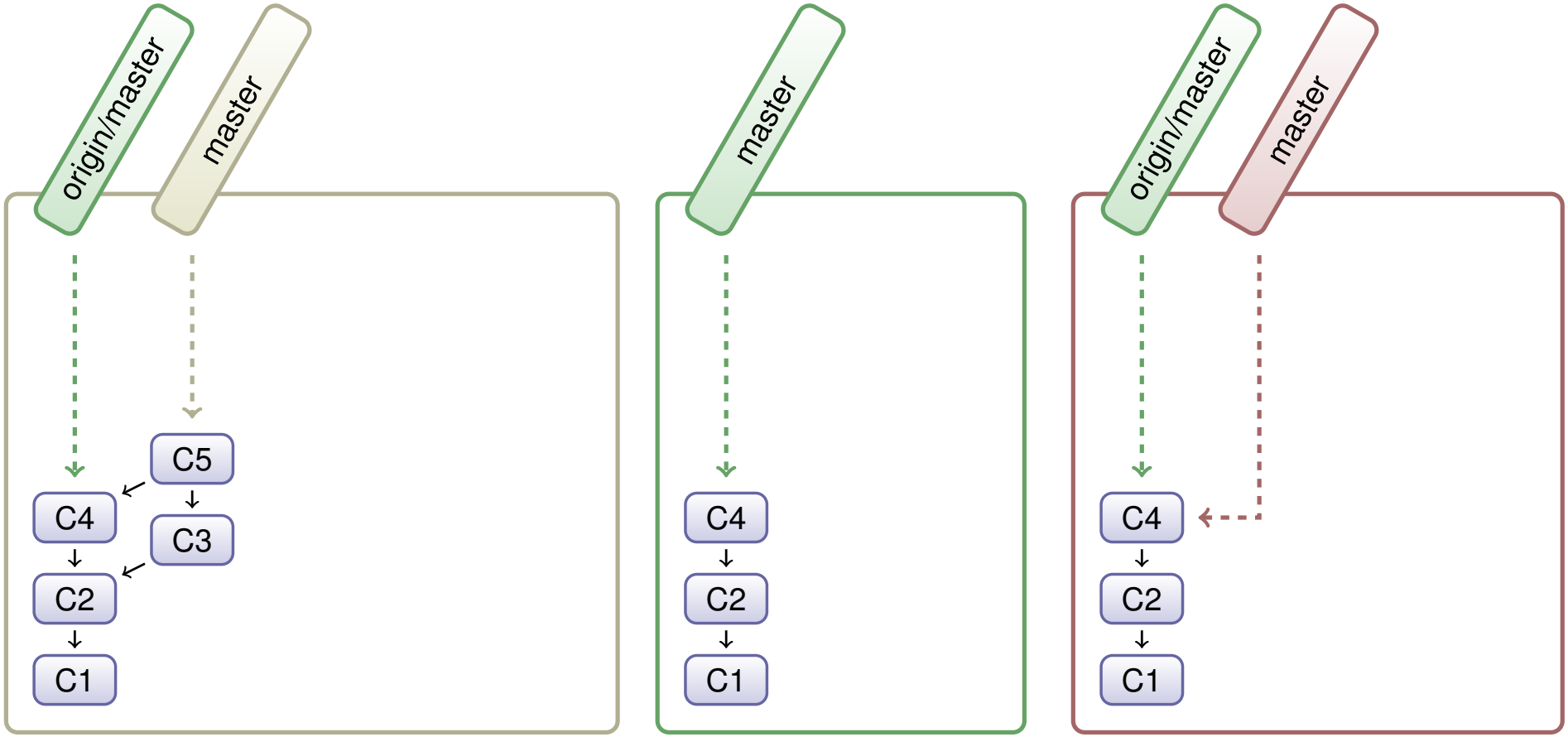

```
git merge origin
```

Dépôts distants et gestion de la concurrence.



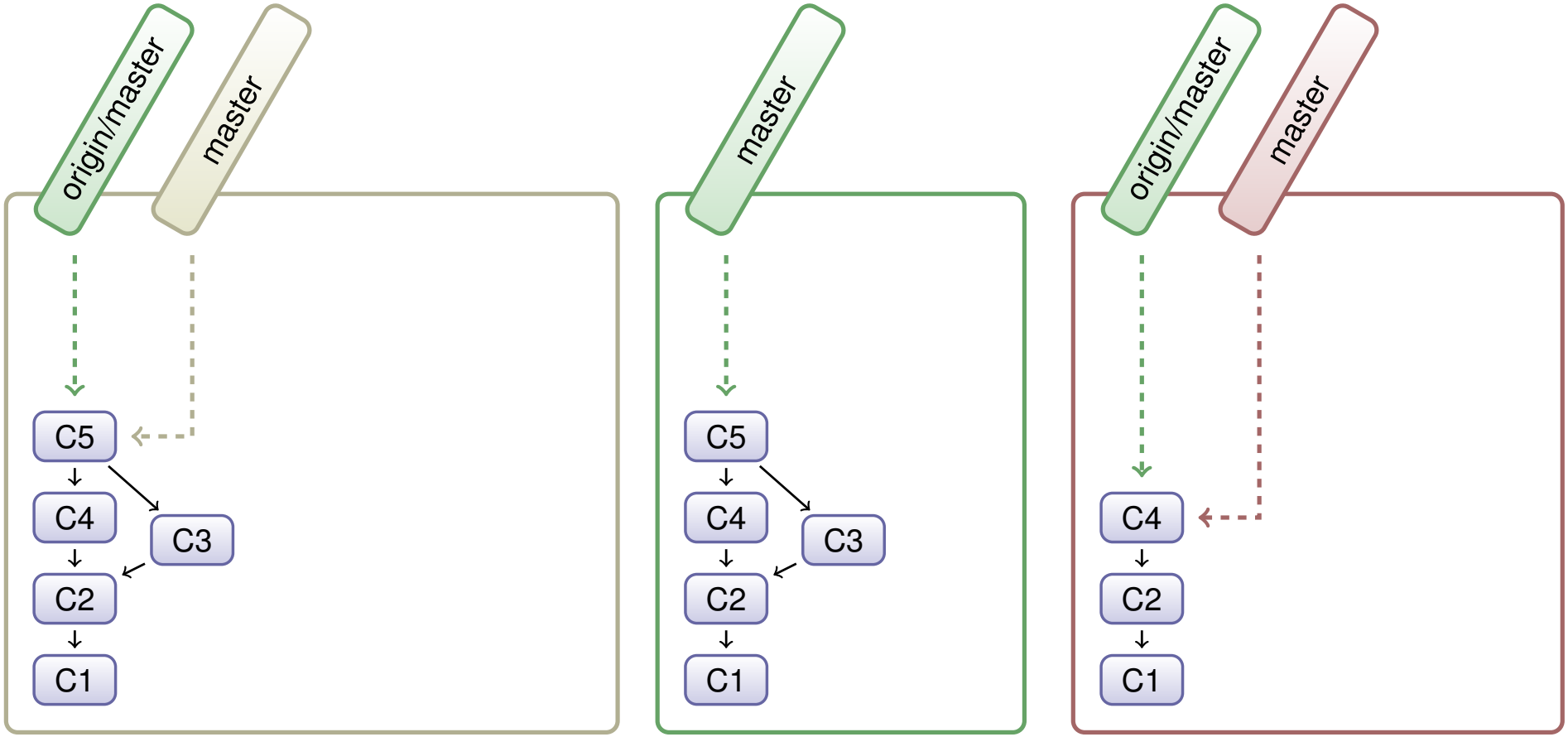

```
git merge origin
```

Dépôts distants et gestion de la concurrence.



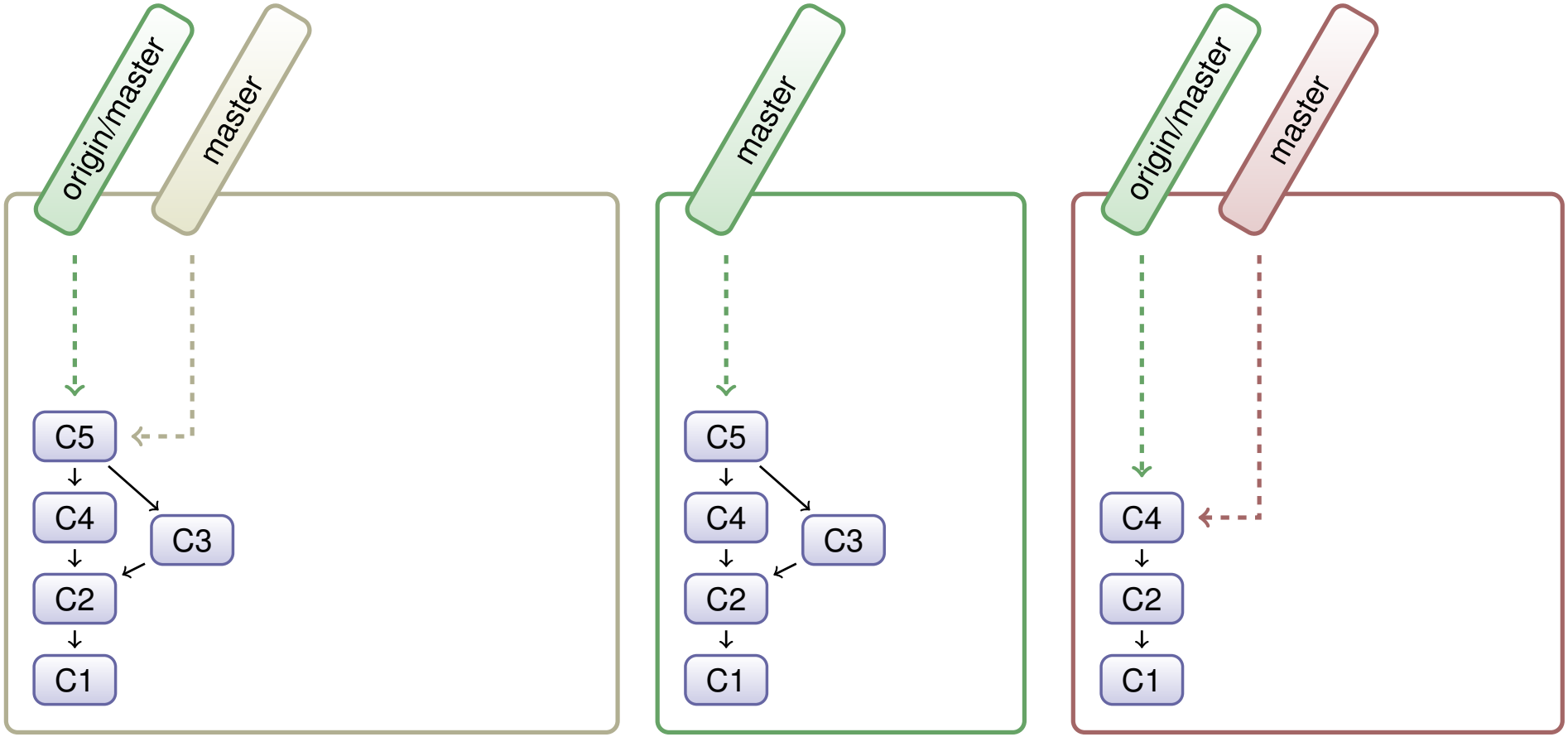

```
git push origin
```

Dépôts distants et gestion de la concurrence.



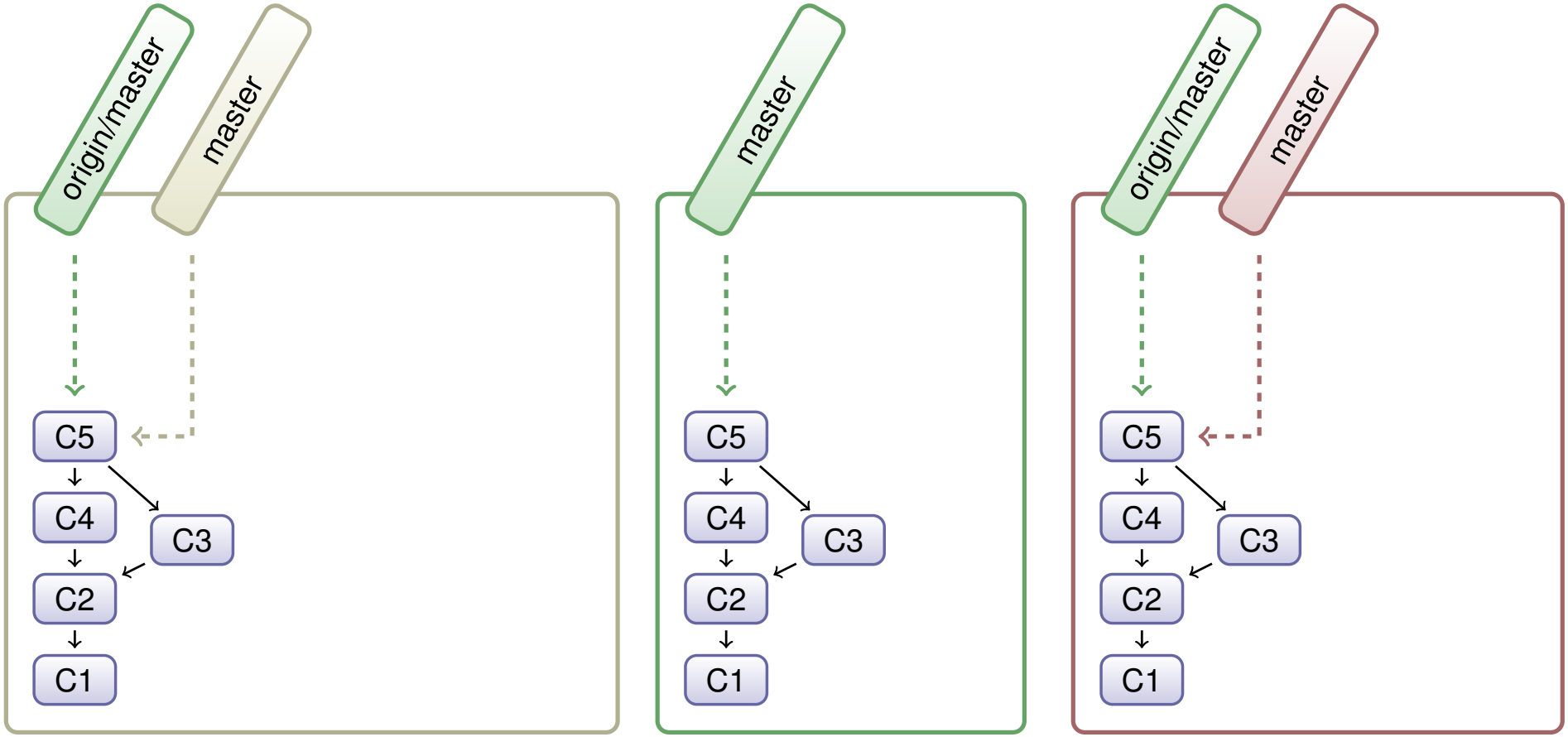

```
git push origin
```

Dépôts distants et gestion de la concurrence.



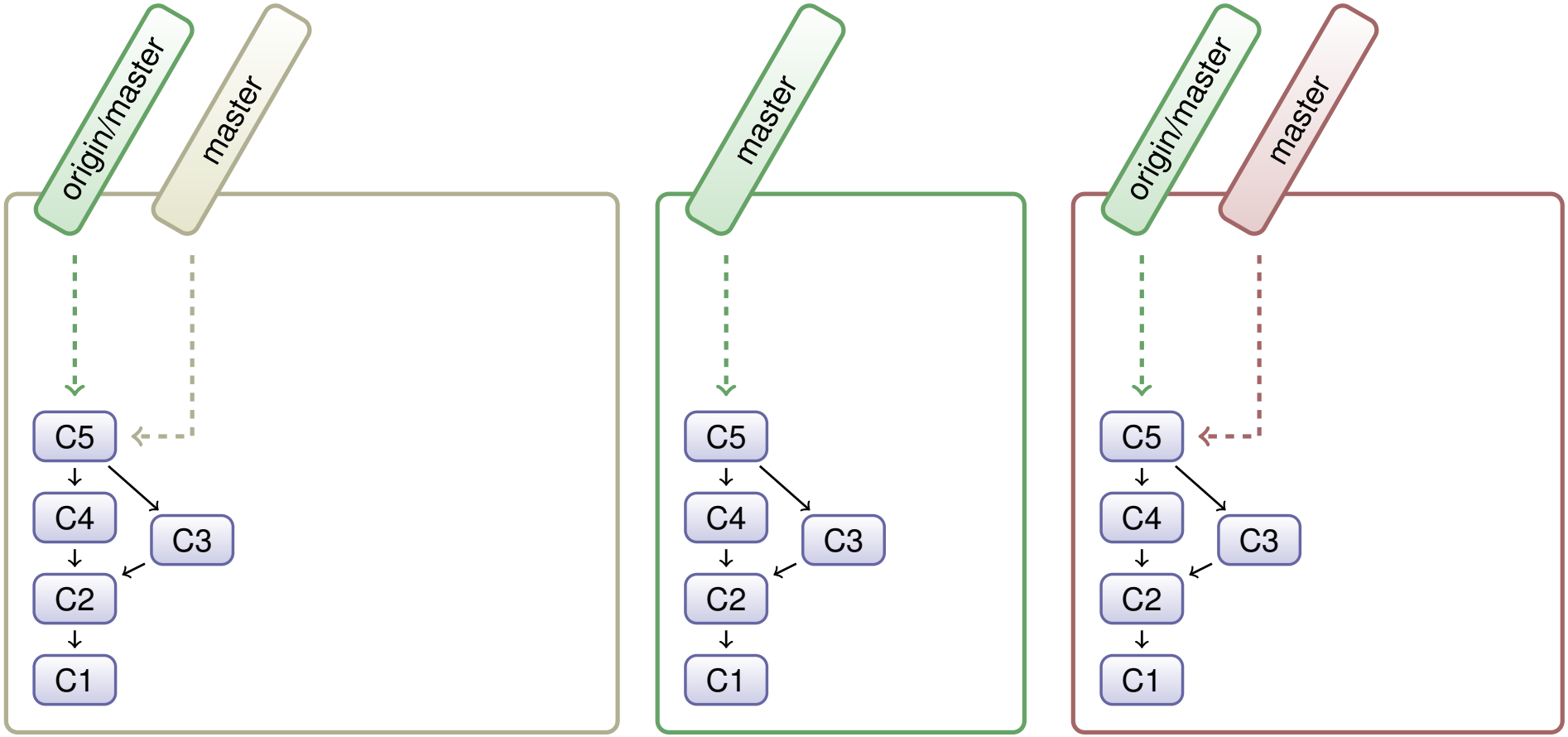

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



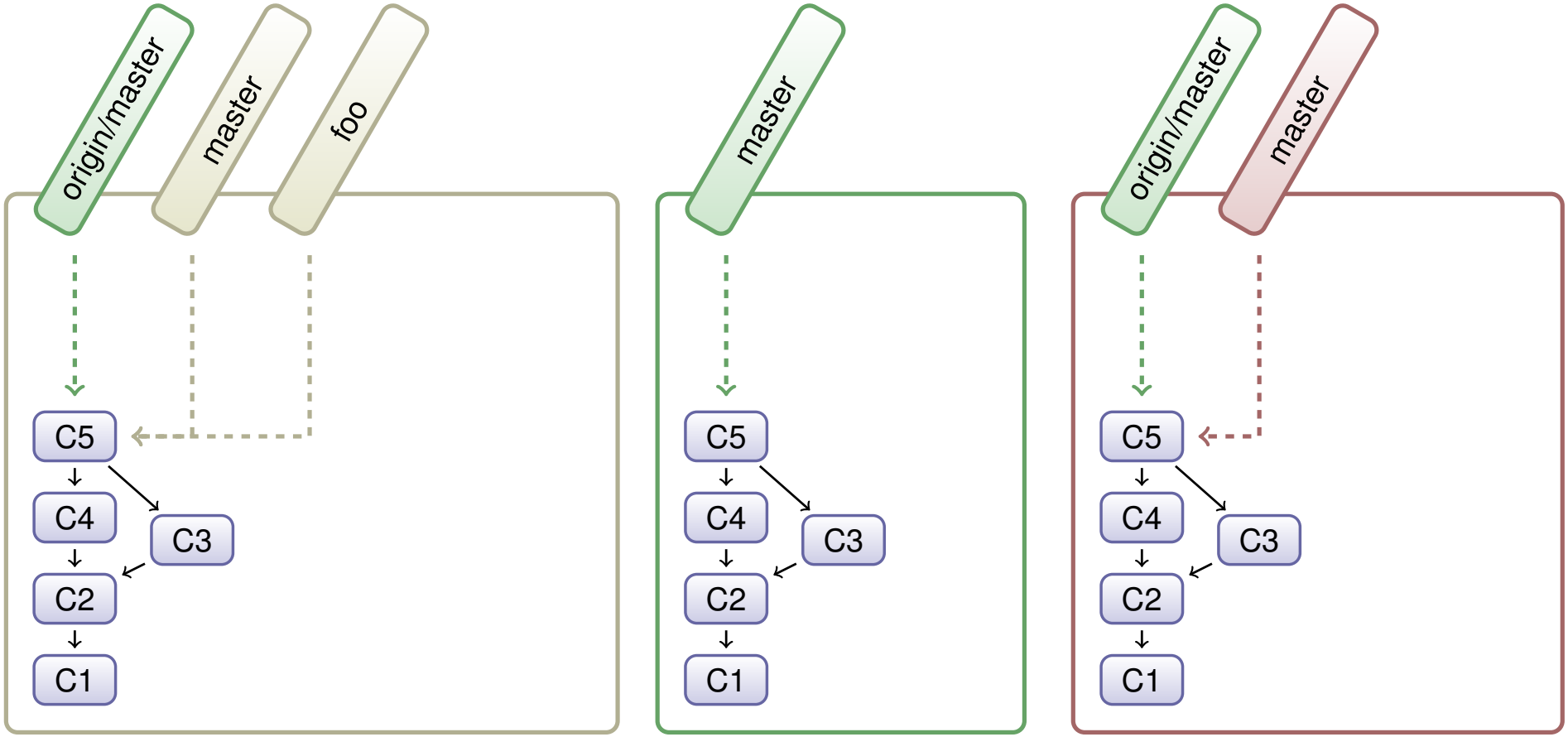

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



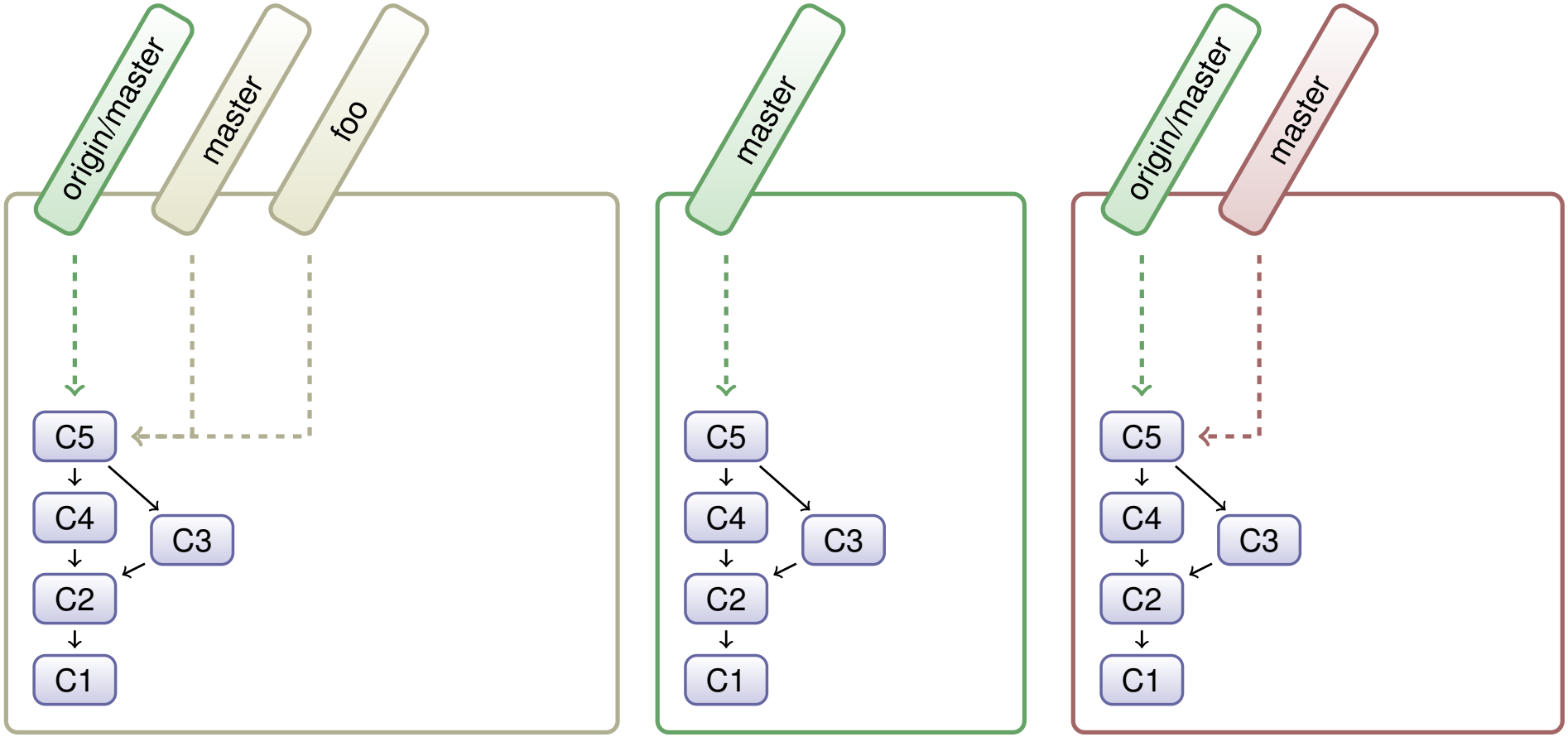

```
git checkout -b foo
```

Dépôts distants et gestion de la concurrence.



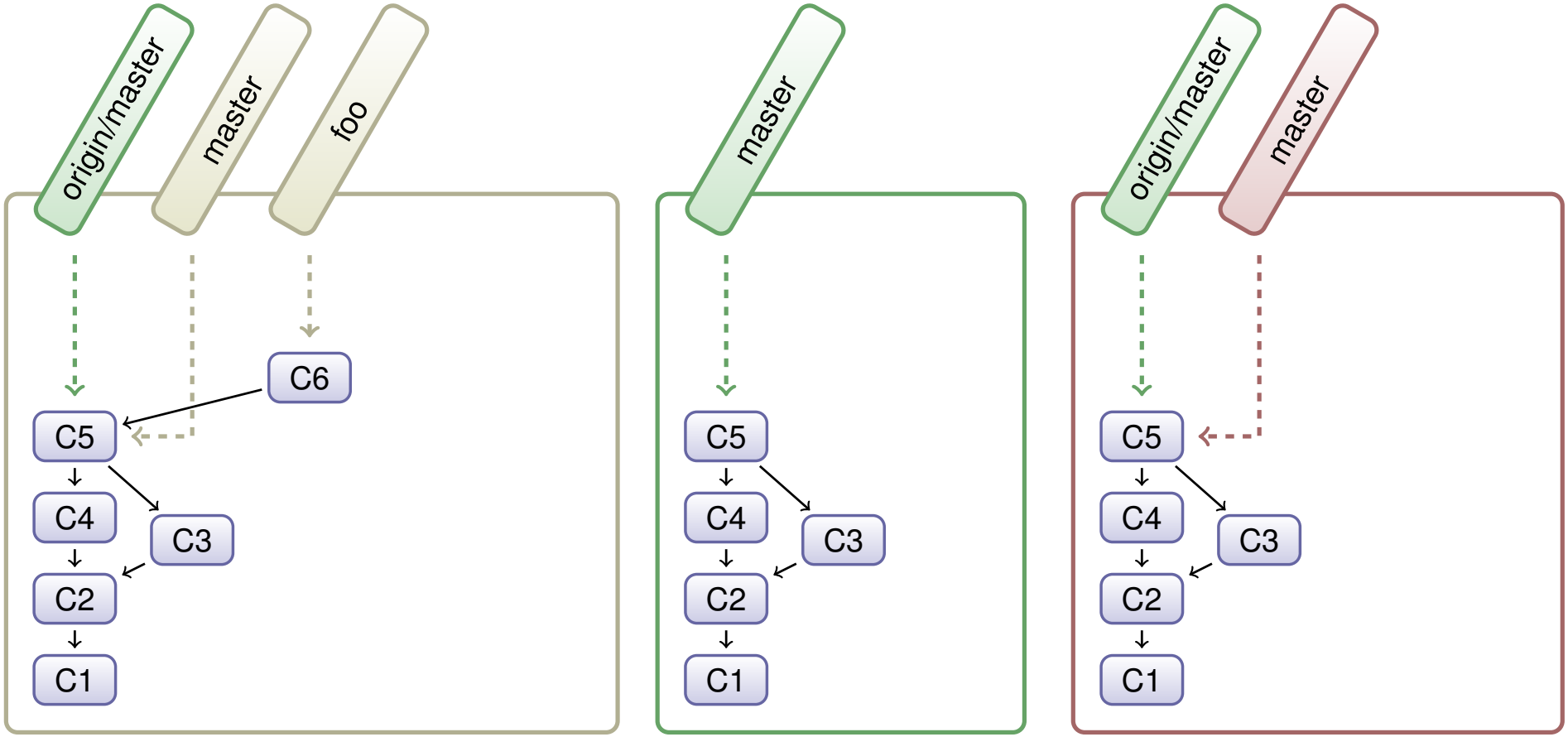

```
git checkout -b foo
```

Dépôts distants et gestion de la concurrence.



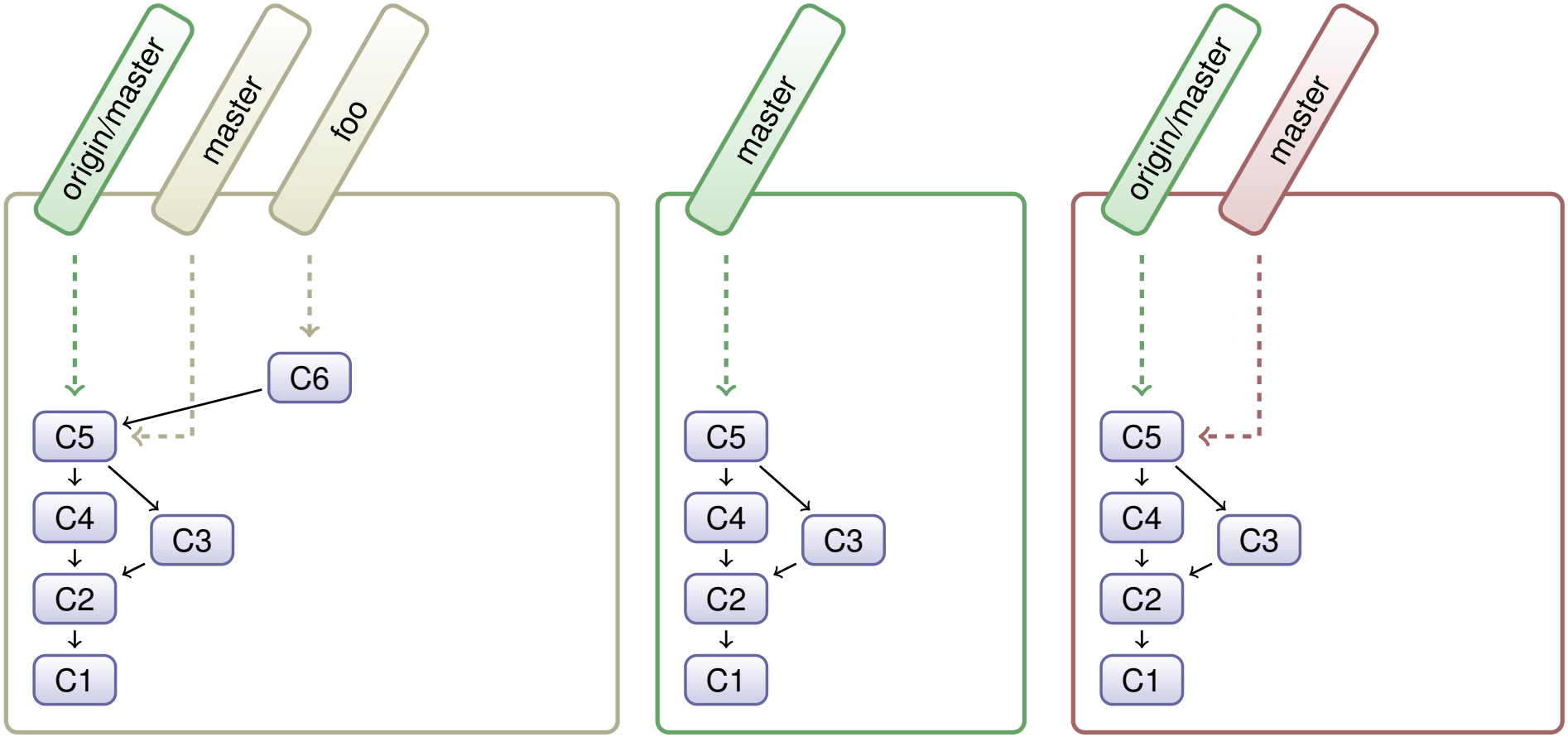

```
git commit
```

Dépôts distants et gestion de la concurrence.



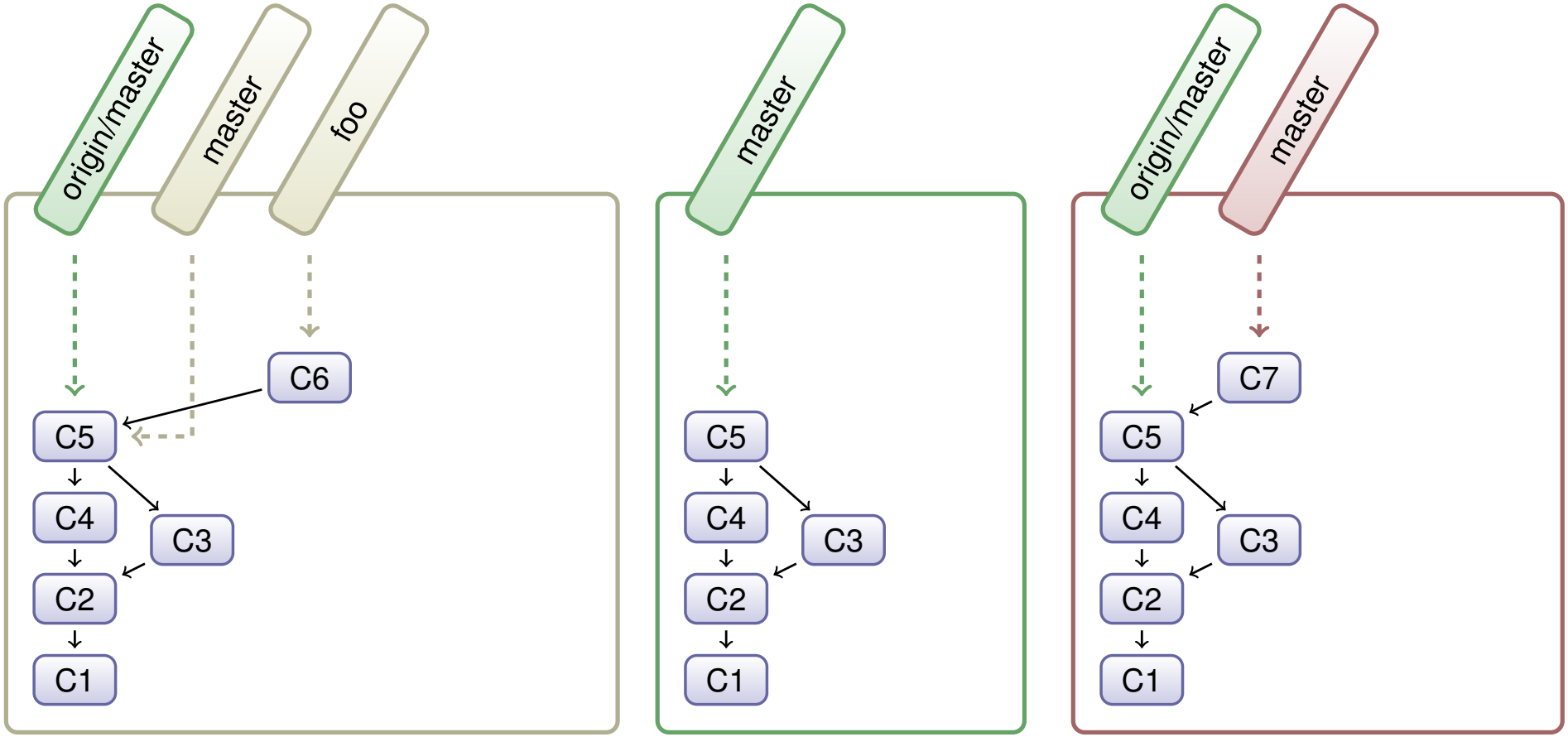

```
git commit
```

Dépôts distants et gestion de la concurrence.



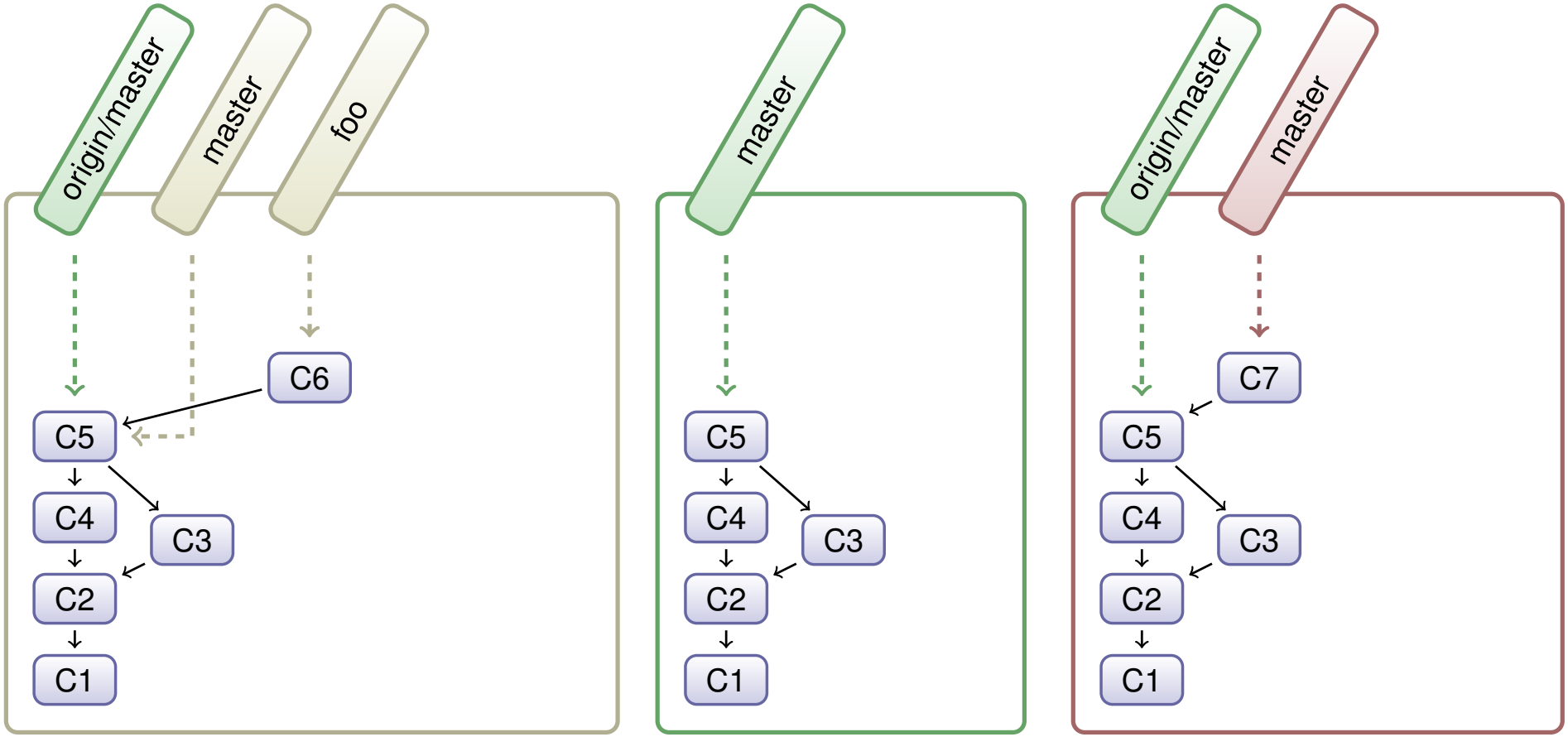

```
git commit
```

Dépôts distants et gestion de la concurrence.



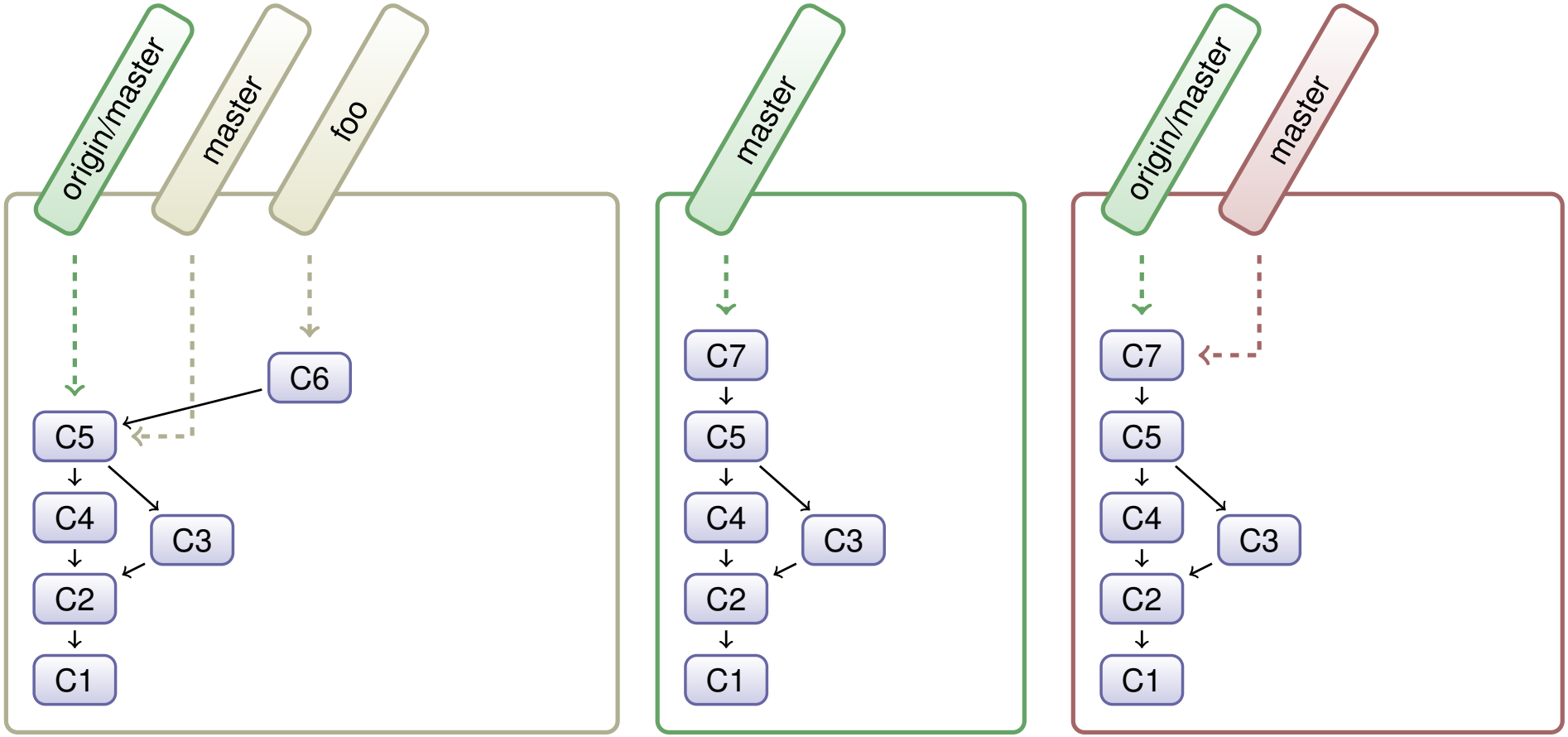

```
git commit
```

Dépôts distants et gestion de la concurrence.



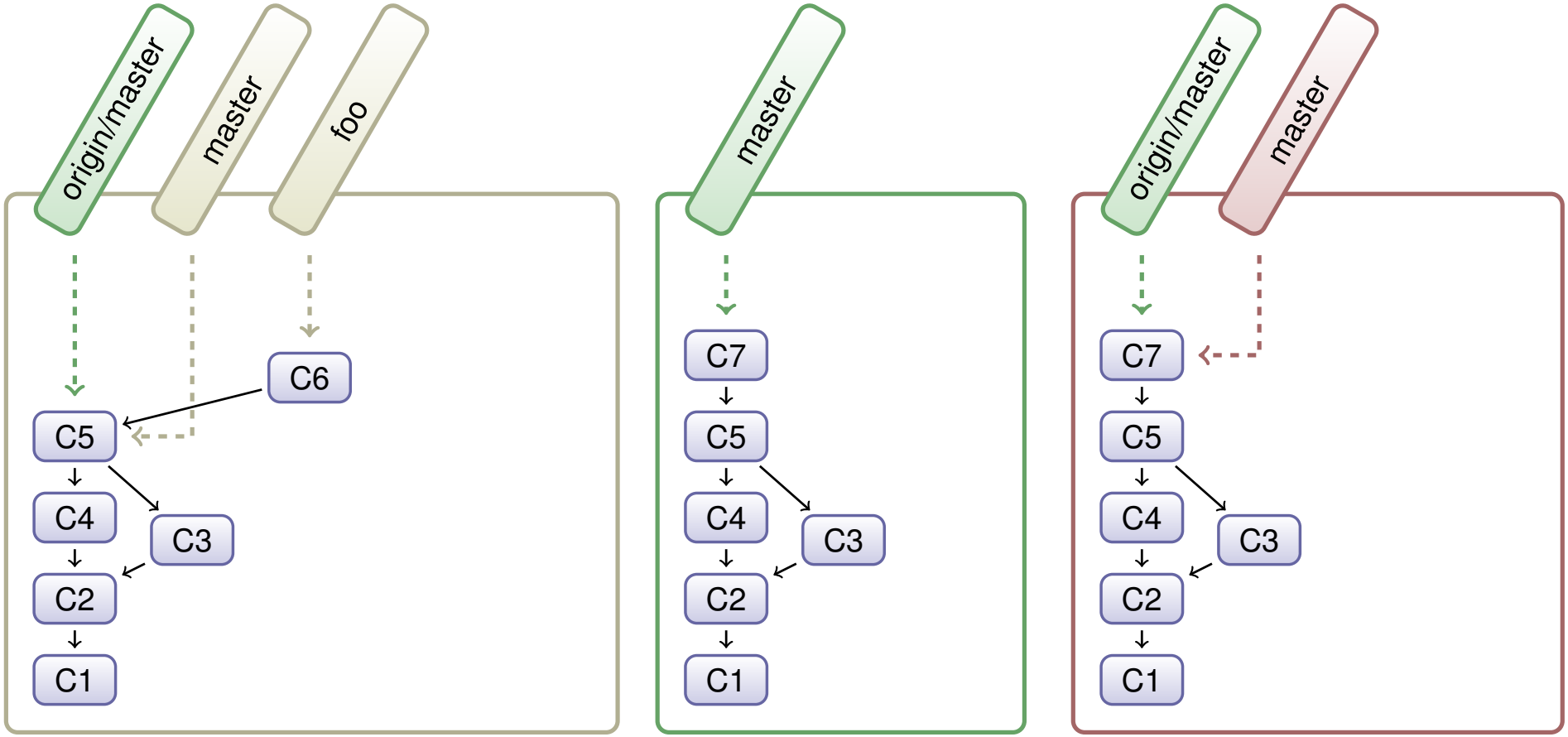

```
git push origin
```

Dépôts distants et gestion de la concurrence.



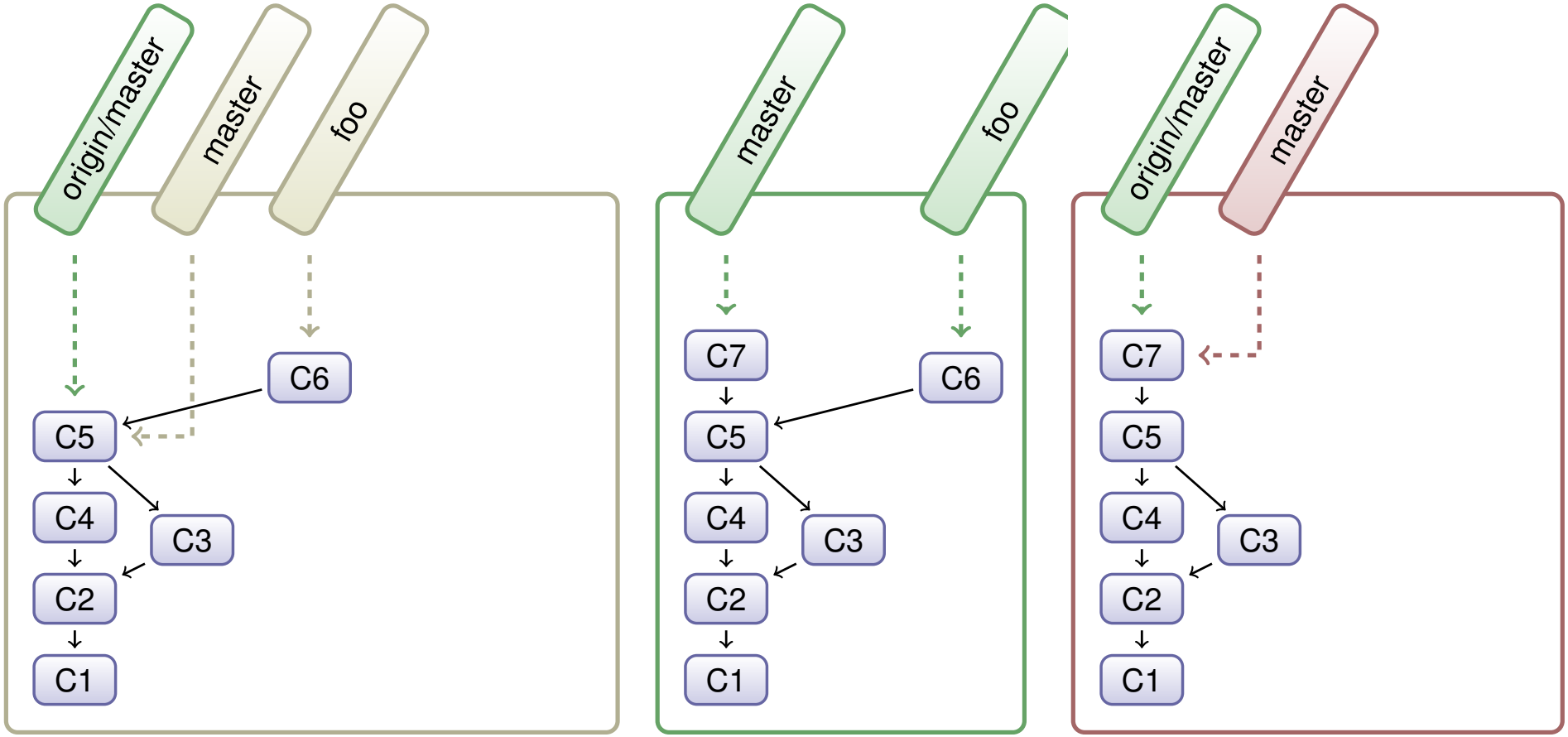

```
git push origin
```

Dépôts distants et gestion de la concurrence.



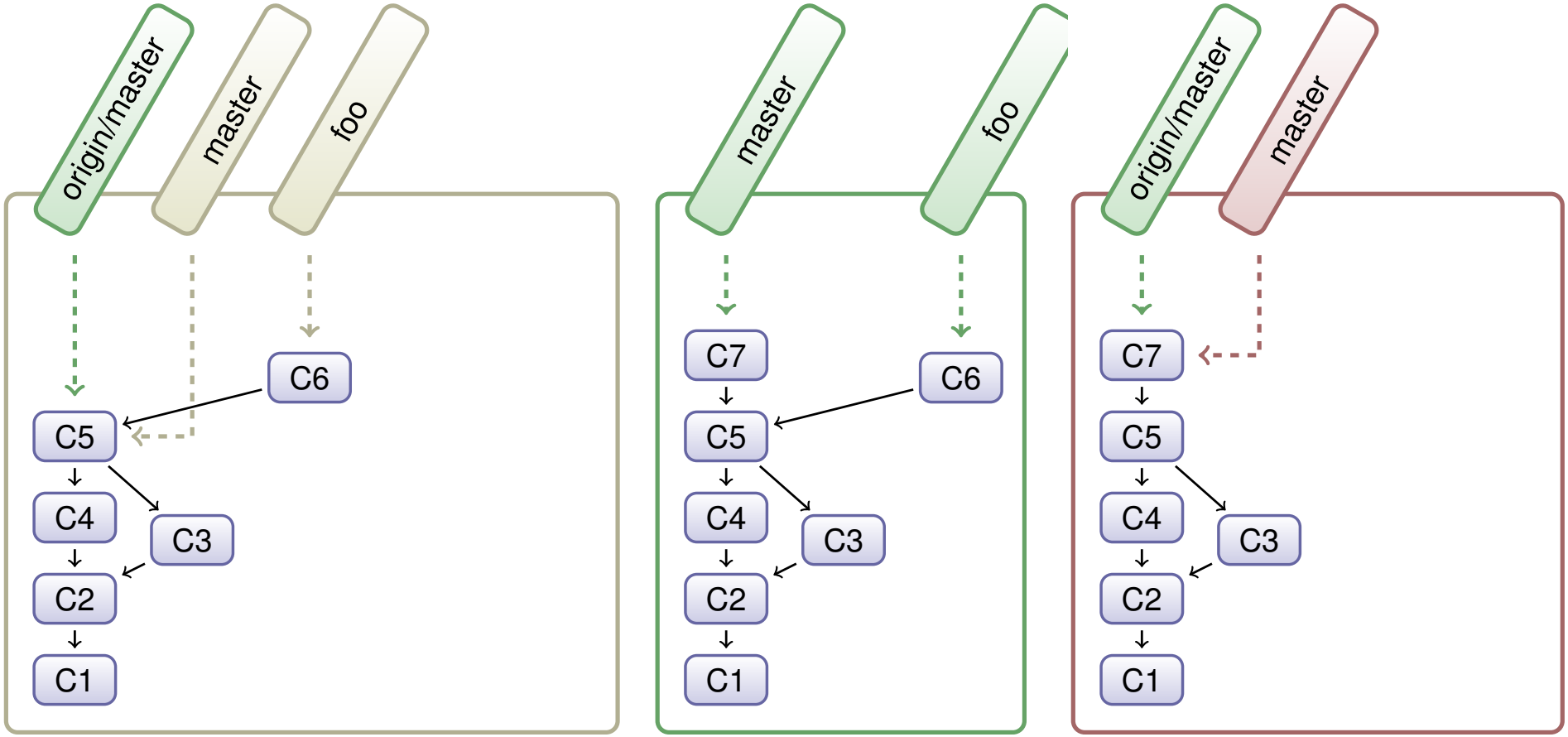

```
git push origin foo
```

Dépôts distants et gestion de la concurrence.



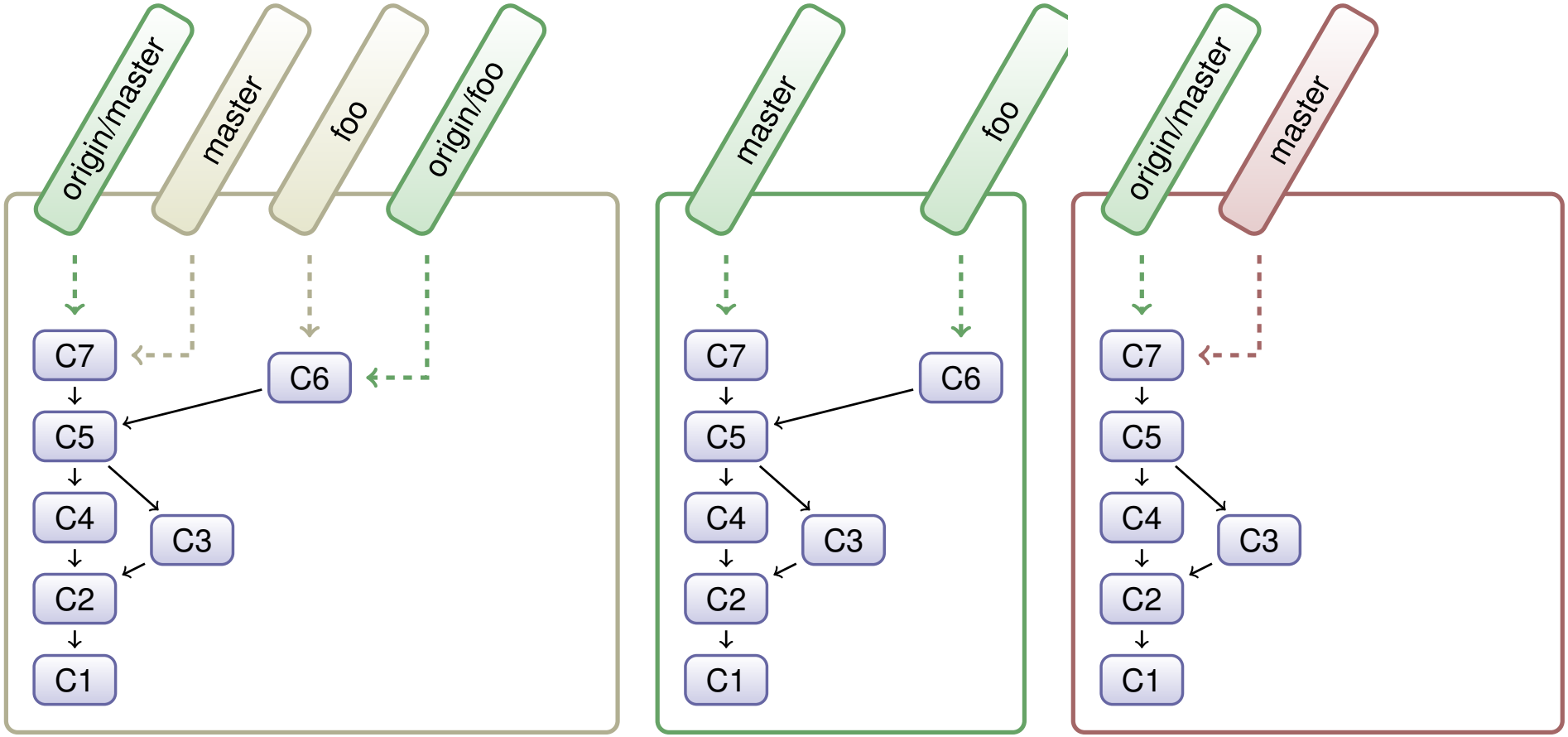

```
git push origin foo
```

Dépôts distants et gestion de la concurrence.



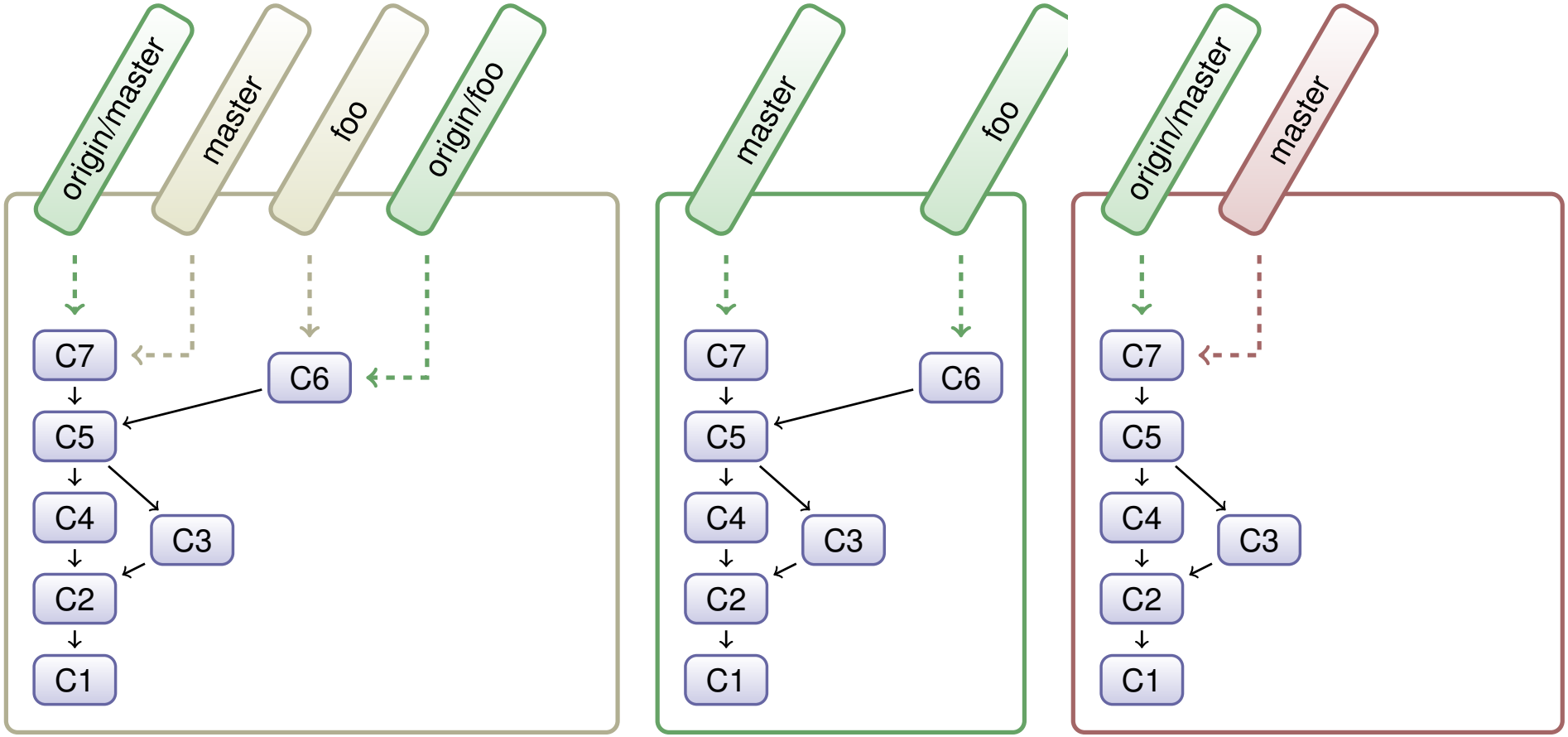

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



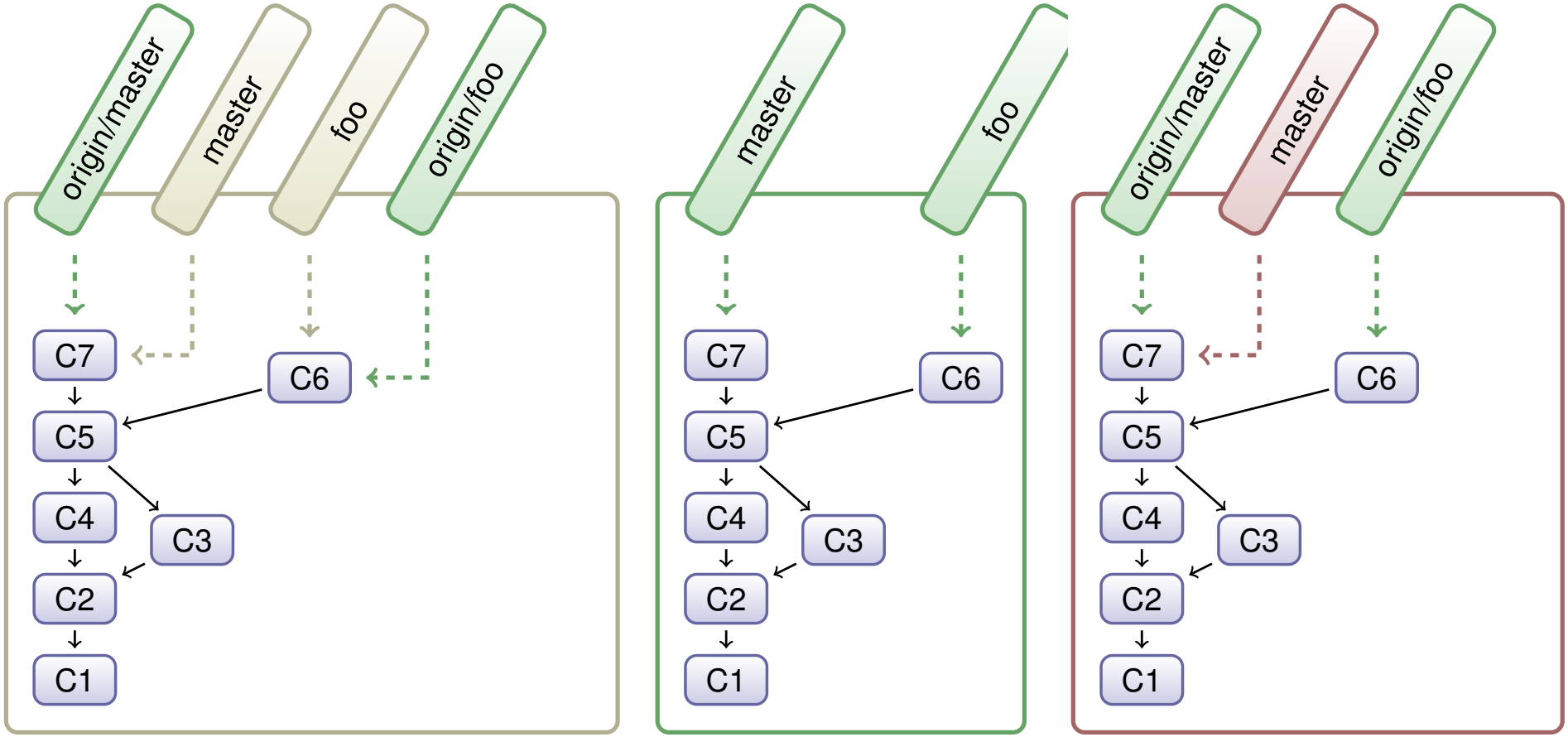

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



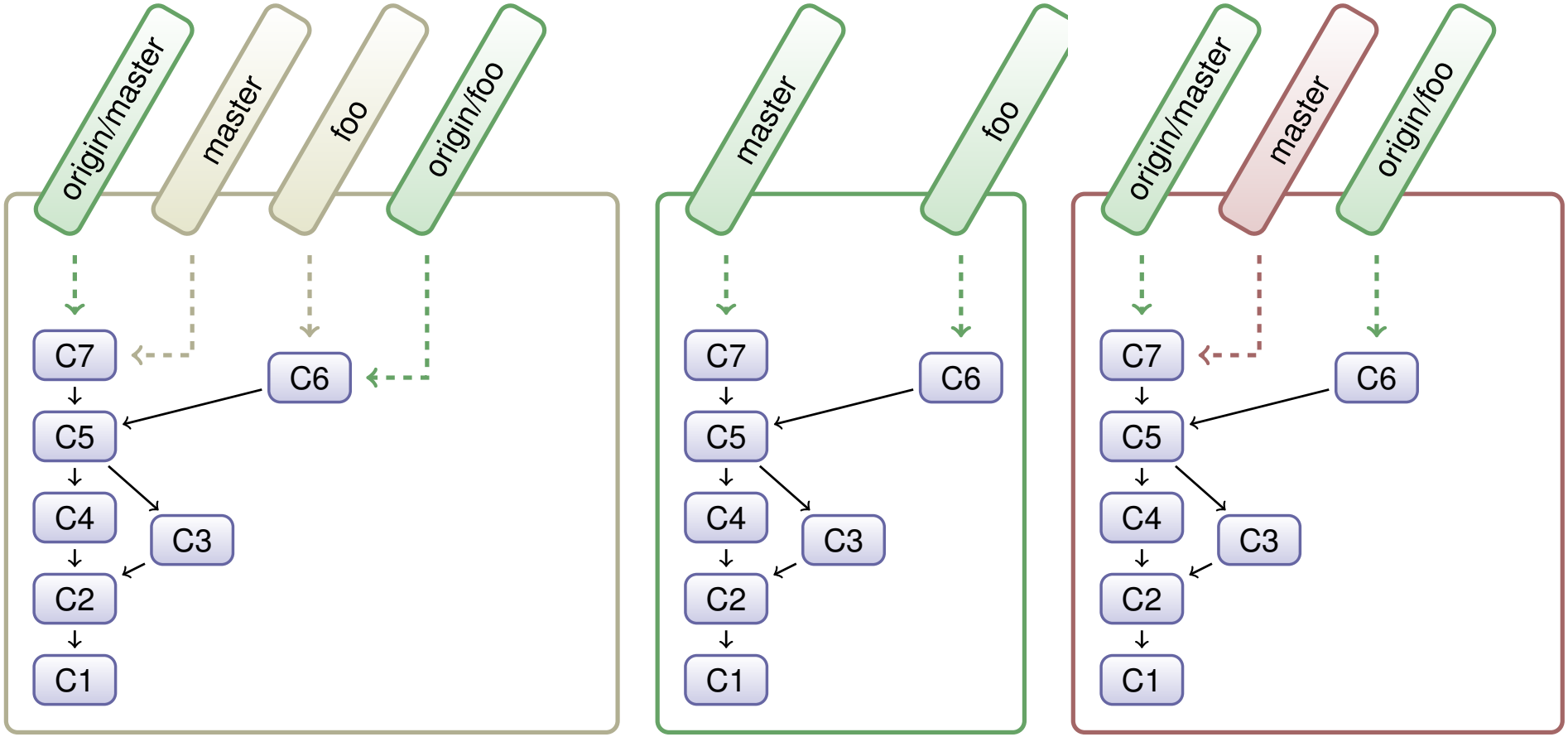

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



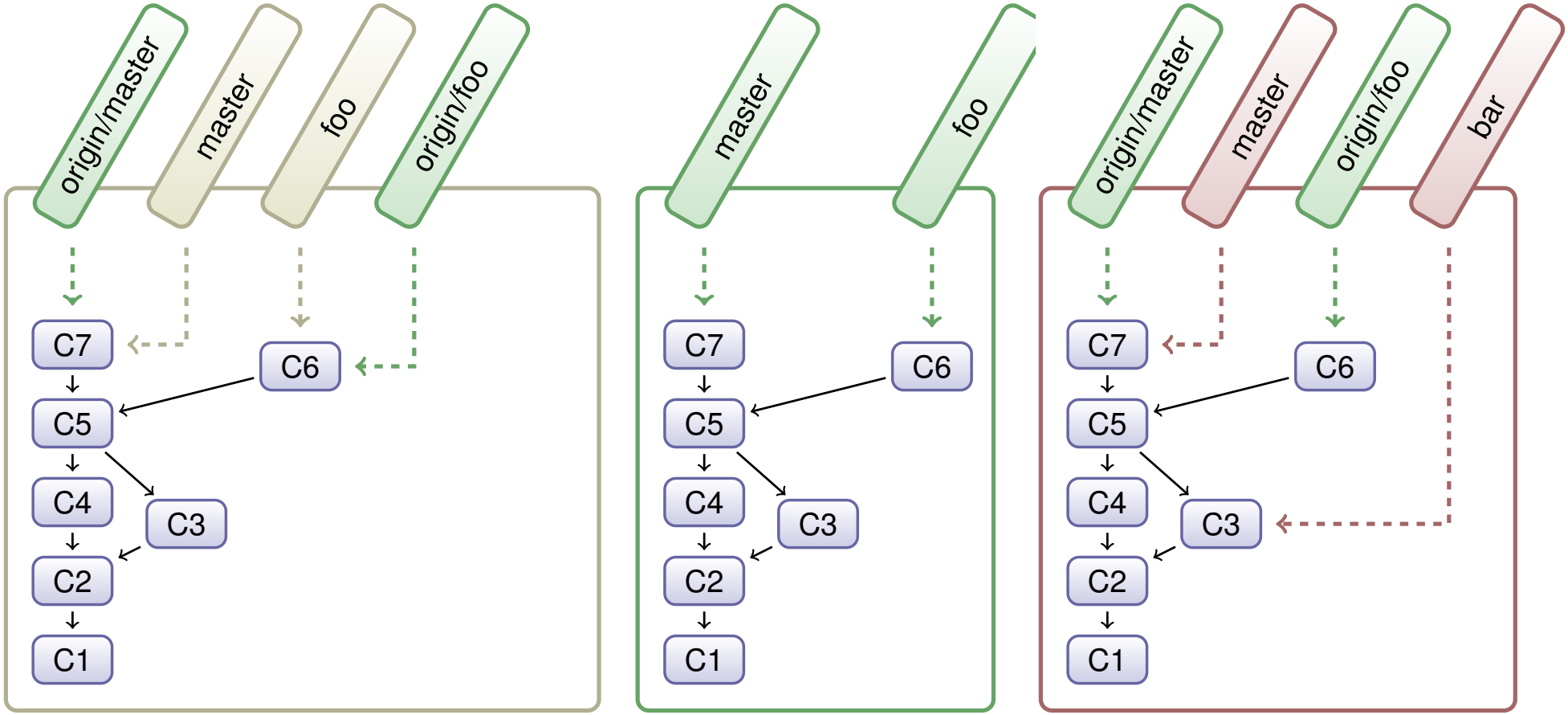

```
git pull origin
```

Dépôts distants et gestion de la concurrence.



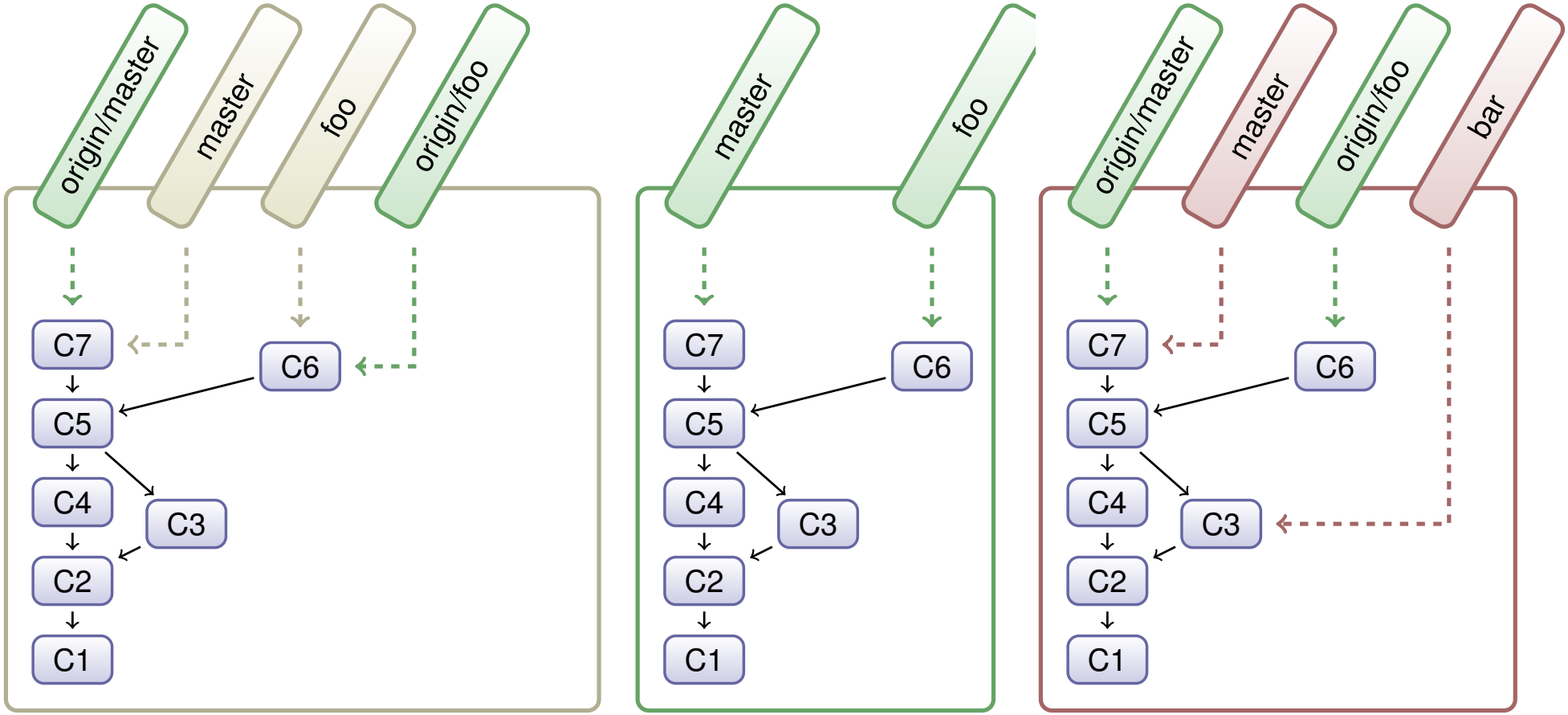

```
git branch bar
git checkout bar
```

Dépôts distants et gestion de la concurrence.



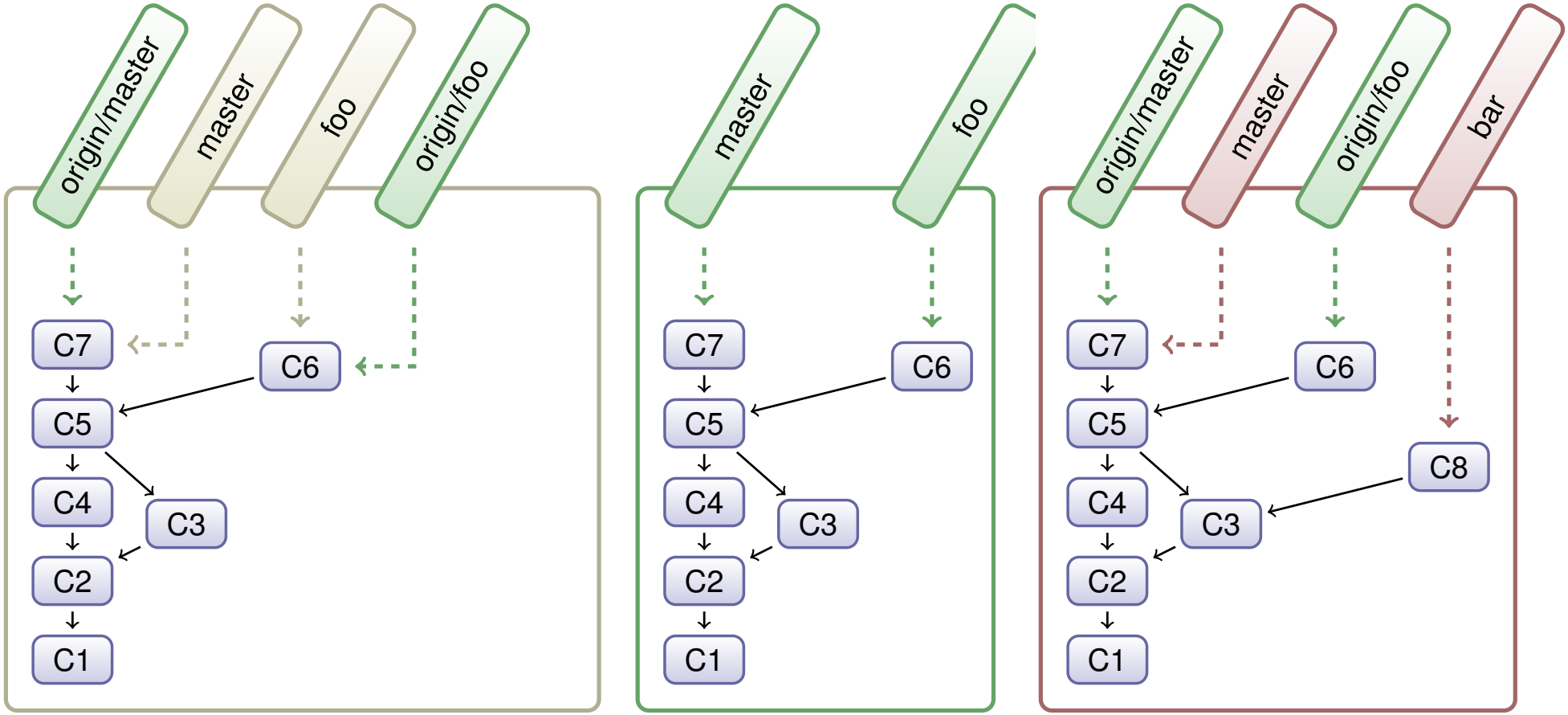

```
git branch bar
git checkout bar
```

Dépôts distants et gestion de la concurrence.



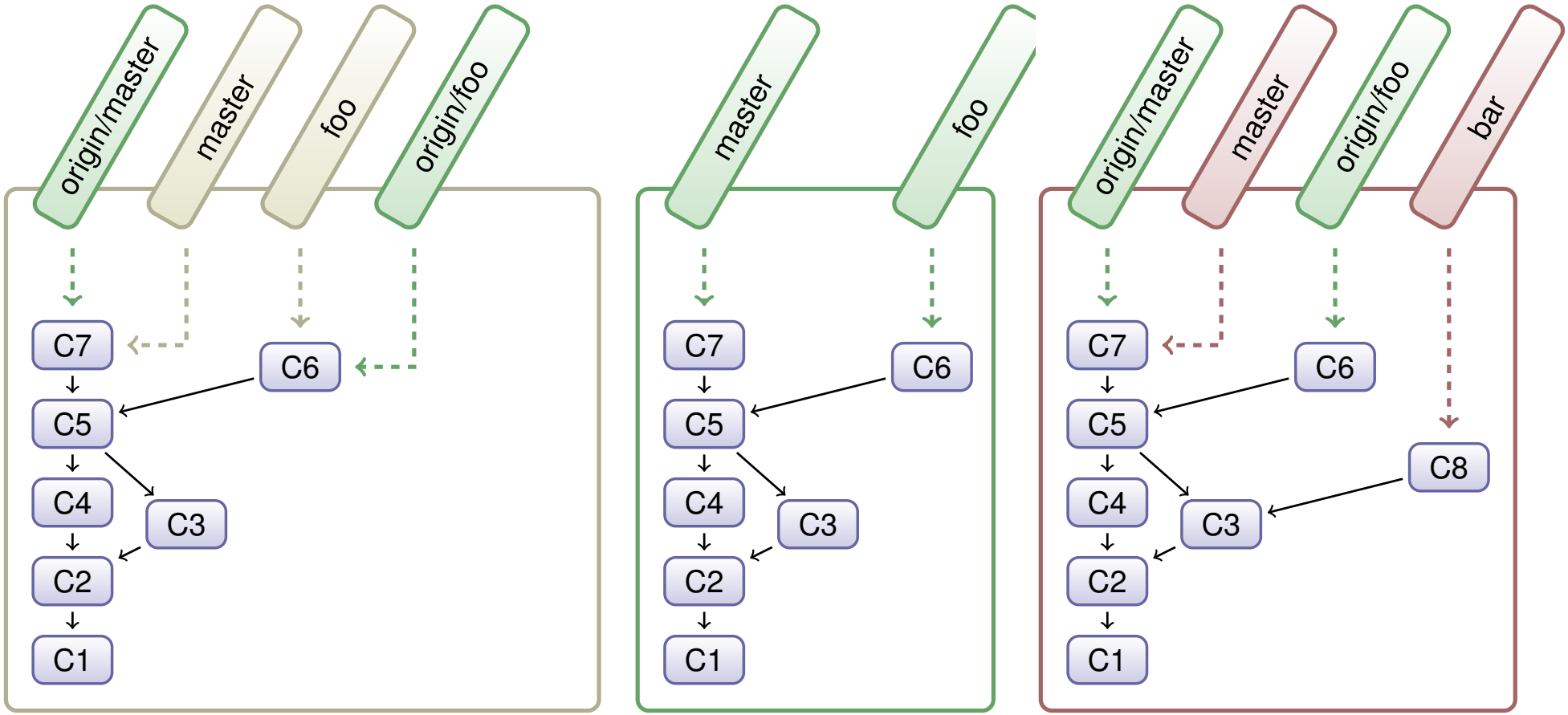

```
git commit
```

Dépôts distants et gestion de la concurrence.



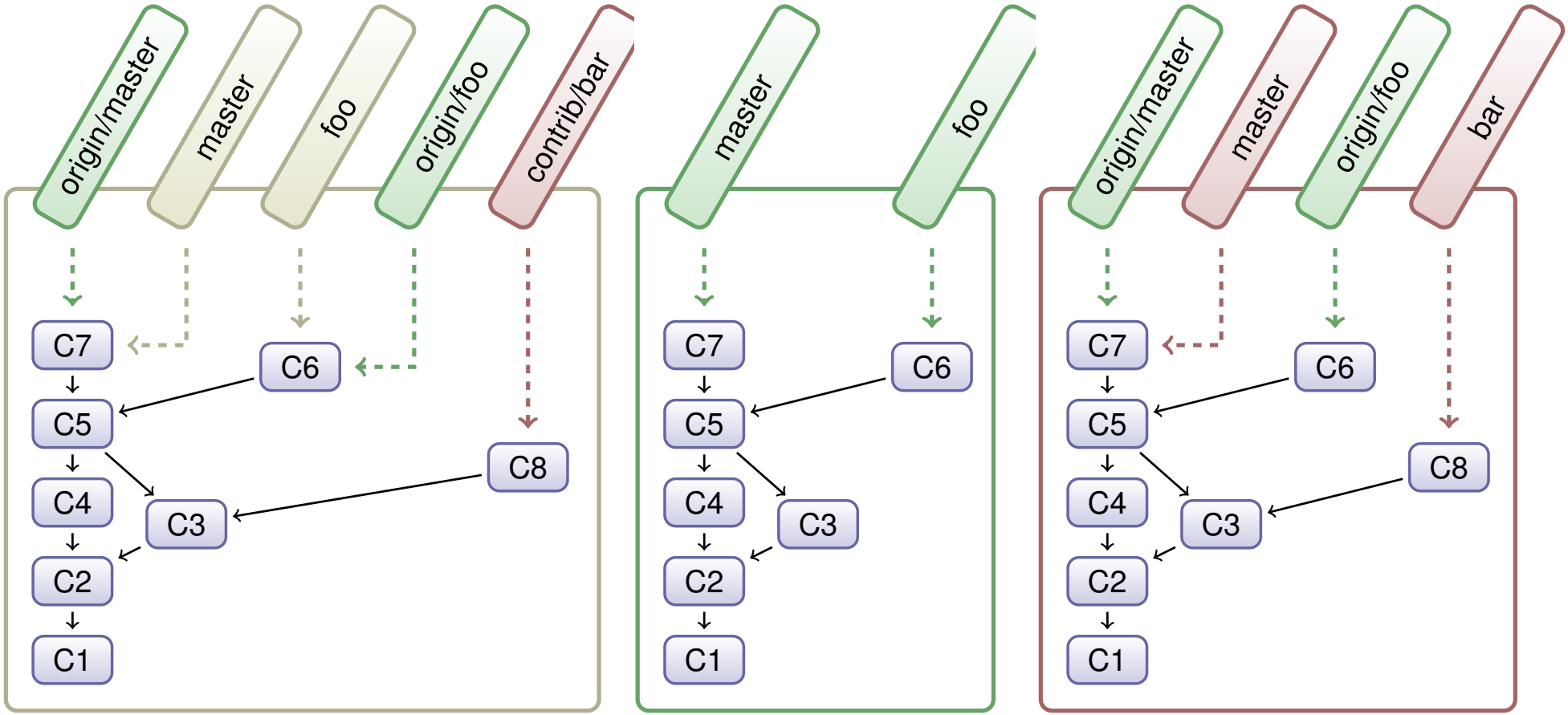

```
git commit
```

Dépôts distants et gestion de la concurrence.




```
git remote add contrib git://...
git fetchcontrib/bar
```

Dépôts distants et gestion de la concurrence.




```
git remote add contrib git://...
git fetchcontrib/bar
```

Outline

Modes de développement

Git permet aux développeurs de gérer leurs sources de 4 manières :

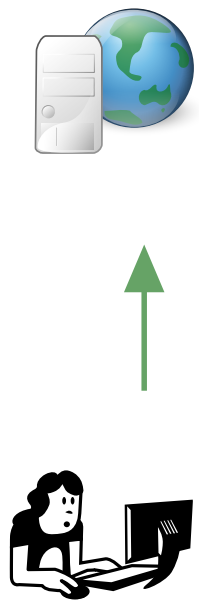
- un dépôt centralisé à la CVS, SVN tout en conservant les avantages de la conservation du dépôt local ;
- un dépôt pour chaque développeur, chacun se synchronise chez les autres, méthode traditionnelle de Arch ;
- un dépôt pour chaque développeur et manager qui se synchronise, fait les merges nécessaires et envoie le tout sur un dépôt publique.
- une gestion par mails de dépôts, méthode de développement du noyau Linux et de son équipe de maintenance.

Modèle avec dépôt centralisé

"Centralized Workflow."

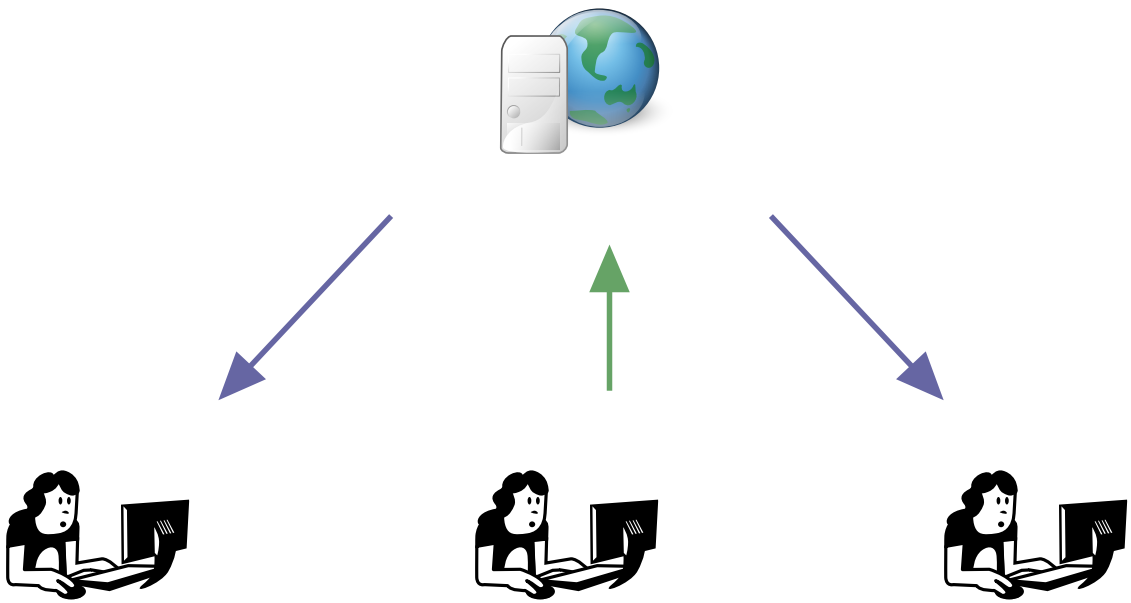
Modèle avec dépôt centralisé

"Centralized Workflow."



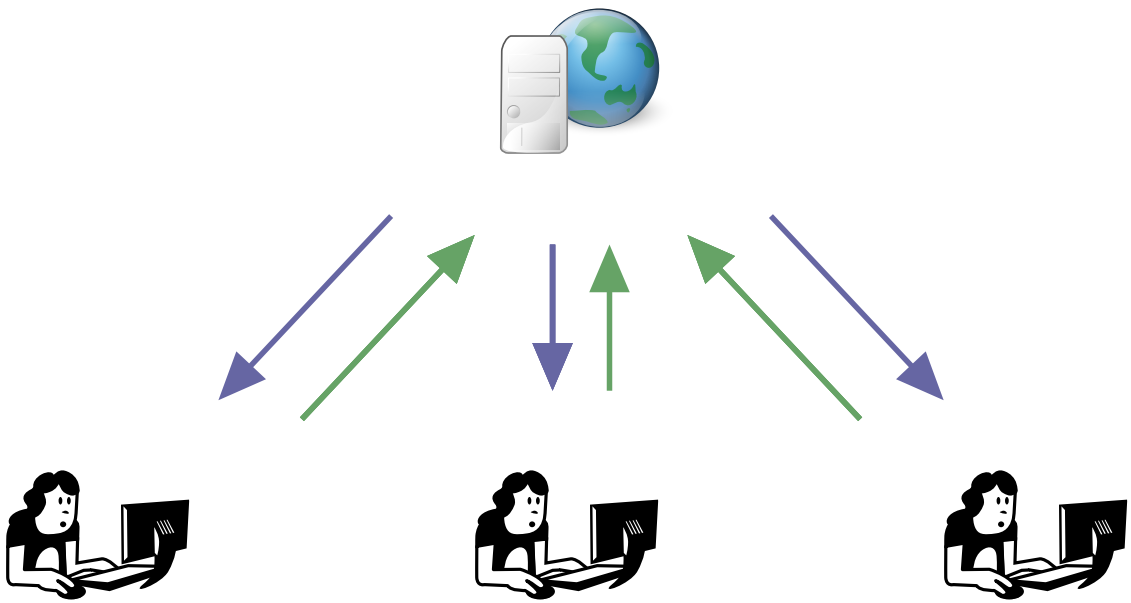
Modèle avec dépôt centralisé

"Centralized Workflow."



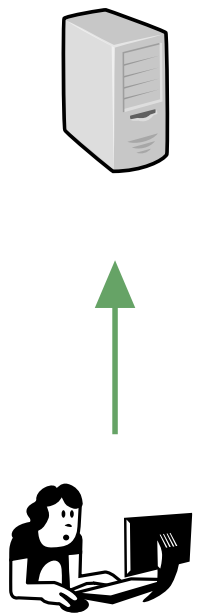
Modèle avec dépôt centralisé

"Centralized Workflow."



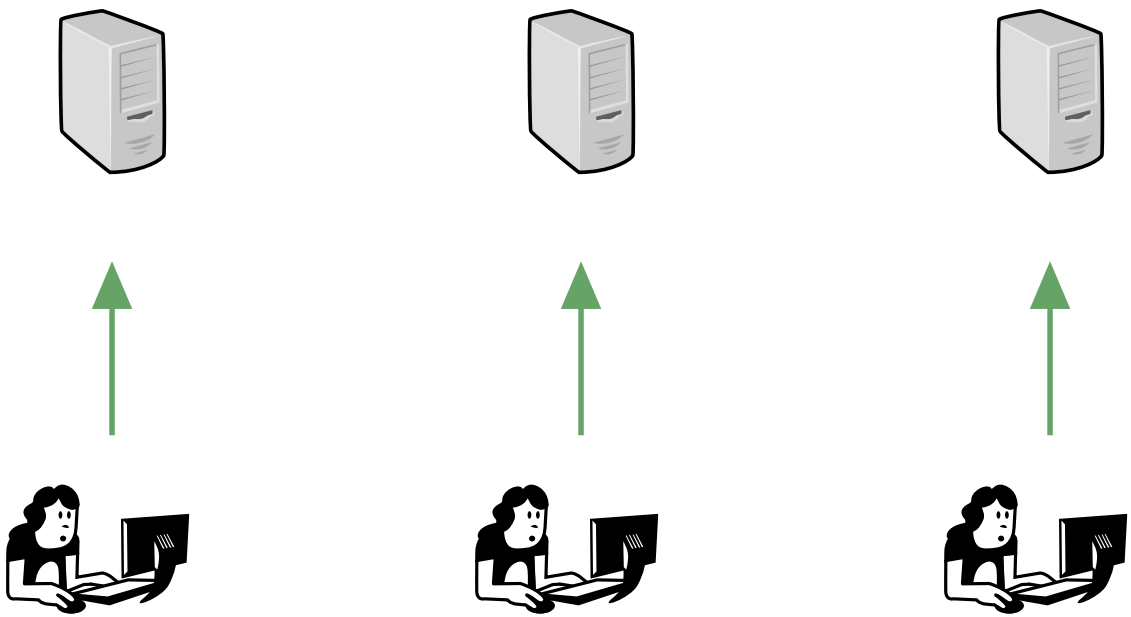
Modèle décentralisé avec dépôts publics.

"Cooperative and Decentralized Workflow."



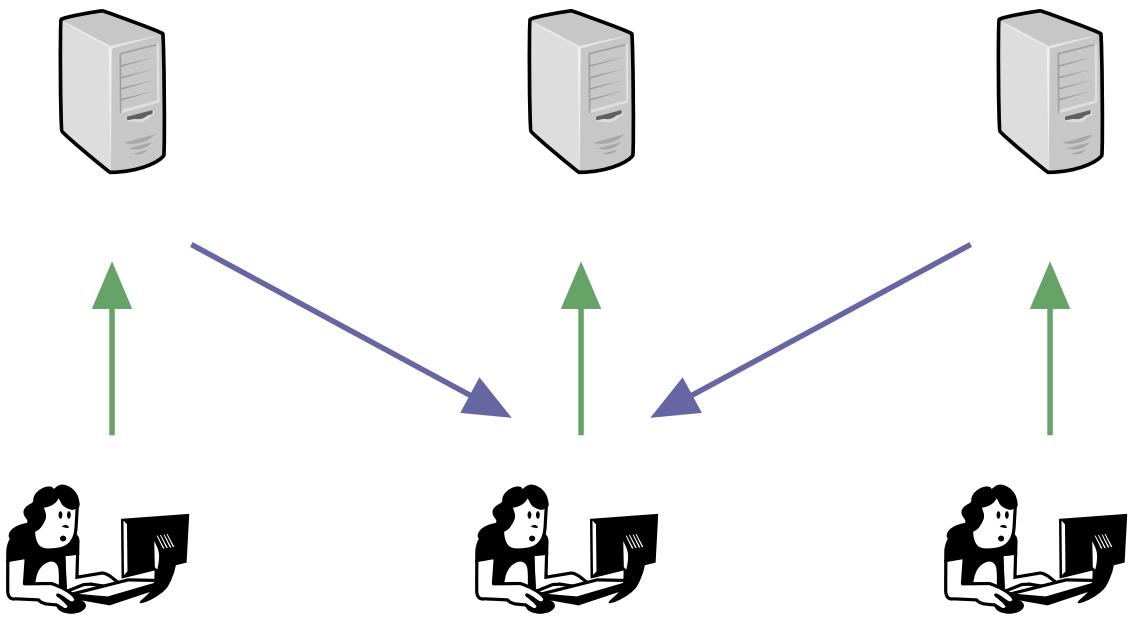
Modèle décentralisé avec dépôts publics.

"Cooperative and Decentralized Workflow."



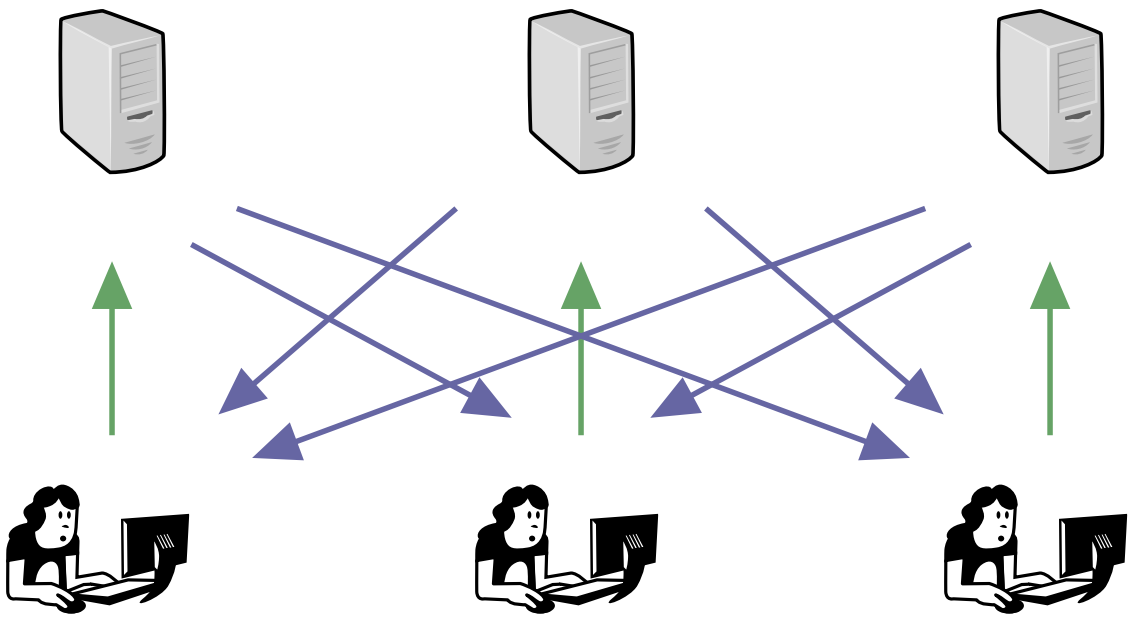
Modèle décentralisé avec dépôts publics.

"Cooperative and Decentralized Workflow."



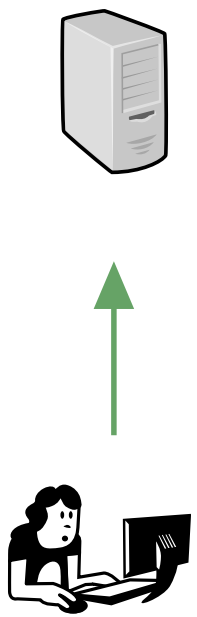
Modèle décentralisé avec dépôts publics.

"Cooperative and Decentralized Workflow."



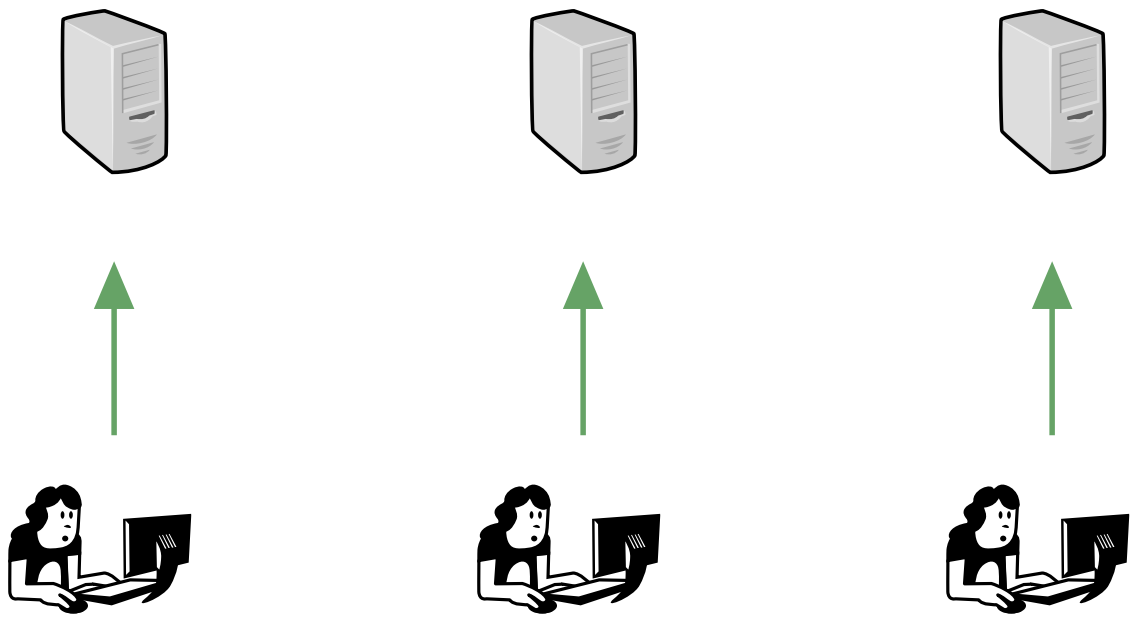
Modèle avec manager de dépôt

"Integration-Manager Workflow."



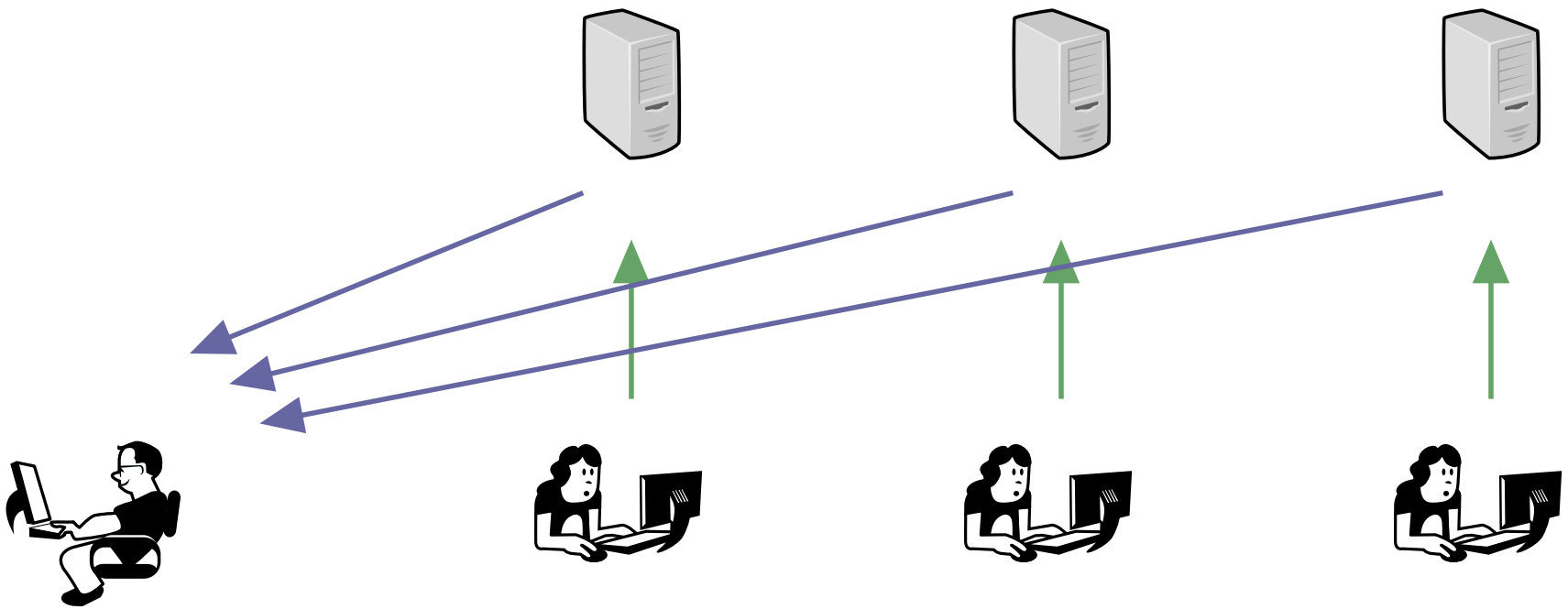
Modèle avec manager de dépôt

"Integration-Manager Workflow."



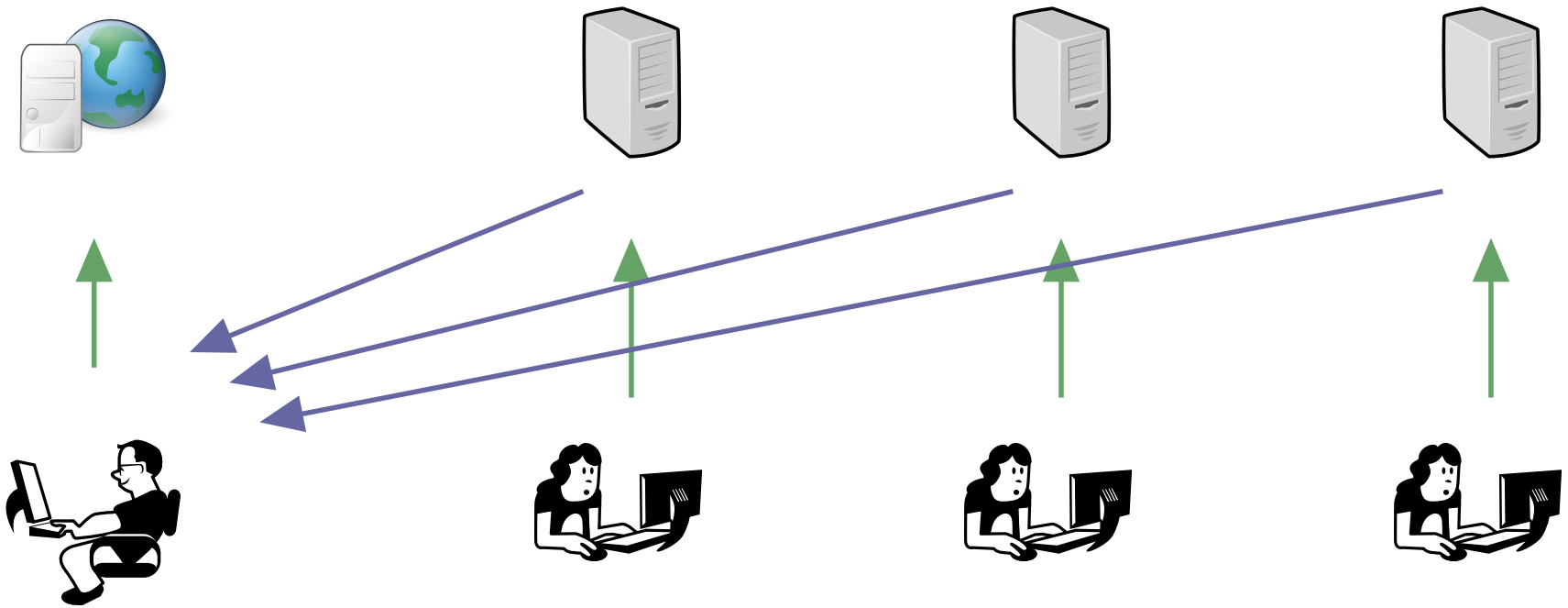
Modèle avec manager de dépôt

"Integration-Manager Workflow."



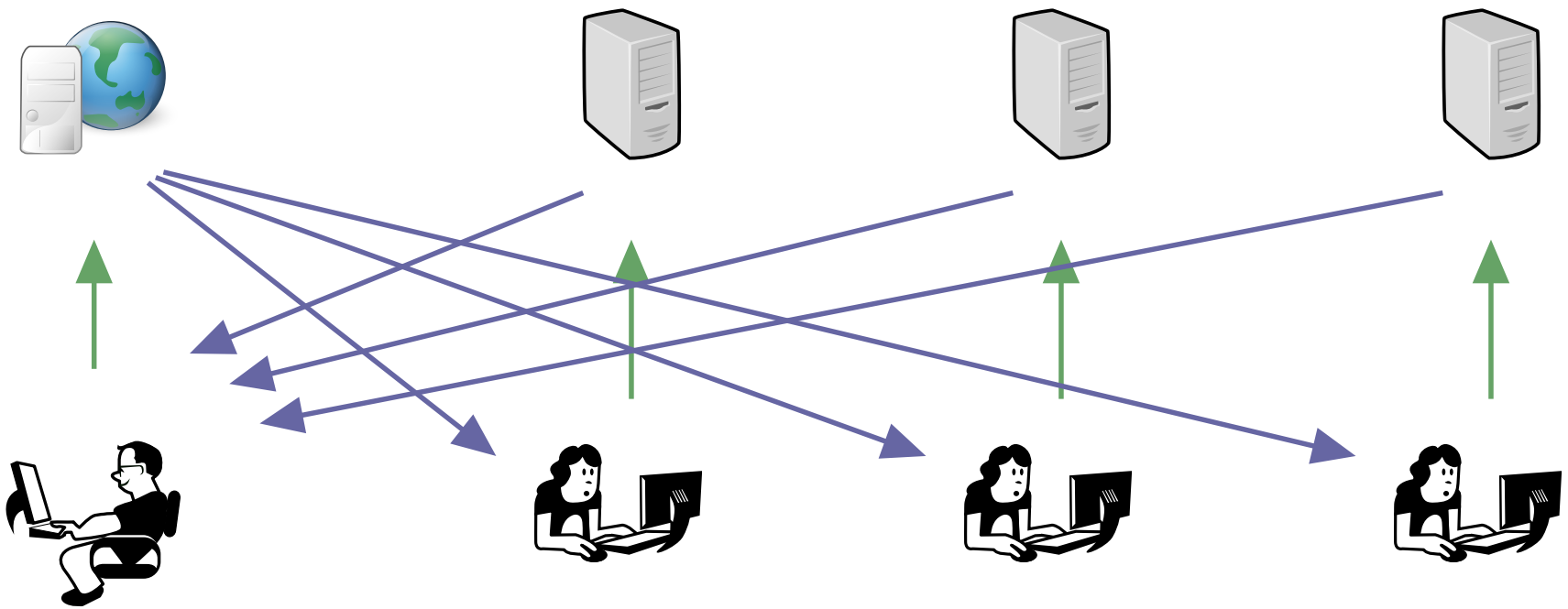
Modèle avec manager de dépôt

"Integration-Manager Workflow."



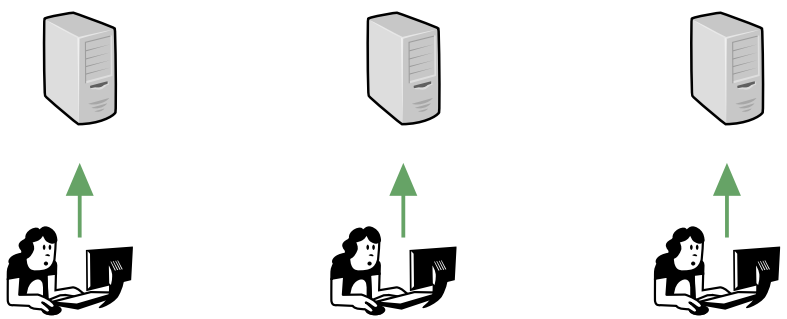
Modèle avec manager de dépôt

"Integration-Manager Workflow."



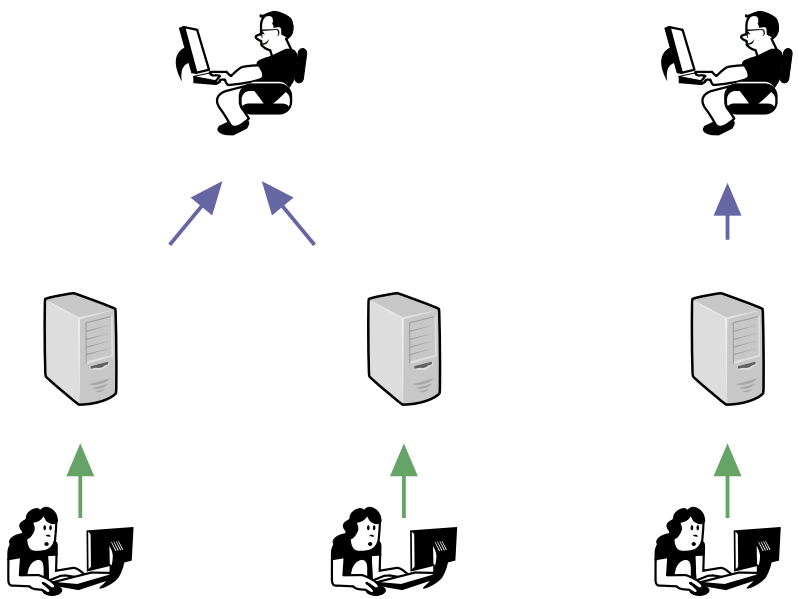
Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



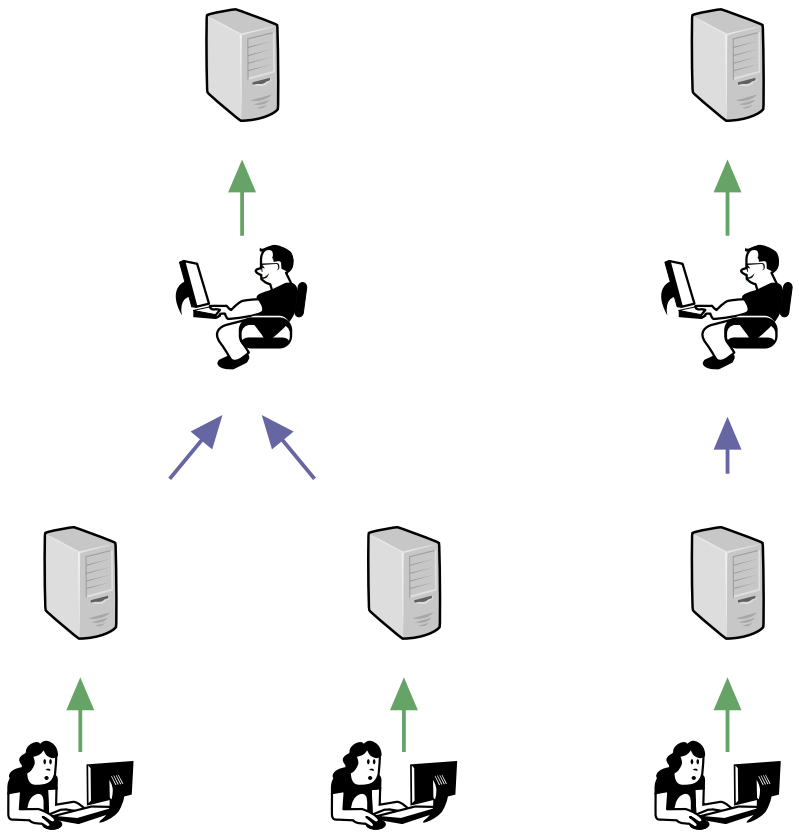
Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



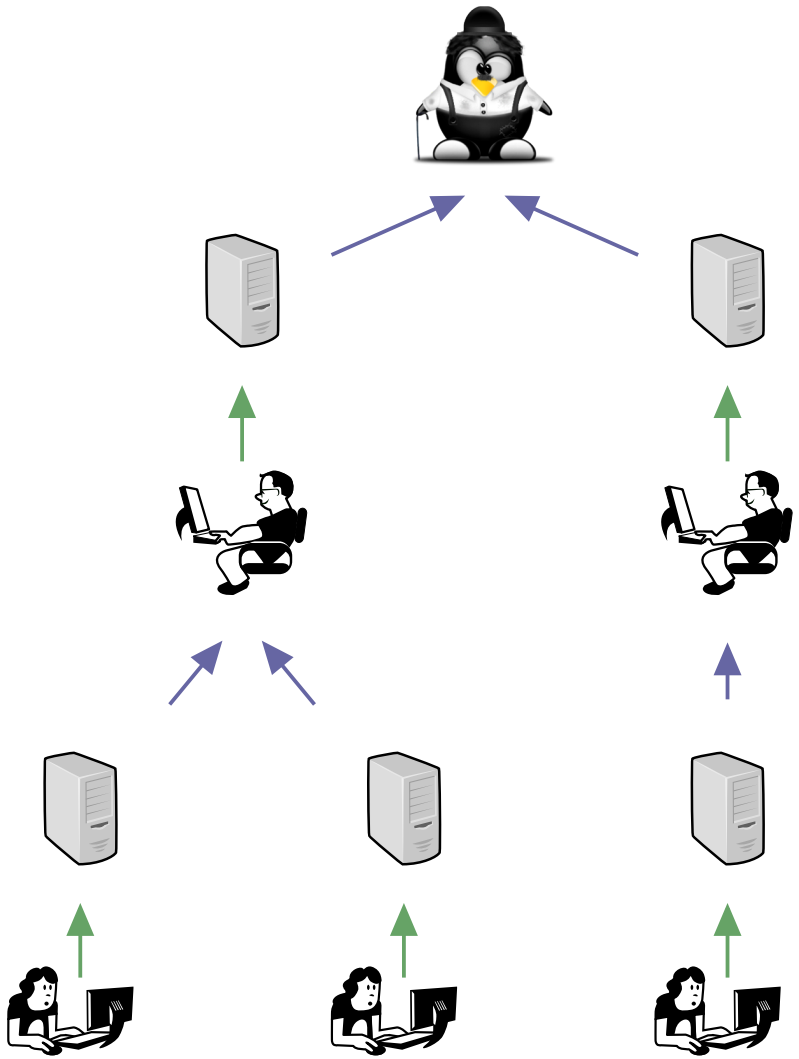
Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



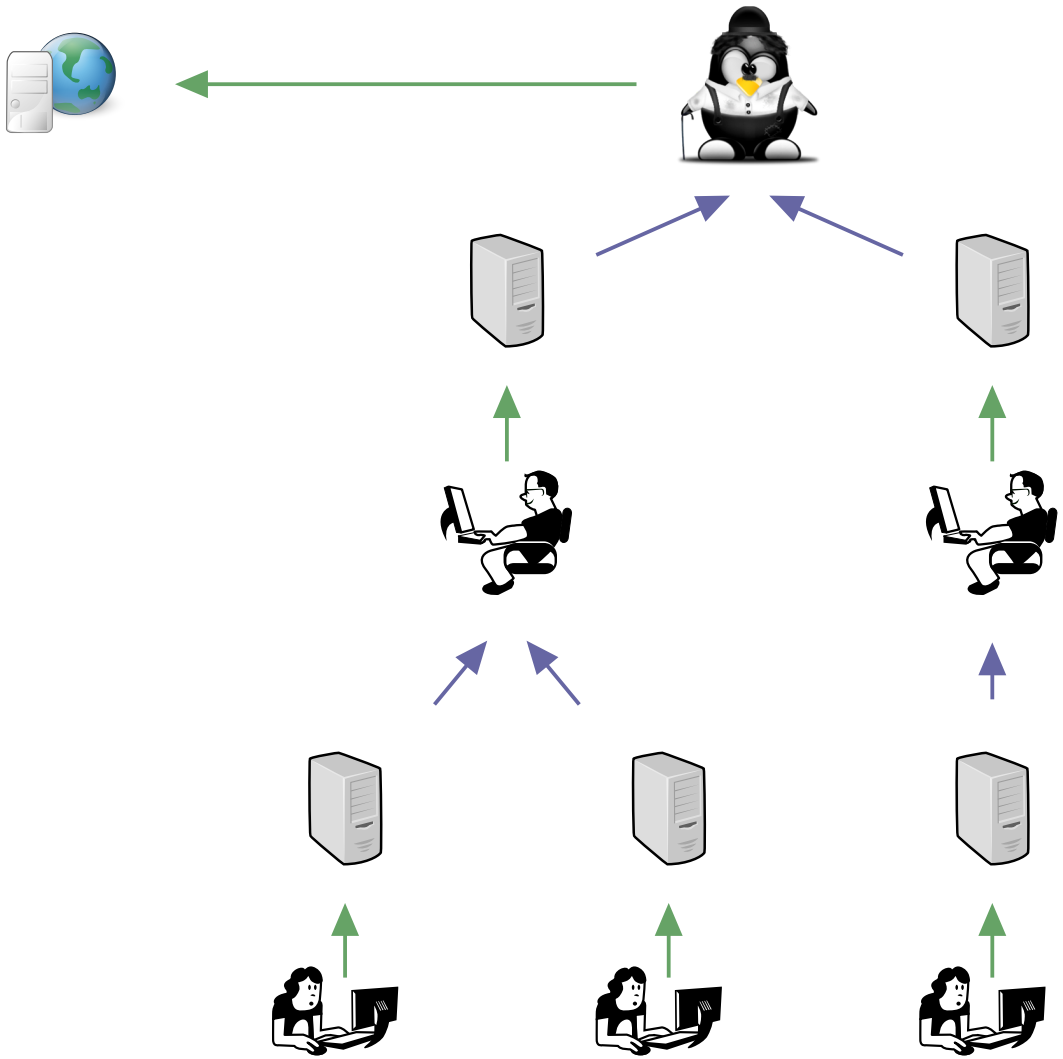
Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



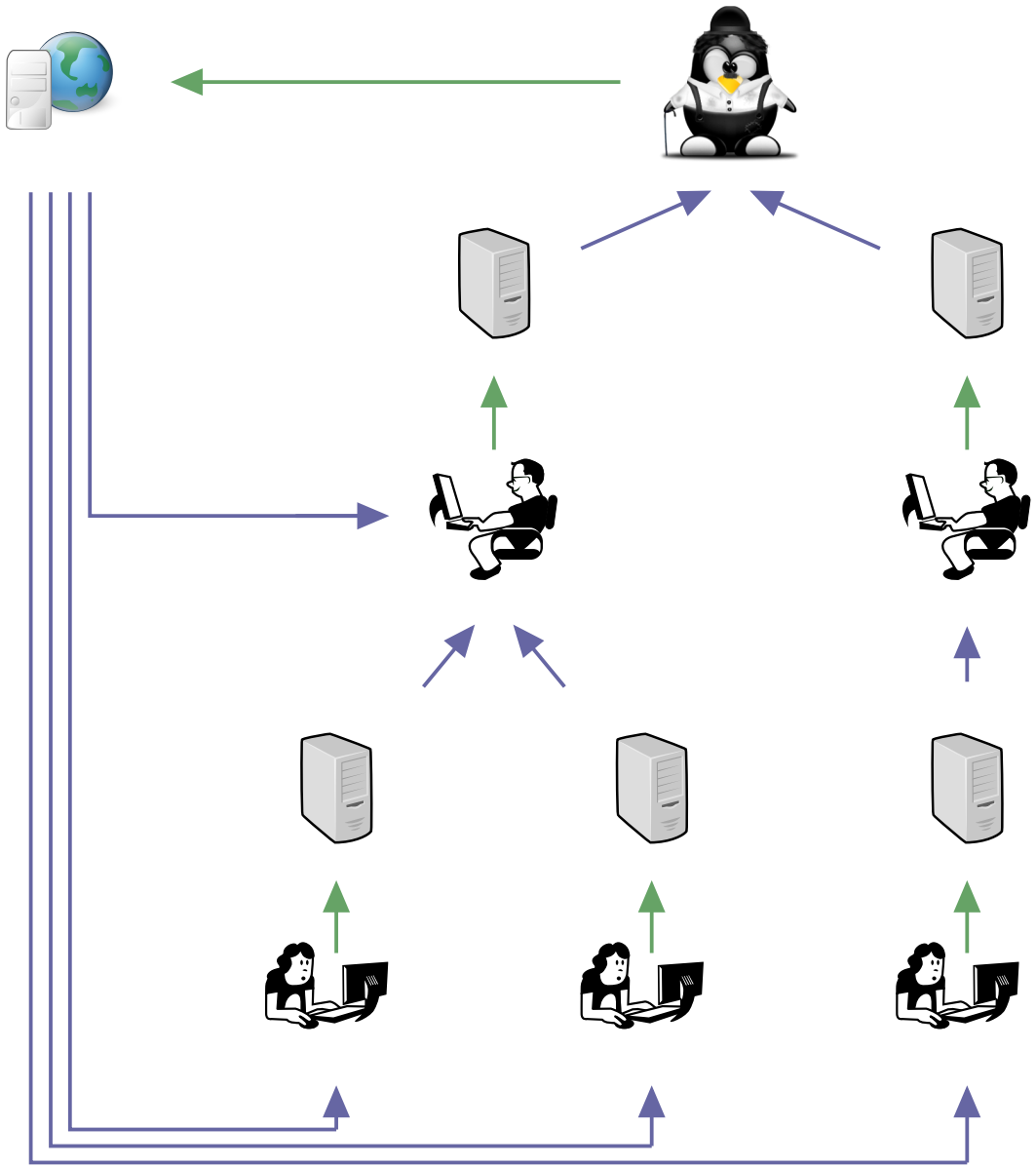
Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



Modèle avec dictateur et lieutenants.

"Dictator and Lieutenants Workflow."



Outline

Outline

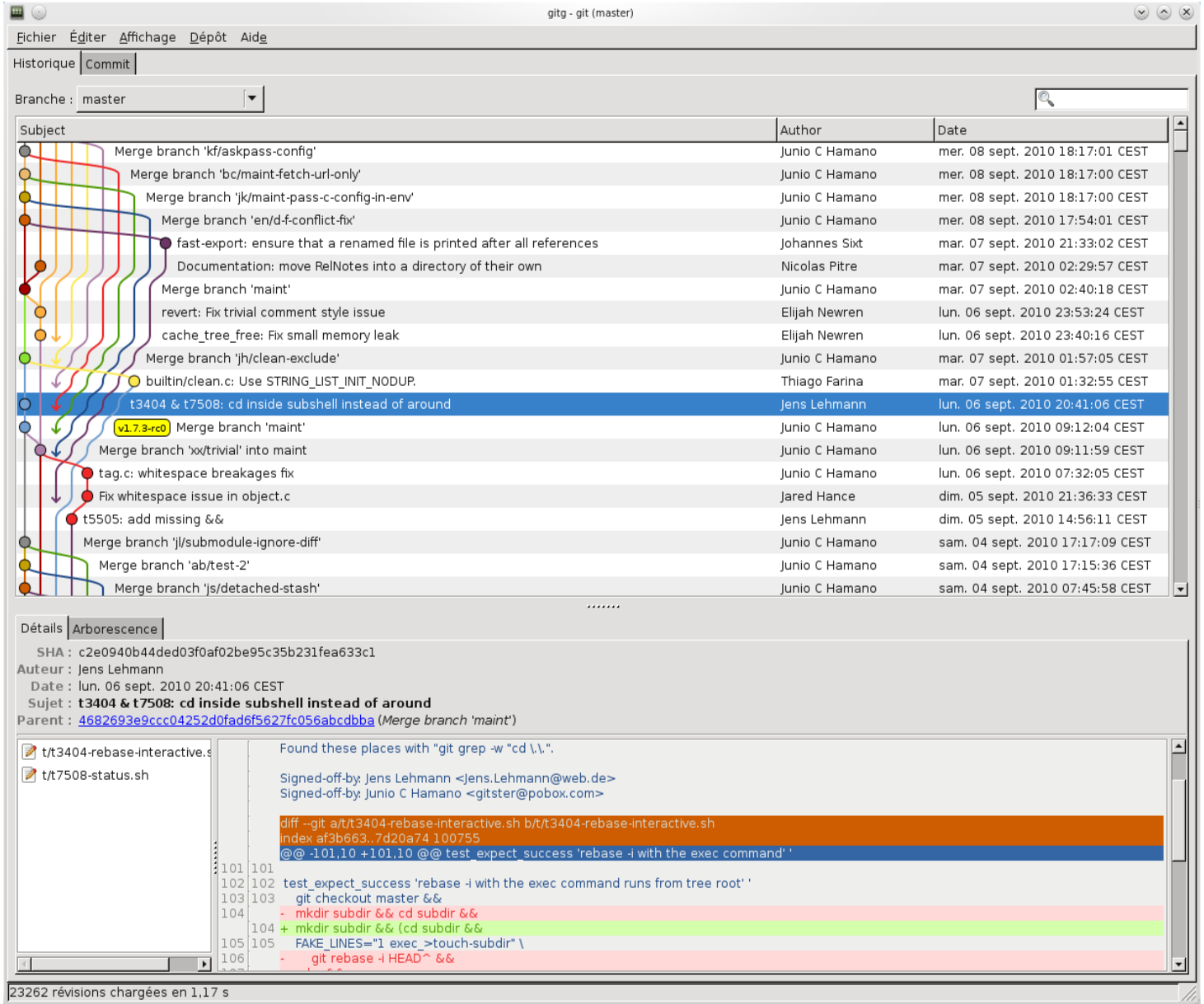
Les interfaces graphiques

Il existe de nombreuses interfaces graphiques permettant de gérer vos projets quelle que soit votre plate-forme de développement :

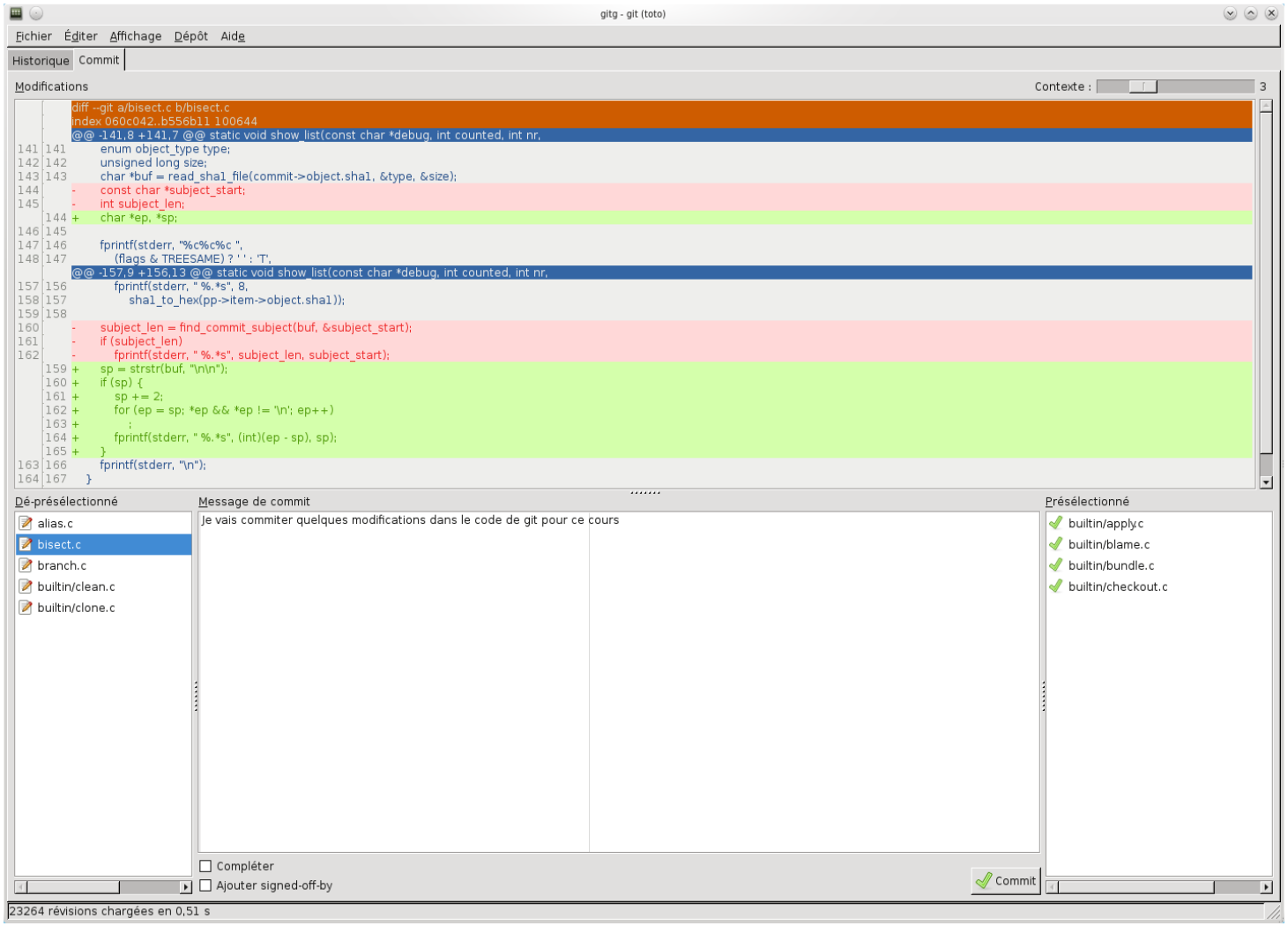
- **Avec git :**
  - **gitk** : l'interface de visualisation détaillée et graphique d'un historique git
  - **git-gui** : outil permettant de construire les commits
- **Linux :** gitg, Giggle, ...
- **Windows :** TortoiseGit, GitExtensions, ...
- **Apple :** GitX, Gitti, ...
- **Eclipse :** EGit, ...

Outline

gitg : visualisation de l'historique



gitg : commit interactif



Outline

Git vs Rhinocéros

Mercurial, Darcs, Bazaar, Arch, etc

Les points forts :

+ Certainement le plus rapide à appliquer des patchs
+ Simple à mettre en place (mode sans serveur)
+ Petits outils puissants (esprit Unix)
+ Git est distribué sous licence GNU GPL 2
+ Développement actif
+ Linus

Les points faibles :

- Courbe d'apprentissage
- Linus

Bibliographie sur le web

- Documentation officielle et usuelle : `http://www.kernel.org/pub/software/scm/git/docs/everyday.html`
- Documentation officielle complète : `http://www.kernel.org/pub/software/scm/git/docs/`
- Manuel de référence communautaire et en français : `http://alexgirard.com/git-book/index.html`
- Manuel de référence communautaire (ultra complet) et en anglais : `http://progit.org/book/`
- Tutoriel en video et en anglais : `http://www.gitcasts.com/`
- Linus Torvalds parle de git chez *Google* (1 heure environ) : `http://www.youtube.com/watch?v=4XpnKHJAok8`
- N'oubliez pas les *manpages*...

Références

- http://download.ikaaro.org/doc/git/200607-ols.pdf
- http://kernel.org/pub/software/scm/cogito/
- http://git.or.cz/
- http://git.or.cz/gitwiki/
- http://www.kernel.org/pub/software/scm/git/docs/everyday.html
- http://www.kernel.org/pub/software/scm/git/docs/