Institut d'Informatique Claude Chappe
Le Mans Université

Licence Sciences, Technologies, Santé
L2- Mention Informatique

OUTILS DE PROGRAMMATION

Gestionnaire de Version – Git (Très inspiré de ProGit)

D'après le diaporama de V. Jousse

Claudine Piau-Toffolon
claudine.piau-toffolon@univ-lemans.fr

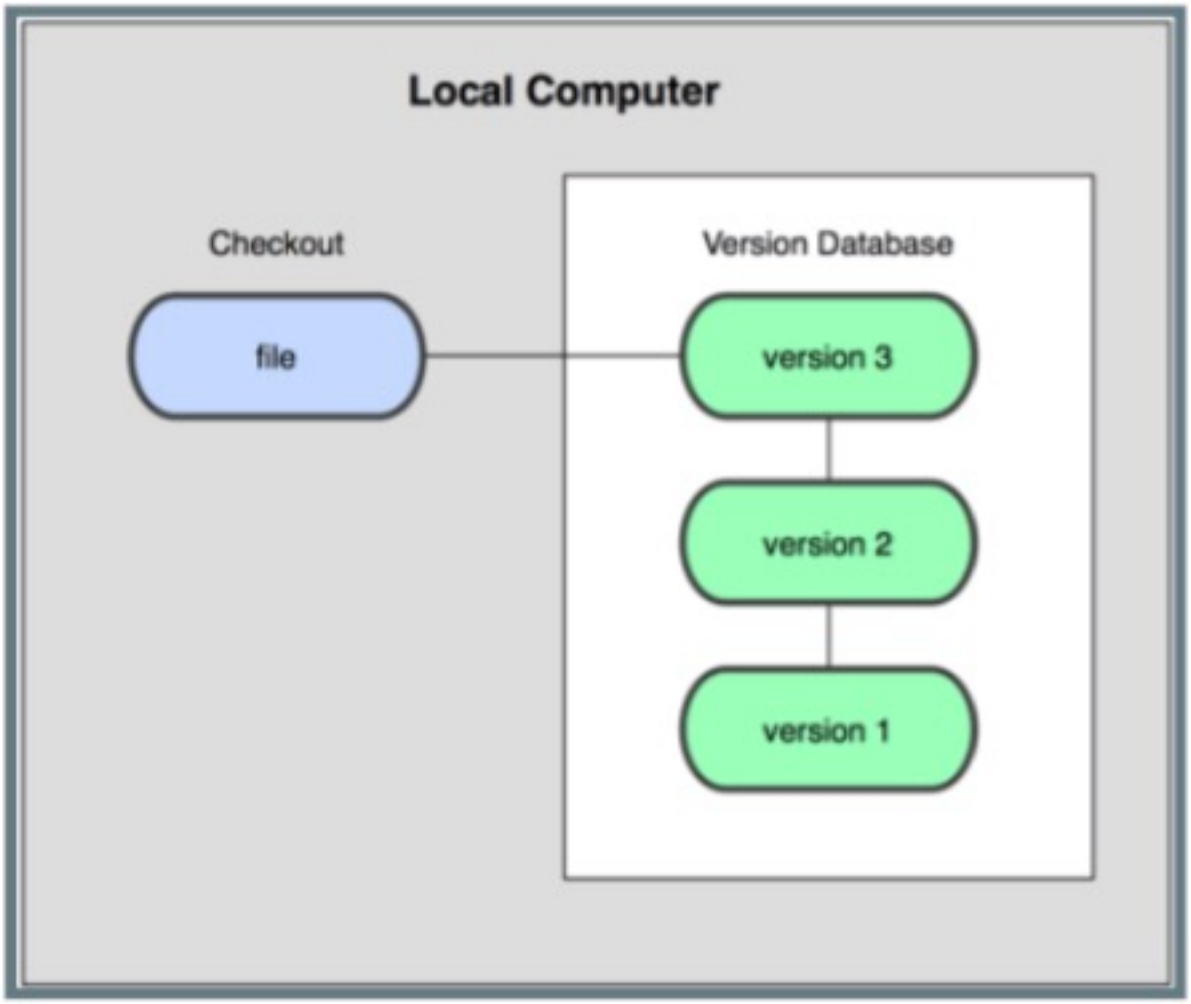
Système de Gestion de version
Version Controm System (VCS)

« *Un gestionnaire de version est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce que l'on puisse rappeler une version antérieure d'un fichier à tout moment* »

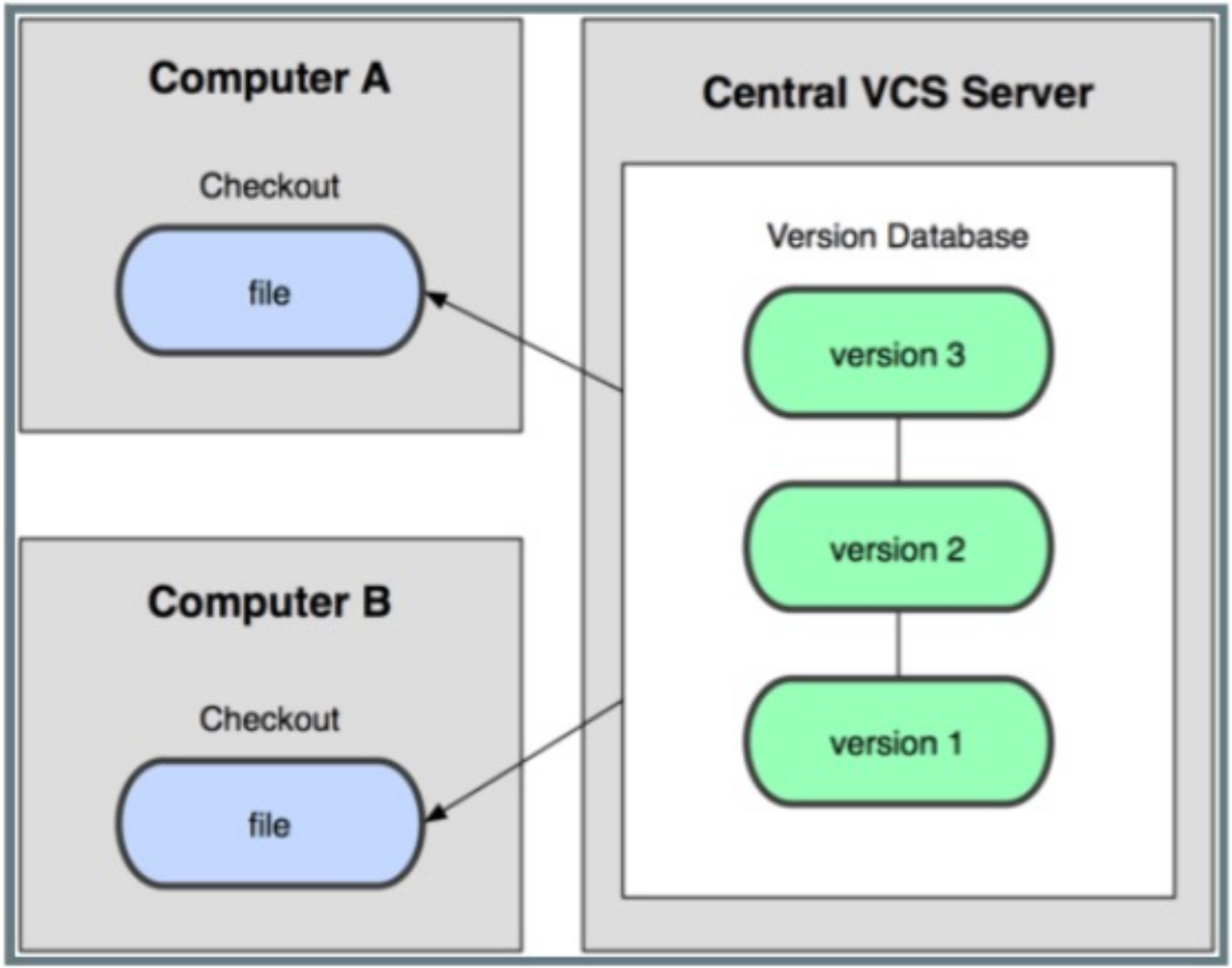
3 types de système

- Les systèmes locaux :RCS, copies locales, …
- Les systèmes centralisés (CVCS): CVS, SVN, Perforce,...
- Les systèmes distribués (DVCS): Git, Mercurial, Bazaar, Darcs, …

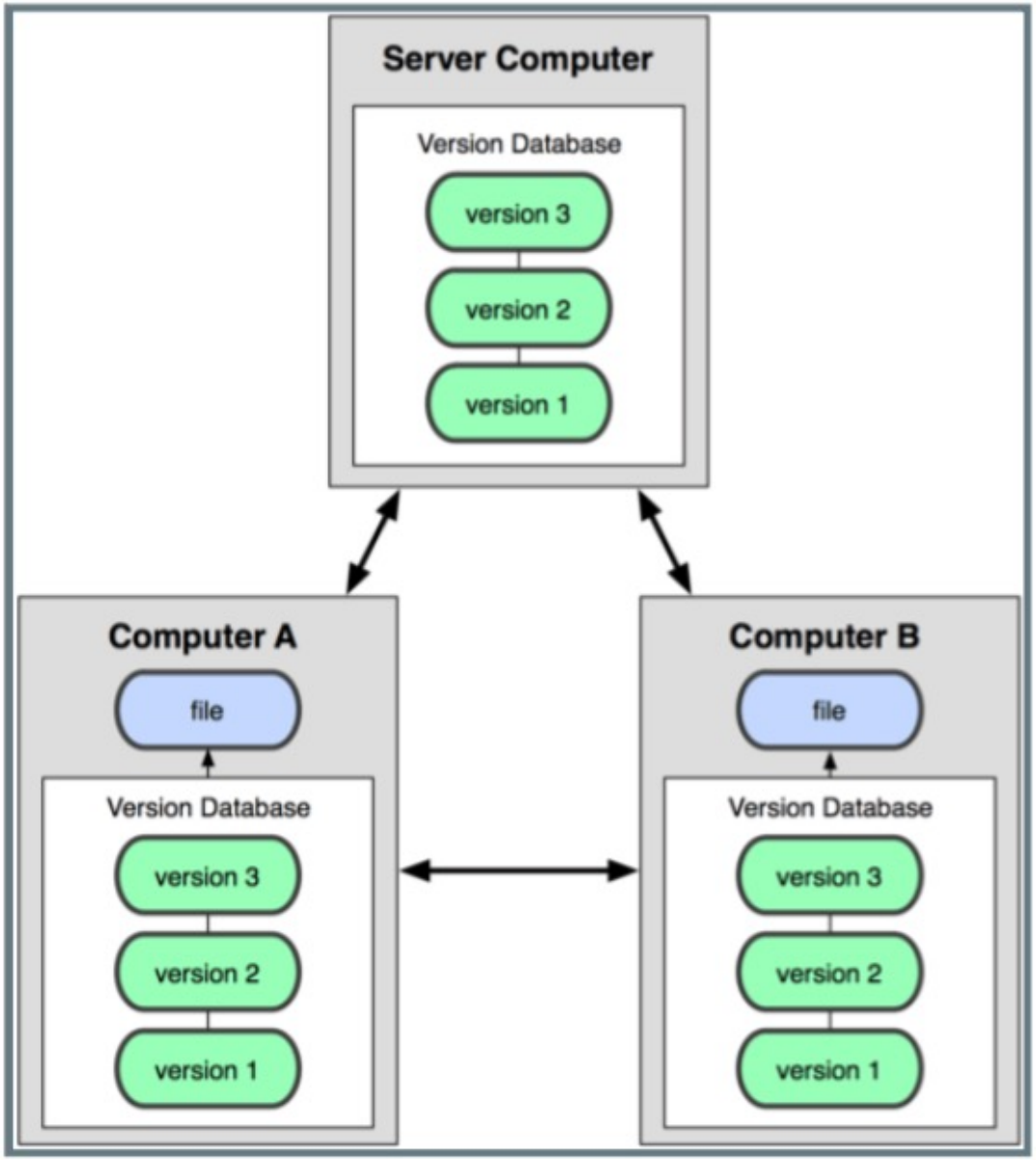
Systèmes locaux



Systèmes centralisés (CVCS)



Systèmes distribués (DVCS)

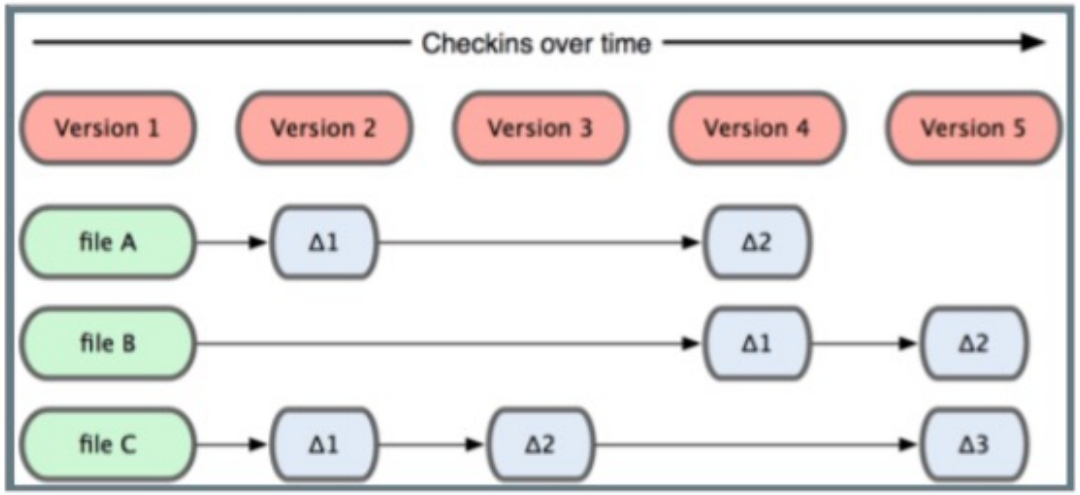


GIT

- Historique
  - Création en 2005
  - Développé par l'équipe de Linus Torvald pour le noyau Linux
- Objectifs
  - Vitesse
  - Conception simple
  - Support pour les développements non linéaires (milliers de branches parallèles)
  - complètement distribué
  - Capacité à gérer efficacement des projets d'envergure tels que le noyau Linux

Bases de GIT

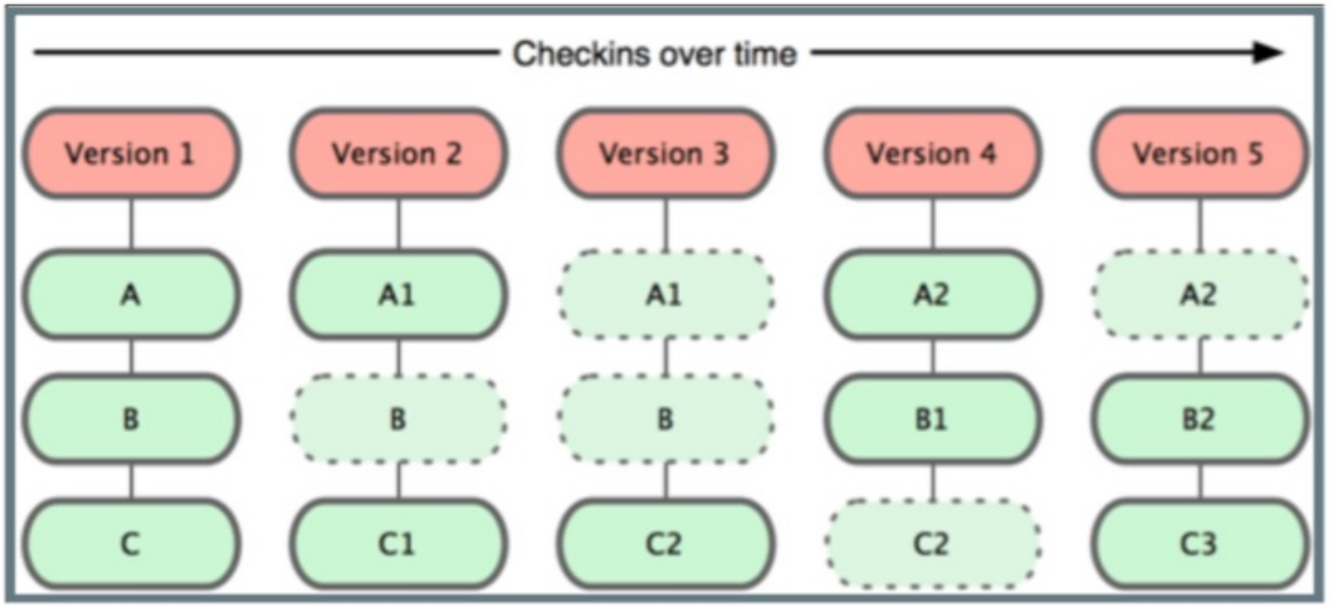
Git Gère des instantanés et non des différences



Historiquement

<--------

Git →



Bases de Git

- La majorité des opérations sont locales
- Pas de latence réseau
- Accès immédiats et rapides
- Possibilité de travailler déconnecté
- Git gère l'intégrité
- Généralement Git ne fait qu'ajouter des données
- Git calcule des sommes de contrôle = empreinte SHA-1

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

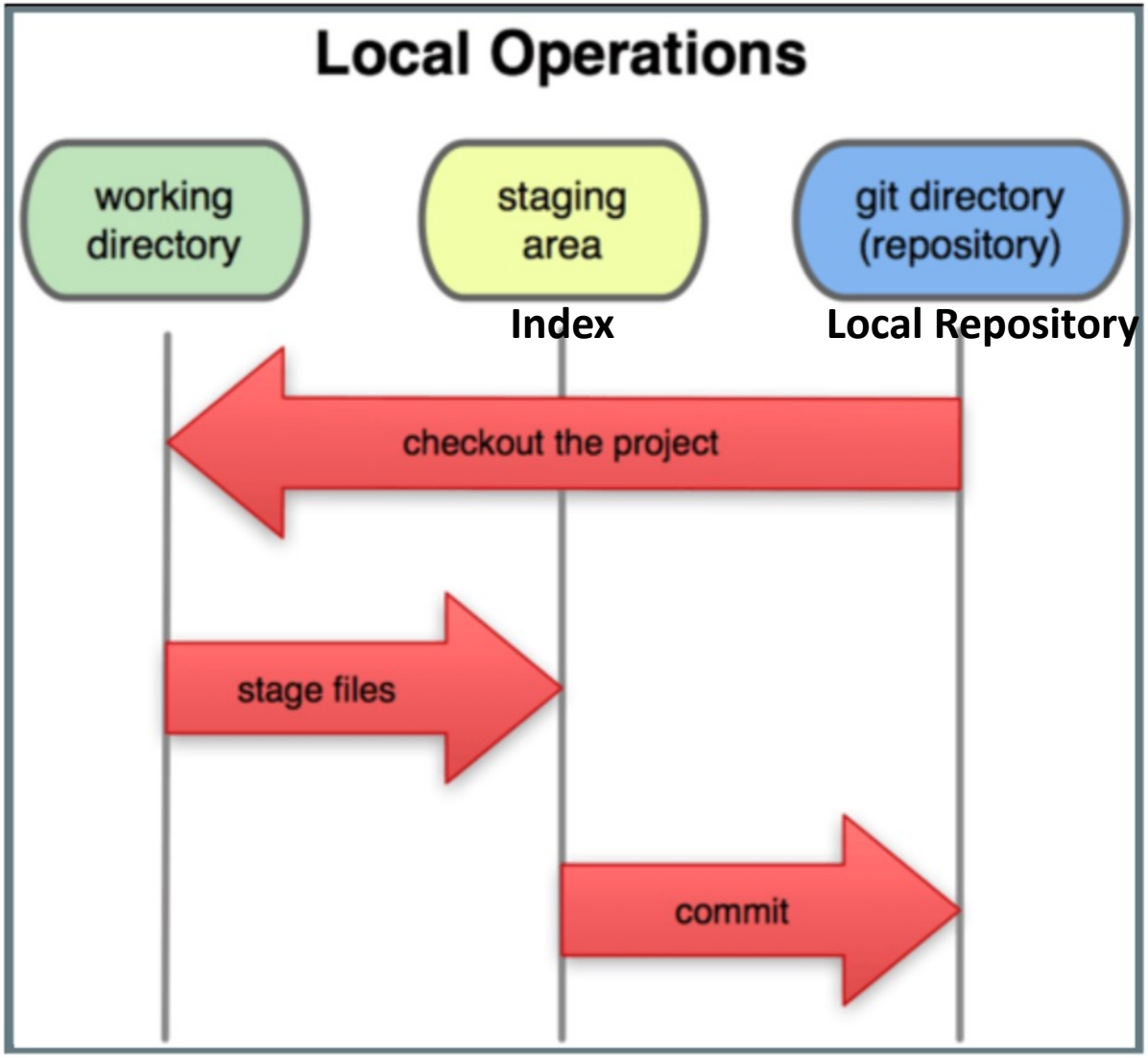
Vous les verrez partout !!

Bases de Git: Les 3 états

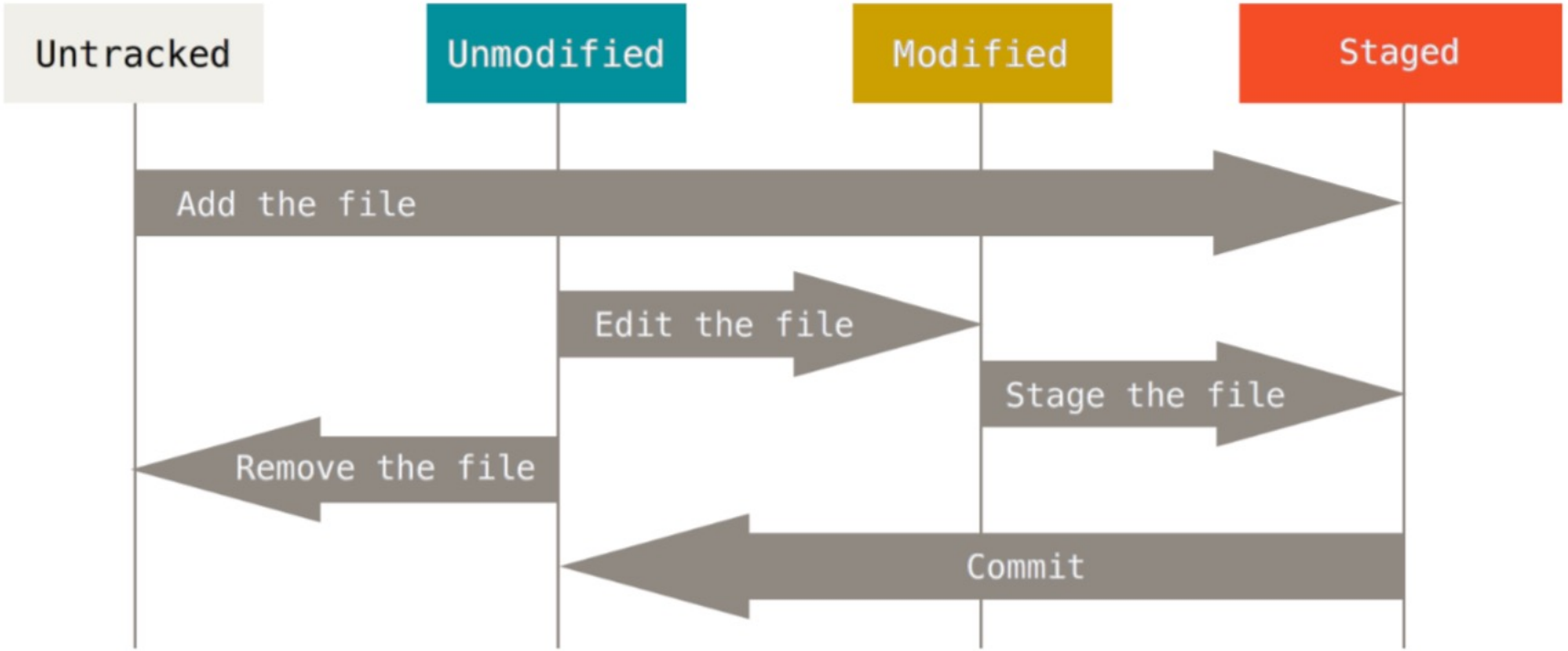
Git gère trois états pour vos fichiers:

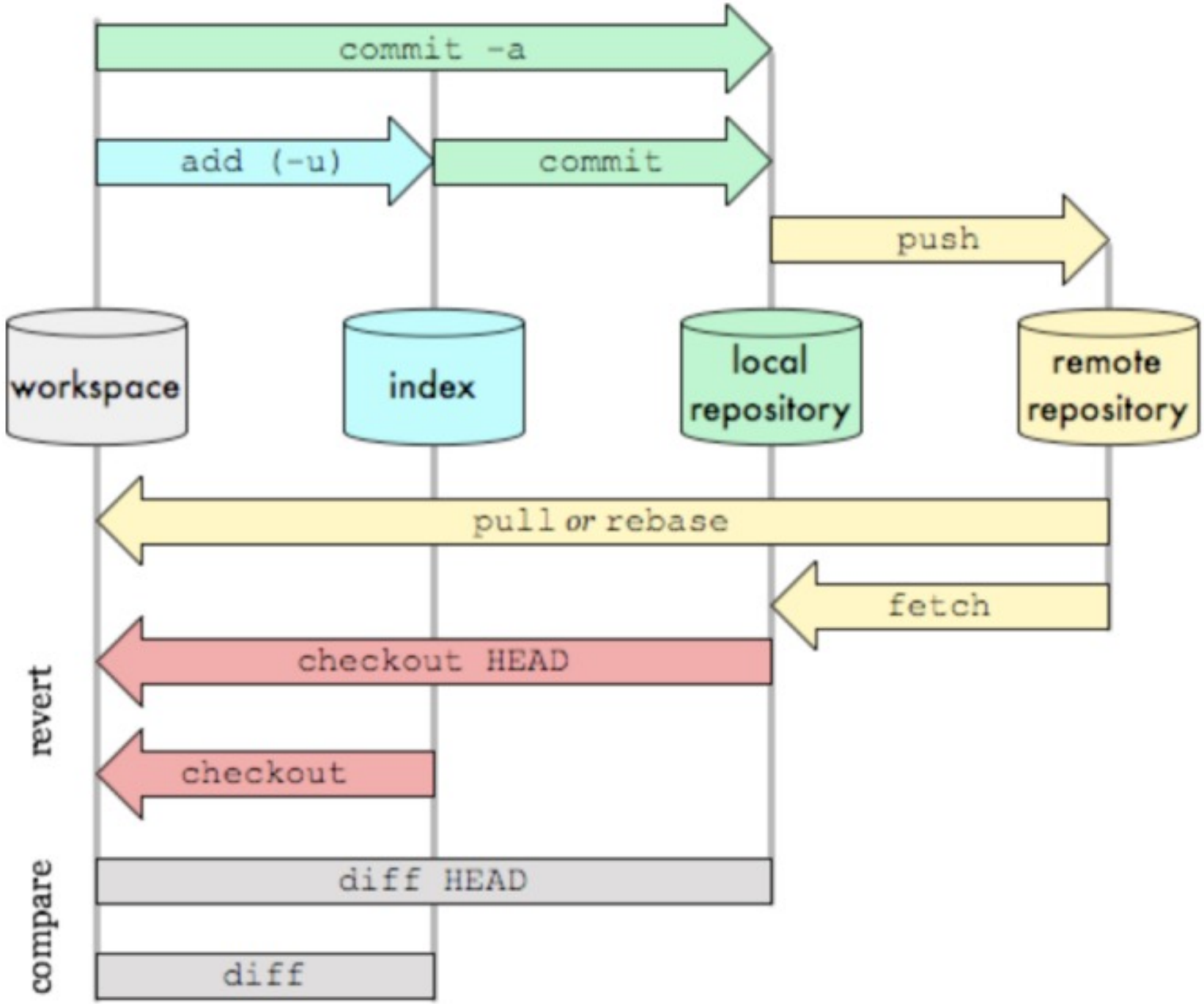
- **Validé**
  - Les données sont stockées en sécurité dans votre base de données locale
- **Modifié**
  - Vous avez modifié le fichier mais qu'il n'a pas encore été validé en base
- **Indexé**
  - Vous avez marqué un fichier modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet

Répertoire de travail, Zone d'index et Répertoire Git



Les commandes de base
File Status Lifecycle




commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
revert
checkout
compare
diff HEAD
diff

Utilisation classique

1. ***Créer / Modifier*** des fichiers dans son répertoire de travail
2. ***Indexer*** les fichiers modifiés: ce qui ajoute des instantanés de ces fichiers dans la zone d'index
3. ***Valider***: ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la BD du répertoire Git

Concrètement …

Initialiser un dépôt Git

→ *Pour placer un répertoire existant sous Git :*

```
$ cd monrepertoire
$ git init
```

= Création d'un sous –répertoire "caché" nommé .git

Commencer à suivre / Indexer tous les fichiers .c

```
$ git add *.c
```

Vérifier l'état des fichiers

$ git status

Valider

```
$ git commit –m 'version initiale du projet'
```

```
anoat-1:ProgC piau$ cd monGit1
anoat-1:monGit1 piau$ git init
Initialized empty Git repository in /Users/piau/Documents/ProgC/monGit1/.git/
anoat-1:monGit1 piau$ git add *.c
anoat-1:monGit1 piau$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hello.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        LISEZMOI.md
```

```
anoat-1:monGit1 piau$ git commit -m 'Ajout de mes premiers fichiers .c'
[master (root-commit) a7698e2] Ajout de mes premiers fichiers .c
 1 file changed, 18 insertions(+)
 create mode 100644 hello.c
anoat-1:monGit1 piau$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        LISEZMOI.md

nothing added to commit but untracked files present (use "git add" to track)
```

Placer un nouveau fichier sous suivi de version

On ajoute le fichier

```
$ git add LISEZMOI
```

On vérifie le statut

```
 piau:~/Documents/E/projetgit$ git add LISEZMOI
 piau:~/Documents/E/projetgit$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   LISEZMOI
```

Indexer des fichiers modifiés

EXERCICE

Modifiez un fichier sous suivi de version
Créez un nouveau fichier et l'éditer
Ajoutez -le avec Git

```
$  git status
```

Observer le message: Que s'est-il passé?

Ignorer des fichiers

```
$ cat .gitignore
```

```
piau:~/Documents/E/projetgit$ nano .gitignore
piau:~/Documents/E/projetgit$ cat .gitignore
*.[oa]
*.~
# ignorer uniquement le fichier TODO à la racine du projet
/TODO
# ignorer tous les fichiers dans le répertoire build
build/
# ignorer doc/notes.txt, mais pas doc/server/arch.txt
doc/*.txt
# ignorer tous les fichiers .txt sous le répertoire doc/
doc/**/*.txt
```

Exemples de fichiers .gitignore: https://github.com/github/gitignore

Valider vos modifications

```
$ git commit
```

Supprimer des fichiers

```
$ rm monfichier.c
```

Pour voir ce qui s'est passé:

```
$ git status
```

Arrêter de suivre un fichier

```
$ git rm --cached LISEZMOI.txt
```

Visualiser les différences

Visualiser les modifications non indexées

```
$ git diff
```

```
piau:~/ProgC/cours/Git$ git diff
diff --git a/README.txt b/README.txt
deleted file mode 100644
index e69de29..0000000
```

Visualiser les différences entre les fichiers indexés et le dernier instantané

```
$ git diff --cached
```

```
piau:~/ProgC/cours/Git$ git diff --cached
diff --git a/LISEZMOI b/LISEZMOI
deleted file mode 100644
index e69de29..0000000
diff --git a/README b/README
deleted file mode 100644
index e69de29..0000000
```

Visualiser l’historique

```
$ git log


$git log –p -2


$git log --since=2.weeks
```

Cloner un dépôt existant

```
$ git clone git@github.com:cpiau72/conduite-projet.git
```

→ Création d'un répertoire `conduite-projet`

```
$ git clone git@github.com:cpiau72/conduite-projet.git  autre nom
```

→ Création d'un répertoire `autrenom`

Travailler avec des dépôts existants

```
$ git clone git@github.com:cpiau72/conduite-projet.git CP

$ cd CP
#Se positionner sur ce dépôt:
$ git remote


#pour lister les dépôts exsistants (et vérifier où on est)
$ git remote -v
```

```
claudine@MacBook CP % git remote
origin
claudine@MacBook CP % git remote -v
origin git@github.com:cpiau72/conduite-projet.git (fetch)
origin git@github.com:cpiau72/conduite-projet.git (push)
```

Travailler avec des dépôts existants

`git remote`

#Ajouter un depot distant

```
$ git remote add nompointdebut adresse_depot
```

#Lister les dépôts existants

```
$ git remote –v
```

#Repérer où l'on se trouve

```
$ git remote
```

Récupérer depuis des dépôts existants

Récupération de code distant dans une nouvelle branche

$ git fetch **depart**

Récupérer depuis des dépôts existants

Récupération de code distant et fusion dans la branche courante

A partir du remote **depart**

```
$ git pull depart
```

A partir du remote par défaut (origin)

```
$ git pull
```

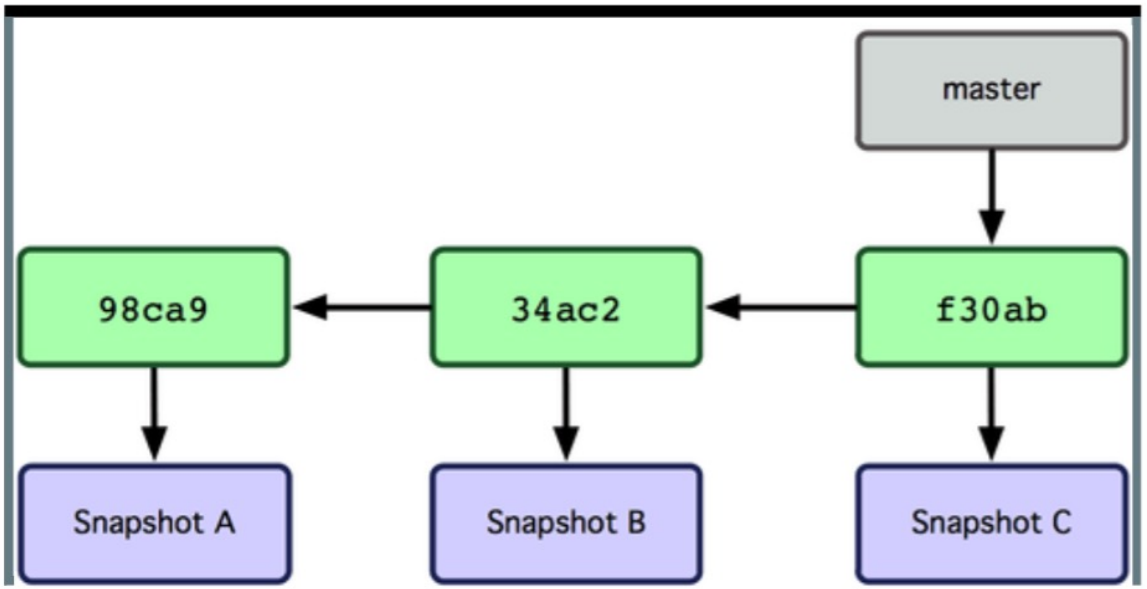
Pousser son travail sur un dépôt distant

`git push [nom-distant] [nom-de-branche]`

Exemple: **`$ git push origin master`**

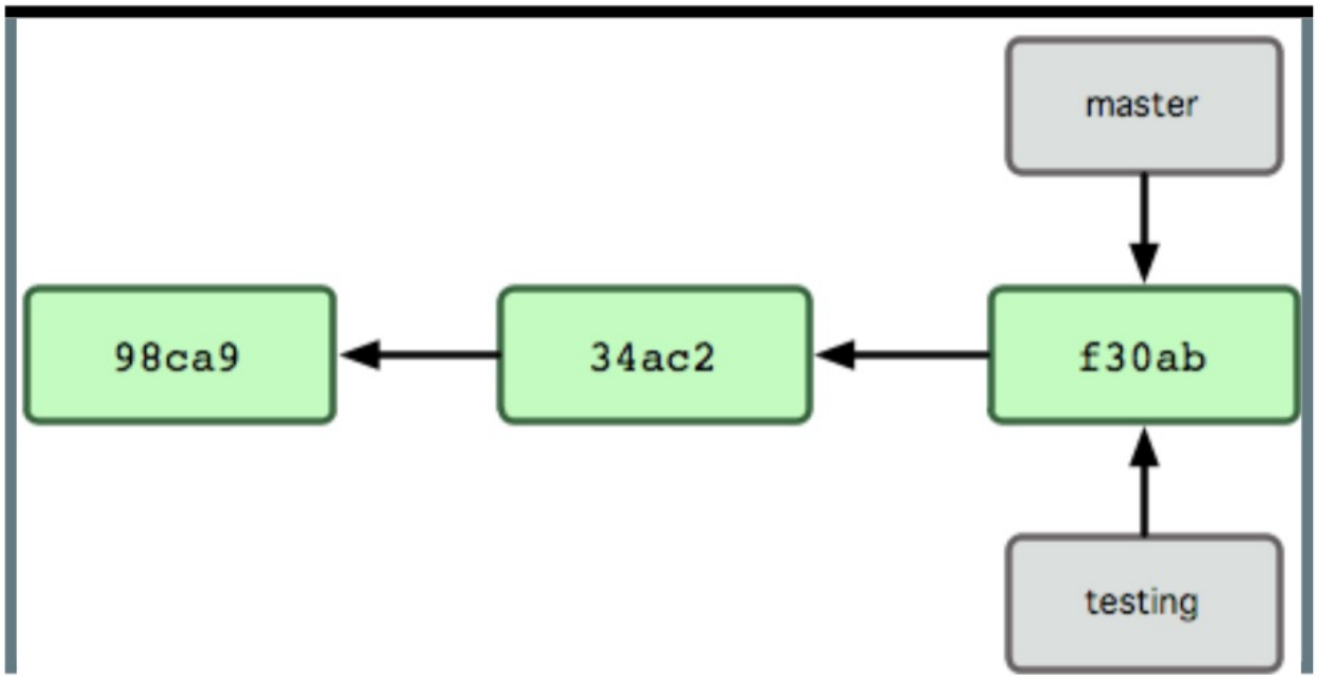
Les branches dans Git

- « Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans se préoccuper de cette ligne principale. »
- La branche par défaut de Git s'appelle **master**

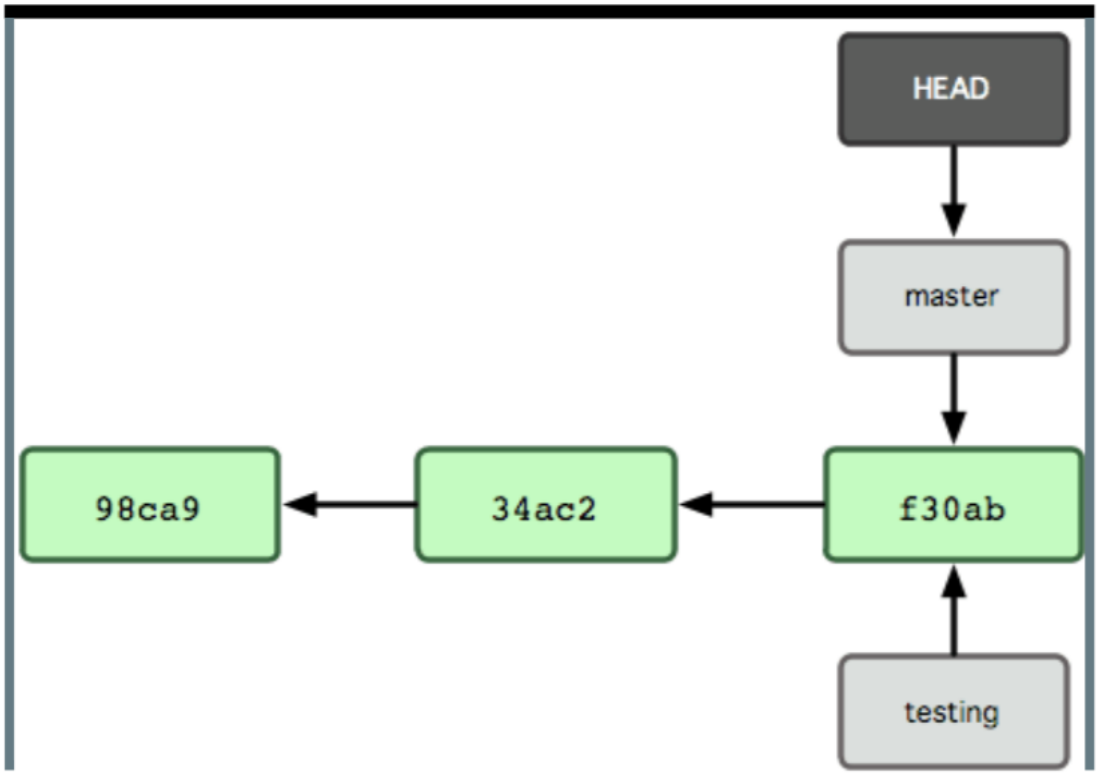


Création d'une nouvelle branche

```
$ git branch testing
```

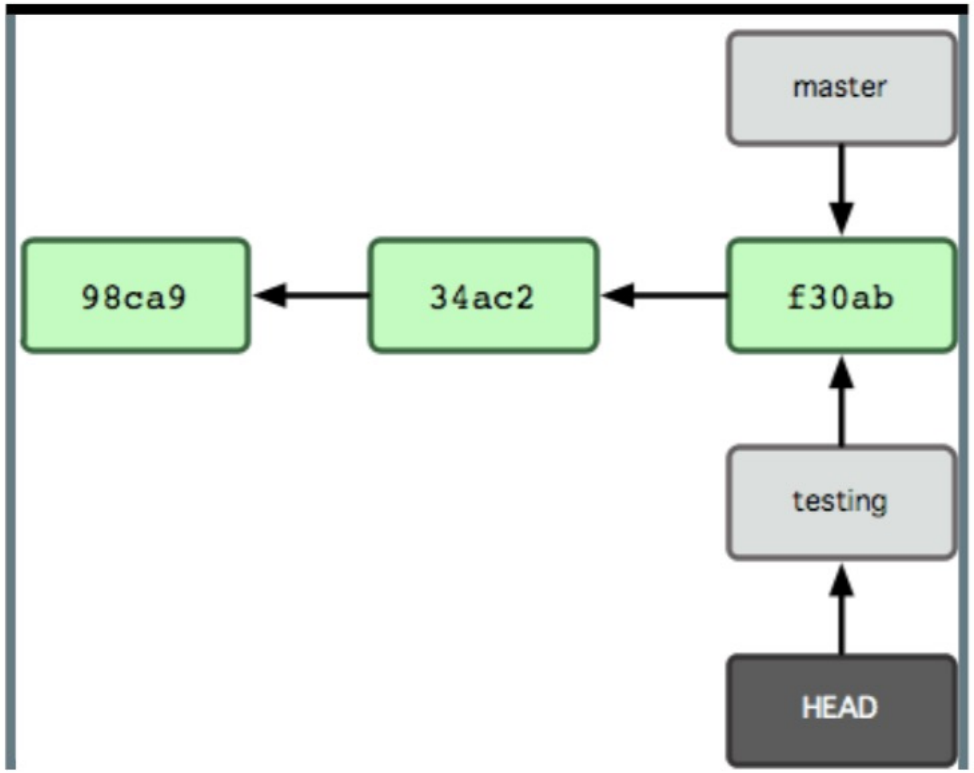


Le pointeur spécial HEAD vers la branche courante



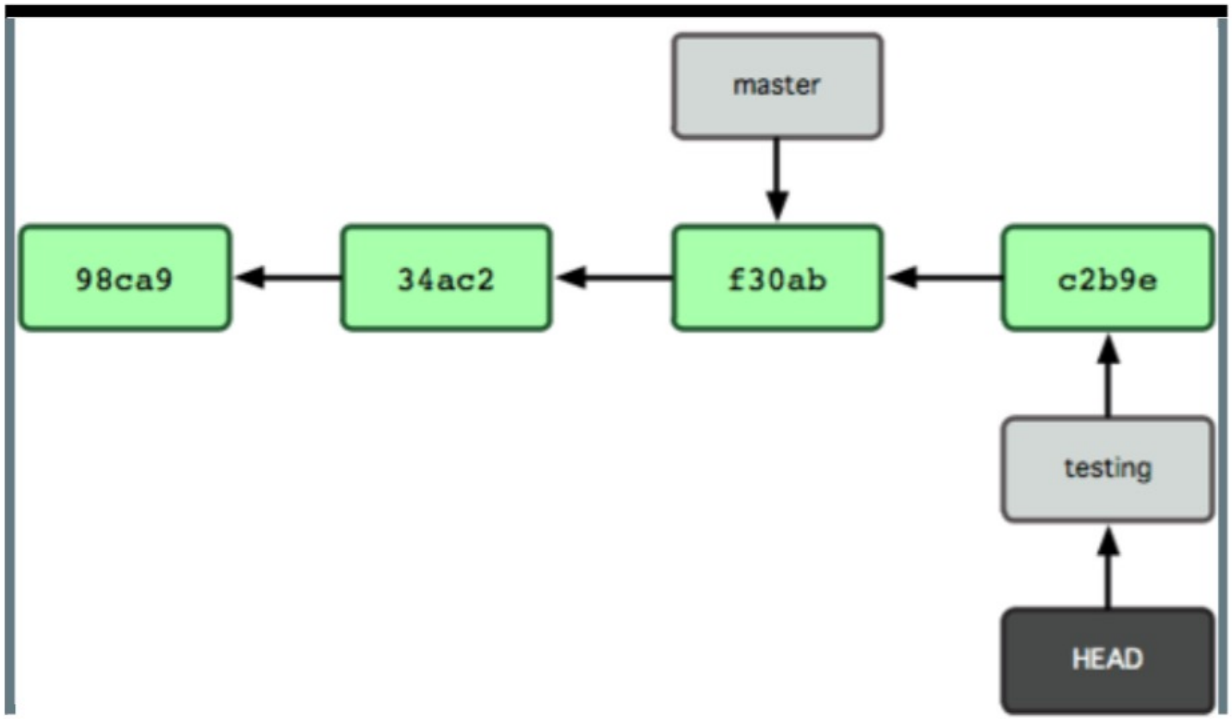
Changer de branche

```
$ git checkout testing
```



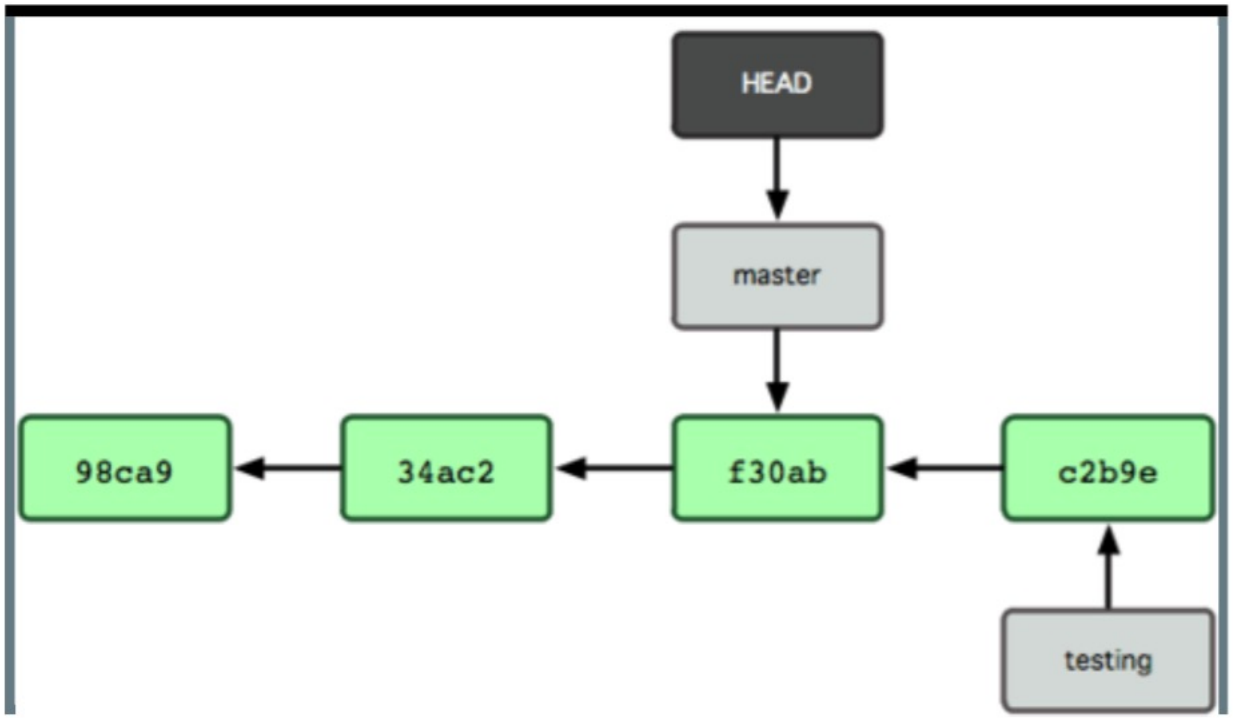
Que se passe-t-il si nous modifions un fichier?

```
$ vim test.txt
$ git commit -a -m 'petite modification'
```



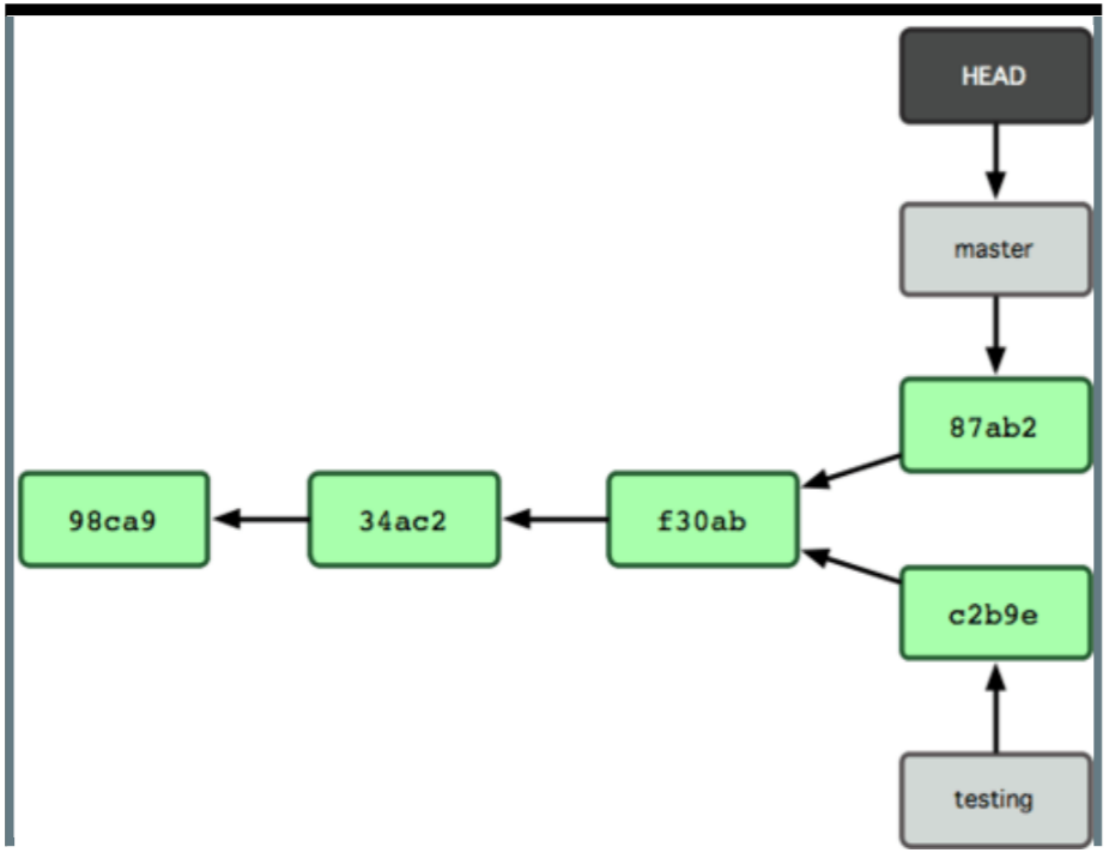
On retourne sur master

```
$ git checkout master
```



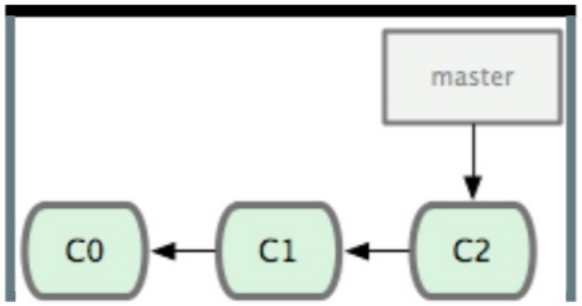
Et si on modifie de nouveau?

```
$ vim test.txt
$ git commit -a -m 'autres modifications'
```



EXERCICE : Brancher et fusionner

Dépôt Git de base

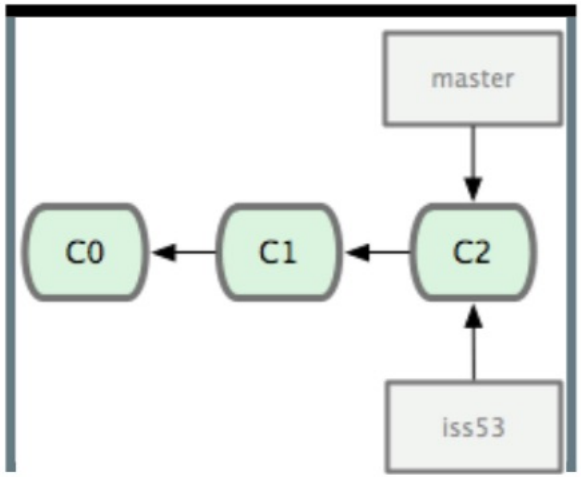


Création d'une nouvelle branche iss53

```
$ git checkout –b iss53
```

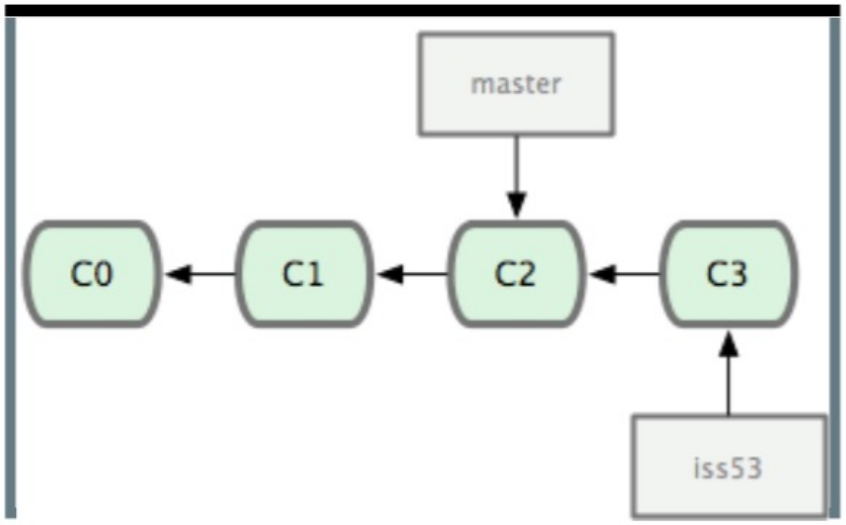
Raccourci pour :

```
$ git branch iss53
$ git checkout iss53
```



Quelques modifications dans iss53 ...

```
$ vim index.html
$ git commit -a -m 'ajout d'un pied
de page [issue 53]'
```



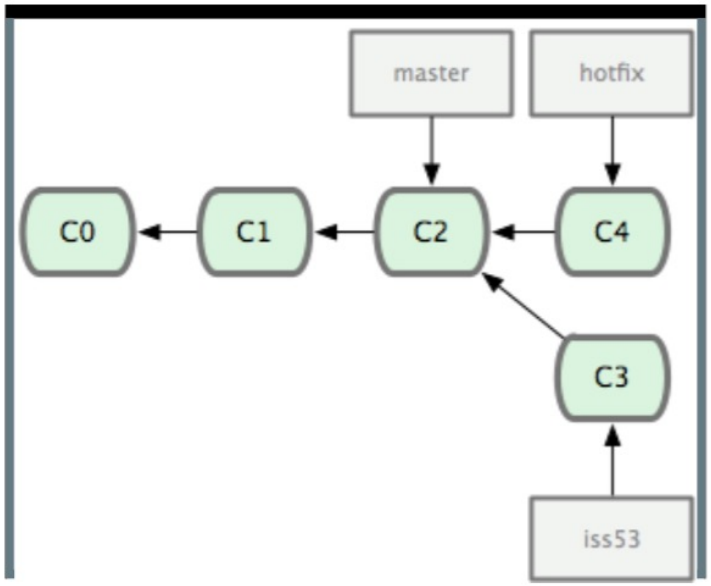
Un client appelle, il faut faire un correctif urgent ... mais vous n'êtes pas prêts à publier iss53 !

On retourne sur **master**

```
$ git checkout master
```

Et on créé une nouvelle branche pour ce correctif (hotfix)

```
$ git checkout –b ‘hotfix’
$ vim index.html
$ git commit –a –m ‘correction d’une adresse mail incorrecte’
```

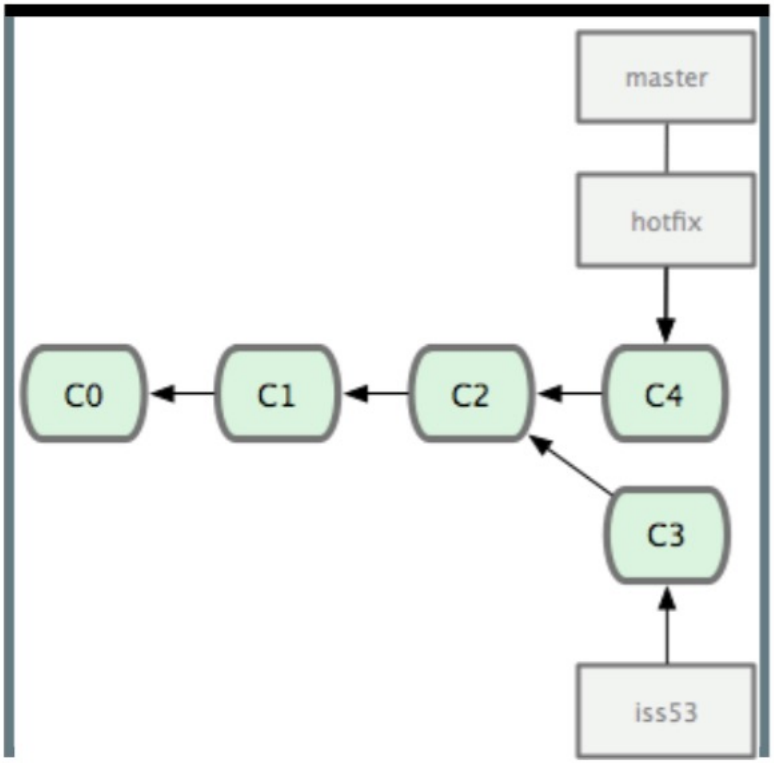



```
$ git checkout -b 'hotfix'
Switched to a new branch "hotfix"
$ vim index.html
$ git commit -a -m "correction d'une adresse mail incorrecte"
[hotfix]: created 3a0874c: "correction d'une adresse mail incorrecte"
 1 files changed, 0 insertions(+), 1 deletions(-)
```

Le correctif est ok – on le fusionne dans le master

$ git checkout master

$ git merge hotfix



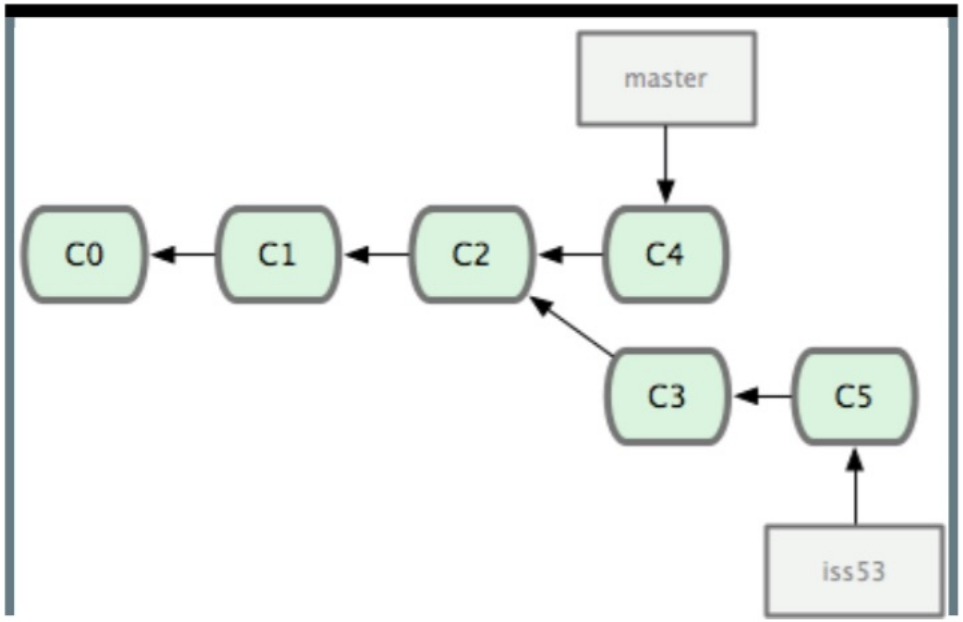
On fait un peu de ménage …

…..On supprime la branche hotfix

$ git branch –d hotfix

On retourne sur ISS53

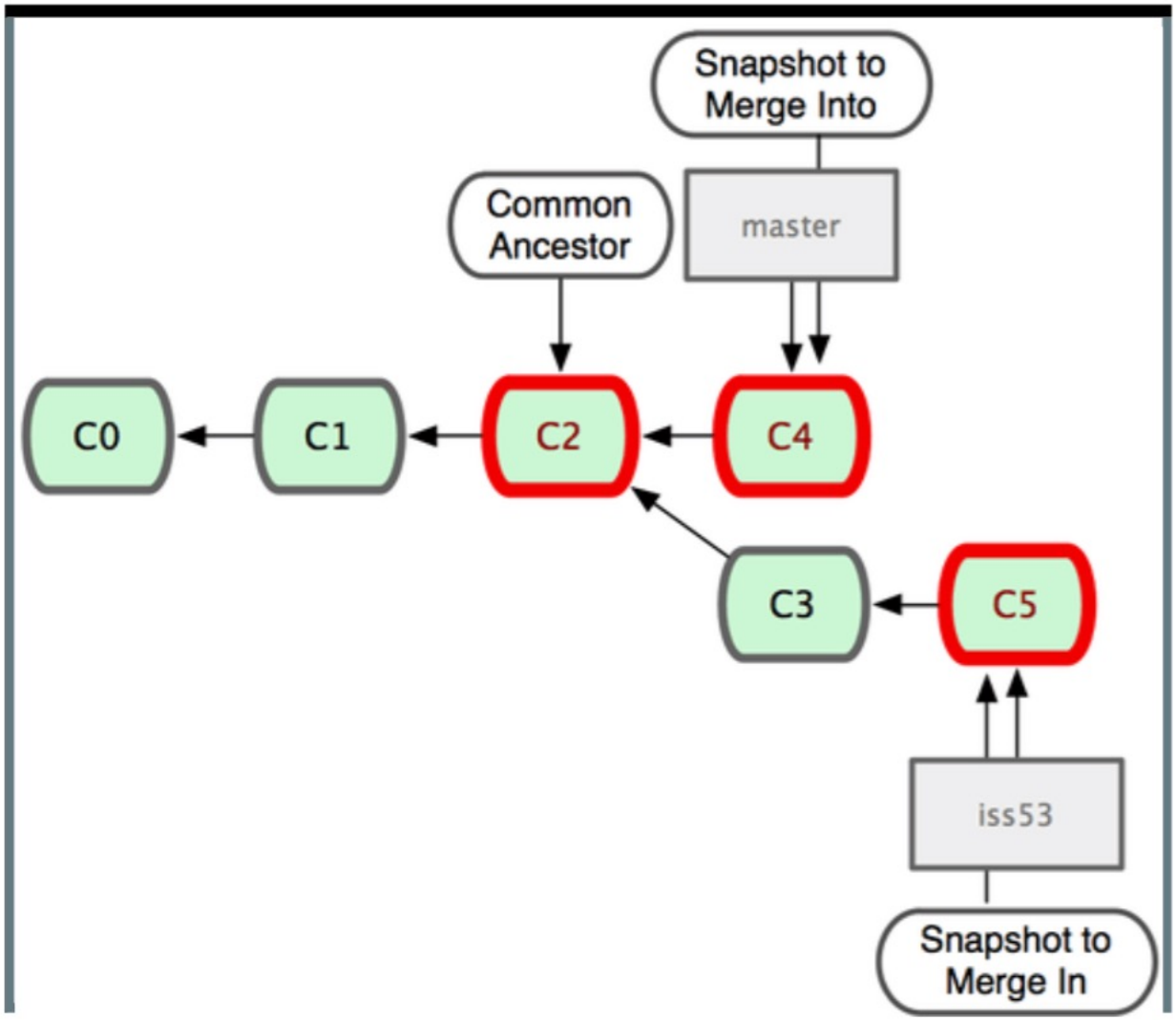
```
$ git checkout iss53
$ vim index.html
$ git commit –a –m ‘Nouveau pied de
page terminé [issue 53]’
```



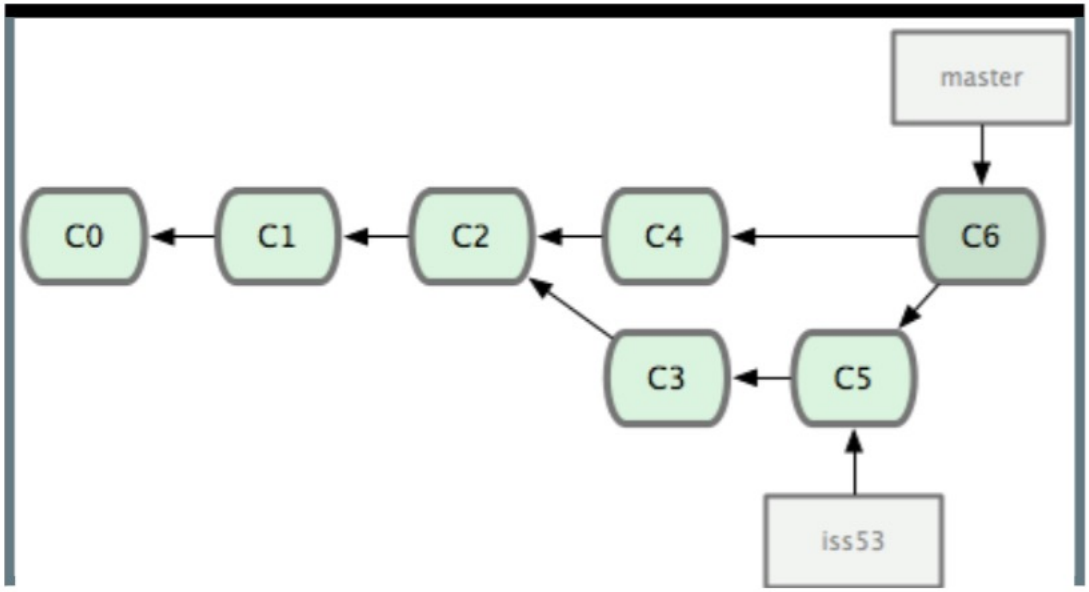
Les bases de la fusion

- Fusionnons iss53 dans master

```
$ git checkout master
$ git merge iss53
Merge made by recursive.
 README |    1 +
  1 files changed, 1 insertions(+), 0 deletions(-)
```


Snapshot to
Merge Into
Common
Ancestor
master
C0
C1
C2
C4
C3
C5
iss53
Snapshot to
Merge In

Le résultat …



Un peu de ménage …

```
$ git branch -d iss53
```