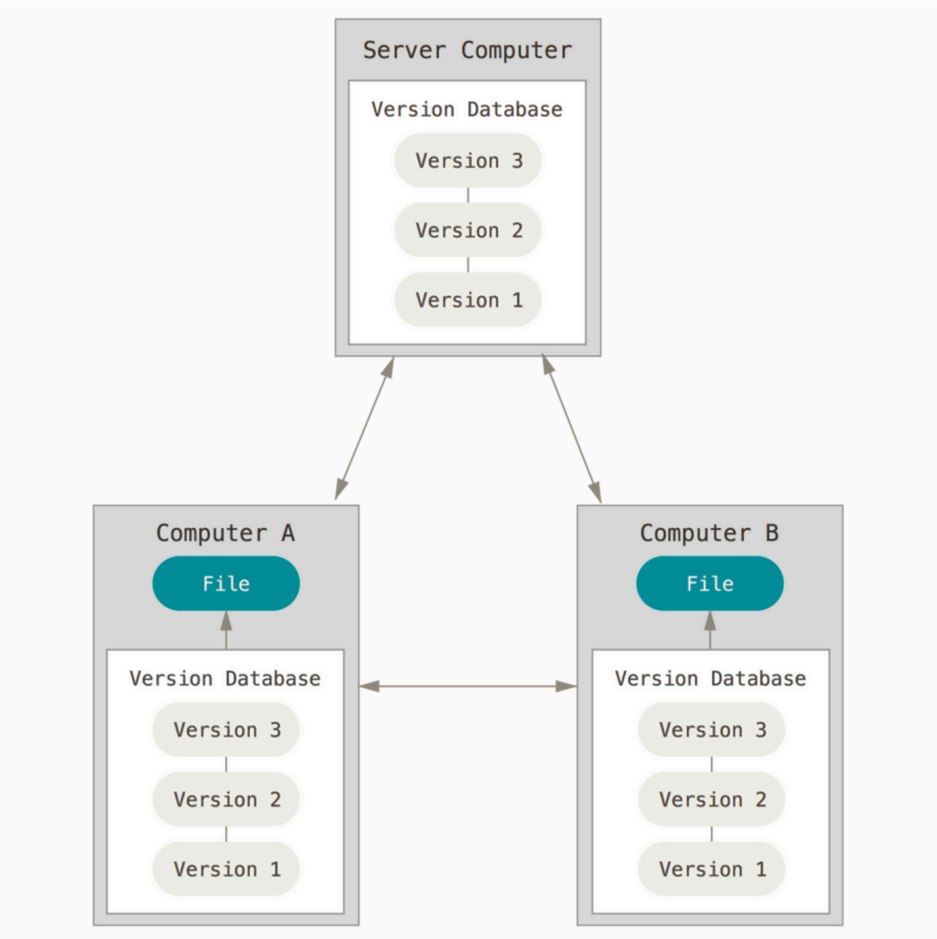
TUTORIEL D'UTILISATION DE GIT

Git est un logiciel de gestion de version décentralisé. Il est gratuit (free) et libre (open source) : https://git-scm.com/.

Qu'est-ce qu'un gestionnaire de version ?
C'est un logiciel qui permet stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées. Ceci est très utile pour nimporte quel projet, et en particulier pour les projets impliquant du développement. A tout moment, vous pouvez revenir à une ancienne version de votre programme.
Les gestionnaires de version facilitent le travail collaboratif : plusieurs personnes travaillant sur un même projet peuvent contribuer de manière individuelle au projet, et enregistrer les évolutions du projet qu'elles ont implémentés.

Pourquoi décentralisé ?
Le principal avantage de Git par rapport aux autres gestionnaires de version (svn, cvs…) réside dans son architecture distribuée, qui favorise l'autonomie des contributeurs et sécurise les données.



Le projet, incluant toutes les versions passées et la version courante, est stocké dans un dépôt (repository) sur une machine distante (Server Computer, accès via le Web). Chaque contributeur (A, B) possède une copie (ou un clone) de l'entièreté du dépôt sur sa machine (Computer A, Computer B). Chaque contributeur travaille en local sur le projet, puis « pousse » ses modifications/évolutions sur le serveur distant. Si le serveur crashe, le dépôt peut être restoré grâce aux clones des contributeurs.

1/ Notions de base

Git définit 3 états possibles pour chaque fichier : engagé (committed), modifié (modified) ou en attente (staged). Un fichier engagé est enregistré/stocké en local, sur votre machine. Un fichier modifié n'a pas encore été engagé dans la base locale. Un fichier en attente est marqué comme modifié, pour qu'il soit ensuite engagé (prochain commit).

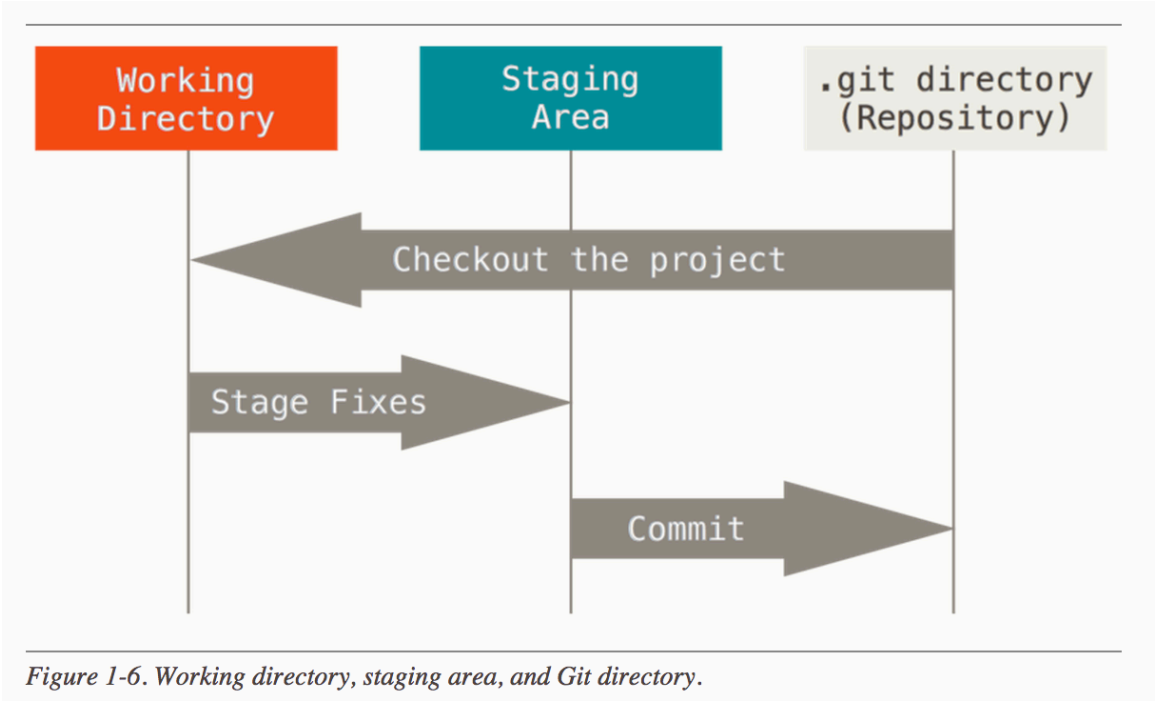


Figure 1-6. Working directory, staging area, and Git directory.

.git directory (dépôt) : répertoire Git, qui contient tout le projet et les info sur le projet. C'est ce qui est cloné sur les machines des contributeurs.
Working Directory : répertoire de travail, qui contient un snapshot (une version) du projet.
Staging Area : fichier, généralement localisé dans le .git directory, qui contient des informations sur le prochain commit.

La procédure basique pour Git est :

1. vous modifiez des fichiers dans votre répertoire de travail
2. vous marquez les fichiers modifiés (snapshot de ces fichiers dans la Staging Area)
3. vous faites un commit, qui prend les fichiers tels qu'ils étaient quand vous les avez marqués dans la Staging Area, et qui les stocke de manière permanente dans le dépôt.

2/ Développer un projet avec Git

Pour créer un nouveau dépôt
Il faut créer un répertoire pour votre projet, aller dedans et taper la commande **:** `$ git init`
Vous initialisez ainsi un suivi de projet avec Git.

Chacun des fichiers que vous allez créer et modifier dans votre répertoire de travail sera soit considéré par Git (tracked), soit non-considéré (untracked) pour la prochaine mise à jour du projet (commit).

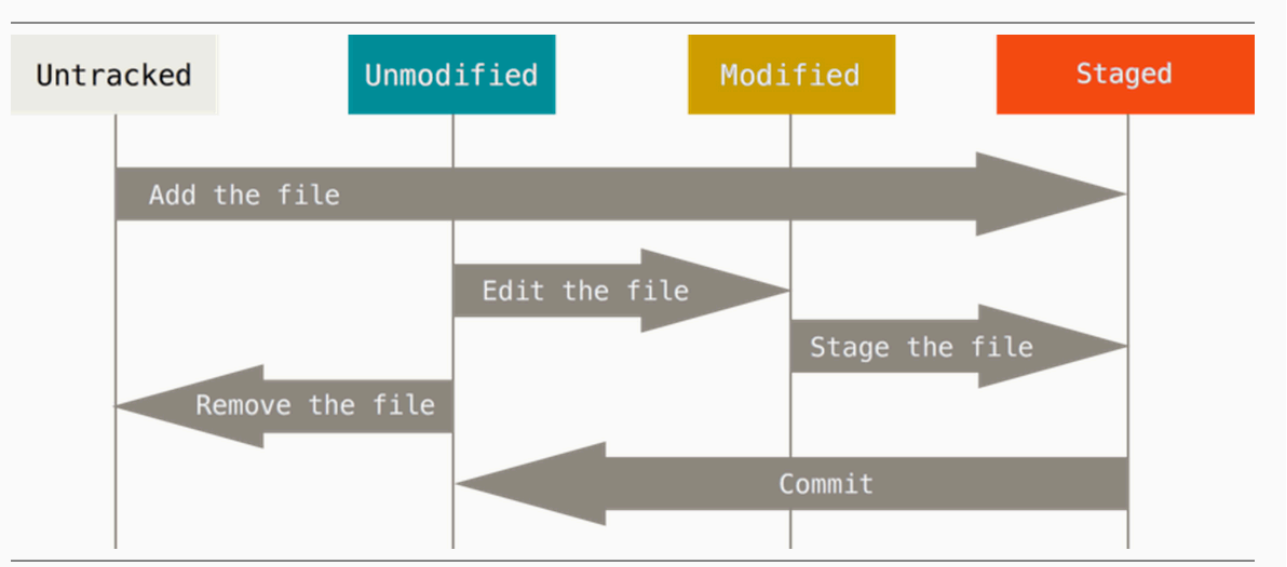


Figure 2-1. The lifecycle of the status of your files.

Pour ajouter un nouveau fichier :

Etat des fichiers de votre projet avant l'ajout

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Création d'un nouveau fichier dans le répertoire de travail

```
$ echo 'My Project' > README
$ git status On branch master Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
      README
nothing added to commit but untracked files present (use "git add" to track)
```

Git a détecté qu'un nouveau fichier a été créé. Il indique que ce fichier n'existe pas dans le dépôt et ne sera donc pas considéré, par défaut, dans le prochain commit. Vous devez dire explicitement à Git d'ajouter le fichier.

```
$ git add README
```

Etat des fichiers de votre projet après l'ajout

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
     new file:   README
```

Le fichier README est maintenant en attente (staged) pour le prochain commit.

Pour enregistrer les changements dans un fichier existant :

Vous modifiez le fichier CONTRIBUTING.md, qui existait déjà.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
     new file:   README
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
    modified:   CONTRIBUTING.md
```

Git a détecté que le fichier a été modifié, mais cette modification ne sera pas prise en compte tant que vous n'aurez pas explicitement dit à Git qu'il faudra le faire au prochain commit.

```
$ git add CONTRIBUTING.md
$ git status
On branch master Your branch is up-to-date with 'origin/master'.
Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)
     new file:   README
     modified:   CONTRIBUTING.md
```

Notez que la commande **add** ne sert pas uniquement à ajouter des fichiers. De manière plus générale, elle sert à ajouter du contenu dans le prochain commit.

Pour voir des changements spécifiques :

La commande **diff** permet de voir exactement :

- ce qui a été modifié mais pas mis en attente : `$ git diff`
- ce qui a été mis en attente pour le prochain commit : `$ git diff --staged`

Pour mettre à jour le dépôt :

Une fois que toutes les modfications voulues ont été marquées (fichiers mis en attente, ou staged), vous pouvez effectuer un commit.

```
$ git commit
```

Cette commande va lancer un éditeur, qui affiche le texte suivant :

```
# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#       new file:   README
#       modified:   CONTRIBUTING.md
# ~ ~ ~ ".git/COMMIT_EDITMSG" 9L, 283C
```

Vous pouvez laisser les commentaires pas défaut, ou taper le message de votre choix.

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master 463dc4f] Story 182: Fix benchmarks for speed
2 files changed, 2 insertions(+)
create mode 100644 README
```

Vous avez fait votre premier commit ! Notez que Git vous indique la branche sur laquelle vous avez committé (master), le code du commit (463dc4f), le nombre de fichiers modifiés et des statistiques sur les lignes ajoutées et supprimées dans le commit.
Attention ! Tout ce qui n'aura pas été mis en attente au préalable ne sera pas committé !

Pour skipper l'étape de staging :

Vous avez la possibilité de skipper l'étape de mise en attente grâce à l'option **–a** de la commande commit.

```
$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
1 file changed, 5 insertions(+), 0 deletions(-)
```

Pour supprimer des fichiers :

Si vous supprimez un fichier de votre répertoire de travail, la suppression de ne sera pas prise en compte dans le prochain commit, par défaut.

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
 (use "git add/rm <file>..." to update what will be committed)
 (use "git checkout -- <file>..." to discard changes in working directory)
        deleted:    PROJECTS.md
no changes added to commit (use "git add" and/or "git commit -a")
```

Il faut explicitement dire à Git que vous voulez supprimer le fichier au prochain commit.

```
$ git rm PROJECTS.md rm 'PROJECTS.md'
$ git status On branch master Your branch is up-to-date with 'origin/master'.
Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)
     deleted:    PROJECTS.md
```

Alternativement, vous pouvez supprimer le fichier du dépôt du projet, au prochain commit, tout en gardant le fichier dans votre répertoire de travail. Il suffit de ne pas faire la commande **rm** préalablement à la commande **git rm**.

Pour renommer ou déplacer des fichiers :

Par défaut, Git ne prend pas en compte le renommage ou le déplacement de fichiers. Il faut explicitement utiliser la commande : `$ git mv`

Pour voir l'historique des changements :

La commande log permet de voir l'historique des changements/évolutions du projet.

3/ Travailler à plusieurs sur un projet

Pour pouvoir collaborer sur un projet Git, il faut utiliser un dépôt distant, stocké sur Internet ou autre, à partir duquel vous récupérez des données et sur lequel vous déposez des données.

Pour cloner un dépôt distant (localisé sur un serveur)

`$ git clone https://github.com/meetU-MasterStudents/partage.git`
Par défaut, un répertoire **partage** qui contient un clone du dépôt distant est créé sur votre machine. Un répertoire .git est initialisé à l'intérieur du répertoire **partage**.
Si vous voulez cloner dans un répertoire du nom de votre choix, vous pouvez taper :
`$ git clone https://github.com/meetU-MasterStudents/partage.git monProjet`

Lorsque vous clonez pour la première fois le dépôt, tous les fichiers sont considérés par Git (tracked) et sont non-modifiés (unmodified) puisque vous n'avez encore fait aucun changement.

Pour lister tous les dépôts distants d'un projet

La commande **remote** liste tous les serveurs distants configurés pour le projet. Par défaut, le serveur duquel a été cloné le dépôt s'appelle **origin**.

```
$ git remote
origin
$ git remote -v
origin   https://github.com/meetU-MasterStudents/partage.git (fetch)
origin   https://github.com/meetU-MasterStudents/partage.git (push)
```

Dans ce cas, il y a un seul serveur distant. Vous pouvez obtenir plus d'information avec la commande **remote show**.

Pour récupérer des données depuis un dépôt distant

```
$ git fetch origin
```

Cette commande permet de récupérer toutes les données du projet distant (dans ce cas-là **origin**) que vous n'avez pas encore. Les données sont téléchargées dans le répertoire local, sans être fusionnées avec votre travail courant. Vous devez les fusionner manuellement. Pour télécharger et fusionner, il faut utiliser la commande **pull**.

Pour partager les données

```
$ git push origin master
```

Cette commande permet de partager votre travail avec vos collaborateurs en le mettant sur le dépôt distant. Elle fonctionne à deux conditions : (1) vous avez les droits d'écriture sur le dépôt distant, (2) personne d'autre n'a fait de push depuis votre dernier clonage du dépôt distant. Si (2) n'est pas remplie, vous devez faire un fetch pour récupérer le travail des autres, le fusionner avec votre travail et ensuite faire un push.

4/ Les branchements sous Git

La création de branches sous Git consiste à diverger de la branche principale de développement (master) et permet de continuer à travailler sans la perturber.

Pour créer une branche : `$ git branch testing`

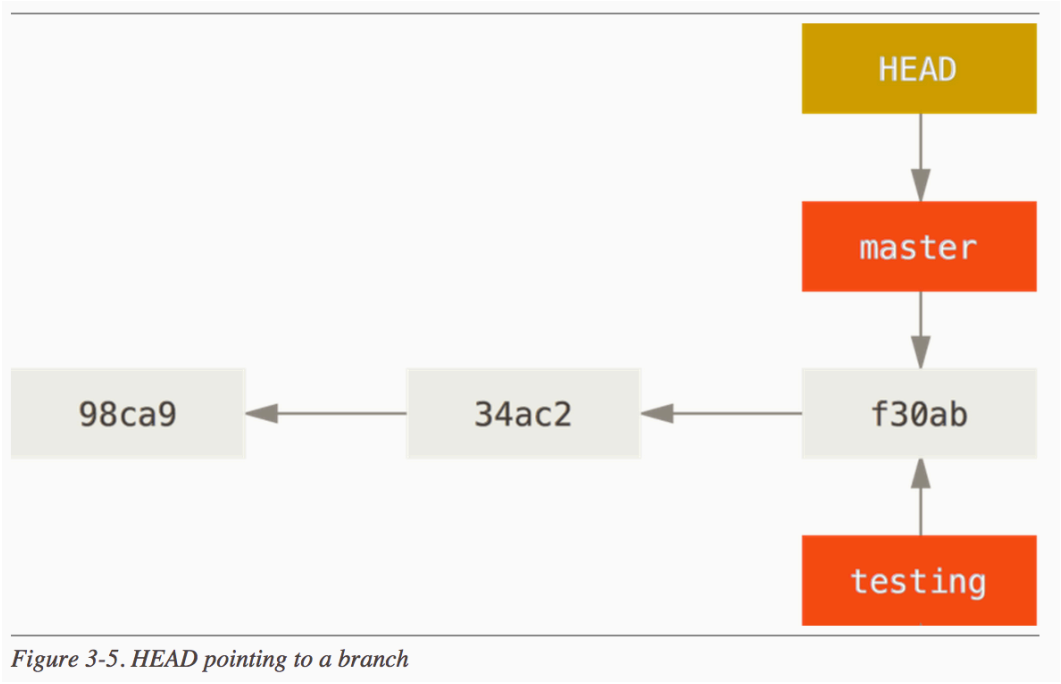


Figure 3-5. HEAD pointing to a branch

La branche principale (master) et la nouvelle branche (testing) pointent vers les commits successifs du projet (boîtes grises). Un pointeur spécial (head) pointe sur la branche sur laquelle vous êtes en train de travailler.

Pour changer de branche : `$ git checkout testing`
Le pointeur head se déplace sur la branche testing.

Si vous faites les opérations suivantes :

```
$ vim test.rb                                    // modification d'un fichier
$ git commit -a -m 'made a change'               // commit sur la branche testing
$ git checkout testing                           // changement de branche (master)
$ vim test.rb                                    // modification d'un fichier
$ git commit -a -m 'made other changes'          // commit sur la branche master
```

Vous obtenez le schema suivant, qui montre bien que vous avez divergé.

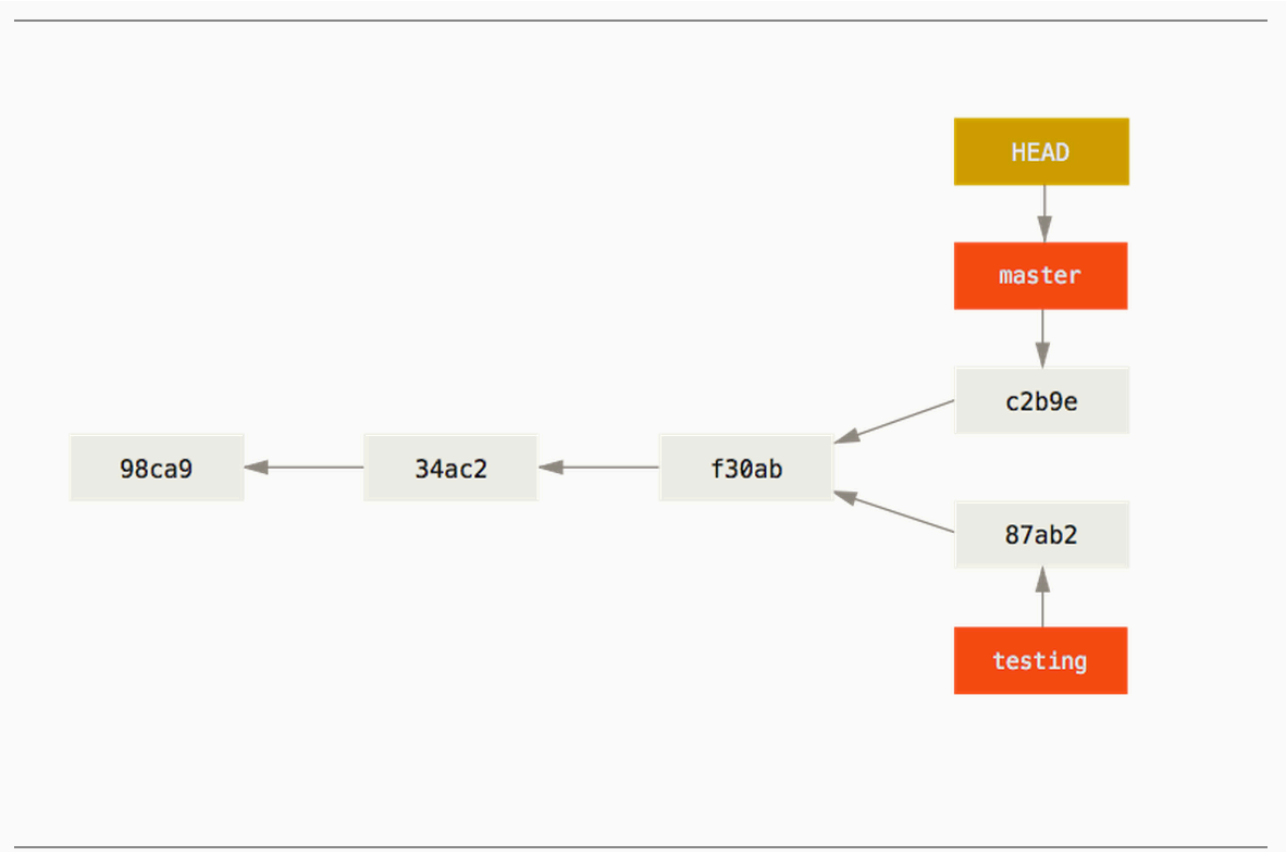


Figure 3-9. Divergent history

Vous pouvez constater cette divergence avec la commande **log**.

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) made other changes |
* 87ab2 (testing) made a change |
/
```

Vous pouvez intégrer les différentes branches avec la commande **merge**.
Attention ! Ceci peut créer des conflits !

Lorsque la branche sur laquelle vous travaillez suit une branche sur le dépôt distant, vous pouvez utiliser la commande **pull** pour télécharger les données de la branche distante et les fusionner avec votre travail.

Plus de détails ici : https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging