

# Introduction à la programmation en Java

## Cours 11

Stéphane Airiau

Université Paris-Dauphine

# Git

- Collaboration avec d'autres développeur
- Pourquoi une modification, qui l'a effectué ? Ou est-ce que ça ne marche plus ?
- Quel a été l'ordre des modifications ?
- Peut-on revenir en arrière ?
- ➡ logiciel de gestion des *versions*
- `git` outil développé par Linus Torvald, auteur du kernel Linux

## Différents types d'outils

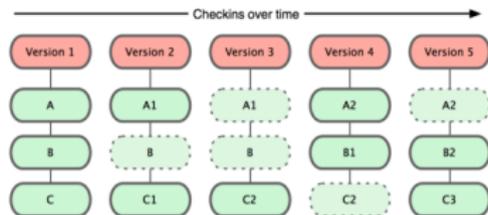
---

- modèle centralisé : le code est stocké sur un serveur central  
SVN, CVS  
problème du point unique de panne ➡ aucun client ne peut enregistrer des changements.
- modèle distribué : pas de serveur central, toutes les machines ont accès à tout le code  
git
- ➡ le code est accessible par plusieurs sources, pas de problèmes si le serveur plante...
- ➡ pas besoin de se connecter sur un serveur central pour travailler. Il faudra internet pour collaborer.
- ➡ chaque extraction est une sauvegarde complète des données !

- enregistrer les modifications de temps en temps ➡ faire un `commit`
- un `commit` ➡ version du code à un instant
- on obtient un historique du projet
- ➡ on peut se déplacer dans l'historique du projet

Git va enregistrer un nouvel état :

- il n'enregistre pas les fichiers qui n'ont pas changé (ref vers le fichier le plus récent)
- il enregistre les fichiers qui ont changé (pas juste les modifications)
- un peu un mini système de fichiers



➡ Git crée une série d'instantanées, un pour chaque `commit`.

## Pertes

---

Une fois les modifications entrées dans la base, il est très difficile de perdre de l'information.  
(avec un dépôt distant)

## Beaucoup d'opérations en local

---

- grande vitesse pour parcourir l'historique
- connaître la différence entre l'état courant et un état de l'historique donné ↔ calculé en local

- Git gère l'intégrité
- tout est vérifié avant d'être stocké
- utilisation d'une somme de contrôle
- si on modifie le contenu d'un fichier, Git va s'en apercevoir !
- identifiant unique : le SHA-1 - une *empreinte* de 40 caractères hexadécimaux (qui contient de l'information)

## Démarrer un dépôt Git

---

- initialiser un dépôt Git

```
$ git init
```



création d'un répertoire `.git` contiendra les fichiers générés par Git

- pour suivre les versions des fichiers existants

```
$ git add
```

- Obtenir une copie d'un dépôt existant

```
$ git clone
```

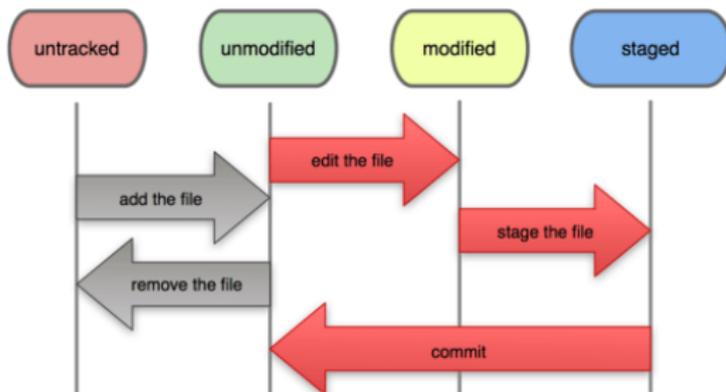
↪ réception de toutes les données dont le serveur dispose (en particuliers, toutes les versions)

## Etat des fichiers

Les fichiers peuvent être sous les états suivants :

- sous suivi de version : fichiers inchangés, modifiés ou indexés
  - modifiés : différents de ceux enregistrés
  - indexés : on dit qu'il faudra les placer dans le prochain dépôt
  - lorsqu'on *commit*, on enregistre les fichiers indexés
- non suivi : on n'est pas obligé de suivre tous les fichiers

### File Status Lifecycle



```
$ git status
```

## Suivre des fichiers

---

Pour commencer à suivre un nouveau fichier

```
$ git add
```

Attention, il faut relancer `git add` pour prendre en compte l'état actuel de la copie de travail

## Valider les modifications : `commit`

---

```
$ git commit
```

- vous pouvez/devez ajouter un message d'explication sur la modification que vous validez
- on vous indique sur quelle branche la validation a été effectuée (cf plus tard)
- somme de contrôle SHA-1
- combien de fichiers ont été modifié
- quelques stats sur les lignes changées

L'option `-a` à la commande `git commit` place tout fichier déjà en suivi dans la zone d'index

➤ évite d'avoir à taper les commandes `git add`

## Historique des validations

---

```
$ git log
```

donne l'historique des commits réalisés (empreinte sha, auteur, date, message)

On peut

- demander les changements (avec un diff)
- avoir quelques stats (option `-stat`) (nombre de lignes ajoutées, retirées, liste des fichiers modifiés, ...)
- pas mal d'options sont disponibles...

## Annuler des actions!

---

- erreur classique : on a oublié d'ajouter des fichiers, ou bien on a fait une erreur dans le message ➡ on aura un seul commit amendé.

```
$ git commit - amend
```

- désindexer un fichier
- réinitialiser un fichier modifié (Git vous dit comment faire dans l'output du commit)

## Etiquetage

---

A vous de choisir ou de nommer les commits importants.

```
$ git tag -a v1.8 -m 'super nouvelle version  
1.8'
```

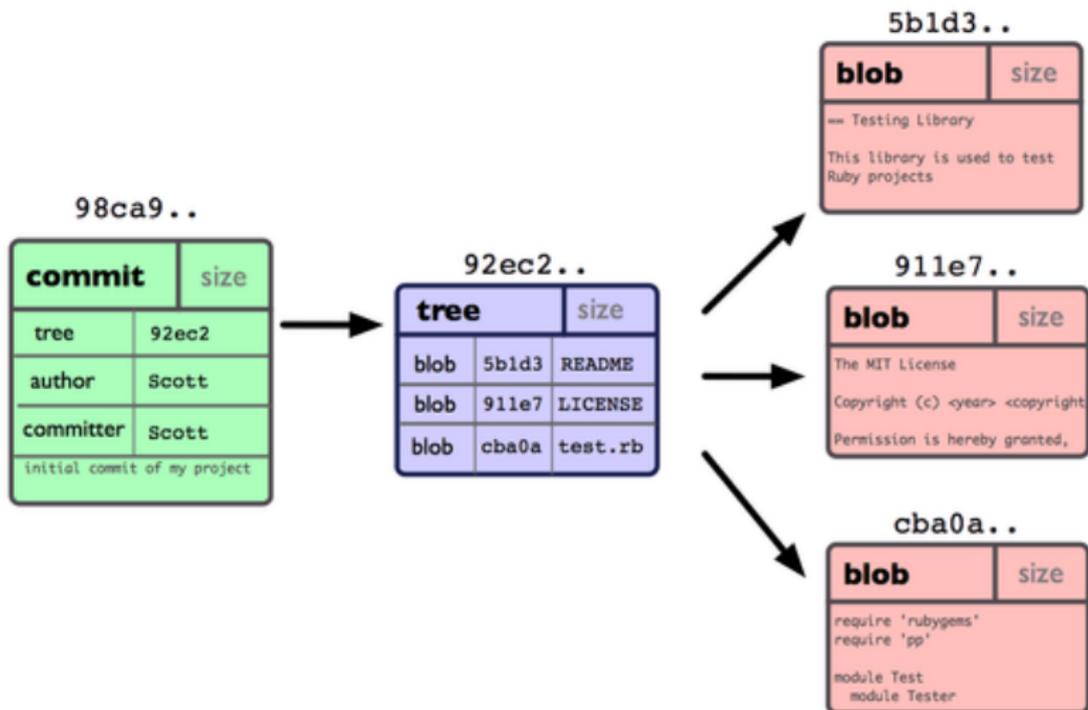
```
$ git show v1.4
```

Parfois, on veut commencer une fonctionnalité sans complètement savoir si ce sera utile :

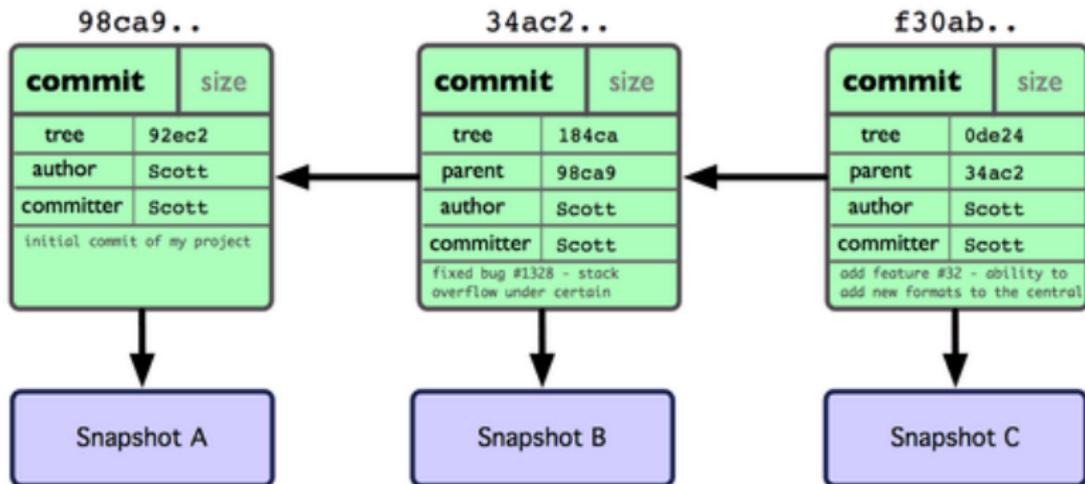
On peut commencer à coder cette fonctionnalité, tout en laissant les autres continuer sur la ligne principale.

- *ouvrir une branche* : Diverger de la ligne principale sans se préoccuper de cette ligne principale.
- une des particularités de Git ➡ assez légère
- créer une branche puis fusionner peut être une bonne façon de développer

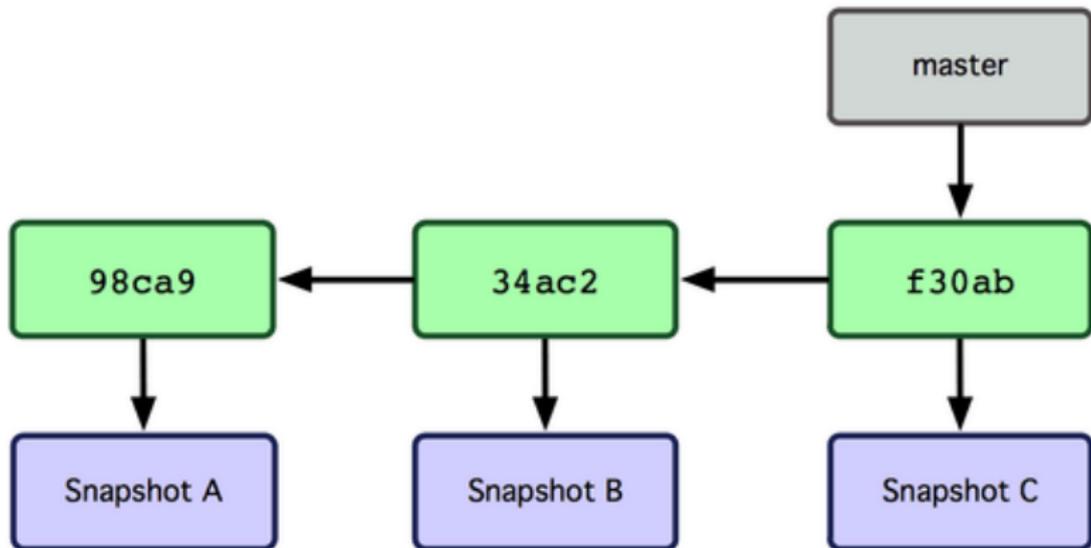
## Le commit et son arbre



## commit et les parents



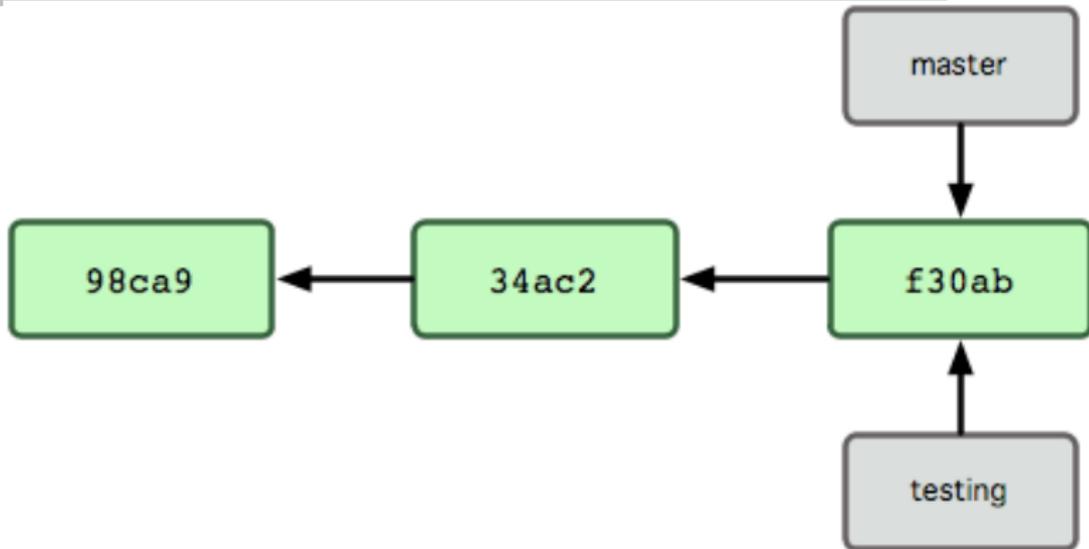
## Le master



## Une branche (qui pointe sur le même commit)

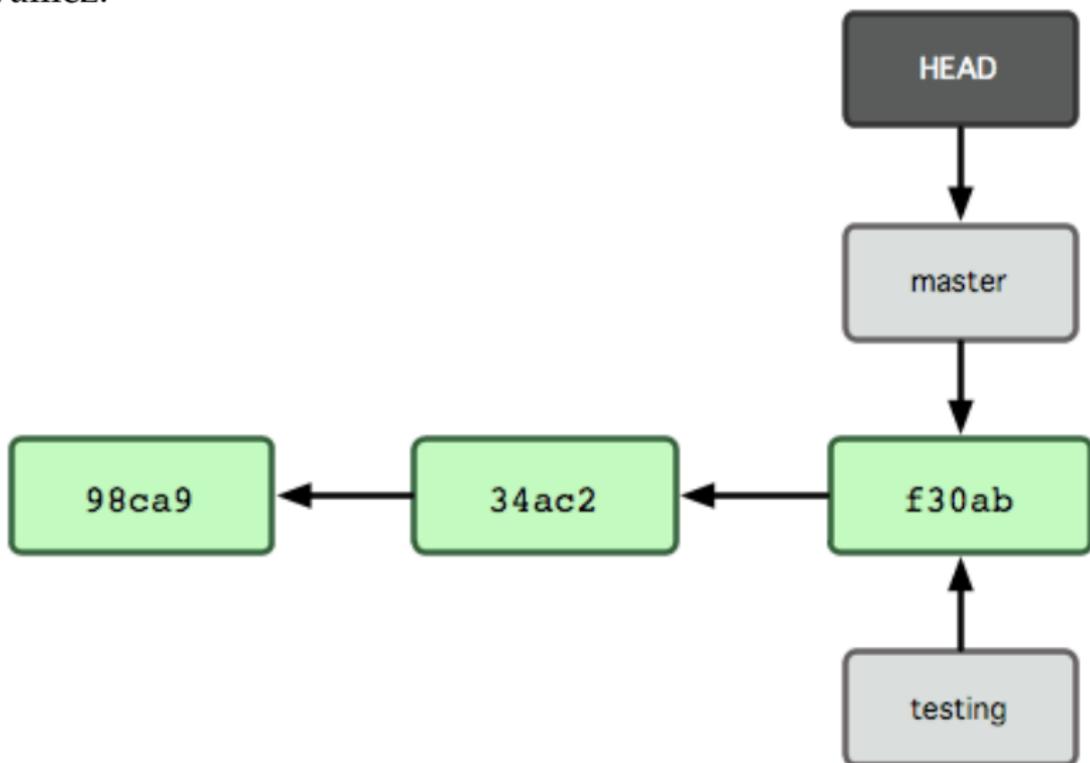
---

```
$ git branch testing
```



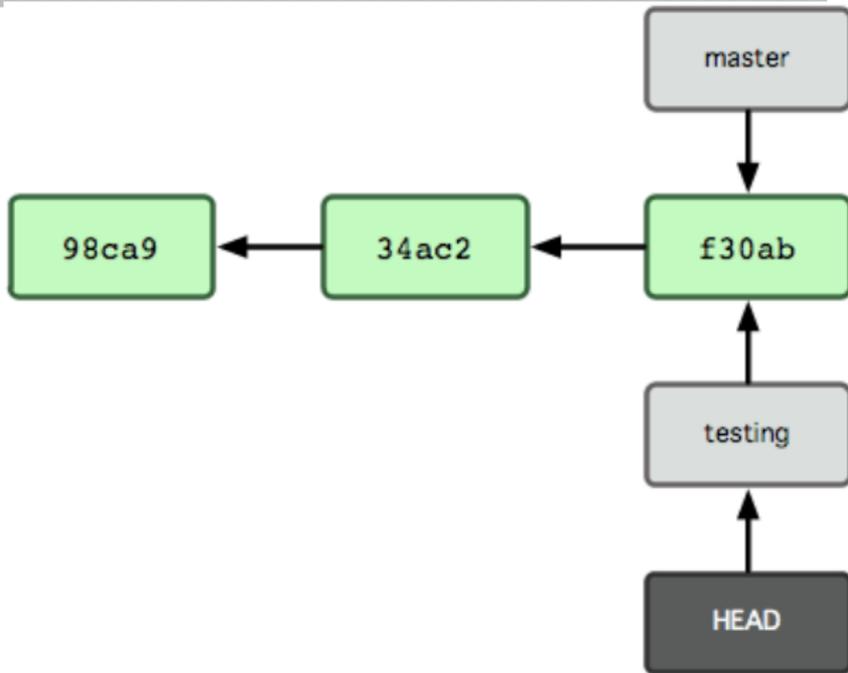
## On pointe sur la nouvelle branche

Pointeur spécial appelé HEAD pour désigner la branche sur laquelle vous travaillez.



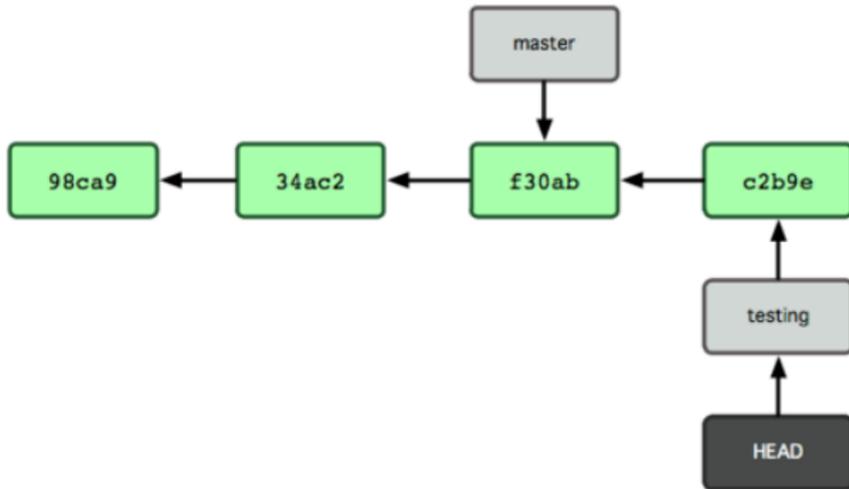
## On peut se déplacer sur une autre branche

```
$ git checkout testing
```



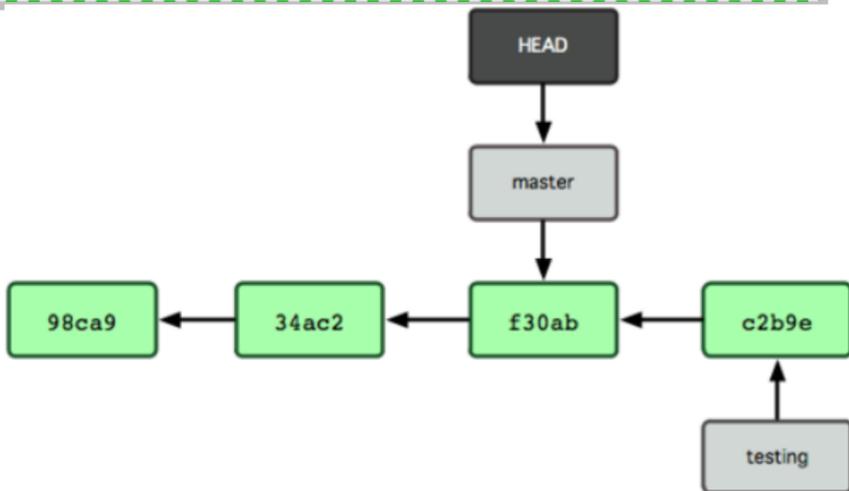
On peut continuer à travailler sur la branche

---



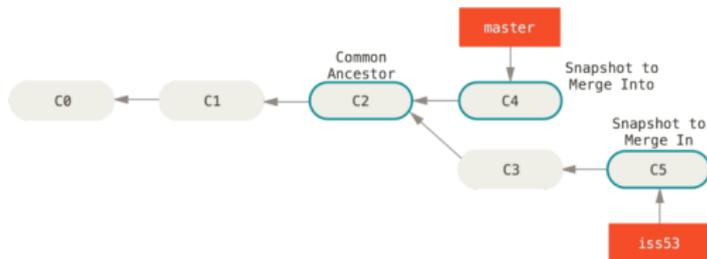
## On peut retourner au master

```
$ git checkout master
```

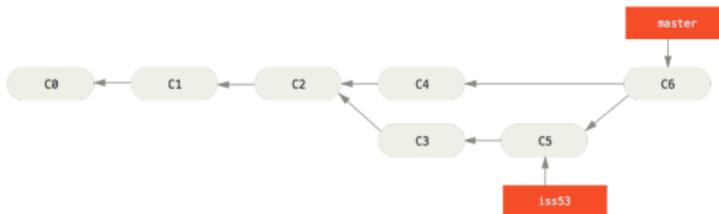


par exemple, quelque chose ne marchait pas et il fallait revenir rapidement à une version qui marche !

# Fusion de branches!



```
$ git checkout master  
Switched to branch 'master'  
$ git merge prob53  
Merge made by the 'recursive' strategy.  
README | 1 +  
1 file changed, 1 insertion(+)
```



## Gestion des conflits!

---

Tout ce qui comporte des conflits et n'a pas été résolu est listé comme `unmerged`

(des régions du code sont différentes et `Git` ne peut pas savoir quelle est la bonne)

`Git` vous indique les zones de conflit

C'est à vous de les régler! (i.e. éditer les fichiers et régler les conflits!)

- en local : dépôt sur un autre répertoire dans le système de fichier
  - ↳ dangereux si la machine plante
- protocole SSH
  - accès authentifié
  - facile à mettre en oeuvre
  - mais pas d'accès anonyme
- protocole git
  - le plus rapide
  - pas d'authentification (mais on peut coupler avec SSH)
  - plus difficile à mettre en place
- avec HTTP HTTPS
  - facile à mettre en place, donner un accès public en lecture
  - pas très efficace pour le client

# GitHub : hébergeurs de dépôts Git

---