
Gérer les codes sources avec GIT

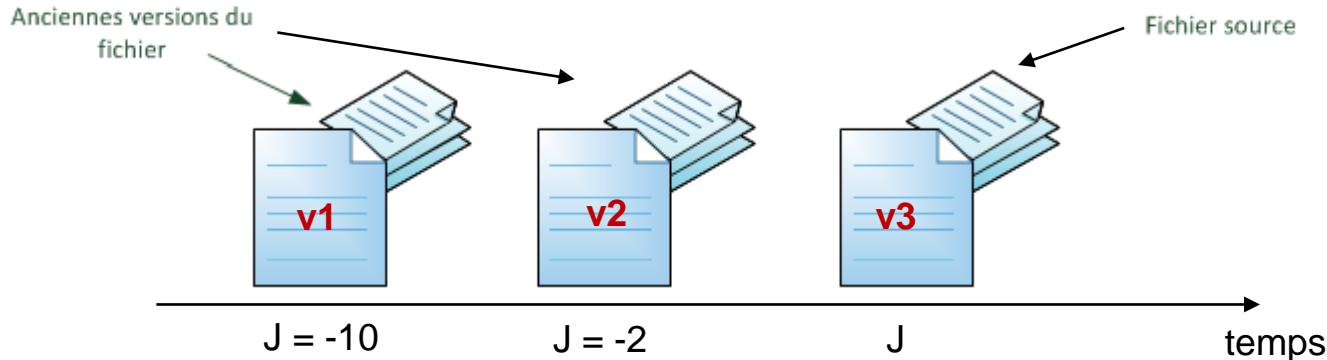
Karine BRIFAUT

Sommaire

- **Généralités sur les gestionnaires de version**
- Présentation de Git
- Serveur d'hébergement
- Conclusion

Concepts généraux (1)

- Un **logiciel de contrôle de versions (SVC)** est un logiciel qui enregistre tous les états d'une arborescence au fil du temps permettant de revenir sur une version précédente.



- Un SVC permet de gérer les **modifications de code** sources mais également toutes sortes d'information telles que documentations ou encore images car tout est donnée.

Concepts généraux (2)

- Un SVC **photographie** (parfois de **manière différentielle**) l'arborescence au fil du temps et **organise** les données sous forme de **révisions** ou **plan de révisions**.
- Un SVC garde une **trace** de toutes les **modifications** (qui, quand, où, quoi, pourquoi).
- Un SVC est **accessible** via le réseau **par tous** les utilisateurs **à tout moment**.
- Un SVC permet de **développer du code** de façon **collaborative** en autorisant plusieurs développeurs de travailler et modifier un même fichier (contrairement à Synergie : technique de lock).
 - ⇒ Evite les « archives » dont on perd vite le compte
 - ⇒ Permet un **gain de place**
 - ⇒ Evite de **devoir prévenir** les utilisateurs à chaque modification
 - ⇒ Evite **l'échange de versions** par mail...

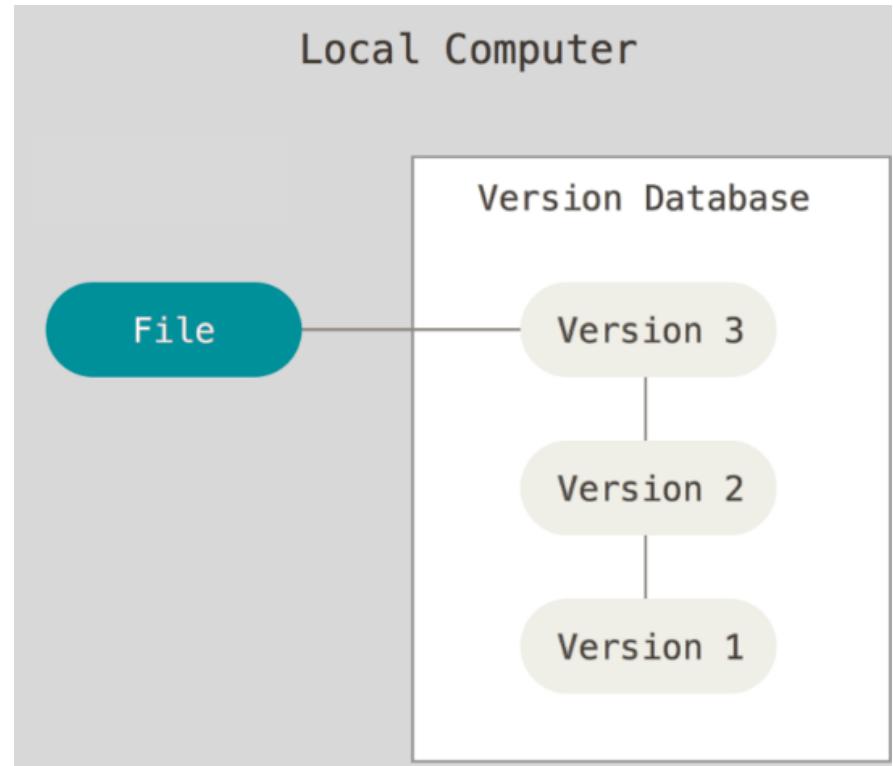
Concepts généraux (3)

❑ Logiciels de gestion locale

- Gestion de version uniquement sur un poste local

⇒ Permettent de garder les versions antérieures et avoir un historique des modifications au cours du temps en local pour soi.

Exemple : RCS (1982) ...



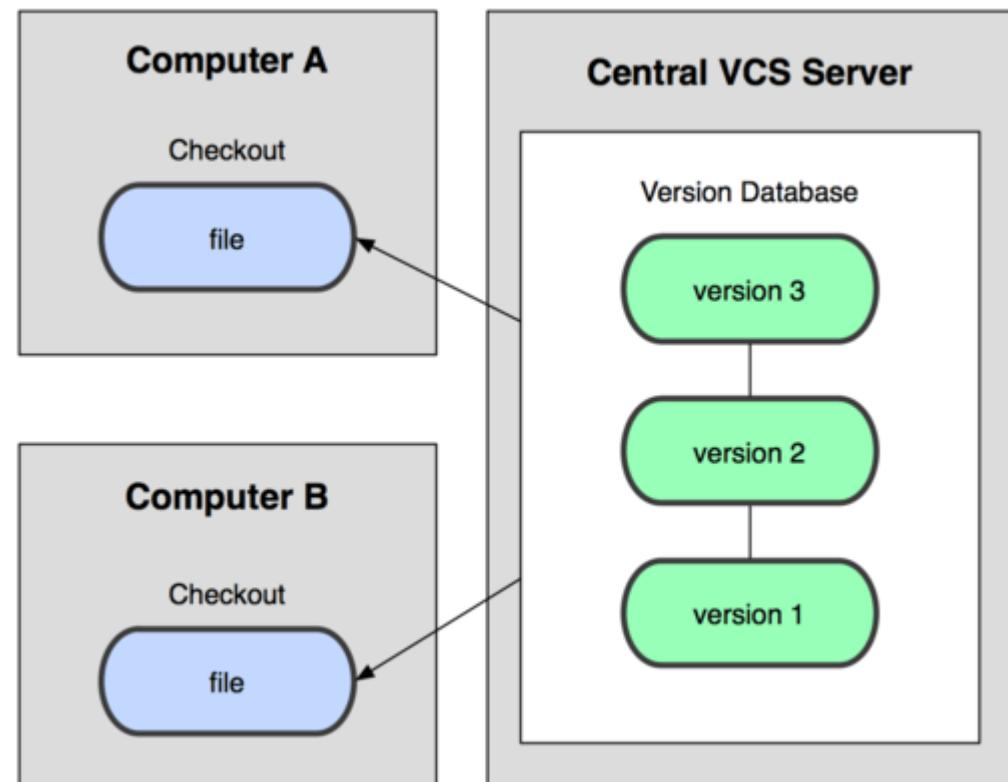
Concepts généraux (4)

□ Logiciels de gestion centralisés (CVCS)

- Réfèrent unique sur serveur.
- Les postes se connectent pour se synchroniser.

⇒ **Permet un travail collaboratif**

Exemples : CVS (1990),
ClearCase, Synergie, PVCS,
SVN (2000)...

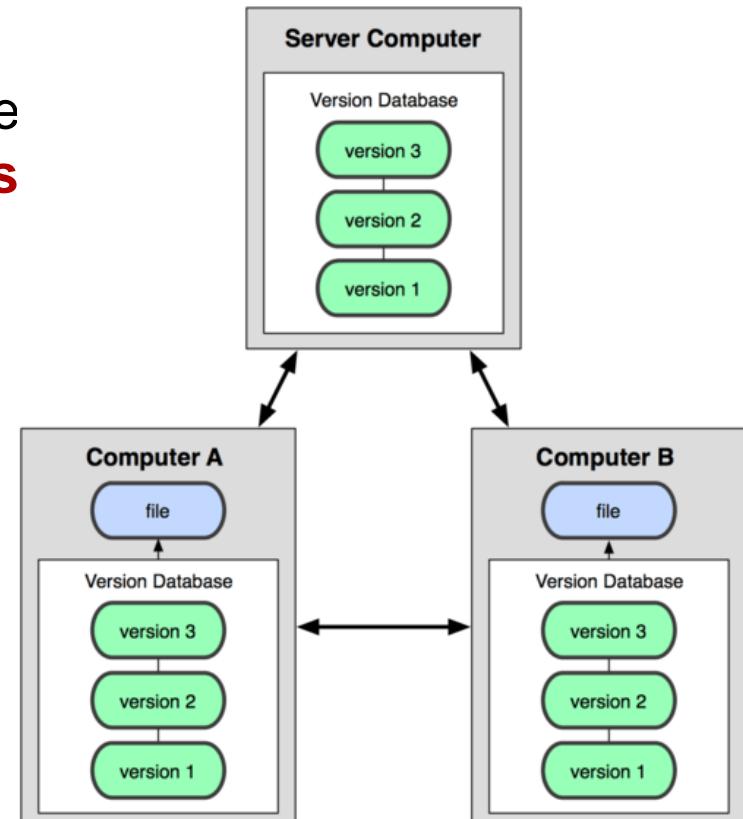


Concepts généraux (5)

□ Logiciels de gestion décentralisés (DVCS)

- Pas de dépôt central unique, chaque développeur dispose en local d'un historique des versions
- Le développeur est responsable de son dépôt et doit intégrer (merger) le travail des autres **en ramenant dans le sien**
- Le travail peut être livré (commit) sans connexion réseau

⇒ **Permet un travail collaboratif sans surcharge du serveur**



Exemples : Bazar, Mercurial, BitKeeper, GIT (2005),...

Panorama des gestionnaires existants

Libres	Gestion locale	GNU RCS (1982) · GNU CSSC
	Client-serveur	CVS (1990) · CVSNT (1992) · SVN (2000)
	Décentralisé	GNU arch (2001) · Darcs (2002) · DCVS (2002) · SVK (2003) · Monotone (2003) · Codeville (2005) · Git (2005) · Mercurial (2005) · Bazaar (2005) · Fossil (2007) · Veracity (2011)
Propriétaires	Gestion locale	SCCS (1972) · PVCS (1985)
	Client-serveur	Rational ClearCase (1992) · CCC/Harvest (années 70) · CMVC (1994) · Visual SourceSafe (1994) · Perforce (1995) · AccuRev SCM (2002) · Sourceanywhere (2003) · Team Foundation Server (2005) · Rational Synergy (2006)
	Décentralisé	BitKeeper (1998) · Plastic SCM (2007)

Pourquoi passer de SVN à Git ?

- **Où pourquoi passer d'un système centralisé à un système décentralisé ?**
 - Possibilité de commiter même sans l'accès au réseau
 - Possibilité de faire des commits en local pour tester des modifications de codes variées
 - Faciliter de faire des branches et de les éliminer sans impacter les autres (si cela est bien fait)
 - On peut utiliser Git pour travailler sur SVN

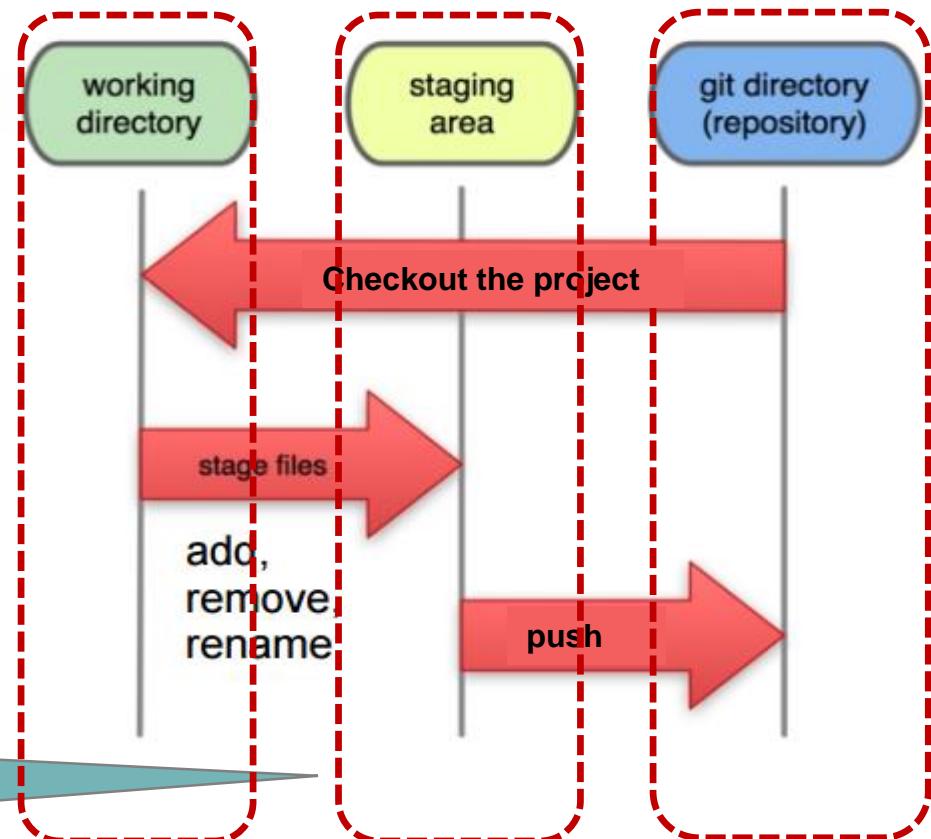
Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - ***Concepts généraux***
 - Qu'est ce qu'un instantané ?
 - Cycle de vie d'un fichier
 - Concept de branche
 - Création de branches
 - Branche et fusion
 - Merge et rebase
- Serveur d'hébergement
- Conclusion

Présentation de Git

□ Git

- Logiciel libre créé par Linus Torvalds en 2005
- **Distribué**, permettant de travailler en local tout en ayant accès au dépôt central
- **Gère 3 zones / états**



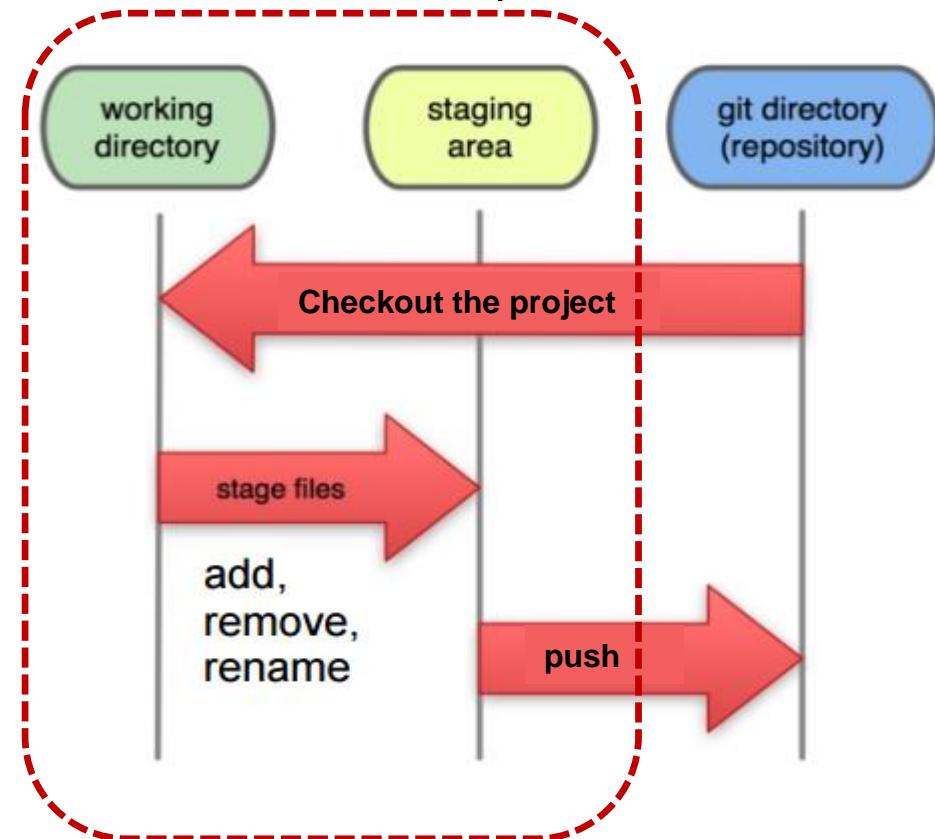
Répertoire de travail

□ Localement

- Le répertoire de travail est une extraction unique d'une version du projet
- La plupart des opérations de Git ne nécessitent que des fichiers et ressources locaux
 - Historique
 - Versions antérieures
 - ...

⇒ Rapidité d'exécution

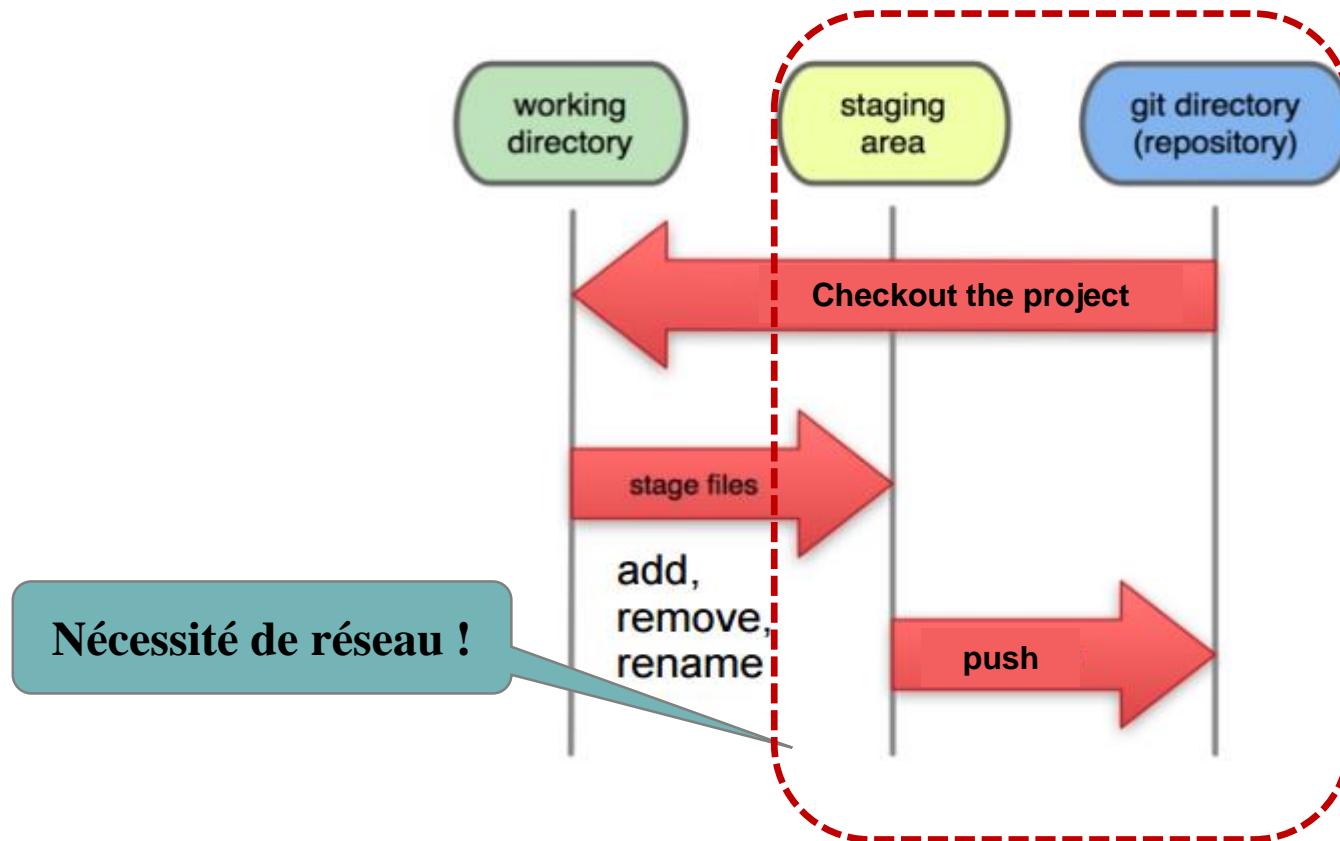
⇒ Pas de nécessité de réseau,
même pour soumettre des modifications



Répertoire Git distant

□ Répertoire Git distant

- Le **push** bascule le contenu de la zone d'index vers le répertoire Git

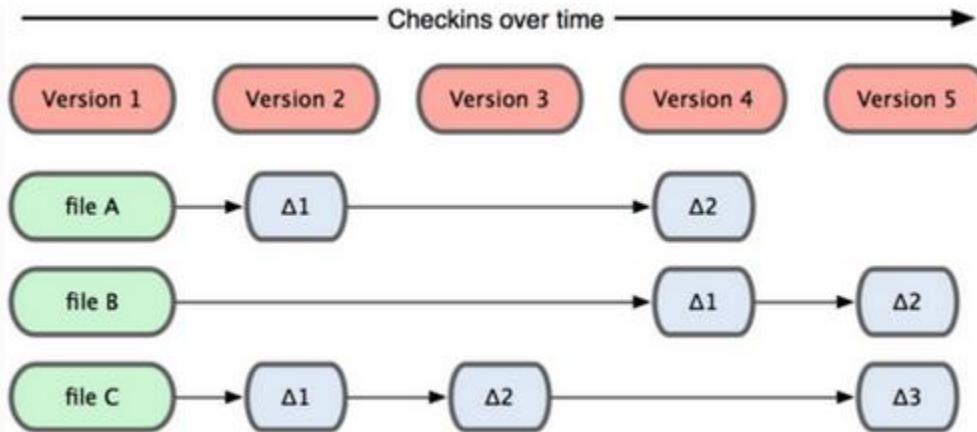


Sommaire

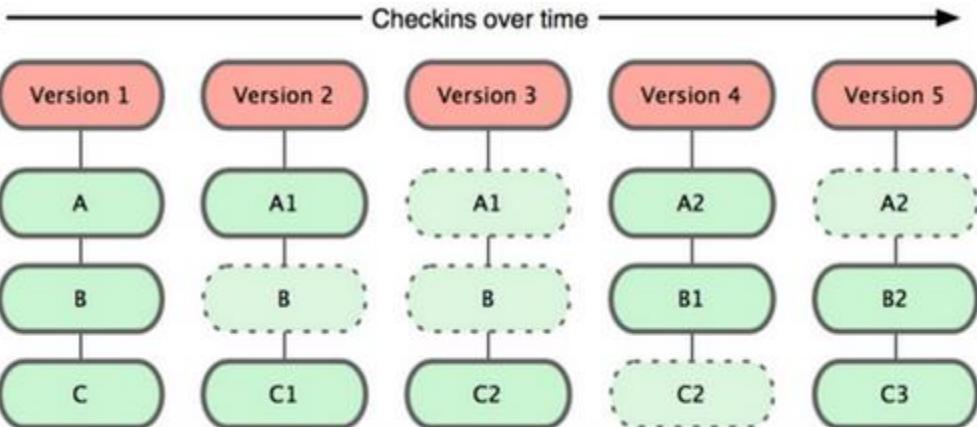
- Généralités sur les gestionnaires de version
 - **Présentation de Git**
 - Concepts généraux
 - ***Qu'est ce qu'un instantané ?***
 - Cycle de vie d'un fichier
 - Concept de branche
 - Création de branches
 - Branche et fusion
 - Merge et rebase
 - Serveur d'hébergement
 - Conclusion
-

Backend : orienté contenu

- La plupart des systèmes gèrent une liste de fichiers et de leurs modifications (**CVS, SVN, ...**)



- **Git** voit comme un instantané (snapshot) d'un mini système de fichier. A chaque validation de l'état du projet, Git prend un **instantané** du contenu de l'espace de travail et enregistre une référence à cet instantané.



Instantané

□ Git gère l'intégrité

- Git enregistre en BD locale une **succession d'instantanés**
- Un instantané est **unique** et identifié par une somme de contrôle ou empreinte SHA-1 composée de 40 caractères hexadécimaux et son **auteur**, son **horodatage** et son **commentaire**

Working Directory - C:\Users\

- Working Directory - C:\Users\
- .git
- mario
- .gitignore
- .project
- lisezmoi.txt

rep_git [f5b4348 - Rebase interactif]

C/commit	date	comment
9415dc1	tdo	1er commit de rep_clone
6c5b74f	tdo	2e commit: fichier lisezmoi a changé
	tdo	1er commit

commit a5eb70749ac9dbaec0d656ff6c0da5d3136342e1
Author: tdo <tdo.formation@gmail.com> 2015-07-14 09:53:29
Committer: tdo <tdo.formation@gmail.com> 2015-07-14 09:53:29
Parent: 4bb3eaf39451dbe054d174bdb7b96bcae2117044 (rep_bare: 2ème
Branches: master, origin/master

Horodatage

Karine Brifault

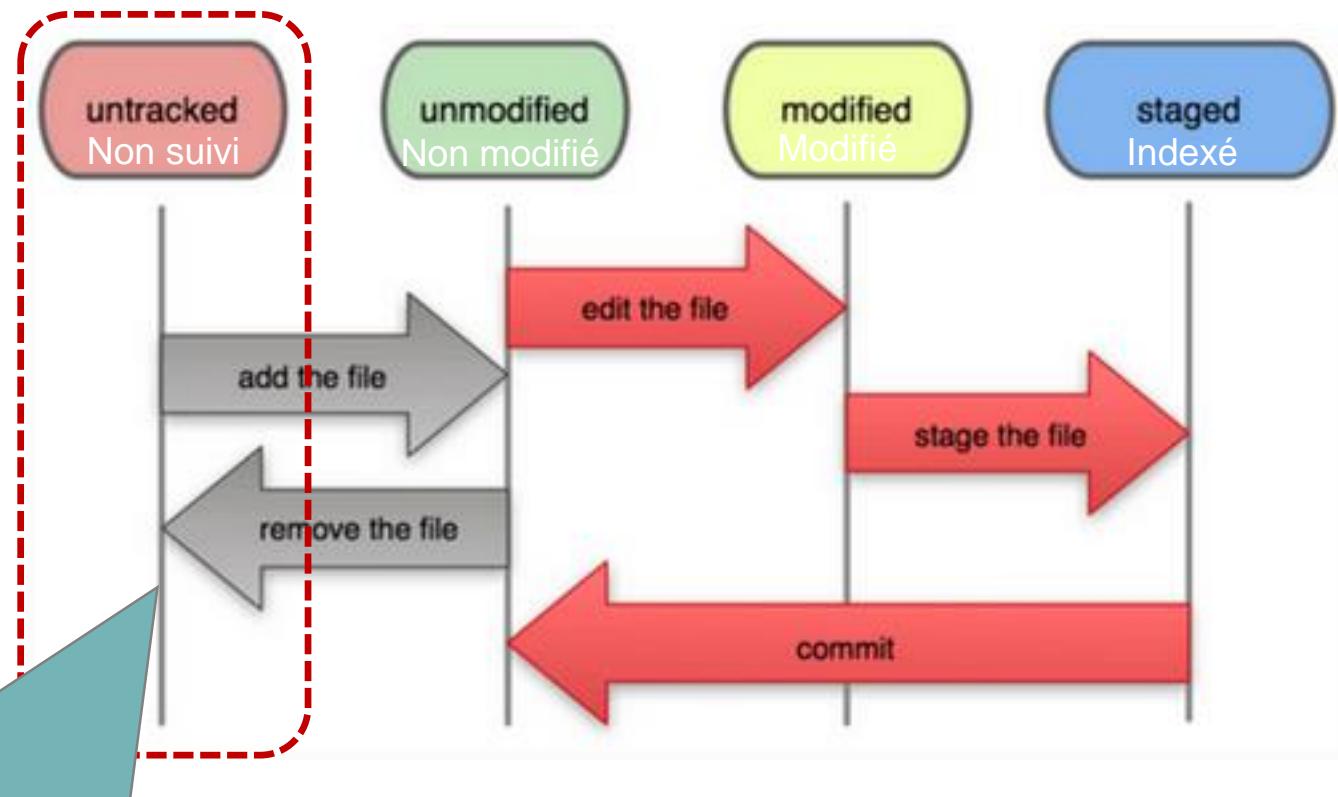
Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - Concepts généraux
 - Qu'est ce qu'un instantané ?
 - ***Cycle de vie d'un fichier***
 - Concept de branche
 - Création de branches
 - Branche et fusion
 - Merge et rebase
- Serveur d'hébergement
- Conclusion

Cycle de vie d'un fichier Git (1)

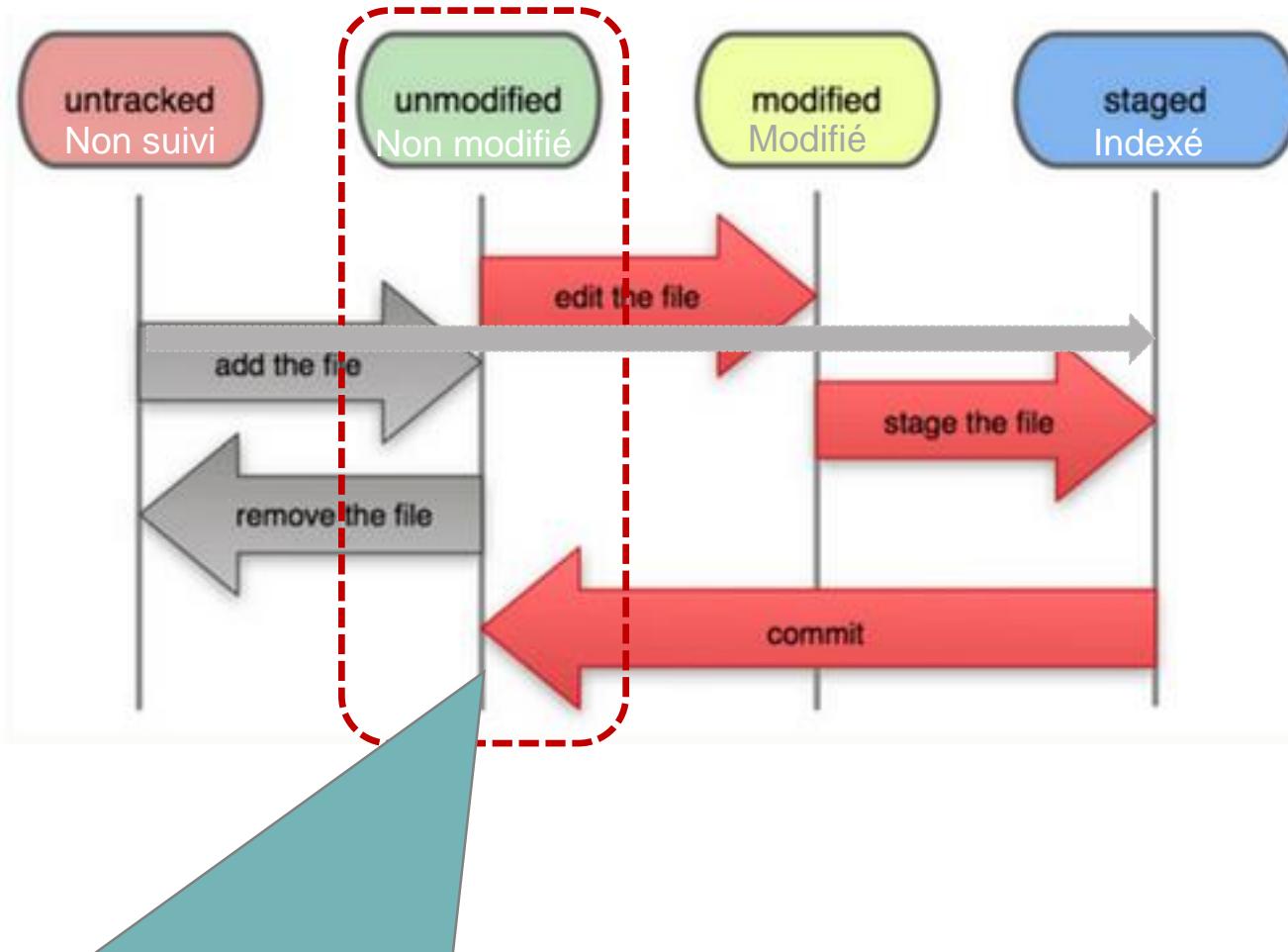
□ Chaque fichier peut avoir 2 états

- Sous suivi de version
- Non suivi

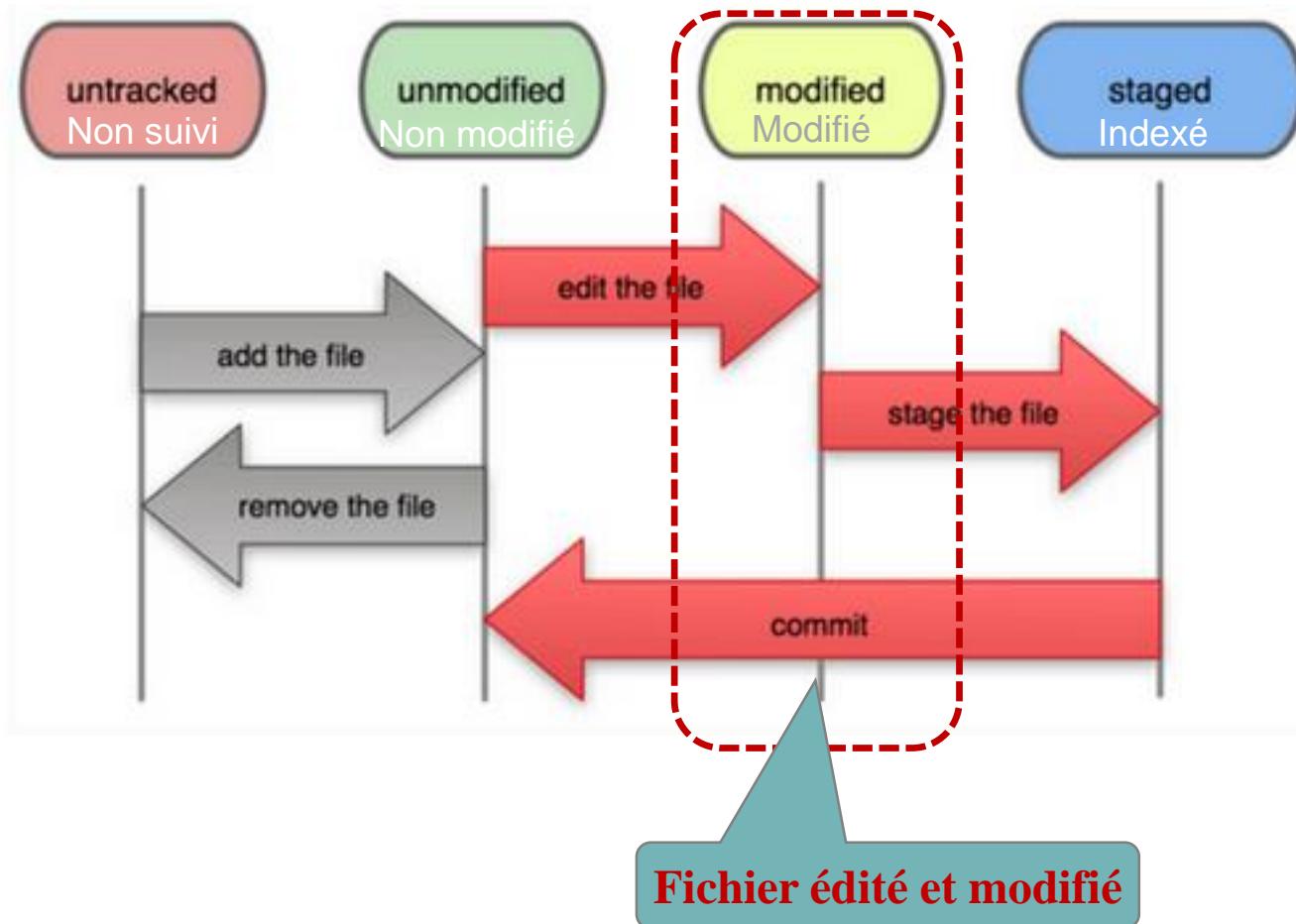


Nouveau Fichier : par défaut il est non suivi, à indexer

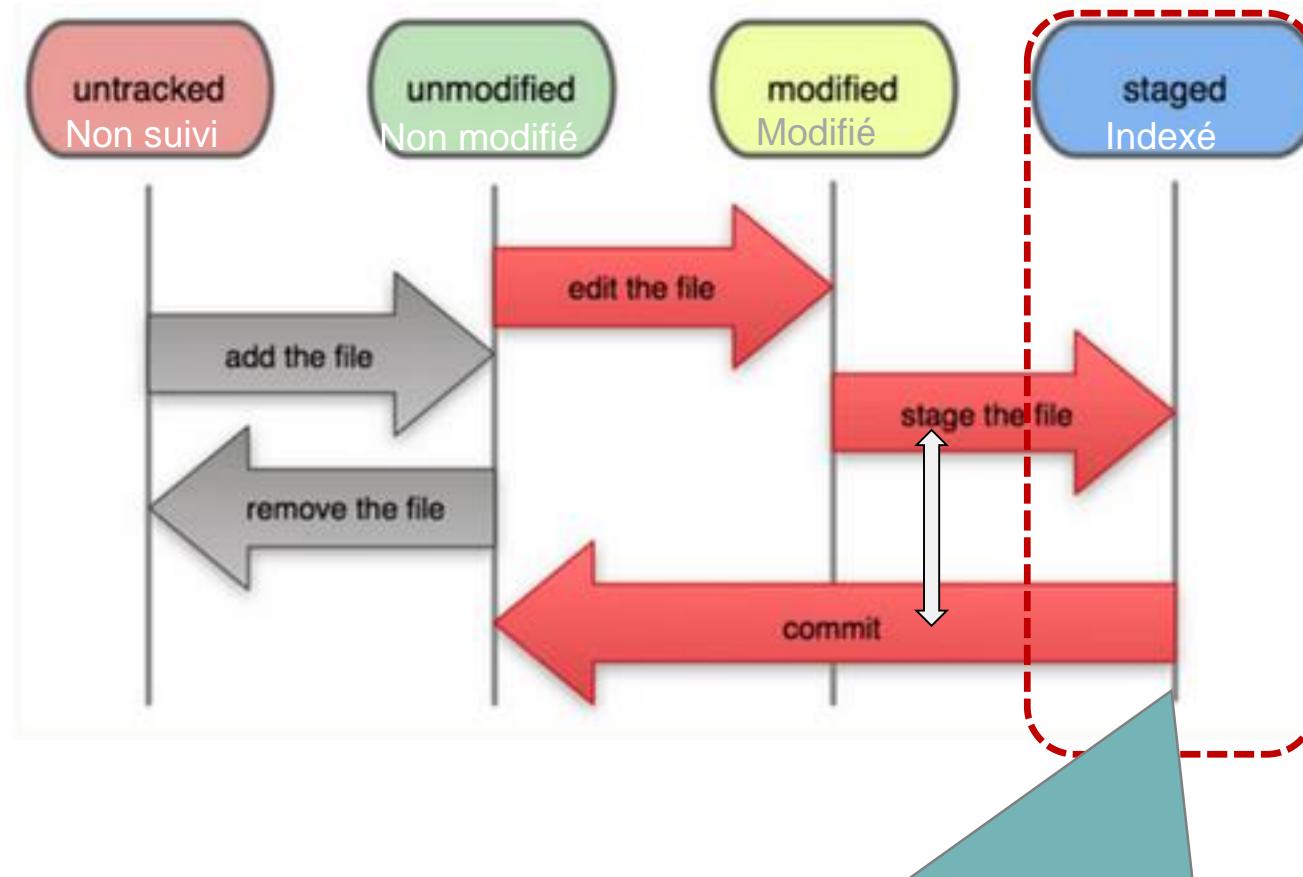
Cycle de vie d'un fichier Git (2)



Cycle de vie d'un fichier Git (3)

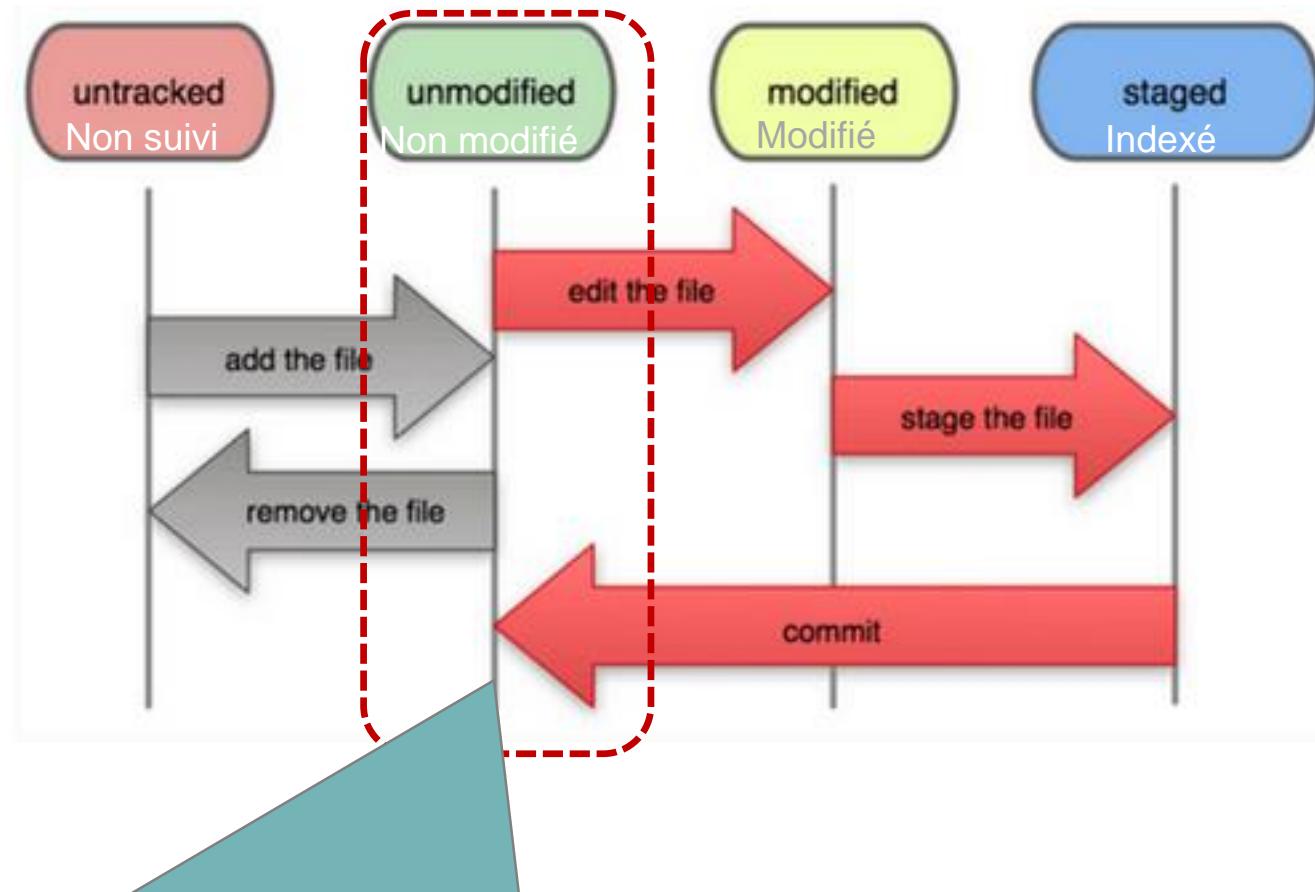


Cycle de vie d'un fichier Git (4)



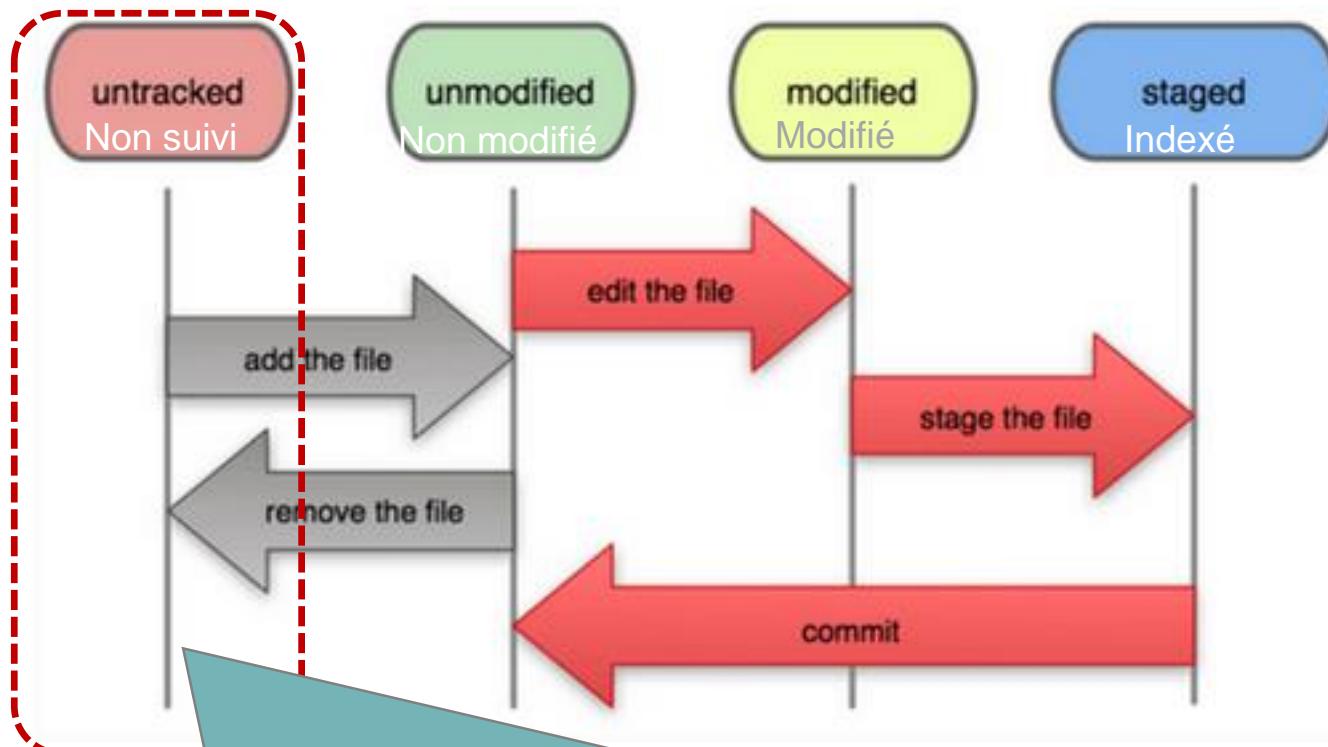
Fichier indexé, sa modification est prise en compte en local et prête à être mise dans le répertoire Git

Cycle de vie d'un fichier Git (5)



Fichier enregistré dans le répertoire Git, son statut est remis à non modifié

Cycle de vie d'un fichier Git (6)



Elimination du fichier dans le répertoire local, mais il reste dans le répertoire Git

Les commandes de base (1)

- Pour ajouter un fichier myfile au prochain commit :

\$ git add *myfile*

- Pour ajouter tous les fichiers créés ou modifiés au prochain commit :

\$ git add -A

- Pour retirer un fichier myfile ajouté avant de commiter :

\$ git reset HEAD *myfile*

- Pour commiter sur le dépôt git local :

\$ git commit -m "un commentaire utile"

- Pour modifier le commentaire joint au dernier commit si il n'est pas encore mis sur le serveur distant :

\$ git commit -- amend -m "nouveau commentaire"

- A tout moment il est possible de connaître le statut de nos fichiers :

\$ git status

Les commandes de base (2)

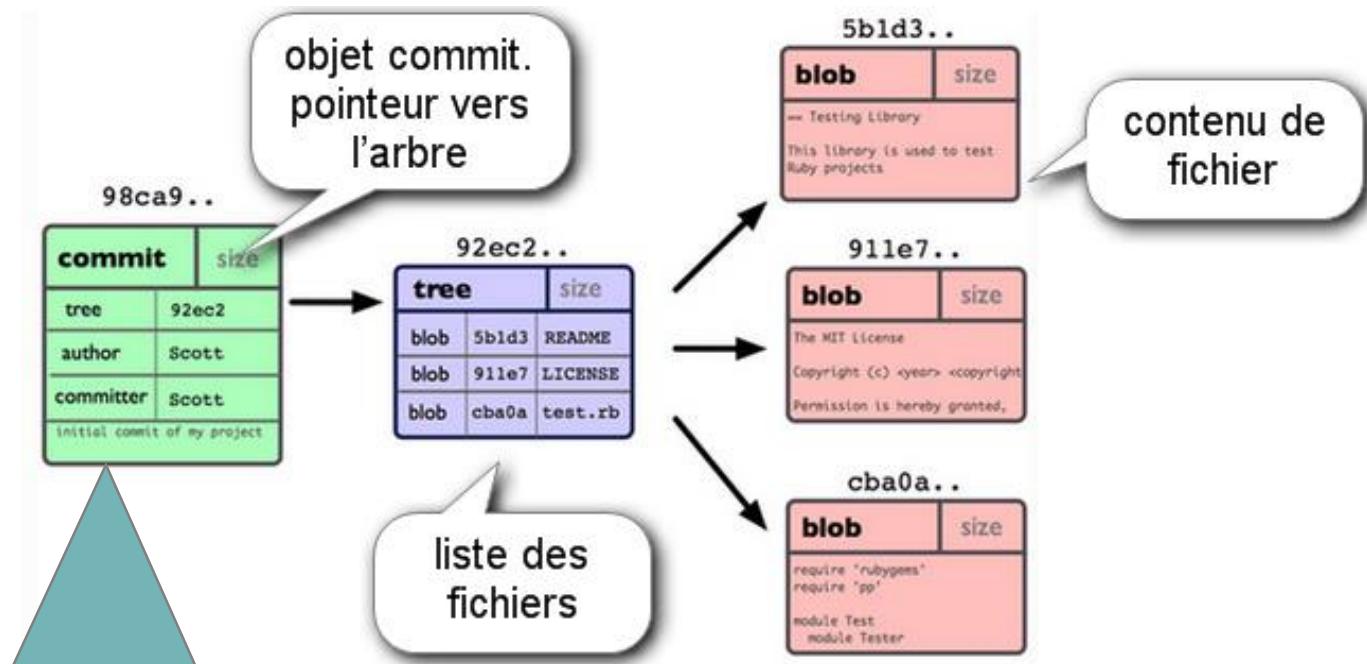
- Pour voir l'historique des commits :
\$ git log
- Avant un push, pour annuler tous les commits fait depuis la dernière synchronisation git avec le dépôt distant :
\$ git reset HEAD
- Pour récupérer un fichier myfile dans sa dernière version committée :
\$ git checkout -- myfile
- Pour récupérer les sources de la branche master du dépôt distant origin :
\$ git fetch
\$ git pull origin master
- Pour pousser ses commits de la branche master sur le dépôt distant origin :
\$ git push origin master
- Pour annuler un commit après un push si personne ne l'a récupéré :
\$ git revert idDuCommit
\$ git push
- Pour annuler les N=5 derniers commits après un push :
\$ git revert HEAD~5 ; git push

Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - Concepts généraux
 - Qu'est ce qu'un instantané ?
 - Cycle de vie d'un fichier
 - ***Concept de branche***
 - Création de branches
 - Branche et fusion
 - Merge et rebase
- Serveur d'hébergement
- Conclusion

Concept de branche : la base (1)

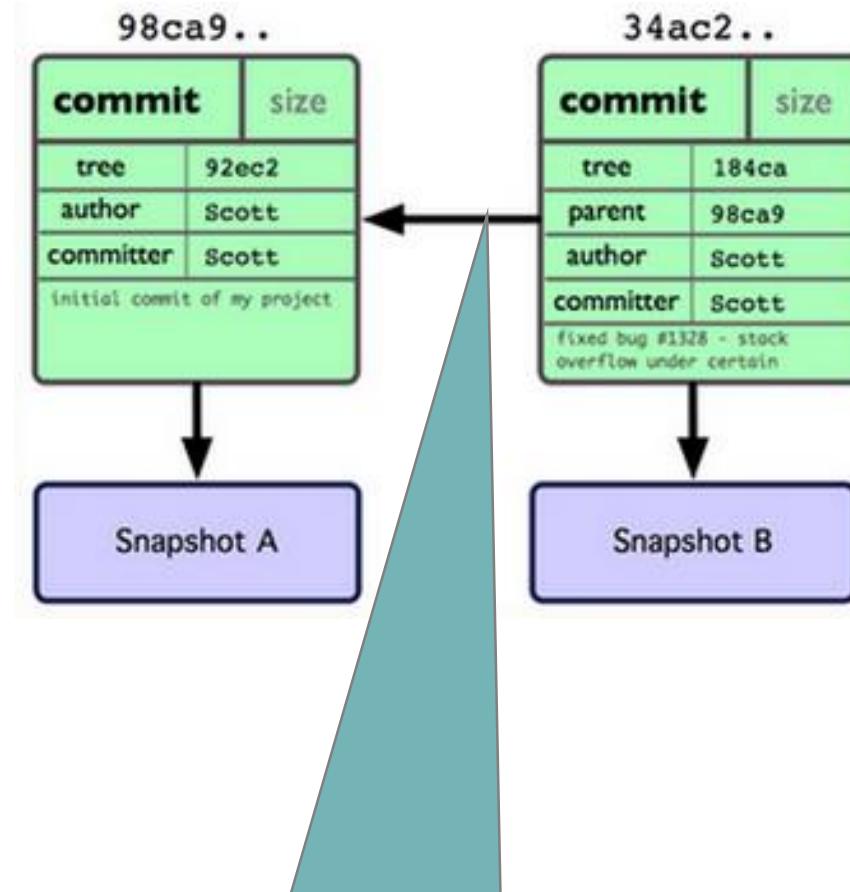
- Un **instantané** d'un projet (5 objets)



« Objet Commit »
qui contient des métadonnées
et un pointeur vers l'arbre racine

Concept de Branche : la base (2)

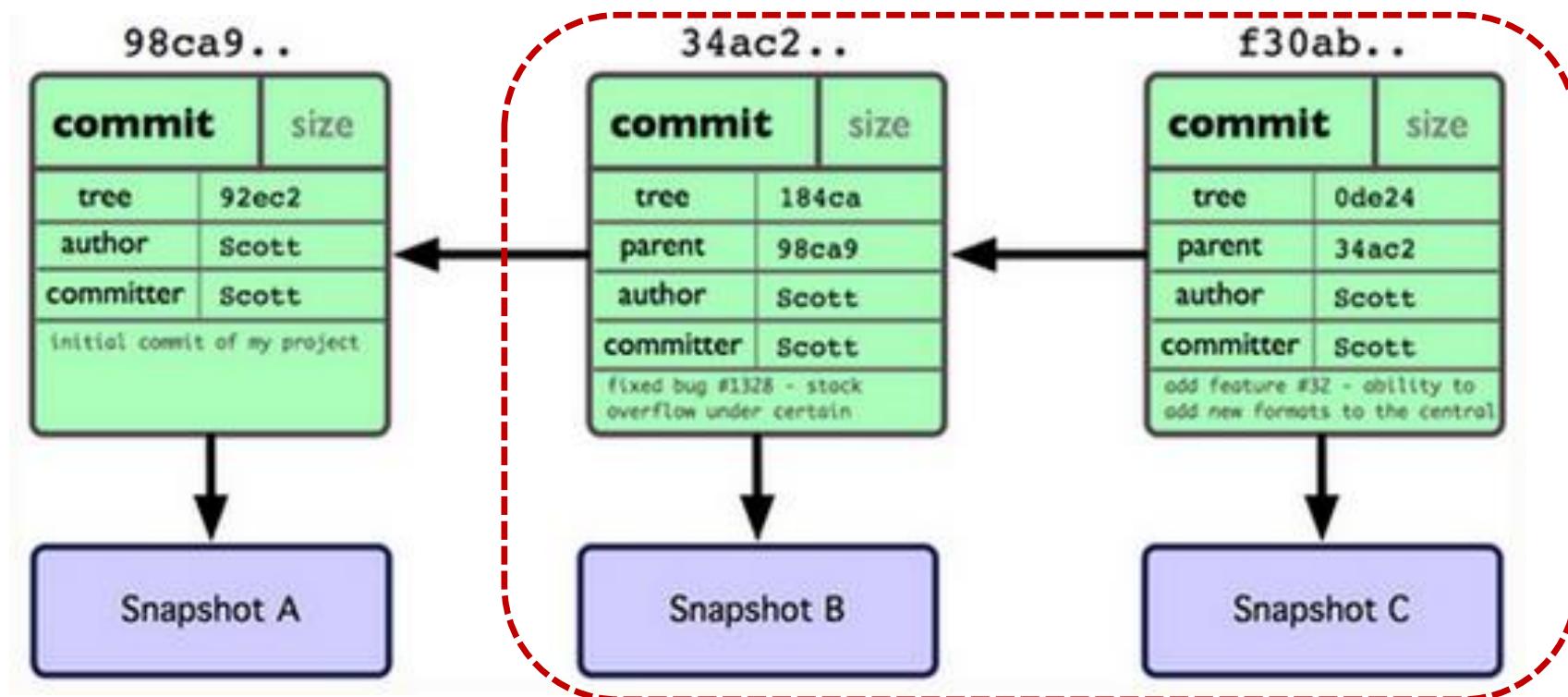
- Si on fait, une modification et que l'on valide



Lien entre 2 instantanés :
le précédent `98ca9` et le courant `34ac2`

Concept de Branche : la base (3)

- Si on fait à nouveau une modification et une validation

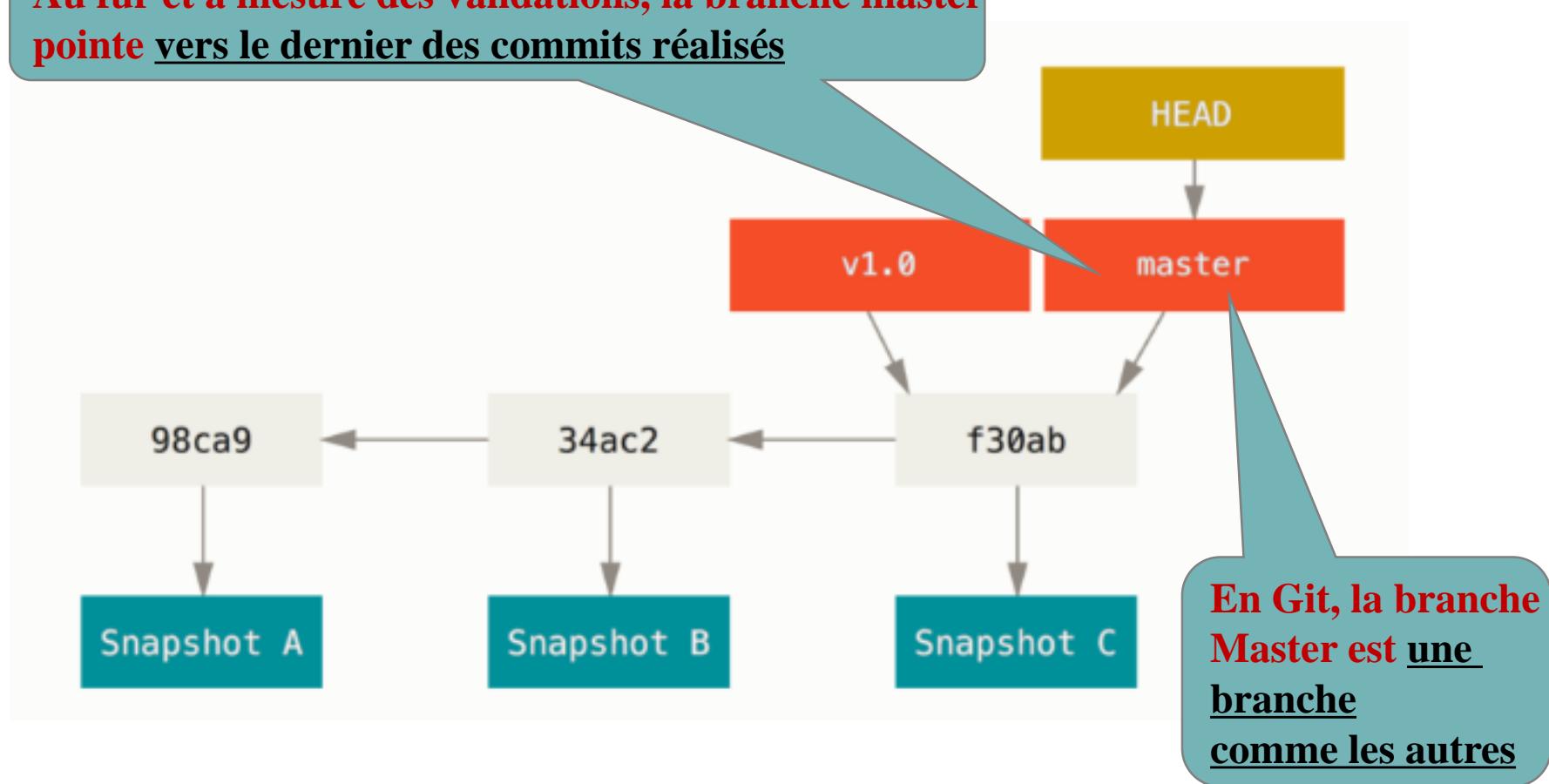


Suivi des modifications sur la branche par défaut qui s'appelle master

Concept de Branche : la base (4)

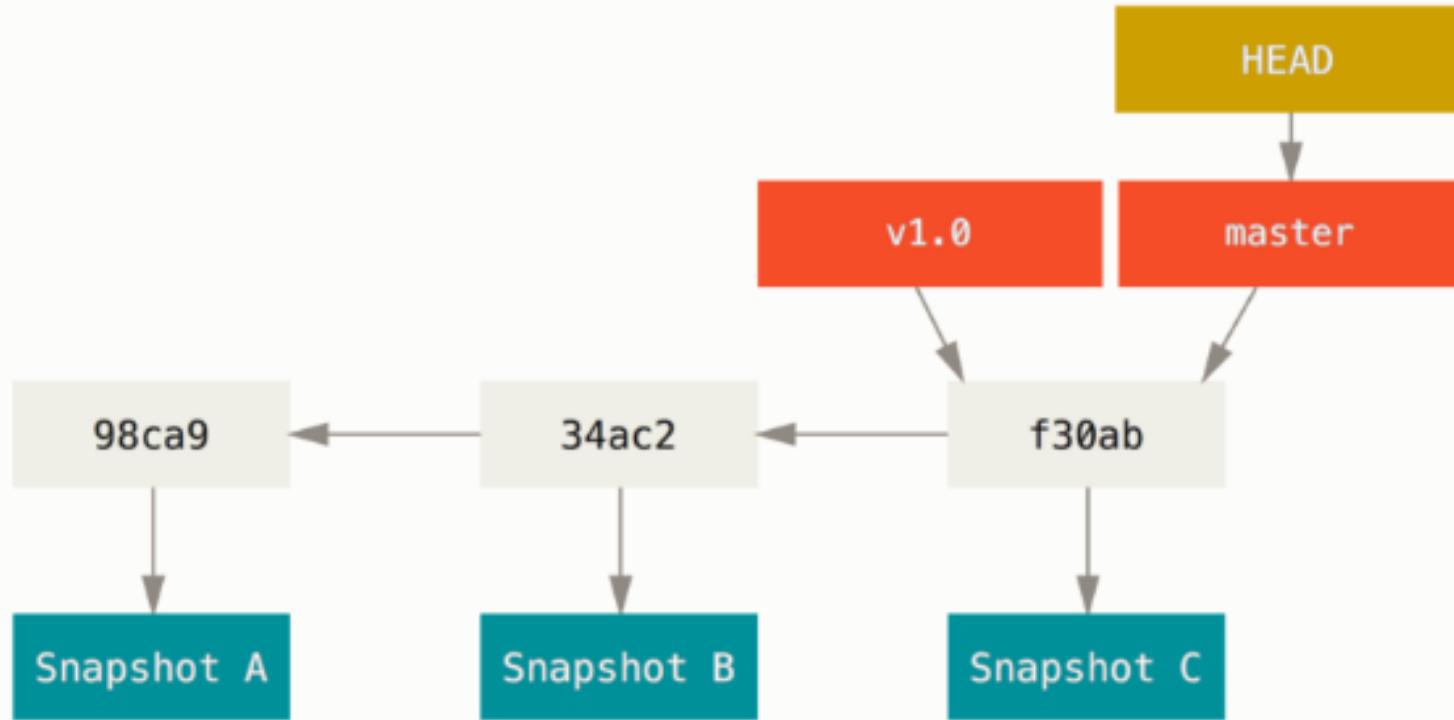
□ La branche par défaut « Master »

Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés



Concept de Branche : la base (5)

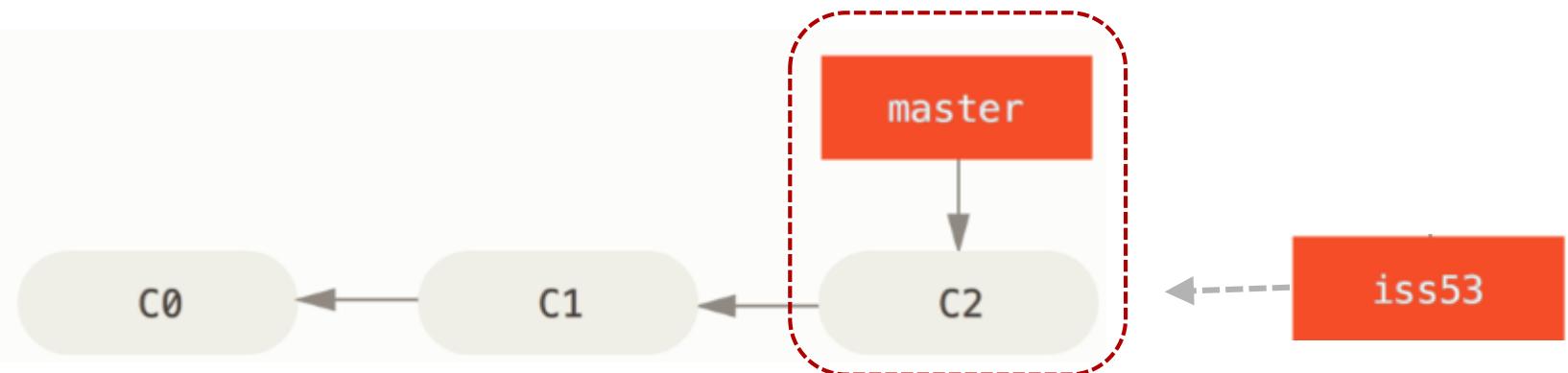
Une branche dans Git est simplement un pointeur léger et déplaçable vers l'un des commits !!!



Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - Concepts généraux
 - Qu'est ce qu'un instantané ?
 - Cycle de vie d'un fichier
 - Concept de branche
 - ***Création de branches***
 - Branche et fusion
 - Merge et rebase
- Serveur d'hébergement
- Conclusion

Une nouvelle branche : exemple



- Les opérations de base sont : **branch** et **checkout**

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

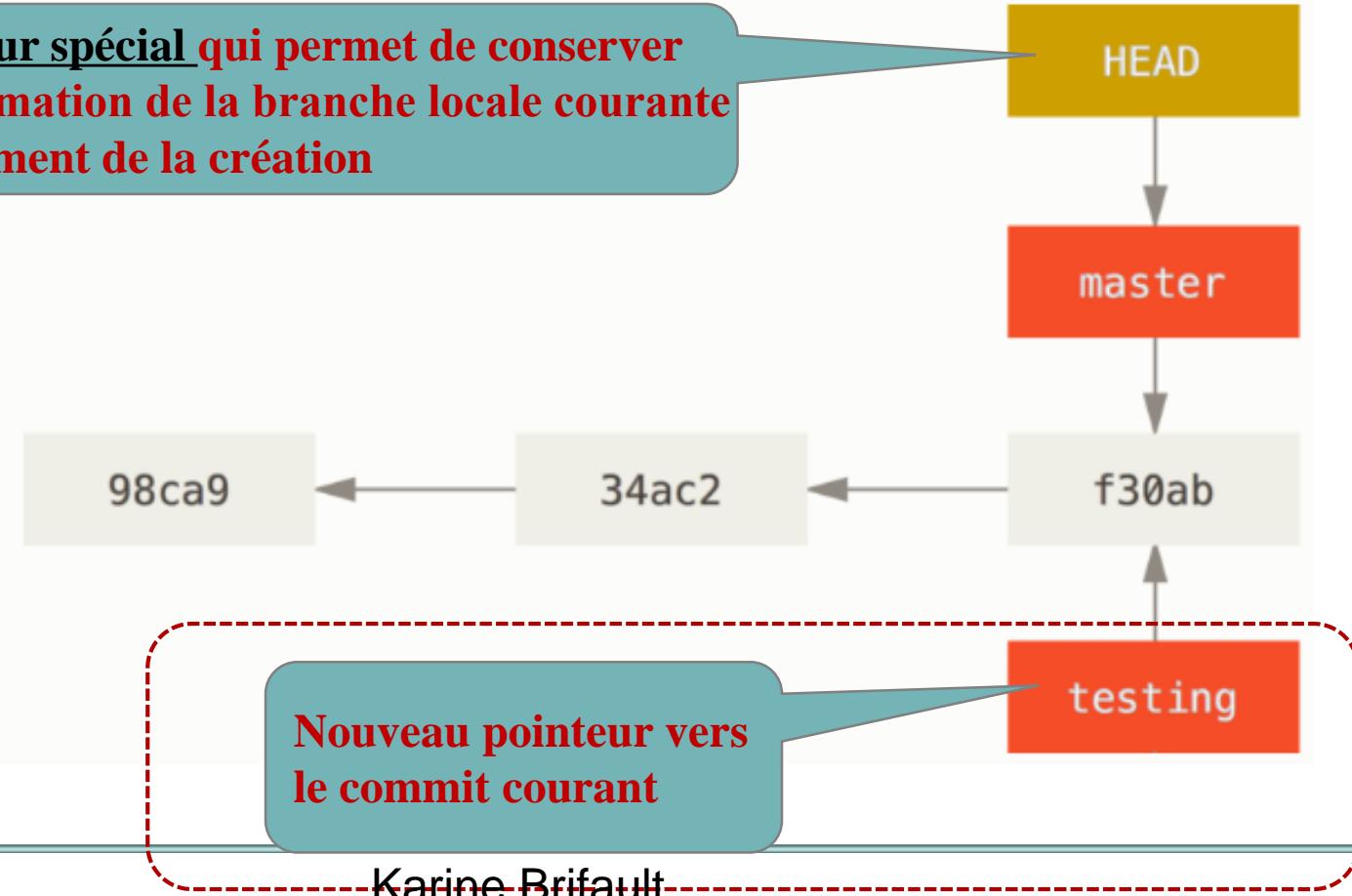


```
$ git checkout -b iss53  
Switched to a new branch "iss53"
```

Création d'une nouvelle branche (1)

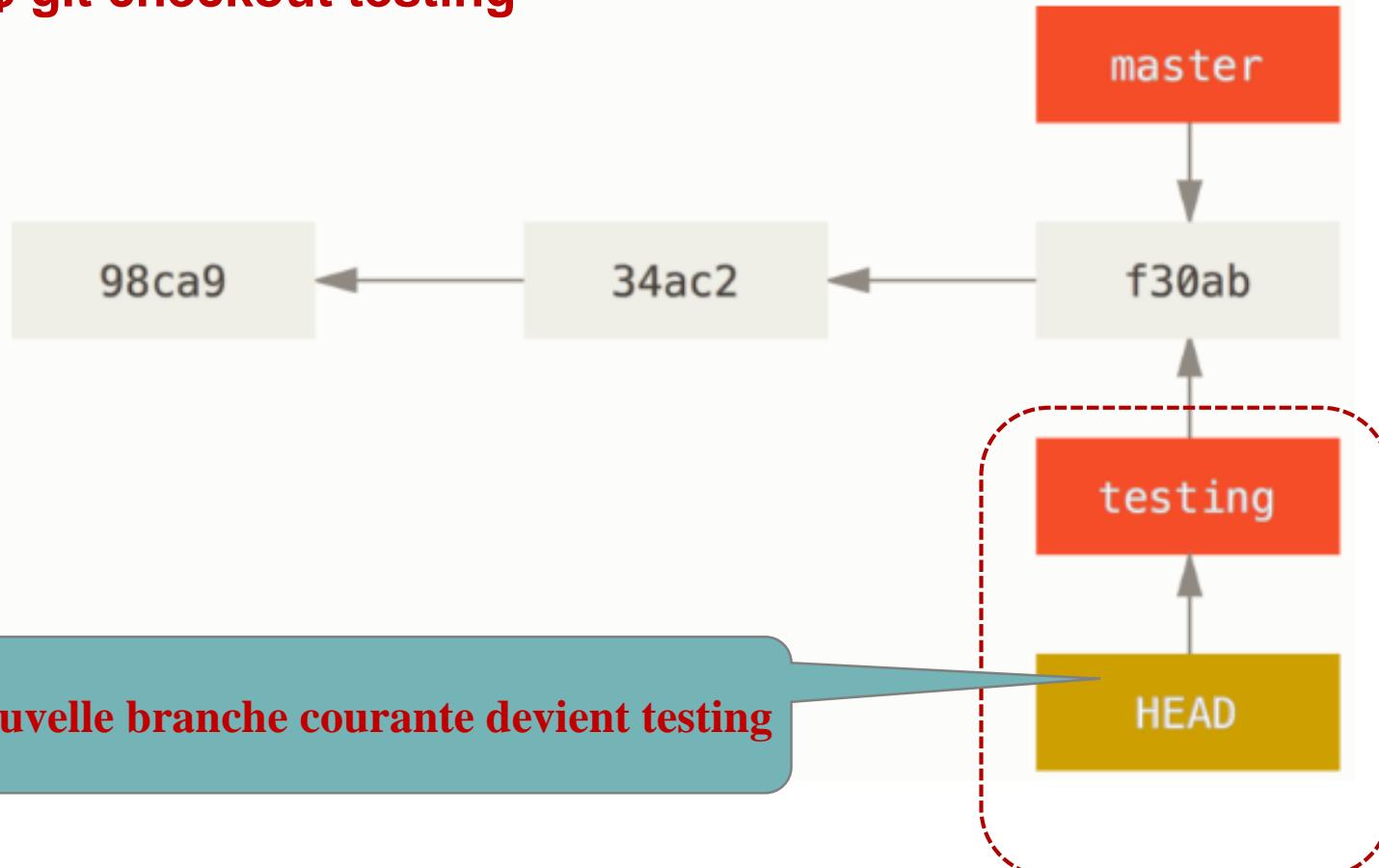
- On crée une nouvelle branche « testing »
\$ git branch testing

Pointeur spécial qui permet de conserver
l'information de la branche locale courante
au moment de la création



Création d'une nouvelle branche (2)

- On bascule vers la nouvelle branche « testing »
\$ git checkout testing

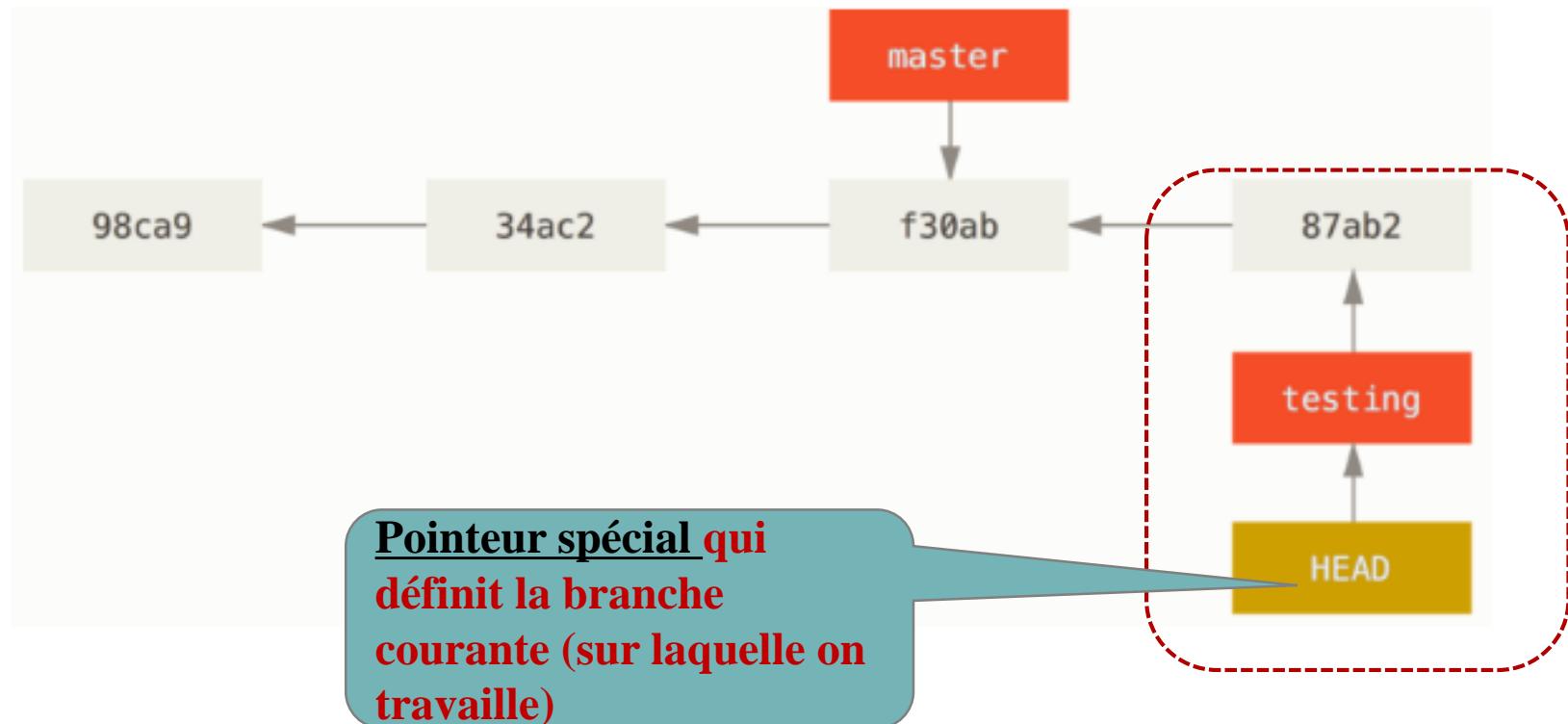


Création d'une nouvelle branche (3)

- On fait des modifications puis on « committe »

\$ git commit -a -m 'une modif qui fait...'

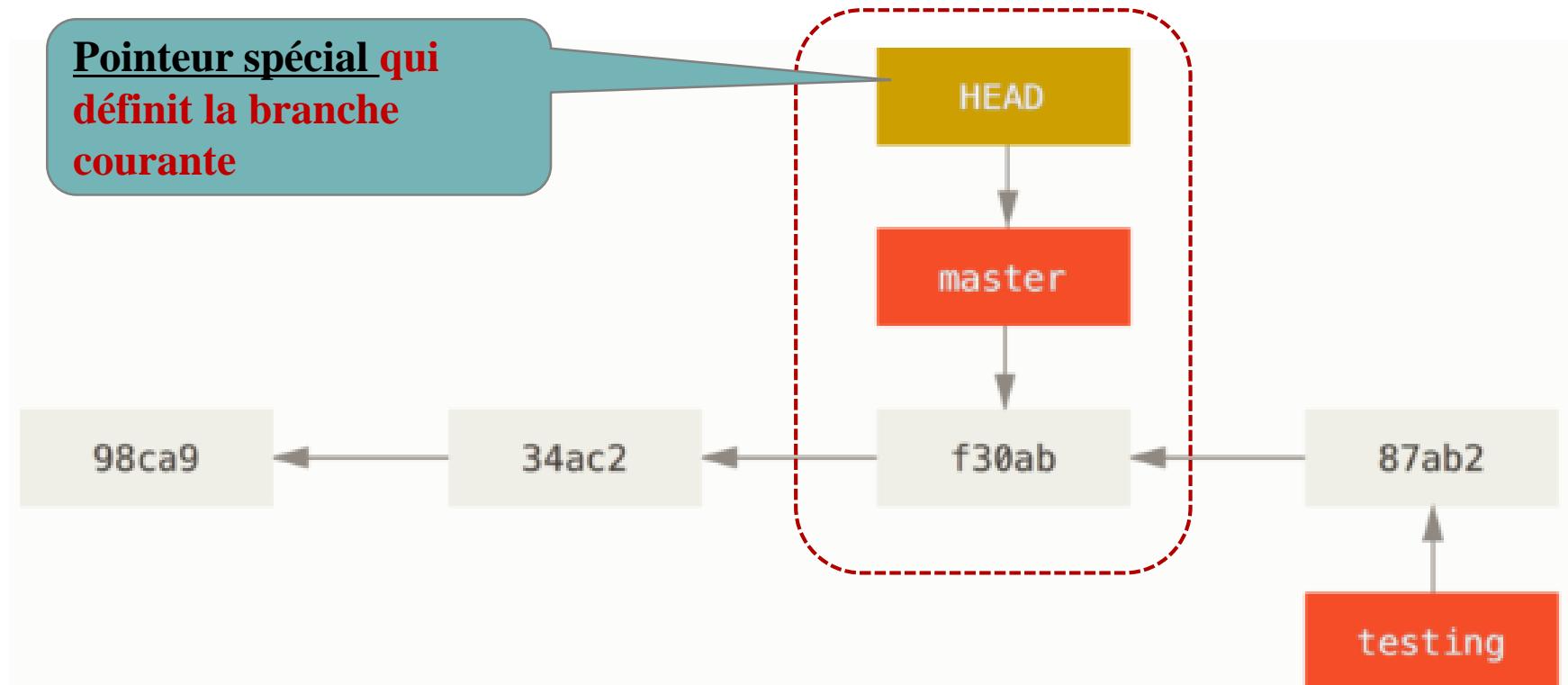
Avec le commentaire qui va bien !!!



Création d'une nouvelle branche (4)

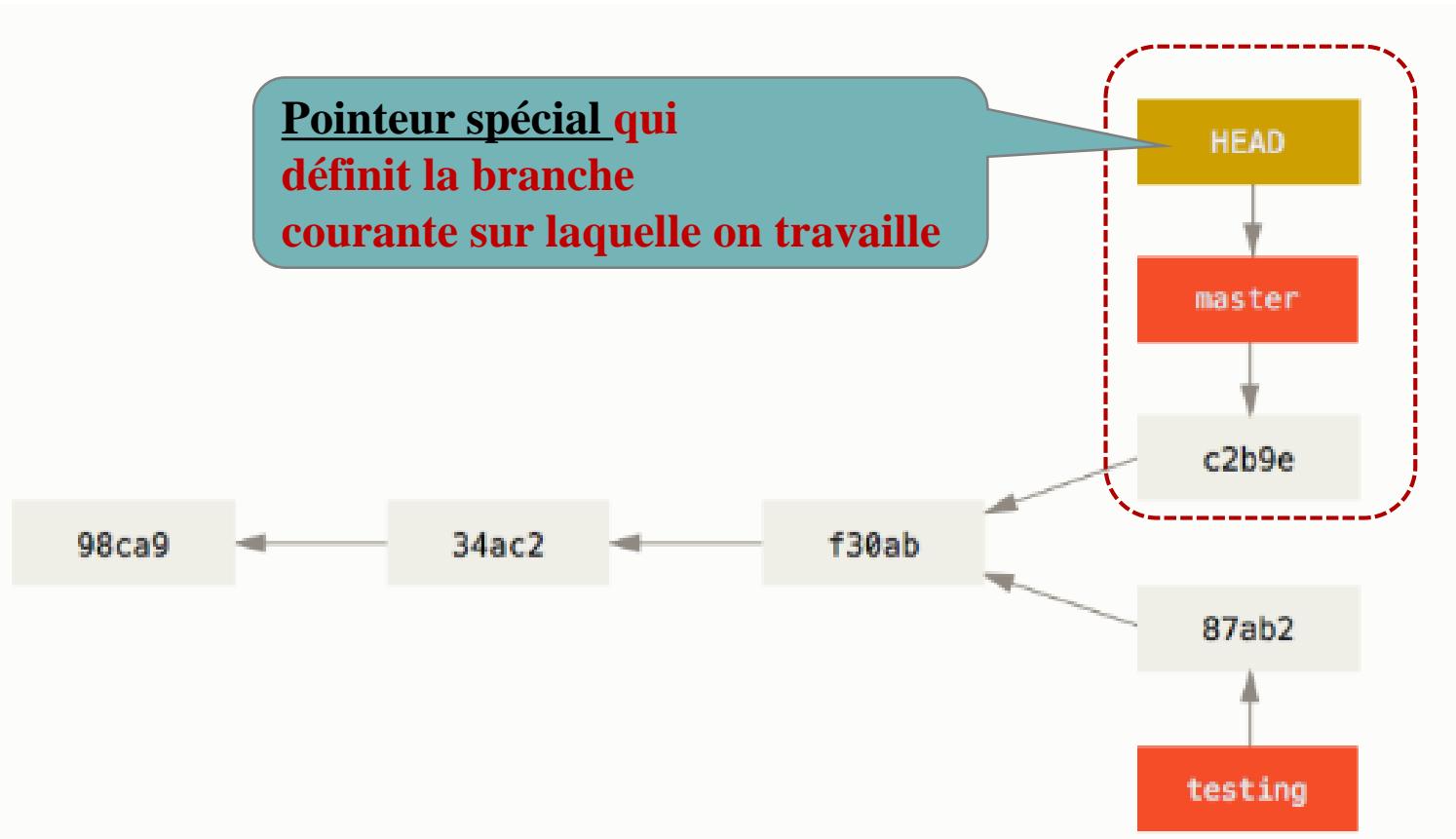
- On peut **revenir à la branche Master** pour faire des modifications en jouant avec le pointeur spécial HEAD

\$ git checkout master



Création d'une nouvelle branche (5)

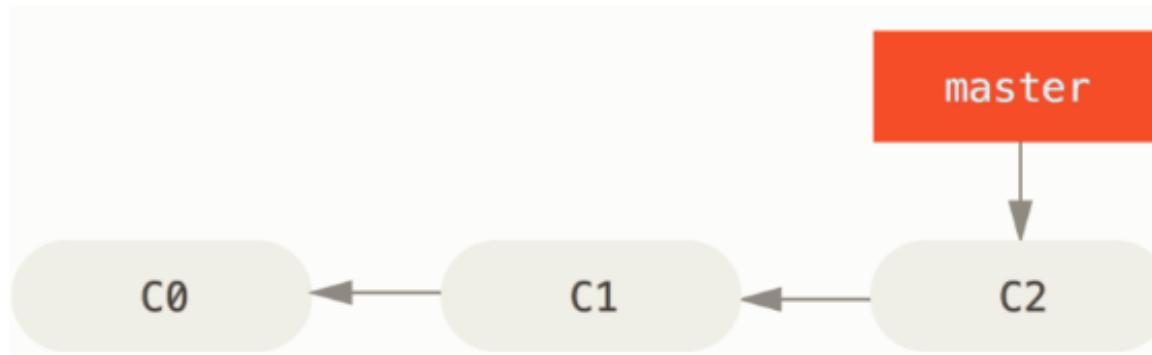
- Et, on peut à nouveau faire des modifications sur la branche master



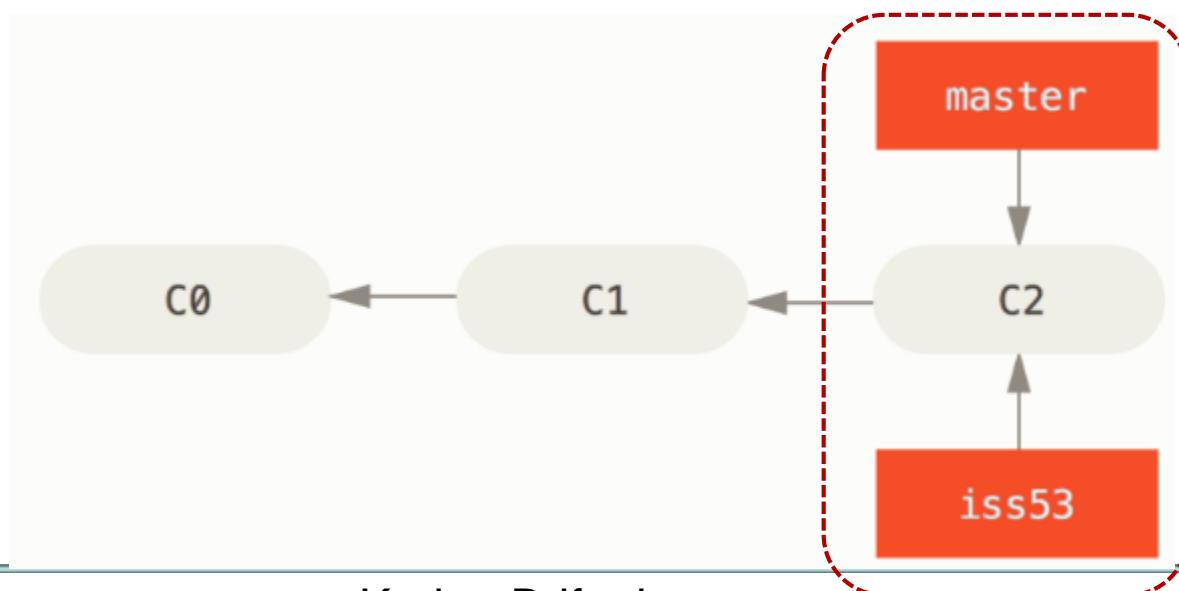
Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - Concepts généraux
 - Qu'est ce qu'un instantané ?
 - Cycle de vie d'un fichier
 - Concept de branche
 - Création de branches
 - ***Branche et fusion***
 - Merge et rebase
- Serveur d'hébergement
- Conclusion

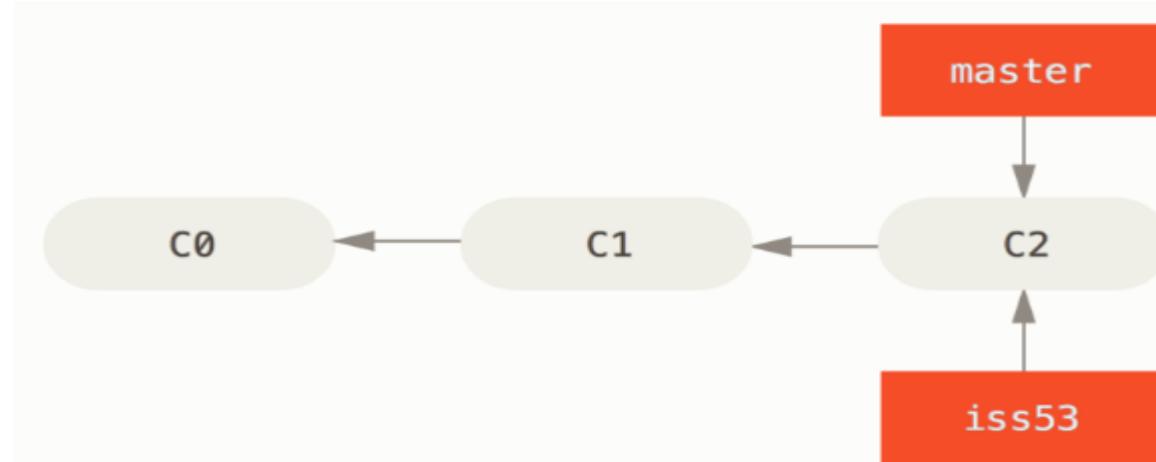
Branches et fusions : les bases (1)



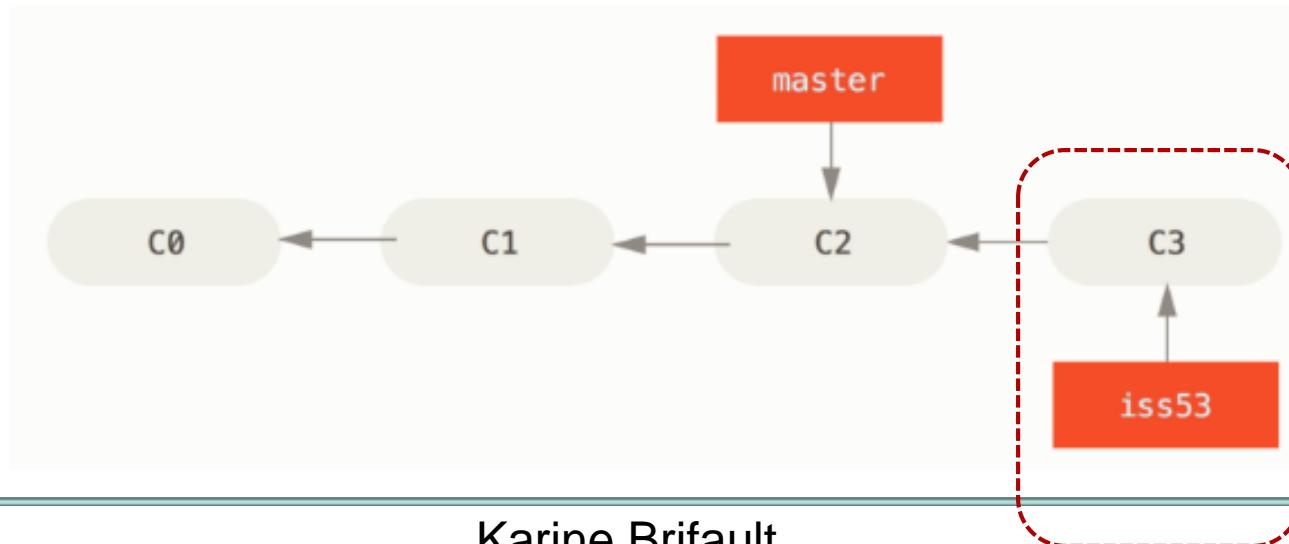
- On crée une branche **iss53** pour corriger un problème à partir du Master : **\$ git checkout -b iss53**



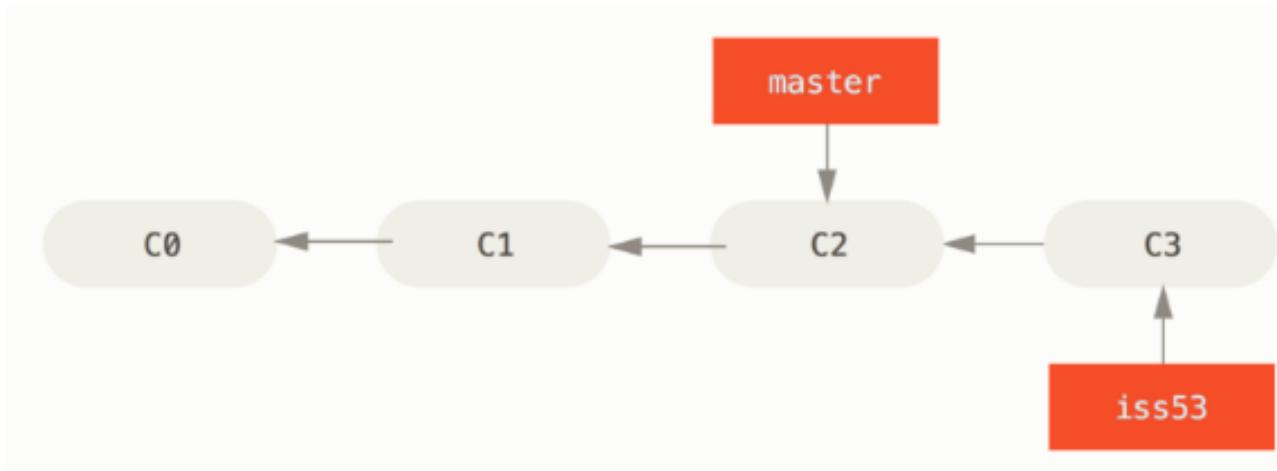
Branches et fusions : les bases (2)



- On corrige le problème sur la branche iss53 nouvellement créée et on committe : **\$ git commit –a –m 'blabla UTILE'**

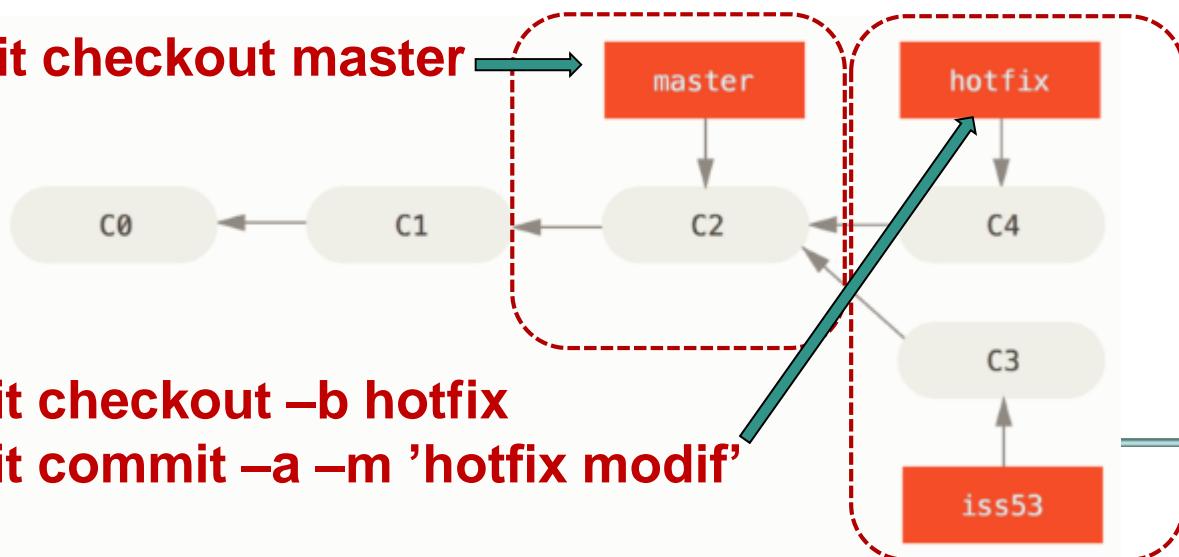


Branches et fusions : les bases (3)



- On peut alors directement rebasculer sur la branche master sans s'occuper de déployer les modifications validées sur iss53. Et, on peut revenir sur master et faire une nouvelle **branche hotfix**.

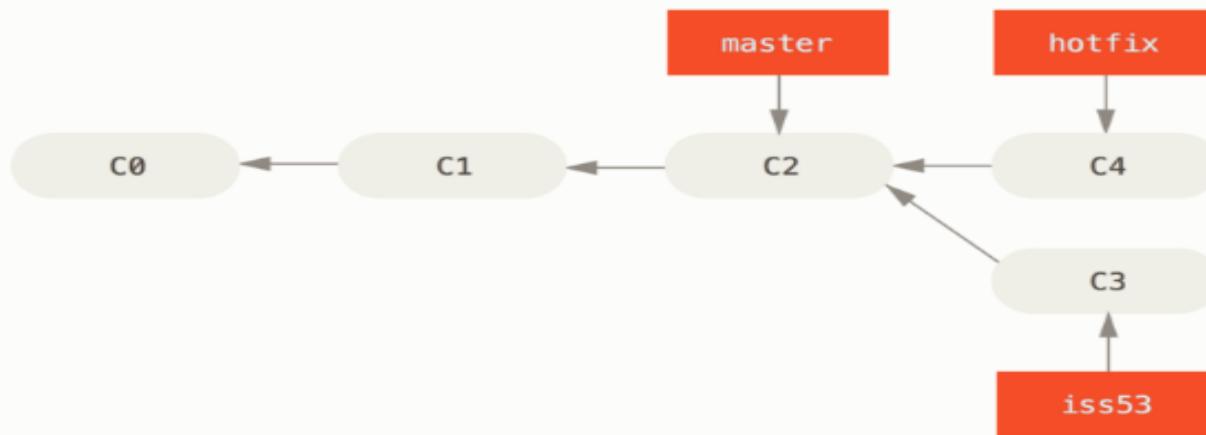
\$ git checkout master



\$ git checkout -b hotfix

\$ git commit -a -m 'hotfix modif'

Branches et fusions : les bases (4)

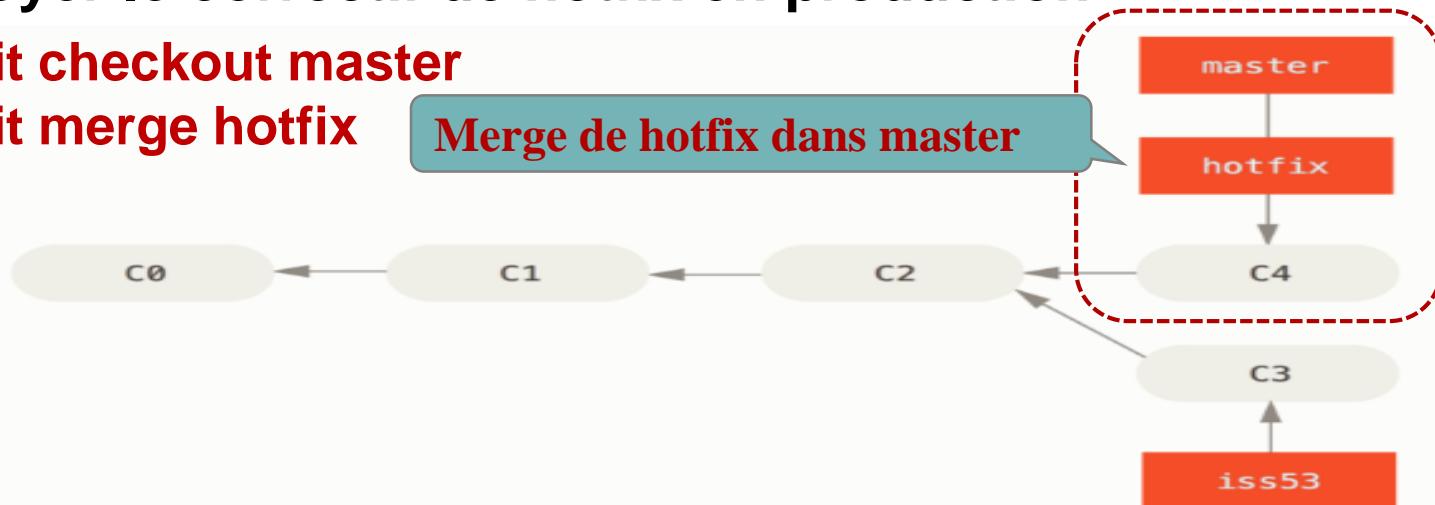


- On peut alors **fusionner les branches master et hotfix** pour déployer le correctif de hotfix en production.

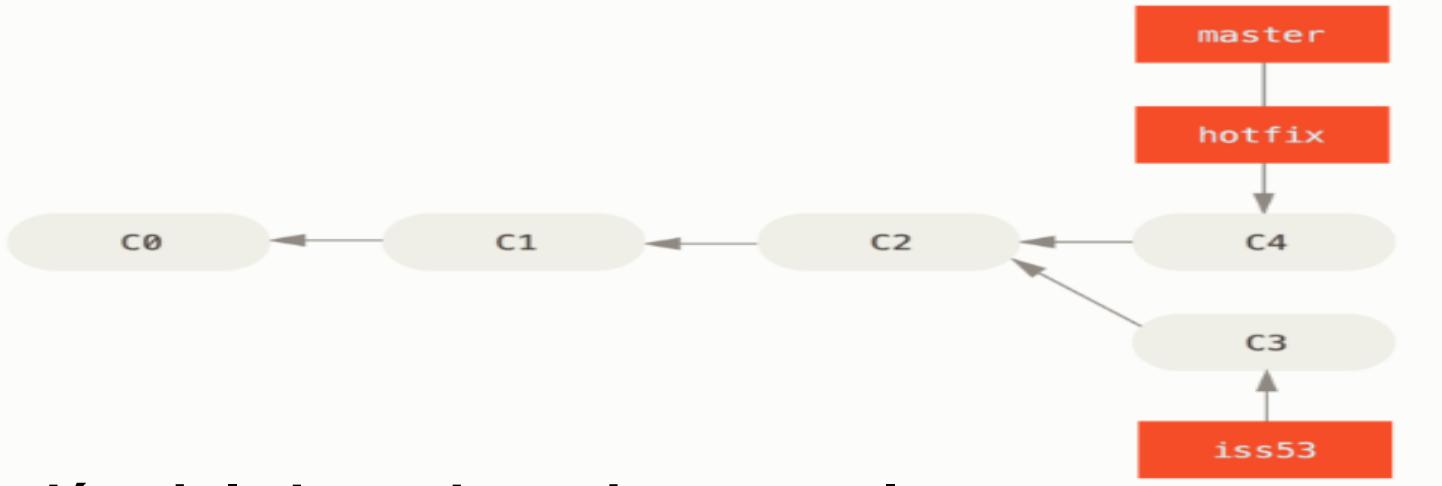
\$ git checkout master

\$ git merge hotfix

Merge de hotfix dans master

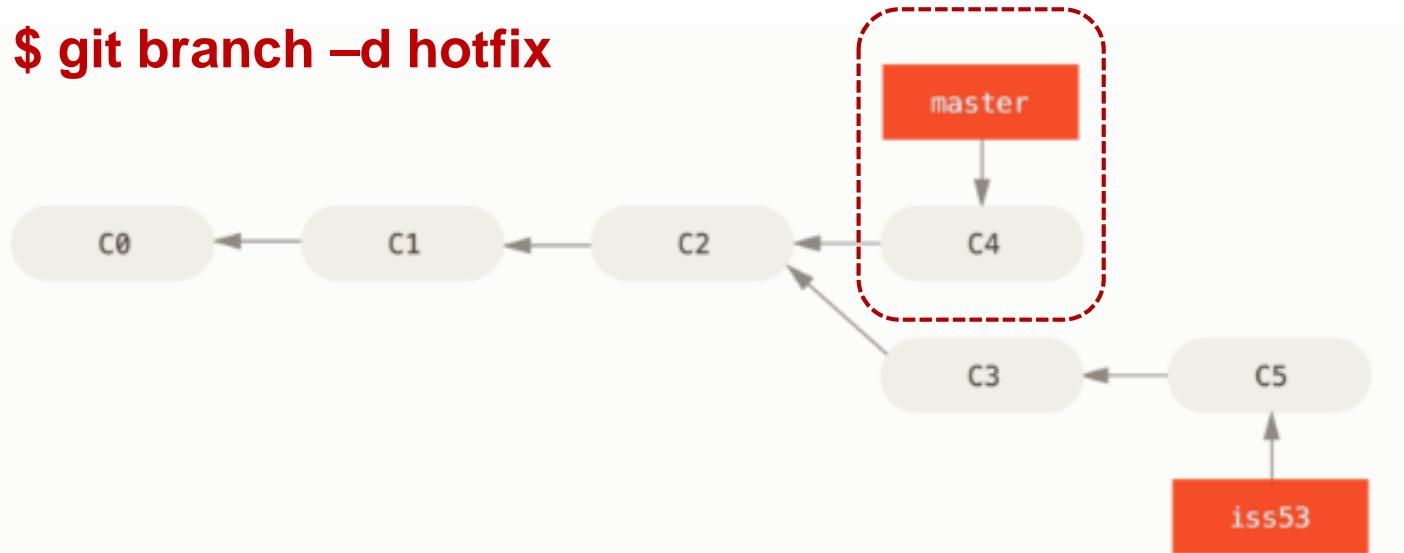


Branches et fusions : les bases (5)

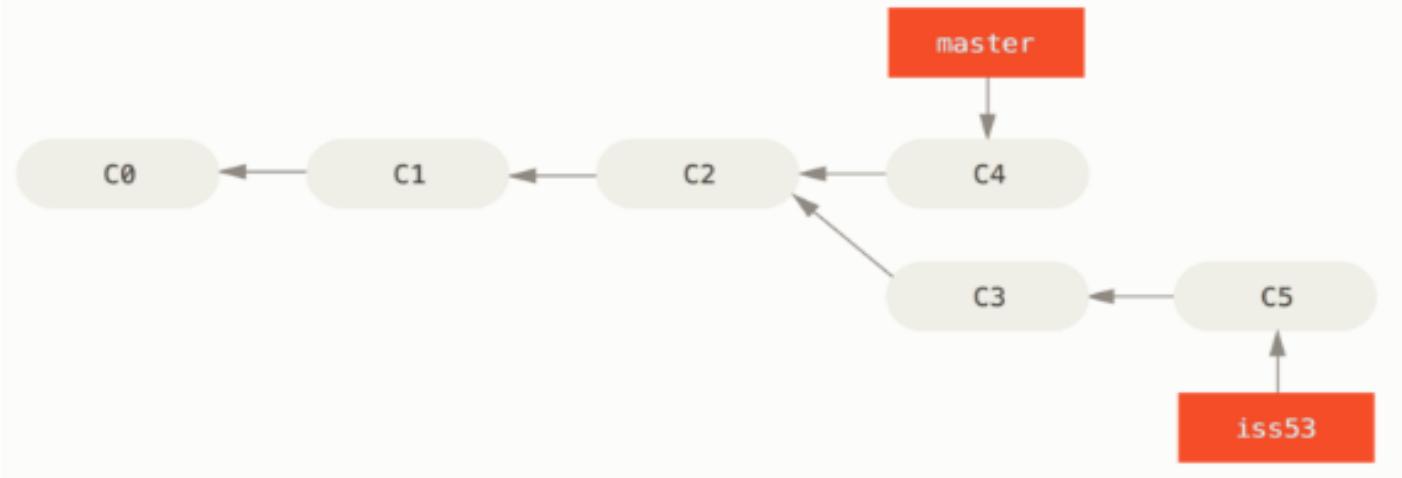


- On détruit la branche qui ne sert plus.

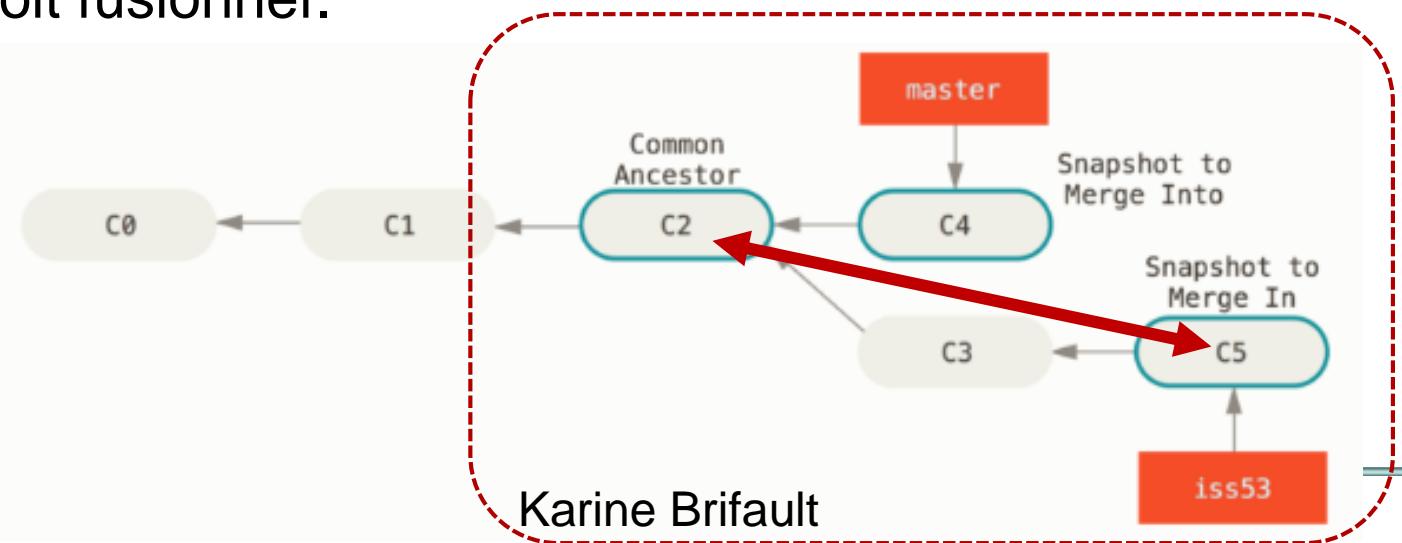
\$ git branch -d hotfix



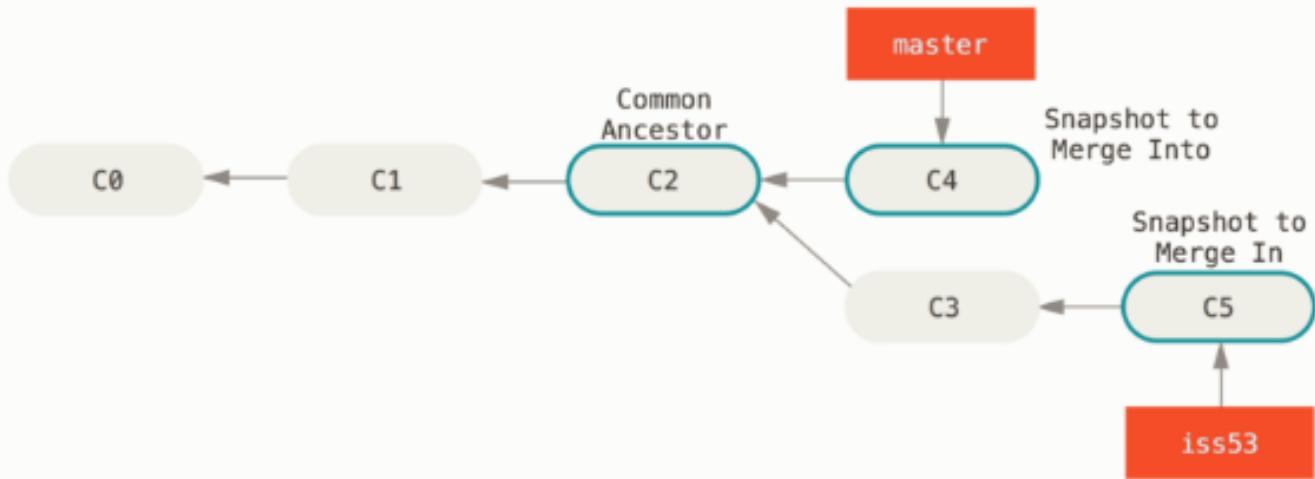
Branches et fusions : les bases (6)



- L'historique de développement a divergé depuis C2 et le commit que l'on veut merger n'est plus un ancêtre direct de la branche qui doit fusionner.



Branches et fusions : les bases (7)



- Commit automatique de GIT avec une fusion à 3 branches.

\$ git checkout master

\$ git merge iss53

\$ git branch -d iss53

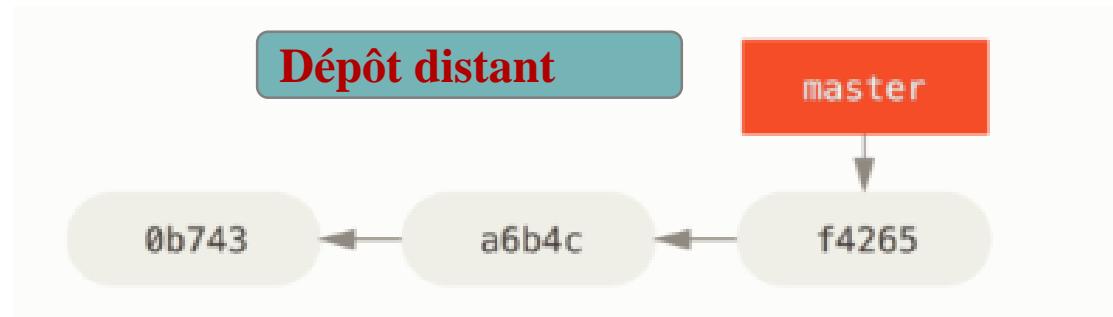
Commit de fusion qui
comporte plus d'un parent



Automatique dans le meilleur des cas !
Dans le cas contraire, le merge est fait
manuellement avec git mergetool

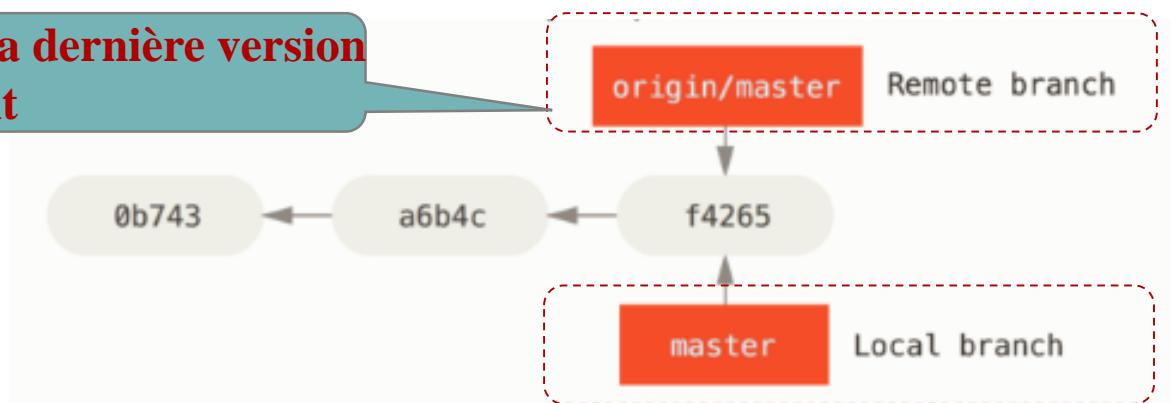
Branches distantes (1)

- Ce sont des références à l'état des branches sur votre **dépôt distant**.



\$ git clone dev1@serveur_distant:projet.git

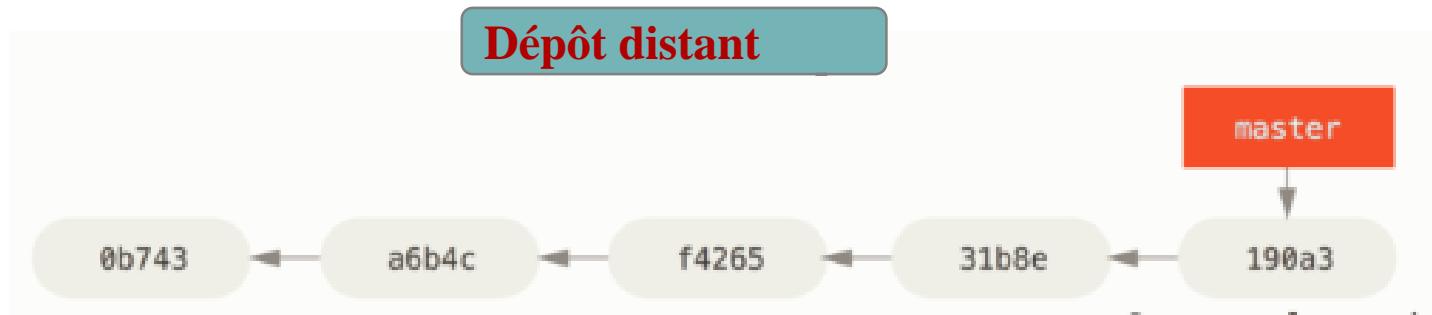
Nom par défaut de la dernière version sur le serveur distant



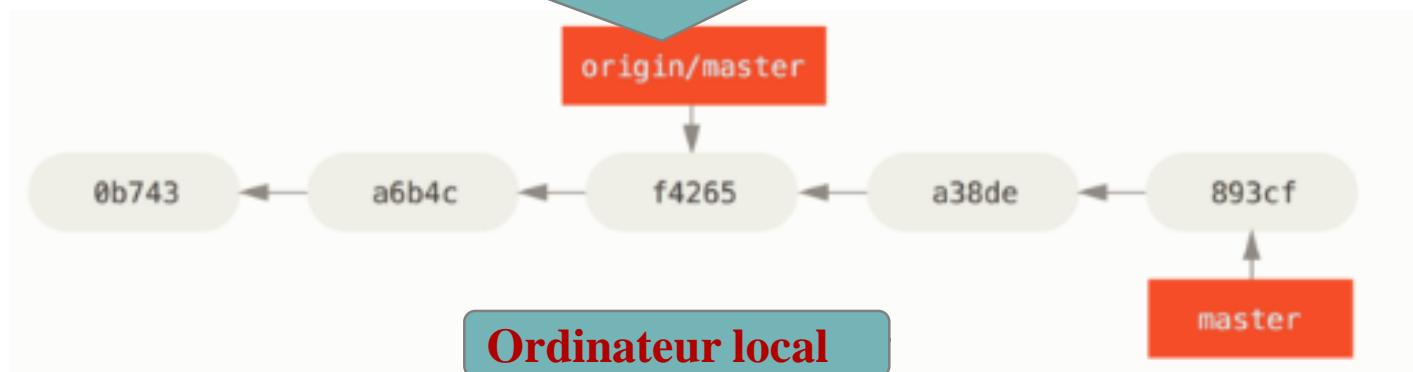
Ordinateur local

Branches distantes (2)

- Quelqu'un travaille sur le dépôt distant



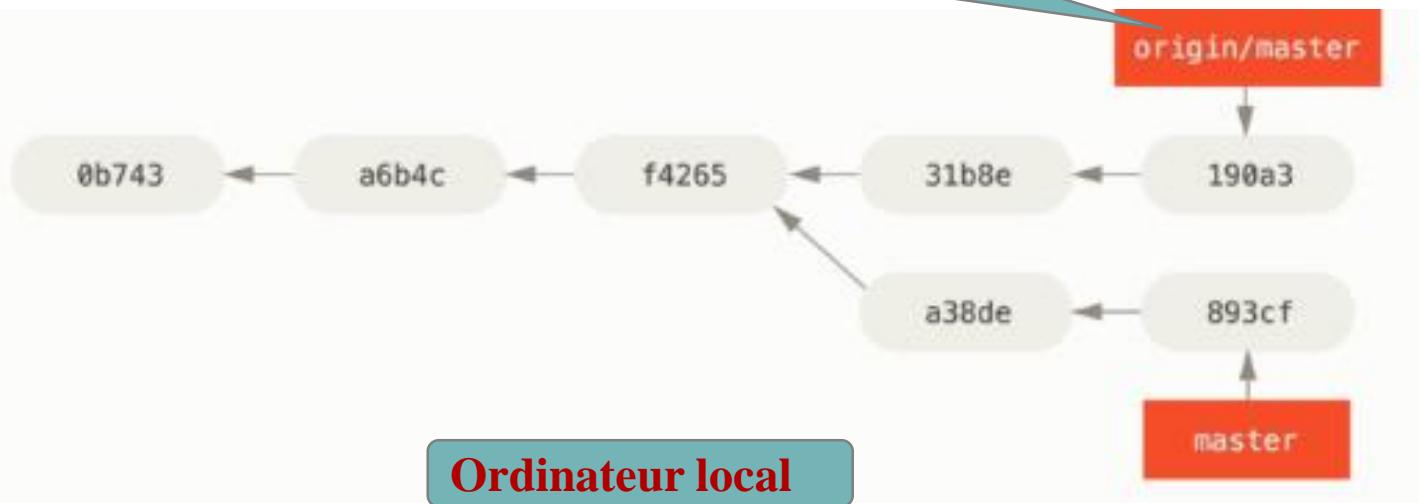
Sans synchronisation les infos du dépôt distant sont périmés !



Branches distantes (3)

- On synchronise le travail par rapport au dépôt distant
\$ git fetch [serveur_distant]

Fetch met à jour les références distantes



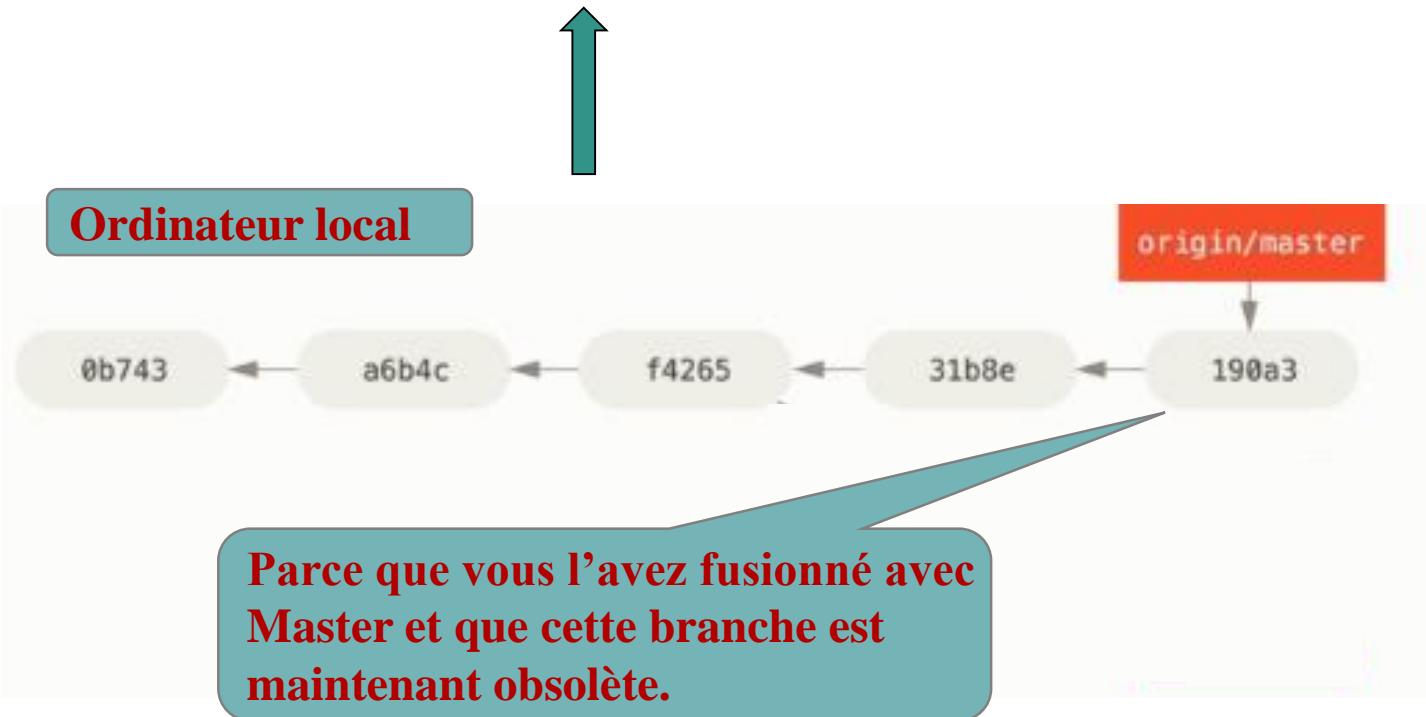
Branches distantes (4)

- ☐ Pour pousser la branche locale dans le dépôt distant
 \$ git push [serveur distant] [branche]



Branches distantes (5)

- ☐ Pour éliminer la branche distante mise dans le dépôt distant
\$ git push [serveur distant] :[branche]



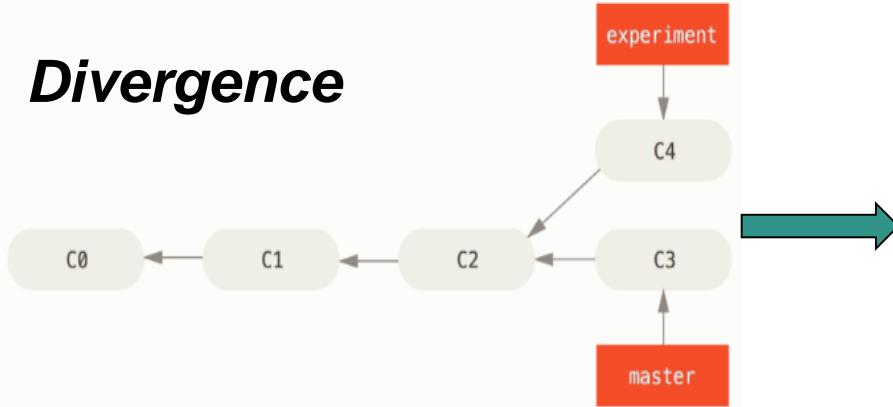
Sommaire

- Généralités sur les gestionnaires de version
- **Présentation de Git**
 - Concepts généraux
 - Qu'est ce qu'un instantané ?
 - Cycle de vie d'un fichier
 - Concept de branche
 - Création de branches
 - Branche et fusion
 - ***Merge et rebase***
- Serveur d'hébergement
- Conclusion

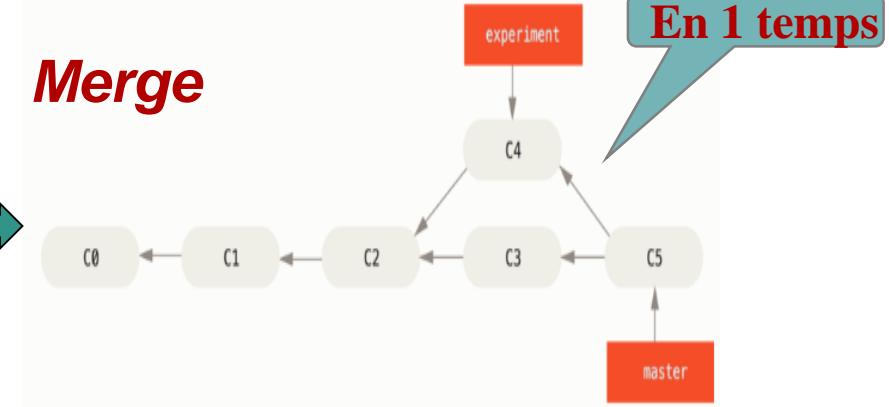
Merge et rebase (1)

□ Méthode « rebaser » une autre manière de fusionner

Divergence

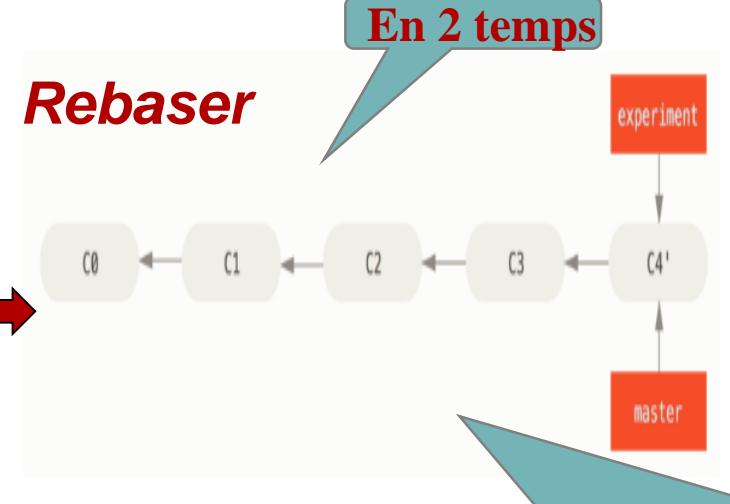


Merge



En 1 temps

Rebaser



En 2 temps

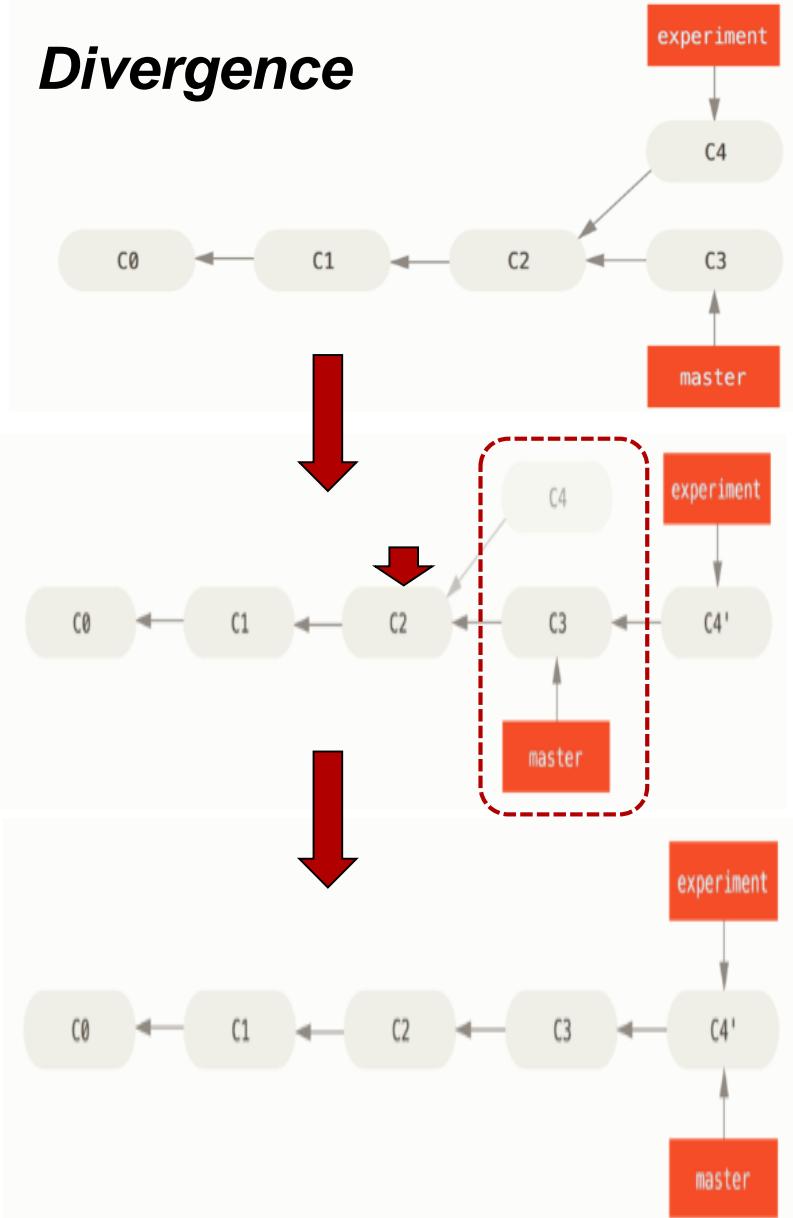
1^{er} temps : merge C4 et C3
Pour obtenir C4'

Karine Brifault

2^{ème} temps : Rendre linéaire l'historique en
éliminant le pointeur inutile

Concept de Branche : rebaser (2)

Divergence



- « rebaser » consiste à prendre le patch de la modification introduite dans C4 et le réappliquer sur C3.

\$ git checkout experiment

...

\$ git rebase master

- Cela qui équivaut à :

- retrouver le **dernier ancêtre commun**
- Faire un commit sur ce dernier pour créer une branche temporaire
- Lui appliquer les **modifications** dans leur **ordre chronologique**
- Faire un merge sur la **branche master**

Merge VS Rebase

□ Que représente la branche d'en face ?

1. ***Une branche locale temporaire faite par précaution***

Inutile qu'elle reste visible dans l'historique dès que les modifications seront appliquées :

La branche master a avancé → **rebase**

La branche master n'a pas avancé → **merge** (fast forward)

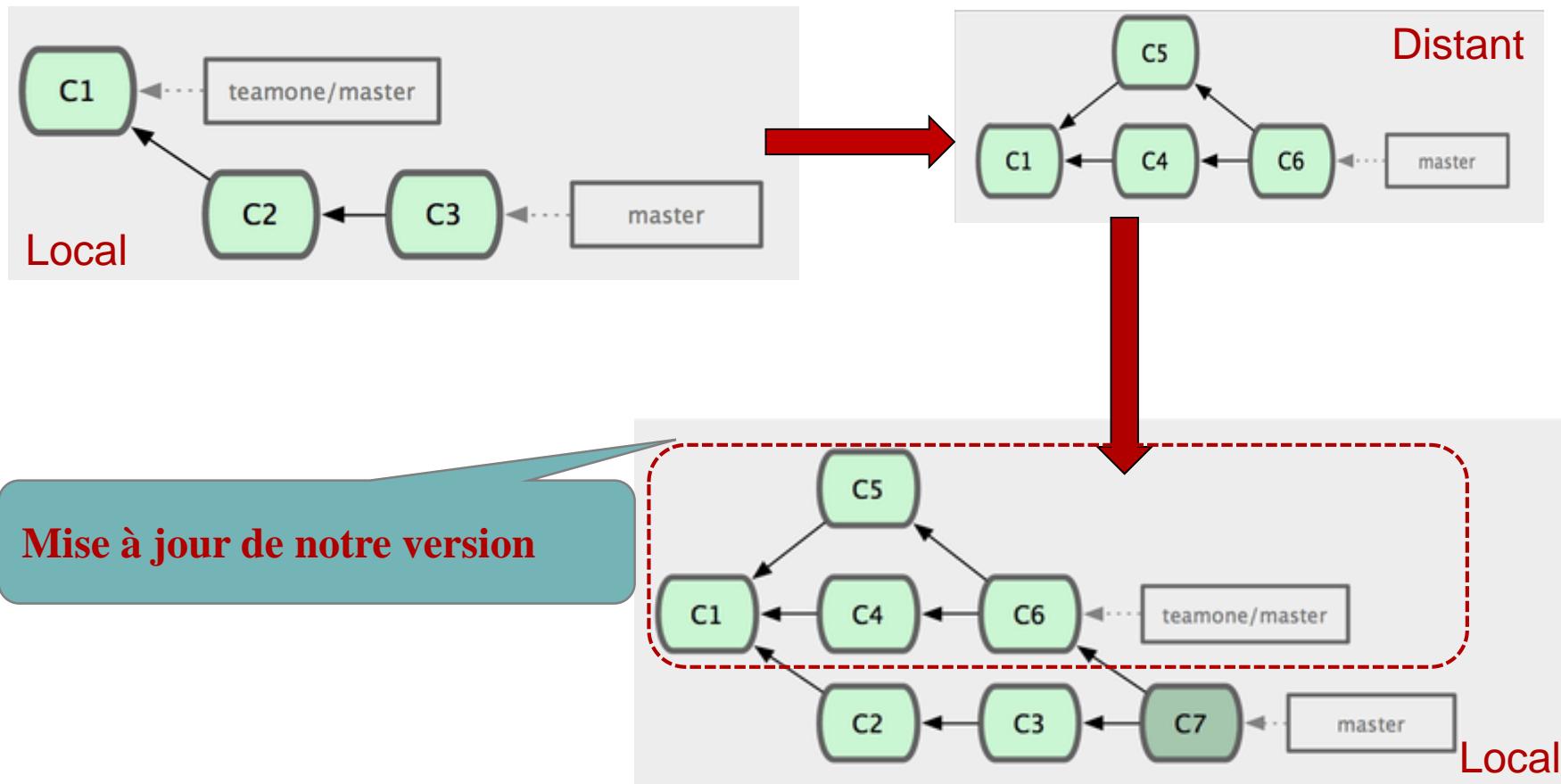
2. ***Une branche connue du projet***

Garder un historique est important pour pouvoir revenir ou continuer sur cette branche → **merge**

3. ***Cas particulier : remote et branche distante obsolète*** → **rebase**

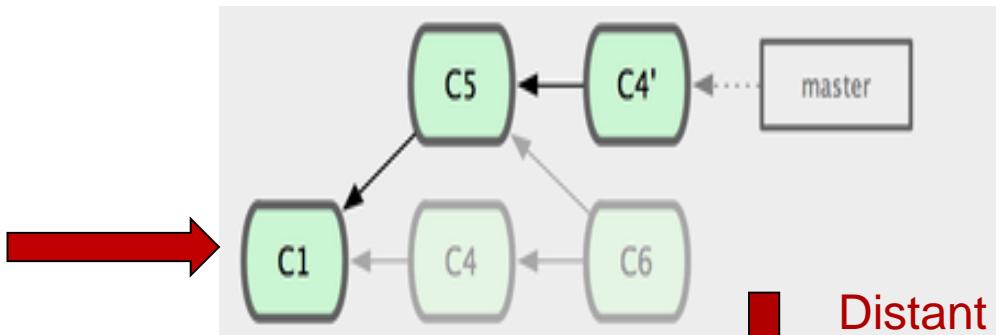
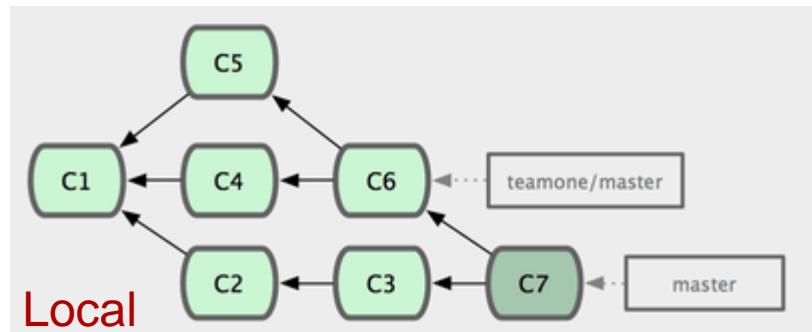
Les dangers du rebase (1)

- Ne jamais rebaser des commits qui ont déjà été poussés sur un dépôt public

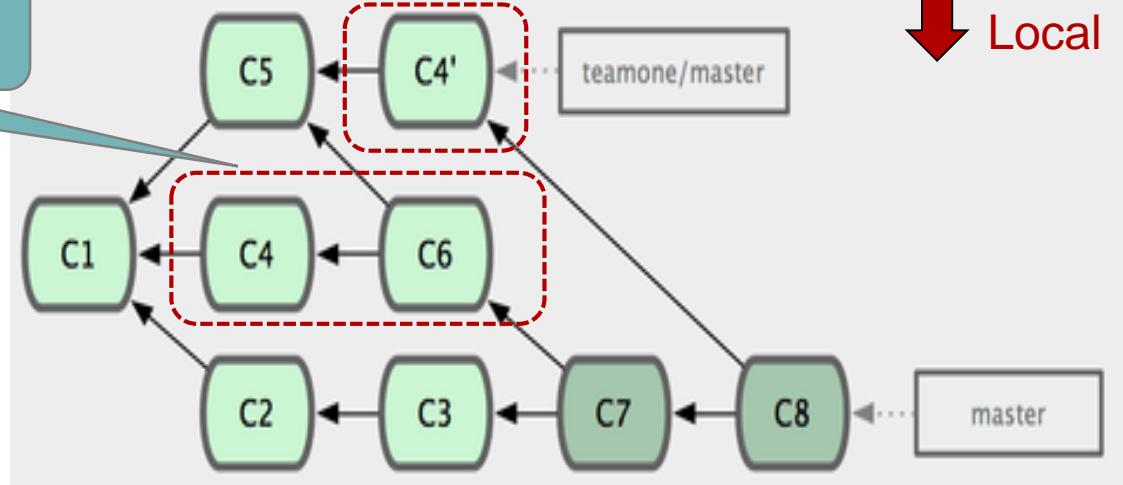


Les dangers du rebase (2)

- Ne jamais rebaser des commits qui ont déjà été poussés sur un dépôt public



Incohérence des sources !!!



Sommaire

- Généralités sur les gestionnaires de version
- Présentation de Git
- ***Serveur d'hébergement***
 - ***GitHub***
 - ***GitEye***
 - ***Egit***
- Conclusion

Interfaces et serveurs d'hébergement

Différentes solutions possibles d'hébergement sur internet

Bitbucket

Recommend | 101 | 3

The screenshot shows the Bitbucket dashboard for a repository named 'staging'. It includes a 'Recommend' button with 101 votes, a 'Download' button with 3 votes, and a detailed commit history from the last month.

[Read More](#)

GitHub

Recommend | 102 | 5

The screenshot shows the GitHub interface for a repository. It includes a 'Recommend' button with 102 votes, a 'Download' button with 5 votes, and a detailed commit history from the last month.

[Get it now](#)

GitLab

Recommend | 60 | 4

The screenshot shows the GitLab interface for a repository. It includes a 'Recommend' button with 60 votes, a 'Download' button with 4 votes, and a very long list of commit history entries spanning several months.

[Get it now](#)

[Read More](#)

GitHub

Serveur GitHub



Build software
better, together.

GitHub est centré vers l'aspect social du développement

L'esprit communautaire derrière GitHub, offre de nombreuses fonctionnalités habituellement retrouvées sur les réseaux sociaux

- Les flux [Subscribe to your news feed](#)
- Suivre des personnes ou des projets

* L'une des fonctionnalités de GitHub est la capacité de pouvoir  sur quoi d'autres personnes travaillent et avec qui elles se connectent

* Vous pouvez contribuer (ajout de fonctionnalités, corrections de bugs) sur le projet de quelqu'un d'autre, il vous suffit de « forker »

Karine Brifault

60



GitHub

Suivi de projet

Un projet collaboratif passe par plusieurs étapes (bugs, améliorations) lors de sa phase de développement

GitHub propose un moyen de garder une trace des tâches, des améliorations et des bugs pour vos projets, et qui peuvent être partagés et discutés avec le reste de votre équipe et de votre réseau social

Le « Tracker » de GitHub est appelée [Issues](#) possède sa propre section dans chaque référentiel

The screenshot shows the GitHub Issues interface. At the top, there are tabs for 'Issues' (which is selected), 'Pull requests', 'Labels', and 'Milestones'. To the right of the tabs are 'Filters' (set to 'is:issue is:open'), a search bar, and a 'New issue' button. Below the header is a summary row showing '62 Open' and '11,482 Closed'. The main area displays five open issues:

- Panel heading border is cut off (css) - opened 15 hours ago by mking
- .text-left not working on btn-block v3.3.5 (confirmed css) - opened 3 days ago by bslugger
- Bootstrap is not compatible with jQuery 3.0 (confirmed docs js meta) - opened 3 days ago by MaxKorz
- shown.bs.dropdown Related target (js) - opened 4 days ago by DaGLIMiOuX

GitHub

.Wiki du dépôt

Pour formaliser les principes de votre projet. C'est une documentation pour aider les futurs contributeurs

The screenshot shows a GitHub wiki page for the Hystrix project. At the top left is the Hystrix logo, which features a stylized bear inside a sunburst-like circle. To the right of the logo is the word "HYSTRIX" in large, bold, uppercase letters, with "DEFEND YOUR APP" in smaller uppercase letters below it. On the far right, there is a sidebar titled "Pages (13)" containing a list of links to various documentation pages.

Page Category	Links
Home	<ul style="list-style-type: none">HomeGetting StartedHow To Use<ul style="list-style-type: none">Hello World!Synchronous ExecutionAsynchronous ExecutionReactive ExecutionFallbackError PropagationCommand NameCommand GroupCommand Thread PoolRequest CacheRequest CollapsingRequest Context SetupCommon PatternsMigrating to HystrixHow It Works<ul style="list-style-type: none">Execution FlowCircuit BreakerIsolation

What is Hystrix?

In a distributed environment, failure of any given service is inevitable. Hystrix is a library designed to control the interactions between these distributed services providing greater latency and fault tolerance. Hystrix does this by isolating points of access between the services, stopping cascading failures across them, and providing fallback options, all of which improve the system's overall resiliency.

Hystrix evolved out of resilience engineering work that the Netflix API team began in 2011. Over the course of 2012, Hystrix continued to evolve and mature, eventually leading to adoption across many teams within Netflix. Today tens of billions of thread-isolated and hundreds of billions of semaphore-isolated calls are executed via Hystrix every day at Netflix and a dramatic

GitHub

GitHub aujourd'hui :

GitHub propose des souscriptions payantes en marge des comptes gratuits

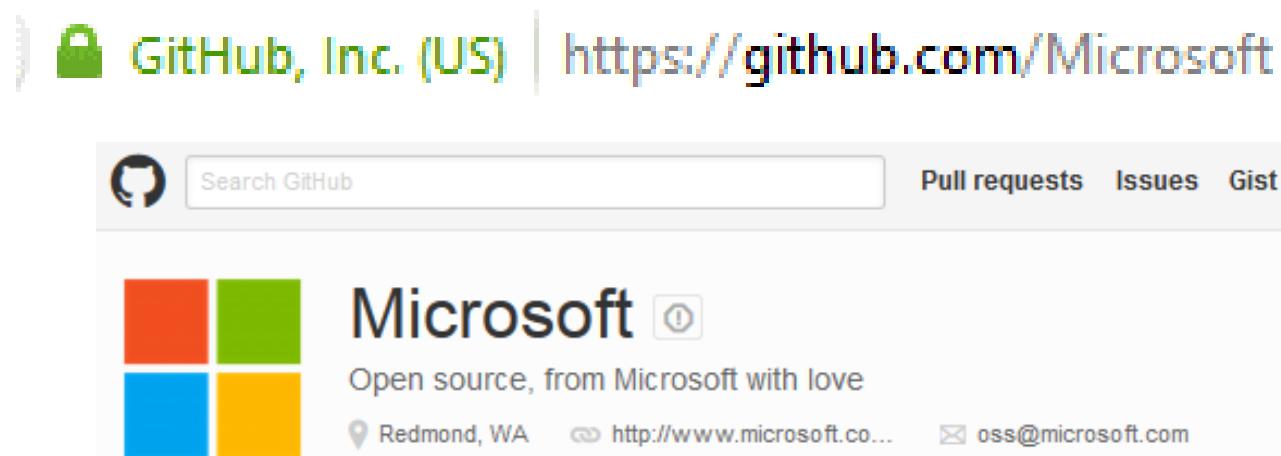
De même, le site vend une version professionnelle de GitHub aux entreprises qui souhaitent faire tourner la plate-forme en interne, sur leur propre réseau

Environ 9 millions de personnes collaboreraient sur GitHub, via une vingtaine de millions de dossiers déposés

GitHub

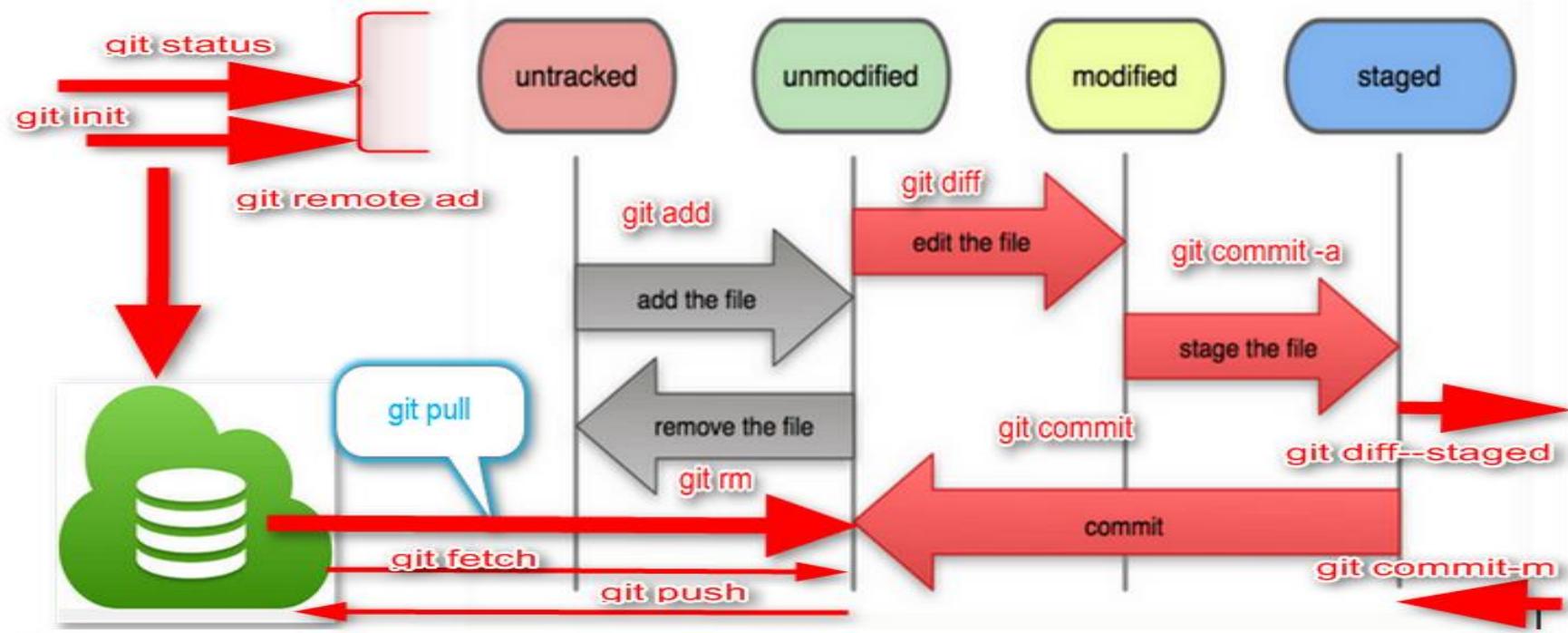
A tel point que même les plus grands groupes du Web s'y sont mis :

- * Google a abandonné sa plate-forme maison, Google code, pour GitHub
- * Microsoft, chantre du logiciel propriétaire, opaque et verrouillé, a mis de l'eau dans son vin



GitHub

Les commandes sont à exécuter sous linux



GitHub



Un client permettant de gérer facilement les fichiers Git à travers un interface graphique

The screenshot shows the CollabNet GitEye application window. On the left, a sidebar titled "Liste des dépôts" displays a tree view of repositories, with a red arrow pointing to the "rep_bare [NO-HEAD - Bare]" entry. The main area has tabs for "Dashboard", "Git Files" (which is selected), "History", "Task List", "Builds", "Synchronize", and "lisezmoi.txt". The "Git Files" tab shows the repository "rep_clone_of_rep_bare [master]". The "Working Tree Files (8)" section lists files: mario, mario/css, mario/images, mario/js, mario/js/jquery, .gitignore, and lisezmoi.txt. A red box highlights the "répertoire de travail" label. The "Staged Changes (0)" section is empty, with a red box highlighting the "répertoire fichier indexé prêt pour le co" label. On the right, the "Commit Message" section shows fields for "Author" (tdo <tdo.formation@gmail.com) and "Committer" (tdo <tdo.formation@gmail.com). Red annotations identify the author and committer as "identification" and "développeur" respectively. Buttons for "Commit and Push" and "Commit" are present. A large red box highlights the "Affichage de différence sur entre le fichier de travail et la dernière version commit" label, which corresponds to the "Diff" tab.

CollabNet GitEye

File Edit Git Window Help

Git Reposit... Help

rep_bare [NO-HEAD - Bare] - C:\gtere
rep_clone_of_rep_bare [master] - C:\gtetest

Branches Tags References Remotes Working Directory - C:\Users\tdo\rep_git [f5b4348 - Rebase interactive]
RepExistant [NO-HEAD] - C:\gittest

Liste des dépôts

Dashboard Git Files History Task List Builds Synchronize lisezmoi.txt

Find

rep_clone_of_rep_bare

Working Tree Files (8)

mario
mario/css
mario/images
mario/js
mario/js/jquery
.gitignore
lisezmoi.txt

répertoire de travail

Staged Changes (0)

répertoire fichier indexé prêt pour le co

Commit Message

Author: tdo <tdo.formation@gmail.com> identification
Committer: tdo <tdo.formation@gmail.com. développeur

Commit and Push Commit

Diff Tasks Show

Affichage de différence sur entre le fichier de travail et la dernière version commit

Karine Brifault

66

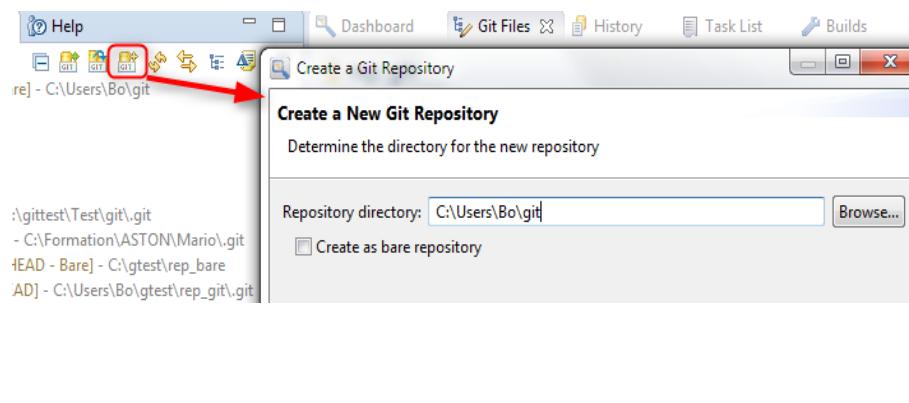
Sommaire

- Généralités sur les gestionnaires de version
- Présentation de Git
- ***Serveur d'hébergement***
 - GitHub
 - ***GitEye***
 - EGit
- Conclusion

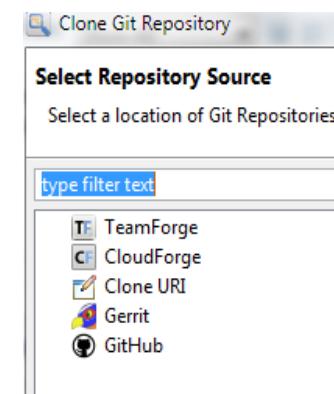
GitEye



Créer un dépôt
- à partir d'un répertoire existant

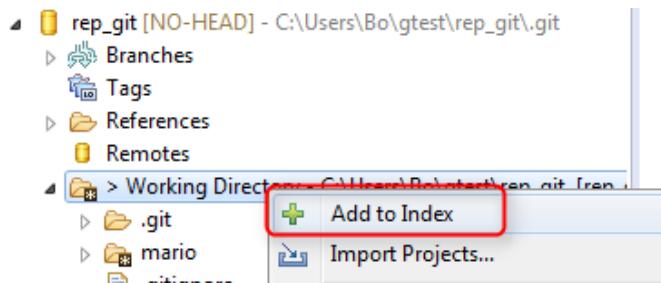


- ou par clonage à partir d'un dépôt

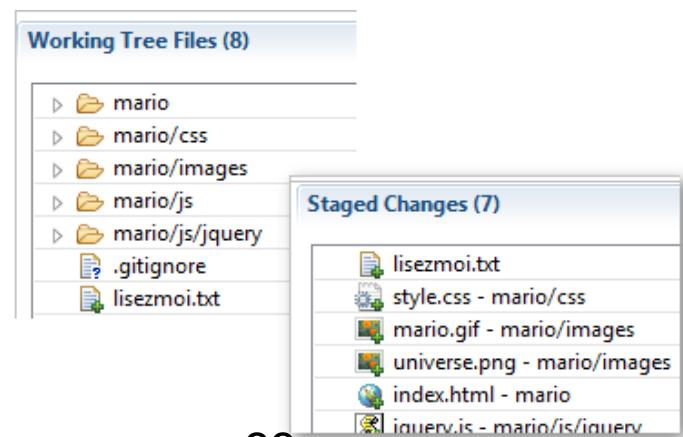


Hébergé sur internet
TeamForge,
CloudForge,
Gerrit,
GitHub

Indexer les nouveaux fichiers pour le suivi



Les fichiers apparaîtront dans répertoire de travail



GitEye



Commit un fichier (valider)

- Le commentaire est obligatoire.
- L'identification permet de savoir qui fait quoi.

Commit Message

```
2e commit: fichier lisezmoi a changé  
commentaire
```

Author: tdo <tdo.formation@gmail.com>
Committer: tdo <tdo.formation@gmail.com>

identification

Commit and Push **Commit**

Suivi historique modification

- sur le dépôt

Dashboard Git Files History Task List Builds

Repository: rep_git

Id	Message	Author	Auth
9415dc1	master [HEAD] 2e commit: fichier lisezmoi a changé	tdo	7 min
6c5b74f	1er commit	tdo	25 mi

commit 9415dc1774d2dc3f6070ff00ebec5e00fd655b12
Author: tdo <tdo.formation@gmail.com> 2015-07-14 09:08:08
Committer: tdo <tdo.formation@gmail.com> 2015-07-14 09:08:08
Parent: [6c5b74ff56c828faa294ebdac4dde82cc559b801](#) (1er commit)
Branches: master

2e commit: fichier lisezmoi a changé

- sur fichier

Karine Brifault

Dashboard Git Files History Task List Builds

File: rep_git/lisezmoi.txt [rep_git]

Id	Message
9415dc1	master [HEAD] 2e commit: fichier lisezmoi a changé
6c5b74f	1er commit

Author: tdo <tdo.formation@gmail.com> 2015-07-14 09:08:08
Committer: tdo <tdo.formation@gmail.com> 2015-07-14 09:08:08
Parent: [6c5b74ff56c828faa294ebdac4dde82cc559b801](#) (1er commit)
Branches: master

2e commit: fichier lisezmoi a changé

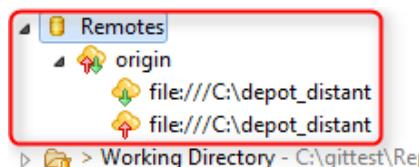
diff --git a/lisezmoi.txt b/lisezmoi.txt
index 930b513..37dbc83 100644
--- a/lisezmoi.txt
+++ b/lisezmoi.txt
@@ -1,3 +1,5 @@
 Répertoire avec git
 ======
-Créé en local
 \ No newline at end of file
+Créé en local
===== 69 =====
+1ere modif
 \ No newline at end of file

GitEye

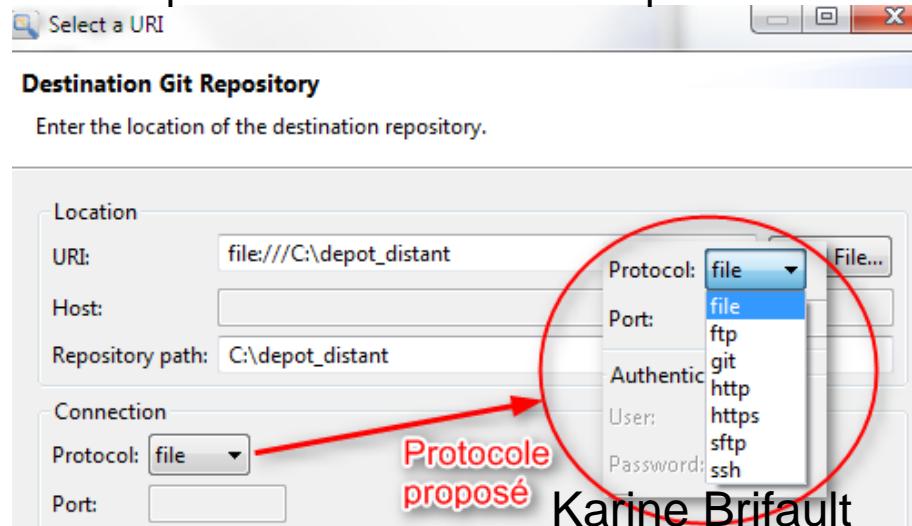


Synchroniser avec un dépôt distant

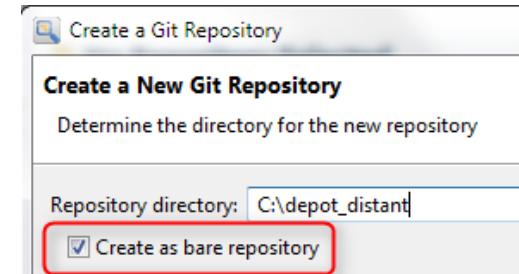
4 Résultat du paramétrage



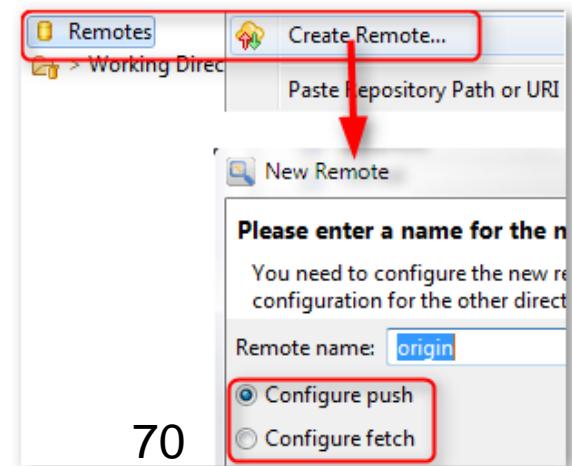
3 Donner le nom du répertoire où se trouve le dépôt distant en utilisant le protocole « file »



1 Créer un dépôt nu « bare repository » (ne contient pas de fichier).



2 Paramétrer dans dépôt local le « Remote » pour pousser (push) et tirer (fetch)

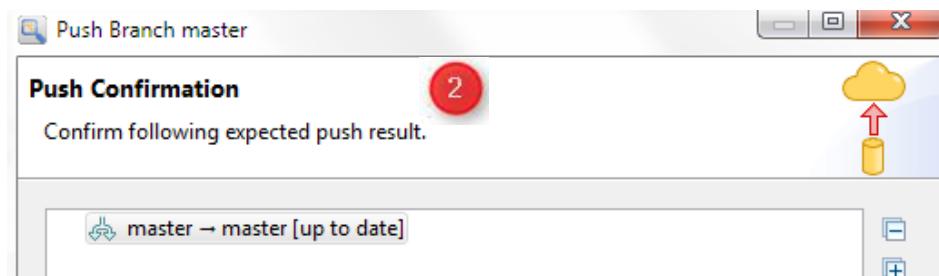
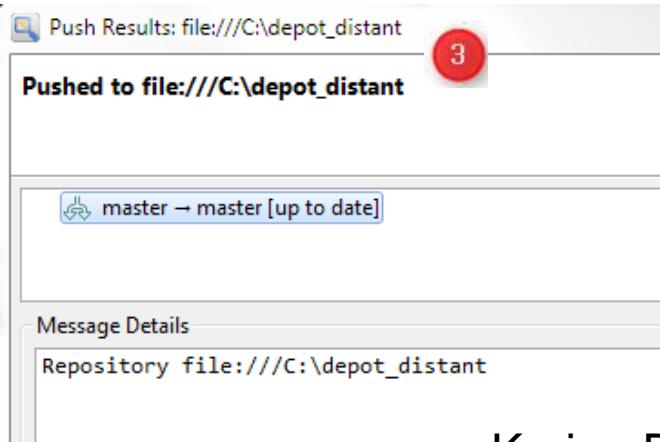
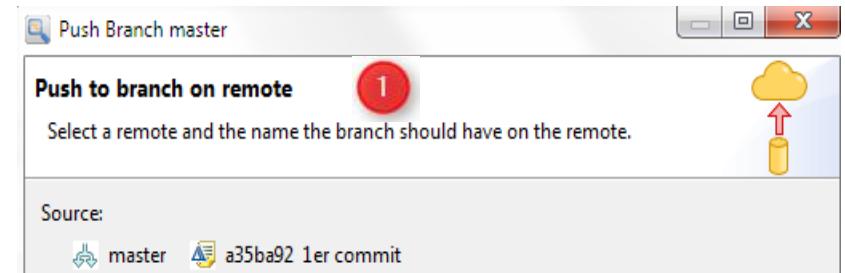
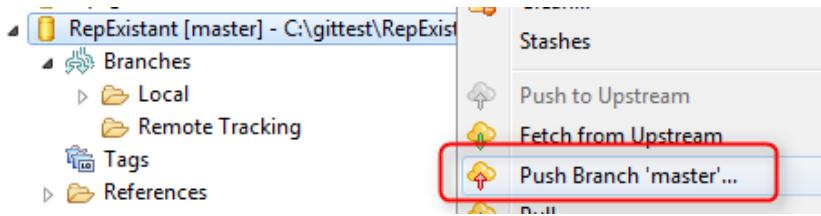


GitEye



Synchroniser avec un dépôt distant

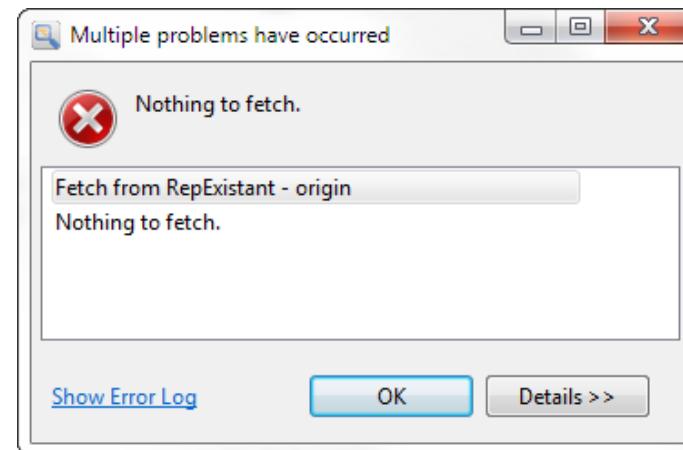
Pousser (push) les données vers dépôt distant



GitEye



Synchroniser avec un dépôt distant

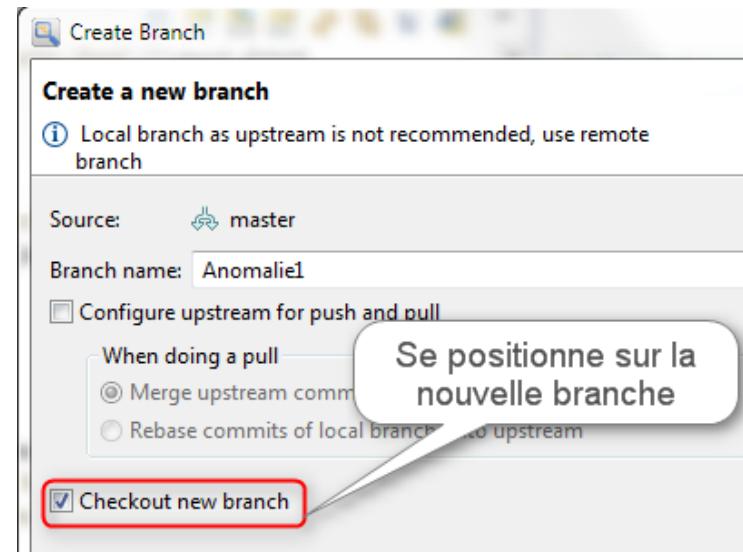
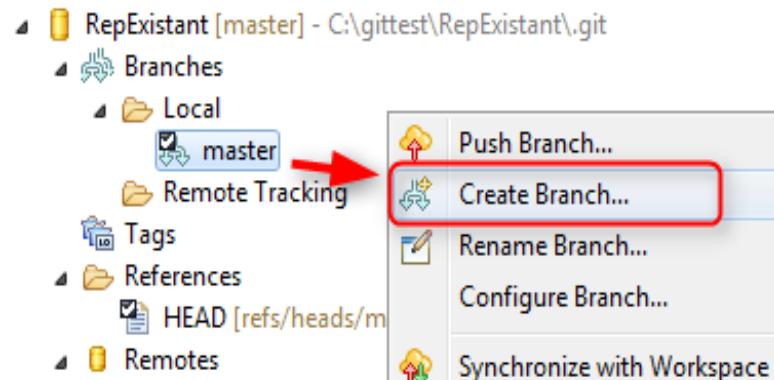


GitEye

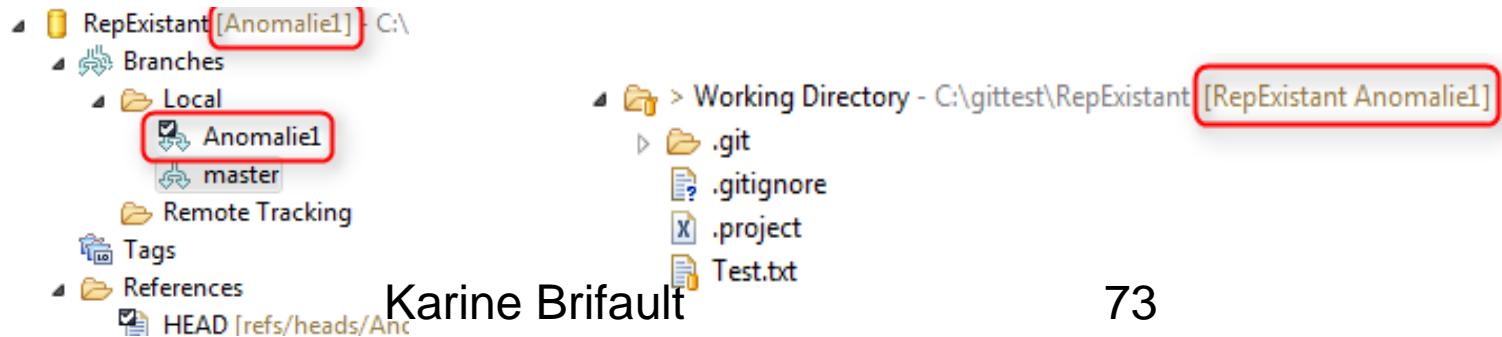


Gestion de branche

Création de Branche



Résultat La Branche Anomalie1 devient active



Karine Brifault

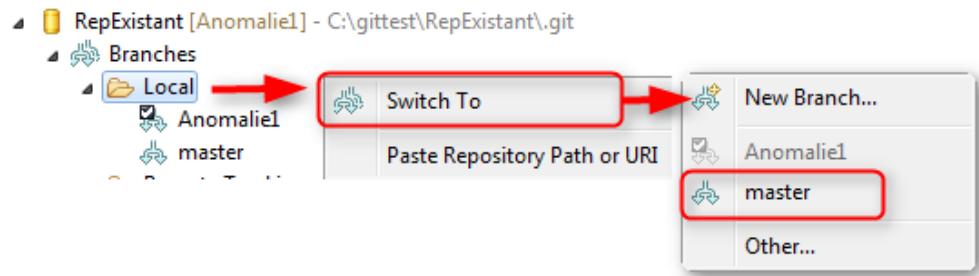
73

GitEye

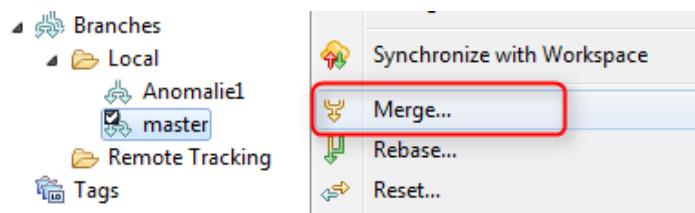


Gestion de branche

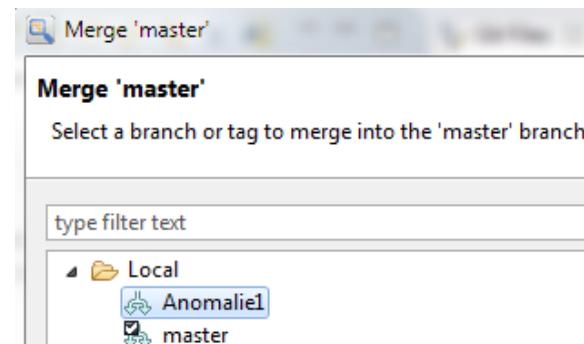
Changer de branche (checkout)



Fusion (merge / rebase) une branche dans la branche master



Sélectionner la branche à fusion

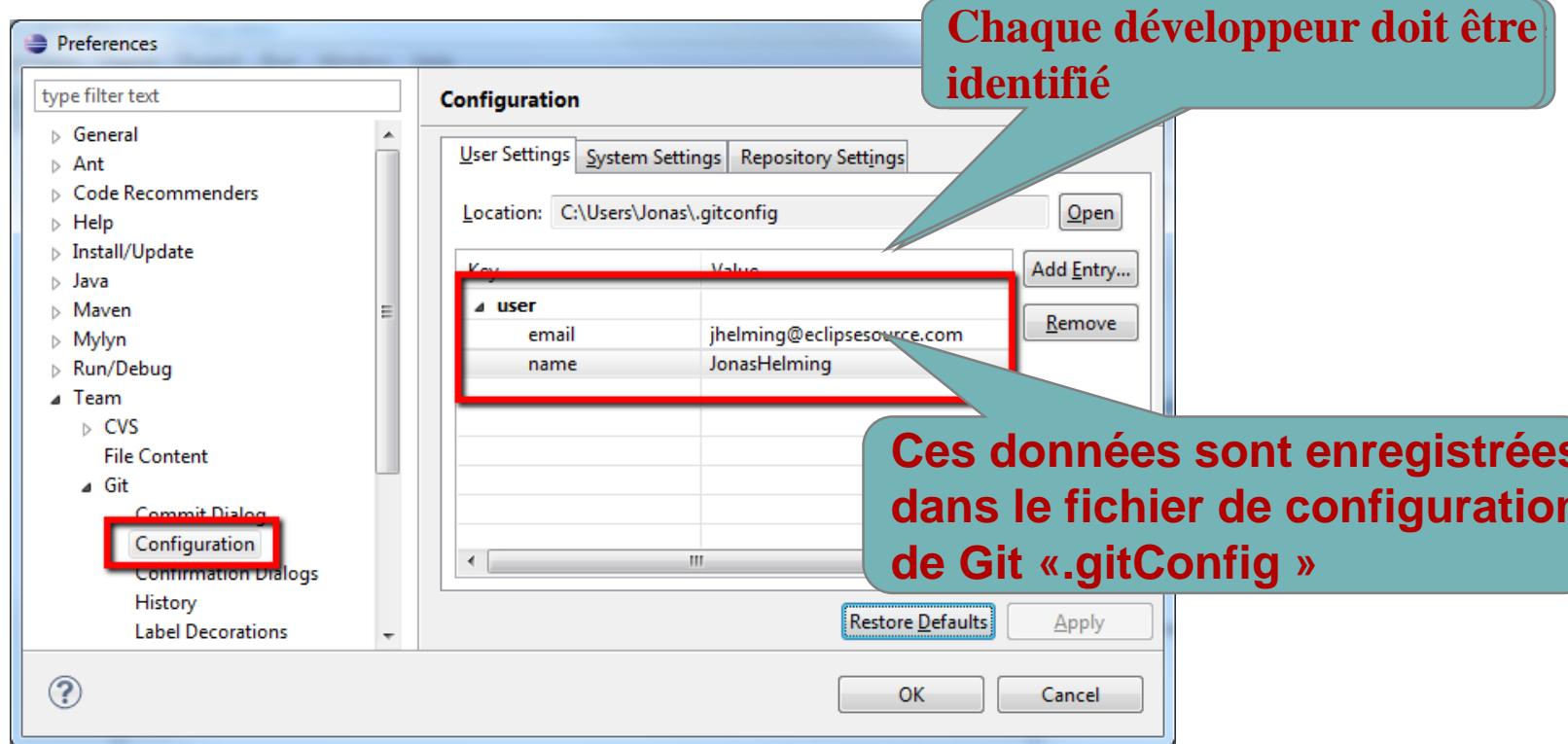


Sommaire

- Généralités sur les gestionnaires de version
- Présentation de Git
- ***Serveur d'hébergement***
 - GitHub
 - GitEye
 - ***Egit***
- Conclusion

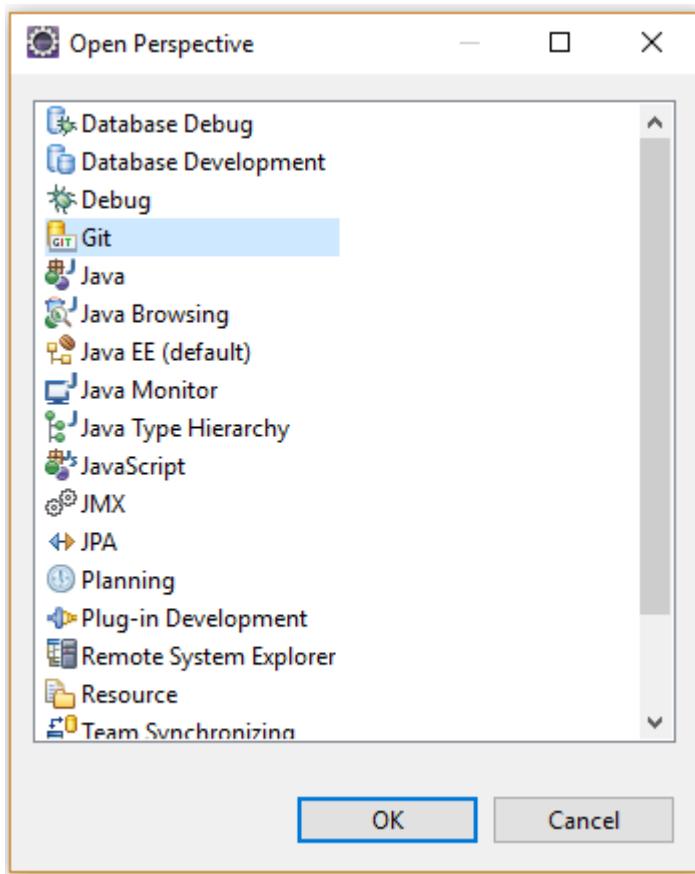
EGit : configuration

- Le plugin EGit est intégré depuis la version Juno dans l'IDE Eclipse
- Pour configurer EGit : **Window/Preferences/Team/Git**



Egit : création du répertoire local (1)

- Pour créer un projet Git, choisir la perspective Git :



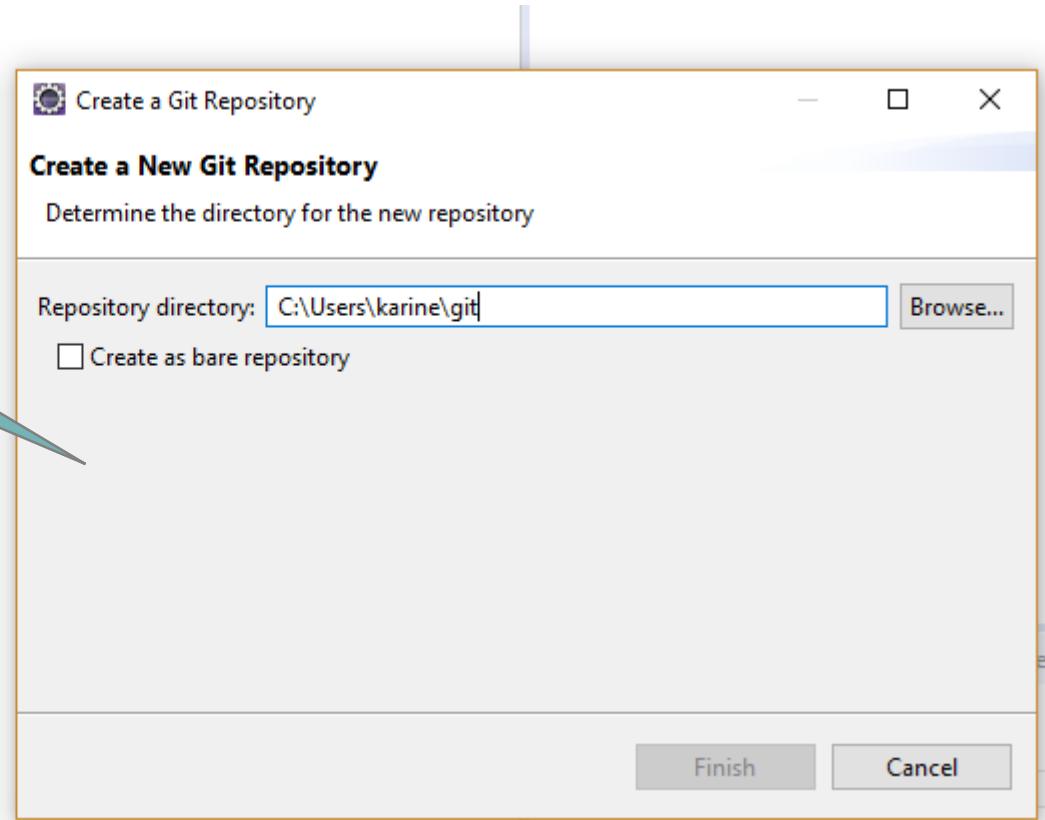
Egit : création du répertoire local (2)

- Pour créer un projet Git, choisir la perspective Git, puis

Vous pouvez alors vous connectez à un dépôt Git déjà existant

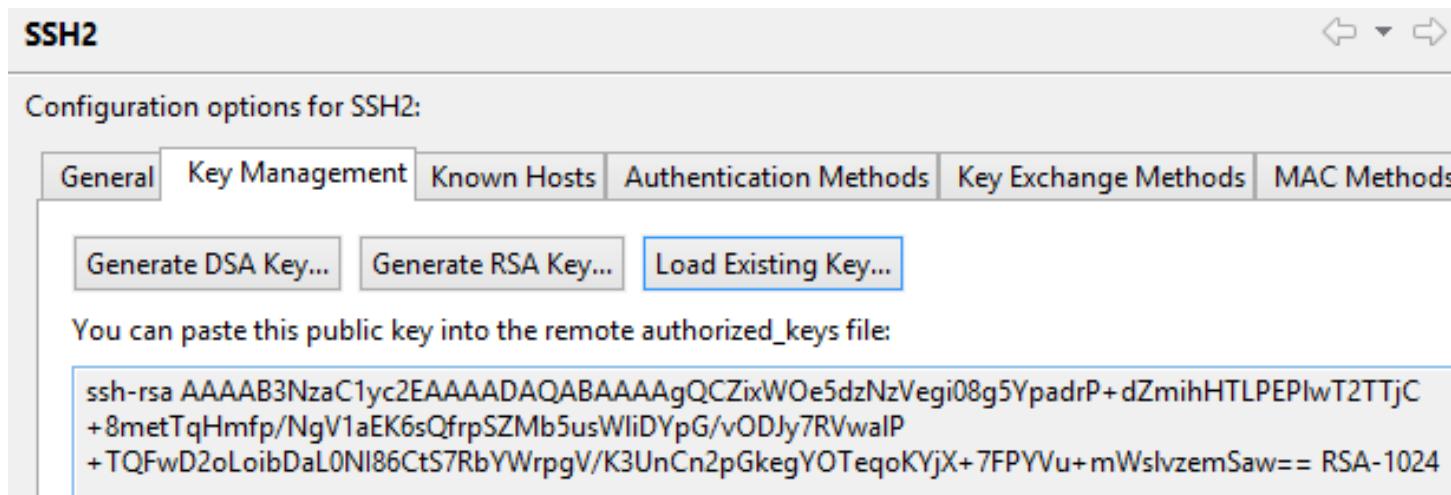
Select one of the following to add a repository to this view:

-  [Add an existing local Git repository](#)
-  [Clone a Git repository](#)
-  [Create a new local Git repository](#)



Egit : configuration ssh

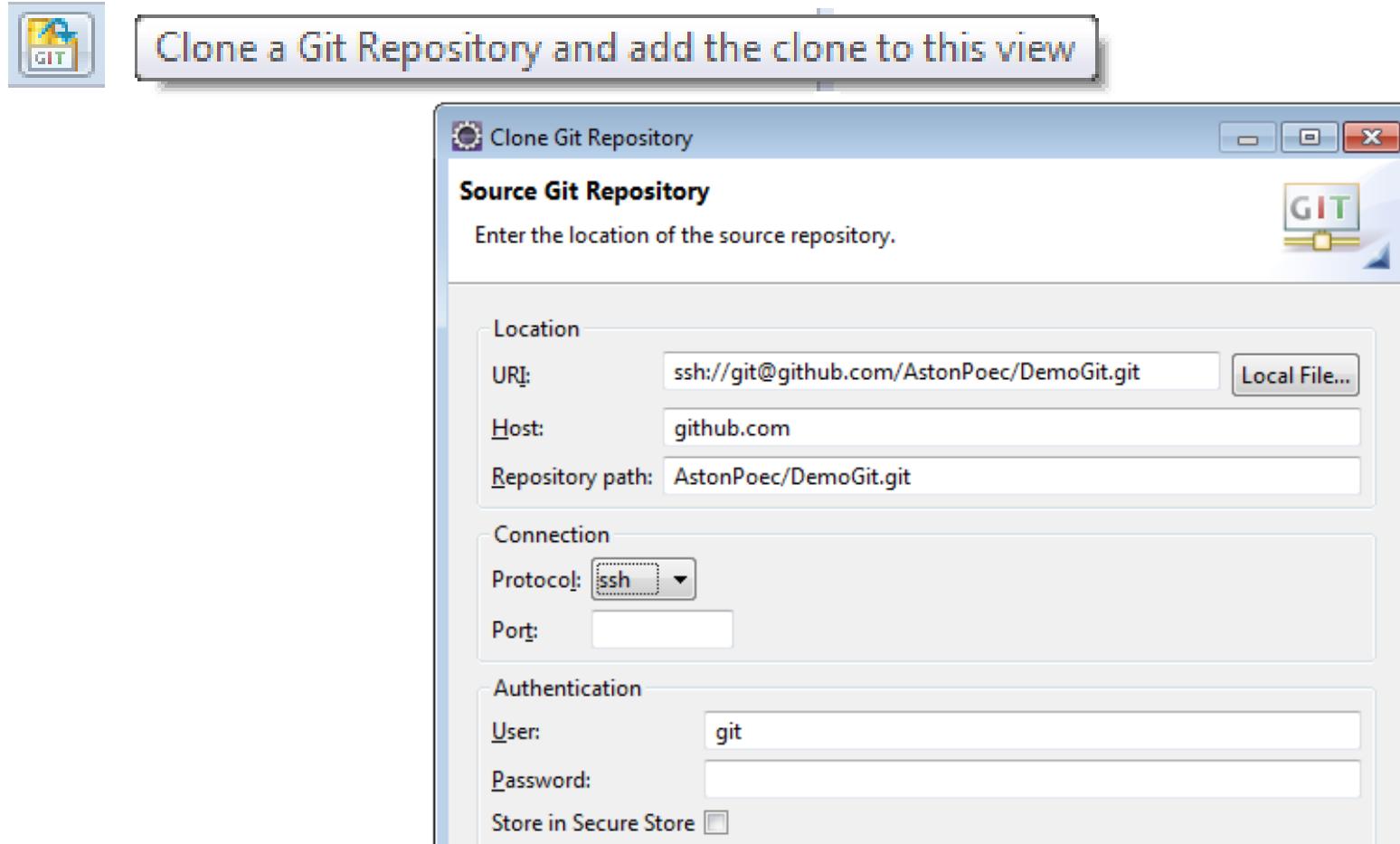
- L'accès du développeur à un dépôt distant doit être sécurisé par une clé SSH
- Pour configurer la clé SSH :



- La clé publique doit être ajoutée au dépôt distant pour autoriser l'accès à un utilisateur

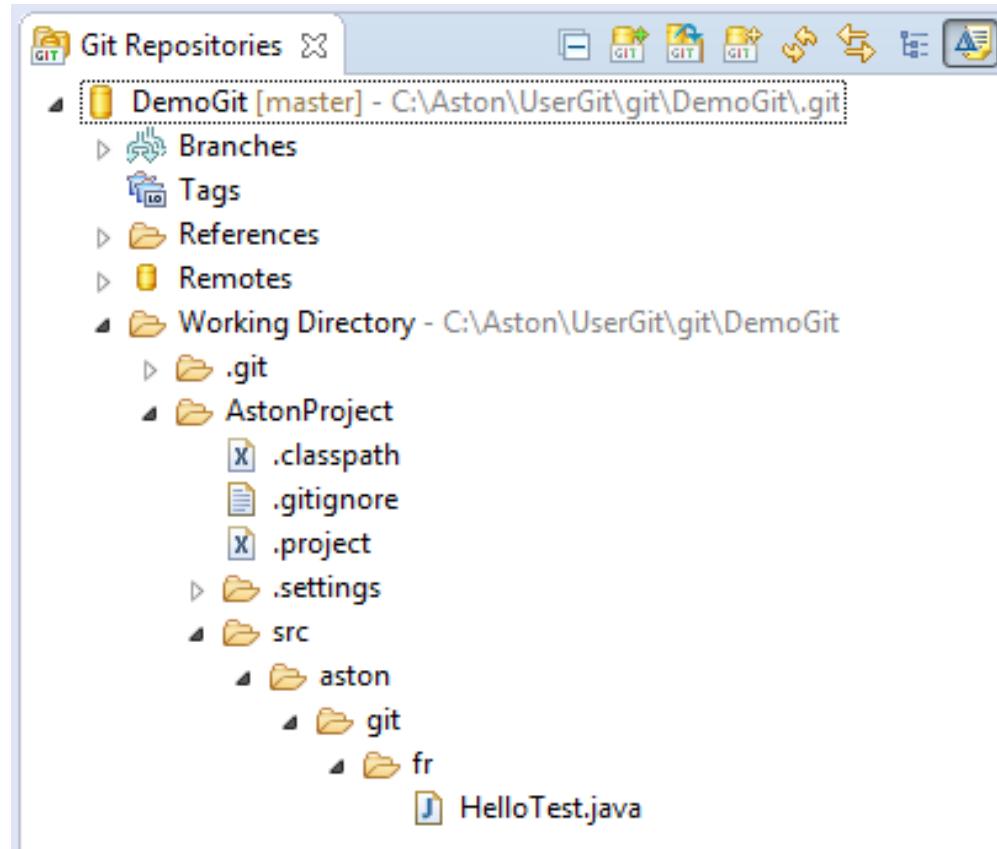
Egit : cloner le dépôt en local

□ Pour cloner en local un dépôt DISTANT



Egit : visualisation du répertoire local

□ Clone du dépôt DISTANT dans Eclipse



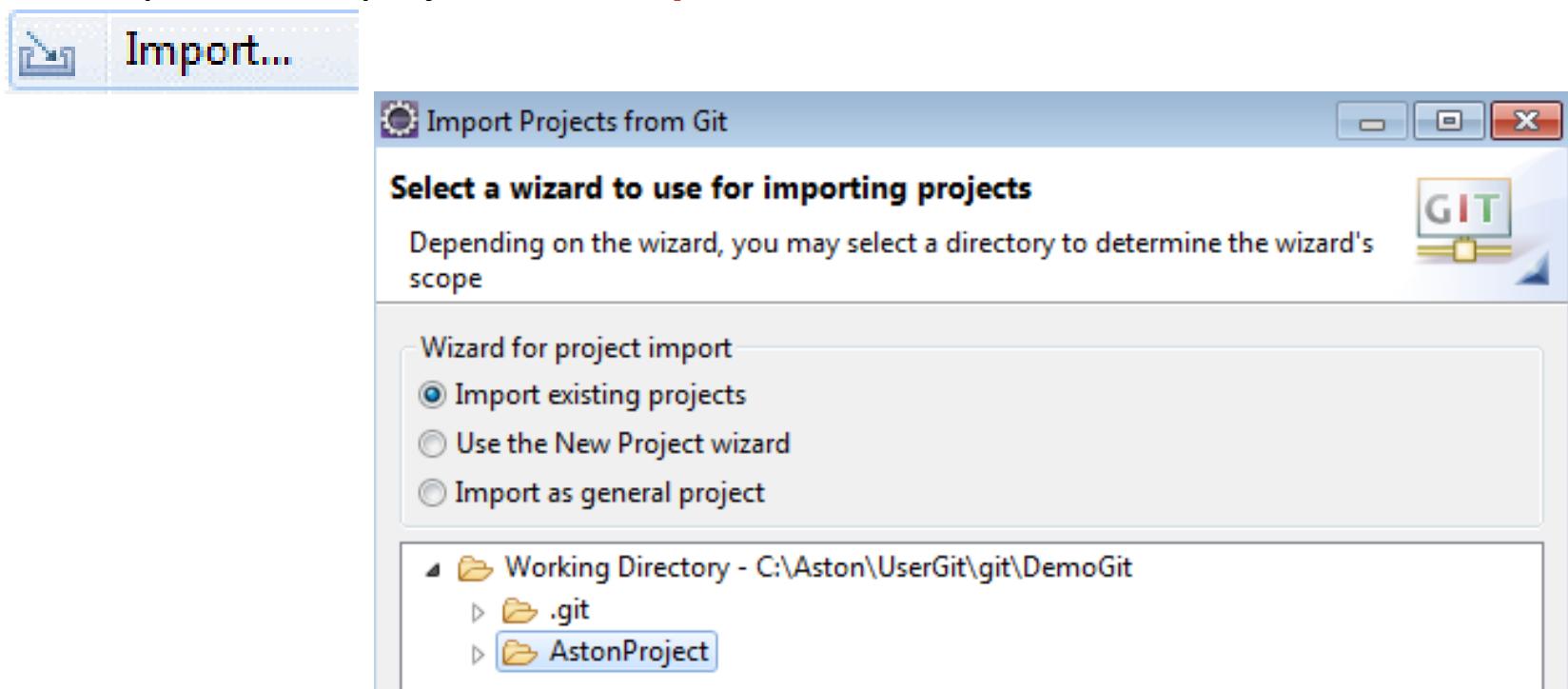
EGit : importer son projet dans la vue Java

□ Pour importer ces sources dans la vue Java d'Eclipse

- Choisir la vue Java

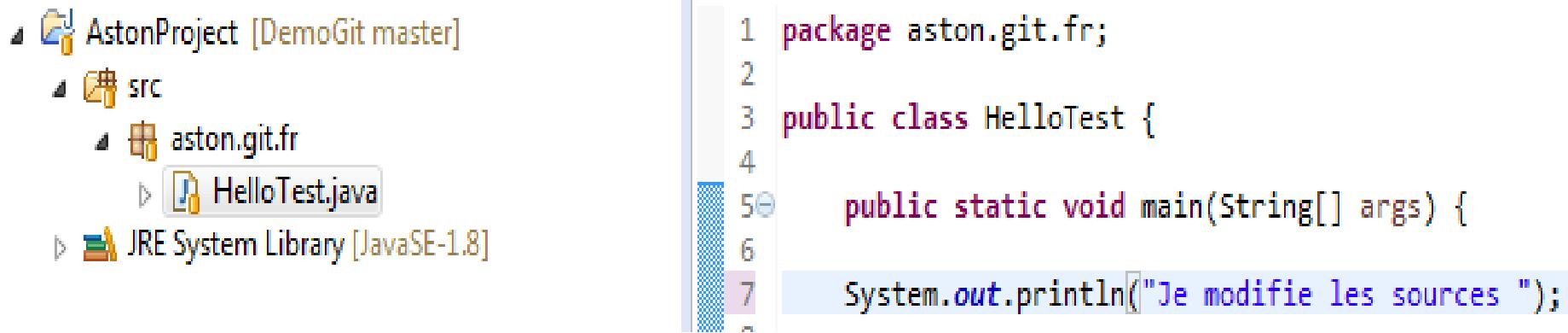


- Importer son projet : **File/import...**



Egit : modification des sources

□ Modifications du code source dans l'IDE Eclipse classique :



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays the project structure:

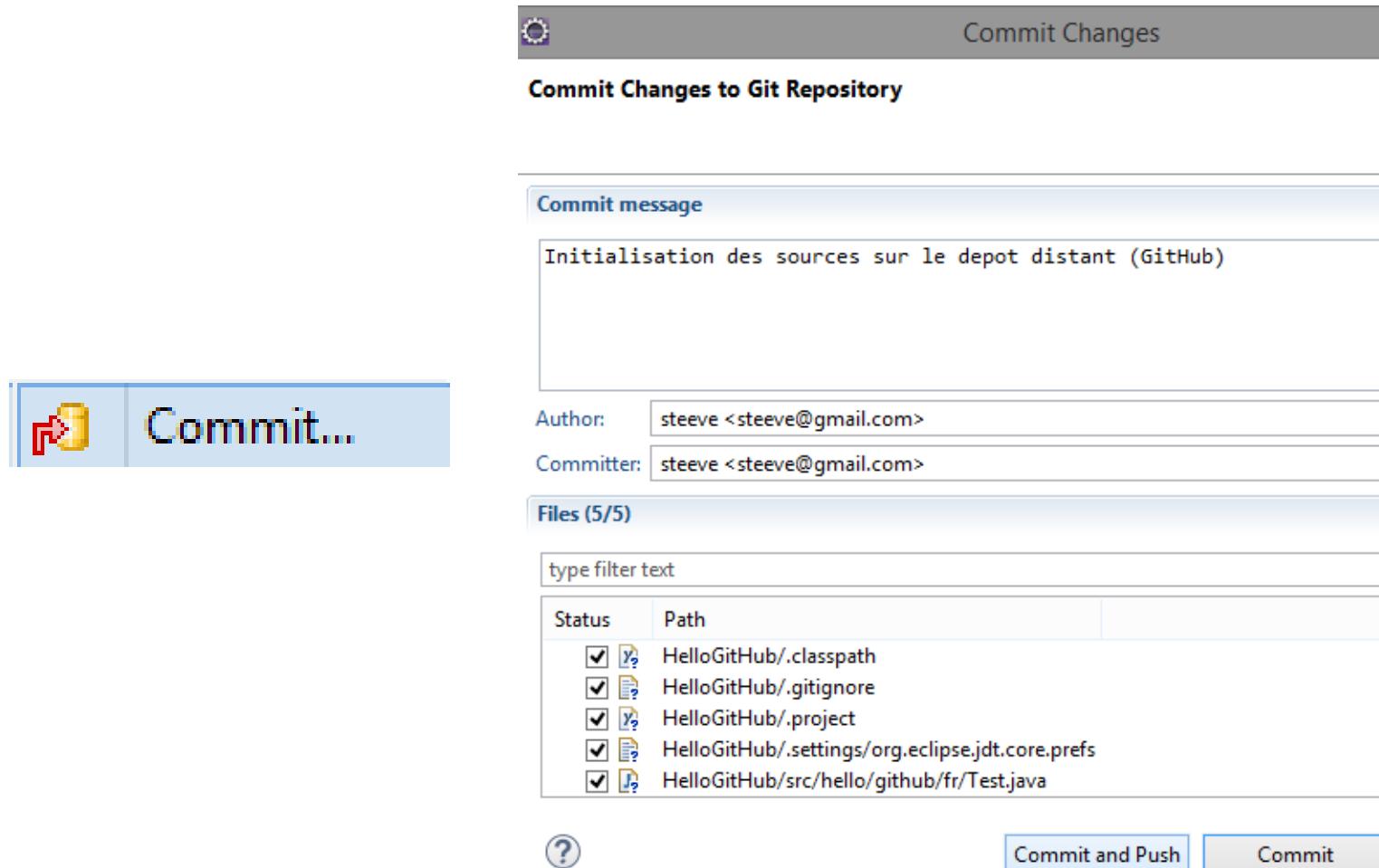
- AstonProject [DemoGit master]
 - src
 - aston.git.fr
 - HelloTest.java
 - JRE System Library [JavaSE-1.8]

On the right, the editor view shows the Java code for `HelloTest.java`:

```
1 package aston.git.fr;
2
3 public class HelloTest {
4
5     public static void main(String[] args) {
6
7         System.out.println("Je modifie les sources ");
8     }
9 }
```

Egit : commit / push (1)

- Archivage des modifications dans le dépôt LOCAL et/ou DISTANT :



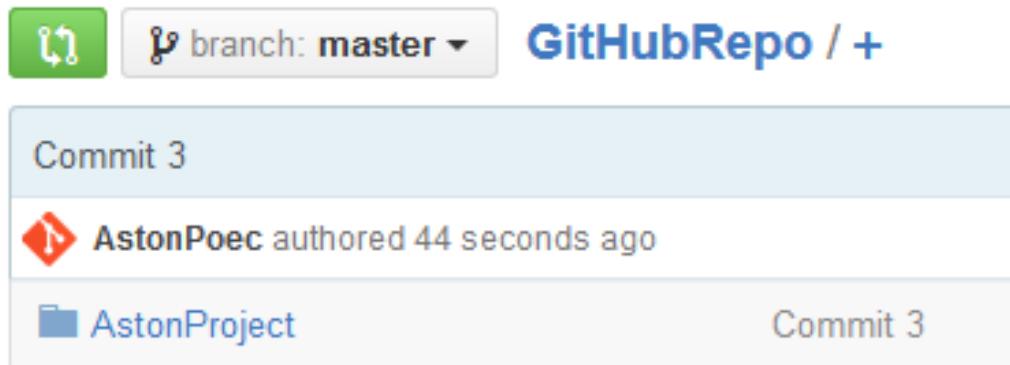
Egit : commit / push (2)

Commit

permet d'archiver ses modifications sur le dépôt LOCAL

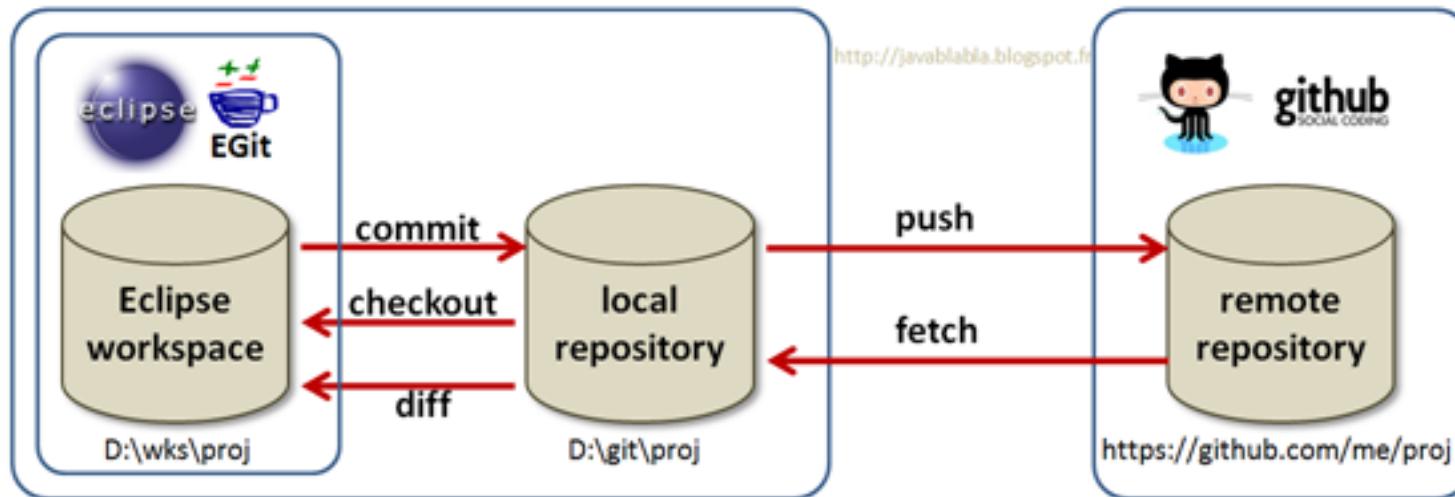
Commit and Push

permet d'archiver ses modifications en LOCAL puis de les pousser sur le dépôt DISTANT



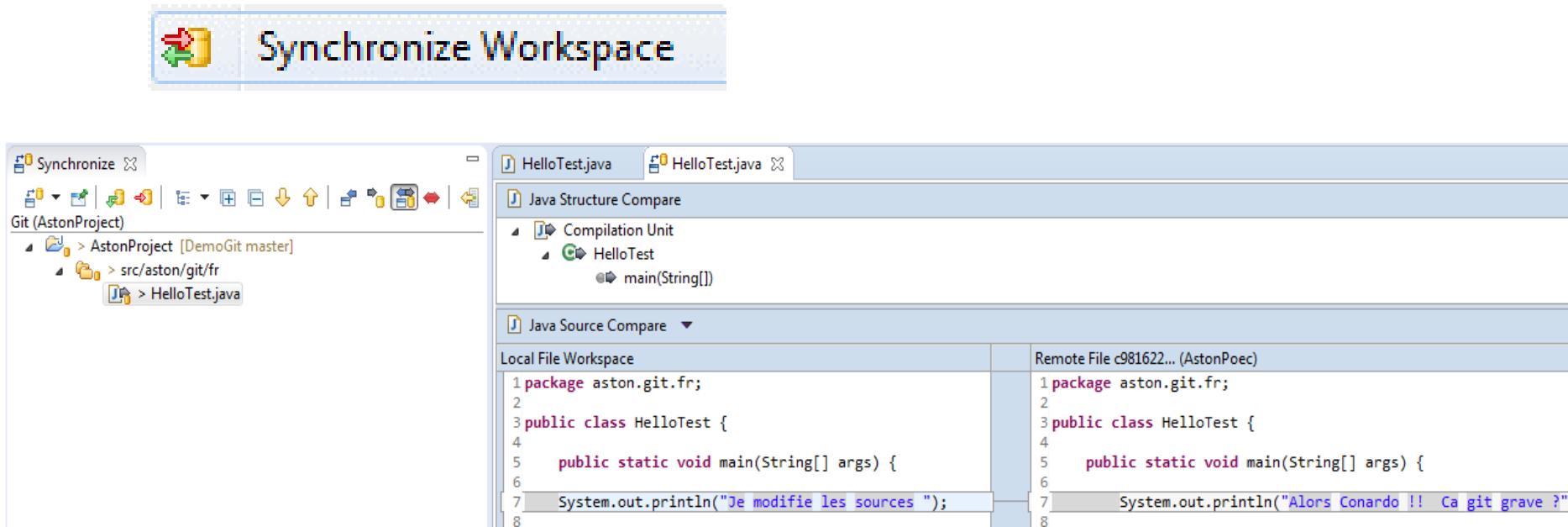
EGit : Vue d'ensemble

- Les étapes de COMMIT & PUSH peuvent être résumées ainsi



Egit : Local / Distant

- Visionnage des différences de sources entre les dépôts LOCAL et DISTANT



Egit : Comparaison de versions

□ Visionnage des anciennes versions de fichiers

The screenshot shows the Egit interface with two code editors side-by-side. The top editor displays the file `HelloTest.java` with the following code:

```
1 package aston.github.fr;
2
3 public class HelloTest {
4
5     public static void main(String[] args) {
6
7         System.out.println("Commit version 3");
8     }
9 }
```

The bottom editor displays the same file `HelloTest.java` but with a different commit message:

```
1 package aston.github.fr;
2
3 public class HelloTest {
4
5     public static void main(String[] args) {
6
7         System.out.println("Commit version 2");
8     }
9 }
```

Below the code editors, the interface includes tabs for **Problems**, **@ Javadoc**, **Declaration**, and **History**. The **History** tab is currently selected, showing the commit history for the project **AstonProject [GitHubRepo]**:

Id	Message
eb5516f	master (HEAD) Commit version 2
81c0bb0	origin/master Commi version 1
58e9652	Initialisation des sources

Egit : les différences entre versions

- Lors des commits, les modifications sont visibles :

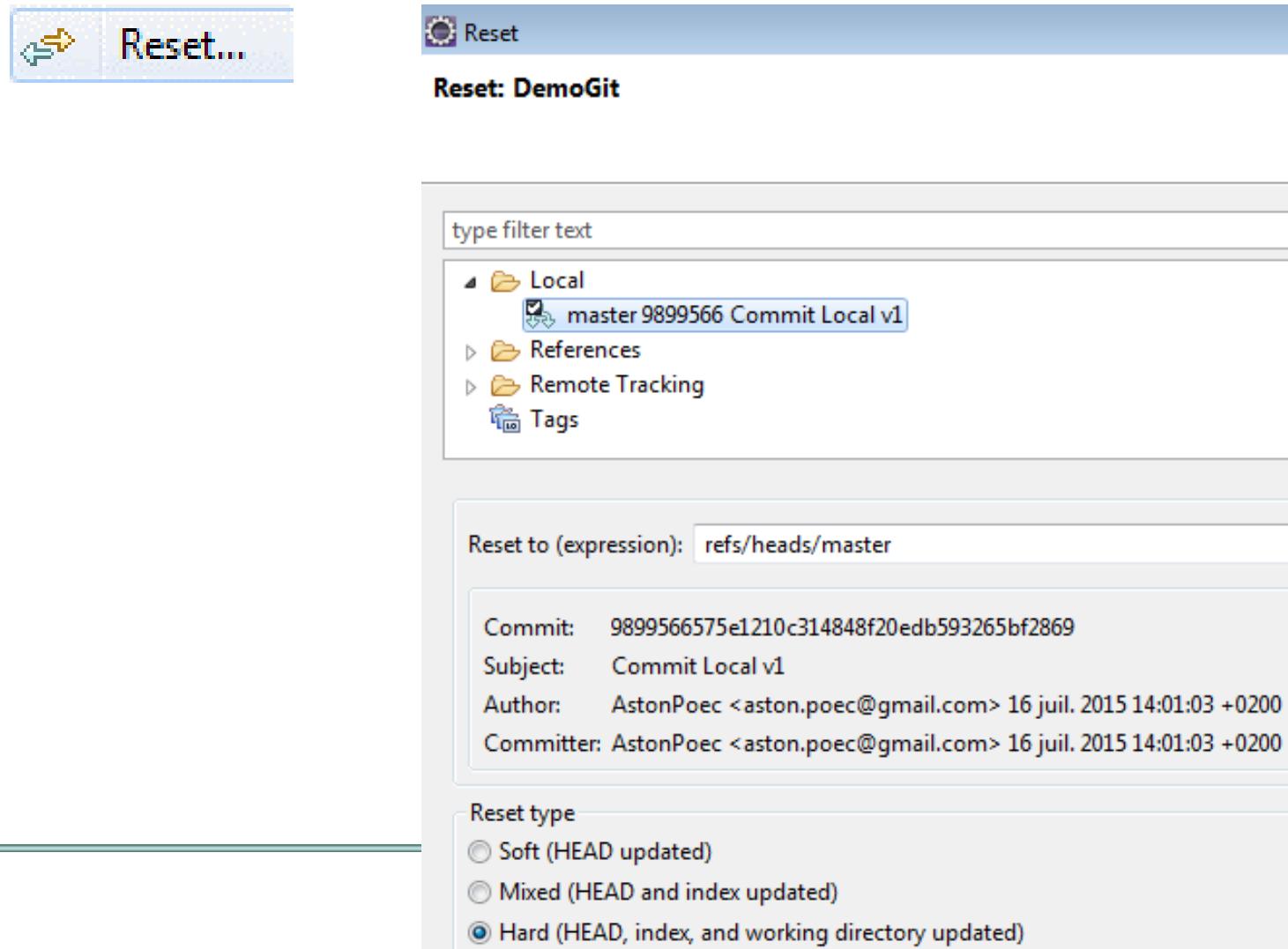
```
commit eb5516f264a66f4818a7cca4cecd309aa626faf1
Author: AstonPoec <aston.poec@gmail.com> 2015-07-16 15:40:23
Committer: AstonPoec <aston.poec@gmail.com> 2015-07-16 15:40:23
Parent: 81c0bb09a8c5aaalad18cc0134ad979bd65388ca (Commit version 1)
Branches: master

Commit version 2

----- AstonProject/src/aston/github/fr/HelloTest.java -----
diff --git a/AstonProject/src/aston/github/fr/HelloTest.java
b/AstonProject/src/aston/github/fr/HelloTest.java
index cc6d70b..28369db 100644
--- a/AstonProject/src/aston/github/fr/HelloTest.java
+++ b/AstonProject/src/aston/github/fr/HelloTest.java
@@ -4,7 +4,7 @@
 
     public static void main(String[] args) {
 
-        System.out.println("Commit version 1");
+        System.out.println("Commit version 2");
    }
```

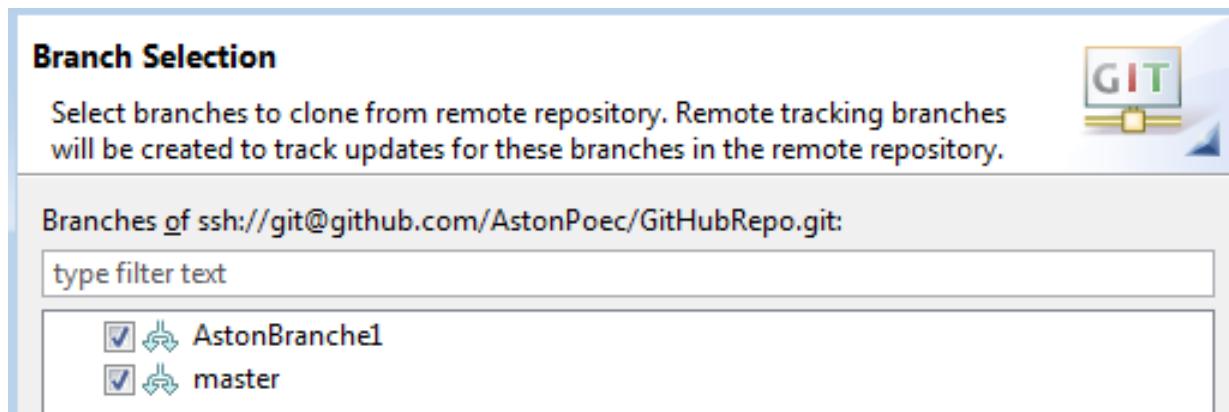
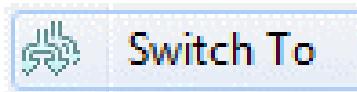
Egit : reset

- Possibilité de revenir sur une ancienne version du fichier



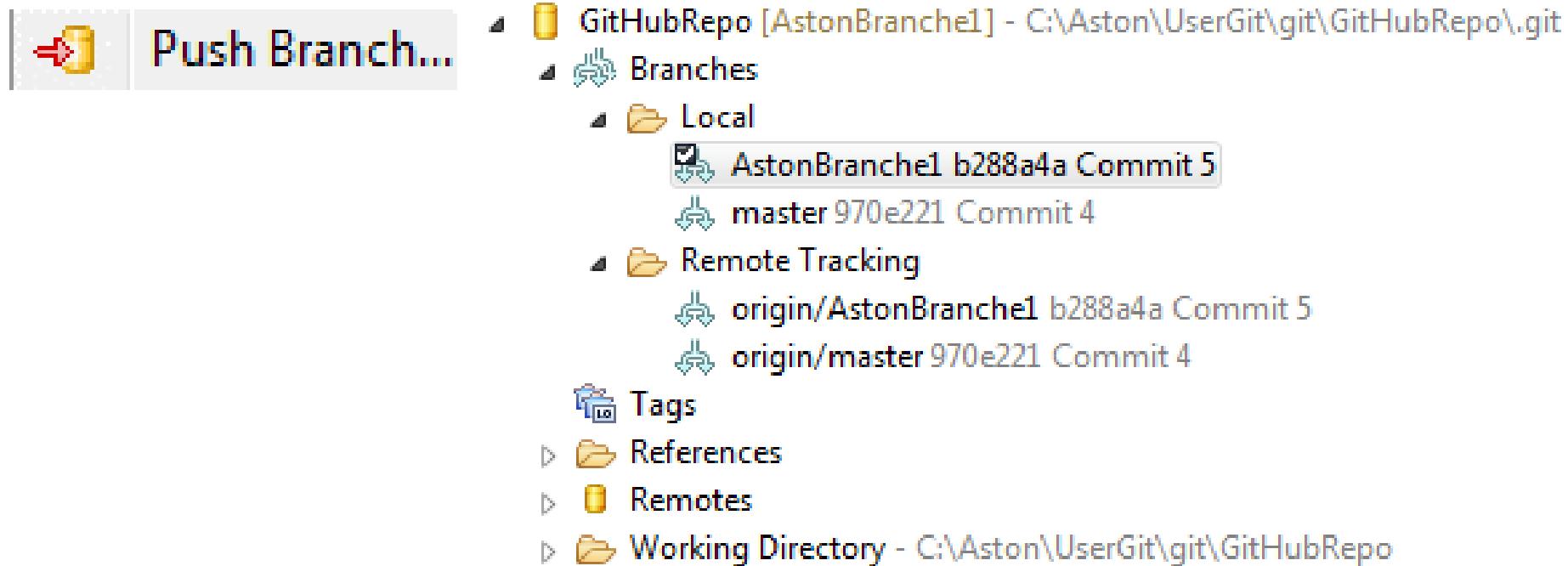
Egit : Switch To

- ❑ Permet de choisir de la branche du dépôt DISTANT avec laquelle on souhaite travailler quand il y a plusieurs branches



Egit : push une branche

- Quand le code d'une branche est stable, on peut alors intégrer ces sources dans la branche principale (Master)



Sommaire

- Généralités sur les gestionnaires de version
- Présentation de Git
- Serveur d'hébergement
- ***Conclusion***

Conclusion

	GIT	SVN	CVS
Vitesse d'exécution	Très rapide	Rapide	Moyen
Gestion de branche	Souplette, rapidité sur milliers de branche	Limitée sur dizaine de branche	Lourde et fastidieux
Travail hors connexion	Oui car système distribué	Non car système centralisé	Non car système centralisé
Sécurité de données	Commit de fichiers sécurisé grâce SHA-1	Données peuvent être corrompues	Données peuvent être corrompues
Taille mémoire RAM utilisée	Réduite	Gourmande	Gourmande
