

Logiciels de Gestion de Version

BTS SIO SLAM



git

Problématique

- Gestion d'un projet avec plusieurs développeurs devant intervenir sur plusieurs fichiers. Comment faire ?
 - Comment se passer les informations, les changements ?
 - Supports ? DD externe, clefs, cloud ?
 - Pannes des supports
 - On a fait une petite modification sur un gros fichiers de plusieurs Go. Cela ne passe pas par courrier ...
 - Qui a fait quoi ? Modification... Suppression.. Ajout...

Problématique

- Gestion d'un projet faisant intervenir plusieurs développeurs intervenant sur plusieurs fichiers. Comment faire ?
 - On veut retrouver le fichier tel qu'il était il y a une semaine
 - 4 développeurs travaillent sur un même fichier. Comment fusionner ?

Logiciel de gestion de versions

- ☑ Répond à toutes ces questions
- ☑ Pour le travail à plusieurs sur des projets de plusieurs fichiers et dossiers
- ☑ Seul il permet d'avoir un historique de vos fichiers
- ☑ Offre aussi des outils de communication

Pourquoi versionner votre code ?

- ✓ revenir en arrière sur un code valide
- ✓ pour valider une modification et une étape de votre code

Par exemple, si vous travaillez sur un formulaire de newsletter en ligne :

- Vous allez d'abord faire une 1re série de modifications pour créer la base HTML/CSS de votre formulaire.
=> Une fois ces modifications faites, vous pourrez faire un premier commit dans Git et le nommer "1re passe sur le formulaire de newsletter".
- Ensuite, vous rendrez peut-être votre formulaire dynamique avec du PHP.
=> 2e commit : "PHP ajouté".
- Enfin, un collègue vous demandera de changer le wording sur le champs du formulaire.
=> 3e commit : "Texte reformulé".



La gestion de versions : généralités, vocabulaire

- Un commit correspond à une version de votre code à un instant t
- La gestion de versions consiste à maintenir l'ensemble des versions d'un projet c'est l'historique.
- Chaque étape d'avancement d'un projet est appelée révision.
- Entre deux révisions on a des modifications —> Delta
- Ce Delta est appelé « diff » ou « patch »
- On parle parfois de versions mais il ne faut pas confondre avec la version d'un logiciel qui est une étape de distribution sous forme « finie »

L'étiquetage ou tag : marquage de version

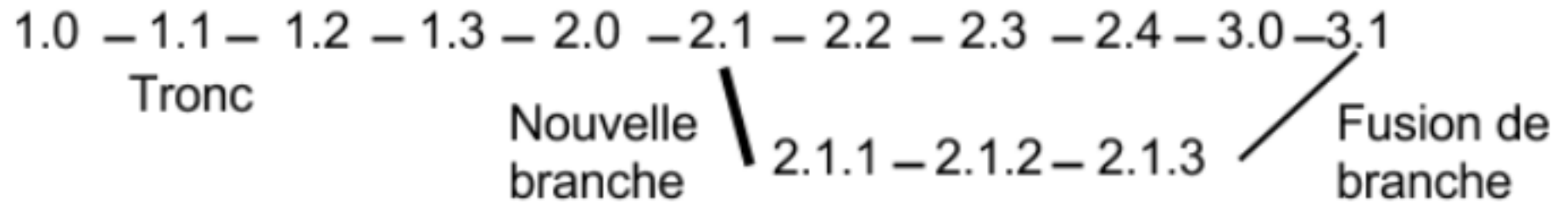
Versions Android



Vocabulaire

- ☐ versionné
- ☐ copie locale
- ☐ dépôt
- ☐ conflits
 - ☐ pessimiste -> verrouiller-modifier-déverrouiller
 - ☐ optimiste -> copier-modifier-fusionner

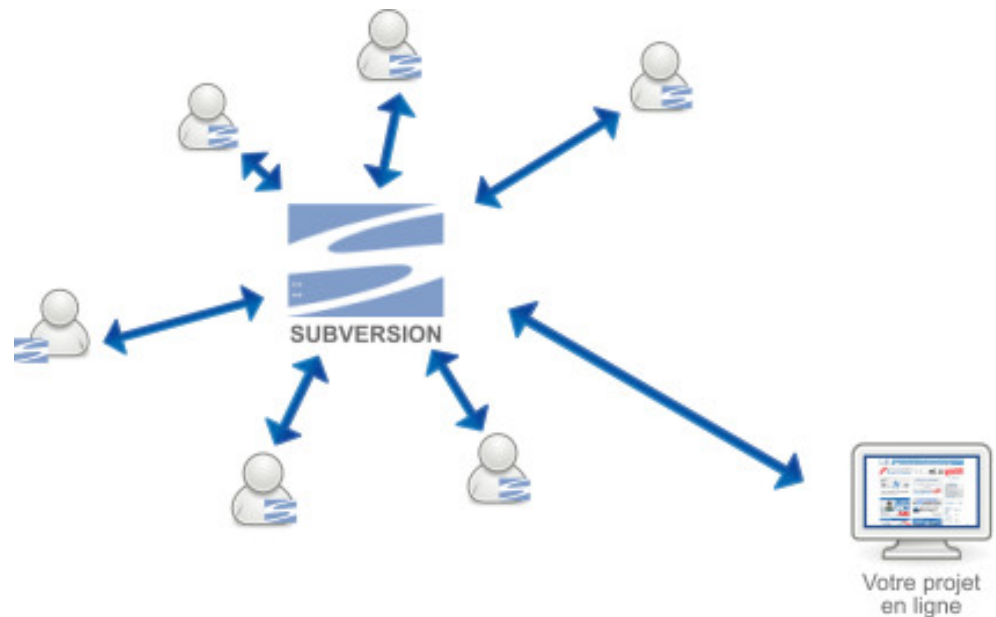
Arbre de gestion de versions



- Tronc - MainLine -> phénomène linéaire
- Branche -> nouvelles fonctionnalités, test, maintenance d'anciennes versions.
- Fusion -> Nouvelle version, intégration

Gestion de versions centralisée et décentralisée

- Gestion de versions centralisée : un dépôt fait référence
 - CVS
 - SVN -> Subversion



Gestion de versions centralisée et décentralisée

- Gestion de versions décentralisée

- Git
- Mercurial
- Bazaar



Bazaar

Git

Système de gestion de versions distribué



git

Définitions

- Git permet d'enregistrer l'évolution d'une arborescence de fichiers (le **répertoire de travail**) au sein d'un **dépôt** (le sous répertoire .git).
- **Système de gestion de versions distribué :**
dans les systèmes distribués, il n'est pas nécessaire d'avoir un dépôt central.
 - Chaque intervenant dispose d'une copie complète du dépôt avec toute l'historique et peut effectuer autant de changement locaux qu'il le souhaite, sans devoir en référer au «serveur».
 - Dans la pratique, on conserve des dépôts centraux pour faciliter les échanges et servir de référence commune. Mais les développeurs sont maître de ce qu'ils envoient sur ces dépôts. Le partage et l'enregistrement sont deux choses différentes !

Particularités de GIT

- très rapide
- sait travailler par branche de manière très flexible;
- assez complexe
- est à l'origine prévu pour Linux (mais fonctionne sous Mac OS/X et Windows)
- Sites collaboratifs : GitHub, Gitorious

Installation de Git

About

Documentation

Blog

Downloads

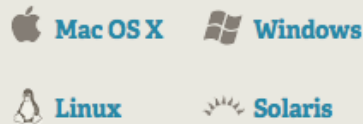
GUI Clients

Logos

Community

The entire [Pro Git book](#) written by Scott Chacon is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Downloads



Older releases are available and the [Git source repository](#) is on GitHub.



GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the [web interface](#).

Installer Git

Puis tester si git est bien sur votre système :

```
macbook-de-seb:~ sebastien$ git --version
git version 2.10.0
macbook-de-seb:~ sebastien$
```

Download and Install Git

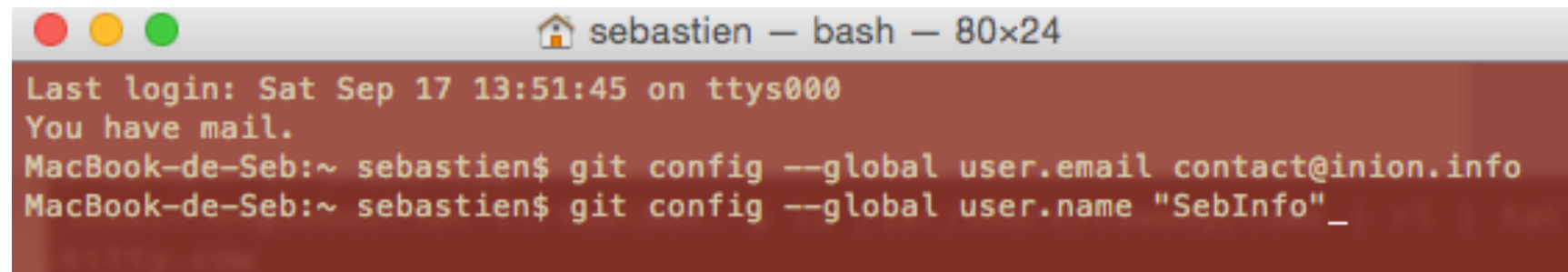
At the heart of GitHub is an open source version control system (VCS) called Git*. Created by the same team that created Linux, Git is responsible for everything GitHub related that happens locally on your computer.

**If you don't already know what Git is, [take a crash course](#).*

Download and install [the latest version of Git](#).

Tip: Git won't add an icon to your dock, it's not that sort of application.

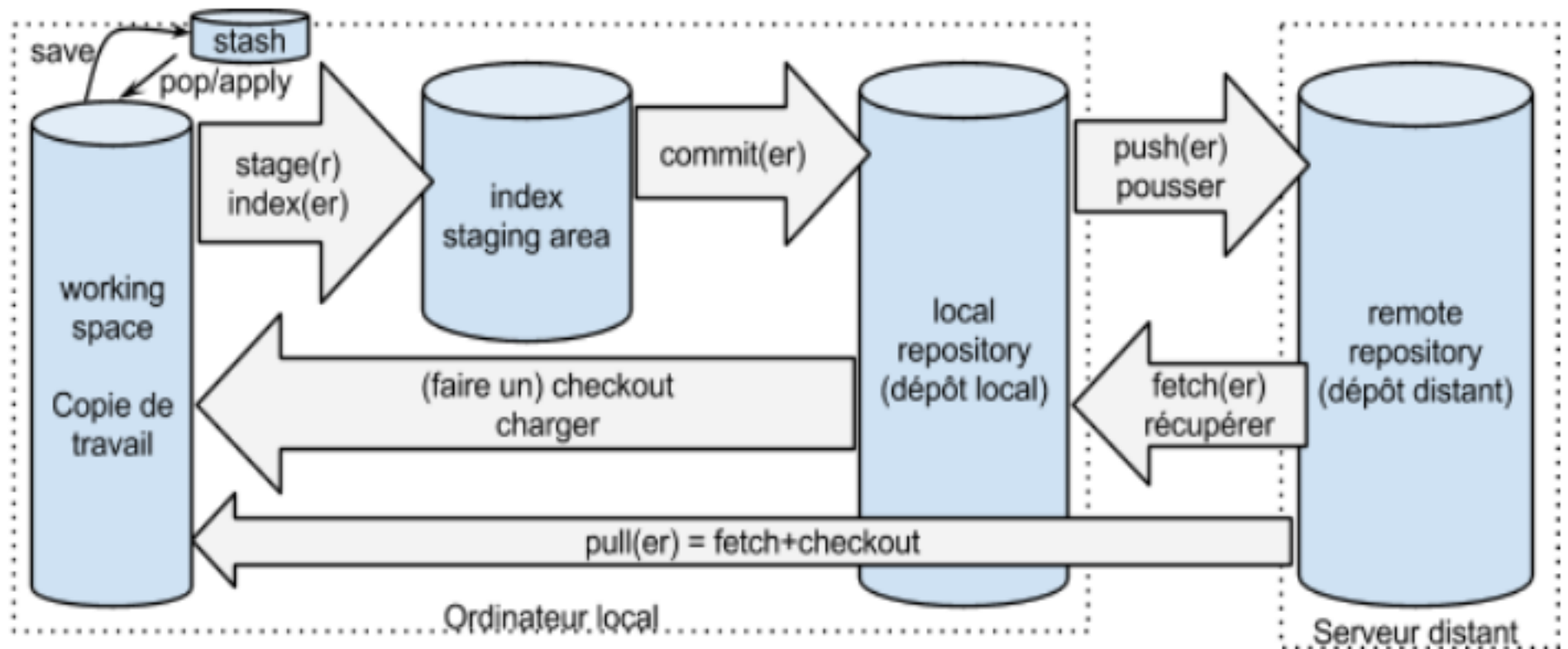
Première Etape : indiquer votre identité

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, and a home icon followed by the text 'sebastien — bash — 80x24' on the right. The terminal content is as follows:

```
Last login: Sat Sep 17 13:51:45 on ttys000
You have mail.
MacBook-de-Seb:~ sebastien$ git config --global user.email contact@inion.info
MacBook-de-Seb:~ sebastien$ git config --global user.name "SebInfo" _
```

Vocabulaire spécifique à Git

Vocabulaire spécifique à GIT



Création d'un dépôt

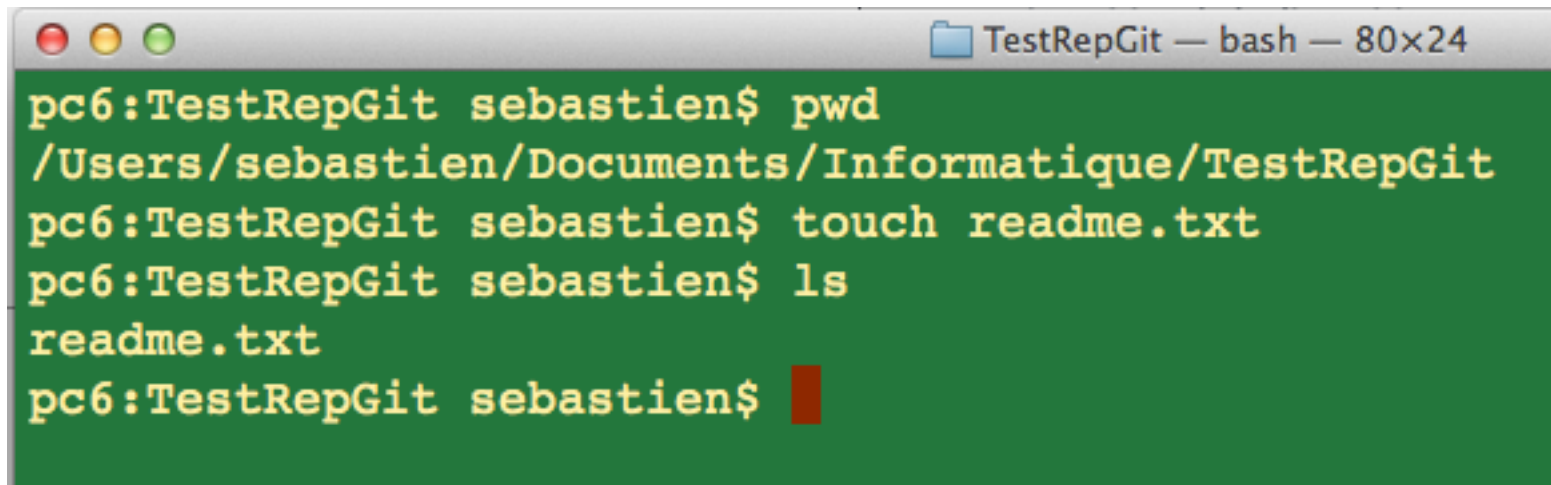
On se place dans le répertoire que l'on veut (ici je viens de le créer) :

- commande **git init**
- Ceci crée un répertoire **.git** avec les méta-informations nécessaire à Git
- Le répertoire de travail est **TestRepGit**

```
pc6:Informatique sebastien$ ls
Android          TestRepGit
pc6:Informatique sebastien$ cd TestRepGit/
pc6:TestRepGit sebastien$ ls
pc6:TestRepGit sebastien$ git init
Initialized empty Git repository in /Users/sebastien/Documents/Informatique/TestRepGit/.git/
pc6:TestRepGit sebastien$ ls
pc6:TestRepGit sebastien$ ls -l
pc6:TestRepGit sebastien$ ls -al
total 0
drwxr-xr-x  3 sebastien  staff  102 12 mar 15:07 .
drwxr-xr-x  4 sebastien  staff  136 12 mar 15:07 ..
drwxr-xr-x 10 sebastien  staff  340 12 mar 15:07 .git
pc6:TestRepGit sebastien$
```

Troisième Etape : création d'un fichier

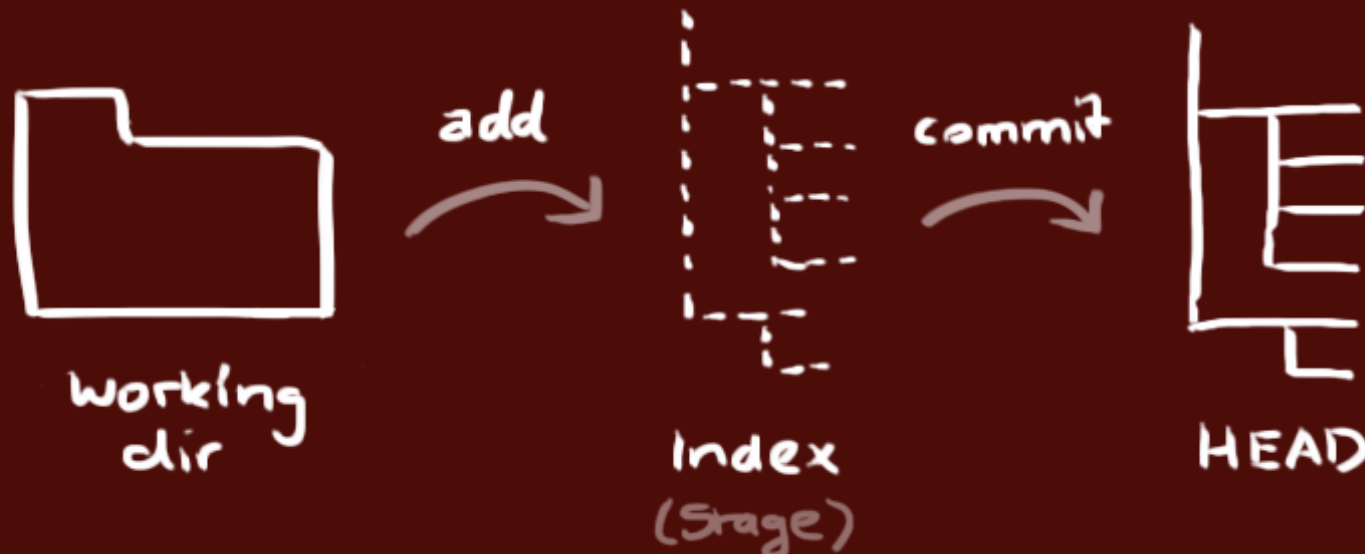
On se place dans le répertoire TestRepGit :

A terminal window with a grey title bar containing three window control buttons (red, yellow, green) on the left and a folder icon followed by the text 'TestRepGit — bash — 80x24' on the right. The terminal has a green background and yellow text. It shows a sequence of commands and their outputs: 'pwd' returns the full path to the directory, 'touch readme.txt' creates the file, and 'ls' lists the file. The prompt 'pc6:TestRepGit sebastien\$' is repeated for each command.

```
pc6:TestRepGit sebastien$ pwd
/Users/sebastien/Documents/Informatique/TestRepGit
pc6:TestRepGit sebastien$ touch readme.txt
pc6:TestRepGit sebastien$ ls
readme.txt
pc6:TestRepGit sebastien$
```

arbres

votre dépôt local est composé de trois "arbres" gérés par git. le premier est votre **espace de travail** qui contient réellement vos fichiers. le second est un **Index** qui joue un rôle d'espace de transit pour vos fichiers et enfin **HEAD** qui pointe vers la dernière validation que vous ayez fait.

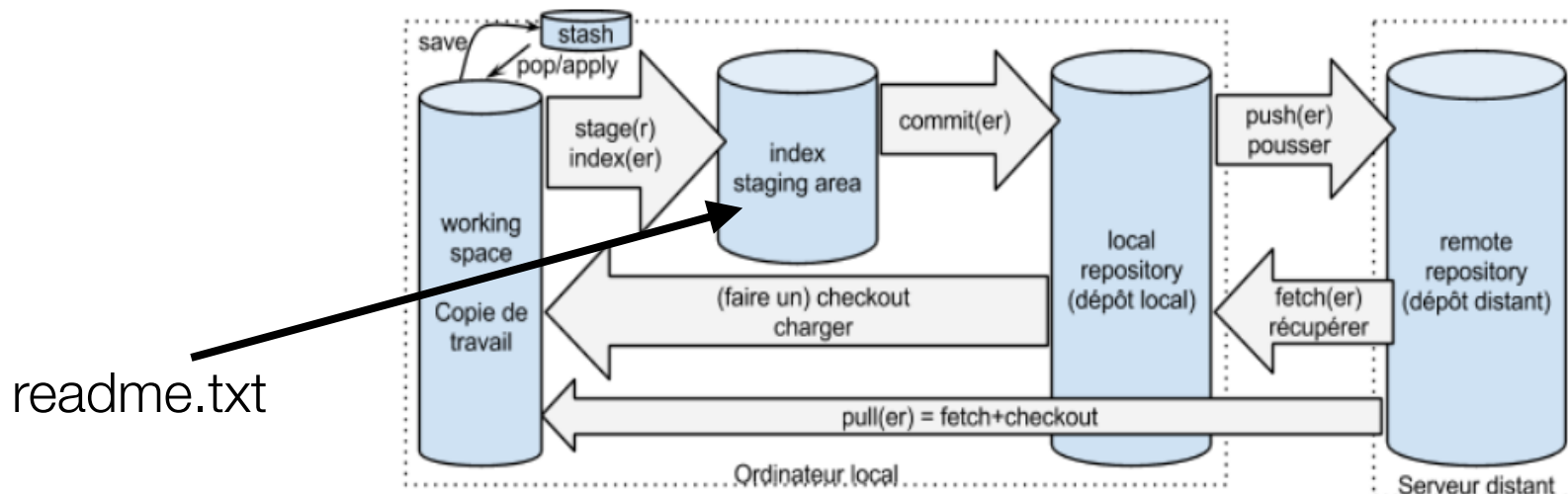


Ajouter un fichier à l'Index

- Pour l'instant votre fichier readme.txt n'est pas connu de Git car il ne figure pas dans l'index. Il est dans le working space.
- `git add fichier`: ajoute le fichier à l'index

```
pc6:TestRepGit sebastien$ git add readme.txt
```

Vocabulaire spécifique à GIT



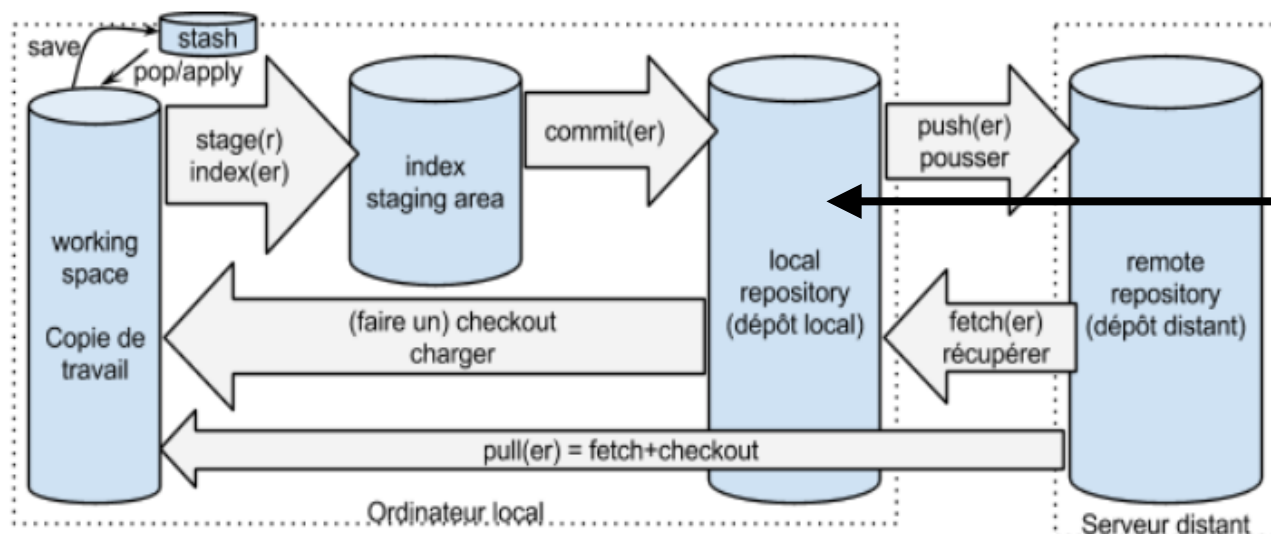
HEAD

- Votre fichier est présent dans l'index maintenant mais n'est pas encore dans le HEAD. Il n'a pas été comité !

- Pour le valider :

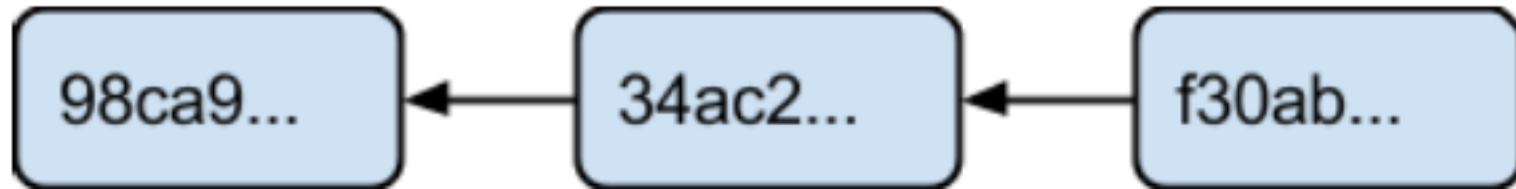
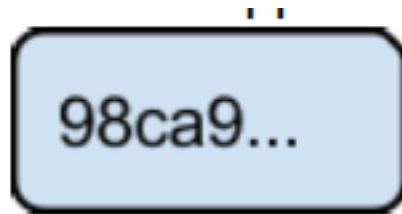
```
pc6:TestRepGit sebastien$ ls
readme.txt
pc6:TestRepGit sebastien$ git commit -m "Premier commit"
[master (root-commit) 79663ee] Premier commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 readme.txt
pc6:TestRepGit sebastien$
```

Vocabulaire spécifique à GIT



readme.txt

Git : Commit



Un dépôt Git est finalement un graphe de commits

Modifier le fichier

- Je vais modifier le contenu du fichier.

```
pc6:TestRepGit sebastien$ cat readme.txt
Bonjour amis terriens !
pc6:TestRepGit sebastien$ git status
# On branch master
# 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:   readme.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
pc6:TestRepGit sebastien$ git diff
diff --git a/readme.txt b/readme.txt
index e69de29..fdb33ae 100644
--- a/readme.txt
+++ b/readme.txt
@@ -0,0 +1 @@
+Bonjour amis terriens !
pc6:TestRepGit sebastien$
```

Récupérer la dernière version du HEAD

- `git checkout -- nomdufichier`

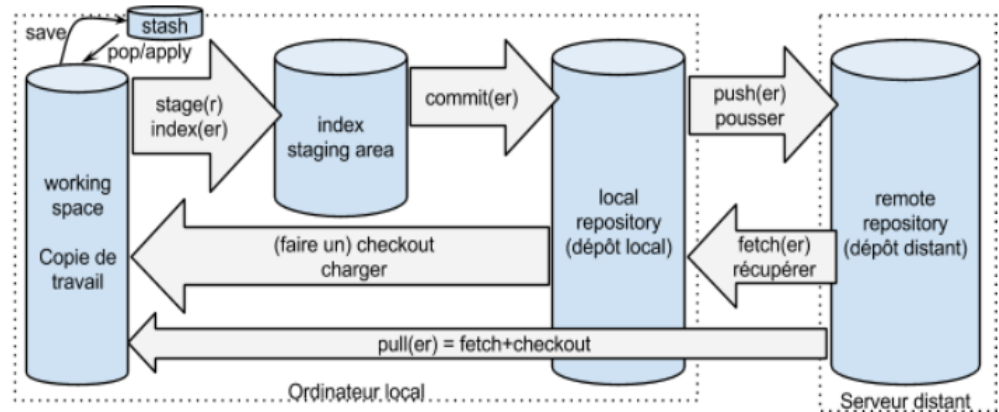
```
pc6:TestRepGit sebastien$ git diff
diff --git a/readme.txt b/readme.txt
index e69de29..fdb33ae 100644
--- a/readme.txt
+++ b/readme.txt
@@ -0,0 +1 @@
+Bonjour amis terriens !
pc6:TestRepGit sebastien$ git checkout -- readme.txt
pc6:TestRepGit sebastien$ cat readme.txt
pc6:TestRepGit sebastien$
```

Création d'un dépôt distant avec Github

En effet pour l'instant le fichier est dans le HEAD de votre dépôt local.

Si vous voulez le partager avec d'autres développeurs il faut le transférer sur un dépôt distant.

Vocabulaire spécifique à GIT



Build software better, together.

Powerful collaboration, code review, and code management for open source and private projects. Need private repositories?

[Upgraded plans start at \\$7/mo.](#)

Use at least one lowercase letter, one numeral, and seven characters.

[Sign up for GitHub](#)

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#).

Why you'll love GitHub.



[Powerful features](#) to make software development more collaborative.

GitHub

Création d'un compte

Création du dépôt distant


Owner **Repository name**


PUBLIC   **SebInfo** / **TestBTSSIO** ✓

Great repository names are short and memorable. Need inspiration? How about **tripping-octo-shame**.

Description (optional)

TestPourLeCours

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

attention
ne pas cocher

Création du dépôt distant

Create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/SebInfo/TestBTSSIO.git
git push -u origin master
```

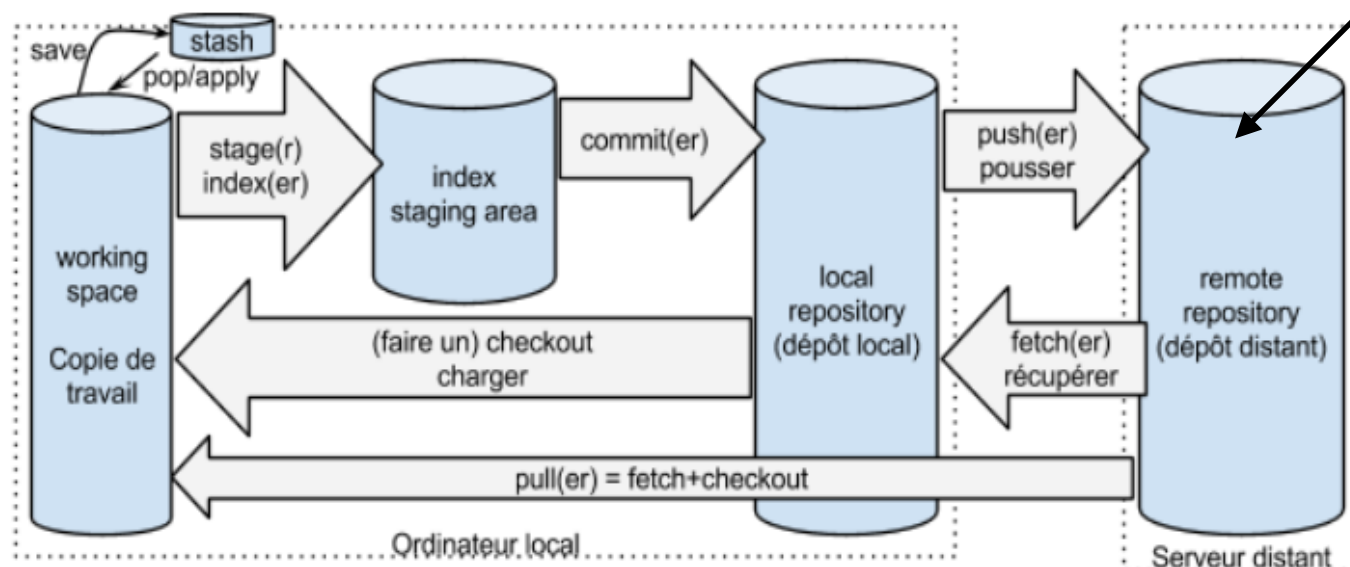
Push an existing repository from the command line

```
git remote add origin https://github.com/SebInfo/TestBTSSIO.git
git push -u origin master
```

On envoie le fichier avec le push


```
pc6:TestRepGit sebastien$ git remote add origin https://github.com/SebInfo/TestBTSSIO.git
pc6:TestRepGit sebastien$ push -u origin master
-bash: push: command not found
pc6:TestRepGit sebastien$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 212 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/SebInfo/TestBTSSIO.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
pc6:TestRepGit sebastien$
```


Vocabulaire spécifique à GIT




Le fichier est bien sur le dépôt distant

PUBLIC

 **SebInfo / TestBTSSIO**


 Unwatch ▾


1


 Star


0


TestPourLeCours — Edit


 1 commit

 1 branch


 0 releases


 1 contributor


 branch: master ▾

TestBTSSIO / 

Premier commit


 **SebInfo** authored 34 minutes ago


latest commit 79663ee797 


 [readme.txt](#)

Premier commit


34 minutes ago

 **readme.txt**


 **Code**


 [Issues](#)


0


 [Pull Requests](#)


0

 [Wiki](#)

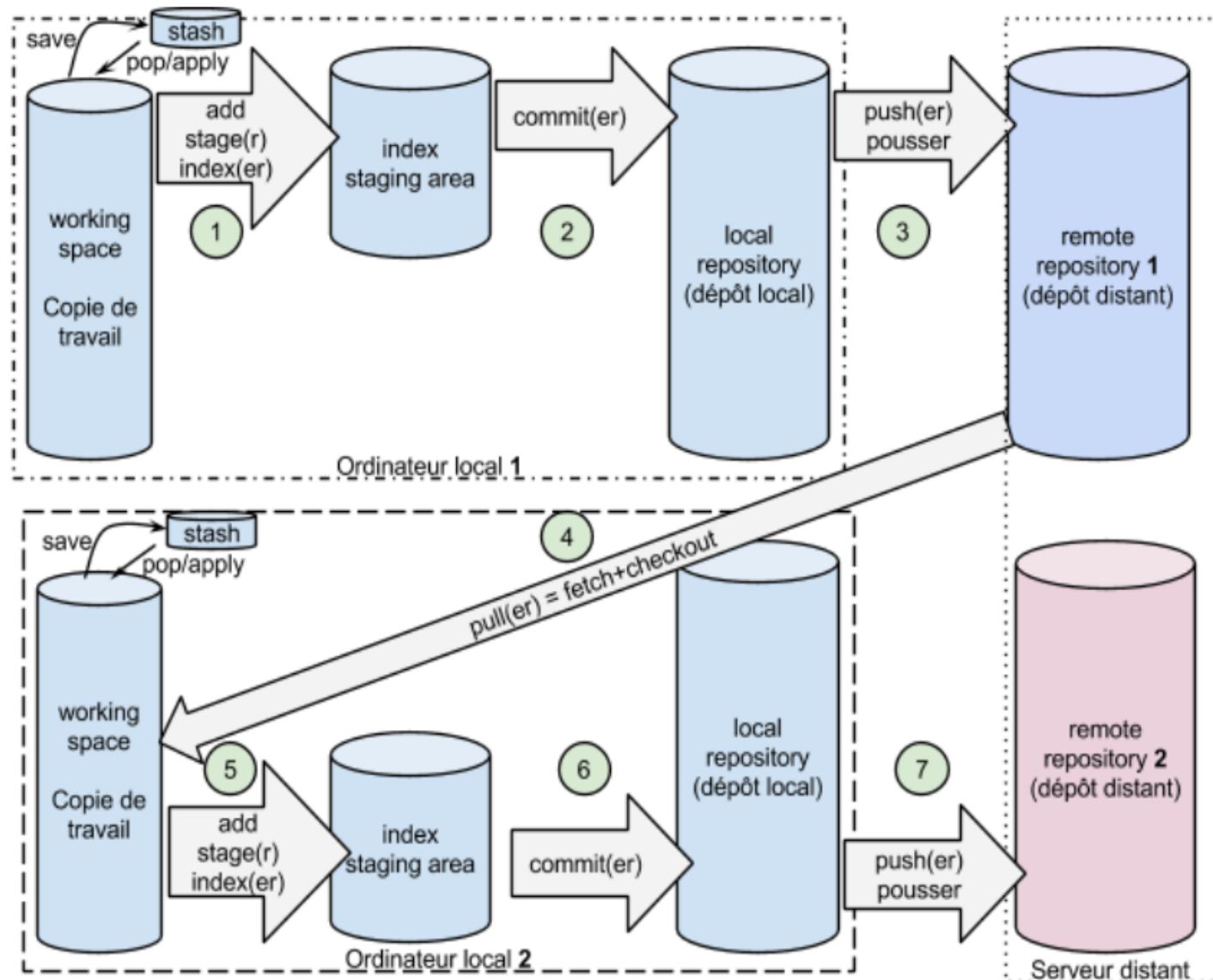
 [Pulse](#)

 [Graphs](#)

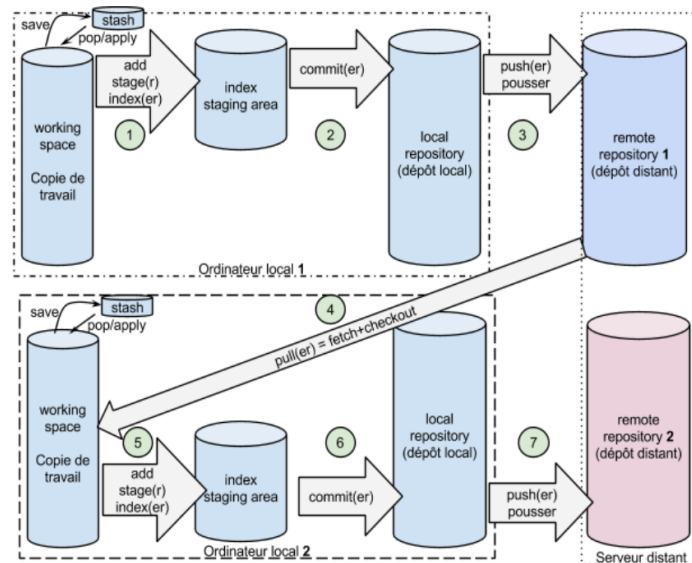
 [Network](#)

 [Settings](#)

Ordre de création des dépôts, clonage de dépôt et fork de dépôt.



Ordre de création des dépôts, clonage de dépôt et fork de dépôt.



- Préférable de cloner le dépôt distant
- Un fork est un clonage entre deux dépôts distants -> branche !

Branches

branches

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche *master* est la branche par défaut quand vous créez un dépôt. Utilisez les autres branches pour le développement et fusionnez ensuite à la branche principale quand vous avez fini.



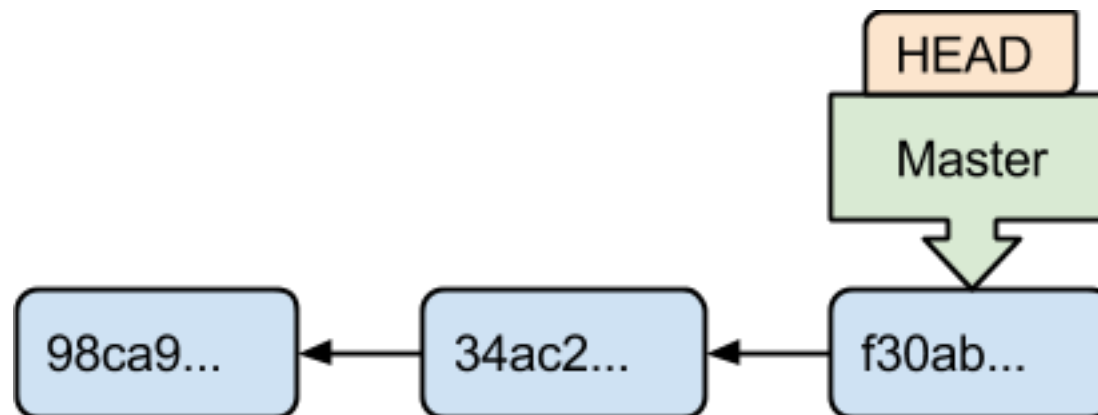
Git et les branches

Première branche : **Master**
Une branche pointe sur un commit



Git et les branches

Un pointeur spécial nommé HEAD pointe vers la branche courante de travail !

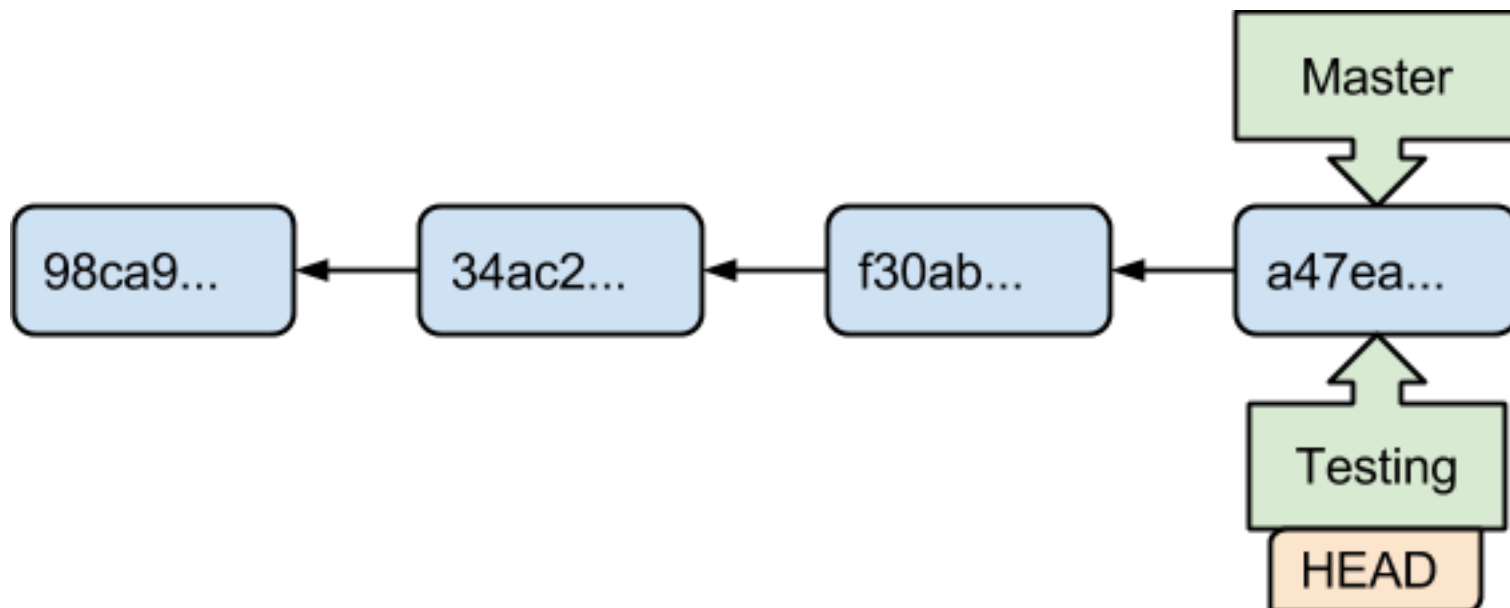


Git et les branches

Création d'une nouvelle branche

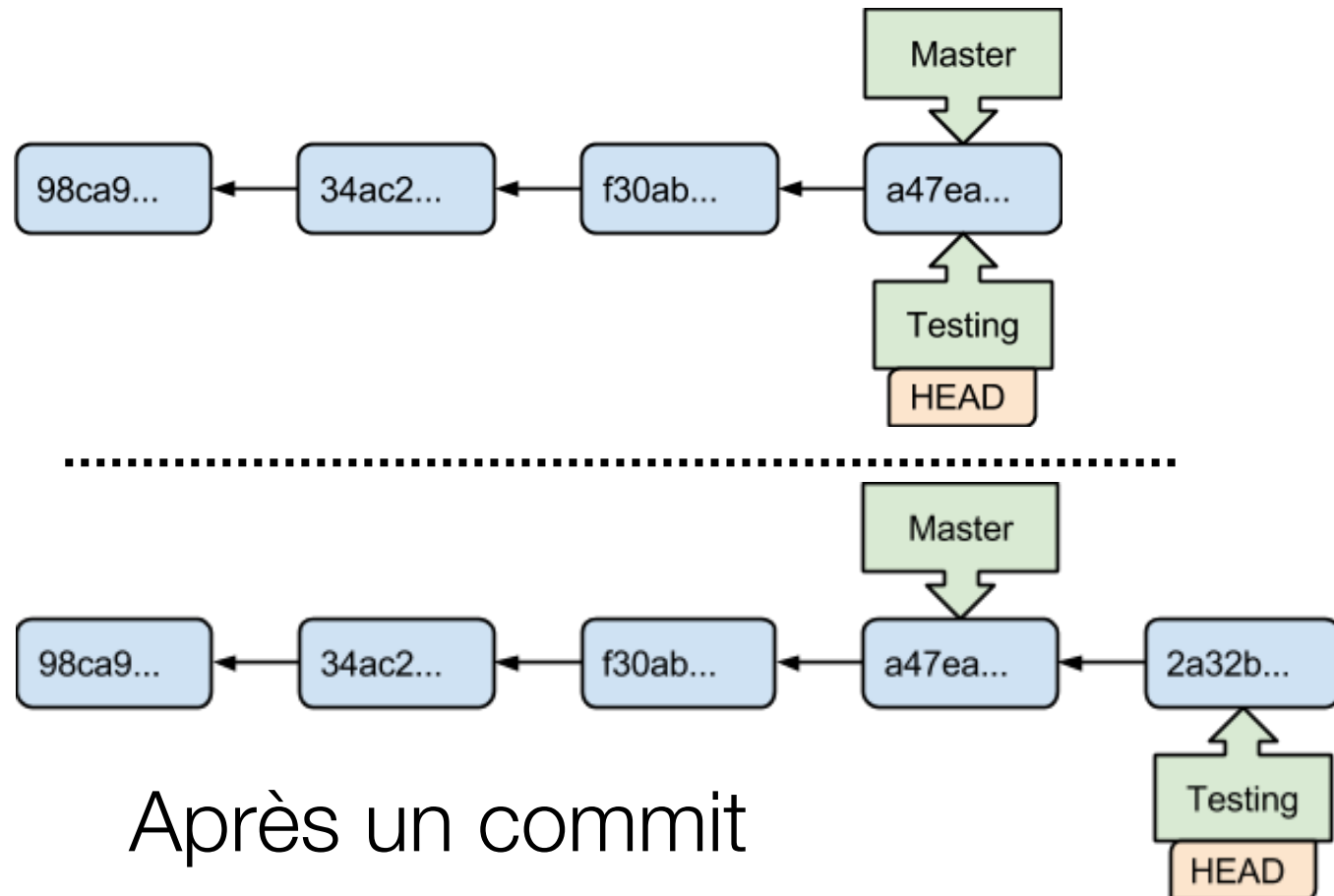
\$git branch Testing

\$git checkout Testing



Git et les branches

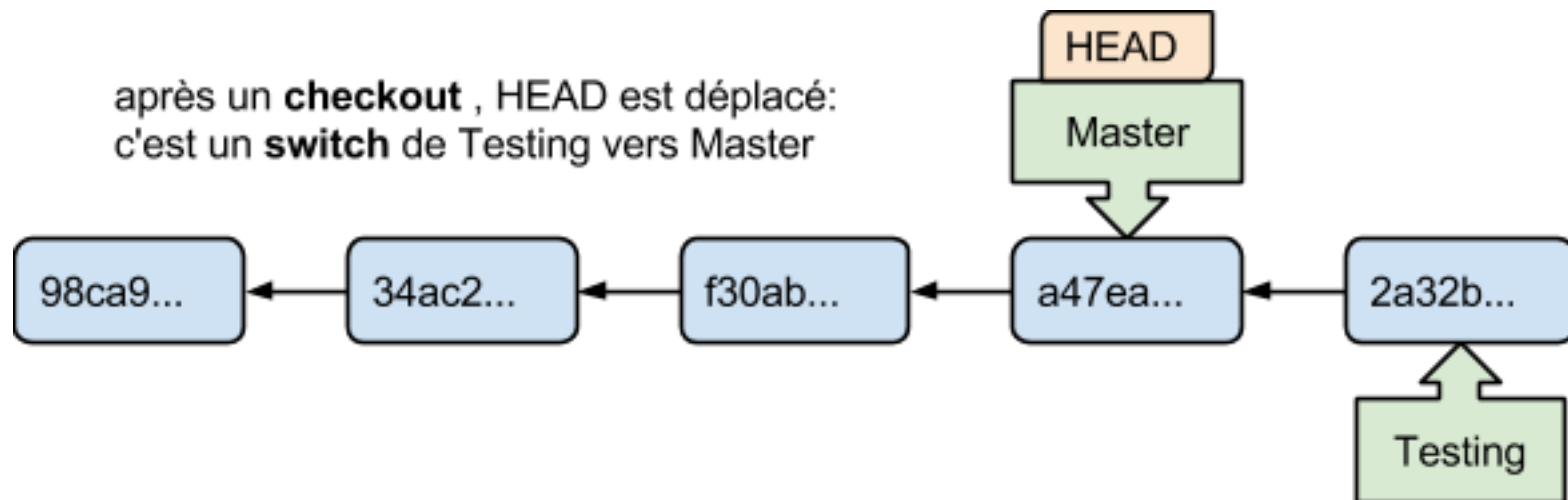
Les commits ne font se déplacer que la branche active (pointée par HEAD)



Git et les branches

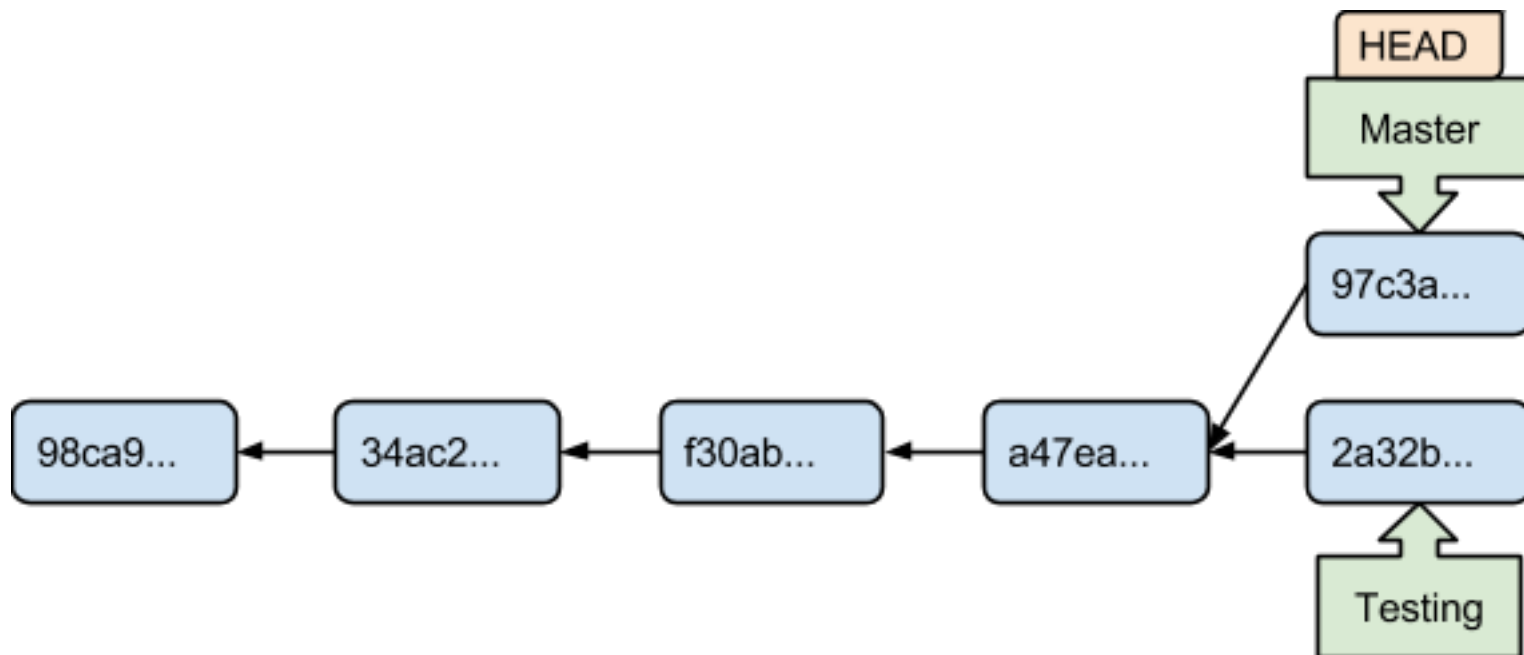
\$git checkout Master

après un **checkout** , HEAD est déplacé:
c'est un **switch** de Testing vers Master



Git et les branches

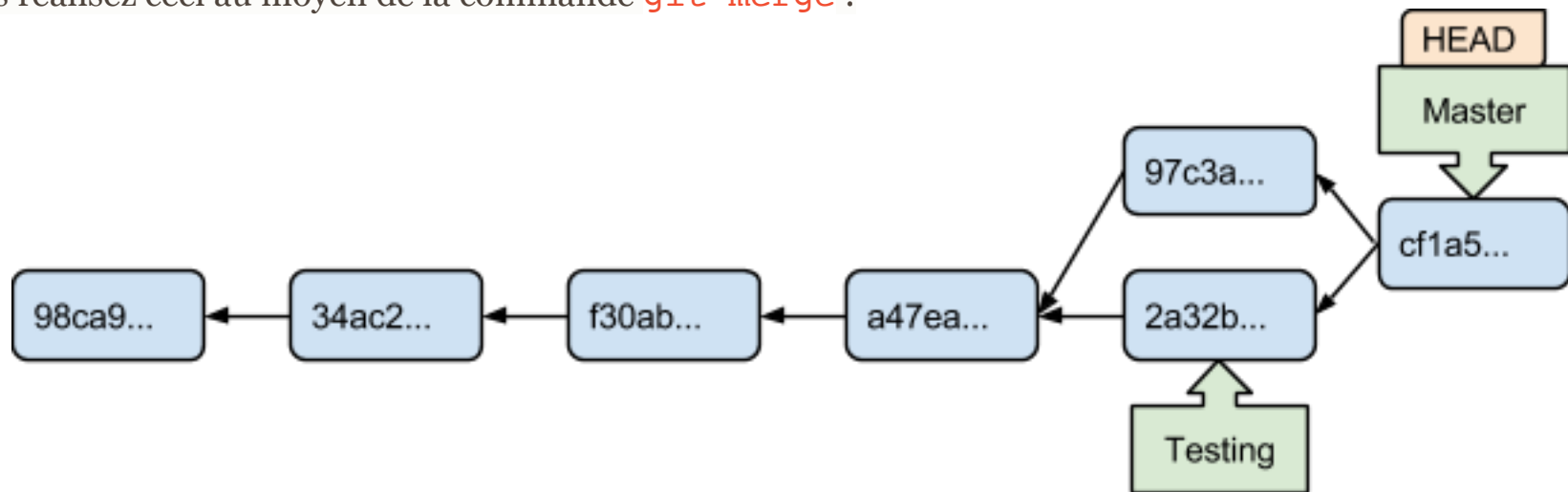
Maintenant que le **HEAD** est sur Master, faisons un commit, il va se créer une divergence et on distingue bien les deux “branches” de l’arbre:



Git et les branches

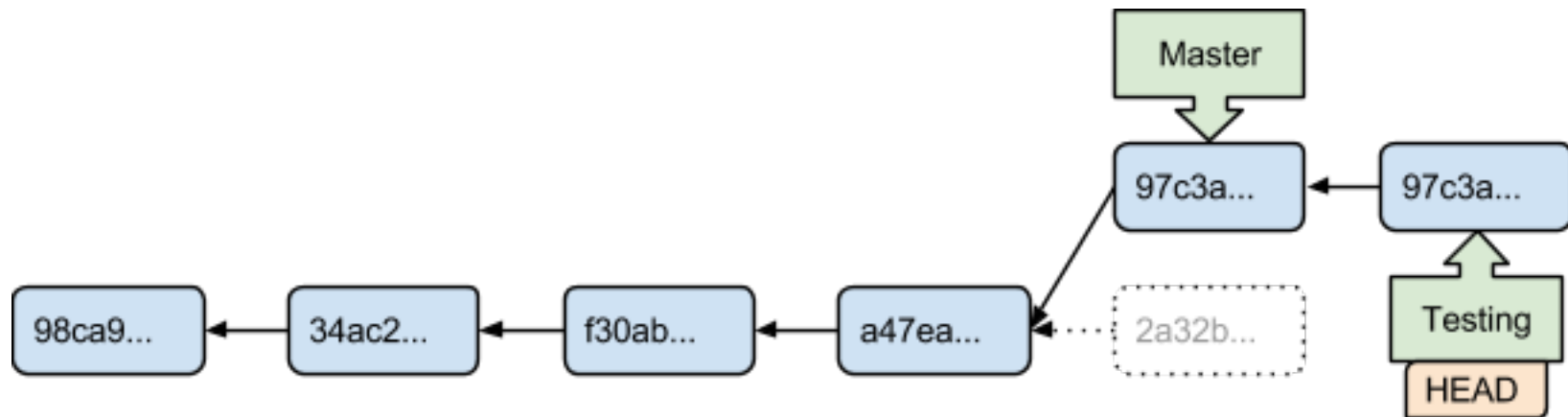
Si on fusionne **Testing** dans **Master** on obtient :

Vous réalisez ceci au moyen de la commande `git merge` :

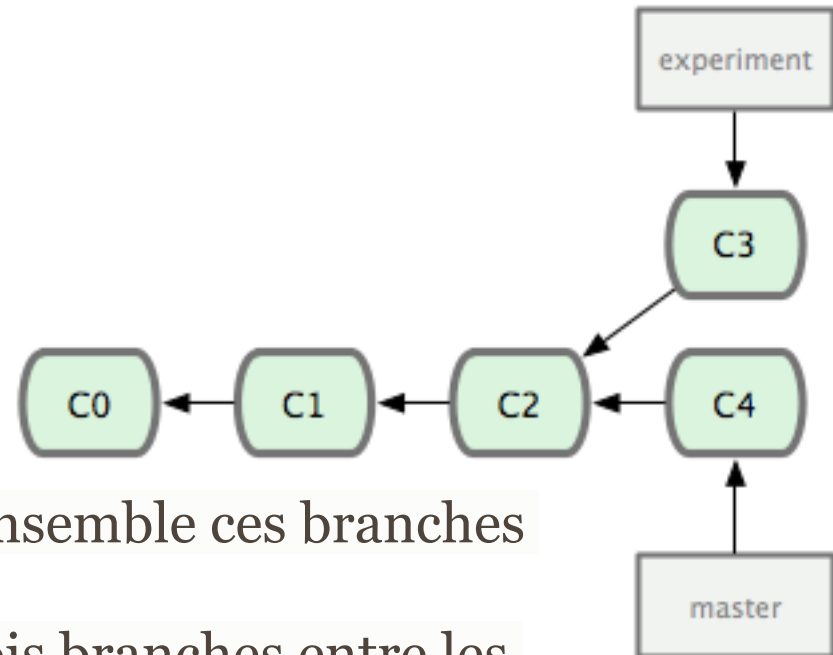


Git et les branches

On rebase **Testing** dans **Master** on obtient :



Différence entre fusion et rebaser

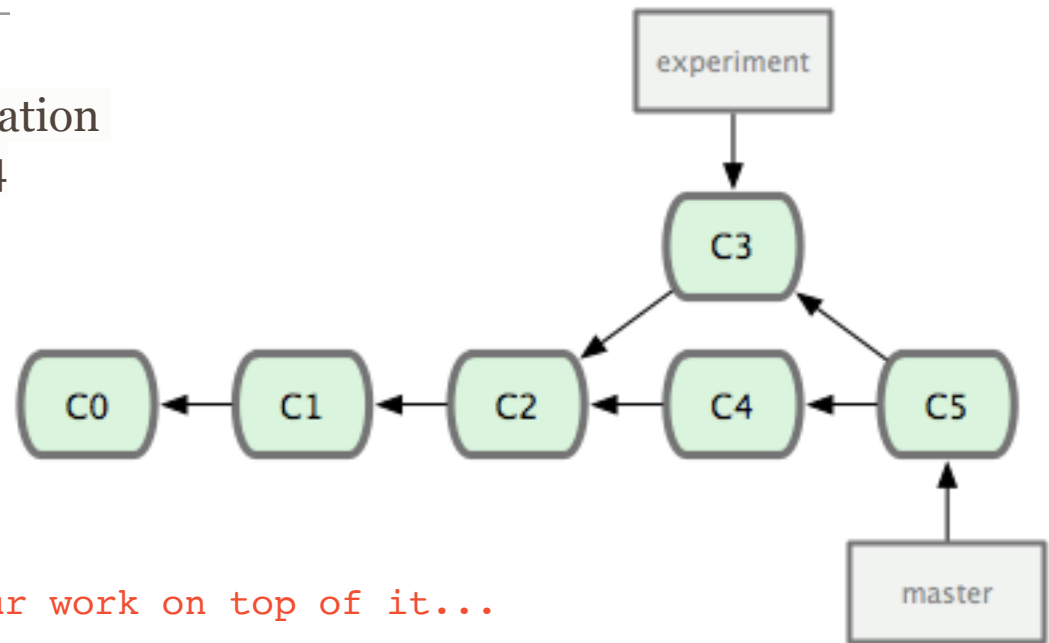


Le moyen le plus simple pour intégrer ensemble ces branches est la fusion via la commande **merge**.

Cette commande réalise une fusion à trois branches entre les deux derniers instantanés de chaque branche (C3 et C4) et l'ancêtre commun le plus récent (C2), créant un nouvel instantané (et un *commit*)

Différence entre fusion et rebaser

Rebaser : prendre le patch de la modification introduite en C3 et le réappliquer sur C4



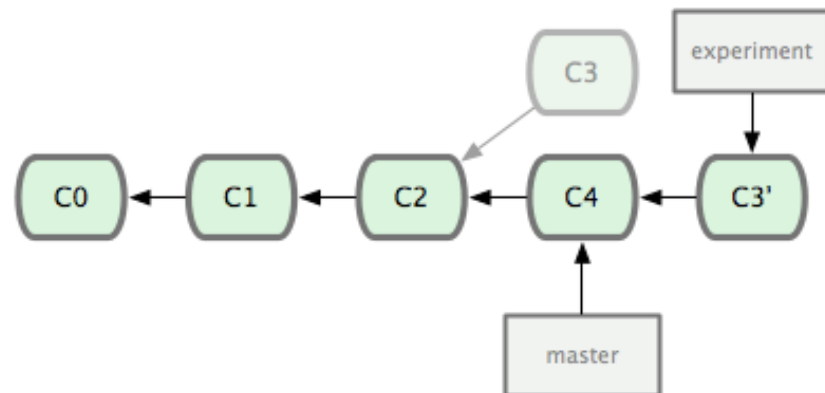
```
$ git checkout experience
```

```
$ git rebase master
```

First, rewinding head to replay your work on top of it...

Applying: added staged command

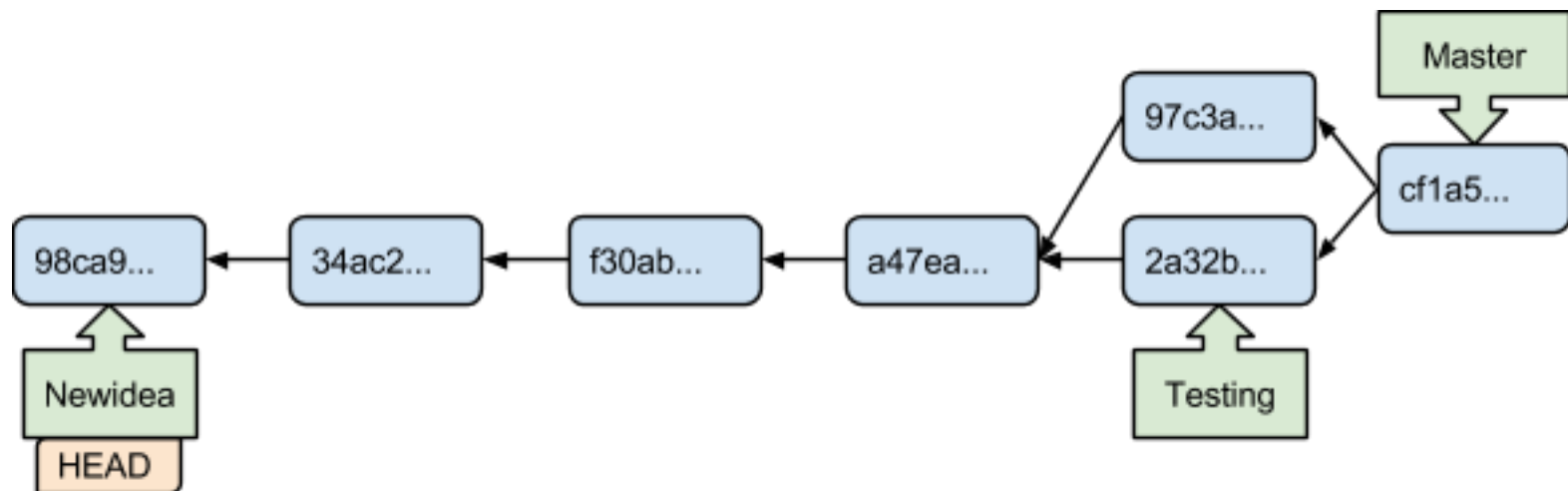
Rebaser les modifications introduites par C3 sur C4.



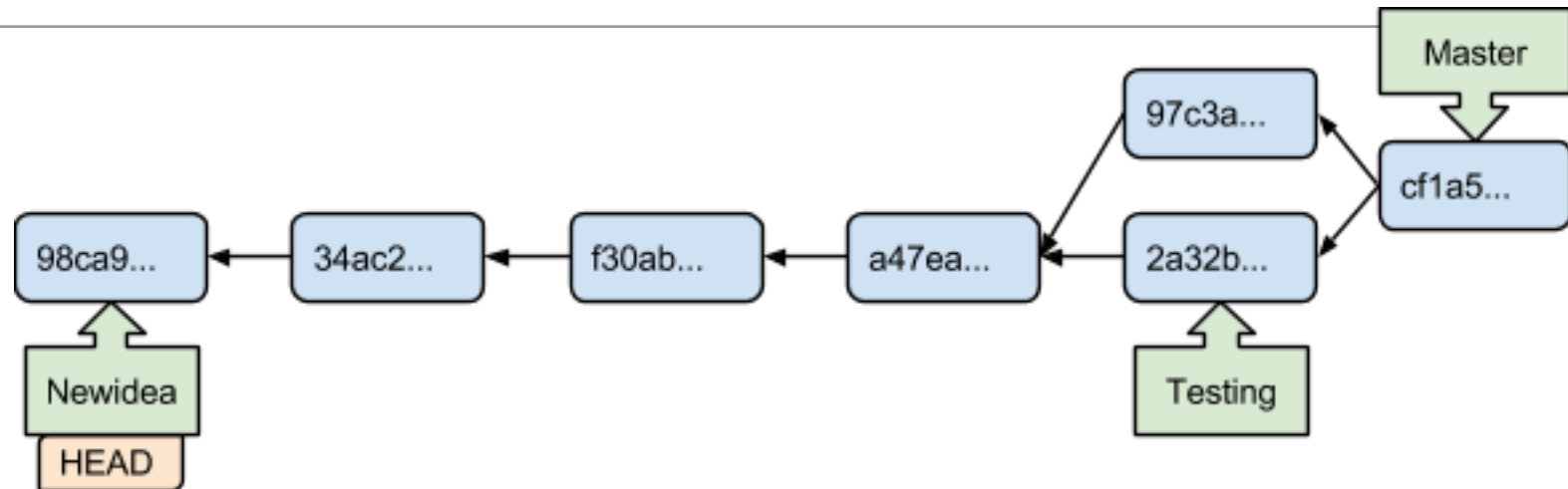
Git et les branches

Important: en général si on veut repartir d'une ancienne version il est préférable de créer une nouvelle branche.

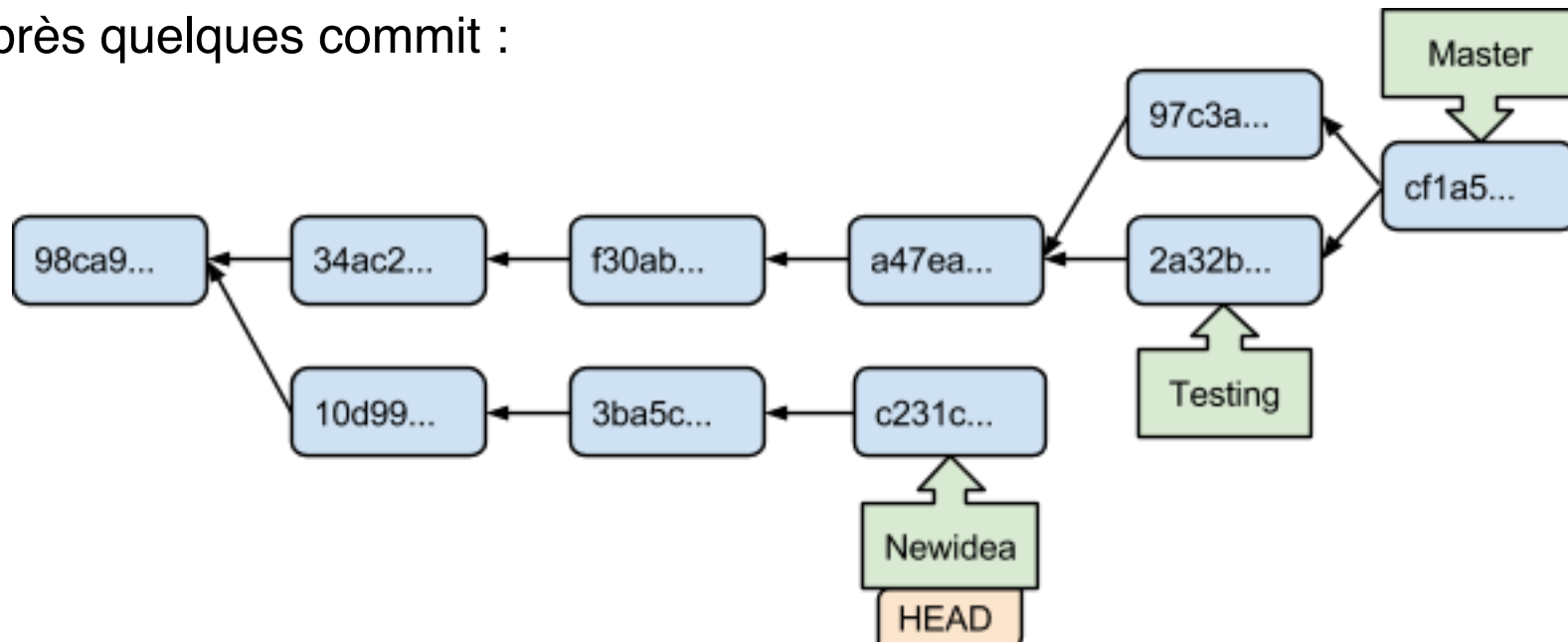
Par exemple repartons de la version originale car nous avons une nouvelle idée et créons une branche Newidea !



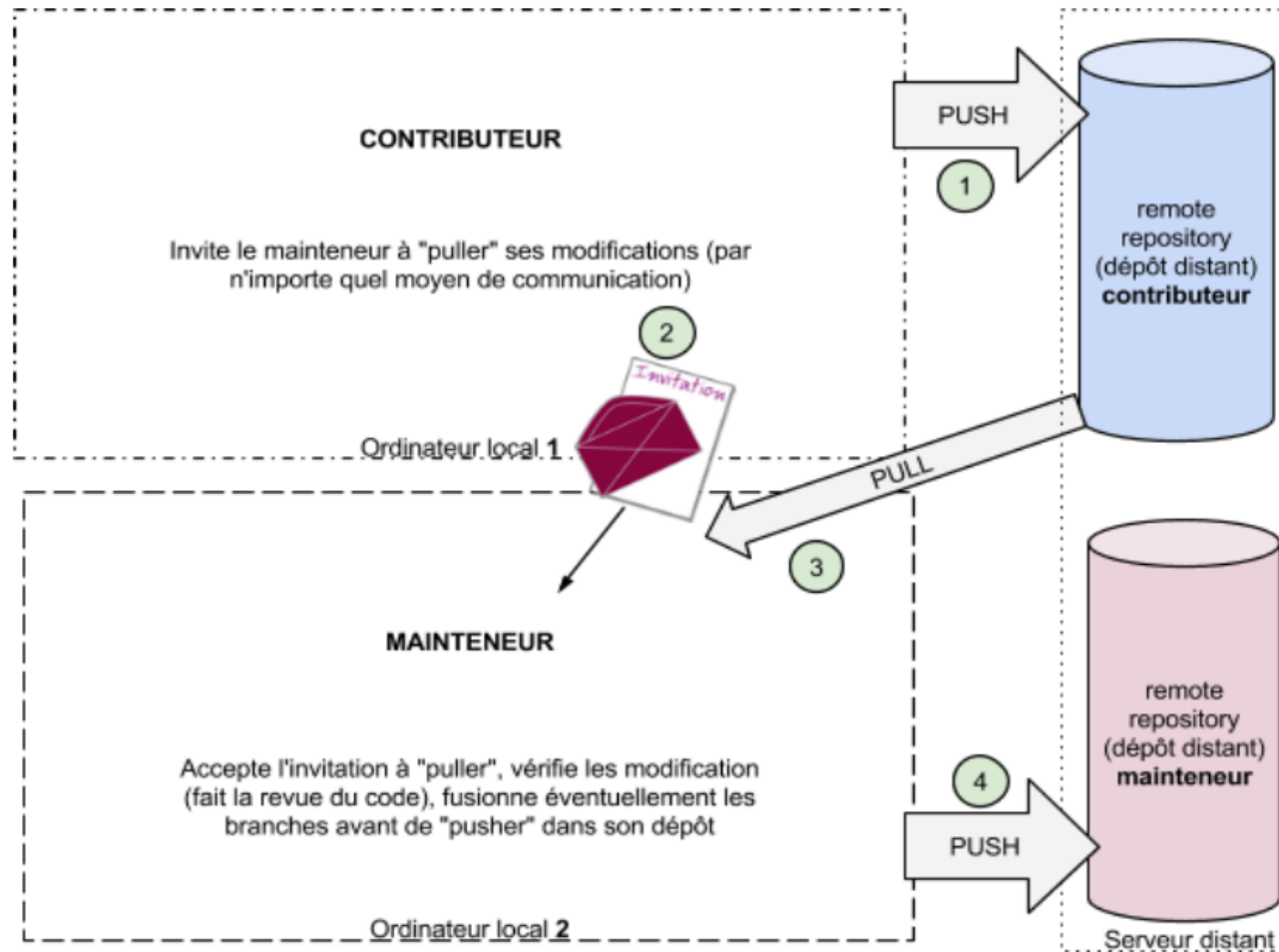
Git et les branches



Après quelques commit :



Pull request



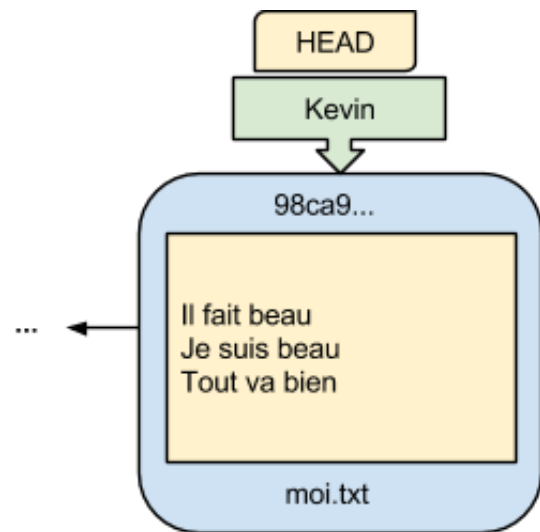
Ce schéma est simplificateur car en général les deux personnes vont travailler sur des branches différentes.

Le forking workflow

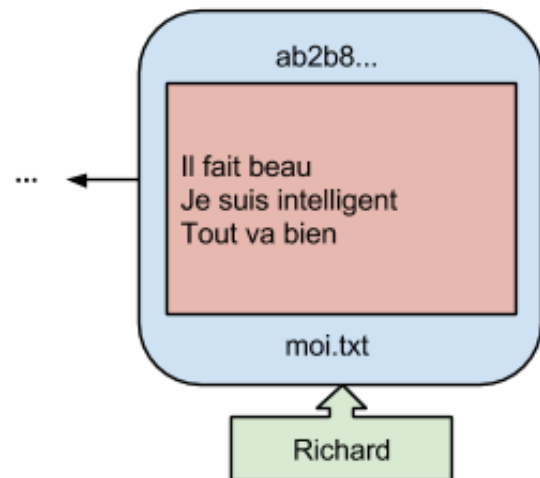
1. Le mainteneur crée le dépôt distant « officiel »
2. Les contributeurs « fork » le dépôt officiel dans des dépôts distants.
3. Tout le monde clone des dépôts distants dans des dépôts locaux.
4. Les contributeurs travaillent et modifient leur dépôt local.
5. Les contributeurs poussent les modifications dans leurs dépôts distants respectifs.
6. Les contributeurs invitent le mainteneur à puller leurs modifications et le mainteneur révisé le code et pousse les modifications dans le dépôt initial.
7. Les contributeurs pull le dépôt initial dans leur dépôt local.
8. Retour à l'étape 4

Git et la Gestion des conflits

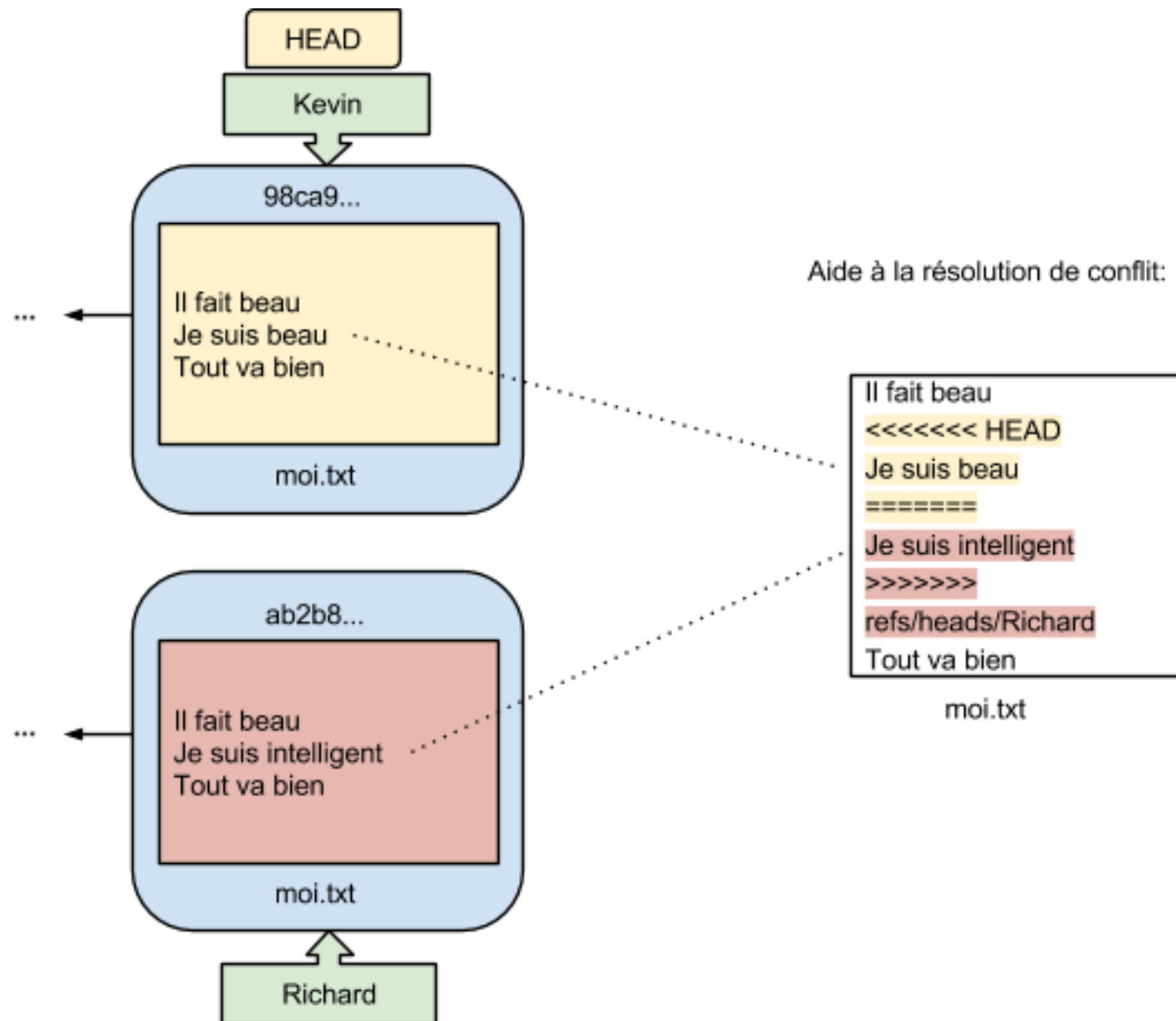
Imaginons deux branches Kevin et Richard



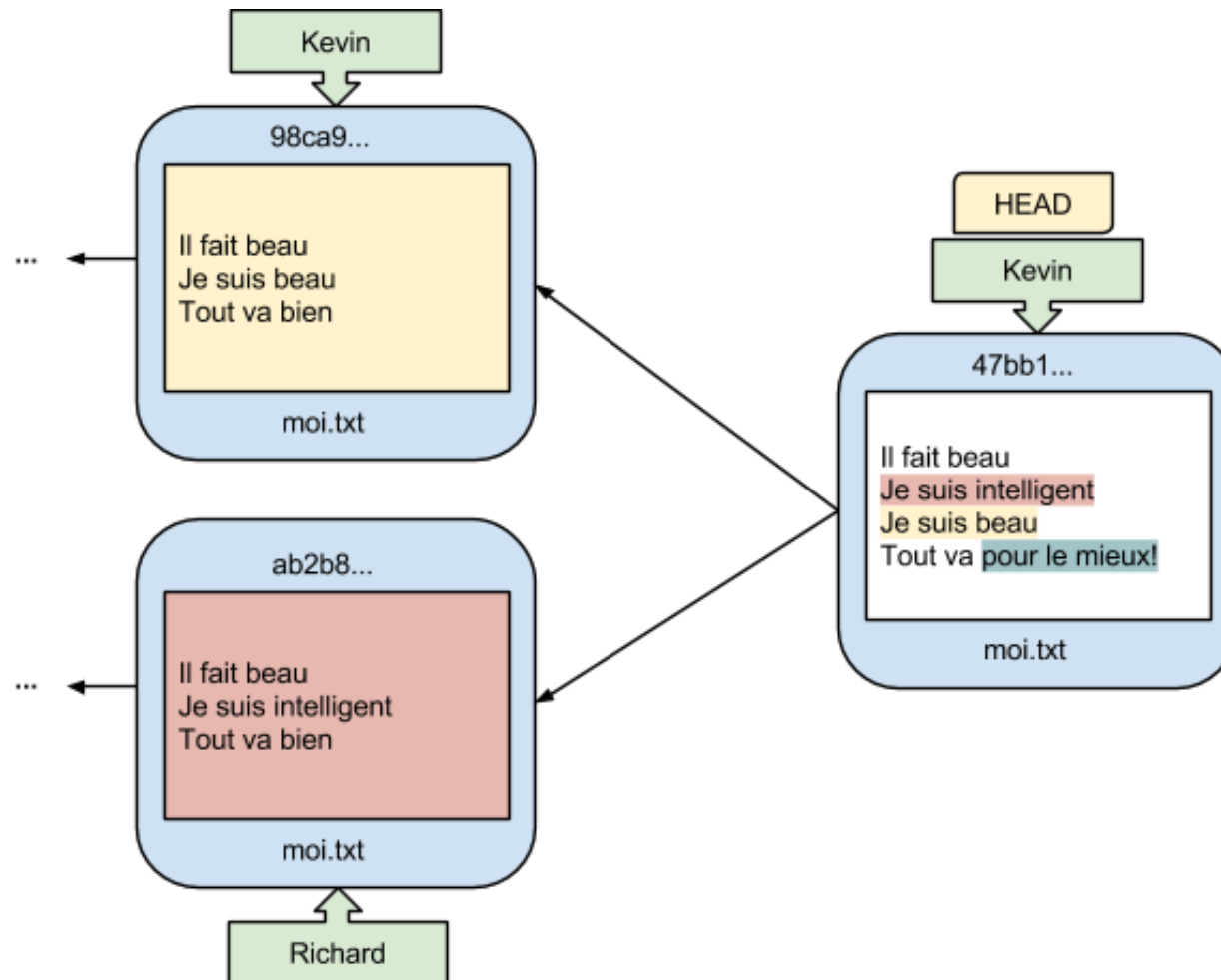
Si on veut fusionner (merge) la branche Richard avec la branche Kevin, nous aurons un conflit.



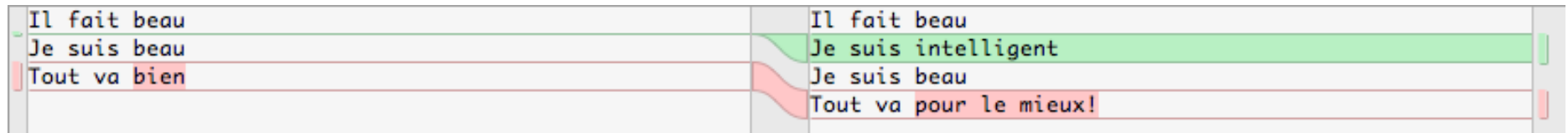
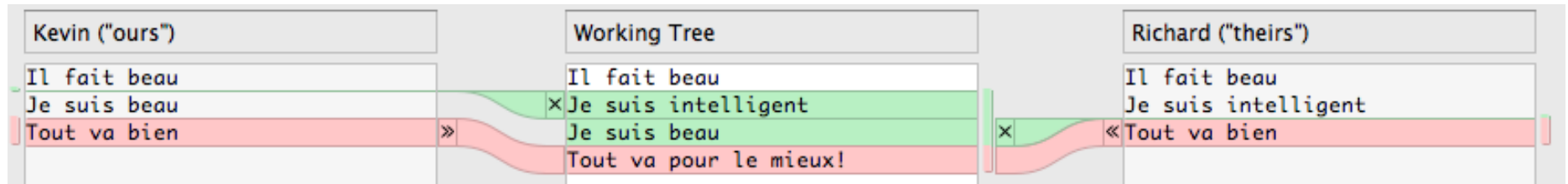
Git et la Gestion des conflits



Git et la Gestion des conflits



Gestion de conflit avec des outils intégrés



Git et les bonnes pratiques

Git et les bonnes pratiques

- Commiter des modifications en relation les unes des autres
-> deux aspects = deux commit
- Tester le code avant de comiter
- Utiliser les branches
- Commiter souvent
- Ecrire de bonne description dans les commit
- Ne pas commiter un travail à moitié fait
- Git n'est pas un système de Sauvegarde/Backup

Documentation

<https://git-scm.com/>