CONDUITE DE PROJET

GIT

Vincent Jousse / @vjousse

Très fortement inspiré de Pro Git

SYSTÈME DE GESTION DE VERSION

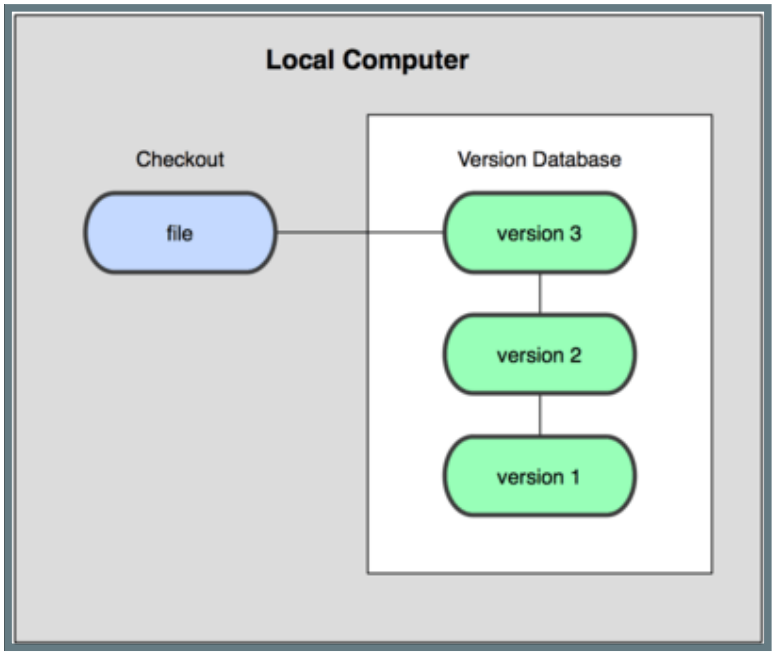
VERSION CONTROL SYSTEM (VCS)

KÉSAKO ?

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

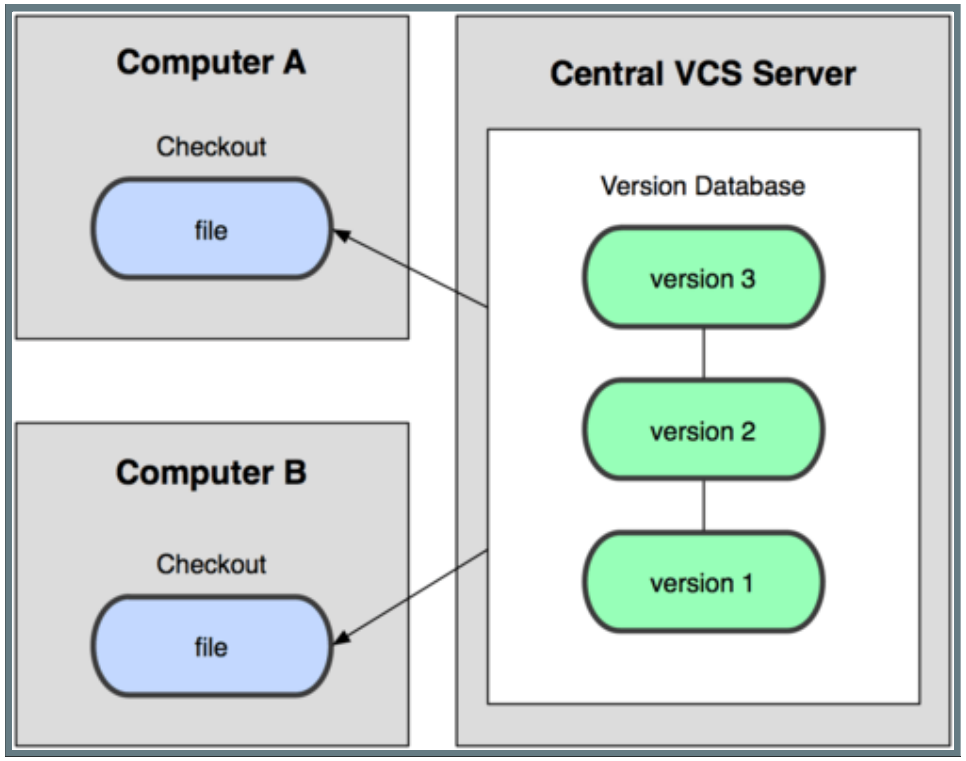
3 TYPES DE SYSTÈME

LES SYSTÈMES LOCAUX



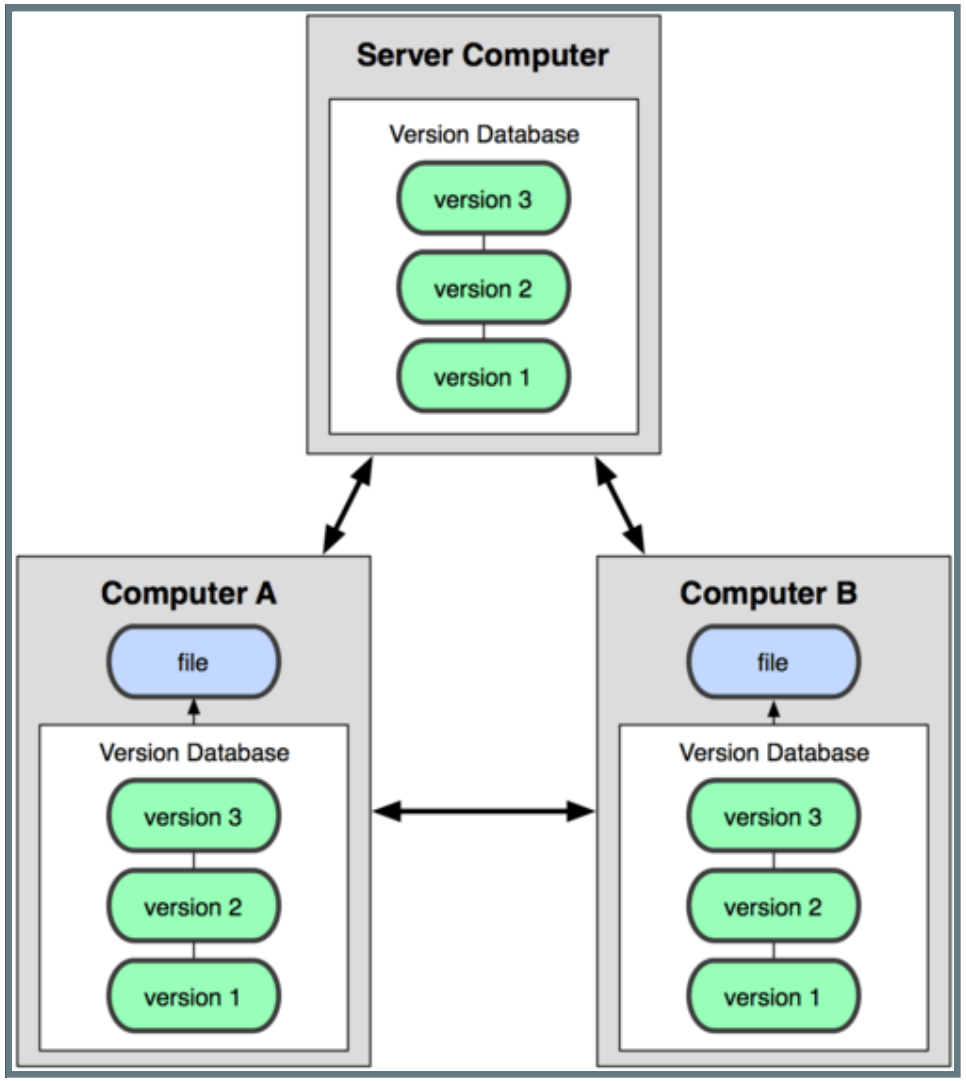
Exemples : RCS, copies locales, …

LES SYSTÈMES CENTRALISÉS (CVCS)



Exemples : CVS, SVN, Perforce, …

LES SYSTÈMES DISTRIBUÉS (DVCS)



Exemples : Git, Mercurial, Bazaar, Darcs, …

HISTORIQUE

- Création en 2005
- Noyau Linux
- Développeur : Linus Torvald

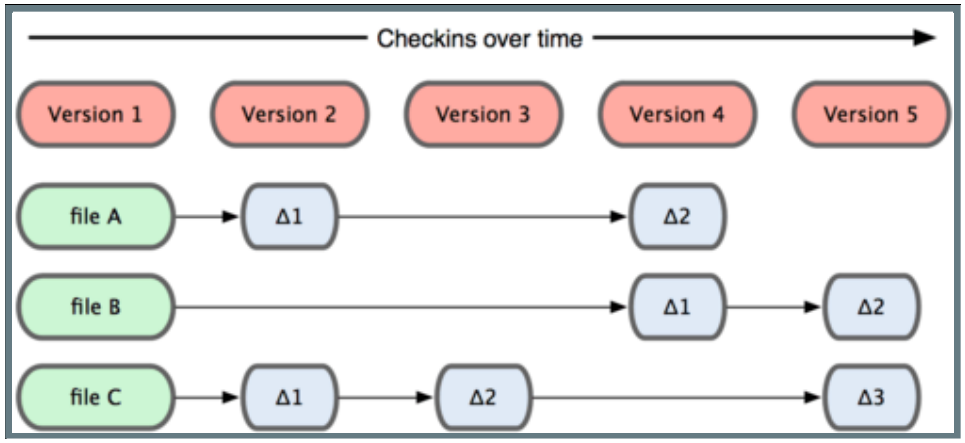
OBJECTIFS

- vitesse
- conception simple
- support pour les développements non linéaires (milliers de branches parallèles)
- complètement distribué
- capacité à gérer efficacement des projets d'envergure tels que le noyau Linux (vitesse et compacité des données)

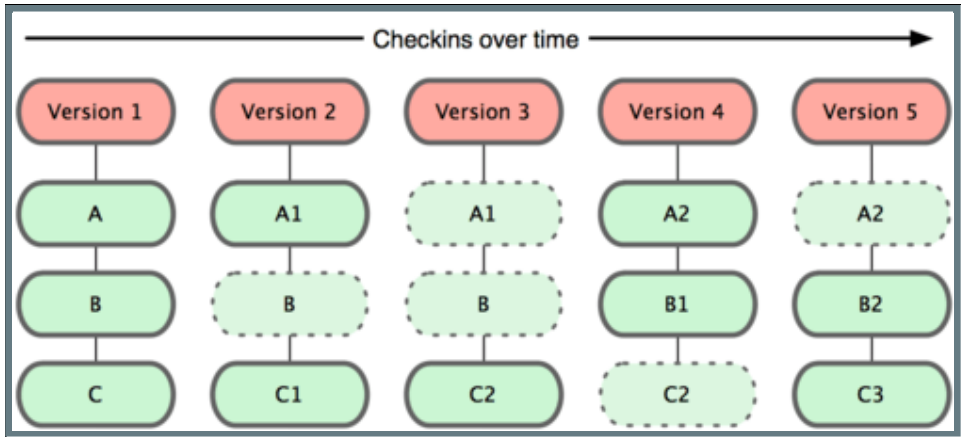
LES BASES DE GIT

Git gère des instantanés et non des différences.

HISTORIQUEMENT



GIT


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

La majorité des opération est locale

- Pas le latence réseau
- Accès immédiats et rapides
- Possibilité de travailler déconnecté

Git gère l'intégrité

Git calcule des sommes de contrôle (SHA-1)

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Vous les verrez partout !

Généralement, Git ne fait qu'ajouter

LES 3 ÉTATS

Git gère trois états pour vos fichiers :

- validé
- modifié
- indexé

VALIDÉ

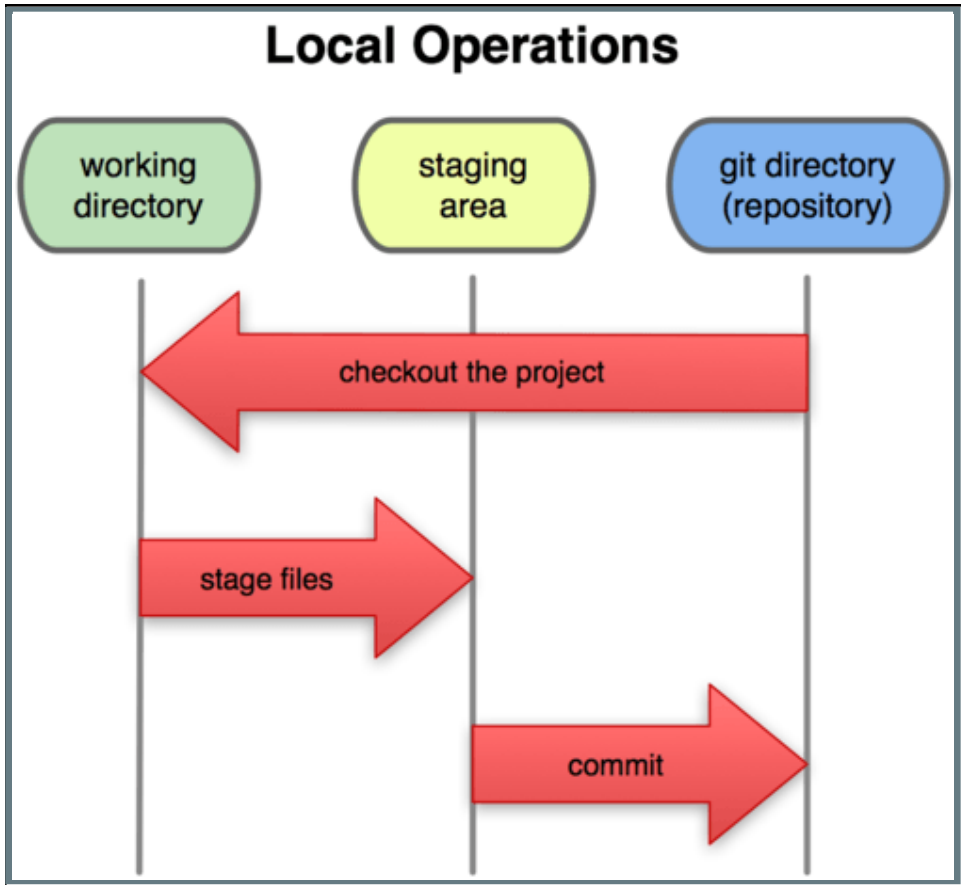
les données sont stockées en sécurité dans votre base de données locale

MODIFIÉ

vous avez modifié le fichier mais qu'il n'a pas encore été validé en base

INDEXÉ

vous avez marqué un fichier modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet



Répertoire de travail, zone d'index et répertoire git

Le répertoire Git est l'endroit où **Git stocke les méta-données et la base de données des objets de votre projet**. C'est **la partie la plus importante de Git**, et c'est ce qui est copié lorsque vous clonez un dépôt depuis un autre ordinateur.

Le répertoire de travail est **une extraction unique d'une version du projet**. Ces fichiers sont extraits depuis la base de données compressée dans le répertoire Git et placés sur le disque pour pouvoir être utilisés ou modifiés.

La zone d'index est un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant **ce qui fera partie du prochain instantané**.

UTILISATION CLASSIQUE

1. vous modifiez des fichiers dans votre répertoire de travail
2. vous indexez les fichiers modifiés, ce qui ajoute des instantanés de ces fichiers dans la zone d'index
3. vous validez, ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la base de données du répertoire Git

UN PEU DE CONCRÊT

INITIALISER UN DÉPÔT GIT

Pour placer un répertoire existant sous Git

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

Création d'un sous répertoire « caché » nommé .git

COMMENCER À SUIVRE SES FICHIERS

```
$ git add *.c
$ git add README
$ git commit –m 'version initiale du projet'
```

CLONER UN DÉPÔT EXISTANT

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

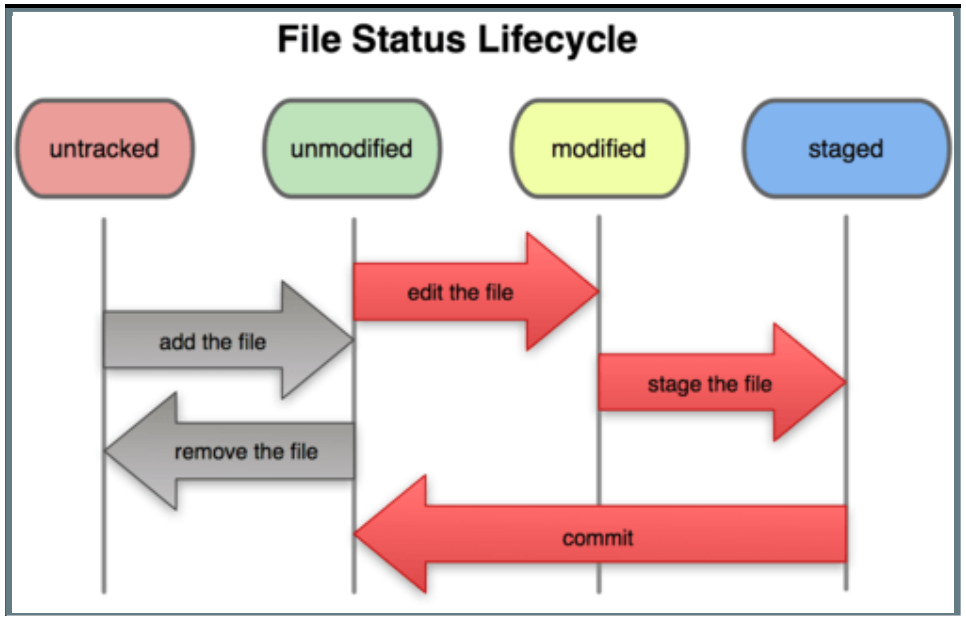
=> création d'un répertoire **conduite-projet**

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

=> création d'un répertoire **autrenom**

LES COMMANDES DE BASE

ENREGISTRER LES MODIFICATIONS



VÉRIFIER L'ÉTAT DES FICHIERS

```
$ git status
On branch master
nothing to commit, working directory clean
```

AJOUT D'UN FICHIER

```
$ vim LISEZMOI
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        LISEZMOI

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

PLACER UN NOUVEAU FICHIER SOUS SUIVI DE VERSION

On ajoute le fichier

```
$ git add LISEZMOI
```

On vérifie le statut

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   LISEZMOI
```

INDEXER DES FICHIERS MODIFIÉS

```
$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

        new file:   LISEZMOI

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:   benchmarks.rb
```

```
$ git add benchmarks.rb
$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

        new file:   LISEZMOI
        modified:   benchmarks.rb
```

```
$ vim benchmarks.rb
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   LISEZMOI
        modified:   benchmarks.rb

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:   benchmarks.rb
```

```
$ git add benchmarks.rb
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   LISEZMOI
        modified:   benchmarks.rb
```

IGNORER DES FICHIERS

Créer un fichier .gitignore

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

VALIDER VOS MODIFICATIONS

```
git commit
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   LISEZMOI
#       modified:   benchmarks.rb
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master]: created 463dc4f: "Fix benchmarks for speed"
 2 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 LISEZMOI
```

SUPPRIMER DES FICHIERS

```
$ rm grit.gemspec
$ git status
# On branch master
#
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#
#       deleted:    grit.gemspec
#
```

```
$ git rm grit.gemspec
rm 'grit.gemspec'
$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    grit.gemspec
#
```

ARRÊTER DE SUIVRE UN FICHIER

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

VISUALISER LES DIFFÉRENCES

Visualiser les modifications non indexées

```
$ git diff
```

```
$ git diff
diff --git a/benchmarks.rb b/benchmarks.rb
index e445e28..86b2f7c 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -127,3 +127,4 @@ end
 main()

 ##pp Grit::GitRuby.cache_client.stats
+# test line
```

Visualiser les différences entre les fichiers indexés et le dernier instantané

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

```
$ git diff --cached
diff --git a/benchmarks.rb b/benchmarks.rb
index 3cb747f..e445e28 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -36,6 +36,10 @@ def main
          @commit.parents[0].parents[0].parents[0]
        end

+       run_code(x, 'commits 1') do
+         git.commits.size
+       end
+
        run_code(x, 'commits 2') do
          log = git.commits('master', 15)
          log.size
```

VISUALISER L'HISTORIQUE

Dépôt d'exemple :

```
$ git clone git://github.com/schacon/simplegit-progit.git
```

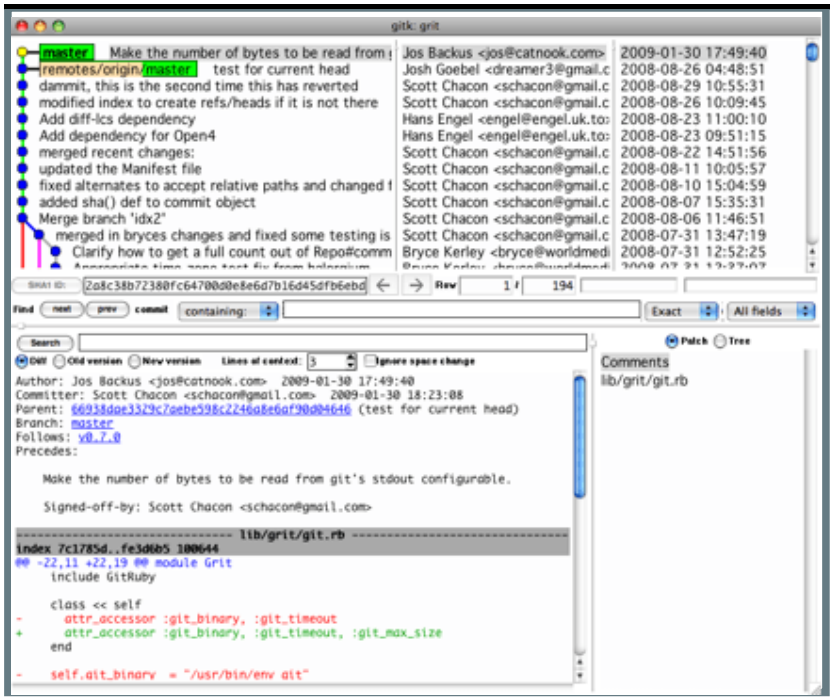
```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,5 +5,5 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
     s.name      =   "simplegit"
-    s.version   =   "0.1.0"
+    s.version   =   "0.1.1"
     s.author    =   "Scott Chacon"
     s.email     =   "schacon@gee-mail.com
```

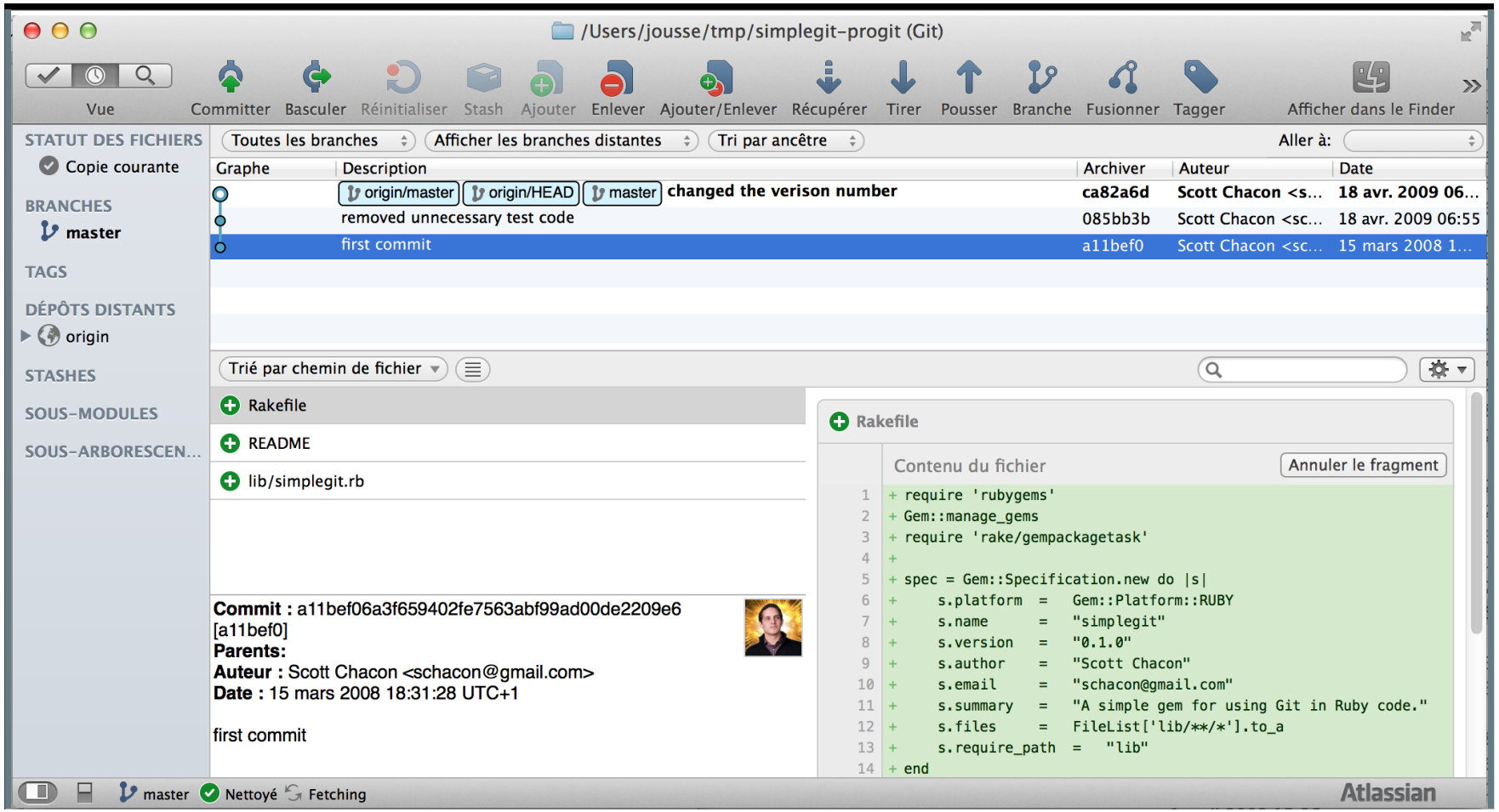
```
$ git log --since=2.weeks
```

INTERFACES GRAPHIQUES

```
$ gitk
```



http://www.git-scm.com/downloads/guis



TRAVAILLER AVEC DES DÉPÔTS DISTANTS

```
$ git clone git://github.com/schacon/ticgit.git
Initialized empty Git repository in /private/tmp/ticgit/.git/
remote: Counting objects: 595, done.
remote: Compressing objects: 100% (269/269), done.
remote: Total 595 (delta 255), reused 589 (delta 253)
Receiving objects: 100% (595/595), 73.31 KiB | 1 KiB/s, done.
Resolving deltas: 100% (255/255), done.
$ cd ticgit
$ git remote
origin
```

```
$ git remote -v
origin  git://github.com/schacon/ticgit.git (fetch)
origin  git://github.com/schacon/ticgit.git (push)
```

AJOUTER DES DÉPÔTS

```
$ git remote
origin
$ git remote add pb git://github.com/paulboone/ticgit.git
$ git remote -v
origin  git://github.com/schacon/ticgit.git
pb  git://github.com/paulboone/ticgit.git
```

RÉCUPÉRER DEPUIS DES DÉPÔTS DISTANTS

Récupération de code distant dans une nouvelle branche

```
$ git fetch pb
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 44 (delta 24), reused 1 (delta 0)
Unpacking objects: 100% (44/44), done.
From git://github.com/paulboone/ticgit
 * [new branch]      master     -> pb/master
 * [new branch]      ticgit     -> pb/ticgit
```

Récupération de code distant et fusion dans la branche courante

À partir du remote **pb**

```
$ git pull pb
```

À partir du remote par défaut (**origin**)

```
$ git pull
```

POUSSER SON TRAVAIL SUR UN DÉPÔT DISTANT

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

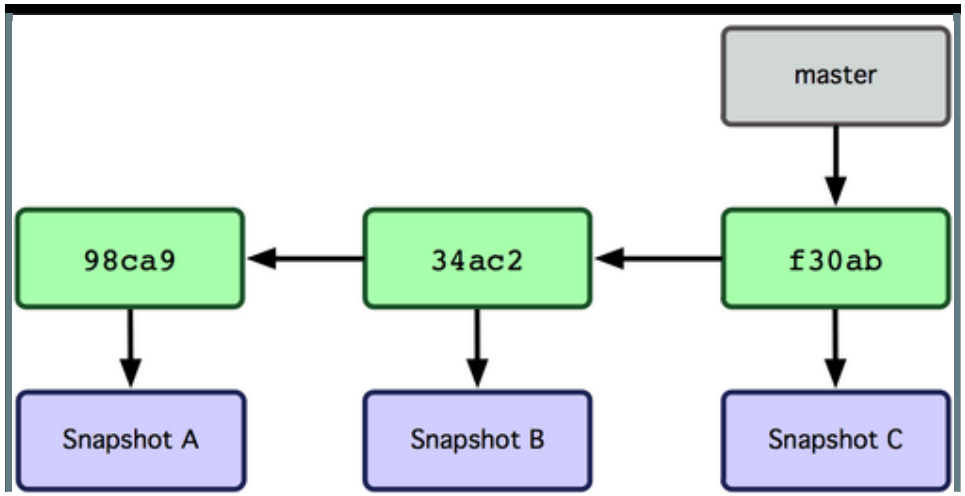
```
$ git push origin master
```

LES BRANCHES AVEC GIT

« Créer une branche signifie **diverger** de la ligne principale de développement et **continuer à travailler sans se préoccuper** de cette ligne principale. »

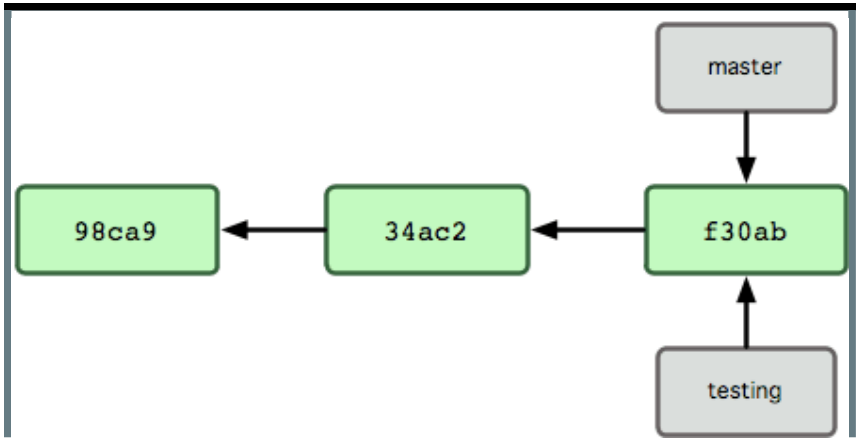
C'est **LA** fonctionnalité de Git

La branche par défaut de Git s'appelle **master**

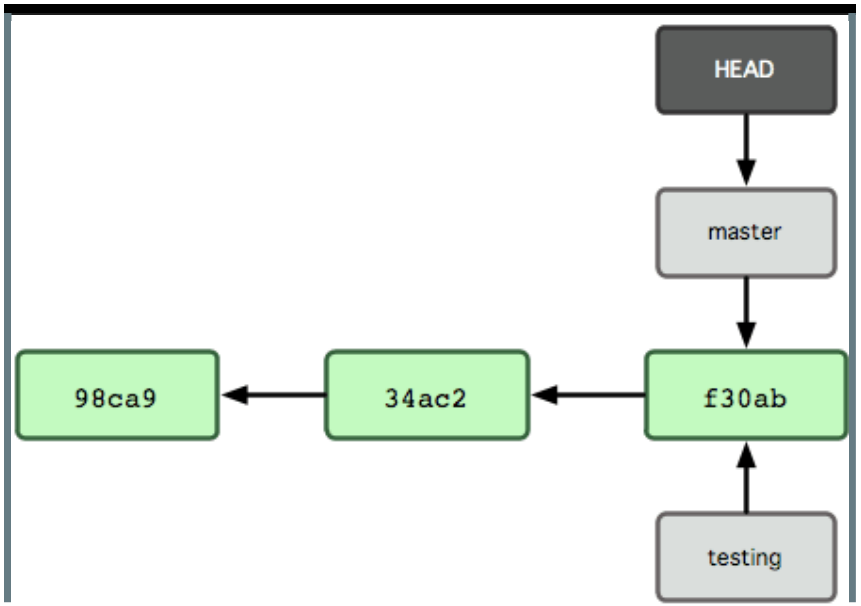


CRÉATION D'UNE NOUVELLE BRANCHE

```
$ git branch testing
```

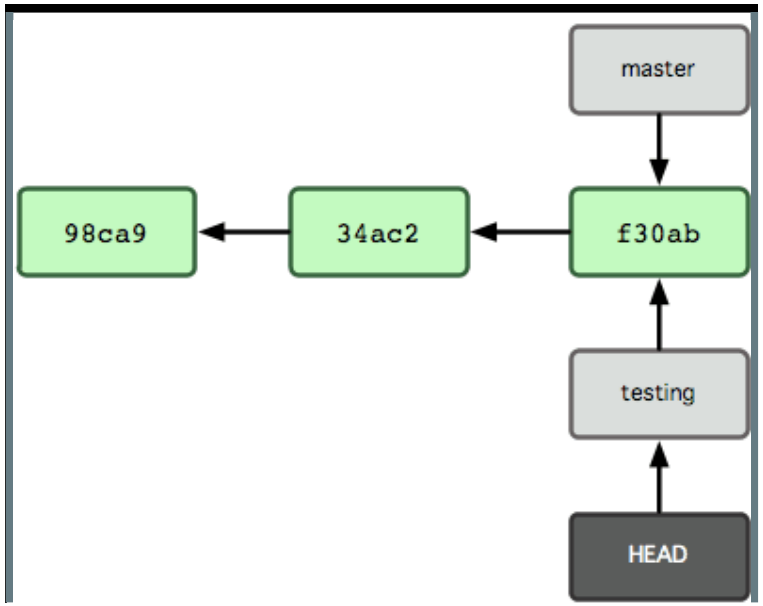


Le pointeur spécial **HEAD** vers la branche courante



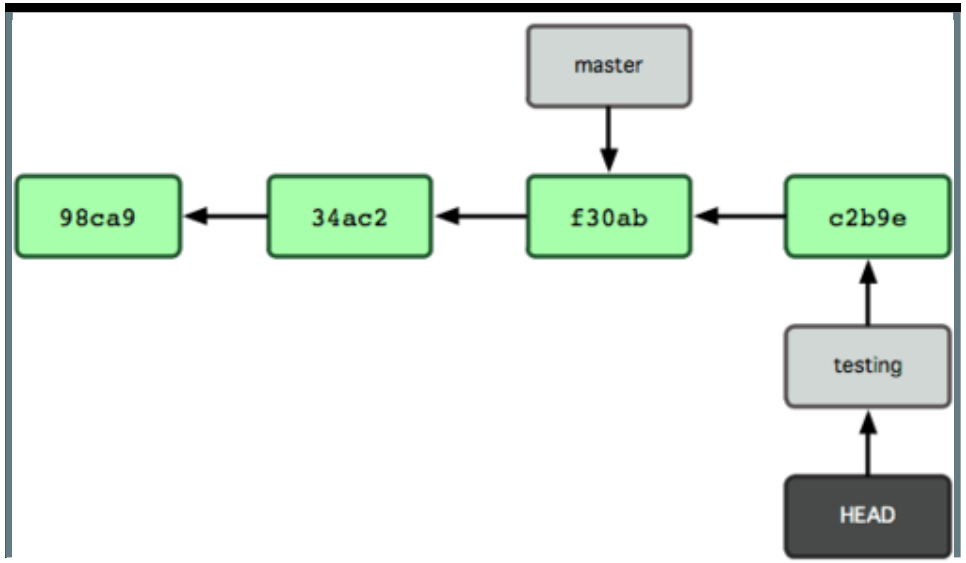
CHANGER DE BRANCHE

```
$ git checkout testing
```



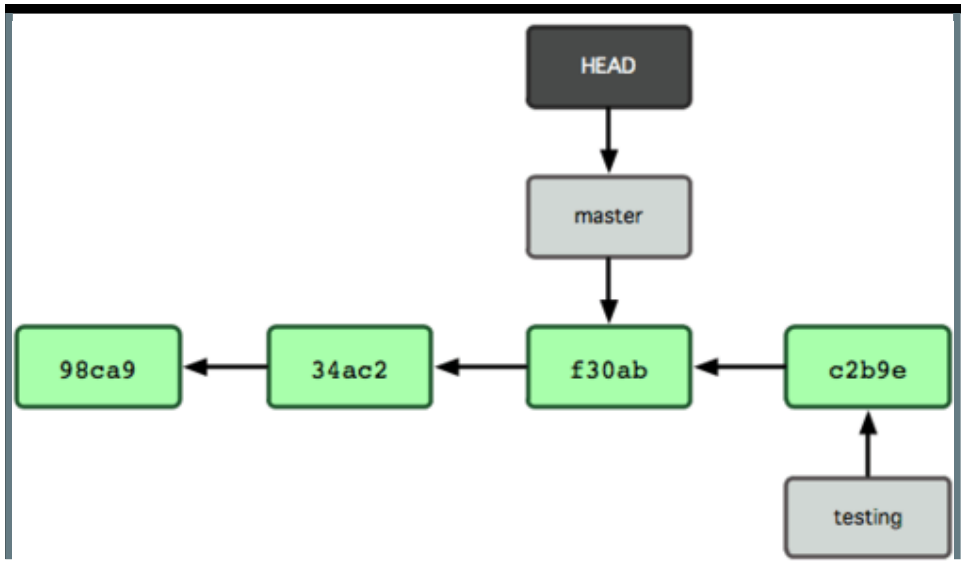
Qu'est-ce qu'il se passe si nous modifions un fichier ?

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



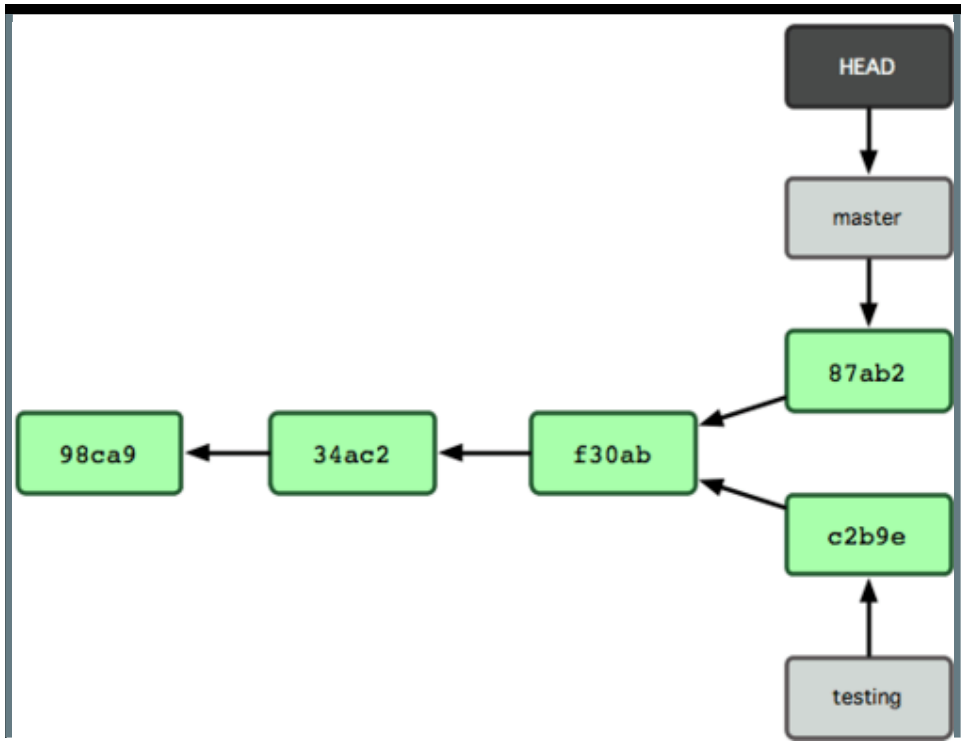
Retournons sur **master**

```
$ git checkout master
```



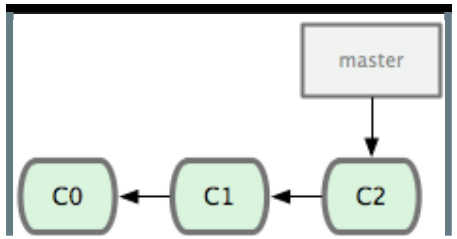
Et si nous modifions de nouveau ?

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



BRANCHER ET FUSIONNER

DÉPÔT GIT DE BASE



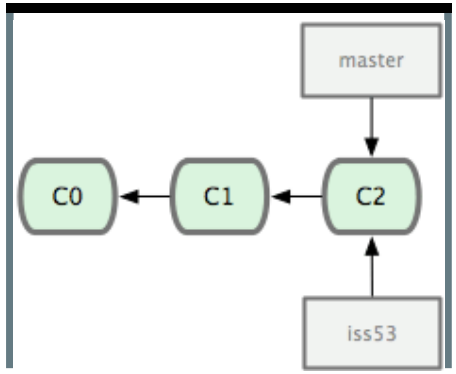
Créons une nouvelle branche **iss53**

Création de la branche **iss53**

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

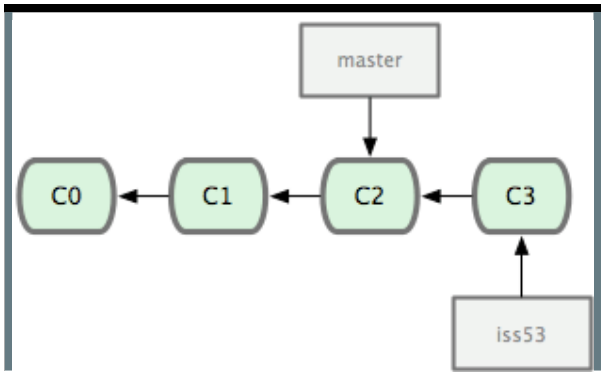
Raccourci pour

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



Faisons quelques modifications dans **iss53**

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



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

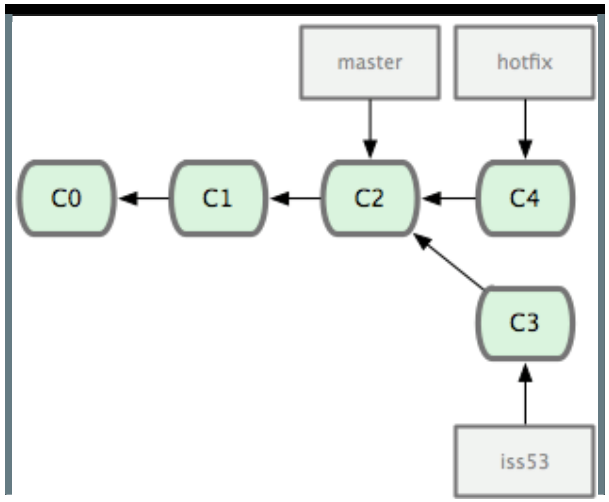
Pas de problème, revenons sur **master**

```
$ git checkout master
Switched to branch "master"
```

et créons une nouvelle branche pour ce correctif (hotfix)

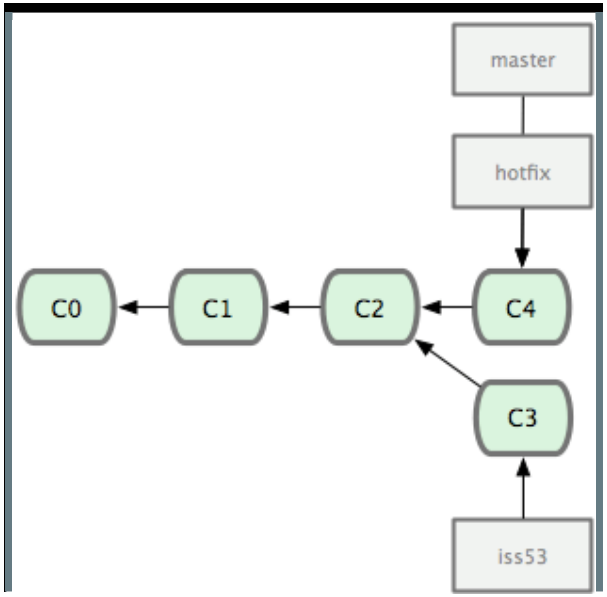
```
$ git checkout -b 'hotfix'
Switched to a new branch "hotfix"
$ vim index.html
$ git commit -a -m "correction d'une adresse mail incorrecte"
[hotfix]: created 3a0874c: "correction d'une adresse mail incorrecte"
 1 files changed, 0 insertions(+), 1 deletions(-)
```

Ce qui nous donne



Le correctif est ok, fusionnons le dans **master**

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast forward
 LISEZMOI |    1 -
  1 files changed, 0 insertions(+), 1 deletions(-)
```

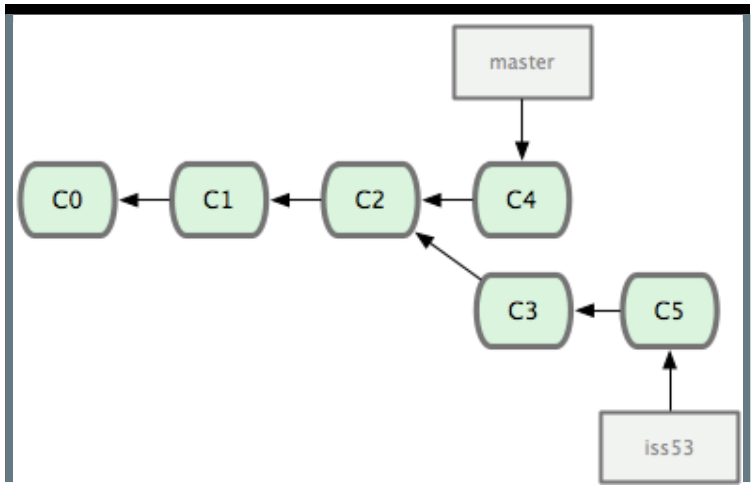


Et faisons un peu de ménage en supprimant la branche **hotfix**

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

Retournons à nos moutons sur **iss53**

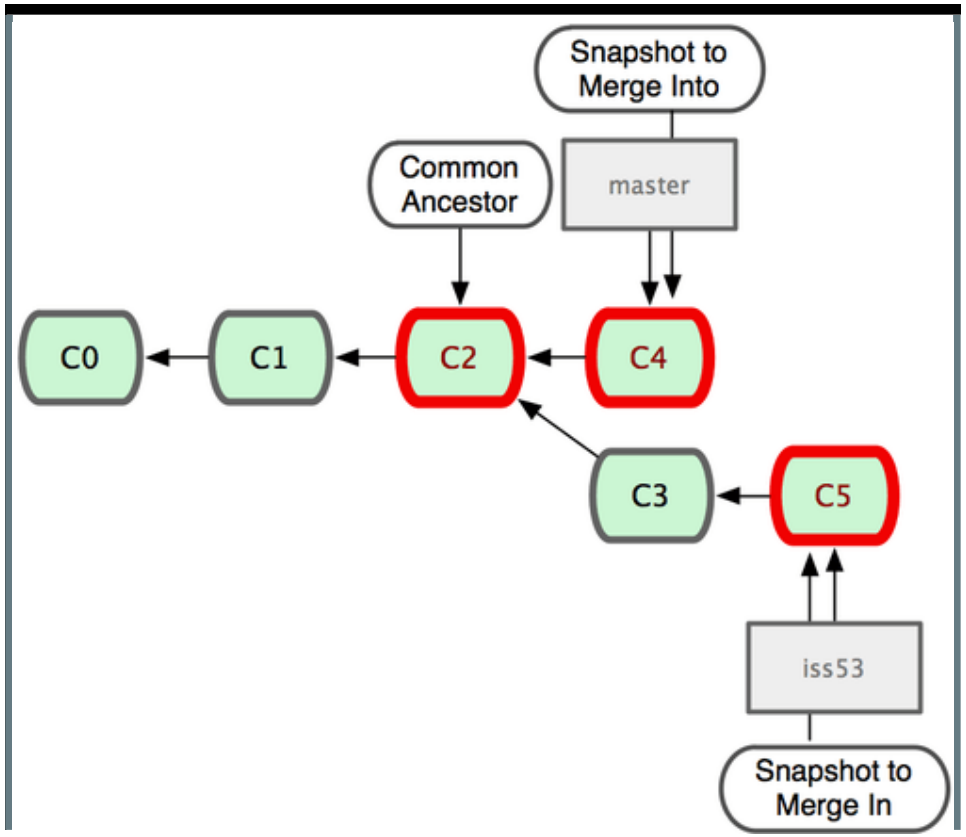
```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'Nouveau pied de page terminé [issue 53]'
[iss53]: created ad82d7a: "Nouveau pied de page terminé [issue 53]"
 1 files changed, 1 insertions(+), 0 deletions(-)
```



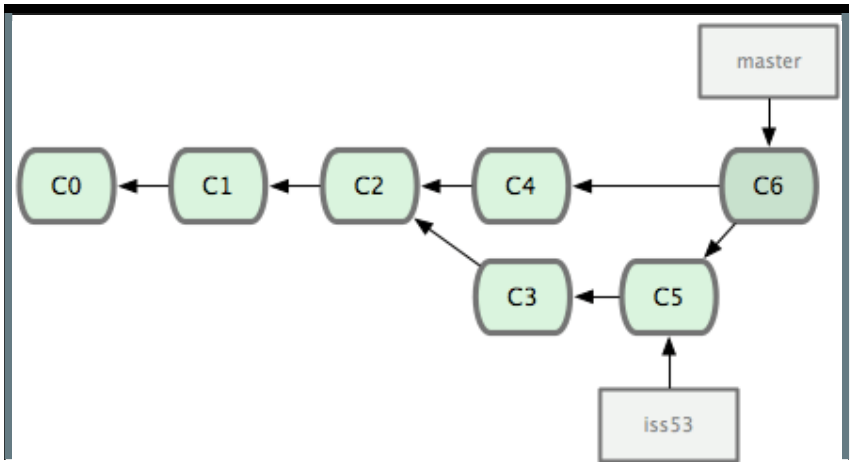
LES BASES DE LA FUSION

Fusionnons **iss53** dans **master**

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


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

Et voilà le résultat



Un peu de ménage

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