

GIT Tous les jours

Cedric Dumoulin

rev 2014

Questions

- Comment faites-vous pour :
 - Faire des sauvegardes de vos projets ?
 - Travailler avec vos co-équipiers ?
 - Pour partager les sources ?
 - Pour fusionner vos modifications ?
- Du vécu :
 - Je n'ai pas la bonne version
 - Je ne sais plus où est la dernière version
 - Nous n'avons pas encore pu fusionner nos modifications
 - ...

La solution

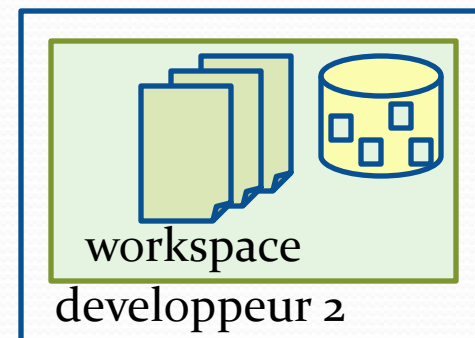
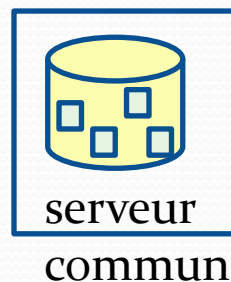
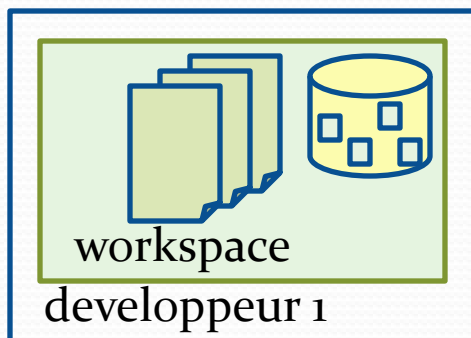
- Utiliser un gestionnaire de version
- Deux types:
 - Centralisé
 - CVS, SVN
 - Décentralisé
 - **GIT**, Mercurial

Qu'est-ce qu'un gestionnaire de version ?

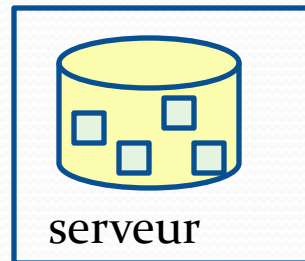
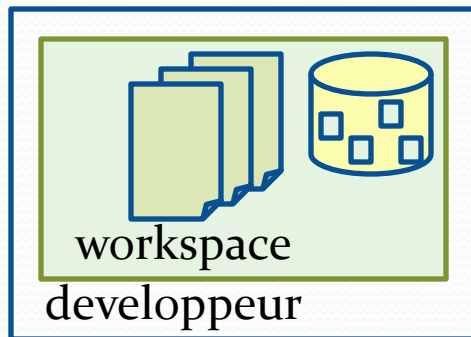
- Un outil permettant
 - De gérer les versions d'un projet
 - De travailler en commun sur des projets
 - De retrouver facilement les anciennes versions
- Un gestionnaire de version stocke des instantanés (versions) d'un projet
 - On peut voir ça comme un ensemble de zip
 - Ce sont les développeurs qui choisissent quand il faut sauvegarder une version

Gestionnaire décentralisé

- **GIT**
- Principe
 - Chaque développeur a une copie locale du dépôt
 - Un développeur travaille sur son dépôt local
- Pas de dépôt central
 - Mais il peut y avoir un dépôt commun pour un projet
 - Il faut de temps en temps synchroniser les dépôts



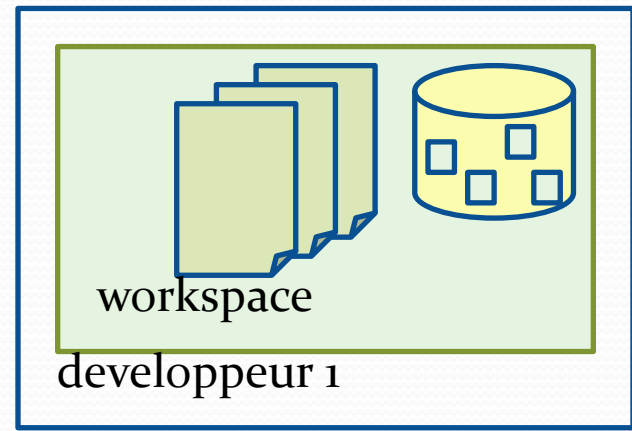
Gestionnaire centralisé



Gestionnaire décentralisé

Travail en local

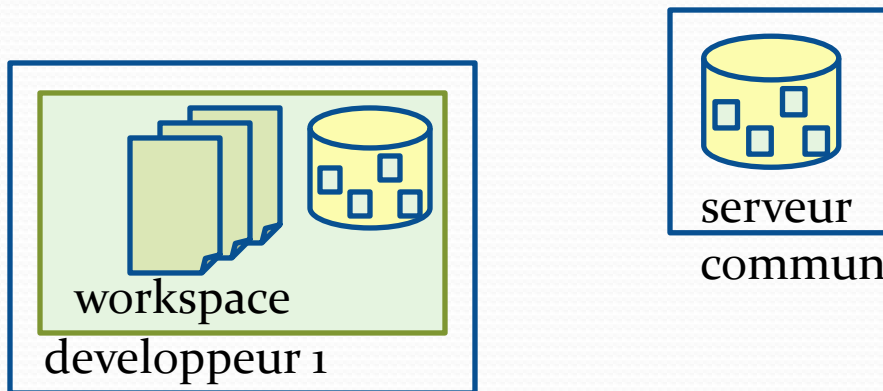
- Travail en local
 - Fait un `checkout` pour avoir une version particulière du projet
 - Travaille sur des branches
 - Faire des `add` pour construire l'ensemble des fichiers modifié devant être « mémorisé »,
 - Puis fait un `commit` pour mémoriser un nouvel instantané
 - La mémorisation est locale



Gestionnaire décentralisé

Synchronisation des dépôts

- Synchronisation des dépôts
 - Fait un clone pour cloner un dépôt distant
 - Fait un 'pull' pour récupérer les versions du serveurs, et les synchroniser avec sa branche de travail locale
 - Fait un 'push' pour pousser les modifications de la branche local vers le serveur distant



Glossaire GIT

Glossaire

- Commit

- Correspond à une version du projet
- C'est un instantané de votre projet.
- Contient tous les fichiers et répertoire de votre projet
- obtenu par `git commit`

- Dépôt

- Contient tous les commits

- Dépôt local

- C'est votre copie locale du dépôt

- Dépôt distant ou 'remote'

- C'est une version distante du dépôt
- Peut être dans un état différent de votre dépôt

Glossaire

- **Workspace**
 - C'est votre répertoire de travail
 - C'est là où vous travaillez, modifiez ou créez des fichiers
- **Staging area** ou **index**
 - C'est une zone contenant les refs des fichiers et répertoire qui seront mis dans le prochain commit
 - Par défaut, contient les refs des fichiers et répertoire du commit courant

Travailler en Local

Créer un nouveau dépôt

- `git init`
 - Crée un dépôt local
- Crée un répertoire contenant :
 - l'espace de travail (workspace - WS)
 - C'est l'ensemble des fichiers sur lesquels vous travaillez
 - Correspond à la version sur laquelle vous travaillez
 - un répertoire '.git'
 - C'est le dépôt local
 - Le nom peut être différent

Versionner son projet

- Aller dans le répertoire à versionner

```
git add .  
git commit -m 'premier commit'
```

Add et Commit

```
git add [nomFichier]  
git commit -m 'premier commit'
```

- `git add [nomFichier]`
 - Ajoute des fichiers dans la 'staged area' (scène) ou index
 - Indique que le fichier devra faire parti du prochain commit
 - Les fichiers dans la 'scène' sont dit 'staged' ou 'indexé'
- `git commit -m 'premier commit'`
 - construit un nouveau commit dans le dépôt local
 - prend les fichiers du commit courant
 - cad ceux dans le dépôt
 - et ceux indiqué dans l 'index
 - qui peuvent remplacer ceux du dépôt

Add et Commit

Tout en un

```
git commit -a -m 'premier commit'
```

- Fait un add + commit

Connaître l'état de l'espace de travail (WS)

- Espace de travail = WorkSpace (WS)
- `git status`
- Indique
 - la branche courante
 - l'état des fichiers
 - des indications sur les commandes possible pour les tâches courantes

Qu'est-ce qui est sauvegardé ?

- Chaque commit fait un instantané du projet complet !
- L'instantané s'appelle un 'commit' ou 'nœud de commit'
 - Il a un nom unique sous forme de hashcode
 - ex: 6226a665caa20a562598cfb2d62b798dd78e1826
 - nom unique dans tous les dépôts (locale, distants)
 - Il référence le nœud le précédent
 - cela forme un graphe de nœud, avec des branches
- Un nœud de commit ne contient que la différence par rapport au commit précédent
 - ➔ prend peu de place sur le serveur

visualiser l'historique

- git log

```
$ git log
commit e5c92c35e20b602c0d52bd3f681ba22a5b212607
Merge: a81ca4b bae9861
Author: Cedric Dumoulin <cedric.dumoulin@lifl.fr>
Date: Thu Nov 21 19:22:01 2013 +0100

    Merge branch 'branch1'

commit a81ca4beacacb05cf7a4680fdf2a9f1979c8f1e8
Author: Cedric Dumoulin <cedric.dumoulin@lifl.fr>
Date: Thu Nov 21 09:06:57 2013 +0100

    master txt4

commit bae986120ea5654b671ebbd18d10eadf582800f4
Author: Cedric Dumoulin <cedric.dumoulin@lifl.fr>
Date: Thu Nov 21 09:04:30 2013 +0100

    br1 - commit 4

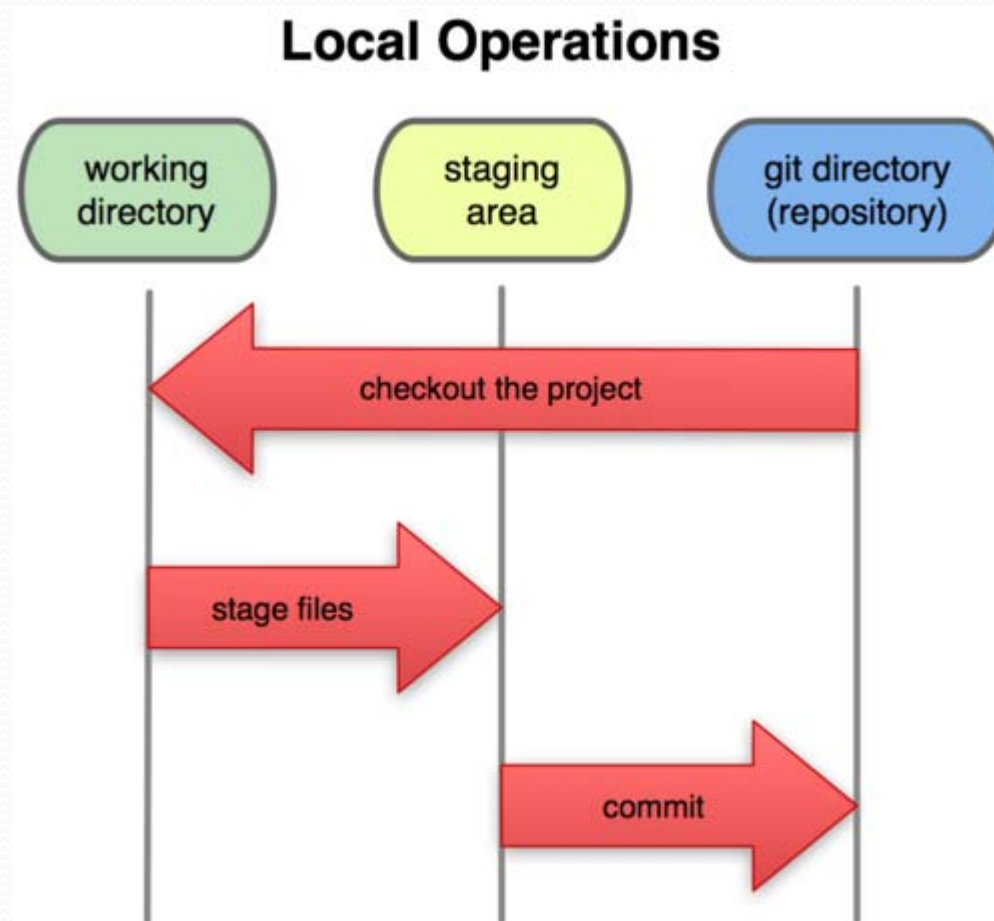
commit 5ab21760a857ba83dc8633e0259813a996e3f3e1
Author: Cedric Dumoulin <cedric.dumoulin@lifl.fr>
Date: Wed Nov 20 23:53:12 2013 +0100

    commit file 3
:
```

Récupérer une version du projet

- `git checkout`
 - récupère la version la plus récente de la branche courante
- `git checkout [hashcode]`
 - récupère la version demandée
- `git checkout [shortHashcode]`
 - permet de mettre les premiers caractères du hash (3 par exemple)

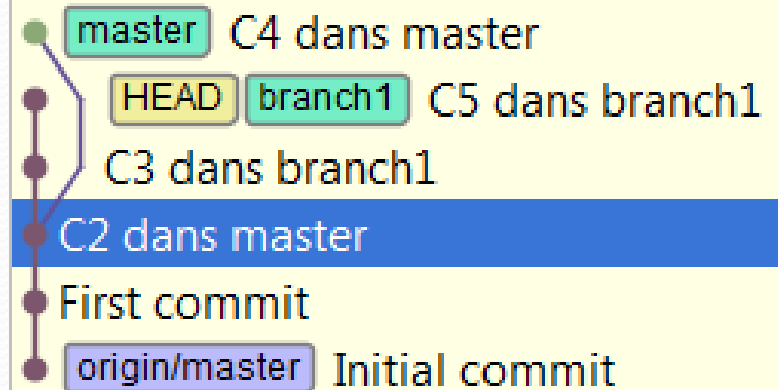
Trois 'répertoires'



Les branches

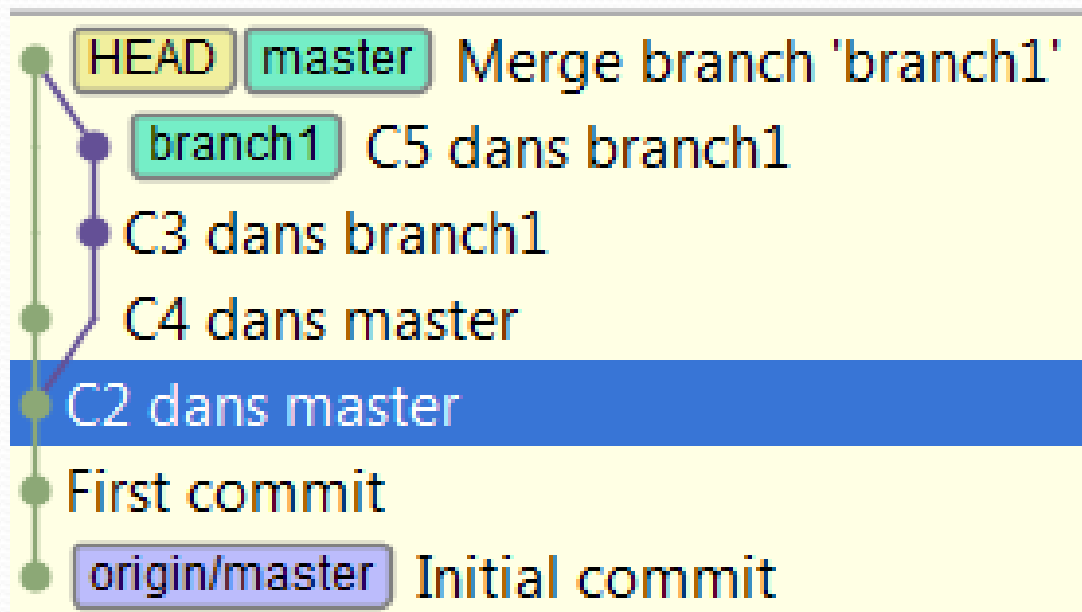
Qu'est-ce qu'une branche

- Une branche est un ensemble de nœuds de commit partant d'un commit 'terminal' jusque la racine.
- Toutes les branches ont un nom
- Le nom référence le nœud en tête de la branche ('head')
- HEAD (majuscules) référence le nœud en tête de la branche courante



Travailler avec les branches

- Des nœuds peuvent être commun à plusieurs branches
- On peut « réunir » 2 branches (**merge**)
 - Cela fusionne les modifications



Créer une branche

```
git branch bug53  
git checkout bug53
```

- `git branch nom_de_branche`
- puis checkout pour basculer sur la branche

Merge de branches

```
git checkout master  
git merge bug53
```

- Faire un checkout de la branche devant recevoir le merge
- `git merge 'branche_a_merger'`
- Un nouveau commit est créé
 - il fusionne les deux branches
 - il contient les modifications des 2 branches
- La branche courante avance sur ce nouveau commit

Deux sortes de merge

- Fast-forward

- consiste a faire avancer l'etiquette de la branche courante au même endroit que l'étiquette de la branche a fusionner
- Est possible uniquement si la branche a fusionner est une branche partant du nœud head de la branche courante (pas de ramification)

- Normal

- les deux branches sont fusionnées dans un nouveau nœud de commit

Régler les conflits de merge

- Si un conflit est détecté, le commit ne se fait pas
 - git arrête le processus de fusion le temps que vous régliez le problème
 - rechercher le fichier
 - `git status` ➔ unmerged
 - régler le conflit
 - modifier le fichier en conflit
 - enlever les chevrons <<<<<<
 - dire que le conflit est réglé
 - `git add <file>`
 - `git commit -m 'un message'`
 - ou dire que l'on abandonne le merge
 - `git checkout -- <file>`

Quand utiliser les branches ?

- Toujours avec GIT !!!
- Ajout d'une fonctionnalité :
 - crée une branche
 - switch vers la branch
 - développe la fonctionnalité
 - commit
 - merge la branche avec la branche master
 - switch vers master
 - merge
 - détruit la branche

```
git branch bug53  
git checkout bug53
```

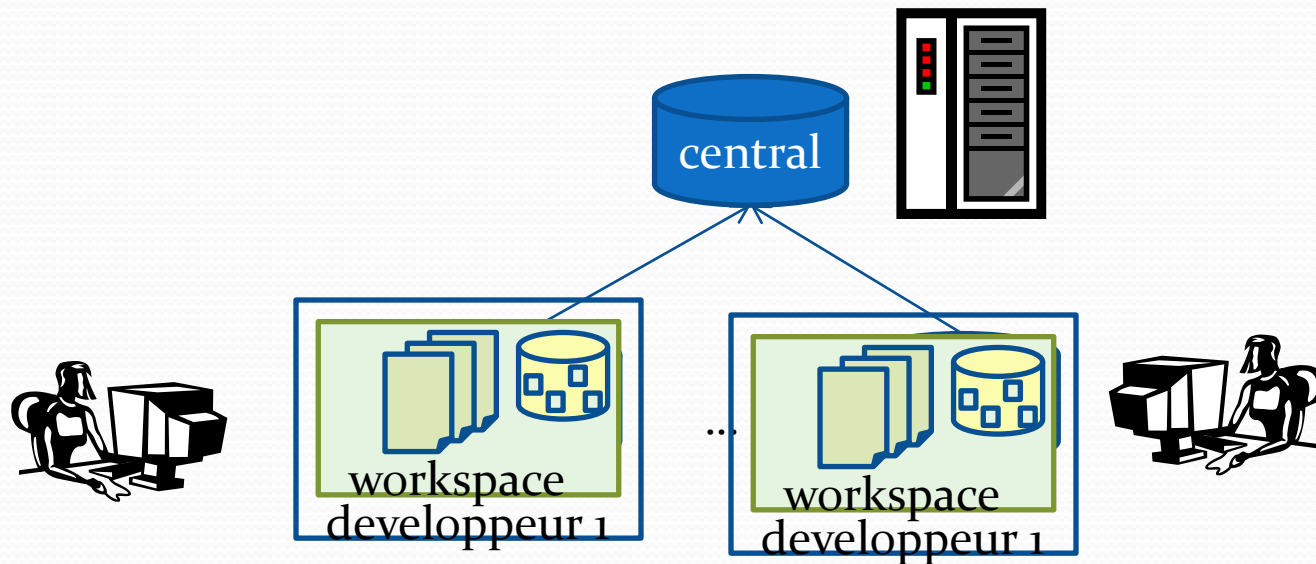
```
git checkout master  
git merge bug53
```

```
git branch -d bug53
```

Travailler à plusieurs

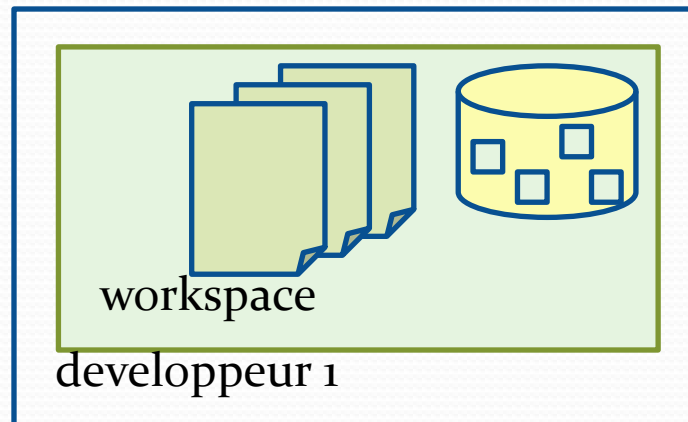
A chacun son dépôt ...

- Chaque participant/développeur à sa copie locale du dépôt
- On peut avoir un dépôt commun
 - approche la plus simple



... et travail en local

- Le développeur travaille sur sa copie locale, dans une branche 'local'
- deux types de branches:
 - **remotes** – représentent les branches dans le dépôt distant
 - **locales** – représentent les branches sur lesquelles le développeur travaille
- on ne **travaille que sur les branches locales**
 - Elles prolongent une branche 'remote'



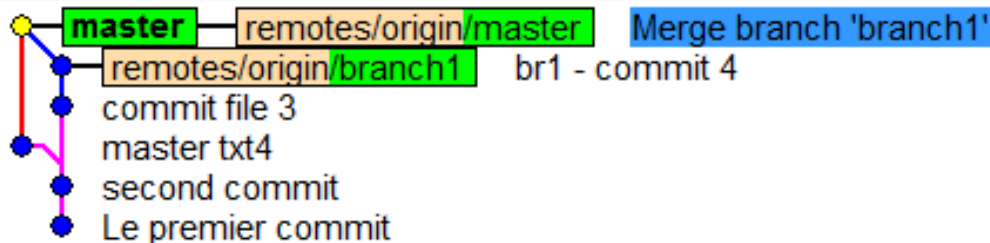
Commencer à travailler sur un projet partagé

- On fait un **clone local du dépôt distant**
 - On obtient les branches distantes
- On **crée une branche locale** à partir de la branche distante sur laquelle on veut travailler
 - On travaille sur cette branche
 - On fait des commits sur cette branche
- Les modifications sont locales au dépôt
 - Comment les propager au dépôt central ?
 - Comment récupérer les modifications enregistrées sur le dépôt central ?

Cloner le répertoire distant

- `git clone [url]`
- On obtient
 - les branches distantes (`remotes/origin/master`)
 - une branche locale (`master`)
 - basé sur son équivalent distante

```
dumoulin@GRIMBERGEN3 /F/temp/testgitLoc
$ git clone git://localhost/.git clonelocal
Cloning into 'clonelocal'...
remote: Counting objects: 23, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 23 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (23/23), done.
Resolving deltas: 100% (5/5), done.
Checking connectivity... done
```



Afficher un dépôt distant

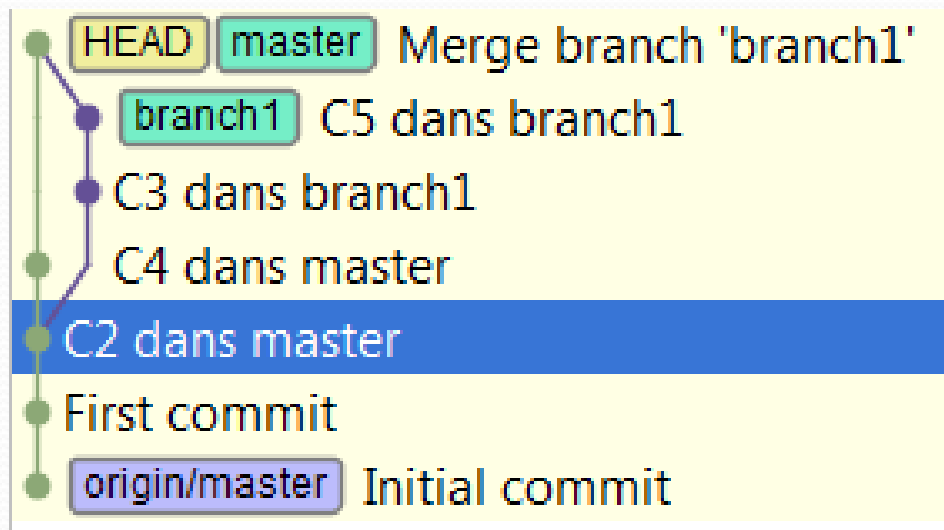
- `git remote`
- `git remote -v` // Affiche aussi les urls

```
dumoulin@GRIMBERGEN3 /F/temp/testgitLoc/clonelocal (master)
$ git remote
origin
```

```
dumoulin@GRIMBERGEN3 /F/temp/testgitLoc/clonelocal (master)
$ git remote -v
origin  git://localhost/.git (fetch)
origin  git://localhost/.git (push)
```

Committer ses modifications

- Commit en local !!
- aller dans le répertoire
- git add .
- git commit -m 'message' [file]



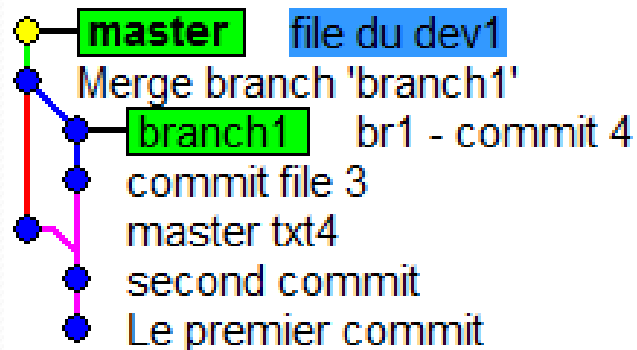
Propager ses modifications sur le dépôt central

- Si il n'y a pas eu de modifications sur le dépôt central:
 - `git status`
 - « your branch is ahead of ... »
- `git push remote master`
 - remote – nom local du dépôt distant
 - master – nom de la branche

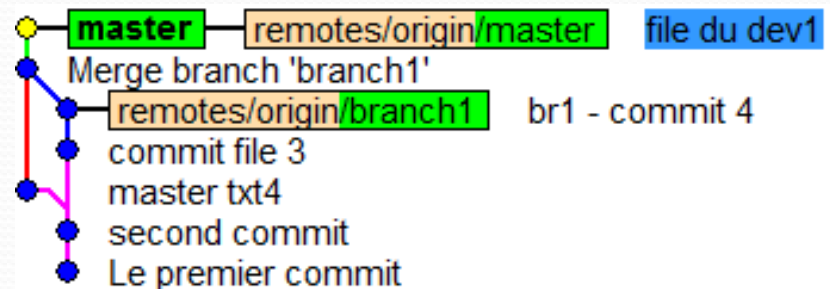
```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

```
$ git push origin master
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3)
```

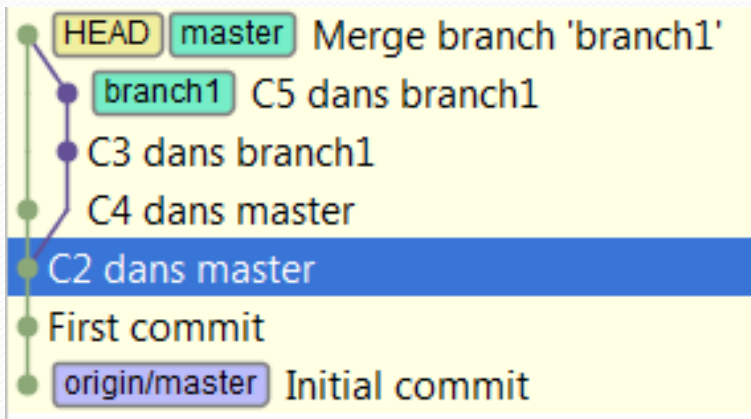
Dépôt origin



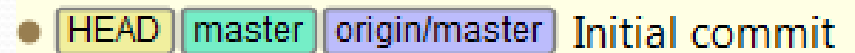
Dépôt local



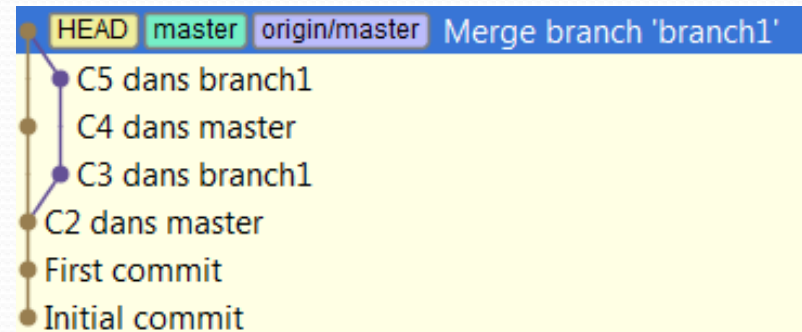
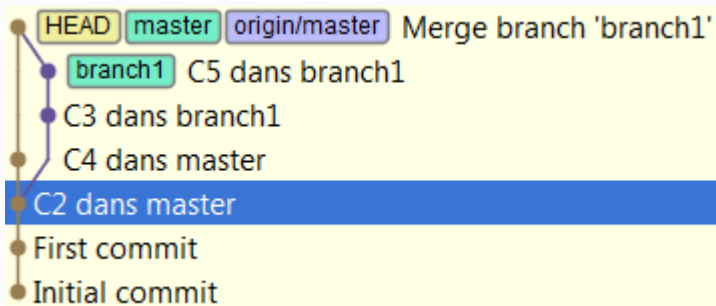
Dépôt local



Dépôt origin



• Après un Push :



Si il y a eu des modifications sur le serveur central ...

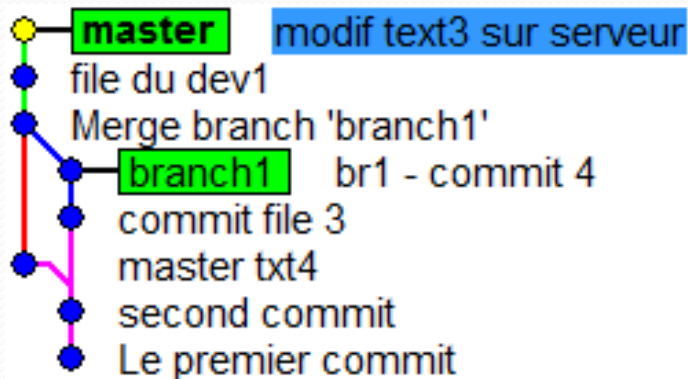
- Il faut rapatrié les modifications sur votre serveur
 - `git fetch`
- puis faire un merge de votre branche local/master avec remote/master
 - `git merge`
 - éventuellement régler les conflits
- puis pousser vos modifications sur le serveur

Si il y a eu des modifications sur le serveur central ...

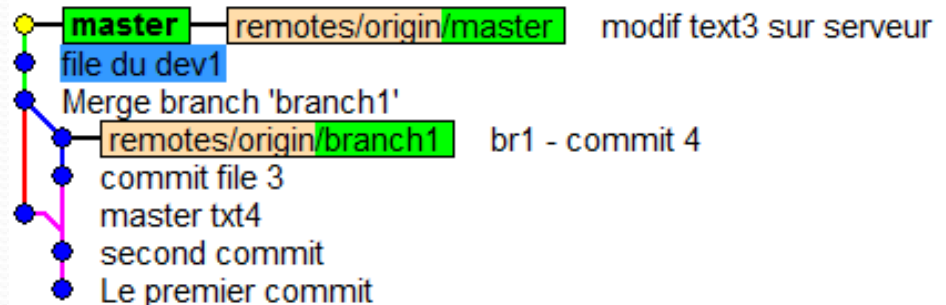
- Tout en un:
 - `git pull`
- S'arrête si il y a des conflits
 - régler les conflits
 - mettre les fichiers modifié dans l'index
 - commit
 - cela continue le processus de pull
- puis pousser vos modifications sur le serveur

Comment récupérer les modifications enregistrées sur le dépôt central ?


- git pull origin



```
$ git pull origin
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From git://localhost/
   5fc6aed..127decb  master      -> origin/master
Updating 5fc6aed..127decb
Fast-forward
 rep1/text3.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```



Git et Eclipse

- 
- GIT est installé par défaut dans Eclipse
 - Interfaces graphiques
 - Perspective GIT
 - utilisez la !
 - Demo !

Bibliographie

- <http://git-scm.com/book/fr>

Atelier

- atelier-git.pdf