Versioning,
avec Git

M1 INFO GL CM

Installation

Installation

http://git-scm.com/downloads *ou*

```
alice$ sudo apt-get install git
```

Important:

```
alice$ git config --global user.name “alice”
alice$ git config --global user.email alice@mail.com
alice$ git config --global core.editor gedit
alice$ git config --global merge.tool kdiff3
alice$ git config --global credential.helper cache
alice$ git config --global push.default simple
alice$ git config --list
alice$ git help config
```

Git?

Système de contrôle de version distribué.

Par Linus Torvalds, depuis 2005.

GNU GPL.

Crée, en local, 1 projet = 2 structures de données:

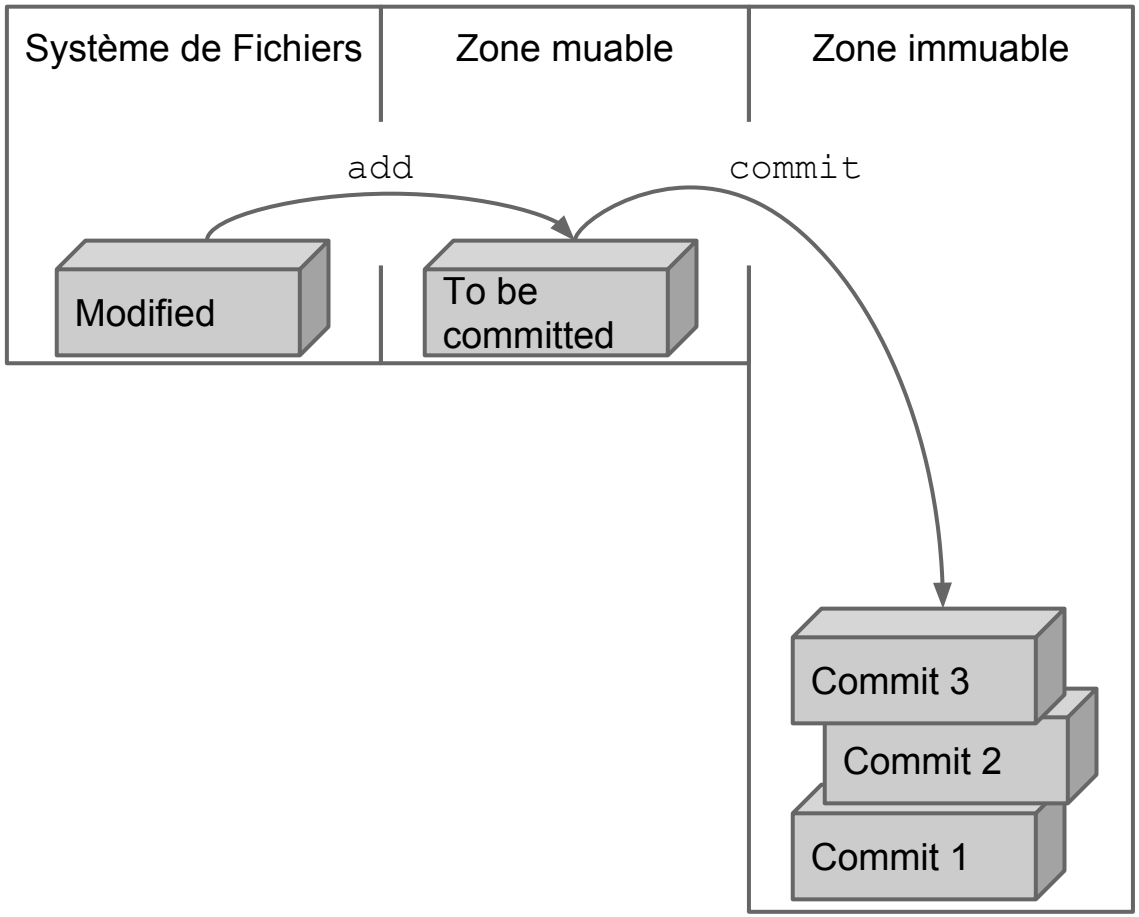
- 1 zone muable (“stage”): pour la préparation des commit = une antichambre;
- 1 zone immuable (“immutable object database”): pour les commits, donc append-only = la cave.

Initialisation : projet

```
alice$ cd myproject
alice$ git init #Crée les 2 zones
alice$ git add .
   #Place les fichiers dans la zone muable “stage”
alice$ git commit
   #Place les fichiers de la zone muable
   #dans la zone immuable.
   #Ouvre un editeur pour le message du commit
   #Ctrl-S, Ctrl-C

ou

alice$ git clone url/file.git
```

Modification du projet

Modification du projet



Modification du projet

```
alice$ git status
       #untracked, tracked-unmodified/modified/staged

alice$ touch test.txt #untracked
alice$ git add test.txt #tracked-staged
alice$ git commit #tracked-unmodified

alice$ gedit test.txt #tracked-modified
alice$ git add test.txt #tracked-staged
alice$ git commit #tracked-unmodified
```

Modification du projet : staging

```
alice$ git status
      #untracked, tracked-unmodified/modified/staged

alice$ gedit test.txt #v1:tracked-modified
alice$ git add test.txt #v1:tracked-staged
alice$ gedit test.txt #careful, v2:tracked-modified!

alice$ git diff #modified vs staged (v2-v1)
alice$ git diff --staged #staged vs committed (v1-v0)
alice$ git diff HEAD # modified vs commited (v2-v0)

alice$ git commit
   #v1:tracked-unmodified but v2:tracked-modified!
```

Modification du projet: no staging

```
alice$ git commit -a
    #Add et commit les tracked
```

Modification du projet: no staging

```
alice$ touch test2.txt
alice$ git commit -a
    #Add et commit les tracked
    #Attention: test2.txt n'est pas pris en compte!
alice$ git commit test2.txt
    #Add et commit test2.txt seulement

alice$ cat .gitignore
        *.[oa]
        *~
    #This files will remain untracked, not even listed
```

(Re)moving files from git

```
alice$ git reset HEAD test.txt
   #unstaged mais tracked
alice$ git rm --cached test.txt
   #untracked
alice$ rm test.txt
   #effacé du répertoire mais pas du projet
alice$ git rm test.txt # effacé et staged
   #committing it will also untrack it

alice$ git mv test.txt test2.txt
   #moved, untracked test.txt, staged test2.txt
   #conserve l'histoire du déplacement
```

Backup

```
alice$ git log #voir les commits
               -p #avec les diffs
               -3 #les trois derniers
               -...
alice$ gitk #version graphique

alice$ git checkout -- test.txt
   #restaure la version du dernier commit
   #depuis la zone muable
```

* * *

Dépôts lointains

Dépôts lointains

Standard:

http://git-scm.com/book/en/Git-on-the-Server-Hosted-Git,
https://github.com/

Créer un compte sur https://education.github.com/.

Faire les étapes 1 & 2 du GitHub bootcamp:

1. Set up Git
2. Create a repository

Copier l'URL de clonage https.

Dépôts lointains

```
alice$ git clone url/file.git
alice$ git remote
       origin

alice$ git remote add bob urlbob/filebob.git
alice$ git remote
       origin
       bob

alice$ git remote rm bob
```

Dépôts lointains : retrieving

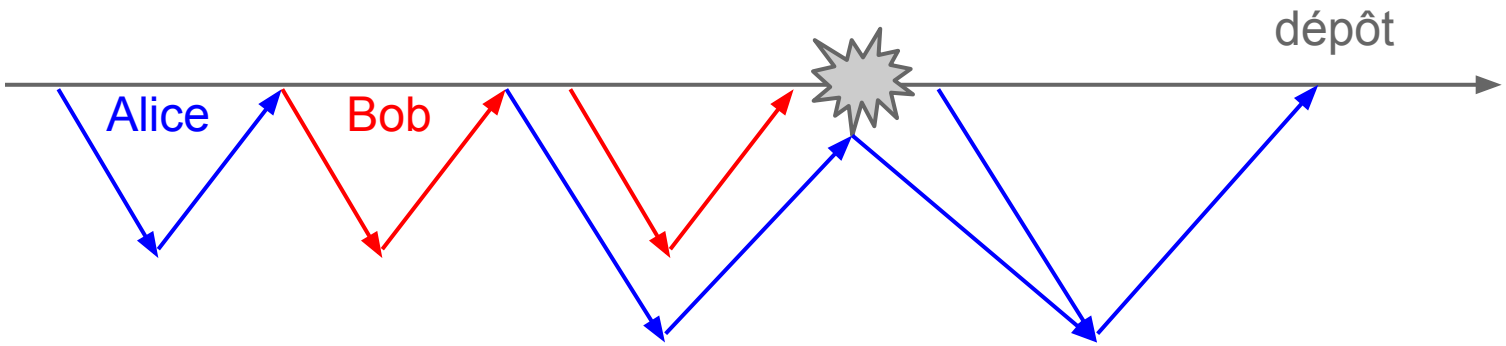
```
alice$ git fetch bob
    #actualise ma copie du dépôt lointain (de bob)

alice$ git merge bob
    #merge la copie avec le projet

alice$ git pull bob
    #pull = fetch + merge
```

Dépôts lointains : pushing

```
alice$ git push bob
    #pousse le projet sur le dépôt (de Bob)
```

Ne marche si personne n’a push avant mon dernier pull.

Sinon, je dois pull, puis push à nouveau.



Pull : conflict

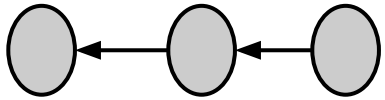
```
alice$ git pull
       CONFLICT (content): Merge conflict in test.txt
alice$ git status
       Unmerged paths: both modified : text.txt
alice$ gedit test.txt
alice$ git status
       modified : text.txt
alice$ git add test.txt
alice$ git commit
```

```
<<<<<<< HEAD
testA
=======
testB
>>>>>>> origin
```

Branches & merges

Zone immuable

Elle est constituée de commits.

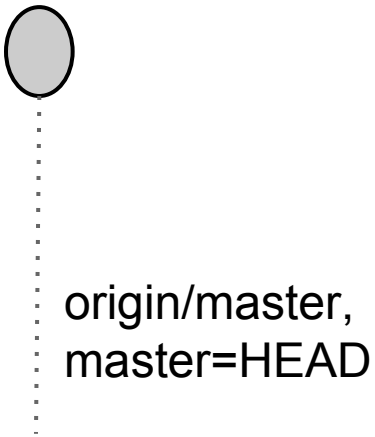


Chaque commit est constitué d'un snapshot du projet, et de pointeurs vers les commits parents.

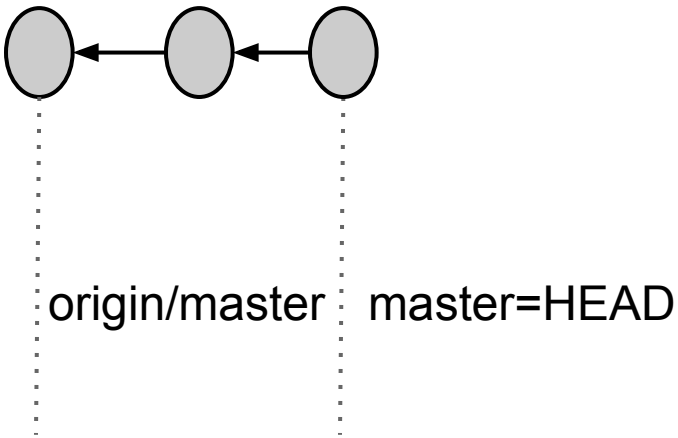
On s'y promène avec des pointeurs:

- `master` : branche locale principale
- `HEAD` : branche locale courante
- `origin/master` : branche principale de ma copie du dépôt d'origine
- `bob/alt` : branche `alt` de ma copie du dépôt bob

Branches



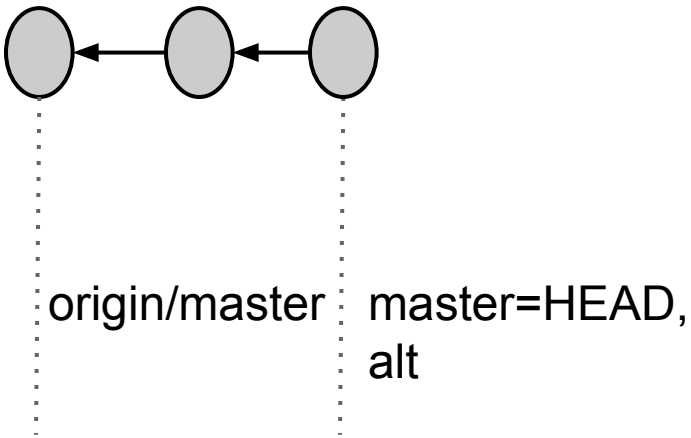
Branches



Branches

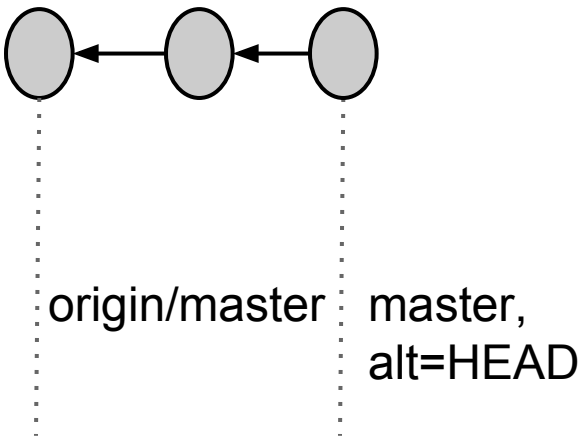
```
alice$ git branch alt
    # creates a pointer “alt” on “master’s” last commit
```

Branches



Branches

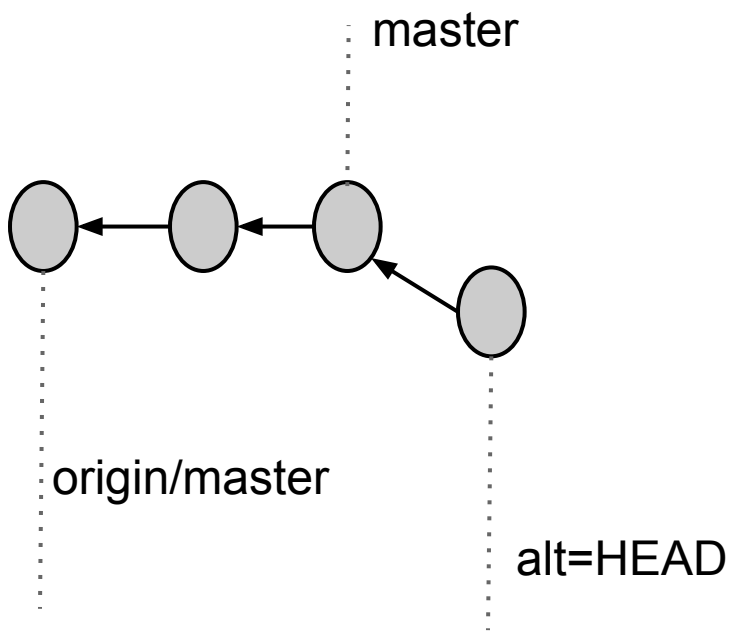
```
alice$ git checkout alt
    # moves the pointer “HEAD” on “alt”
```



Branches

```
alice$ git commit
    # commit mais dans la branche alt
```

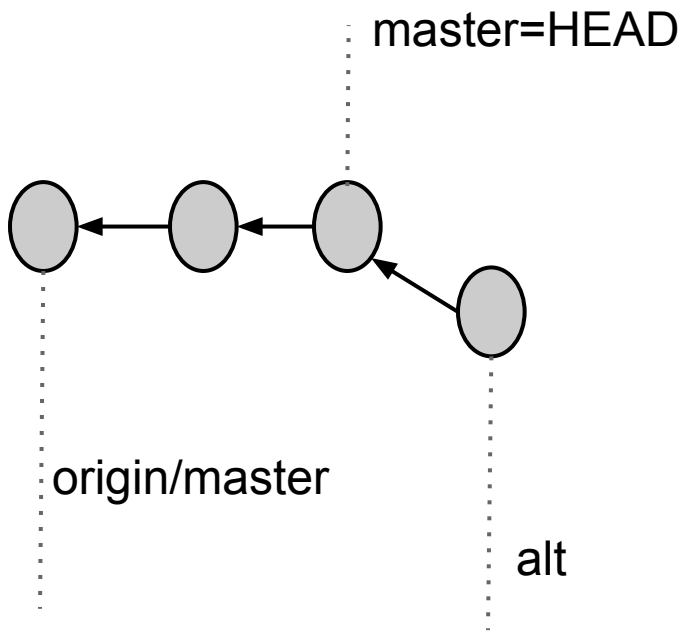
Branches



Branches

```
alice$ git checkout master
    # remet dans le répertoire courant tous les
    # fichiers de master, et
    # met le pointeur “HEAD” sur “master”
```

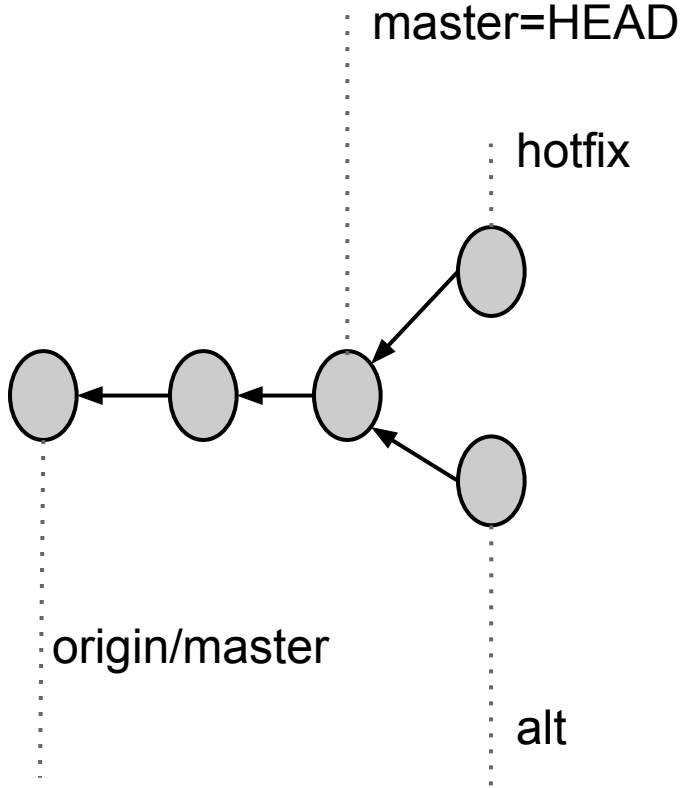
Branches



Merge : fast-forward

```
alice$ git checkout master
alice$ git branch hotfix
alice$ git checkout hotfix
(...)
alice$ git commit
alice$ git checkout master
```

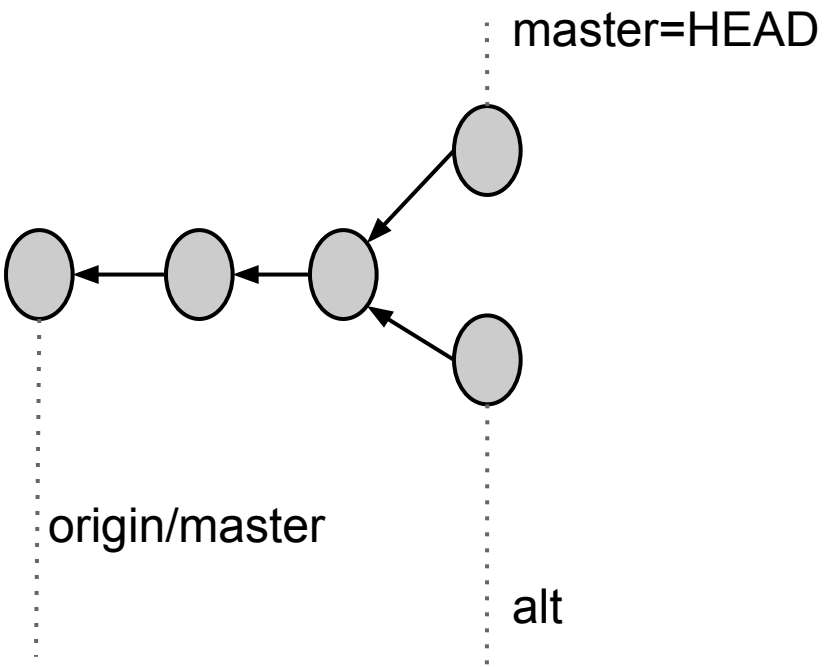
Merge : fast-forward



Merge : fast-forward

```
alice$ git merge hotfix
    #fast-forward merge:
    #“master” pointe vers la cible de “hotfix”
alice$ git branch -d hotfix
    #efface le pointeur hotfix
```

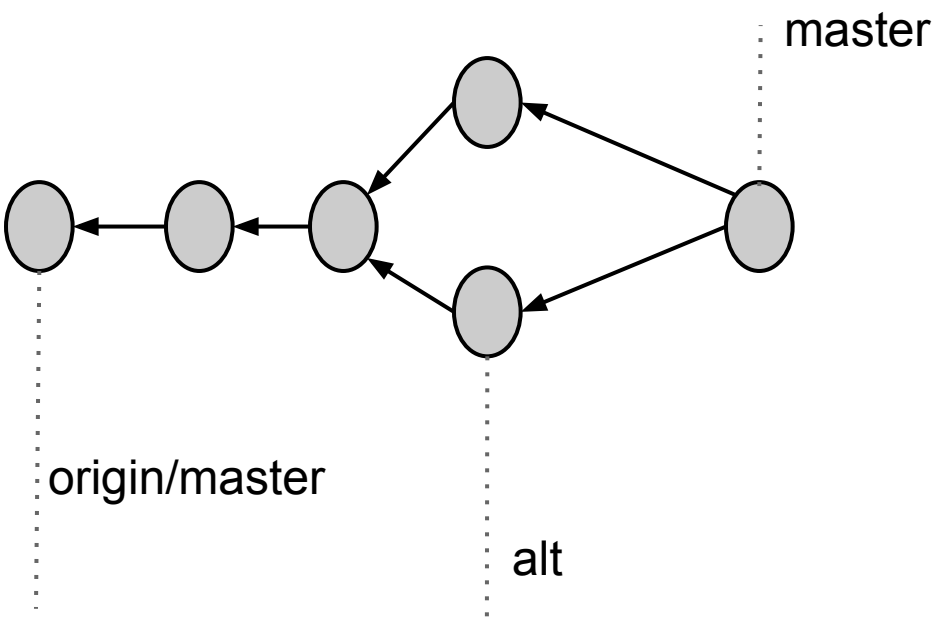
Merge : fast-forward



Merge : three-way

```
alice$ git merge alt
    # three-way merge: crée un “merge commit”
    # dont les parents sont “alt” et l’ancien “master”
    # maintenant “master” pointe sur ce commit,
    # mais pas “alt”
```

Merge : three-way



Merge : conflict

```
alice$ git merge alt
      CONFLICT (content): Merge conflict in test.txt
alice$ git status
      Unmerged paths: both modified : text.txt
alice$ gedit test.txt
alice$ git status
      modified : text.txt
alice$ git add test.txt
alice$ git commit
```

```
<<<<<<< HEAD
testA
=======
testB
>>>>>>> alt
```

Branches

```
alice$ git branch #list all pointers

alice$ git branch --merged #only those merged with HEAD
     alt
   * master
alice$ git branch -d alt
```

Branches lointaines

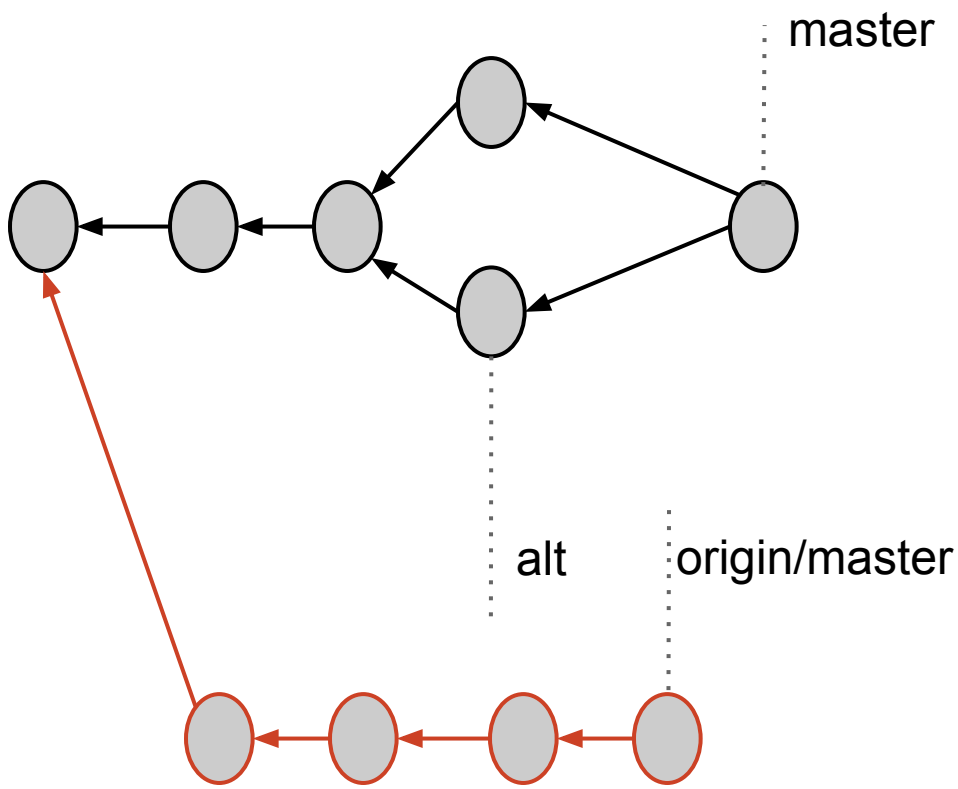
Après clone, `origin` et `master` pointent le même commit. Mais:

- après un commit local, `master` aura avancé.
- après le push d'un tiers sur le dépôt `origin`, le vrai `origin` peut avoir avancé aussi. Pour le savoir:

```
alice$ git fetch origin
   #mettre à jour sa sauvegarde du dépôt lointain
```

En général: `remote/branch` pointe sur ma copie de où pointais `branch` dans le dépôt `remote` lors du dernier fetch.

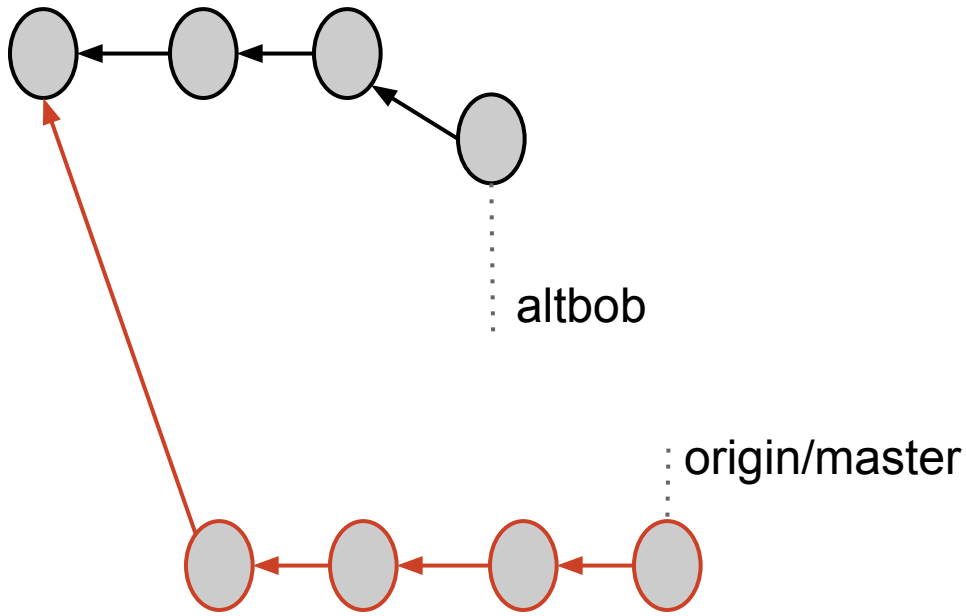
Branches lointaines



Branches lointaines

```
alice$ git push origin alt:altalice
    #pousse la branche “alt” sur le dépôt d’origine,
    #dans une branche baptisée “altalice”.

bob$ git branch altbob origin/altalice
    #crée une branche baptisée “altbob” basée sur
    #la branche “altalice” du dépôt d’origine,
bob$ git checkout altbob
```

Branches lointaines



EGit

EGit

Eclipse > window > Preferences > *Check ssh and git configurations*

Eclipse > File > Import > Git > Local > Add testalicerepo > testalicerepo

Eclipse > Edit test

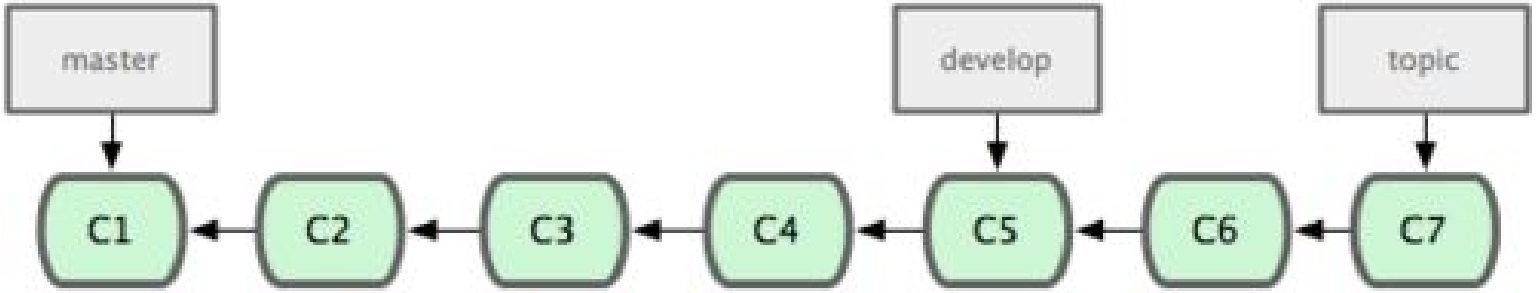
Eclipse > Team > Add to index

Eclipse > Team > Commit…

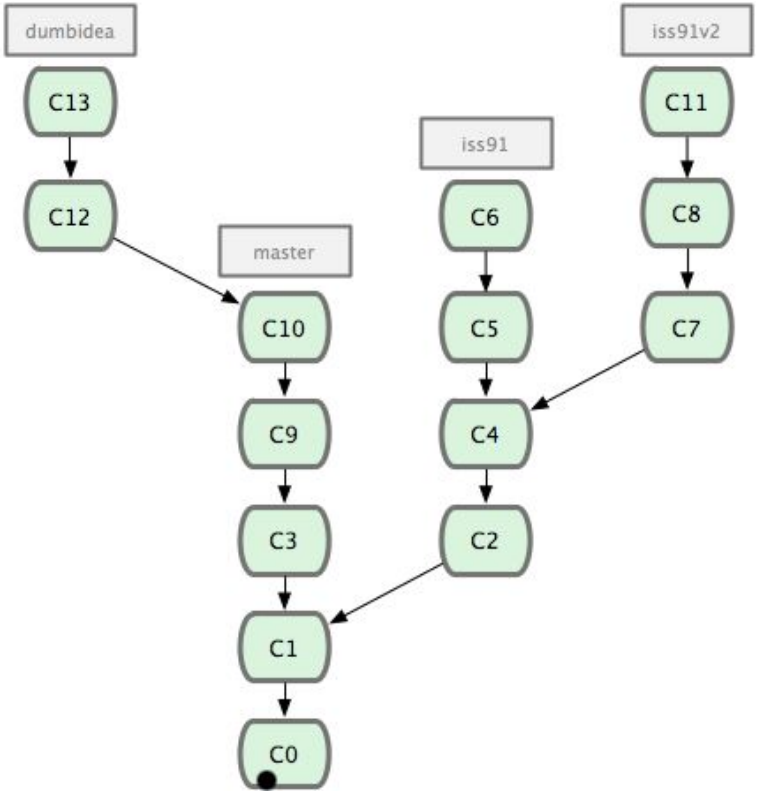
Eclipse > Team > Push upstream

Workflow

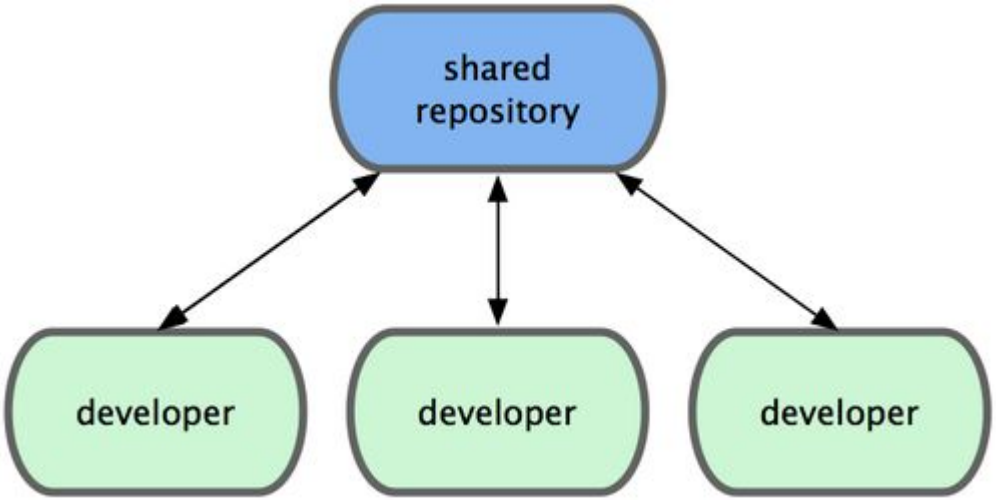
Workflow of branches: Long Running



Workflow of branches: Topic

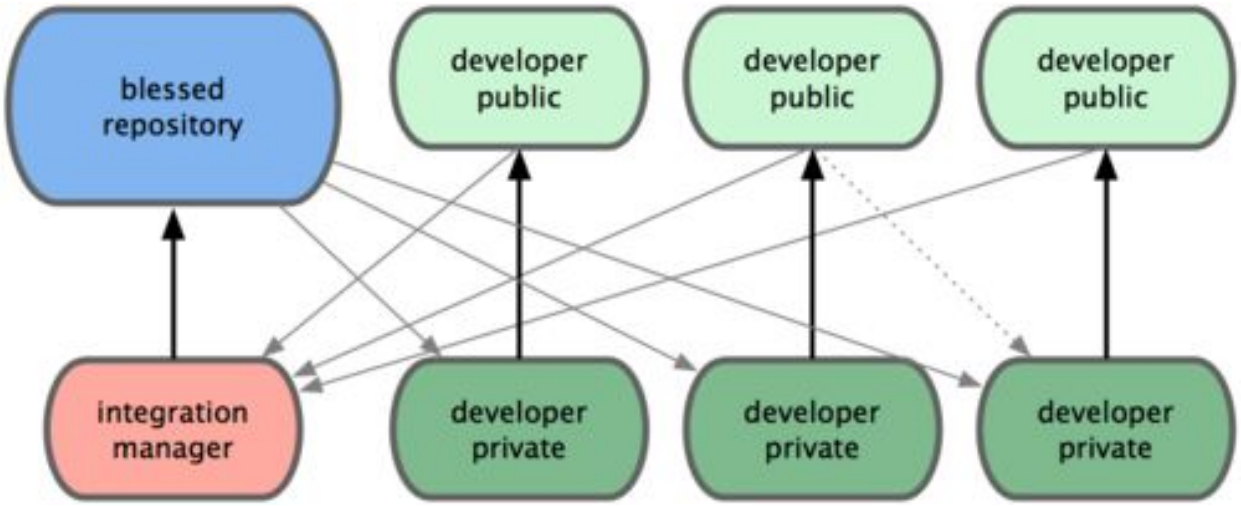


Workflow of repos: Centralized



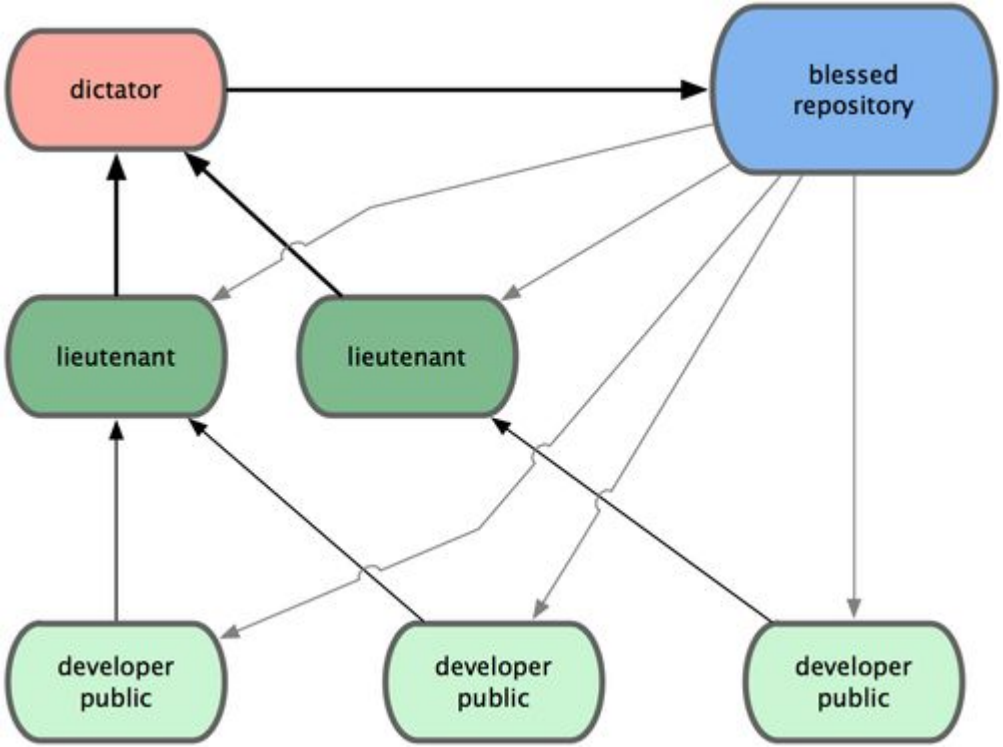
Pull request: sollicite une opinion experte avant de merger sa branche avec le `master` du dépôt

Workflow of repos: Integ. Manager



Pull request: demande à l'integration manager de pull un dépôt de développement.

Workflow of repos: Benevolent dictator



Pull request: demande au dictateur/lieutenant de pull un dépôt de lieutenant/développement.

In case of fire

1. git commit

2. git push

3. leave building

1. git commit

2. git push

3. leave building

Références

http://git-scm.com/book/