

# Git & github

## Gestion collaborative des sources

Laurent Jouanneau  
Miage d'Evry octobre 2012

Licence d'utilisation : <http://creativecommons.org/licenses/by-sa/3.0/>



# Principes des CVCS/DVCS

# Comment sauvegarder ?

- Dans un projet, les fichiers évoluent, sont créés, sont effacés
- Comment faire pour garder des versions intermédiaires de ces fichiers, notamment à chaque version du projet ?
  - 1) On fait une sauvegarde à la main dans des répertoires différents, dans des disques différents
    - fastidieux, sujet à erreurs, collaboration difficile
  - 2) On utilise un logiciel dédié à cette tâche : un « VCS »

# Définition

- VCS = Version Control System
- Un gestionnaire de version de sources stocke les changements successifs effectués sur des fichiers
- Les changements sont enregistrés dans une base de donnée spéciale appelée « dépôt ».
- C'est l'utilisateur qui lance l'enregistrement quand il le désire : il « commit ».
- Un enregistrement dans la base = un état à un instant t du projet = « changeset » ou « commit »

# Pourquoi

- « Versionner » les changements sert à
  - Pouvoir tester des solutions et revenir en arrière
  - Retrouver les modifications qui ont provoqué des régressions, des bugs
  - Trouver les changements effectués pour une fonctionnalité similaire à celle que l'on veut développer → comprendre le fonctionnement de l'application
  - Permettre au système de gestion de pouvoir fusionner le travail de plusieurs développeurs → c'est un **outils de collaboration** indispensable pour le développement

# CVCS et DVCS

- Il y a deux types de gestionnaire de source :
  - Les systèmes centralisés
  - Les systèmes décentralisés

# Central VCS

- Central Version Control System
- Un serveur stocke toutes les modifications
- Le client dialogue avec le serveur pour toute opération
- Le client ne possède qu'une version des fichiers
- schema

# Central VCS

- Avantages
  - Tout le monde a accès aux dernières modifications
  - Réglage fin des droits d'accès
- Inconvénients
  - Nécessité d'avoir accès au serveur pour travailler, pour commiter et faire la plupart des opérations (log, diff...)
  - L'historique est stocké à un seul endroit
  - Serveur inaccessible, base de donnée corrompue, détruite : gros impact sur le développement
- Exemple de logiciels : CVS, Subversion



# Distributed VCS

- Distributed Version Control System
- Chaque utilisateur a sa propre copie du dépôt (un clone), avec tout l'historique et toutes les données
- Chacun peut récupérer les modifs d'un autre clone
- Il peut y avoir un dépôt « central », qui sert de référence
- Ex : Mercurial, Git, Bazaar, ...
- schema

# Distributed VCS

- Avantages
  - Clonage → peu de chance de perdre les sources
  - Pas besoin d'être connecté pour travailler
  - Plus récents, donc plus évolués, plus puissants (merge ...)
  - Expérimentation plus aisée et historisée
- Inconvénients
  - Un peu plus complexe à apprendre
  - Moins évident à administrer un dépôt central (droits..)

# Introduction sur Git

# Pourquoi Git

- DVCS
- Moderne, puissant, performant
- Devient le VCS le plus utilisé
- Popularité « boostée » par les fonctionnalités offertes par le site Github, facilitant les collaborations

# Historique

- Créé par Linus Torvalds
- En remplacement du VCS Bitkeeper (non libre) pour gérer le projet Linux, les autres VCS libres n'ayant pas les fonctionnalités voulues
- Prototype développé en 3 jours et sortie le 6 avril 2005
- Version 1.0 en décembre 2005
- Pour comparaison : CVS en 1990, Visual SourceSafe en 1994, Subversion et Bitkeeper en 2000, Mercurial en 2005, Bazaar en 2007.

# Téléchargement et installation

- Git est avant tout un outil en ligne de commande
  - <http://git-scm.com>
- Téléchargement : <http://git-scm.com/download/>
  - Pour windows cela installe <http://msysgit.github.com/>
- Pour Linux, préférez passer par le système de dépôt de la distribution (apt-get install...)
- Suivre les instructions d'installation
- Ouvrez un terminal, c'est prêt.
  - `git help`

# Fonctionnement interne de Git

# Stockage

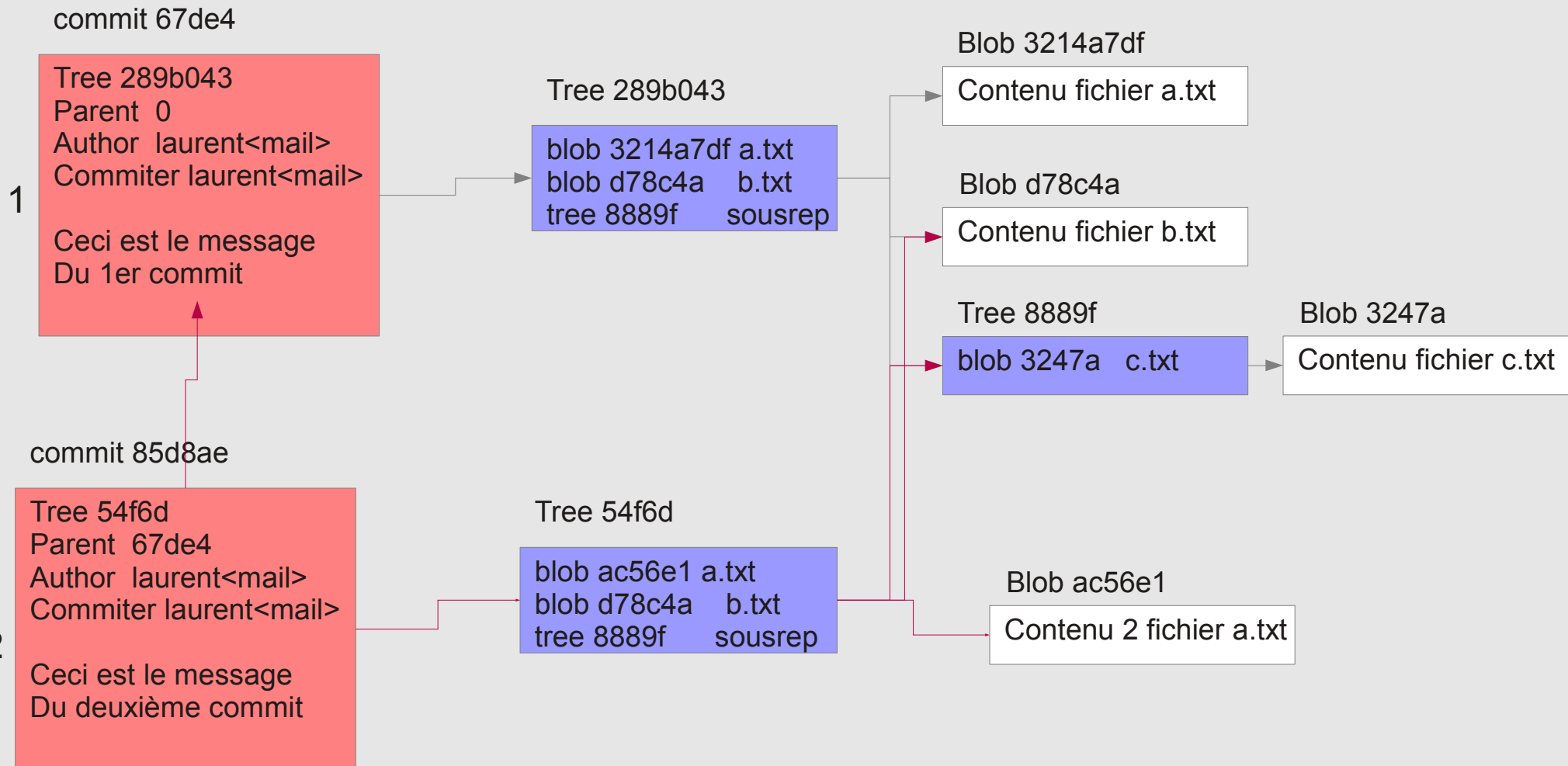
- Ne stocke pas les différences entre chaque version
- Stockes chaque version des fichiers dans une sorte de système de fichiers
- Intégrité du contenu → checksum
- Ajout d'information, pas de suppression



# Stockage

- La base de donnée Git contient quatre type d'objets
- Ils sont tous identifiés par un nombre hexa qui est le checksum sha1 de leur contenu
- **Blob** : contient le contenu d'un fichier. Il ne stockes ni de nom, ni autres attributs `git show 5529b198`
- **Tree** : liste les fichiers d'un répertoire (nom + id du blob correspondant + permissions + type) `git ls-tree 6894ac45`
- **Commit** : pointe vers le tree correspondant et le(s) commit(s) parent(s). Contient également message + auteur + date
- **Tag** : point vers un objet (commit), et identifié par un nom

# stockage



# Utilisation basique de Git

# Configuration de Git

- Avant toute chose, il est préférable de configurer git pour indiquer :
  - Son nom
  - Son email
- Ces informations seront utilisées dans les commits
- Stockées dans ~/.gitconfig (--global) ou dans le fichier « config » du dépôt

```
git config --global user.name "Pierre Durant"  
git config --global user.email "Pierre@durant.name"
```

```
git config --help
```

```
git config --global core.editor vim
```

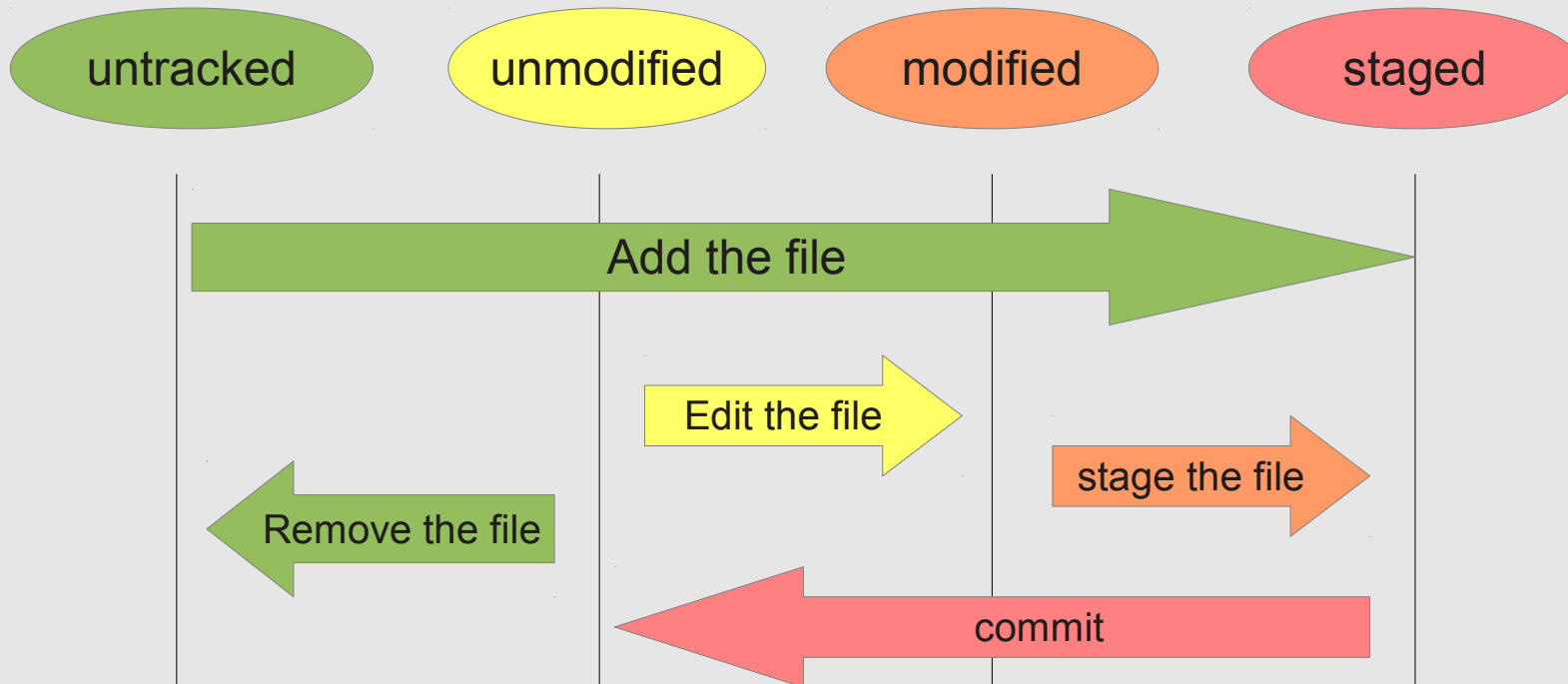
```
git config --global color.ui true
```

# Création d'un dépôt

- On veut commencer à historiser les changements des fichiers d'un répertoire (d'un projet)
- On se place dans un répertoire puis : `git init`
- → répertoire `.git` = la « base de donnée » du dépôt

# Principes de fonctionnement

- Un fichier passe par 4 états



- Pour savoir l'état des fichiers : `git status`

# Principe de fonctionnement

- **git add** : pour ajouter un nouveau fichier dans le dépôt, ou pour indiquer de prendre en compte les modifications d'un fichier dans le prochain commit
- **git rm** : pour supprimer un fichier
- **git mv** : pour renommer un fichier
- **git commit** : pour valider les ajouts de fichiers ou de modifications de fichier, et valider les suppressions de fichier. Cette commande s'accompagne toujours d'un message expliquant le changement.

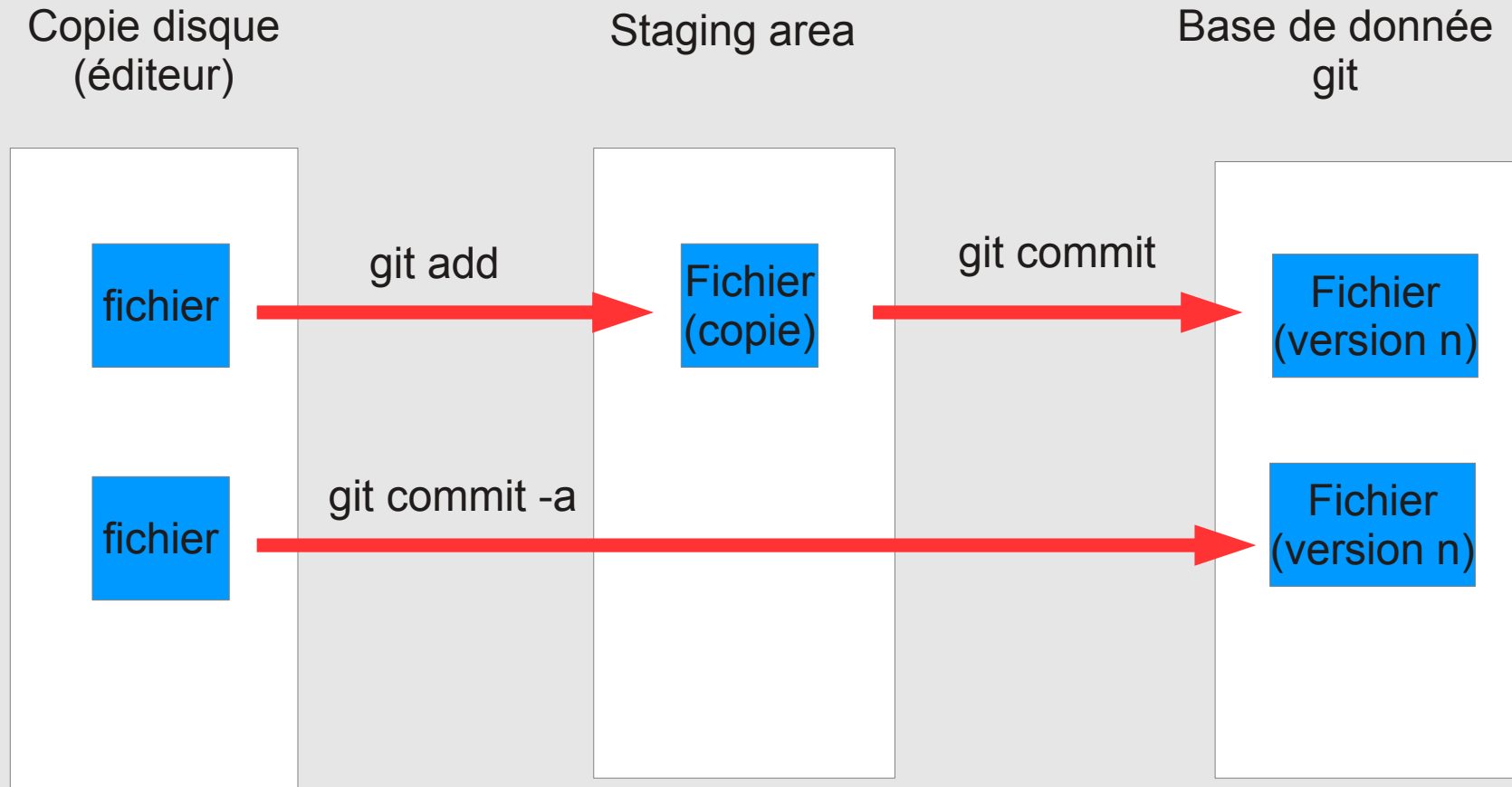
# Staging area

- « zone » qui contient les fichiers en « staged »
- Elle contient donc la version des fichiers tels qu'ils étaient lors du git add.
- Ce sont ces versions qui seront sauvegardées lors du commit
- → permet de sélectionner les modifications pour le prochain commit
- Si un fichier « staged » est modifié avant le commit : il a alors les deux états, « modified » et « staged » correspondant à deux versions différentes du fichier. Seule la version « staged » est prise en compte dans le prochain commit.
- Pour commiter des changements sans passer par la staging area :

```
git commit -a
```

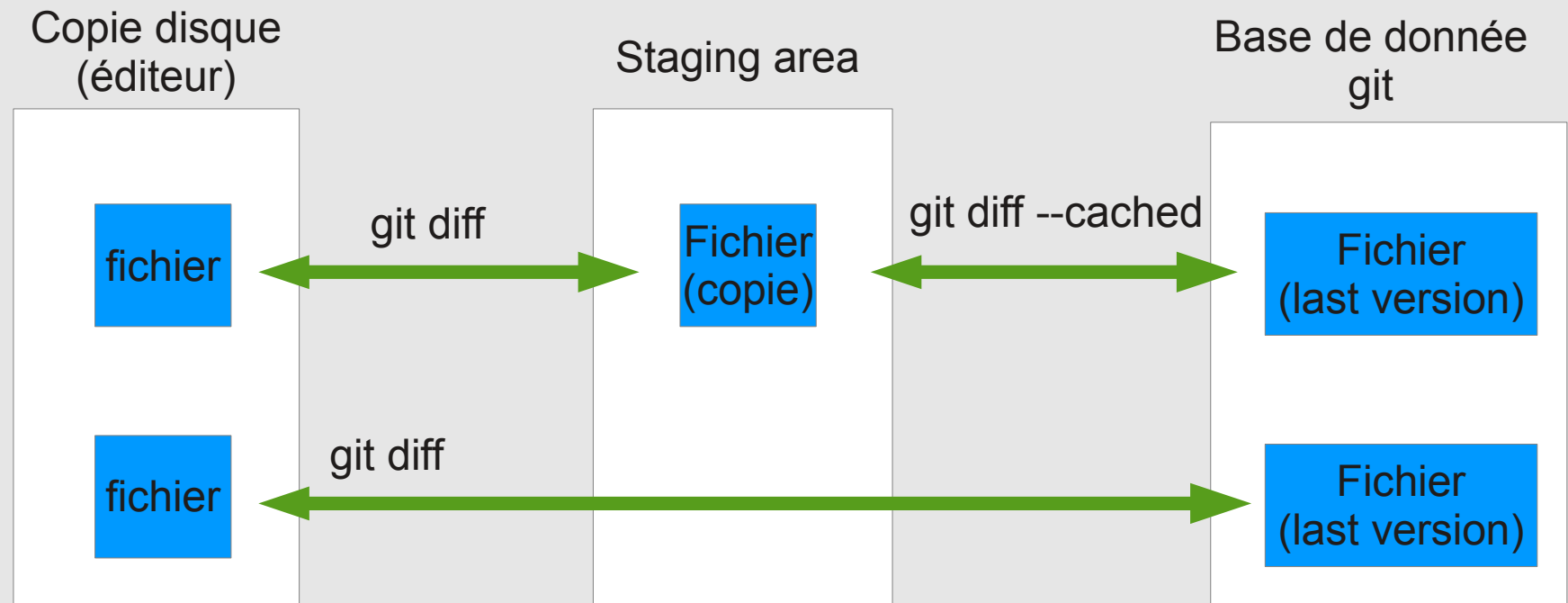


# Staging area



# Voir les modifications

- **git diff** : pour voir les modifications des fichiers ayant l'état « modified »
- **git diff --cached** : pour voir les modifications des fichiers ayant l'état « staged »



# Annuler des modifications

- D'un fichier « modified »
- D'un fichier « staged »
- Modifier le dernier commit
- Annuler le dernier commit

```
git checkout -- fichier.txt
```

```
git reset HEAD fichier.txt
```

```
git commit --amend
```

```
git reset --hard HEAD^
```

attention, peut être dangereux

# Commit

- Un commit est toujours accompagné par un texte, un message
  - Une première ligne faisant office de résumé
  - Si on veut ajouter des détails (pourquoi ces modifications, conséquences à prendre en compte par la suite etc...), plusieurs lignes de texte séparé du résumé par un saut de ligne.
- Paramètre « -m » en ligne de commande ou dans l'éditeur qui s'ouvre automatiquement

```
git commit -m "correction bug #1234 : crash lorsque v=0"
```

# Quand commiter ?

- Un commit = une étape d'un développement
  - Une petite fonctionnalité
  - Une étape de refactorisation
  - Correction d'un bug
- Pas de commit pour chaque modification de fichiers
- Pas de commit non plus contenant une tonne de modifications. Il y a forcément des étapes qui ont été faites.
- Faire en sorte que les modifications enregistrées dans un commit fassent sens : il faut pouvoir avoir un historique clair et utilisable = pouvoir identifier qui quoi quand pourquoi.

# Ignorer des fichiers

- Fichier .gitignore
- Contient la liste des fichiers à ne pas prendre en compte dans le dépôt
- Exemple :

```
TAGS
.DS_Store
_dist/
*.komodoproject
.komodotools
```

# Historique d'un dépôt

- `git log`
- `gitk`

# Tags

- Étiquette pour retrouver facilement un commit particulier
- Le plus souvent pour étiqueter le commit qui finalise la sortie d'une version du projet.
- Pour étiqueter le dernier commit : `git tag VERSION_1.0`
- Lister les tags : `git tag`
- Mettre à jour le code source correspondant à un tag :  
`git checkout VERSION_1.0`
- Supprimer un tag : `git tag -d VERSION_1.0`



# Utilisation des branches avec Git

# Qu'est-ce qu'une branche ?

- Conceptuellement :
  - c'est une suite de commit qui a été réalisée en parallèle à une autre suite de commit
  - L'historique a donc divergé à un instant T
  - Il peut y avoir plusieurs branches en même temps. Elles ont chacune un nom
  - Branche principale : « master »
  - Une branche peut rejoindre une autre branche : « fusion de branche », ou « merge »

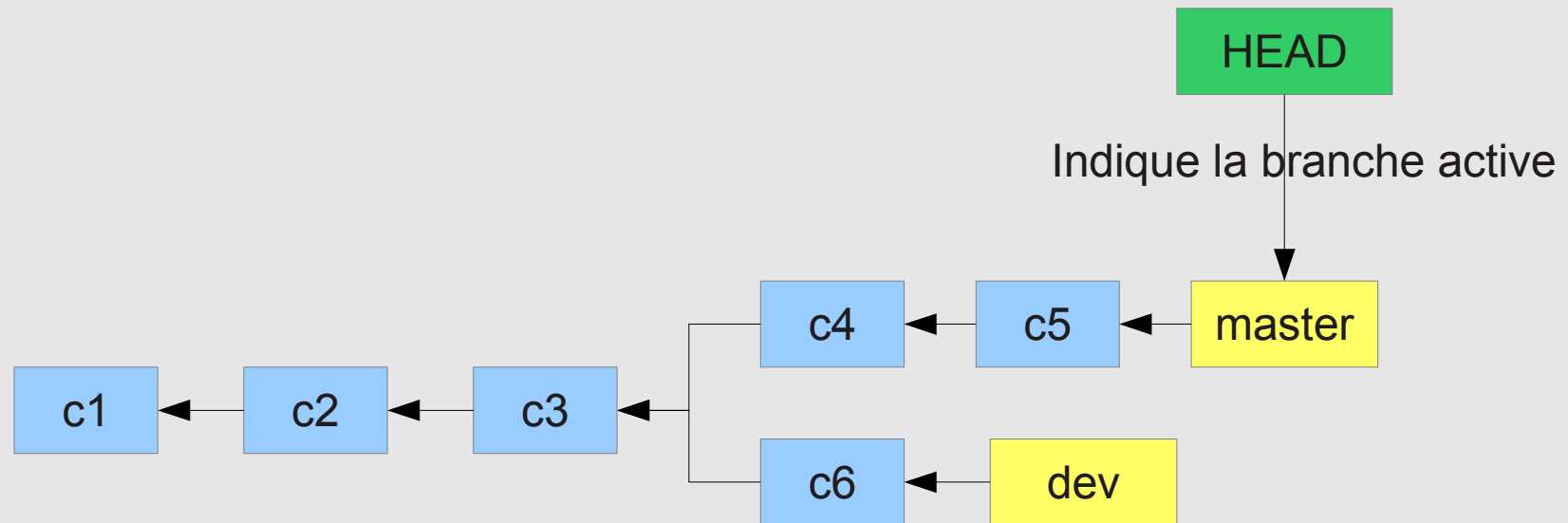
# Pourquoi des branches ?

- Création d'une branche pour le développement d'une fonctionnalité, la correction d'un bug, ou expérimenter : la branche principale reste ainsi dans un état « stable »
- Branches pour maintenir des versions stables : une branche par version, dans lesquelles on ne corrige que les bugs.
- Branches pour des versions particulières d'un projet
- Avec Git, la création/fusion/suppression des branches est très peu couteux (disque, perf), on peut donc en utiliser tant qu'on veut

# Qu'est-ce qu'une branche ?

- Techniquement :
  - C'est un pointeur vers un commit, en l'occurrence, le dernier commit d'une suite de commit.
  - Ce pointeur est modifié à l'ajout d'un nouveau commit, si ce dernier est le successeur du commit pointé.
  - Pointeur spécial : HEAD, qui pointe vers le pointeur de la branche active

# Qu'est-ce qu'une branche ?



# Créer / activer une branche

- Pour créer une branche :

```
git branch foo
```

```
git branch foo 5defc
```

- Le point de départ de la branche est le commit courant

- Pour activer une branche :

```
git checkout foo
```

- HEAD est mis à jour

```
git checkout 84a96
```

- Les sources du répertoire sont mis à jour

- Les prochains commits seront rattachés à la branche courante

- Création + activation en même temps :

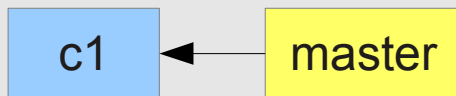
```
git checkout -b foo 65df
```

- Lister les branches

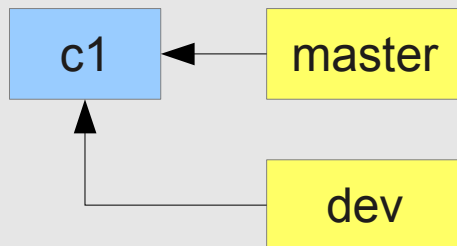
```
git branch
```

# Utilisation des branches

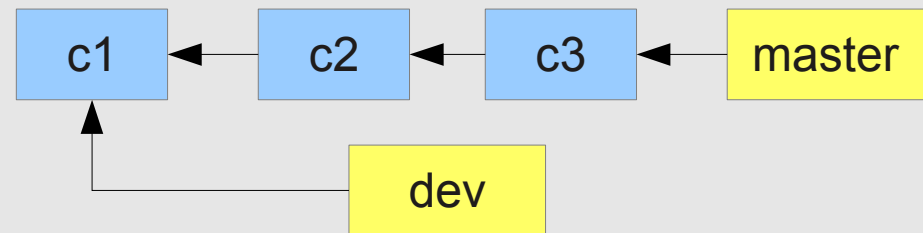
1 `git add && git commit`



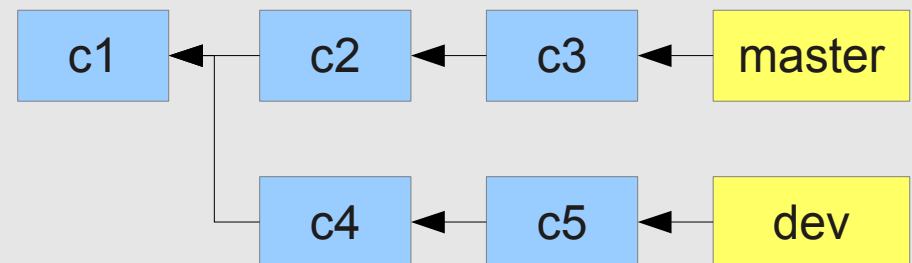
2 `git branch dev`



3 `git add && git commit`  
`git add && git commit`



4 `git checkout dev`  
`git add && git commit`  
`git add && git commit`

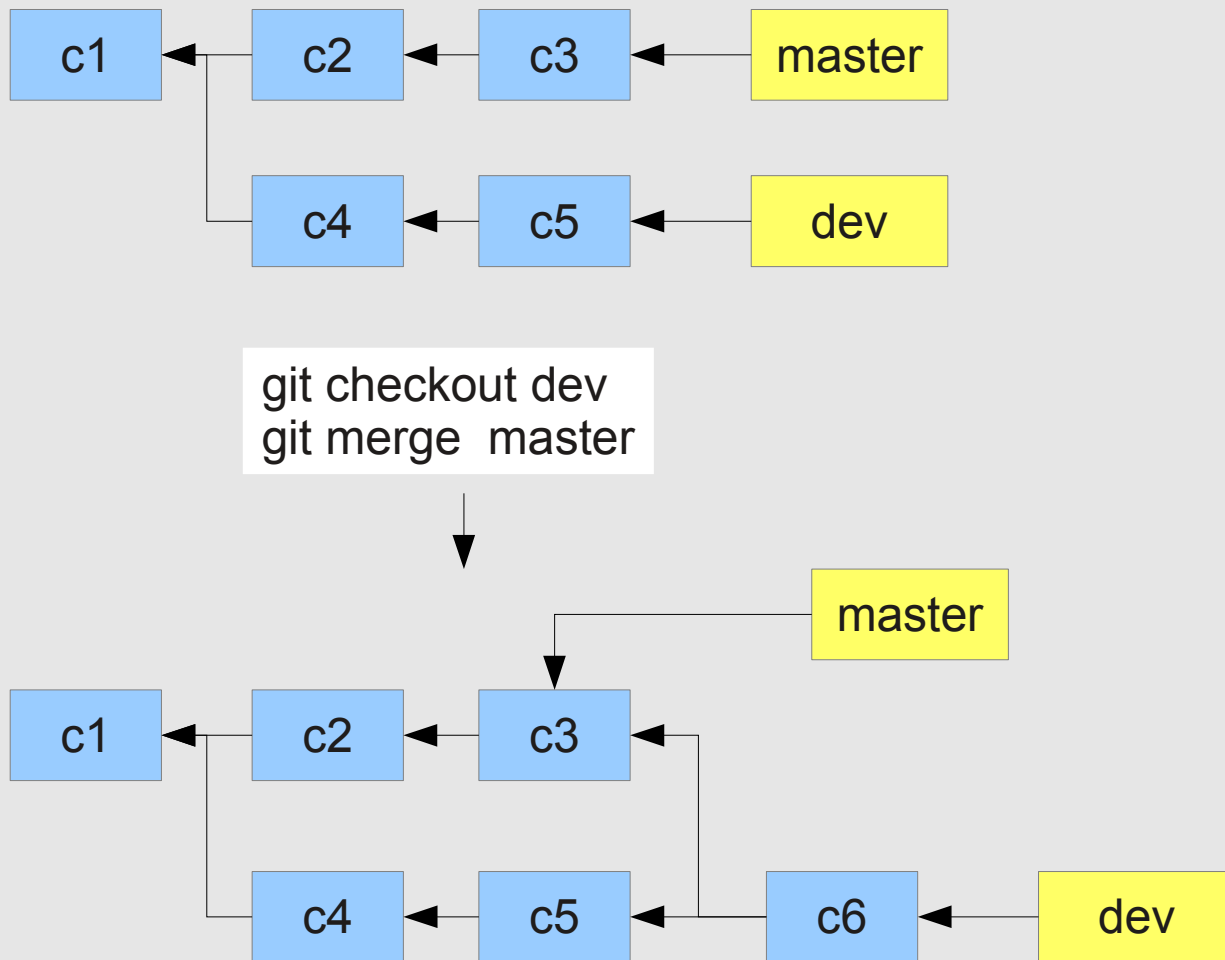


# Fusion de branches

- But : intégrer les changements d'une branche (« test » par exemple) dans une autre (« master » par ex)
- Actions :
  - Se mettre dans la branche dans laquelle on veut intégrer les commits de l'autre branche `git checkout master`
  - Fusionner avec la commande merge `git merge test`
- La commande merge crée un nouveau commit dont les changements sont le résultat de la fusion



# Fusion de branche



# Conflits de fusion

- La fusion peut échouer à cause de conflits
- Des marqueurs ont été ajoutés dans les fichiers concernés, permettant de repérer les parties de code en conflits

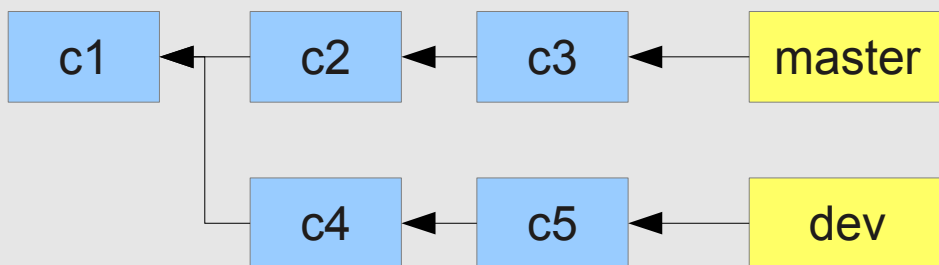
```
<<<<<<<<<
Ici le contenu d'origine
=====
Ici le contenu à fusionner
>>>>>>>>>
```

- Il faut alors corriger ces fichiers
- « git add »
- Puis commiter

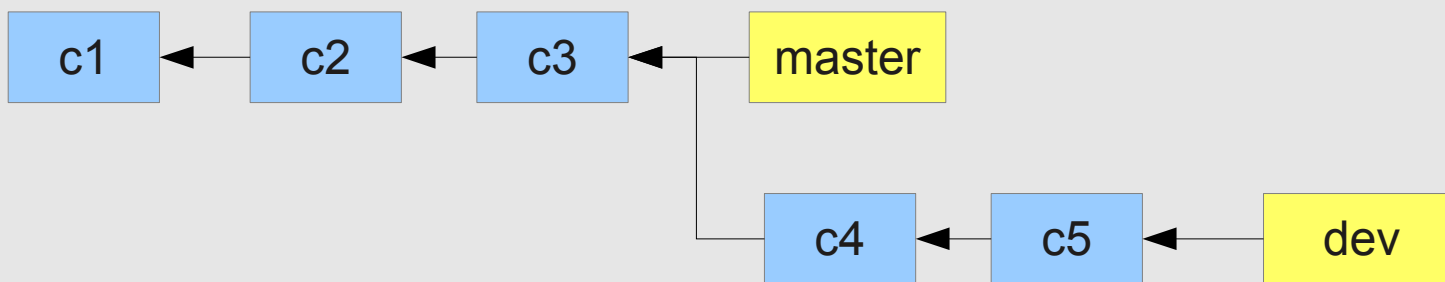
# Rebasing

- Autre manière de fusionner les changements d'une autre branche
- Les modifications d'une branche à fusionner sont re-committées à la suite des commits de la branche courante
- → historique plus clair
- Important : les commits sont recréés ! Aussi, si la branche à fusionner a déjà été publiée, le rebase n'est pas possible, risque de dysfonctionnement.
- **Ne pas faire un rebase dans une branche dont tout les commits ont déjà été publiés (push)**

# rebasing



```
git checkout dev  
git merge --rebase master
```



# Développement collaboratif avec Git

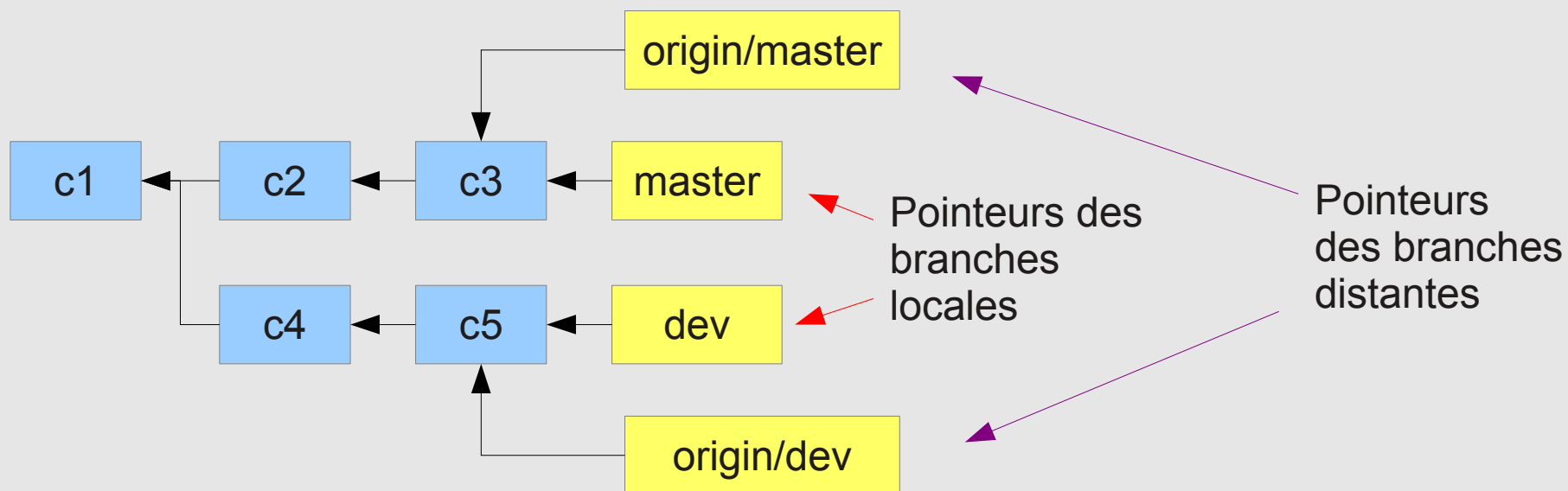
# Clonage d'un dépôt

- Quand on travaille à plusieurs, il faut s'échanger les modifications, les commits
- En général : création du dépôt local en clonant un dépôt distant

```
git clone git@github.com:laurentj/comete2011.git
```

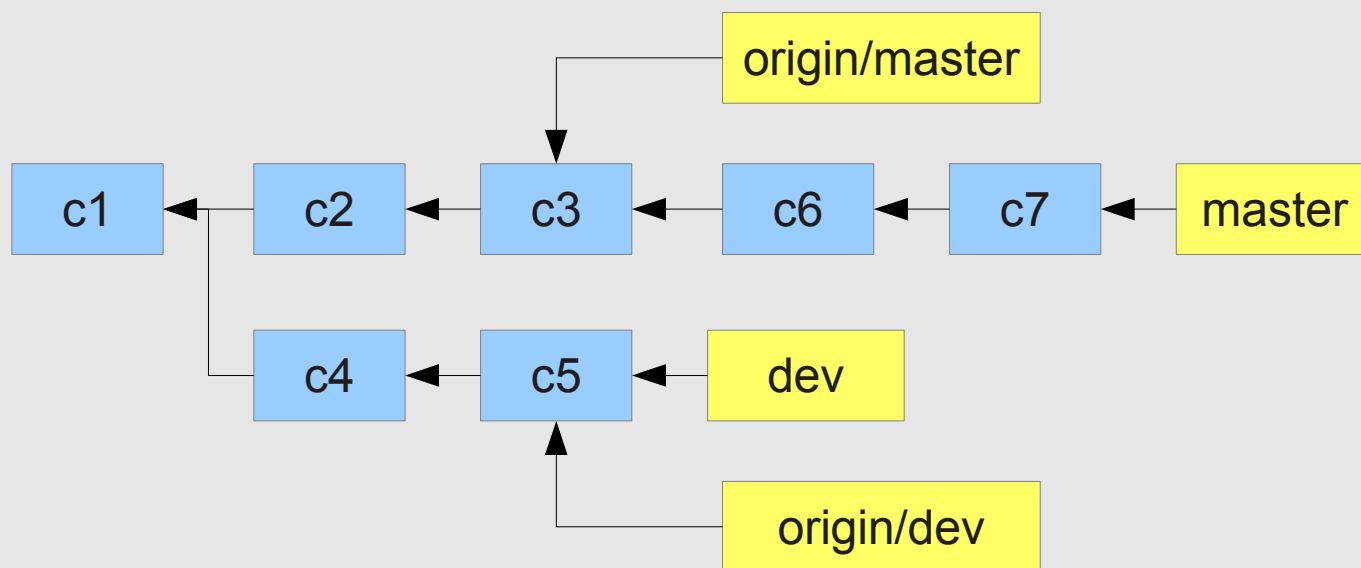
- Il y a alors les branches locales, et les branches distantes
- En local, les branches distantes se nomment « aliasdepot/nombranche »
- Le dépôt d'où l'on a cloné s'appelle « origin ».
- Le nom peut être changé

# Clonage d'un dépôt



# Clonage d'un dépôt

Après des commits (c6 & c7) dans master





# Mises à jour : pull & push

- Pour pousser les modifications vers le dépôt distant :

```
git checkout mabranche  
git push origin branchedistante
```

- Pour récupérer des modifications :

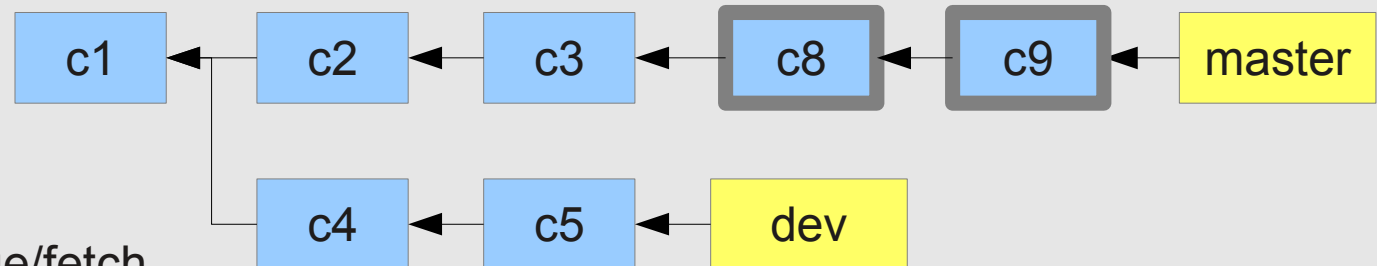
```
git checkout mabranche  
git pull origin branchedistante
```

- Pull = fetch + merge
- Avec rebase (les commits locaux sont déplacés après le dernier commit de la branche distante)

```
git pull --rebase origin master
```

# Fetch + merge #1

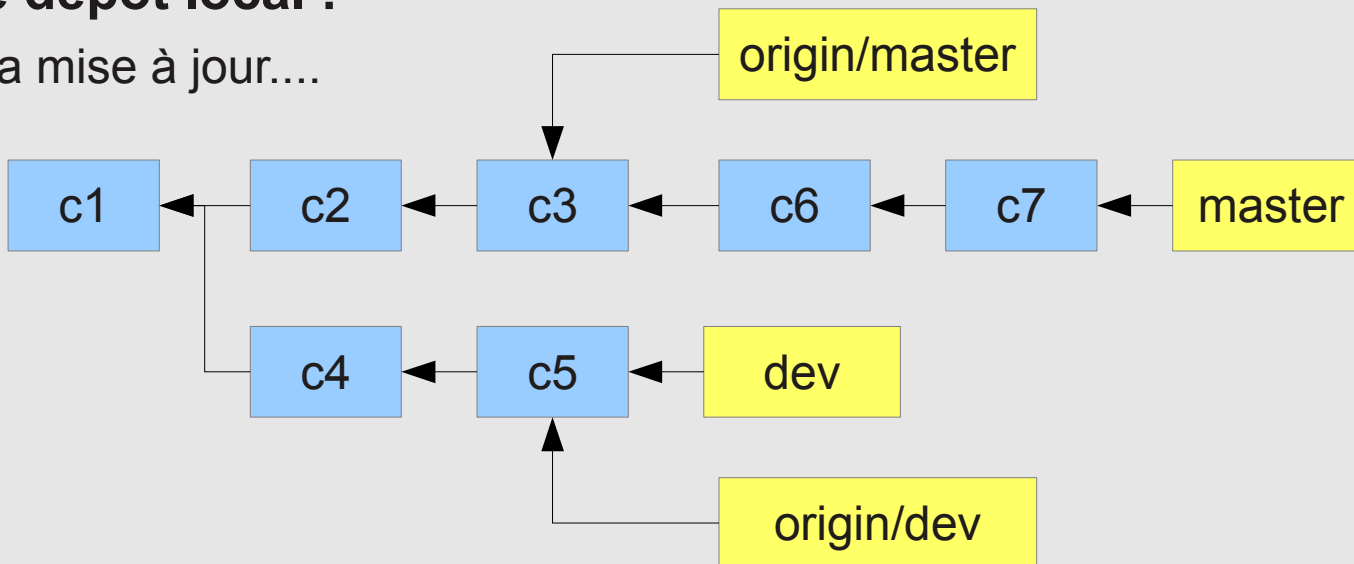
**Dans le  
dépôt distant :**



Deux commits c8 et c9  
En plus depuis le clonage/fetch

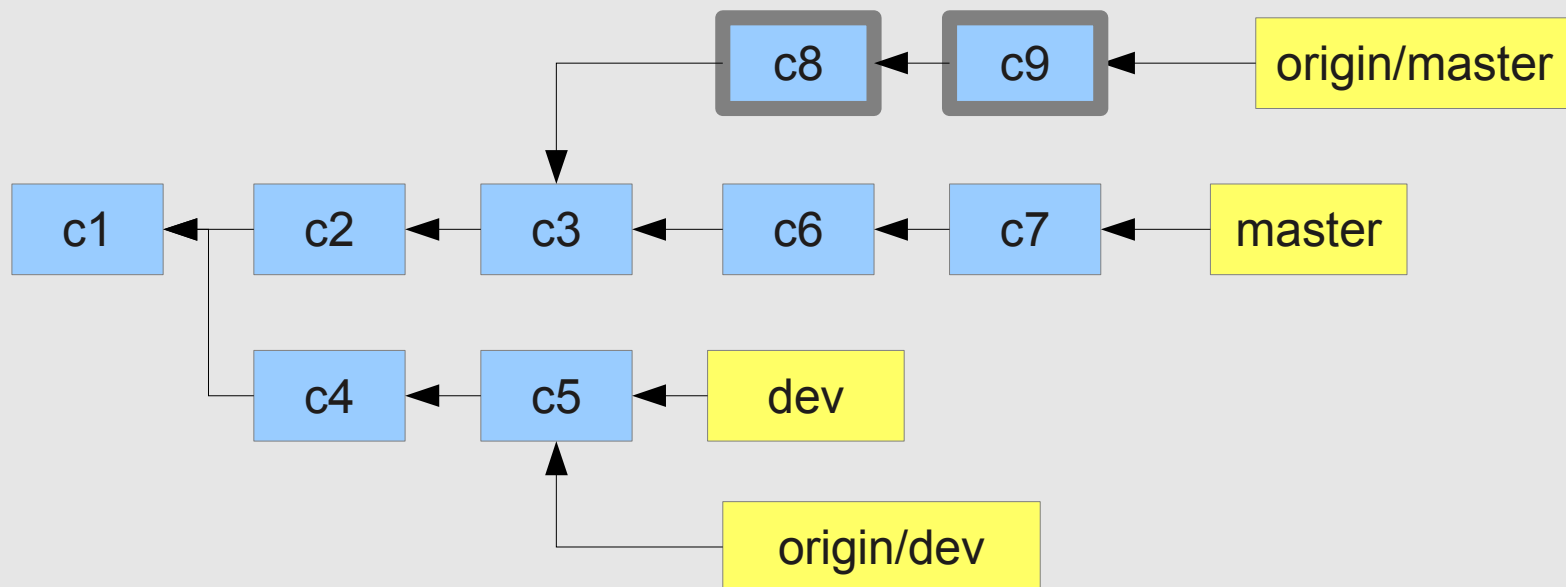
**Dans le dépôt local :**

Avant la mise à jour....



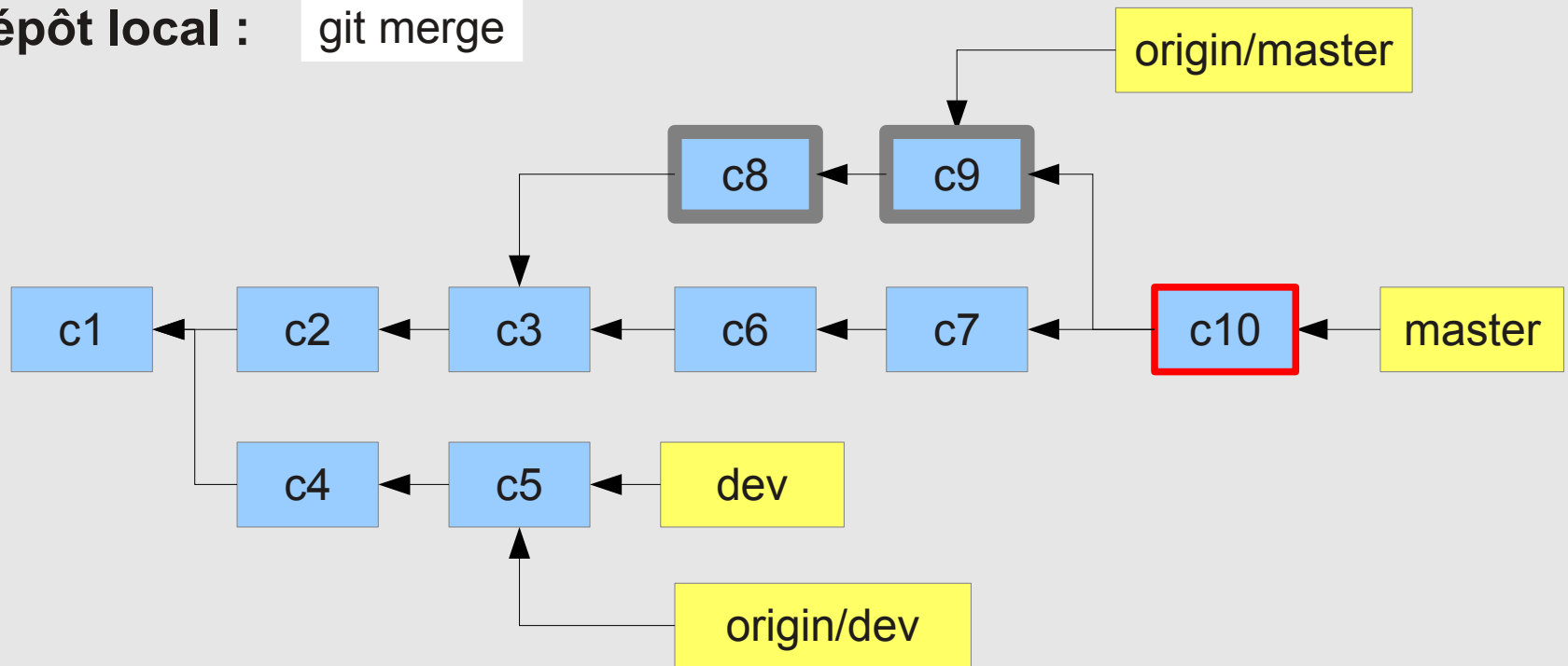
# Fetch + merge #2

Dans le dépôt local : `git fetch origin master`



# Fetch + merge #3 == pull

Dans le dépôt local : `git merge`



`git fetch origin && git merge`

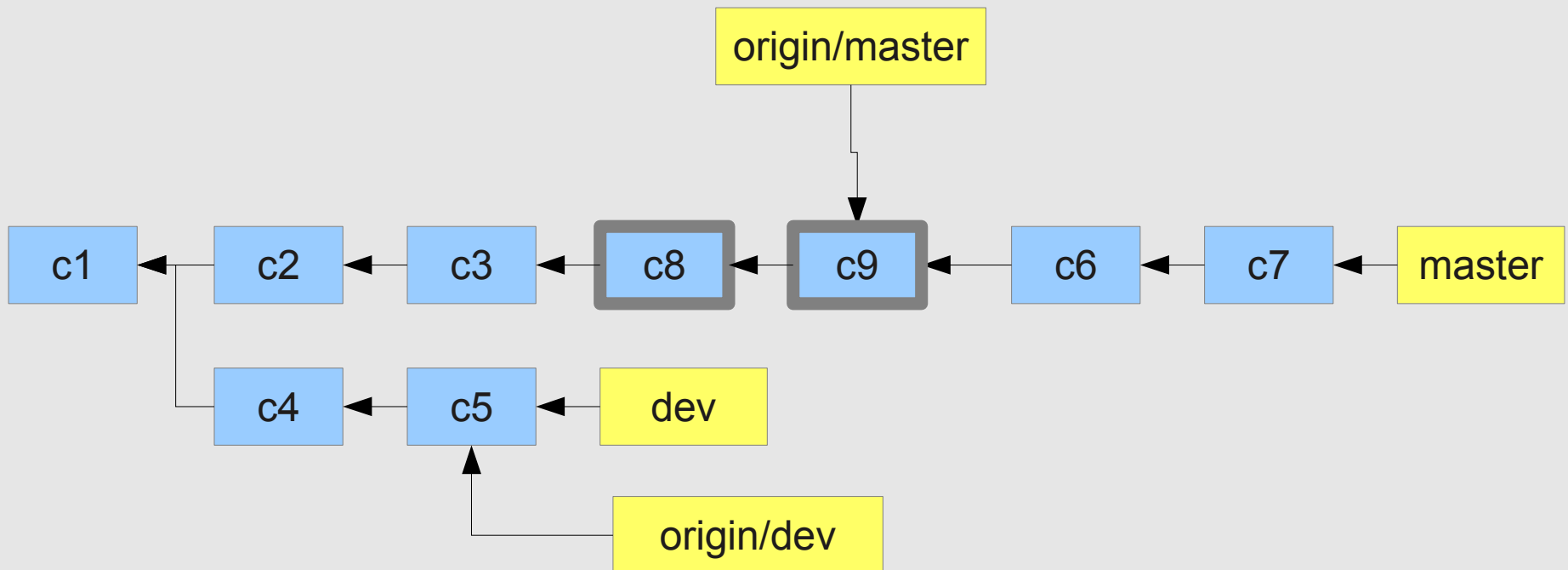
`==`

`git pull origin master`

Après un `git push`, le dépôt distant contiendra les mêmes commit qu'en local

# Fetch + rebase

Dans le dépôt local : `git rebase` (À la place de merge)



`git fetch origin && git rebase`

`==`

`git pull --rebase origin master`

# Plusieurs dépôts distants

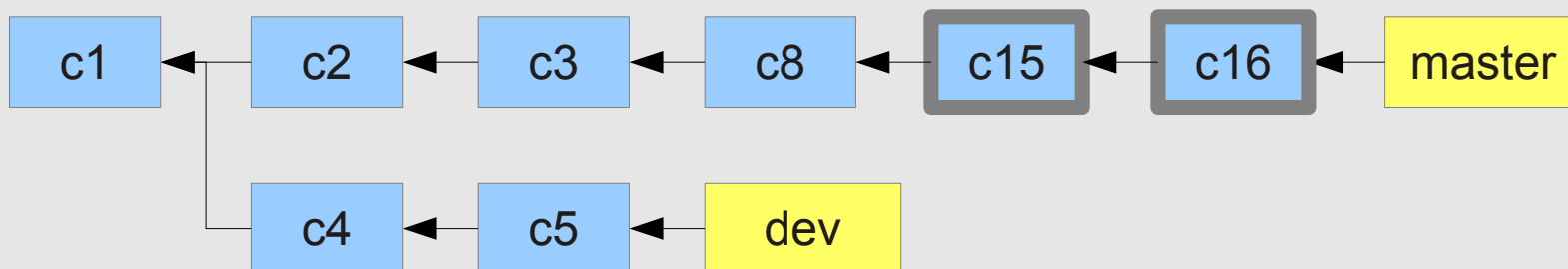
- À la place de « origin », on peut indiquer des chemins ou des URLS d'autres dépôts distants : on peut s'échanger des modifications entre plusieurs dépôts
- On peut créer des alias de ces URLS
- Gestion de ces alias et des informations des dépôts distant : « git remote »

```
git remote          # liste des dépôts
git remote add unAlias URLdepot

→ git pull unAlias master
```

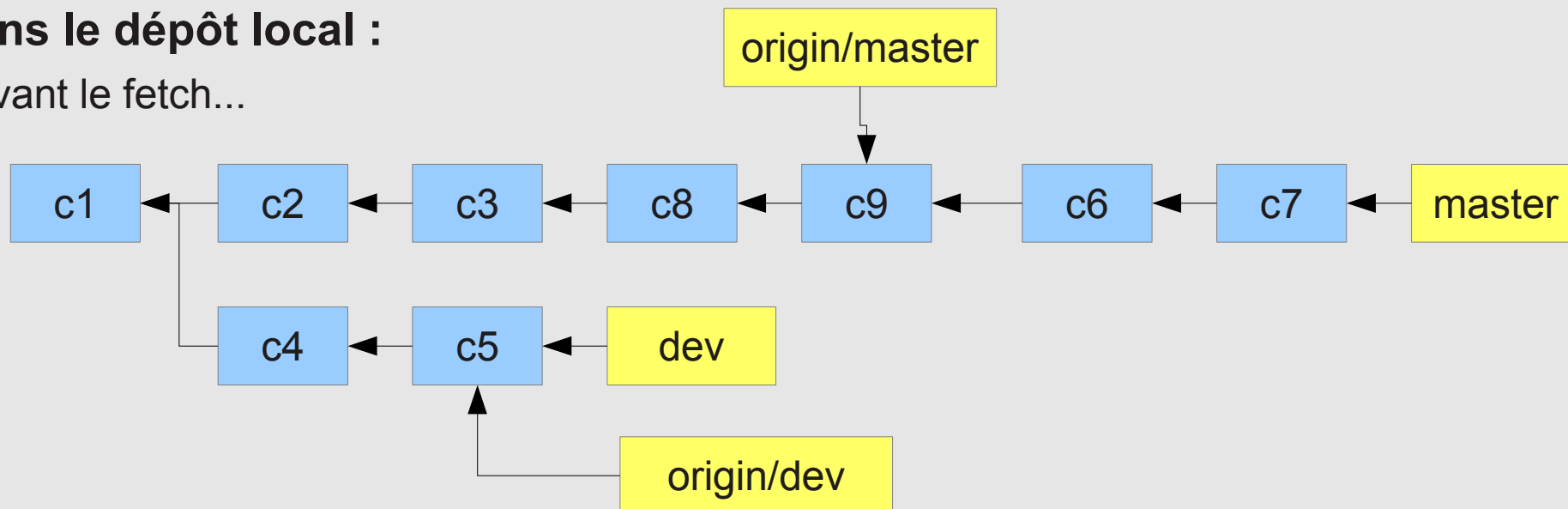
## 2 dépôt distants

Dans le 2ieme dépôt distant (celui de paul) :



Dans le dépôt local :

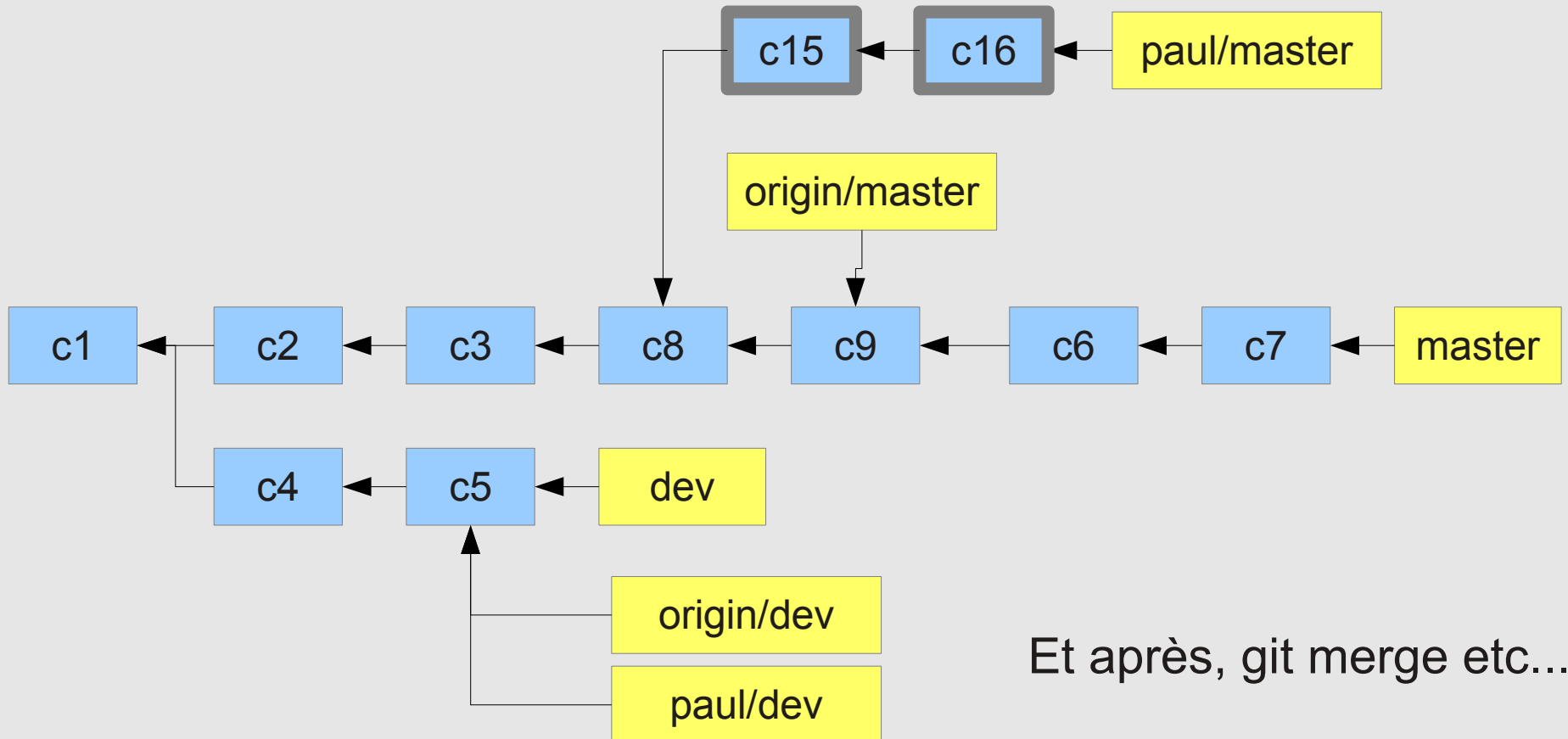
Avant le fetch...



## 2 dépôt distants #2

Dans le dépôt local :

```
git remote add paul git@paul.com:projet.git  
git fetch paul
```



Et après, git merge etc...



# Workflows

# Workflow de branches

- Suivant les besoins d'un projet, sa criticité, ses développeurs, l'utilisation des branches va être différentes
- Plusieurs manière de travailler sont possibles

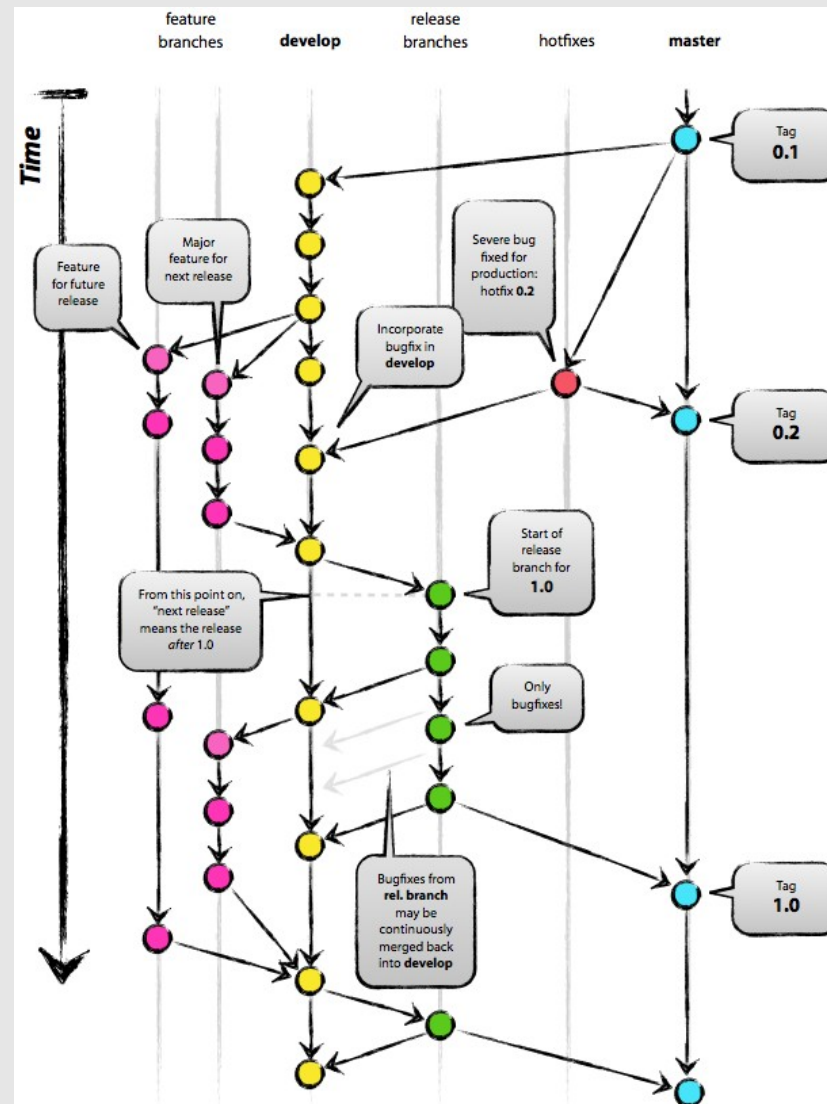
# Workflow de branches simple

- La branche master contenant le projet « stable »
- Une branche « develop » pour y effectuer le développement des fonctionnalités en cours et les stabiliser. Quand « develop » est stable → fusion dans « master »
- Une branche « topic » issue de « develop » pour les nouveaux développement. Fusion avec « develop » quand cela devient utilisable
- Pour les petits projets

# Workflow de branche #2

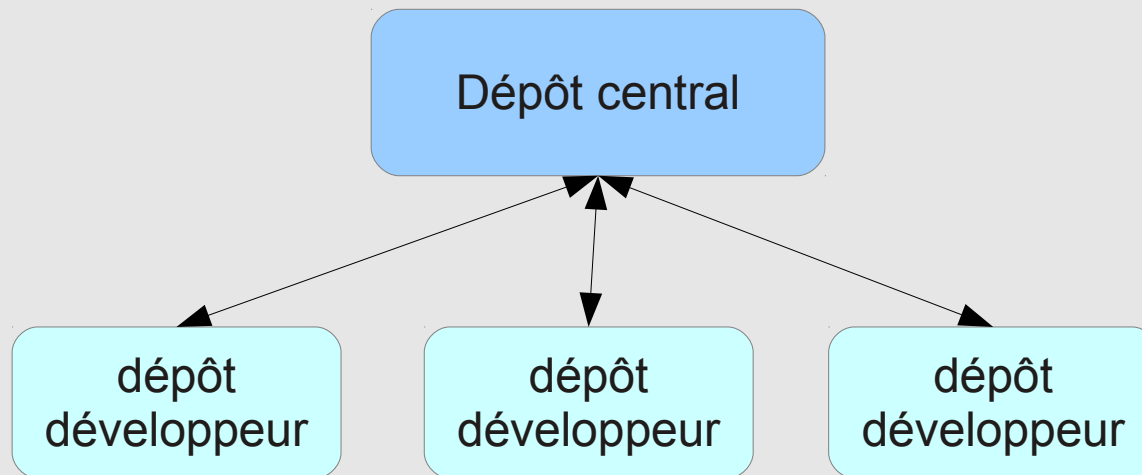
- Une branche par sujet de développement
- Une branche « master » dans laquelle on fusionne les dernières nouveautés
- Des branches pour chaque version stables publiées, où l'on y fait que des corrections de bugs
- D'autres workflows existent et plus complexe. Ex :  
« gitflow »
  - <http://nvie.com/posts/a-successful-git-branching-model/>
  - <https://github.com/nvie/gitflow>

# gitflow



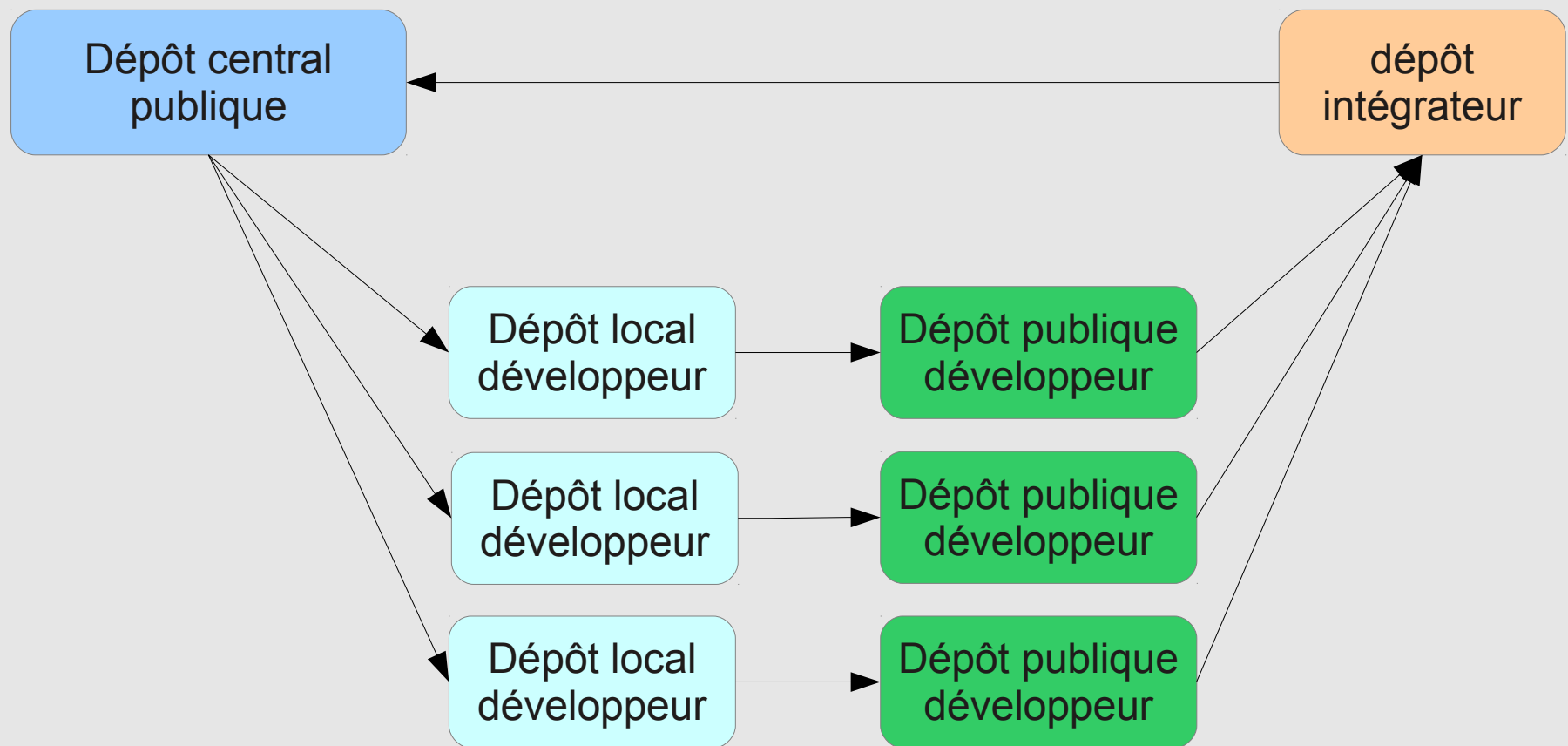
# Workflows collaboratifs

- But : définir la manière de collaborer
- En voici des exemples
- Workflow centralisé :



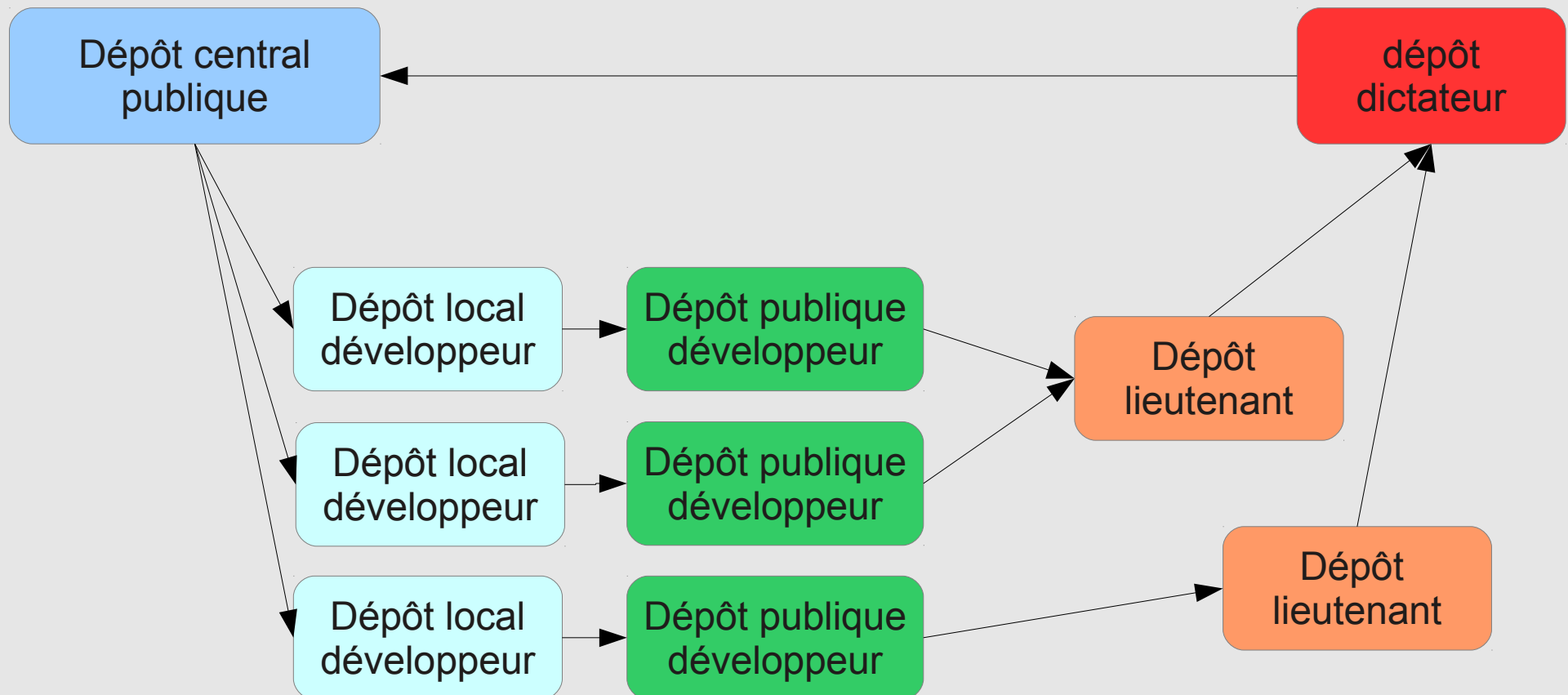
# Workflows collaboratifs

- Workflow d'intégration



# Workflows collaboratifs

- Workflow dictatorial : délégation d'intégration à des « lieutenants »





Github



# Github.com


- Site favorisant la collaboration du développement d'un projet (privé ou publique)
- Interface utilisateur facilitant
  - Le clonage des dépôts (« fork »)
  - La fusion des contributions
  - La revue de code
- Utilisé par de nombreux projets libres
- Équivalent « libre » : [gitorious.org](https://gitorious.org)
- Équivalent pour Mercurial : [bitbucket.org](https://bitbucket.org)


# Compte





- Création d'un compte
  - Indiquez une clé ssh : facilite l'authentification


# Compte & Création dépôt


Search or Type a Command 


 Explore Gist Blog Help


 **laurentj**   




**Laurent Jouanneau**  
laurentj

 Innophi

 [ljouanneau@gmail.com](mailto:ljouanneau@gmail.com)

 <http://ljouanneau.com>



 Joined on Jul 18, 2010


**11**  
followers


**16**  
starred


**1**  
following

**Organizations**


 **Repositories**


 Public Activity


 **Edit Your Profile**


Filter laurentj's repositories...

**All** [Public](#) [Private](#) [Sources](#) [Forks](#) [Mirrors](#)





**jelix** PHP ★ 2 🔗 26  
forked from [jelix/jelix](#)  
a PHP5 framework  
Last updated a day ago





**jcommunity** PHP ★ 0 🔗 2  
A set of modules for the Jelix framework, to bring the support of a community management in an application : registration, profile editing etc  
Last updated 4 days ago

**password\_compat** PHP ★ 0 🔗 15  
forked from [ircmaxell/password\\_compat](#)  
Compatibility with the password\_\* functions being worked on for PHP 5.5  
Last updated 16 days ago



**havefnubb** PHP ★ 1 🔗 4  
forked from [havefnubb/havefnubb](#)  
HaveFnuBB - the powerfull Forum made with Jelix PHP5 Framework  
Last updated a month ago

# Création dépôt


[Explore](#) [Gist](#) [Blog](#) [Help](#)


 **laurentj**

**Owner**

 **laurentj** 

**Repository name**







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

---


**Description** (optional)

☒ **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.


Add .gitignore: **None** 

---

**Create repository**

# Création dépôt

**Quick setup** — if you've done this kind of thing before

or **HTTP** **SSH** `git@github.com:laurentj/cours-git.git` 

We recommend that every repository has a **README**, **LICENSE**, and **.gitignore**

## 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 git@github.com:laurentj/cours-git.git
git push -u origin master
```

## Push an existing repository from the command line

```
git remote add origin git@github.com:laurentj/cours-git.git
git push -u origin master
```

# Navigateur de code

Après un push vers le dépôt nouvellement créé...

CodeNetworkPull Requests0Issues0WikiGraphsAdmin

exemple de depot git — [Read more](#)

ZIP

HTTP

SSH

Git Read-Only

git@github.com:laurentj/cours-git.git

Read+Write access

branch: master

Files

Commits

Branches1

Tags

Downloads

Latest commit to the master branch

ajout plugin pour jquery

laurentj

 authored 2 minutes ago

commit fe9aaeaf5

cours-git /

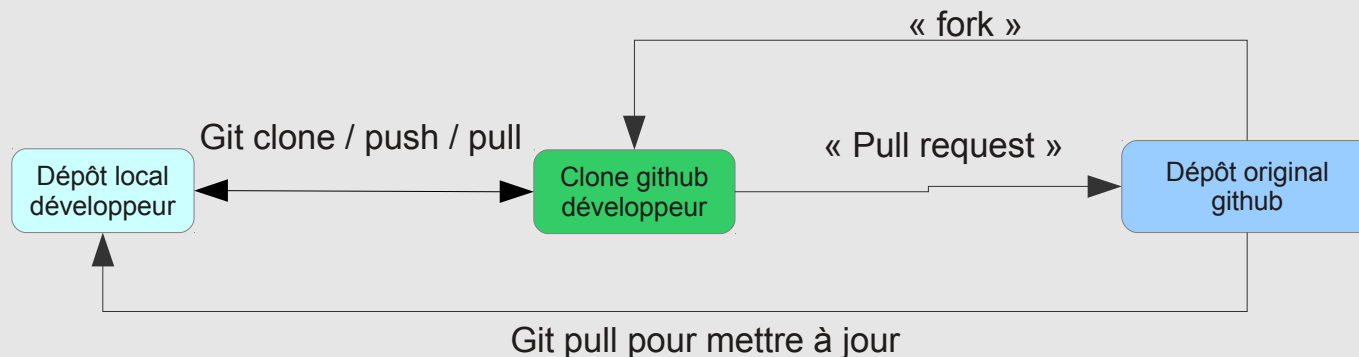
name	age	message	history
<div>README.txt</div>	9 days ago	deuxieme commit [laurentj]	
<div>jquery.modal.js</div>	2 minutes ago	ajout plugin pour jquery [laurentj]	

README.txt

Ceci est mon super projet.  
  
C'est un projet demonstration.

# Contribuer

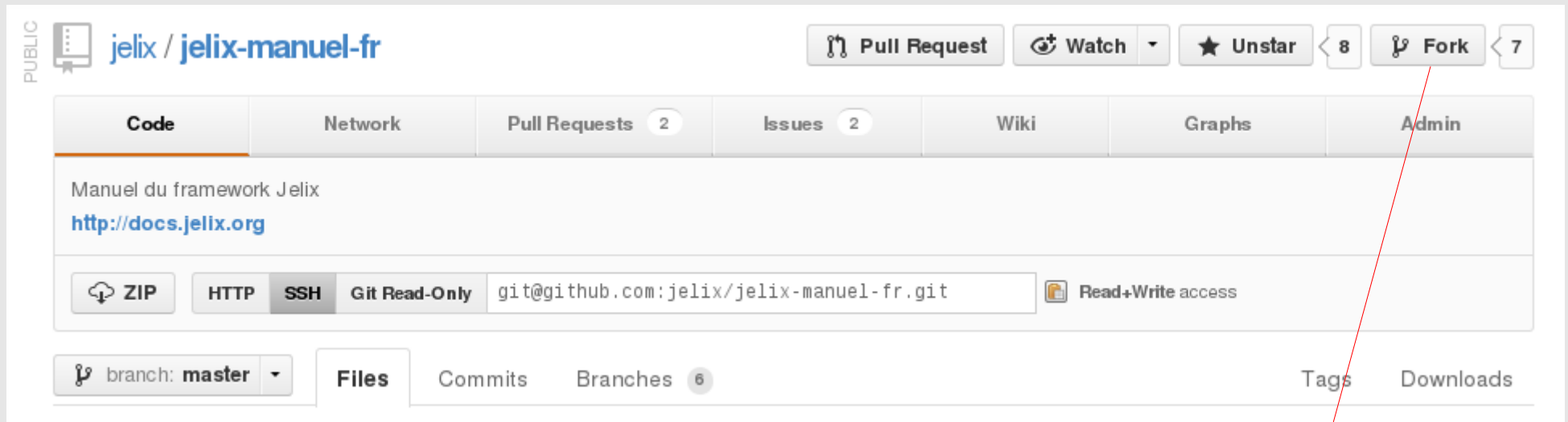
- La première chose : « forker » le dépôt cible
  - → on a un clone dans son compte
- On clone ce clone sur sa machine en local
- On commit
- On « push » vers son clone github
- On fait un « pull request » vers le dépôt original sur github






# Fork

Pour contribuer à un projet : cloner le dépôt sur son compte → « fork »



Public  jelix / jelix-manuel-fr

Pull Request Watch Unstar 8 Fork 7

Code Network Pull Requests 2 Issues 2 Wiki Graphs Admin

Manuel du framework Jelix  
<http://docs.jelix.org>

ZIP HTTP SSH Git Read-Only git@github.com:jelix/jelix-manuel-fr.git Read+Write access

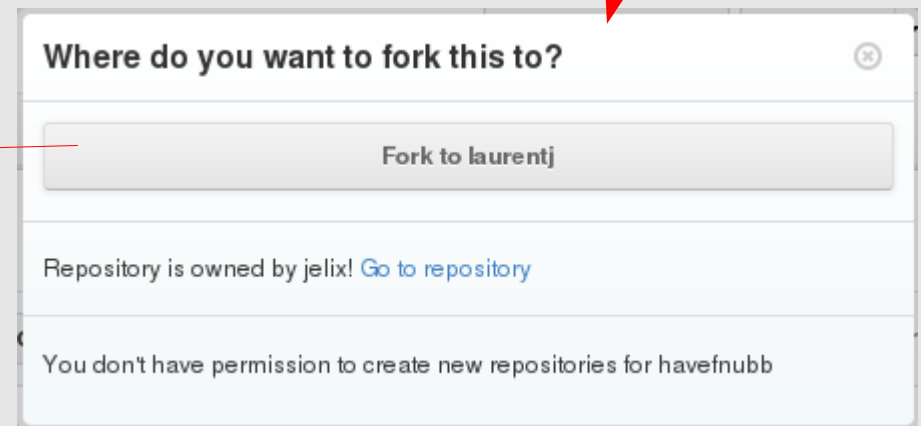
branch: master Files Commits Branches 6 Tags Downloads



Public  laurentj / jelix-manuel-fr  
forked from jelix/jelix-manuel-fr

Code Network Pull Requests 0

Manuel du framework Jelix  
<http://docs.jelix.org>



Where do you want to fork this to?

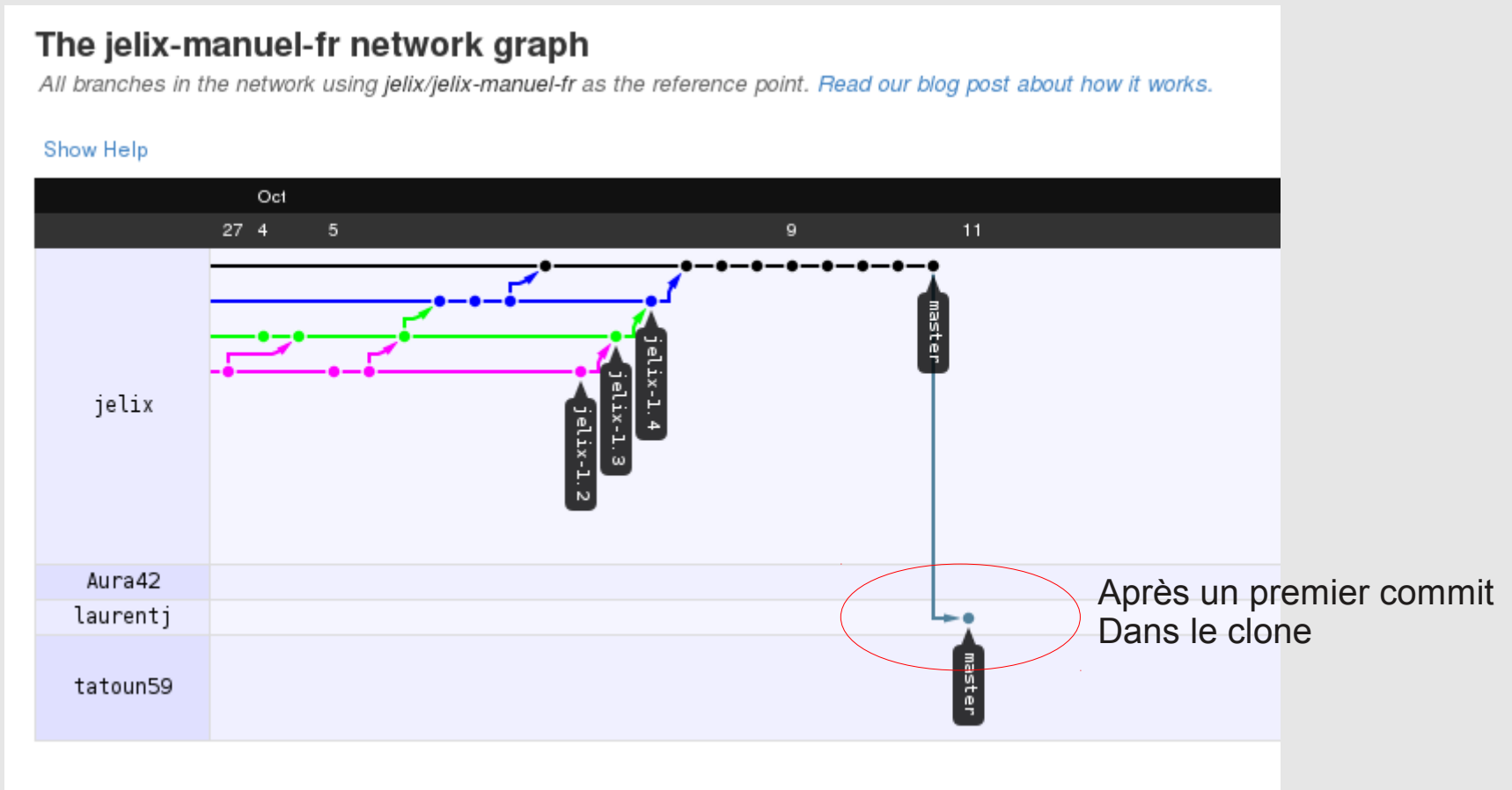
Fork to laurentj

Repository is owned by jelix! [Go to repository](#)

You don't have permission to create new repositories for havefnubb

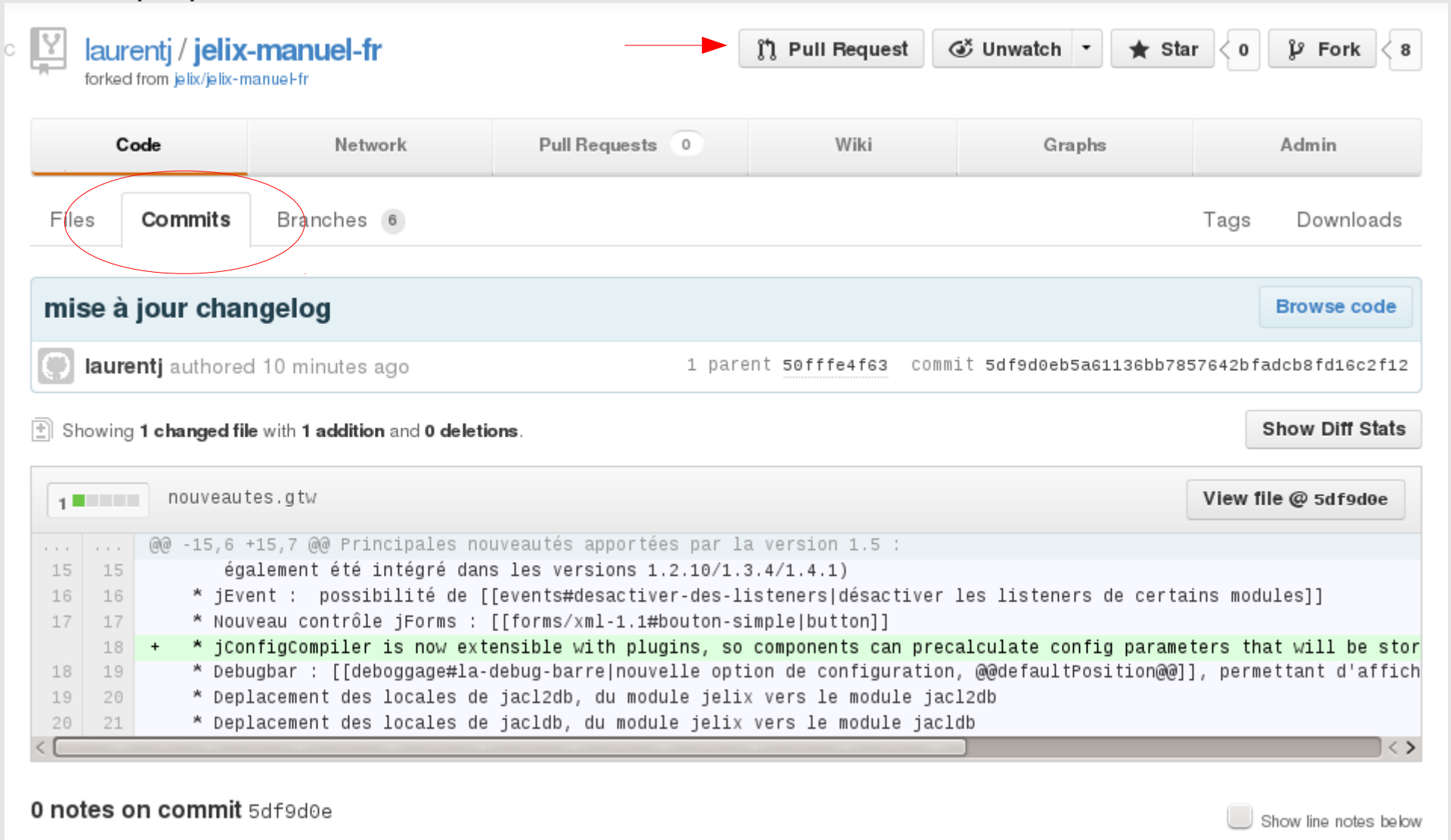
# Visualiser les clones

Onglet « network » : graphe des commits / branches, du projet et de ses clones




# Pull request

On veut proposer une modification





The screenshot shows the GitHub interface for the repository **laurentj / jelix-manuel-fr**, which is forked from **jelix/jelix-manuel-fr**. At the top, navigation buttons include **Pull Request** (highlighted with a red arrow), **Unwatch**, **Star** (0), and **Fork** (8). Below these are tabs for **Code**, **Network**, **Pull Requests** (0), **Wiki**, **Graphs**, and **Admin**. Under the **Code** tab, there are sub-tabs for **Files**, **Commits** (circled in red), and **Branches** (6). The main content area displays a commit titled **mise à jour changelog** by **laurentj**, authored 10 minutes ago. The commit message is: `@@ -15,6 +15,7 @@ Principales nouveautés apportées par la version 1.5 :  
 également été intégré dans les versions 1.2.10/1.3.4/1.4.1)  
 * jEvent : possibilité de [[events#desactiver-des-listeners|désactiver les listeners de certains modules]]  
 * Nouveau contrôle jForms : [[forms/xml-1.1#bouton-simple|button]]  
 + * jConfigCompiler is now extensible with plugins, so components can precalculate config parameters that will be stor  
 * Debugger : [[debuggage#la-debug-barre|nouvelle option de configuration, @@defaultPosition@@]], permettant d'affich  
 * Déplacement des locales de jac12db, du module jelix vers le module jac12db  
 * Déplacement des locales de jac1db, du module jelix vers le module jac1db`. The diff shows one file changed, **nouveautes.gtw**, with 1 addition and 0 deletions. A **Show Diff Stats** button is present. At the bottom, it states **0 notes on commit 5df9d0e** and includes a **Show line notes below** option.



# Pull request

PUBLIC  **laurentj / jelix-manuel-fr**  
forked from [jelix/jelix-manuel-fr](#)

Choix des dépôts/branches sources et cibles

base repo: **jelix/jelix-manuel-fr** head repo: **laurentj/jelix-manuel-fr**  
base branch: **master** head branch: **master**

**New Pull Request**  Commits 1  Files Changed 1

 mise à jour de la liste des changements  message


**Write** Preview Comments are parsed with [GitHub Flavored Markdown](#)

Leave a comment


Explication de ce qui est proposé (quoi/pourquoi)

**Send pull request**


# Pull Request

PUBLIC  [jelix / jelix-manuel-fr](#) [Pull Request](#) [Watch](#) [Unstar](#) [8](#) [Fo](#)



[Code](#) [Network](#) [Pull Requests](#) [3](#) [Issues](#) [3](#) [Wiki](#) [Graphs](#) [Admi](#)

[Open](#) **laurentj** wants to merge 1 commit into `jelix:master` from `laurentj:master` 1 


[Discussion](#) [Commits](#) [1](#) [Files Changed](#) [1](#)


 **laurentj** opened this pull request a minute ago [Edit](#)


**mise à jour de la liste des changements**

No one is assigned  No milestone 


No description given.


1 participant 

 **laurentj** added a commit 17 minutes ago

 **laurentj** mise à jour changelog [5df9d0e](#)

You can add more commits to this pull request by pushing to the **master** branch on **laurentj/jelix-manuel-fr**

 This pull request can be automatically merged. [Merge pull request](#)


 [Write](#) [Preview](#) Comments are parsed with [GitHub Flavored Markdown](#)

# PR : voir modifications


[Open](#) **laurentj** wants to merge 1 commit into `jelix:master` from `laurentj:master`

[Discussion](#) [Commits](#) 1 [Files Changed](#) 1

Showing 1 changed file with 1 addition and 0 deletions.

1  nouveautes.gtw

...	...	@@ -15,6 +15,7 @@ Principales nouveautés apportées par la version 1.5 :
15	15	également été intégré dans les versions 1.2.10/1.3.4/1.4.1)
16	16	* jEvent : possibilité de [[events#desactiver-des-listeners désactiver les listeners de c
17	17	* Nouveau contrôle jForms : [[forms/xml-1.1#bouton-simple button]]
18	18	+ * jConfigCompiler is now extensible with plugins, so components can precalculate config pa
18	19	* Debugbar : [[deboggage#la-debug-barre nouvelle option de configuration, @@defaultPosition
19	20	* Déplacement des locales de jac12db, du module jelix vers le module jac12db
20	21	* Déplacement des locales de jac1db, du module jelix vers le module jac1db

 38 58  
39 59  
40 60

**Tip:** You can add notes to lines in a file.  
Hover to the left of a line to make a note

# PR : Commenter des modifications

[Discussion](#) [Commits 1](#) [Files Changed 1](#)

Showing 1 changed file with 1 addition and 0 deletions. [Show Diff Stats](#)

1  nouveautes.gtw [View file @ 5df9d0e](#)

...

15

16

17

18

@@ -15,6 +15,7 @@ Principales nouveautés apportées par la version 1.5 :  
également été intégré dans les versions 1.2.10/1.3.4/1.4.1)  
\* jEvent : possibilité de [[events#desactiver-des-listeners|désactiver les listeners de certains modules]]  
\* Nouveau contrôle jForms : [[forms/xml-1.1#bouton-simple|button]]  
+ \* jConfigCompiler is now extensible with plugins, so components can precalculate config parameters that will be stor

0

Write

Preview

attention le texte est en anglais

Close form

Comment on this line

18


19


20

21

\* Debugbar : [[deboggage#la-debug-barre|nouvelle option de configuration, @@defaultPosition@@]], permettant d'affich  
\* Deplacement des locales de jac12db, du module jelix vers le module jac12db  
\* Deplacement des locales de jacldb, du module jelix vers le module jacldb




# Pull Request : fusion

 **laurentj** started a discussion in the diff 4 minutes ago

 nouveautes.gtw [View full changes](#)

```
... @@ -15,6 +15,7 @@ Principales nouveautés apportées par la version 1.5 :
15 15     également été intégré dans les versions 1.2.10/1.3.4/1.4.1)
16 16     * jEvent : possibilité de [[events#desactiver-des-listeners|désactiver les listeners de certains m
17 17     * Nouveau contrôle jForms : [[forms/xml-1.1#bouton-simple|button]]
18 + * jConfigCompiler is now extensible with plugins, so components can precalculate config parameters
```

 1

 **laurentj** [repo collab](#) 4 minutes ago  

attention le texte est en anglais

[Add a line note](#)

You can add more commits to this pull request by pushing to the **master** branch on **laurentj/jelix-manuel-fr**

**Preview the merge commit** [Cancel](#) [Confirm Merge](#)


 **laurentj**  
ljouanneau@gmail...

Merge pull request #16 from laurentj/master

mise à jour de la liste des changements




# Pull Request : résultat

**Closed** laurentj merged 1 commit into jelix:master from laurentj:master less than a minute ago 1  #16

Discussion


Commits 1


Files Changed 1



laurentj opened this pull request 16 minutes ago

**mise à jour de la liste des changements**

No one is assigned 

No milestone 

No description given.

**Closed**

+1 addition

-0 deletions

# Pull request : graph

## The jelix-manuel-fr network graph

Keyboard shortcuts available 

All branches in the network using jelix/jelix-manuel-fr as the reference point. [Read our blog post about how it works.](#)

[Show Help](#)

Last updated: 7 hours ago

