

Gestion de versions avec Git

Lucas Nussbaum

`lucas.nussbaum@univ-lorraine.fr`



Logiciels de gestion de versions

- ▶ Ou *Version Control System* (VCS)
ou RCS : Revision Control System,
ou SCM : Source/Software Code/Control Management
- ▶ Permettre la **traçabilité** d'un développement
 - ♦ *Qui a changé quoi ? Pourquoi ? Quand ?* (log, blame)
 - ♦ *Qu'est-ce qui a changé exactement ?* (diff)
- ▶ **Collaborer** sur un développement
 - ♦ Travailler à plusieurs sur un fichier, en gérant les conflits
- ▶ Pas limité au développement logiciel
 - ♦ Administration système (avec outils de gestion de configuration comme Puppet ou Chef)
 - ♦ Documents (souvent avec LaTeX)

Historique

Cf https://en.wikipedia.org/wiki/List_of_revision_control_software

- ▶ **SCCS** (1972), **RCS** (1982) : local, fichier par fichier
- ▶ **CVS** (1990) :
 - ◆ Gère un ensemble de fichiers, mais chaque fichier a sa version
 - ◆ Client-serveur (*repository* CVS)
 - ◆ Surcouche à RCS
- ▶ **Subversion** (2000) :
 - ◆ Corriger la plupart des défauts de CVS
commits atomiques, versionnement des répertoires
 - ◆ Mais toujours client-serveur

Historique (2)

Modèle **client-serveur limité** :

- ▶ Commits entremêlés si développements simultanés
- ▶ Incite à faire de gros commits pour garder une séparation logique
- ▶ Ou à utiliser des branches (mais mal supporté dans Subversion)

Emergence des VCS (Version Control Systems) **distribués** :

- ▶ BitKeeper (1998), Arch (2001), Darcs (2002), Monotone (2003), Bazaar (2005), Git (2005), Mercurial (2005)

Git est sorti vainqueur :

- ▶ Le plus performant
- ▶ Mais l'un des plus difficiles à maîtriser

Commandes de base

- ▶ Créer un dépôt vide dans le répertoire courant : `git init`
 - ◆ Ou récupérer un dépôt existant :
`git clone url`
- ▶ Demander à git de suivre un fichier (après l'avoir créé) :
`git add fichier`
 - ◆ Enregistrer les modifications faites sur un fichier :
`git commit fichier`
 - ◆ Ou préparer progressivement un commit :
`git add f1 ; git add f2 ; git commit`
(ou `git commit -p`)
 - ◆ Ou committer toutes les modifications en attente :
`git commit -a`
- ▶ Consulter l'état courant du dépôt : `git status`
- ▶ Envoyer des modifications vers le dépôt distant : `git push`
- ▶ Récupérer les modifications du dépôt distant : `git pull`

Commandes de base (suite)

- ▶ Supprimer un fichier : `git rm fichier`
- ▶ Renommer un fichier : `git mv f1 f2`
- ▶ Consulter l'historique des commits : `git log`
- ▶ Consulter l'historique du code : `git blame`
- ▶ Consulter un commit particulier :
`git show a245a4e41db52ed76a30732ca97277fdaaad44e9`
- ▶ Consulter les différences : `git diff ...`

Manipuler les tags, l'historique et les branches

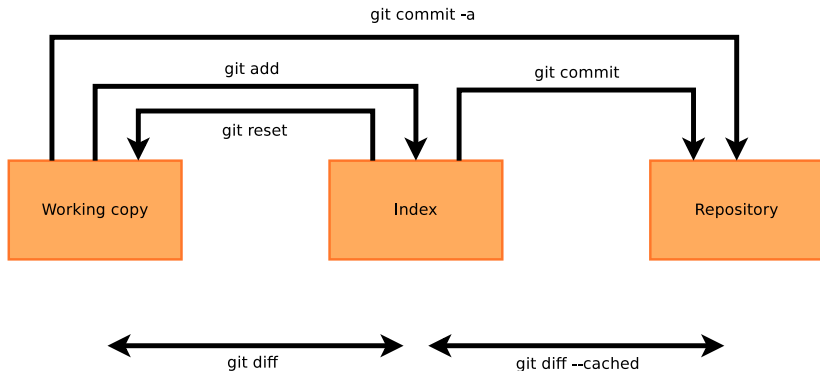
- ▶ `git tag`
- ▶ `git branch`
- ▶ `git rebase`
- ▶ `git checkout`
- ▶ `git reset`
- ▶ `git commit -amend`
- ▶ `git merge`
- ▶ `git fetch`
- ▶ `git bisect`

Commits Git et SHA1

- ▶ Pas d'ordre global sur les commits (\neq Subversion)
- ▶ Chaque commit est identifié par un hash SHA1 de son contenu
f01765d134d897ff373e70c4f1df7610b810392e
- ▶ On peut aussi y faire référence sous une forme plus courte :
git show f017
git show f01765d1

Staging area / Index

- ▶ Git utilise une zone intermédiaire pour préparer les commits, appelée *Index*
- ▶ `git add` permet de préparer un commit progressivement (y compris avec des bouts de fichiers : `git add -p`)

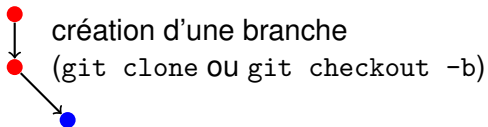


(image : T. Petazzoni, Free Electrons)

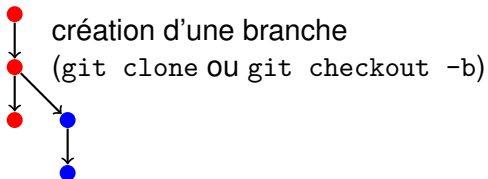
Travailler avec un VCS distribué



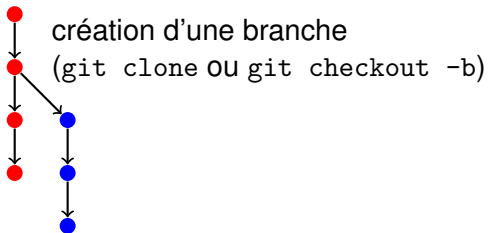
Travailler avec un VCS distribué



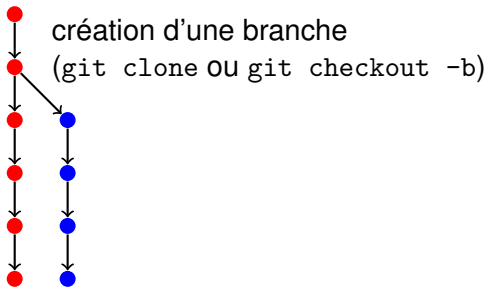
Travailler avec un VCS distribué



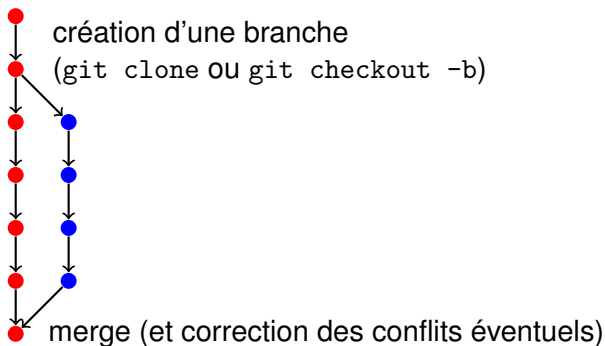
Travailler avec un VCS distribué



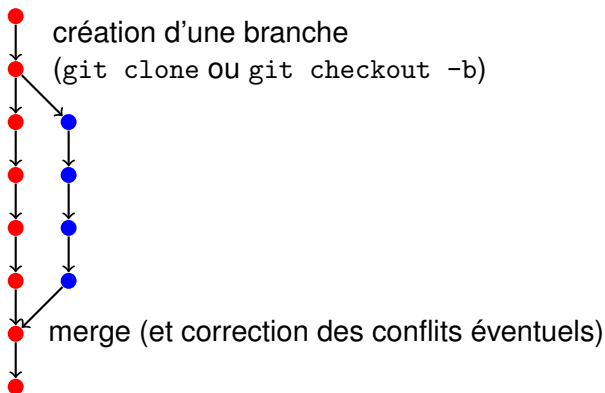
Travailler avec un VCS distribué



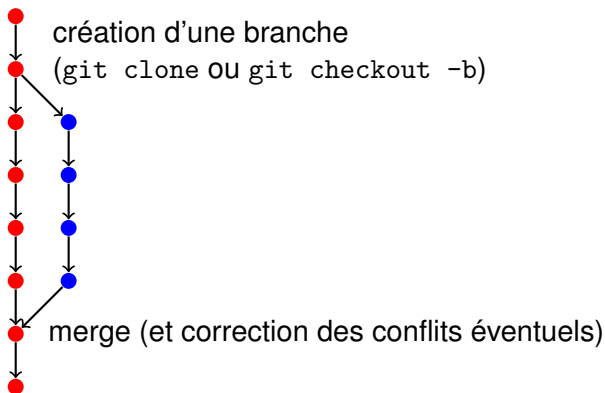
Travailler avec un VCS distribué



Travailler avec un VCS distribué



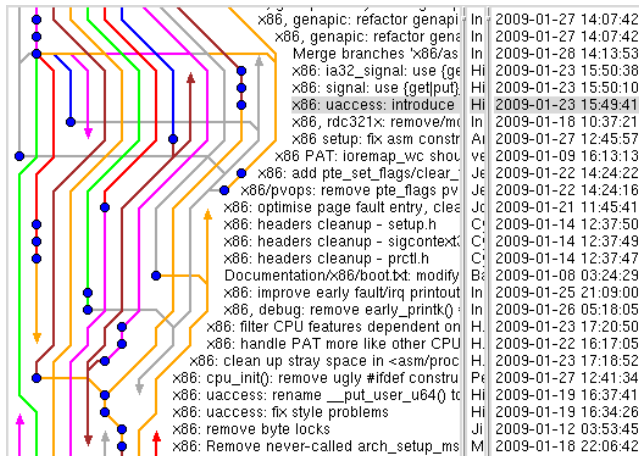
Travailler avec un VCS distribué



Une branche peut-être utilisée :

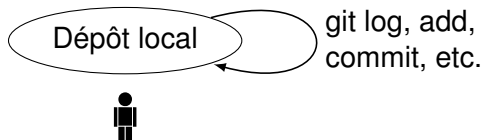
- ▶ Par le même développeur, pour travailler sur une fonctionnalité différente (*feature branch*)
- ▶ Par un autre développeur, pour travailler simultanément

En pratique, de nombreuses branches utilisées



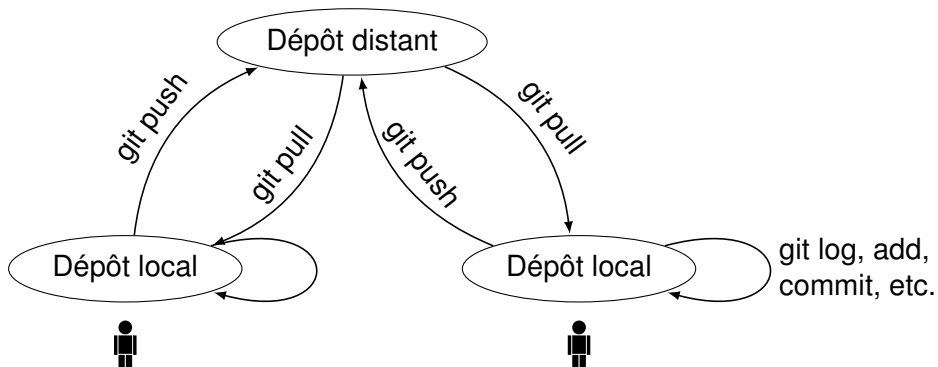
Pas d'organisation ou de workflow imposés (1)

Pas d'organisation ou de workflow imposés (1)



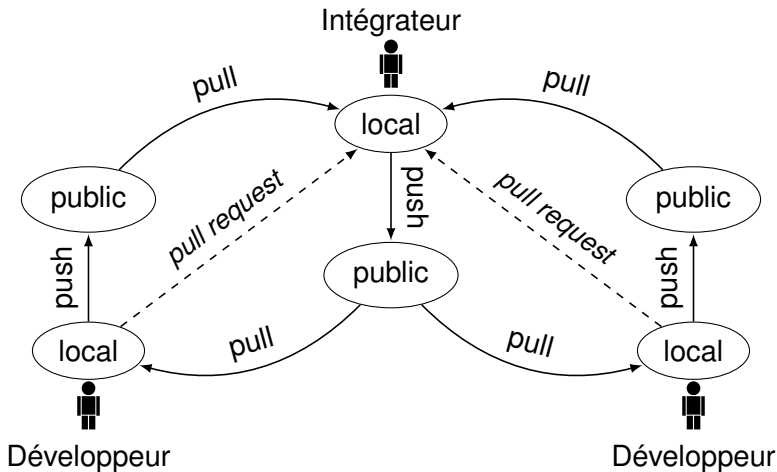
Utilisation locale

Pas d'organisation ou de workflow imposés (2)



Dépôt commun distant
(Gitolite, Redmine, FusionForge, *GitHub*)

Pas d'organisation ou de workflow imposés (3)

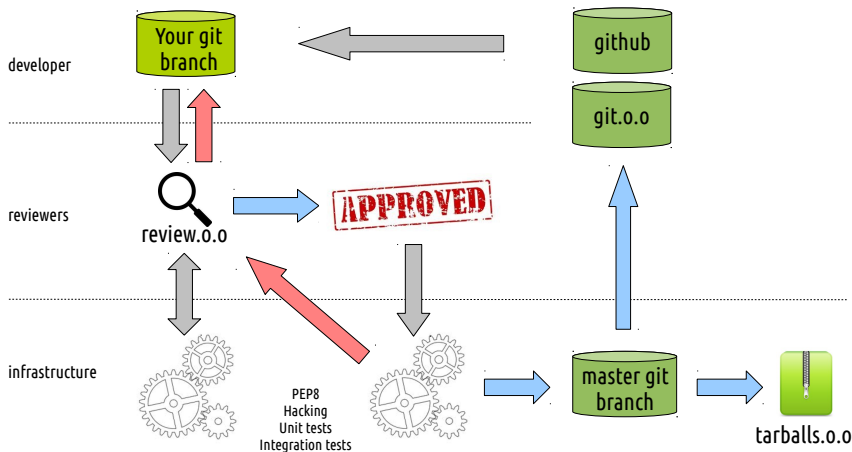


Utilisation distribuée (*GitHub*, *Linux*)

L'écosystème de Git

- ▶ Services d'hébergement : **GitHub**, **BitBucket**, **Gitorious** (libre)
- ▶ Créer son propre hébergement :
 - ◆ **Gitolite** : simple gestionnaire de dépôts git, sans interface web
 - ◆ **GitLab** : interface web, gestion de projets (bugs, etc.),
 - ◆ **FusionForge** : solution intégrée (git, svn, mailing lists, bug tracker, etc.), initialement basé sur SourceForge
<https://sourcesup.cru.fr/>, <http://gforge.inria.fr/>
 - ◆ **Redmine** : solution intégrée, plus simple que FusionForge
- ▶ Visualisation de dépôts sur le web : **gitweb**
- ▶ Outils graphiques :
 - ◆ Visualisation : **tig** (curses), **gitk**, **giggle**
 - ◆ Manipulation : **git cola**
- ▶ Revue de code : **Gerrit** (cf <https://review.openstack.org/>)

Exemple : OpenStack



(figure : Thierry Carrez)

Apprendre Git

De nombreux documents existent (inutile de réinventer la roue) :

- ▶ **Pro Git** : <http://git-scm.com/book/fr> – **livre en français**
Objectif aujourd'hui : survoler le ch 1, faire les exemples du ch 2.
Le ch 3 décrit les branches (quasi-nécessaire pour travailler à plusieurs)
- ▶ Git Magique : tutoriel, traduit en français
(<http://www-cs-students.stanford.edu/~blynn/gitmagic/>)
Analogie avec les jeux vidéos, qui permet de bien expliquer certains concepts difficiles
- ▶ <http://try.github.com/> : essayer git dans son navigateur
- ▶ gittutorial(7)
- ▶ Des centaines de ressources sur le Web