Git

Système de versionnement distribué

Nicolas Dandrimont

Mardi 30 Mars 2010

Sommaire

Qu'est-ce que c'est ?

Git est un système de versionnement...

- distribué
- rapide
- léger
- basé sur les fichiers

Historique

Git a été créé par Linus Torvalds (créateur du Noyau Linux) en avril 2005.
Pourquoi créer un nouveau système ?

- L'ancien système utilisé par le Noyau Linux (BitKeeper) était propriétaire, sous une licence d'utilisation gratuite spéciale pour certains projets libres
- Cette licence d'utilisation a été révoquée à cause d'un supposé reverse-engineering des protocoles sous-jacents
- Linus Torvalds désirait un système rapide et fiable, et aucun des DVCS existant à cette époque (Monotone, Darcs, ...) ne répondait à l'appel

Qui l'utilise

- Le Noyau Linux depuis la version 2.6.12, sortie en Juin 2005 —oui, deux mois après le début du développement—
- X.org
- GNOME
- VLC
- ...
- Le Cr@ns pour certains projets (WiFi, Intranet2, ...)

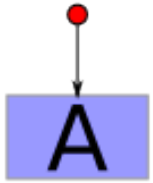
Sommaire

Oui mais, comment ça marche ?

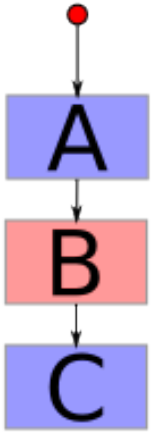
- git, tout comme darcs, stocke les modification sous forme de patches, appliqués les uns à la suite des autres pour former la version courante du dépôt.
- *mais* l'historique d'un dépôt git n'a pas vocation à être linéaire, contrairement à un dépôt darcs

...introduction des branches

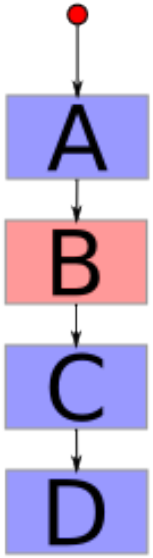
...introduction des branches



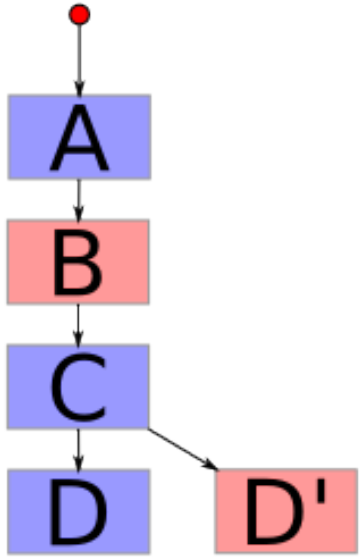
...introduction des branches



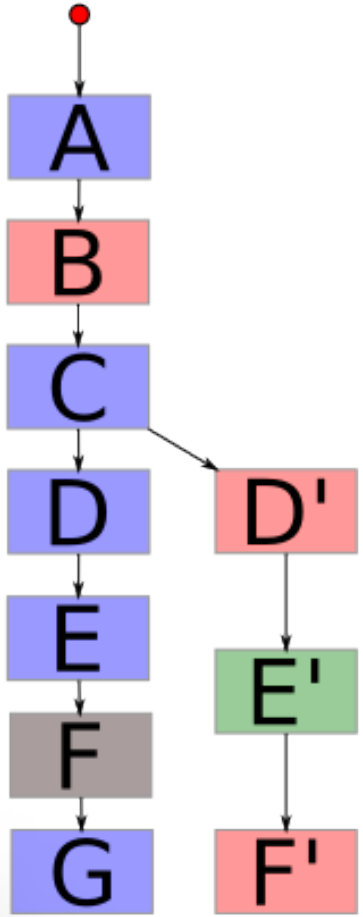
...introduction des branches



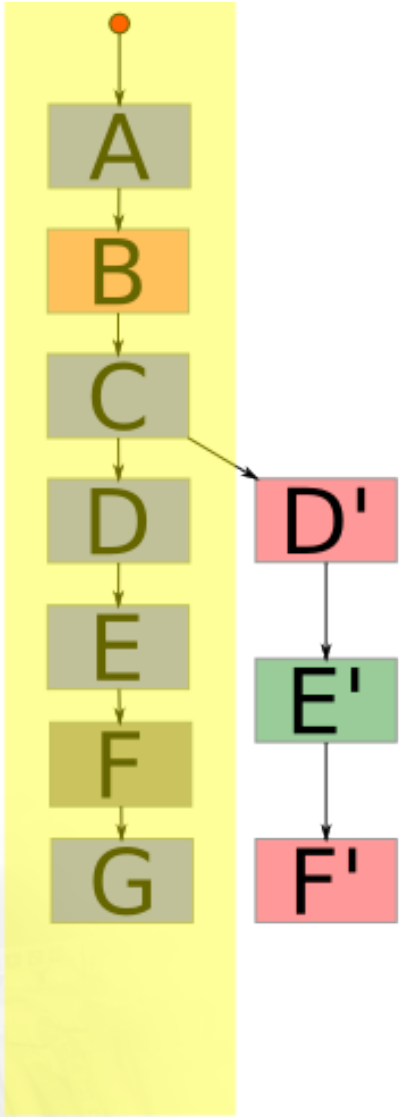
...introduction des branches



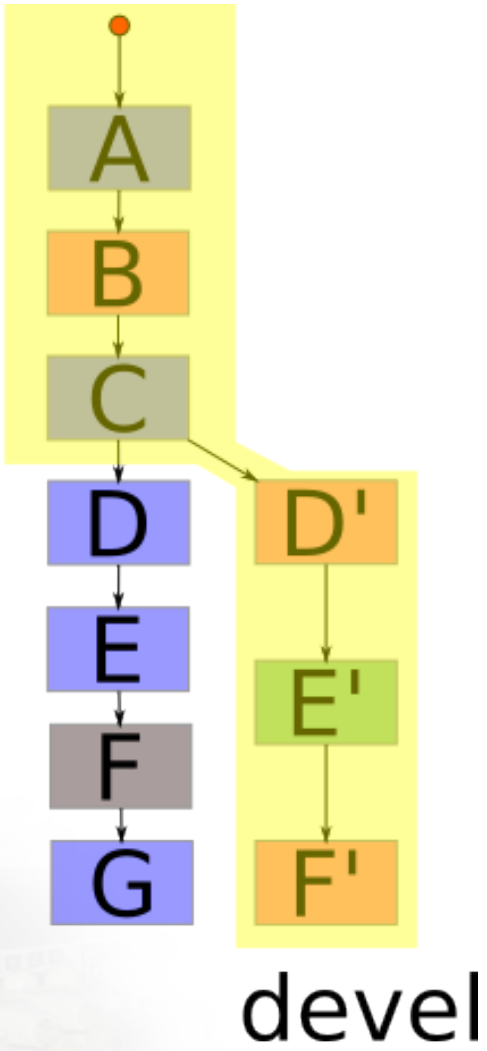
...introduction des branches



...introduction des branches



...introduction des branches

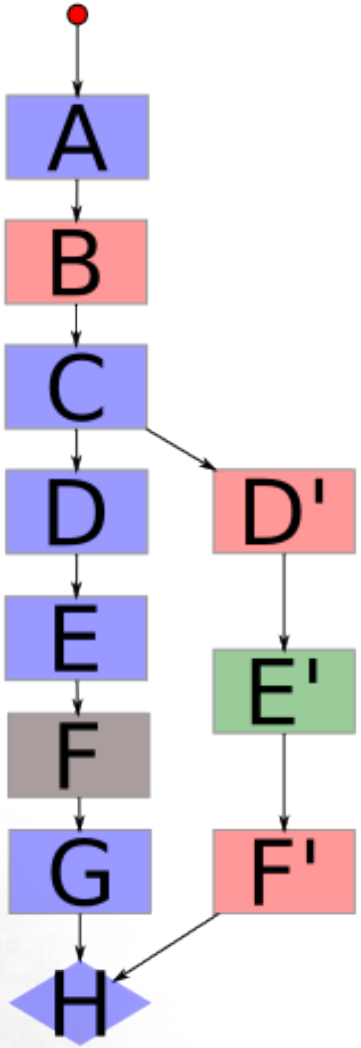


et ensuite ?

On veut profiter du travail dans les deux branches...
Deux solutions

- Fusionner les branches : merge
- Aplatir les branches : rebase

merge



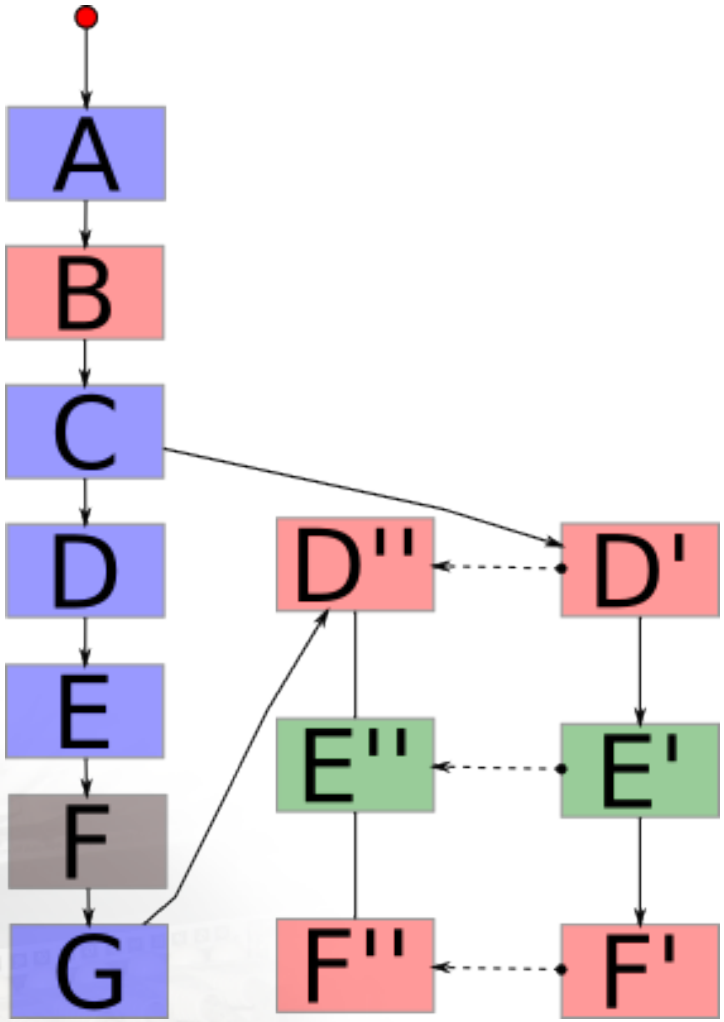
“Commit de merge”

Le commit de merge *H* permet deux choses :

- Faire savoir à git que l’historique a deux branches
- Résoudre les conflits entre les deux branches

Le merge est la solution *propre* et *publique* pour fusionner des branches

Rebase



Rebase

Rebase réécrit l'historique pour aplatir les branches
Un dépôt public ne doit donc pas être rebasé, sous peine de casser l'historique pour tous les autres clients référençant le dépôt.

Cherry-pick

Équivalent de rebase, mais en choisissant uniquement certains commits à réécrire

Sommaire

Commandes utiles : création de dépôt, commit

- `git init` : Initialise un dépôt vide
- `git status` : Affiche le statut de la branche courante (modifications à commiter, en attente, nouveaux fichiers, ...)
- `git add` : Ajoute un fichier et/ou les modifications contenues dans le fichier au prochain commit
- `git add -interactive` : Ajout intéractif de modifications (à la `darcs record`)
- `git commit` : Enregistre les modifications en attente (staged) dans un commit

Commandes utiles : gestion de branches

- `git branch` : Crée une nouvelle branche partant du commit actuel
- `git checkout` : Passe le répertoire de travail d'une branche à l'autre
- `git merge` : Fusionne une branche dans la branche courante
- `git rebase` : Aplatit la branche courante au dessus d'une autre branche
- `git cherry-pick` : Pioche le commit donné et l'ajoute dans la branche courante

Commandes utiles : travail avec des dépôts distants

- Système de `remote`s. On peut suivre les branches de dépôts distants, les comparer à nos branches, les fusionner, ...
- `git clone` : Crée une copie d'une branche d'un dépôt distant
- `git push` : Recopie les modifications de la branche courante dans une branche d'un dépôt distant
- `git fetch` : Récupère les modifications d'une branche distante et les enregistre dans les métadonnées de git
- `git pull` : `git fetch` + `git merge`

La dernière commande est un raccourci. Si c'est possible, il vaut mieux `rebase`r les modifications locales sur les modifications du dépôt externe. Cela évite d'élargir l'historique sans intérêt.

Sommaire

Travail collaboratif

- Deux branches sur le dépôt distant : `master` (branche de production) et `devel` (branche de développement)
- Si nécessaire, plus de branches de développement
- Sur le serveur de production, seulement une copie de la branche `master`, mise à jour à chaque merge depuis `devel`
- Sur le serveur de développement, la branche `master`, la branche `devel`, et une branche `local` permettant d'ajouter des modifications locales (chemins de fichiers divergeant de la production, ...).

Outils

- `gitg` : Visualisation de branches en mode graphique
- `git log` : Historique en mode texte

Pour plus d'info...

- `git help <cmd>` : aide de la commande cmd
- `http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html`
- `http://git-scm.com`
- `http://www.gitready.com/`