



Sistemas de control de versiones distribuidos

Controla las versiones de tu trabajo con GIT

Nacho Álvarez

🐦 @neonigmacdb

✉ neonigma@gmail.com

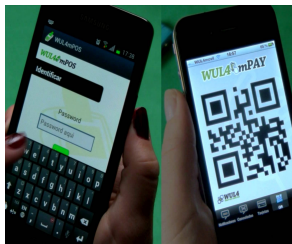


27 de noviembre de 2013

- 1 Acerca de mí
- 2 Definiciones
- 3 ¿Por qué GIT?
- 4 Arquitectura SCV
- 5 Flujo de trabajo en GIT
- 6 Demo
- 7 Gestión de conflictos
- 8 Algo más avanzado
- 9 Enlaces de interés

Who?

- ▶ **Trayectoria profesional:** soporte UCO, desarrollador Web, desarrollador / integrador distribuciones GNU/Linux.
- ▶ **Actualmente:** WUL4 Córdoba (mobile + backend developer)
- ▶ **Involucrado en:**



Definiciones

- ▶ **Control de versiones:** gestión de los diversos cambios que se realizan sobre los elementos de algún producto
- ▶ Una **versión**, **revisión** o **edición** de un producto, es el **estado** en el que se encuentra dicho producto en un **momento** dado de su desarrollo o modificación
- ▶ Los **sistemas** de control de versiones (SCV) vienen a automatizar parcialmente la gestión de este control de cambios
- ▶ Existen SCV **centralizados** (repositorio único remoto) y SCV **distribuidos** (cada usuario su repositorio local + remoto)
- ▶ Los más famosos y de más trayectoria son: SVN, GIT, Mercurial, Bazaar, ClearCase, Perforce...


Ventajas SCV centralizados

- ▶ En los sistemas distribuidos hay un **mayor bloqueo** del estado final del proyecto que en los sistemas centralizados.
- ▶ En los sistemas centralizados las versiones vienen identificadas por un **número de versión**. En lugar de eso cada versión tiene un identificador (hash) al que se le puede asociar una etiqueta (tag).
- ▶ La **curva de aprendizaje** es sensiblemente menor que en los sistemas distribuidos
- ▶ Requiere **menor intervención** del mantenedor

Ventajas SCV distribuidos

- ▶ Necesita **menos operaciones en red** => mayor autonomía y una mayor rapidez.
- ▶ Aunque se **caiga** el **repositorio remoto** la gente puede seguir trabajando
- ▶ Alta probabilidad de **reconstrucción** en caso de falla debido a su arquitectura distribuida
- ▶ Permite mantener **repositorios centrales más limpios**, el mantenedor decide
- ▶ El **servidor remoto** requiere **menos recursos** que los que necesitaría un servidor centralizado ya que gran parte del trabajo lo realizan los **repositorios locales**.
- ▶ Al ser los sistemas distribuidos **más recientes** que los sistemas centralizados, y al tener **más flexibilidad** por tener un repositorio local y otro/s remotos, estos sistemas han sido diseñados para hacer fácil el uso de **ramas locales y remotas** (creación, evolución y fusión) y poder aprovechar al máximo su potencial.

¿Por qué GIT?

A man with sunglasses and a white t-shirt stands on a sandy beach with turquoise water in the background. A large black speech bubble is overlaid on the right side of the image, containing text in Spanish.

**A MI ME SOBRA CON SVN
Y HAGO TODO
LO QUE NECESITO**

¿Por qué GIT?



PostgreSQL



En números



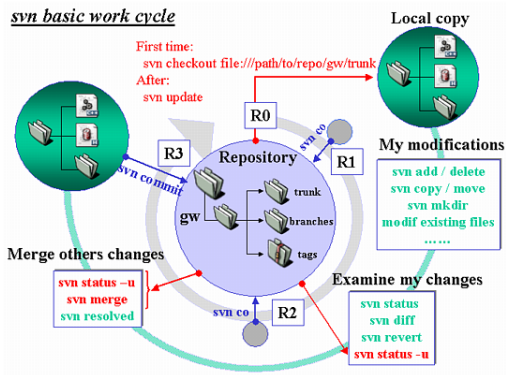
- ▶ Enfocado a **código privado** (Mailchimp, Opera...)
- ▶ Más de **1 millón** de usuarios registrados
- ▶ Integración del resto del ecosistema software: Bamboo (CI), Confluence (Doc), Jira (project tracking), SourceTree (GUI)...
- ▶ Se cobra por **número de integrantes de equipo**: 0-5 (gratis), 6-10 (\$10 month), 11-25 (\$25 month), 26-50 (\$50 month)...



- ▶ Enfocado a **código abierto** (bootstrap, nodejs, jquery...)
- ▶ Más de **4 millones** de usuarios registrados y **8 millones** de **repositorios** creados
- ▶ Se cobra por **repositorios privados**: 5 (\$7 month), 10 (\$12 month), 20 (\$22 month), 50 (\$50 month)

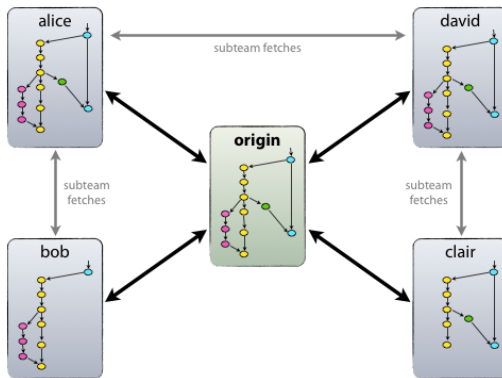
Arquitectura SCV centralizado

- Todo el mundo actualiza un mismo repositorio central remoto

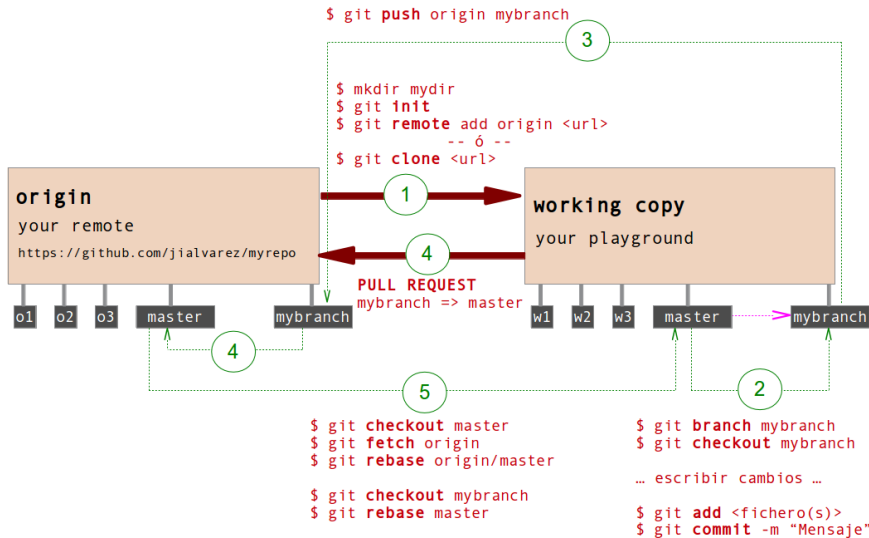


Arquitectura SCV distribuido

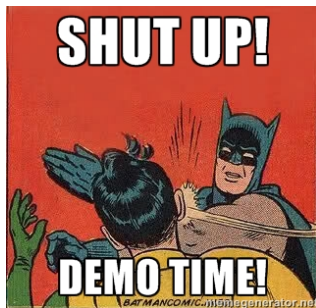
- ▶ Todo el mundo mantiene su copia del proyecto
- ▶ Cada integrante del equipo trabaja en sus propias funcionalidades en su repositorio local particular
- ▶ El mantenedor del repositorio acepta o no las modificaciones de los integrantes



Flujo de trabajo en GIT



Demo



- ▶ Crearemos nuestro propio repositorio Git en Bitbucket
- ▶ Crearemos nuestra rama de trabajo y subiremos cambios
- ▶ Propondremos la integración de estos cambios haciendo *pull request*
- ▶ Simularemos la descarga del repositorio por parte de un compañero y subiremos nuevos cambios
- ▶ Actualizaremos el repo haciendo *rebase* y subiremos más cambios
- ▶ Mostraremos cómo funciona el *rebase* de manera gráfica

Gestión de conflictos

Algo más avanzado

- ▶ squash
- ▶ cherry-pick

Enlaces de interés

- ▶ <http://nudowdeployer.wordpress.com/2013/07/23/github-vs-bitbucket-2/#tldr>
- ▶ <http://www.infoworld.com/d/application-development/bitbucket-vs-github-which-project-host-has-the-most-22706>
- ▶ <http://pcottle.github.io/learnGitBranching/>