

git: Control de versiones estilo Linus

Horst H. von Brand

Departamento de Informática
Universidad Técnica Federico Santa María

14 de octubre de 2010



Visión general

Control de versiones

Requisitos para Linux

Plomería y porcelanas

Comparaciones con otros SCM

Comandos de git

Referencias para git

Definición de *SCM*

- ▶ *Software Configuration Management* (SCM) es el conjunto de actividades relacionadas con la administración de versiones de software
- ▶ Muchos productos se encuentran en versiones diferentes en diversos sitios. Desarrolladores tienen versiones en prueba.
- ▶ Es una actividad vital en el desarrollo de todo proyecto.

Modelos de *SCM*

- Centralizado:** Hay un repositorio central, todos los cambios se registran allí. Los desarrolladores sacan una versión del repositorio, la modifican, posiblemente se sincronizan con otros cambios que hayan ocurrido en el intertanto y luego envían sus modificaciones de vuelta
- Distribuido:** Cada desarrollador tiene un repositorio propio. Maneja sus propias modificaciones, intercambia parches con los demás.

Modelos de *SCM*

Un sistema distribuido puede manejarse en forma centralizada (a través de designar un repositorio como “oficial”), en la práctica no se usan *realmente* todos contra todos. En cambio, un sistema centralizado fuerza a trabajo sincronizado a través de él.

Al ser más livianos de usar, los sistemas distribuidos promueven mejor separación de los cambios.

Los sistemas distribuidos son cómodos para uso personal al no requerir configuración y acceso a un repositorio central.

Características de *SCM* distribuido

En un sistema distribuido es esencial manejar bien ramas y *merge* para integrar líneas de desarrollo externas. Esto permite también manejar ramas privadas (experimentos, búsqueda de alternativas de solución de problemas, situaciones particulares) en forma cómoda, aprovechando control de versiones.

No hay un punto de falla único al no haber repositorio central. Simplifican el manejo de respaldos (basta tener una réplica del repositorio), protección contra condiciones de carrera y atomicidad de transacciones las provee el sistema.

El modelo de desarrollo de Linux

- ▶ El núcleo es grande (2.6.30 es 11 560 971 líneas) y complejo
- ▶ Muchos desarrolladores independientes (entre 2.6.11 y 2.6.30 participaron 4 910, alrededor de 1/3 con un único parche)
- ▶ El tráfico de parches es **gigantesco** (11 989 parches en los 78 días entre 2.6.29 y 2.6.30)

Linux es marca registrada de Linus Torvalds

El modelo de desarrollo de Linux

- ▶ Los **desarrolladores** se organizan en una jerarquía informal: Linus a la cabeza, lugartenientes de Linus, responsables por grandes subsistemas, ...
- ▶ Hay múltiples ramas: Versiones experimentales para arquitecturas específicas, versiones estables (base de distribuciones “enterprise”), áreas completas como manejo de redes o desarrollo de WiFi, drivers individuales, colecciones de parches en prueba, ...

Requisitos de SCM para Linux

- ▶ Sistema distribuido (son muchas líneas de desarrollo independientes)
- ▶ Gran rendimiento (ingresar decenas de parches por minuto al núcleo)
- ▶ No dar una posición especial al “dueño del repositorio maestro”

GIT – The stupid content tracker

“git” can mean anything, depending on your mood:

- ▶ Random three-letter combination that is pronounceable, and not actually used by any common Unix command. The fact that it is a mispronunciation of “get” may or may not be relevant
- ▶ Stupid. Contemptible and despicable. Simple. Take your pick from the dictionary of slang.
- ▶ “Global information tracker”: You’re in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- ▶ “Goddamn idiotic truckload of sh*t”: When it breaks

GIT – The stupid content tracker

*I'm an egotistical bastard, so I name all my projects after myself.
First Linux, now git.*

– Linus Torvalds

(En el slang británico, git es “pig headed, think they are always correct, argumentative”)

Plomería

En el diseño original los comandos de `git` son la plomería con la cual se puede construir un sistema de control de versiones que funciona.



Porcelana

Sobre la plomería se instalan sistemas más estéticos, fáciles de usar, hechos de porcelana.



Estado actual de git

- ▶ git ha avanzado mucho en usabilidad. El paquete oficial trae algunas porcelanas (no tan) básicas.
- ▶ La plomería aún existe como comandos aparte, pero hoy son poco más que llamadas a una biblioteca. Pueden combinarse para crear nuevos comandos, o migrar a C si resulta importante el rendimiento.
- ▶ En 1.7.3 son 159 comandos, pero sólo una veintena es para consumo humano. De éstos, se usa comúnmente una media docena.

Diferencias de git con otros SCM

- ▶ git no guarda las diferencias entre versiones sucesivas, guarda los contenidos mismos. Cuando se requieren diferencias, éstas se reconstruyen. Nótese que así no se amarra a una manera particular de determinar “diferencias.” Obtener la diferencia entre dos ramas o dos versiones sucesivas es lo mismo. . .
- ▶ No hay “relación entre repositorios,” cualquiera puede tomar libremente de los demás. No hay *nada* que haga especial el “repositorio original.”
- ▶ La identificación de un objeto en git (como un commit) es simplemente un string hexadecimal, sin significado global. ¡Dos commit con el mismo contenido pueden perfectamente tener identificaciones diferentes!

Formato general

Los comandos se invocan por ejemplo mediante `git help`, la página de manual de éste está bajo `git-help(1)`. Este comando, en particular, entrega la página de manual para el comando `git` indicado como argumento. El mismo efecto se obtiene con el flag `--help` al comando.

El formato general es:

```
git <flags> <comando> <argumentos>
```

O sea, por ejemplo:

```
git --paginate diff --color HEAD~4..HEAD
```

Formas de referenciar *commits*

SHA1: 6a06307a4fa68ce026fd18ef8ef6d9db67a98085,
6a0630

Rótulo, rama: v1.0, master, HEAD

Relativo: HEAD^, HEAD~5, v1.0~3, local@{17}, @{5}

Fecha: @{yesterday},
master@{3 days 1 hour 1 second ago},
experiment@{2007-02-26 18:30:00}

Favorito personal: `git diff @{yesterday} Documentation`

Rangos

Pueden indicarse rangos mediante “..”, o sea por ejemplo:

```
git diff v1.0..v1.3
```

da la diferencia entre la versión 1.0 y la 1.3, suponiendo esos rótulos y desarrollo lineal. Muy útil es

```
git diff HEAD^..HEAD
```

La situación es más compleja, eso sí: El “rango” uno..otro es todo lo que puede alcanzarse desde otro, excluyendo lo que puede alcanzarse desde uno. Si uno es un ancestro de otro, realmente es un rango.

También está la forma con tres puntos: En git uno...otro es todo lo que es alcanzable desde uno u otro, pero no ambos. Muestra cómo divergieron dos ramas independientes.

Comandos de git

- ▶ `git init`: Crea un repositorio
- ▶ `git config`: Manipula la configuración
- ▶ `git add`: Pone archivos bajo control de versiones
- ▶ `git rm`: Elimina archivos del control de versiones
- ▶ `git mv`: Similar a `mv(1)`, `-f` fuerza sobrescribir destinos
- ▶ `git diff`: Muestra diferencias varias (entre el directorio y lo almacenado, entre versiones, entre ramas, ...)
- ▶ `git grep`: Busca un patrón en el repositorio
- ▶ `git commit`: Registra los cambios, `-a` todos los cambios, `-m` comentario, `-s` firma

Comandos locales de git

- ▶ `git tag`: Crea y manipula *tags*
- ▶ `git checkout`: Va a otra rama (con `-b` la crea)
- ▶ `git branch`: Manipula ramas (crearlas, eliminarlas, ...)
- ▶ `git merge`: Integra cambios desde otra rama
- ▶ `git rebase`: Trasplanta cambios a la punta de la rama
- ▶ `git cherry-pick`: Elige cambios individuales de otra rama
- ▶ `git log`: Muestra registro de cambios, en una variedad de formatos y niveles de detalle

Comandos git para uso distribuido

- ▶ `git clone`: Copia un repositorio remoto (o local)
- ▶ `git pull`: Trae cambios remotos
- ▶ `git push`: Envía cambios
- ▶ `git rebase`: Aplica cambios “debajo” de los locales
- ▶ `git cherry`: Compara cambios locales con remotos para ver si fueron aplicados

Comandos porcelanosos de git

- ▶ `git format-patch`: Crea una secuencia de parches
- ▶ `git send-email`: Envía parches creados por el anterior por correo electrónico
- ▶ `git am`: Aplica de una vez la colección de parches en un *mbox*
- ▶ `git archive`: Exporta una versión particular como archivo `tar(1)` o `zip(1)`
- ▶ `git whatchanged`: Muestra cambios
- ▶ `gitk`: Interfaz gráfica para ver la historia
- ▶ `git gui`: Interfaz gráfica general

Comandos porcelanosos de git

- ▶ `git pickaxe`: Herramienta para arqueología
- ▶ `git bisect`: Dadas una versión “buena” y una “mala,” extrae el punto medio para prueba (búsqueda binaria del cambio culpable)

Manipular la historia

git permite manipular la historia, editando o reordenando *commits*, “transplantándolos” de un punto a otro, corrigiendo comentarios, aplicando modificaciones parcialmente, ...

Estas operaciones son indispensables para crear una historia limpia, pero deben usarse con mucho cuidado. La recomendación básica es usarlas sólo en áreas que no se han compartido, ya que fácilmente dejan varado a quien trata de sincronizarse luego.

Un ejemplillo

Hacer un mini-ejemplo práctico
acá

Comentarios finales sobre git

Algunos de los comandos en realidad tienen una variedad de funciones similares, véanse las páginas de manual. El paquete git contiene un excelente **tutorial**.

El paquete incluye herramientas para importar desde una variedad de otros sistemas (CVS, SVN, Perforce, arch, ...), y hay cómo importar una secuencia de tarballs. Está documentado cómo generar comandos para importar contenidos a un repositorio (`git-fast-import(1)`).

Incluso puede emular un servidor CVS...

Referencias para git

git es el sistema base. Incluye herramientas para migrar desde otros sistemas y un extenso tutorial. (Particularmente relevante es “**Git for CVS Users**”, que viene también como parte del paquete mismo. No sólo es relevante para refugiados de CVS sino también para damnificados por SVN, o sistemas centralizados en general.) Hay un buen **resumen** de lo que hay alrededor de git, incluyendo tutoriales y referencias a charlas y discusiones, y tanto herramientas para usar con git como herramientas que usan git para tareas como respaldar configuraciones. El **Wiki** oficial de git probablemente sea el recurso más actualizado. Tiene documentación, el **FAQ** obligatorio, referencia sitios que ofrecen *githosting*, ...

Referencias para git

El libro [Pro Git](#) es una muy buena introducción, va más allá del camino trillado en algunos puntos. En [gitmagic](#) hay un montón de “paltas.”

Una buena [presentación](#) sobre git la escribió Bart Trojanowski. Para entender las estructuras detrás, hay una [introducción para ciencias de computación](#) de Tommi Virtanen (o Tv, como le gusta que lo llamen).

Jeff Garzik da [una perspectiva](#) de un desarrollador activo en el núcleo Linux.

*And then realize that nothing is perfect. Git is just *closer* to perfect than any other SCM out there*

– Linus Torvalds

¿Preguntas?