Introduccion a Git

Contents

[1. Que es Git? 2](#_Toc22164766)

[a. Que es un Controlador de Versiones? 2](#_Toc22164767)

[b. Git vs otros VCS 4](#_Toc22164768)

[c. Git. Estados 5](#_Toc22164769)

[2. Configurando Git 6](#_Toc22164770)

[a. Instalación 6](#_Toc22164771)

[b. Configuración 6](#_Toc22164772)

[3. Comandos Básicos 7](#_Toc22164773)

[a. Git init 7](#_Toc22164774)

[b. Git clone 7](#_Toc22164775)

[c. Git status 7](#_Toc22164776)

[d. Git add 7](#_Toc22164777)

[e. Git Commit 7](#_Toc22164778)

[f. Git Push 7](#_Toc22164779)

[g. Git pull 7](#_Toc22164780)

[4. Trabajando con ramas 7](#_Toc22164781)

[a. Que es una rama o branch? 7](#_Toc22164782)

[b. Git branch 7](#_Toc22164783)

[c. Git checkout 7](#_Toc22164784)

[d. Checkout o branch? 7](#_Toc22164785)

# Que es Git?

## Que es un Controlador de Versiones?

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Si eres Desarrollador Web, o un QA Automation, y quieres mantener cada versión de tu código, un sistema de control de versiones (Version Control System o VCS en inglés) es una elección muy buena. Te permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más.

Sistemas de Control de Versiones Locales

Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos.



Sistemas de Control de Versiones Centralizados

El siguiente gran problema que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés). Estos sistemas, como CVS, Subversion, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Durante muchos años éste ha sido el estándar para el control de versiones.



Esta configuración ofrece muchas ventajas. Por ejemplo, todo el mundo puede saber en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando. Otra desventaja, en el caso de que el disco duro donde reside la base de datos sufriera una falla, y no existan copias de seguridad, se perderían todos los datos.

Sistemas de Control de Versiones Distribuidos

Los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés), como Git, Mercurial, Bazaar o Darcs, los clientes no sólo descargan la última instantánea de los archivos, replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los dato.



## Git vs otros VCS

La principal diferencia entre Git y cualquier otro VCS (Subversion y compañía incluidos) es cómo Git modela sus datos. Conceptualmente, la mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.) modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo.

En cambio, Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar. Por lo general no se necesita información de ningún otro ordenador de tu red. Por ejemplo, para navegar por la historia del proyecto, Git no necesita salir al servidor para obtener la historia y mostrártela, simplemente la lee directamente de tu base de datos local. Esto significa que ves la historia del proyecto casi al instante.

## Git. Estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged).

Confirmado significa que los datos están almacenados de manera segura en tu base de datos local. Modificado significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos. Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

El área de preparación es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.

El flujo de trabajo básico en Git es algo así:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified). En el Capítulo 2 aprenderás más acerca de estos estados, y de cómo puedes aprovecharte de ellos o saltarte toda la parte de preparación.

# Configurando Git

## Instalación

Instalar Git en Windows es muy fácil. El proyecto msysGit tiene uno de los procesos de instalación más sencillos. Simplemente descarga el archivo exe del instalador desde la página de GitHub, y ejecútalo:

* <http://git-scm.com/download>
* <http://msysgit.github.com/>

Una vez instalado, tendrás tanto la versión de línea de comandos (incluido un cliente SSH) como la interfaz gráfica de usuario estándar.

## Configuración

Git trae una herramienta llamada ***git config***, que te permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git.

En sistemas Windows, Git busca el archivo ***.gitconfig*** en el directorio ***$HOME*** (para mucha gente será (C:\Users\$USER). También busca el archivo ***/etc/gitconfig***, aunque esta ruta es relativa a la raíz **MSys**, que es donde decidiste instalar Git en tu sistema Windows cuando ejecutaste el instalador.

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "***commits***" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

De nuevo, sólo necesitas hacer esto una vez si especificas la opción --global, ya que Git siempre usará esta información para todo lo que hagas en ese sistema. Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando sin la opción --global cuando estés en ese proyecto.

Si quieres comprobar tu configuración, puedes usar el comando git config --list para mostrar todas las propiedades que Git ha configurado:

$ git config --list

user.name=John Doe

user.email=johndoe@example.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

# Comandos Básicos

## Git init

## Git clone

## Git status

## Git add

## Git Commit

## Git Push

## Git pull

# Trabajando con ramas

## Que es una rama o branch?

## Git branch

## Git checkout

## Checkout o branch?

Fuente: https://git-scm.com/book/es/v1