¿Qué es Subversion?

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Ésto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo ”.

¿Qué es un sistema de control de versiones?

Un sistema de control de versiones (CVS) te permite realizar un seguimiento de la historia de una colección de archivos y además incluye la funcionalidad de revertir la colección de archivos actual hacia una versión anterior.

¿Qué es un sistema de control de versiones distribuida?

En un sistema de control de versiones distribuida hay un servidor central para almacenar el repositorio y cada usuario puede hacer una copia completa del repositorio central mediante un proceso llamado “clonación”. Cada repositorio clonado es una copia completa del repositorio central y por ser una copia completa posee la mismas funcionalidades que el repositorio original, es decir, contiene la historia completa de la colección de archivos.

Cada repositorio clonado puede a su vez intercambiar las versiones de sus archivos con otros repositorios clonados del mismo nodo padre (ya que deben tener la misma estructura), enviando sus cambios y recibiendo los del otro en forma directa o a través del repositorio central.

GIT

Git es un sistema de control de versiones distribuído (scvd) escrito en C. Un sistema de control de versiones permite la creación de una historia para una colección de archivos e incluye la funcionalidad para revertir la colección de archivos a otro estado. Otro estado puede significar a otra colección diferente de archivos o contenido diferente de los archivos.

Git es un sistema de control de versiones distribuida que se origina a partir del desarrollo del kernel de Linux y es usado por muchos proyectos populares Open Source como ser Android o Eclipse, así como tantos otros proyectos comerciales.

Entonces, la principal diferencia entre Git y cualquier otro sistema de control de versiones 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 modelan la información que almacenan como un

conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo argo 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 estado 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.

A description...

**El repositorio local**

Luego de clonar o crear un repositorio el usuario tiene una copia completa del repositorio, y puede realizar operaciones de control de versiones contra este repositorio local, como por ejemplo crear nuevas versiones, revertir cambios, etc. El flujo de trabajo básico en Git es algo así:

* Modificas una serie de archivos en tu directorio de trabajo (working directory).
* Añadís instantáneas de los archivos a tu área de preparación (staging area).
* Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esa instantánea de manera permanente en tu directorio de Git (git directory).

A description...

Staging area: Es el lugar en el que se almacenan los cambios del working tree previos al commit. Es decir, contiene el set de cambios relevantes para el próximo commit.

Index: Es un término alternativo para referirnos al staging area.

**Repositorios remotos**

Git permite a los usuarios sincronizar el repositorio local con otros repositorios remotos al ambiente de trabajo local. Los usuarios que posean los permisos necesarios pueden hacer un “push” (comando utilizado para subir cambios locales) de los cambios locales al repositorio remoto. A su vez también pueden hacer un “fetch” o “pull” (comandos para buscar cambios) de los cambios realizados en repositorios remotos al local.

**Comandos básicos en Git**

Para hacer cualquier cosa con Git, primero hay que tener un repositorio creado, que es donde Git almacena los cambios que haces al código. Hay dos maneras de obtener un repositorio Git.

Una forma es simplemente inicializar uno nuevo desde un carpeta existente, como un nuevo proyecto. La segunda forma consiste en clonar un repositorio Git público, como si quisieras una copia o quisieras trabajar con alguien en un proyecto.

**Git init - inicializar una carpeta como un repositorio Git**

Para crear un repositorio en una carpeta existente de archivos, podes ejecutar el comando git init en esa carpeta. Por ejemplo, digamos que tenemos una carpeta con algunos archivos adentro, así:

$ cd repositorio\_nuevo

$ ls

README hello.rb

Para iniciar el control de versión con Git, podemos ejecutar git init.

$ git init

Initialized empty Git repository in /repositorio\_nuevo/.git/

Ahora se puede ver que hay una subcarpeta oculta llamada “.git” en el proyecto. Este es tu repositorio donde se almacenan todos los cambios del proyecto.

$ ls -a

. .. .git

README

hello.rb

En pocas palabras, se usa “git init” para convertir una carpeta existente en un nuevo repositorio Git. Se puede hacer esto en cualquier carpeta en cualquier momento.

Nota: Cuando se crea un repositorio con init, la primera vez que haces un push, tenes que correr

git push origin master.

Además hay que crear el branch master y hacerle un commit con los archivos agregados para que aparezca.

**Git clone - copiar un repositorio Git**

Si tenes que colaborar con alguien en un proyecto, o si deseas obtener una copia de un proyecto para poder ver o usar el código, debes clonarlo. Para lograrlo sólo tenes que ejecutar el comando git clone [url] con la URL del proyecto que deseas copiar.

$ git clone git://github.com/schacon/simplegit.git

Por defecto, Git va a crear una carpeta que tiene el mismo nombre que el proyecto en la dirección que le indiques - básicamente cualquier cosa que aparezca después de la última barra de la URL. Si queres un nombre diferente, podes ponerlo al final del comando, después de la URL. Entonces, se utiliza el

comando git clone para obtener una copia local de un repositorio Git, para que puedas verlo y empezar a modificarlo.

**Agregando y subiendo cambios**

Un concepto que no se debe perder de vista es que Git tiene un índice (index), que funciona como una especie de área de desarrollo (staging area) para los cambios que uno va generando en los archivos a medida que va trabajando.

git add - agregar los contenidos de archivos al staging area

En Git tenes que agregar previamente los cambios realizados al staging area para luego poder hacer el commit correspondiente (confirmar los cambios). Si el archivo que estás agregando es nuevo, entonces tenes que correr el comando git add para añadirlo inicialmente en tu staging area. Si el archivo ya está en “seguimiento” también tenes que correr el mismo comando (git add), no para agregar el archivo, sino para agregar las nuevas modificaciones en tu staging area.

$ git status -s

$ git add README hello.rb

Es también común agregar en forma recursiva todos los archivos en un nuevo proyecto especificando sólo el directorio de trabajo que se desea agregar, como por ejemplo:

$ git add .

Para el caso también “git add \*” hubiese hecho lo mismo, además de meterse en subdirectorios que estuviesen dentro del directorio actual.

El estado ‘AM’ significa que el archivo fue modificado en el disco desde que lo agregamos.

git diff HEAD – muestra las diferencias de todos los cambios, estén o no en el staging

git diff –stat – muestra una lista de cambios en lugar de un diff completo por archivo

git diff – muestra diferencias entre el stage y lo modificado que no esta en el stage

git commit – graba una instantánea del staging area

Una vez que agregaste los cambios al staging area, vas a necesitar confirmar esos cambios con el comando add, pero para poder hacerlo git necesita grabar tus datos personales para poder hacer un seguimiento de los cambios. Entonces, el primer paso es configurar estos datos:

$ git config --global user.name 'Tu nombre'

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

$ git add hello.rb

$ git status -s

M hello.rb

$ git commit -m 'mis cambios al archivo hola mundo'

[master 68aa034] my hola mundo changes

1 files changed, 2 insertions(+), 1 deletions(-

git commit –a – agrega automáticamente en el staging area los cambios de archivos que están en seguimiento .

$ git commit -am 'changes to hello file'

[master 78b2670] changes to hello file

1 files changed, 2 insertions(+), 1 deletions(-)

git reset – deshace cambios y commits

git rm - remueve archivos del staging area

echo "# aop\_execute" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/lovdav/aop\_execute.git

git push -u origin master

**Hacer checkout a un repositorio**

Crea una copia local del repositorio ejecutando

$git clone /path/to/repository

Si utilizas un servidor remoto, ejecuta

$git clone [username@host](mailto:username@host):/path/to/repository

**Flujo de trabajo**

Tu repositorio local esta compuesto por tres "árboles" administrados por git. El primero es tu Directorio de trabajo que contiene los archivos, el segundo es el Index que actúa como una zona intermedia, y el último es el HEAD que apunta al último commit realizado.

A description...

Puedes registrar cambios (añadirlos al Index) usando

$git add <filename>

$git add .

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa

$git commit -m "Commit message"

Ahora el archivo esta incluído en el HEAD, pero aún no en tu repositorio remoto.

**Envío de cambios**

Tus cambios están ahora en el HEAD de tu copia local. Para enviar estos cambios a tu repositorio remoto ejecuta

$git push origin master

*Reemplaza master por la rama a la que quieres enviar tus cambios.*

Si no has clonado un repositorio ya existente y quieres conectar tu repositorio local a un repositorio remoto, usa:

$git remote add origin <server>

Ahora podrás subir tus cambios al repositorio remoto seleccionado.

**Ramas**

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama master es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.

A description...

Crea una nueva rama llamada "feature\_x" y cámbiate a ella usando

$git checkout -b feature\_x

vuelve a la rama principal

$git checkout master

y borra la rama

$git branch -d feature\_x

Una rama nueva no estará disponible para los demás a menos que subas (push) la rama a tu repositorio remoto

$git push origin <branch>

**actualiza & fusiona**

Para actualizar tu repositorio local al commit más nuevo, ejecuta

$git pull

En tu directorio de trabajo para bajar y fusionar los cambios remotos. Para fusionar otra rama a tu rama activa (por ejemplo master), utiliza

$git merge <branch>

En ambos casos git intentará fusionar automáticamente los cambios. Desafortunadamente, no siempre será posible y se podrán producir conflictos. Tú eres responsable de fusionar esos conflictos manualmente al editar los archivos mostrados por git. Después de modificarlos, necesitas marcarlos como fusionados con

$git add <filename>

Antes de fusionar los cambios, puedes revisarlos usando

$git diff <source\_branch> <target\_branch>

**Etiquetas**

Se recomienda crear etiquetas para cada nueva versión publicada de un software. Este concepto no es nuevo, ya que estaba disponible en SVN. Puedes crear una nueva etiqueta llamada 1.0.0 ejecutando

$git tag 1.0.0 1b2e1d63ff

1b2e1d63ff se refiere a los 10 caracteres del commit id al cual quieres referirte con tu etiqueta. Puedes obtener el commit id con

$git log

También puedes usar menos caracteres que el commit id, pero debe ser un valor único.

**Reemplaza cambios locales**

En caso de que hagas algo mal (lo que seguramente nunca suceda ;) puedes reemplazar cambios locales usando el comando

$git checkout -- <filename>

Este comando reemplaza los cambios en tu directorio de trabajo con el último contenido de HEAD. Los cambios que ya han sido agregados al Index, así como también los nuevos archivos, se mantendrán sin cambio.

Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

$git fetch origin

$git reset --hard origin/master

**Datos útiles**

Interfaz gráfica por defecto

$gitk

Colores especiales para la consola

$git config color.ui true

Mostrar sólo una línea por cada commit en la traza

$git config format.pretty oneline

Agregar archivos de forma interactiva

$git add -i

**Ignorar ciertos archivos**

Git puede ser configurado para que ignore ciertos archivos y directorios. Esta configuración es realizada mediante el archivo .gitignore. Este archivo puede estar en cualquier directorio y puede contener patrones para archivos. Por ejemplo, es posible decirle a git que ignore el directorio bin y todos los archivos que finalicen con la extensión pyc (archivos de python compilados) mediante el siguiente patrón en el archivo .gitignore en el directorio principal.

**Github y Bitbucket: servicios de Git en la nube**

GitHub y Bitbucket son dos servicios que te permiten administrar tus proyectos usando el popular sistema de control de versiones Git, pero en la nube.

Los sistemas de control de versiones son los mejores amigos de un desarrollador, ofrecen un respaldo permanente de todas las fases de desarrollo de sus proyectos. Por varias razones Git es uno de los más populares, pero ¿qué pasa cuando no tienes el hardware, el conocimiento o el tiempo para instalar y mantener tu propio servidor de Git?

GitHub y Bitbucket son dos servicios que te permiten administrar tus proyectos usando Git pero en la nube. Estos servicios representan una ventaja para muchos pues eliminan la necesidad de administrar un servidor local y otorgan la tranquilidad de contar con el respaldo de su información, siempre accesible a través de la red.

**Github**

Github es un servicio de alojamiento basado en la web para proyectos de desarrollo de software que utilizan el sistema de control de versiones Git. Fue lanzado en el año 2008 y pertenece a GitHub, Inc. Fue escrito usando Ruby on Rails y Erlang.

***GitHub ofrece cuentas gratuitas para proyectos open source y planes de pago para repositorios privados bajo la figura de GitHub Enterprise, sin embargo brindan un periodo de prueba de 45 días para usar este servicio antes de tomar la decisión de comprar una suscripción.***

Github fue el precursor de una funcionalidad muy usada hoy en día conocida como pull request. Un pull request es una útil característica que le permite a un desarrollador notificar a otro acerca de los cambios que realizo en una aplicación de una rama o branch de su repositorio para que este, si así lo decide, fusione o pueda hacer merge.

**Bitbucket**

Bitbucket es un servicio de alojamiento basado en la web para proyectos que utilizan el sistema de control de versiones Mercurial y Git. Fue lanzado en el año 2008 por la empresa Atlassian Software y esta escrito en Python mediante el framework web Django.

***Bitbucket ofrece cuentas gratuitas y comerciales. Las gratuitas cuentan con número ilimitado de repositorios privados y cinco usuarios, aunque tienen la opción de llegar a un total de 8 si invitas a 3 a unirse al servicio.*** Los repositorios privados van desde 10$ al mes por 10 usuarios hasta planes ilimitados. Estos no se muestran en las páginas de perfil, es decir, si un usuario sólo tiene repositorios privados el sitio web indicara que no tiene.

Entre sus características más llamativas están su integración con Jira, un bug tracking system muy usado actualmente. Gracias a esto se puede seguir cada commit de un problema en este popular gestor de incidencias. También permite controlar las actividades por branch, agregar keys de Google Analytics y seguir otros repositorios mediante la opción watch. Tienen un cliente gratuito para Windows y Mac llamado SourceTree.

$ apt-get install git

http://blog.santiagobasulto.com.ar/programacion/2011/11/27/tutorial-de-git-en-espanol.html#Git