2015

Cuarto Ensayo

Universidad Nacional de Ingeniería

Autor: Chunqui Vela José Lendel 20131032H

1. Sistema de control de versiones

Un sistema de control de versiones es un programa destinado a controlar las modificaciones en el desarrollo de cualquier tipo de software. Permite, pues, conocer los cambios realizados en un proyecto de software, quienes son los que lo manipulan, el estado actual del mismo, etc.

2. Importancia de un de control de versiones

El control de versiones es un factor fundamental en el desarrollo de software, y nace de la necesidad de mantener y controlar las modificaciones en un proyecto de software que se desarrolla en equipo.

Si bien el control de versiones es fundamental en un trabajo en equipo, resulta útil incluso para desarrolladores independientes.

Un control de versiones se puede realizar de forma manual, pero existen herramientas que facilitan estas tareas, y son los llamados sistemas de control de versiones o VCS (Versión Control System). Algunos de estos VCS se enlistan a continuación:

- CVS
- Subversion
- SourceSafe
- ClearCase
- Darcs
- Bazaar
- Plastic SCM
- Git
- Mercurial
- Perforce
- Fossil SCM

Estos sistemas nos ayudan con las siguientes funcionalidades:

- Comparar el código de un archivo, de modo que podamos ver las diferencias entre versiones
- Restaurar versiones antiguas
- Fusionar cambios entre distintas versiones
- Trabajar con distintas ramas de un proyecto, por ejemplo la de producción y desarrollo

3. Evolución de los sistemas de control de versiones

Los CVS comenzaron a aparecer por los años 70, y en un principio eran bastante elementales. Los primeros CVS solo permitían que una sola persona manipulase el código del software en desarrollo a la vez, lo cual generaba enormes retrasos en los proyectos de la época. Conforme fueron pasando los años, surgieron nuevos sistemas de control de versiones, siempre evolucionando con el objetivo de resolver las necesidades de los equipos de desarrollo.

4. Clasificación de los sistemas de control de versiones

Según la arquitectura del almacenamiento del código del software en desarrollo, los sistemas de control de versiones se pueden clasificar en:

4.1 Sistemas de control de versiones centralizados

En estos sistemas existe un repositorio centralizado para todo el código del software en desarrollo, del cual es responsable solo una parte de los usuarios, o a veces uno solo. Esta manera de trabajar facilita la administración del software pero reduce la flexibilidad, pues todas las decisiones trascendentales en el código (como crear una nueva rama de código) necesitan de la aprobación del responsable.

En estos sistemas, cada programador mantiene en local únicamente aquellos archivos con los que está trabajando en un momento dado. Obtienen estas porciones de código conectándose con el servidor. Los cambios que se realizan al software en desarrollo se envían al servidor. El sistema centralizado es el único lugar donde está todo el código del proyecto en desarrollo de manera completa.

Ejemplos de estos sistemas son: Subversion y CVS.

4.2 Sistemas de control de versión distribuido

En los sistemas de control de versión distribuido, cada integrante del equipo de programación mantiene una copia local del repositorio completo donde está alojado el código total del proyecto. De esta forma, un programador puede hacer cambios en el código del programa y hacer un *commit* (enviar cambios al sistema de control de

versiones) pero de manera local, sin necesidad de estar conectado a internet, y luego, en cualquier momento compartir lo realizado cuando se disponga de red. Esta forma de trabajar facilita el ser autónomo y trabajar en cualquier situación para los miembros del equipo de programación.

Ejemplos de estos sistemas son: Git y Mercurial.

5. El sistema de control de versiones Git

Git es un sistema de control de versiones distribuido cuyo objetivo es el de permitir mantener una gran cantidad de código a una gran cantidad de programadores eficientemente.

5.1 Diferencias de Git con el resto de sistemas de control de versiones

Hay dos grandes diferencias entre Git y el resto de sistemas de control de versiones de su clase:

La primera gran diferencia de Git con respecto a otros sistemas de control de versiones es la forma que tiene de manejar los cambios en los ficheros. Mientras que otros sistemas de control de versiones almacenan los archivos originales, conservando una lista de los cambios realizados a dichos archivos en cada versión, Git guarda una “foto” (snapshot) del estado de cada archivo en un momento concreto. Si uno de los archivos no ha cambiado no crea una nueva copia del mismo, simplemente crea una referencia al archivo original.

La segunda diferencia es la eficiencia. Git se basa en que cada programador almacena una copia completa del repositorio en su máquina de forma local, incluido el historial de cambios. Esto implica que muchas de las operaciones realizadas sobre el código fuente no tienen lugar en la red, permitiendo que la velocidad de proceso dependa únicamente en los recursos locales.

5.2 GitHub

Githbu es un hosting online para los repositorios que utiliza Git para el mantenimiento y versionado del código fuente, añadiendo una serie de servicios extras para la gestión del proyecto y el código fuente. La parte gratuita de este hosting permite alojar

nuestro código en repositorios públicos, si queremos repositorios privados entramos en la parte “premium” del servicio.

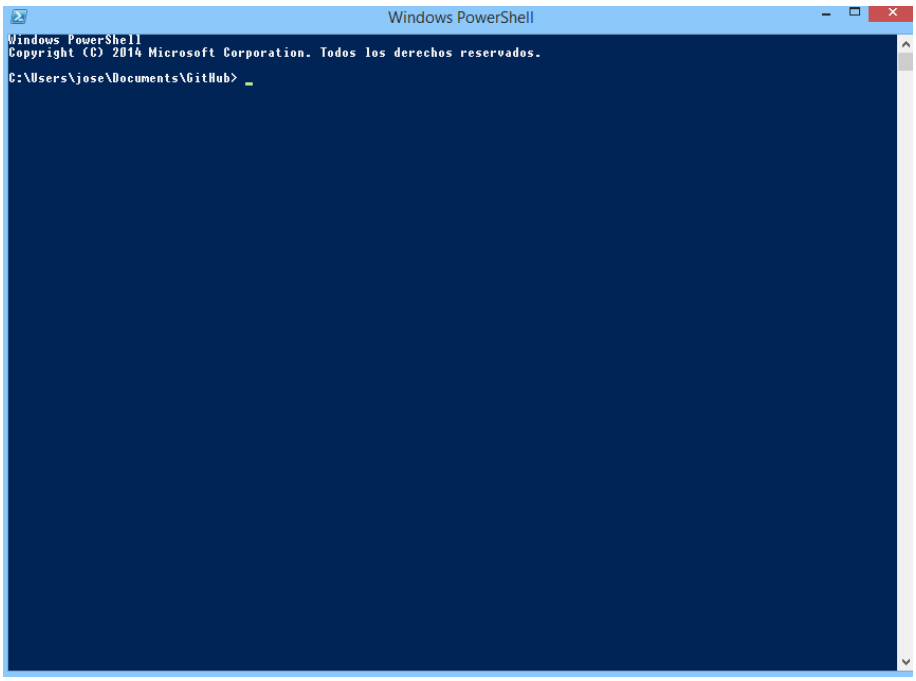
Cabe destacar que Github es un proyecto comercial, a diferencia de la herramienta Git que es un proyecto de código abierto. No es el único sitio en Internet que mantiene ese modelo de negocio, pues existen otros sitios populares como Bitbucket que tienen la misma fórmula.

5.2 Características de GitHub

- Wiki para cada proyecto
- Página web para cada proyecto1
- Gráfico para ver cómo los desarrolladores trabajan en sus repositorios y bifurcaciones del proyecto
- Funcionalidades como si se tratase de una red social, como por ejemplo: seguidores;
- Bueno para trabajo colaborativo entre programadores.

5.2 GitHub para Windows

A continuación algunos comandos para manejar GitHub Shell



Crear un repositorio nuevo

Crea un directorio nuevo:

```
git init
```

Hacer checkout a un repositorio

Crear una copia local del repositorio ejecutando

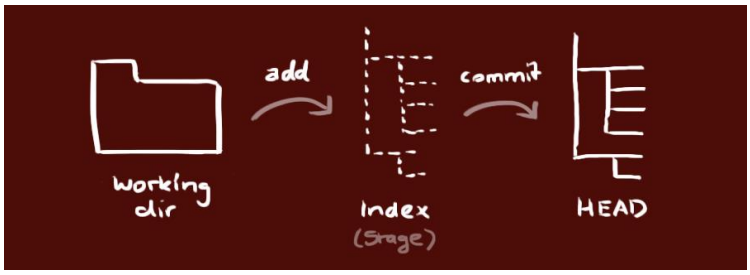
```
git clone /path/to/repository
```

Si se utiliza un repositorio remoto, se usa:

```
git clone username@host:/path/to/repository
```

Flujo de trabajo

El repositorio local está compuesto por tres "árboles" administrados por git. El primero es el `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.



add & commit

Se puede 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 incluido en el **HEAD**, pero aún no en el repositorio remoto.

Envío de cambios

Los cambios están ahora en el **HEAD** de la copia local. Para enviar estos cambios al repositorio remoto se ejecuta:

```
git push origin master
```

Reemplaza *master* por la rama a la que se quiere enviar los cambios.

Si no se ha clonado un repositorio ya existente y quieres conectar el repositorio local a un repositorio remoto, se usa:

```
it remote add origin <server>
```

Ahora se podrá subir cambios al repositorio remoto seleccionado.

6. Bibliografía

https://barradevblog.wordpress.com/2013/01/21/que-es-gitgithub/

http://www.desarrolloweb.com/articulos/introduccion-git-github.html

http://es.wikipedia.org/wiki/Git

http://es.wikipedia.org/wiki/Control_de_versiones

http://es.wikipedia.org/wiki/Programas_para_control_de_versiones

http://es.wikipedia.org/wiki/GitHub

http://rogerdudler.github.io/git-guide/index.es.html