4TO ENSAYO DE LA CLASE

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 versión anterior. Cada versión podría considerarse como una fotografía del estado de la colección un momento determinado de tiempo. La colección de archivos usualmente es código fuente de algún lenguaje de programación, sin embargo, un sistema de control de versiones funciona con cualquier tipo de archivo. Tanto la colección de archivos como su historia completa están guardadas en un repositorio.

SISTEMA DE CONTROL DE VERSIONES DISTRIBUIDAS

Es un sistema de control de versión distribuida hay un servidor central para almacenar el repositorio y cada usuario puede hacer una copia completa del repositorio central mediante un procesollamado "clonación". Cada repositorio clonado es una copia completa del repositorio central y por ser una copia completa posee las 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 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. Para usar Git debes instalarlo en tu sistema. Hay unas instrucciones distintas dependiendo de tu sistema operativo, pero en realidad es muy sencillo.

Características del GIT:

- Diseñado para manejar proyectos grandes.
- Es extremadamente rápido.
- Autentificación criptográfica del historial.
- Formato de archivo muy sencillo y compacto.
- 100% distribuido.
- Se puede sincronizar por cualquier medio.

Conceptos usados en GIT

- **Repositorio.-**Un repositorio contiene la historia, las diferentes versiones en el tiempo y todas las diferentes ramas. En Git cada copia del repositorio es un repositorio completo.
- **Commit.-**Un git commit es una instantánea de un conjunto de cambios o manipulaciones a su árbol de Trabajo. Por ejemplo, si se ha añadido 5 archivos y eliminado otros 2, estos cambios estarán contenidos en un commit (o instantánea). Esto crea un nuevo objeto commit en el repositorio que unívocamente identifica una nueva versión del contenido del

repositorio. Esta revisión puede ser consultada posteriormente, por ejemplo si uno quiere ver el código fuente de una versión anterior. Cada commit posee metadata que nos informa acerca del autor, la fecha y otros datos que nos pueden resultar prácticos a la hora de tratar de encontrar uno determinado.

- **Working tree.-**Se trata básicamente de los directorios y archivos en su repositorio. Se refiere a menudo como su directorio de trabajo.
- **Branch (rama).-**Un branch es un puntero con un nombre determinado por el usuario que apunta a un commit. Posicionarse en un branch utilizando git es denominado como "hacer un checkout" de ese branch. Si estás trabajando en un determinado branch, la creación de un nuevo commit hace avanzar el puntero a esta nueva instancia. Cada commit conoce sus antecesores así como a sus sucesores en caso de tenerlos. Uno de los branches es el default, generalmente llamado master.
- **Tag.-** Un tag apunta a un commit que unívocamente identifica una versión del repositorio. Con un tag, puedes tener un puntero con nombre al que siempre puedas revertir los cambios. Por ejemplo, la versión de 25.01.2009 del branch "testing".
- **URL.-**Una URL en Git determina la ubicación de un repositorio.
- Revisión.-Representa una versión del código fuente. Git implementa las revisiones de la misma manera que los objetos commit.
- **HEAD.-**Es un objeto simbólico que apunta generalmente al branch sobre el que estamos trabajando (lo que también conocemos como "checked out branch"). Si uno cambia de un branch al otro el HEAD apunta al último commit del branch seleccionado. Si uno hace un checkout de un determinado commit, el HEAD apunta a ese commit.
- **Staging área.-** 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.

COMANDOS BÁSICOS EN GIT

CONFIGURAR HERRAMIENTAS

Configura la información del usuario para todos los repositorios locales

COMANDO	DESCRIPCIÓN
$ git config --global user.name "[name]"	Establece el nombre que desea esté anexado a sus transacciones de commit
$ git config --global user.email "[email address]"	Establece el e-mail que desea esté anexado a sus transacciones de commit
$ git config --global color.ui auto	Habilita la útil colorización del producto de la línea de comando

CREAR REPOSITORIOS

Inicia un nuevo repositorio u obtiene uno de una URL existente

COMANDO	DESCRIPCIÓN
$ git init [project-name]	Crea un nuevo repositorio local con el nombre especificado
$ git clone [url]	Descarga un proyecto y toda su historia de versión

EFECTUAR CAMBIOS

Revisa las ediciones y elabora una transacción de commit

COMANDO	DESCRIPCIÓN
$ git status	Enumera todos los archivos nuevos o modificados que se deben confirmar
$ git add [file]	Toma una instantánea del archivo para preparar la versión
$ git reset [file]	Mueve el archivo del área de espera, pero preserva su contenido
$ git diff	Muestra las diferencias de archivos que no se han enviado aún al área de espera
$ git diff --staged	Muestra las diferencias del archivo entre el área de espera y la última versión del archivo
$ git commit -m "[descriptive message]"	Registra las instantáneas del archivo permanente en el historial de versión

CAMBIOS GRUPALES

Nombra una serie de commits y combina esfuerzos ya culminados

COMANDO	DESCRIPCIÓN
$ git branch	Enumera todas las ramas en el repositorio actual
$ git branch [branch-name]	Crea una nueva rama

$ git checkout [branch-name]	Cambia a la rama especificada y actualiza el directorio activo
$ git merge [branch]	Combina el historial de la rama especificada con la rama actual
$ git branch -d [branch-name]	Borra la rama especificada

GITHUB

Herramienta web para gestionar nuestros repositorios. Gratis si los repositorios son abiertos, de pago si queremos tener repositorios privados y múltiples colaboradores. Github github.com es un servicio para alojamiento de repositorios de software gestionados por el sistema de control de versiones Git. Por tanto, Git es algo más general que nos sirve para controlar el estado de un desarrollo a lo largo del tiempo, mientras que Github es algo más particular: un sitio web que usa Git para ofrecer a la comunidad de desarrolladores repositorios de software. En definitiva, Github es un sitio web pensado para hacer posible el compartir el código de una manera más fácil y al mismo tiempo darle popularidad a la herramienta de control de versiones en sí, que es Git.

Github ofrece al desarrollador toda la potencia y agilidad del sistema de control de versiones Git, más un interesante set de herramientas añadidas:

- Wiki
- Sistema de seguimiento de incidencias
- Interfaz gráfica para revisión/comparación de código
- Visor de ramas de desarrollo

FUNCIONAMIENTO DE GITHUB

Lo primero que debemos hacer es crear una cuenta en https://github.com. La activamos por mail y ya podemos crear nuestros repositorios. Los repositorios de Github son el almacén que utilizamos para guardar nuestro código.

Github nos ofrece la opción de crear un repositorio vacío, recomendable cuando vamos a iniciar un nuevo desarrollo, o la opción de importar un proyecto ya existente, elegimos la que más nos convenga y mediante unos pocos comandos de consola configuramos la rama principal de nuestro repositorio, que por defecto se llamará “master”.

Cada programador puede crear sus propias ramas de desarrollo, donde tiene que llevar a cabo sus modificaciones, sin interferir en el trabajo de sus compañeros. Cuando terminamos y validamosun desarrollo paralelo, lo unimos con la rama principal y todos los miembros del equipo pueden descargar las nuevas modificaciones, sin alterar los desarrollos que estén llevando cabo en ese momento.

Después de alojar el repositorio público en Github.com, cualquier usuario de la comunidad podrá aportar ideas, hacer un seguimiento del proyecto, incluso copiarlo y modificarlo a su gusto bajo la misma licencia.

¿POR QUÉ USAR GITHUB?

Github hace el trabajo en equipo más ágil y sencillo, ayuda a la detección de fallos, a disminuir errores humanos, al seguimiento por etapas del proyecto, al mantenimiento de diferentesentornos, etc.

Como conclusión podemos decir que alojar proyectos como repositorios en Github, es una decisión profesional inteligente, en caso de los repositorios públicos, porque te beneficiarás de los conocimientos de otros programadores, y en el caso de los privados, dispondrás de un robusto sistema de gestión de proyectos que hará que el trabajo en equipo sea mucho más rápido.