Git Desde Cero

1. Introducción

Introducción

¿Qué es Git?
Git es un sistema de control de versiones

¿Qué es el control de versiones, y por qué debería usarlo?
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.

Ventajas de usar control de versiones

- Es un método de respaldo
- Revisión histórica
- Facilita la Colaboración

Secciones del curso

- Fundamentos de Git
- Repositorios remotos
- Ramificaciones (branches)
- Herramientas
- Portal Github

Sobre el instructor

- 15 años liderando equipos de desarrollo web en Nueva York
- Trabajado para empresas grandes e incipientes (startups)
- Apasionado sobre nuevas tecnologías de aprendizaje

Tipos de sistemas de control de versiones

- Local
- Centralizado
- Distribuído

¿Es Git mejor que otras herramientas similares?

- Distribuido
- Más control sobre la historia
- Mejor manejo de ramas
- Guardar provisionalmente tu trabajo

2. Fundamentos de Git

El ciclo de los archivos

El ciclo de los archivos (untracked/unmodified/modified/staged)

Instalando Git

Instalando Git en Windows

Dirígete en Internet Explorer o navegador similar a la dirección: http://git-scm.com
Utiliza las opciones pre-configuradas.

El programa Git Bash en Windows es una utilidad que permite usar Git exactamente de la misma manera que usas Git en computadoras Linux/Unix.

La mejor manera de usar Git es a traves del terminal Git Bash.
Puedes correr Git Bash desde la opción de “Programas > Git > Git Bash”.

Comandos para manejo de directorios y archivos en Git Bash

ls = permite ver los contenidos del directorio en donde estás en este momento. Cuando empiezas Git Bash, por lo general estás en lo que se llama tu “Home Directory” o el directorio inicial del usuario. Generalmente es representado con una señal de tilde “~”.

mkdir <directorio> = permite crear un directorio con el nombre que pongas a continuación. Es preferible solo usar letras, sin espacios ni acentos. En nuestro ejemplo, creamos el directorio de prueba que llamamos “testgit”, escribiendo mkdir testgit.

cd <directorio> = permite cambiar al directorio que escribas. Puedes usar el nombre de cualquier directorio que veas listado en el comando “ls” (descrito arriba).

Para verificar que tienes Git instalado, escribe “git” en el terminal y para ver la versión de Git que tienes instalada, escribes “git --version”.

Instalando Git en Mac

*** Nota Importante ***
Si estás en el sistema operativo “Mavericks” (Mac OSX 10.9), y has instalado herramientas de desarrollo de Xcode, es posible que ya tengas Git. Antes de instalar verifica si lo tienes instalado abriendo el terminal (“Command” + “Space” + “Terminal”) y escribe el comando “git”. Si te sale un error, prosigue con esta lección. Si te salen los comandos de Git, verifica que la versión sea por lo menos 2.0, haciendo “git --version”. Si tu versión es anterior a 2.0, prosigue con esta lección.

Dirígete en Safari o navegador similar a la dirección: http://git-scm.com

Luego de bajar el paquete, tienes que abrirlo presionando la tecla de “control” y haz click al paquete, y selecciona la opcion “Abrir” o “Open”. Esto es necesario, ya que Git es un paquete que bajas directamente de la web y las Mac no permiten abrir paquetes que no sean bajados por su aplicación de paquetes oficial.

La mejor manera de usar Git es a traves del terminal de Mac. Para abrir el terminal, puedes oprimir la tecla de “command” y la barra de espacio a la vez. Esto
abrirá el “Spotlight” que es una herramienta de búsqueda. Si escribes “terminal”, veras el programa terminal con una pantalla negra como ícono. Haz click para abrir el terminal.

Una vez en el terminal, escribe el comando “git” y presiona “enter”. Si te salen los comandos de Git, estás listo para continuar.

Si te sale un error de que no se encontró Git, tendrás que modificar tu configuración de perfil, y decirle manualmente al sistema operativo dónde conseguir Git.

Para ello edita tu perfil, usando la herramienta de edición de texto “vi”. Escribe “vi ~/.profile”. Esto te abrirá el editor. Presiona la tecla “i”, para insertar texto, mueve el cursor al final del archivo y presiona “enter” para darle más espacio y luego escribe:
export PATH=$PATH:/usr/local/git/bin

Luego le das a la tecla de “esc” y escribe “:wq” (dos puntos, la letra “w” y la letra “q”) y le das a “enter”.

Finalmente cierra el terminal (en la opción de menú “Terminal > Quit Terminal” o haciendo “command” + “q”). y vuelve a abrirlo con “command” + “space” y poniendo “terminal” en la búsqueda y seleccionando la aplicación.

Una vez dentro del terminal, escribe “git” y verifica que te salen los comandos de Git.

El editor de texto

Aquí están los links a los editores de código recomendados en la lección:

Mac
- Textmate (pagado)
- Sublime Text (pagado pero con uso ilimitado gratis)
- Atom (gratis)
- TextEdit (gratis con la Mac)

Windows
- Notepad++ (gratis)
- Sublime Text (pagado pero con uso ilimitado gratis)
- Atom (gratis)
- Notepad (gratis con Windows)

Operaciones básicas

Comandos para manejo de directorios y archivos en el terminal ó Git Bash
pwd = te dice el directorio seleccionado en este momento

ls = permite ver los contenidos del directorio en donde estás en este momento. Cuando empiezas Git Bash, por lo general estás en lo que se llama tu “Home Directory” o el directorio inicial del usuario. Generalmente es representado con una señal de tilde “~”. Si quieres ver todos los directorios y archivos, incluyendo los escondidos (que empiezan con “.”) pones “ls -a”.

mkdir <directorio> = permite crear un directorio con el nombre que pongas a continuación. Es preferible solo usar letras, sin espacios ni acentos. En nuestro ejemplo, creamos el directorio de prueba que llamamos “testgit”, escribiendo mkdir testgit.

cd <directorio> = permite cambiar al directorio que escribas. Puedes usar el nombre de cualquier directorio que veas listado en el comando “ls” (descrito arriba). Si quieres salir al directorio anterior, escribes “cd ..” (cd y dos puntos seguidos). Si quieres ir rápidamente a tu directorio inicial de usuario, o “Home folder”, puedes escribir “cd ~” (“cd”, espacio y el signo de “tilde”).

clear = permite limpiar la pantalla.

Comandos de Git
Git init = inicializa un directorio y lo convierte en un repositorio Git. Al inicializar el repositorio, Git crea un directorio escondido llamado “.git” dentro del nivel principal del directorio. Aquí Git guarda información sobre el repositorio.

Antes de hacer cualquier operación, debemos configurar nuestro usuario. Para ello le decimos a Git nuestro nombre y nuestro correo electrónico, haciendo lo siguiente:
git config user.name "Jose Diaz”
git config user.email jose@example.com

Esto solo guarda esta información en el repositorio actual. Si quieres que Git use siempre la información en cualquier repositorio, puedes usar el parámetro “global”, escribiendo:

git config --global user.name "Jose Diaz”
git config --global user.email jose@example.com

Git status = nos da información sobre el estado actual de nuestro repositorio.

En la clase utilizamos el editor de código Atom, que es un proyecto de Github, y que es totalmente gratis y funciona con Windows, Mac y Linux. Para bajarte Atom, dirígete a https://atom.io/.

Git add <archivo o lista de archivos> = permite añadir un archivo nuevo o uno modificado a la lista de archivos preparados (“staging”) para la próxima instantánea. Recuerda que si un archivo que está en preparación es editado, sale de esta lista y pasa al área de “modificados”, por lo tanto no será incluído en la próxima instantánea, ya que Git piensa que todavía estás trabajando con él. Cuando termines tu trabajo en ese archivo, haz “git add” de nuevo a ese archivo.

Git commit -m “<mensaje”> = te permite crear una instantánea con todos los archivos en el área de preparación y con un mensaje. Cualquier archivo en estado “modificado” no será incluído. Es muy importante escribir un mensaje que sea completo describiendo qué cambios se están introduciendo.

Cada instantánea tiene un identificador único llamado “hash”. El hash tiene por lo general letras y números y con ese identificador puedes recuperar el estado del proyecto en ese momento.

Operaciones básicas visualizadas

Diagrama de commits

La historia del proyecto

Git log te permite ver las instantáneas del proyecto en orden descendiente.
Cada línea tiene un identificador de tipo SHA1.

Cuando haces git log, puede ser que entres a una ventana separada que es parte de una utilidad llamada “less” que permite leer contenido. Usando las teclas de flechas hacia arriba y hacia abajo, puedes leer el historial. Para salir de esa ventana, le das a la tecla “q”.

Si no quieres usar “less”, y en vez usar “cat”, que muestra los contenidos en el mismo terminal, puedes entrar el siguiente comando:

git config --global core.pager cat

Para mostrar diferencias entre instantáneas, puedes usar:
git log -p

Si quieres ver sólo las últimas “n” confirmaciones, puedes usar:
git log -<n>

Para ver una versión abreviada de los commits, puedes usar:
git log --oneline

Para ver una representación visual de las ramas, puedes poner:
git log --graph

Si quieres ver el historial de instantáneas controlando las fechas, puedes usar:
git log --since=2.days (ver commits desde hacen 2 días)
git log --since=2.weeks (ver commits desde hacen 2 semanas)
git log --since=”2014-09-12” (ver commits desde septiembre 12 del 2014)
git log --after=”2014-09-12” (ver commits después de septiembre 12 del 2014)
git log --until=”2014-09-12” (ver commits hasta de septiembre 12 del 2014)
git log --before=3.days (ver commits antes de 3 días)

Para ver las contribuciones de un usuario específico escribe:
git log --author=”Juan Perez”

Finalmente, puedes combinar cualquiera de estos parámetros, ejemplo:
git log -3 --oneline --before=2.days
Para ver todas las opciones disponibles para log, puedes escribir:
git help log

Para ver cómo se veía el repositorio en cualquier instantánea, se usa:
git checkout <hash>

Donde “hash” corresponde al identificador.

No es recomendable hacer cambios al proyecto en este estado de “Detached head” o cabezal desprendido. Sin embargo es totalmente válido arrancar una nueva rama de este estado.

Otras operaciones básicas

Para mover un archivo de un directorio a otro, o para renombrarlo se usa:
git mv <origen> <destino>

Ejemplo, si queremos renombrar un archivo foo.txt a bar.txt, se usa:
git mv foo.txt bar.txt

Si queremos mover una fotografía llamada foo.jpg de un directorio a otro:
git mv fotos/foo.jpg imagenes/foo.jpg

Para eliminar un archivo, se usa:
git rm <archivo>

Para ignorar ciertos archivos, como archivos temporales o generados automáticamente, se crea un archivo llamado .gitignore en el nivel superior del directorio del repositorio.

El .gitignore usa patrones para reconocer tipos de archivos. Para una guía de estos patrones, puedes leer aquí.

Finalmente para ver las diferencias entre nuestro trabajo actual y una instantánea específica, puedes hacer:
git diff <SHA1>

También pueden chequear si tienen la herramienta gráfica, utilizando:
gitk

¿Cómo deshacer cosas?

Para cambiar cualquier archivo o texto de confirmación después de hacer commit, podemos usar:
git commit —amend

Por lo general esto comenzará el editor de texto “vi”, que es un editor incorporado. Puedes cambiar el editor que se use, con el comando:
git config --global core.editor <editor>

Si usas “vi”, recuerda que puedes mover el cursor a donde quieras con las flechas del teclado. Luego le das a la tecla “i” para entrar al mode de insertar, haces los cambios, y luego le das a la tecla “escape” y escribes dos puntos, la tecla “w” y la tecla “q”.

Si quieres sacar a un archivo o a un grupo de archivos del área de staging, haces lo siguiente;
git reset HEAD <file>

Esto pondrá el archivo de nuevo en el área de modified.

Si quieres desechar los cambios de un archivo en el que estabamos trabajando y queremos devolverlo al estado en el que estaba en la última instantánea, entonces haces:
git checkout — <file>

Si quieres desechar todos los cambios que has hecho localmente desde la última instantánea, utiliza:
git reset HEAD --hard

Si quieres desechar el último commit y regresar al repositorio como estaba en la anterior instantánea, escribes:

git reset HEAD^ --hard

3. Repositorios Remotos

Introducción a repositorios remotos y git clone

Git te permite trabajar con repositorios remotos utilizando varias utilidades que veremos en esta sección.

La primera herramienta es:
git clone <url> <directorio>

Git clone copiará todo el código que haya en ese URL a tu computador y toda la historia del proyecto, es decir, las instantáneas y las ramas que tenga disponible. El código será copiado en el directorio que elijas, y si no pones un directorio, lo pondrá en un directorio con el mismo nombre del proyecto.

Esta operación es la base de todo el movimiento “Open Source”, ya que cualquier desarrollador en el mundo puede colaborar con cualquier proyecto que le sea útil, encontrando “bugs” o haciendo mejoras.

git remote

Para gestionar repositorios remotos, se usa el comando git remote.

Para ver todos los repositorios remotos que tiene un proyecto, escribes:
git remote -v

Un proyecto puede tener múltiples repositorios remotos asociados.

Para ver los detalles de un repositorio remoto, usamos:
git remote show <nombre del repositorio>

Para añadir un repositorio remoto, pones:
git remote add <nombre> <url del repostorio remoto>

Nombre es el nombre que le pondrás al repositorio. Por lo general se usa “origin” si es el repositorio remoto principal.

Para renombrar un repositorio usamos:
git remote rename <nombre anterior> <nombre nuevo>

Finalmente, para borrar un repositorio remoto, usas:
git remote rm <nombre>

git fetch and pull and push

Hay tres operaciones importantes en el manejo de repositorios remotos.

La primera, git fetch, solo agarra los datos, pero no aplica los cambios localmente que se han bajado. El formato es:
git fetch <nombre repositorio remoto> <rama remota>

Puedes omitir el parámetro de la rama remota si quieres bajarte todos los cambios disponibles.

Para aplicar los cambios a tu repositorio local, usas:
git merge origin master

Para obviar este paso intermedio, se puede usar un sólo comando, que es git pull. El formato es:
git pull <nombre repositorio remoto> <rama remota>

Puedes obviar el nombre de la rama remota y te halará los cambios de la rama “master”.

Para empujar código de nuestro repositorio local al remoto, usamos:
git push <nombre repositorio remoto> <rama remota>

4. Ramificaciones en Git

Operaciones con ramas

Las ramas son básicamente diferentes versiones de un proyecto que viven en tu repositorio. Por ejemplo, la rama “master” puede ser la versión que está al aire en tu website, pero tendrías otra rama llamada “en-desarrollo” que es donde tu equipo está desarrollando código que no está estable.

Para crear una rama, utilizas:
git branch <nombre de rama>

Para cambiarte a esa rama, usas:
git checkout <nombre de rama>

Para incorporar los cambios de una rama dentro de otra, hacemos primero un checkout a la rama a donde van a entrar los cambios y usamos:
git merge <rama a incorporarse>

Finalmente, para borrar una rama, hacemos:
git branch -d <nombre de rama>

5. Herramientas de Git

Herramientas de Git

Una herramienta llamada git stash te permite guardar temporalmente todos los cambios que tengas en ese momento, pero sin necesidad de hacer un commit o instantánea. De esa forma puedes guardar tu trabajo y puedes hacer checkout de otras ramas o traer cambio de un repositorio remoto. El formato del comando es:
git stash

Para ver una lista de tus “stashes”, puedes hacer:
git stash list

Para aplicar esos cambios de nuevo a tu repositorio, usas:
git stash apply

Lo cual aplicará los cambios del “stash” más reciente.

Si quieres aplicar un “stash” específico, pones:
git stash apply <stash id>

Otra herramienta muy útil es “Cherry picking”, que te permite aplicar los cambios que se hayan introducido en cualquier instantánea en tu proyecto actual. El formato es:
git cherry-pick <id de instantánea>

Referencia:
http://git-scm.com/book/es/Empezando-Acerca-del-control-de-versiones

Diccionario:
untracked = sin seguimiento tracked = siguiendo staged = preparado commit = confirmación snapshot = instantánea