# Git y GitHub

Objetivos

* Aprender los conceptos fundamentales de control de versiones
* Instalar Git y ejecutar los comandos más usuales
* Compartir proyectos en la nube utilizando GitHub

Características de Git

Software open source multiplataforma creado en 2005 por Linus Torvalds. Ofrece al programador llevar un control de ediciones, revisar el registro de cambios, ver diferencias entre versiones y cambiar a versiones anteriores.

Git es un sistema de control de versiones distribuido, es decir, cada uno de los clientes tiene su propia copia. Está diseñado para trabajar exclusivamente en archivos de texto, no hay un solo punto de falla y facilita la colaboración entre desarrolladores.

La documentación se encuentra en la siguiente página: <https://git-scm.com/book/es/v1/Empezando>

Configurar Git

Parámetros de usuario, correo y contraseña:

git config –global user.name “IanRaRev”

git config –global user.email [ian.rarev@gmail.com](mailto:ian.rarev@gmail.com)

Inicializar un repositorio

Al crear o inicializar un repositorio con:

git init

Se crea una carpeta que llamada .git (se encuentra oculta) y contiene toda la información de Git sobre el proyecto.

Añadir archivos, status y commit

Para añadir un archivo a nuestro repositorio se utiliza el comando:

git add nombre archivo

Si se quiere añadir más de un archivo se utiliza el comando

git add .

Si hemos añadido un archivo y queremos revertir esto (el archivo está en estado staging) se utiliza el comando:

git reseat HEAD nombreArchivo.extension

El estado staging de los archivos se refiere a que están listos para ser confirmados, cuando no están en staging Git todavía no los reconoce o sus cambios no están listos para confirmarlos.

Ahora, si el un archivo no está en staging, podemos revertir los cambios al último commit que hicimos, para esto utilizamos

git checkout – nombreArchivo

Ahora realizamos un commit, el cual es considerado un punto de control en donde se ha modificado algo importante para el proyecto. Para esto se usa el comando

git commit -m “Descripción del commit”

Ignorar archivos

Esto se hace especialmente cuando tenemos archivos que no son de texto plano. Dentro de la carpeta de trabajo sincronizada con Git, se crea un archivo con el nombre *.gitignore* y se escribe el nombre del archivo y/o carpeta que se desean ignorar. A pesar de modificar el archivo y/o carpeta, no se actualizará.

nombreCarpeta/

nombreArchivo.extensión

Borrar archivos

Si borramos un archivo por accidente o alguien más lo borra en el proyecto, siempre y cuando podemos recuperarlo con el comando

git checkout nombreArchivo.extensión

Ahora, cuando queremos borrar un archivo desde Git, se ejecuta el siguiente comando

git rm nombreArchivo.extensión

Y para guardar los cambios, se realiza un commit.

Si nos arrepentimos y ese archivo no debió ser borrado, para cuando el archivo está en modo unstage se utiliza el comando

git checkout nombreArchivo.extensión

Y en caso de que el archivo esté en estado staging (es decir, ya se usó el comando ***git add .***), se utiliza el comando

git reseat HEAD

Trabajar con el log

El comando

***git log***

nos presenta en consola el registro o historial de todos los commit’s. Los códigos que aparecen en los commit, se les llama hash y son importantes. Un hash es una cadena de números y letras únicos para nuestro proyecto, es decir, un ID de los commit´s. Esta será la llave para regresar a versiones anteriores en caso de error o experimentación.

Para regresar a alguna versión anterior, se ejecuta el comando

git checkout hash

Esto permite “experimentar” con el proyecto a partir de el commit que hayamos elegido. Cuando hayamos hecho nuestros “experimentos”, para guardar (en otra rama) los cambios se utiliza el comando que aparece en consola:

git checkout -b newBranch

Ahora guardamos los cambios y vemos que ahora ya no estamos en la misma rama.

Ramas

Las ramas son en términos coloquiales una línea de tiempo. Todo proyecto de Git parte de una rama original llamada **master** en donde se van haciendo cambios de manera secuencial (commit’s). En algún punto, esa rama puede separarse en dos o más ramas (depende del proyecto) y cada una puede tener distintos cambios.

Para ver las ramas que tenemos en el proyecto, se utiliza el comando

git branch

Ahora, para cambiar de rama se utiliza el comando

git checkout nameBranch

Y Git automáticamente nos redirige a la rama que queremos. Es importante guardar todos los cambios antes de cambiar de rama.

Para borrar una rama (que nunca se combinó con la rama master, es decir, no hizo merge con master), se utiliza el comando

git branch -D nameBranch

Ahora, si queremos realizar merge (fusionar) los cambios entre la rama master y alguna otra rama, primero nos colocamos en la rama donde se van a ver reflejados los cambios y se utiliza el comando:

git merge nameBranch

Donde nameBranch es la rama que queremos integrar.

En el caso de que se hizo el merge correctamente, podemos eliminar la rama utilizada con

git Branch -d nameBranch

Donde nameBranch es la rama con la que “experimentamos”.

Introducción a GitHub

Para subir tu repositorio a GitHub, se hace lo que viene en la pagina de GitHub después de crear un nuevo repositorio.

Al momento de realizar un commit local, se quiere que se publiquen los cambios en GitHub, por lo que se utiliza el comando

git push origin nameBranch

Donde el nombre de la rama es a donde se verán reflejados los cambios.

Para descargar un proyecto desde GitHub, se utiliza el siguiente comando

git clone URL.git

Si alguien más realiza cambios en el proyecto y los sube a la nube, puedes descargar la versión más reciente con el comando

git pull origin nameBranch