Contenido

[¿Qué es GIT? 2](#_Toc126594838)

[¿Para qué usar GIT? 2](#_Toc126594839)

[8 Razones por las que debes usar Git 2](#_Toc126594840)

[Conceptos 5](#_Toc126594841)

[Git Comandos básicos en la terminal de Git Bash en Windows 7](#_Toc126594842)

[Primeros pasos 9](#_Toc126594843)

[Referencias 12](#_Toc126594844)

¿Qué es GIT?

Es un software de control de versiones, su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos (También puedes trabajar solo no hay problema 😃). Existe la posibilidad de trabajar de forma remota y una opción es GitHub.

¿Para qué usar GIT?

Permite regresar a versiones anteriores de forma sencilla y muy rápida.

Facilita el trabajo colaborativo.

Permite respaldar tus proyectos en la nube (ej con github).

Reduce considerablemente los tiempos de deploy.

Las "branches" o ramas, permiten trabajar con una base de código paralela al proyecto en sí.

[Fuente](https://blog.coffeedevs.com/8-razones-para-usar-git/) ver 8 razones por las que debes usar GIT

8 Razones por las que debes usar Git

Para los que aún no han oído hablar de [Git](https://git-scm.com/) (donde estuvieron?!), es un software de gestión de versiones diseñado por [Linus Torvalds](https://es.wikipedia.org/wiki/Linus_Torvalds), creador del kernel de Linux, para mantener de forma eficaz un proyecto de software de forma distribuida, ordenada y robusta.

Si estás leyendo esto, lo más probable es que todavía estés utilizando **FTP** para mostrar cambios en un sitio web alojado en un servidor, reemplazando archivos por su versión más reciente.

*Si todavía están usando FTP o SFTP para desplegar sus proyectos en un servidor, es hora de que reconsideren pasarse a Git, y abrazar todas las ventajas de la gestión de versiones*

Veamos algunos de los problemas que surgen con este enfoque:

1- Trabajo colaborativo. Cómo hacemos para trabajar varios desarrolladores en un mismo archivo? Qué versión es la final? Cómo hacemos para que nuestros cambios no sobrescriban los de mis compañeros?

2- Volver a una versión anterior. Ctrl+Z es la única forma de volver atrás en un mismo archivo, eso teniendo en cuenta que el archivo o el editor tengan la historia del mismo, de lo contrario, estamos perdidos.  
Para evitar eso, tenemos varios backups del mismo archivo y sus distintas versiones editadas a lo largo del tiempo.

3- Subir múltiples archivos en un proyecto muy grande (1000+) lleva muchísimo tiempo, ya que generalmente reemplazamos **TODO** el proyecto, o sino tenemos que ir manualmente, directorio por directorio, subiendo sólo los archivos cambiados.  
Lento y propenso a errores, además del ancho de banda que consume reemplazar todo el proyecto cada vez que hacemos cambios.

4- Facilidades a la hora de compartir el código. Supongamos que queremos hacer una librería o un proyecto **Open Source**. Quienes tienen permiso a que archivos, cómo nos ponemos de acuerdo para editar código sin pisar el trabajo del otro, cómo evitamos que se borre o pierda parte del trabajo por algún error?

Si todavía están usando FTP o SFTP para desplegar sus proyectos en un servidor, es hora de que reconsideren pasarse a Git, y abrazar todas las ventajas de la gestión de versiones

1- **Facilita el trabajo colaborativo**: Distintos programadores pueden estar editando el mismo archivo, o versiones distintas del mismo archivo, y todos los cambios serán reflejados en el documento final.

2- **Reduce considerablemente los tiempos de deploy** (despliegue) de un proyecto, al subir solamente los cambios (no los archivos cambiados, sólo los cambios!), que en Git se conoce como "diff": las diferencias entre la versión local (la que estás trabajando) y la "master" que está en el servidor central.

3- **Permite regresar a versiones anteriores de forma sencilla y muy rápida.** En caso de haber realizado cambios negativos en un proyecto en producción, volver a la última versión estable es un simple comando, que retrocede a su estado previo **todos los cambios realizados en la última modificación**. Esto puede hacerse hacia cualquier versión del proyecto, sin importar la cantidad o calidad de los cambios posteriores.

4- **Permite generar flujos de trabajo** que facilitan el desarrollo y mantenimiento de proyectos de gran tamaño.

5- **El ecosistema Git es increíble**, y agrega un montón de herramientas a nuestra disposición para facilitarnos el trabajo, de forma robusta, rápida y profesional.

A través de los **"hooks"** de Git, los distintos servicios pueden detectar cambios en el historial de versiones y realizar acciones automáticas (como actualizar los archivos en el servidor o ejecutar una suite de tests y enviarnos su resultado), dejándonos tiempo libre para cosas más productivas!

6- Las **"branches"** o ramas, permiten trabajar con una base de código paralela al proyecto en sí, donde podemos corregir bugs o desarrollar nuevas características para el producto sin afectar el **"master"**, pero manteniendo todas las ventajas de usar un sistema de control de versiones. Una vez que estamos contentos con nuestro **"branch"**, podemos combinarlo con el **"master"** o, en lenguaje Git, hacer un **"merge"**.

7- Empezar a trabajar desde otro entorno es tan fácil como **"clonar"** el proyecto a tu nuevo entorno, trabajar sobre los archivos que se quieran, y subir los cambios al **"master"** o a una **"branch"**.

8- Sistema de etiquetas, para etiquetar las distintas versiones del proyecto. Esto es un marcador a una versión específica del proyecto, sólo que en lugar de tener distintos backups de versiones anteriores, apuntamos a distintas versiones dentro de la misma base de código.

Y si todo eso no los entusiasma para cambiarse a Git, les digo que por experiencia personal, una vez que empiecen a usarlo, nunca más van a poder trabajar en un proyecto en el que no lo tengan!

Conceptos

Una captura de un videojuego

Descripción generada automáticamente con confianza baja



Tratando de explicar la imagen: Tenemos nuestro directorio local (una carpeta en nuestro pc) con muchos archivos, Git nos irá registrando los cambios de archivos o códigos cuando nosotros le indiquemos, así podremos viajar en el tiempo retrocediendo cambios o restaurando versiones de código, ya sea en Local o de forma Remota (servidor externo). En la práctica quedará más claro.

Git, relacionado con tu ordenador. Github relacionado con la nube.

Git Comandos básicos en la terminal de Git Bash en Windows

Si estás empezando a utilizar el sistema de control de versiones de Git, estos comandos te serán útiles para navegar y hacer operaciones entre tus carpetas y archivos en Windows.

Primero debemos recordar la diferencia en la estructura de archivos de Windows, Mac o Linux.

La ruta principal en Windows es c:/, en UNIX es solo /.

Windows no hace diferencia entre mayúsculas y minúsculas, pero UNIX si.

Git Bash usa la ruta /c para dirigirse al disco c:/ en Windows. Por lo tanto, la ruta del usuario con el que estás trabajando es */c/Users/NombreUsuario*

Comandos

pwd: nos muestra la carpeta actual en la que nos encontramos.

mkdir: nos permite crear carpetas, p. ej. mkdir *NuevaCarpeta*

touch: nos permite crear archivos nuevos, p.ej. touch *NuevoArchivo.txt*

cat: nos permite ver el contenido de un archivo, p.ej. cat *NuevoArchivo.txt*

cd: nos permite cambiarnos de carpeta, p.ej. cd *NuevaCarpeta.*

cd .. : nos permite regresar al directorio o carpeta anterior.

cd o cd ~: nos lleva a la ruta del usuario.

cd /c: nos vamos al disco C:/.

cd -: nos lleva directamente al último directorio visitado.

ls: nos permite ver los archivos de la carpeta donde estamos actualmente.

ls -l: Ver todos los archivos como una lista en donde incluye el usuario, grupo, permisos sobre el archivo, tamaño, fecha y hora de creación.

ls -lh: Muestra la misma información que ls-l pero con unidades de tamaño, es decir, kb o mb.

ls-R: muestra el contenido de todos los sudirectorios de forma recursiva.

ls -S Ordena los resultados por tamaño de archivo.

rm: Nos permite borrar un archivo o carpeta ej: rm NuevoArchivo.txt

rmdir “nombre del directorio”/: borrar un directorio: Solo funciona con directorios vacíos.

rm -r ‘nombre de la carpeta’: me permite eliminar la carpeta y los archivos dentro de ella de forma recursiva.

cp “nombre del archivo que quremos copiar” “nombre del directorio a donde lo queremos copiar”: nos permite copiar un archivo.

mv “el directorio de donde queremos mover/el nombre del archivo” “el directorio hacia donde lo queremos mover”: nos permite mover un archivo.

clear: nos permite limpiar la pantalla.

history: ver los últimos comandos que ejecutamos y un número especial con el que podemos volver a repetir el comando.

Primeros pasos

¡Importante!

Utilizar la consola de git bash no la cmd de Windows ni power Shell

//Versión de git (sirve para comprobar que lo tenemos bien instalado)

git version

//Registro de usuario asociado a git. La opción global es para que se configure para todos los proyectos frente a local, para el proyecto actual.

git config --global user.name "mi nombre"

git config --global user.mail "mi correo"

(muy conveniente utilizar el nombre y correo de github).

Para ver estos datos

git config user.name

git config user.mail

//para señalar el editor de texto VSCode como editor por defecto. –wait hace que el terminal espera a que se cierre el editor

git config --global core.editor “code --wait”

//para ver el fichero de configuración global, se abre en vscode

git config --global -e

// comportamiento del CRLF true si Windows input si mac o linux

git config --global core.autocrlf true

// Importante: de cara a

git config --global http.proxy https://proxy.jcyl.es:80

//Para ver los parámetros de configuración de git

git config --list

//Mi primer repositorio (sólo una vez por proyecto)

git init

// para ver el estado de los archivos (si han sido registrados o no)

git status

git status -s //versión resumida

//Añadir a la staged área

git add . // todos los ficheros

git add \*.html

//Confirmar los cambios (hacer una todo del proyecto, enviarlo al repositorio)

git commit -m “Comentario con sentido”

Es obligatorio un comentario si no le escribimos el parámetro -m nos abrirá el editor por defecto.

Trabajo con Github

Esto significa subir a un repositorio en la nube para poder compartir un repositorio o acceder a él desde cualquier dispositivo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteSe crea el repositorio en Github (Se pueden ver los pasos en este video, por ejemplo. <https://www.youtube.com/watch?v=PW_A-lOpVV0&t=2818s>)

Lo más sencillo es crear el repositorio primero en github y después enlazarlo con nuestro repositorio local. Basta con copiar las 3 instrucciones del segundo bloque.

git push -u origin main //sube toda la rama main

La operativa más frecuente serán 3 gits add, commint -m y push.

Con esto vemos el registro de los commits realizados interesa el parámetro -–oneline poque captura el id del commit. Importante si tenemos que volver a él

git log --oneline // para ver el histórico de los movimientos

git clone [https://github.com/PDMure/wtt\_repo.git //](https://github.com/PDMure/wtt_repo.git%20//) //clona el repositorio completo en una carpeta local

Esto puede ser útil en caso de que se borre la carpeta completamente. Con este comando se recupera además la secuencia de actualizaciones. Todos los commits respaldados.

git remote -v // para ver la url donde está el repositorio remoto

.gitignore //Fichero de tipo texto para que se escriba en el los ficheros o carpetas a los que no se va a hacer seguimiento. No subirán al repositorio

HOSTING GRATUITO PARA PRUEBAS

Con github page podemos probar los proyectos estáticos (sólo con html, css y javascript) en un hosting gratuito de github. Desde el repositorio en settings | page (se puede hacer uno por cada repositorio).

<https://www.youtube.com/watch?v=wL8gCda3pG4>

Otras alternativas

Crear y publicar un sitio web con netlify (enlazado de forma dinámica a github)

<https://www.youtube.com/watch?v=1sWBVXUQTII>

Comparativa de sitios para hosting de pruebas.

<https://geekflare.com/es/best-static-site-hosting-platform/>

git checkout “ID” //para viajar al estado del commit indicado en el ID

¡¡OJO!! Con checkout sólo viajamos en el tiempo. No debemos hacer modificaciones se debe usar sólo para ver.

git branch -m master main // para cambiar de nombre la rama de master a main

Si queremos que a partir de ahora todos los proyectos que creemos tengan de nombre la rama main usaremos esta sentencia

git config –global init.defaultBranch main

Para viajar en el tiempo y ahora sí, borrar commits usamos

git reset “ID”

git reflog //nos muestra todo el historial de los commits realizados incluso de un git reset –hard (que borra ficheros)

OJO!! No es recomendable hacer un git reset si tenemos subido ya al repositorio remoto información. El git reset borrará commit y darán conflicto cuando se vaya a subir nuevas actualizaciones

git revert “ID” // deshace los cambios de un commit y genera otro

se crean ramas para segmentar el trabajo entre desarrolladores se puede crear una por cada funcionalidad

git branch //para saber en qué rama estamos

git branch “nombre de rama” //para crear una rama

git checkout “nombre de rama” //para viajar a esa rama

Para fusionar las ramas nos debemos posicionar en la rama que va a recibir los cambios (normalmente la rama main)

git merge “nombre de rama”

Referencias

Video de EDTeam desde 0 los conceptos básicos

https://www.youtube.com/watch?v=jGehuhFhtnE

Video del canal bluuweb

https://www.youtube.com/watch?v=hWglK8nWh60

Web de bluuweb con los conceptos básicos

https://bluuweb.github.io/desarrollo-web-bluuweb/07-01-git/#enlaces

Video canal Bye Code. Ejemplo básico y ejemplo con gitflow

https://www.youtube.com/watch?v=XpulbHt5zZ0&t=2510s

Dibujo en blanco y negro

Descripción generada automáticamente con confianza bajaholaMundo

https://www.youtube.com/watch?v=VdGzPZ31ts8

------------------