Git vs GitHub – ¿Qué es el Control de Versiones y Cómo Funciona?

Lo primero de todo, ¿qué es un sistema de control de versiones?

Los sistemas de control de versiones son una categoría de herramientas de software que ayudan a un equipo de trabajo a gestionar cambios en el código fuente a lo largo del tiempo.

El software de control de versiones realiza un seguimiento de cada modificación del código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden retroceder el reloj y comparar las versiones anteriores del código para ayudar a corregir el error y minimizar la interrupción de todos los miembros del equipo.

1. Es colaborativo

Es más que probable que, sin un sistema de control de versiones en tu metodología habitual, andes enredado junto a tus compañeros de trabajo en una carpeta compartida en un mismo grupo o peor aún, que el *workflow* más eficaz sea el de ir dando voces por la oficina avisando sobre el estado de tu trabajo para que tus compañeros de equipo no arruinen los avances (o tú los suyos).

Bueno, este sistema, además de ineficaz es extremadamente arriesgado, ya que se presenta con cierta propensión a la ejecución de errores. Es como una constante operación a corazón abierto donde tarde o temprano, alguien sobrescribirá los cambios de otra persona.

Como comentábamos anteriormente, los sistemas VCS permiten que todos los miembros del equipo de trabajo pueden operar libremente, en cualquier archivo y cualquier momento.

Una vez finalices tus tareas, el VCS te permitirá fusionar todos los cambios en una versión común de modo que no hay duda en torno a dónde se encuentra la última versión de un archivo o de todo el proyecto. Está en un lugar centralizado, controlado y común a todos: tu VCS.

2. Permite almacenar versiones

Guardar una versión de tu proyecto después de hacer cambios sobre él es sin duda un hábito tan lógico como fundamental, pero sin un VCS, este trámite en apariencia inofensivo, se vuelve tedioso y confuso de inmediato:

¿Cómo nombras las versiones de tus proyectos?

Si eres una persona más o menos organizada, seguramente te atendrás a un esquema de nomenclatura comprensible (si te apañas con "mengano-sl-redesign-2016-11-12-v19"). Sin embargo, tan pronto como comiences a trabajar con variantes (por ejemplo, preparar una versión con el área de cabecera y una sin ella), las posibilidades comenzarán a crecer y es más que probable que acabes perdiendo la pista.

La cuestión más importante: ¿cómo sabes, con precisión, ¿cuáles son las diferencias entre versiones? Francamente, muy pocas personas se toman el tiempo para documentar cuidadosamente cada cambio importante e incluirlo en un archivo README en la carpeta del proyecto.

3. En caso de error, podrás restaurar una versión anterior

Ser capaz de restaurar versiones anteriores de un archivo (o incluso todo el proyecto) sólo significa una cosa: no hay proyecto que pueda perderse debido a un fallo técnico. Si los últimos cambios que has hecho son una catástrofe, simplemente deshazlos con unos pocos clics y retorna a un punto anterior.

Saber esto ya es suficiente para trabajar mucho más relajado, especialmente cuando se trabaja en partes importantes de un proyecto.

4. Podrás entender qué ha ocurrido (en el pasado)

Cada vez que guardes una nueva versión de tu proyecto, tu VCS te pedirá que documentes con una breve descripción lo que ha cambiado. Además (si se trata de un archivo de código / texto), puedes ver qué se ha cambiado exactamente en el contenido del archivo. Esto te ayuda a entender cómo ha evolucionado tu proyecto entre versiones.

5. Actúa como backup

Una funcionalidad adicional de agradecer cuando utilizamos un VCS distribuido como Git es que puede actuar como *backup*. Cada miembro del equipo tiene una versión completa del proyecto en su disco, incluyendo la historia completa del proyecto. Si se da el caso de que el cosmos ha conspirado hoy en tu contra y tu servidor se va a freír espárragos (y sus unidades de respaldo fallan), todo lo que necesitas para la recuperación es uno de los repositorios Git locales de tus compañeros de equipo.

6. ¿Qué es Git?

Git es un Sistema de **Control de Versiones Distribuido** (DVCS) utilizado para guardar diferentes versiones de un archivo (o conjunto de archivos) para que cualquier versión sea recuperable cuando lo desee.

Git también facilita el **registro y comparación de diferentes versiones de un archivo**. Esto significa que los detalles sobre **qué cambió, quién cambió qué, o quién ha iniciado una propuesta, se pueden revisar en cualquier momento.**

7. ¿Qué significa "distribuido"?

El término "distribuido" significa que cuando le instruyes a Git que comparta el directorio de un proyecto, Git no sólo comparte la última versión del archivo, distribuye cada versión que ha registrado para ese proyecto.

Este sistema "distribuido" tiene un marcado contraste con otros sistemas de control de versiones. Ellos sólo comparten cualquier versión individual que un usuario haya explícitamente extraído desde la base de datos central/local.

8. ¿Qué es un Sistema de Control de Versiones?

Un Sistema de Control de Versiones (VCS) se refiere al método utilizado para guardar las versiones de un archivo para referencia futura.
De manera intuitiva muchas personas ya utilizan control de versiones en sus proyectos al renombrar las distintas versiones de un mismo archivo de varias formas como blogScript.js, blogScript_v2.js, blogScript_v3.js, blogScript_final.js, blogScript_definite_final.js, etcétera. Pero esta forma de abordarlo es propensa a errores y poco efectivo para proyectos grupales.
Además, con esta forma de abordarlo, rastrear qué cambió, quién lo cambió y porqué se cambió, es un esfuerzo tedioso.

Sin embargo, para obtener lo mejor de Git, es importante entender cómo Git administra tus archivos.

9. Estados de los archivos en Git

En Git hay tres etapas primarias (condiciones) en las cuales un archivo puede estar: **estado modificado**, **estado preparado**, o **estado confirmado**.

Estado modificado

Un archivo en el estado modificado es un archivo revisado – pero no acometido (sin registrar).

En otras palabras, archivos en el estado modificado son archivos que has modificado, pero no le has pedido explícitamente a Git que controle.

Estado preparado

Archivos en la etapa preparado son archivos modificados que han sido seleccionados – en su estado (versión) actual – y están siendo preparados para ser guardados al repositorio .git durante la próxima instantánea de confirmación.

Una vez que el archivo está preparado implica que has explícitamente autorizado a Git que controle la versión de ese archivo.

Estado confirmado

Archivos en el estado confirmado son archivos que se guardaron en el repositorio .git exitosamente.

Por lo tanto, un archivo confirmado es un archivo en el cual has registrado su versión preparada en el directorio (carpeta) Git.

Nota: El estado de un archivo determina la ubicación donde Git lo colocará.

Ubicación de archivos

Existen tres lugares principales donde pueden residir las versiones de un archivo cuando se hace control de versiones con Git: el **directorio de trabajo**, el **sector de preparación**, o el **directorio Git**.

Directorio de trabajo

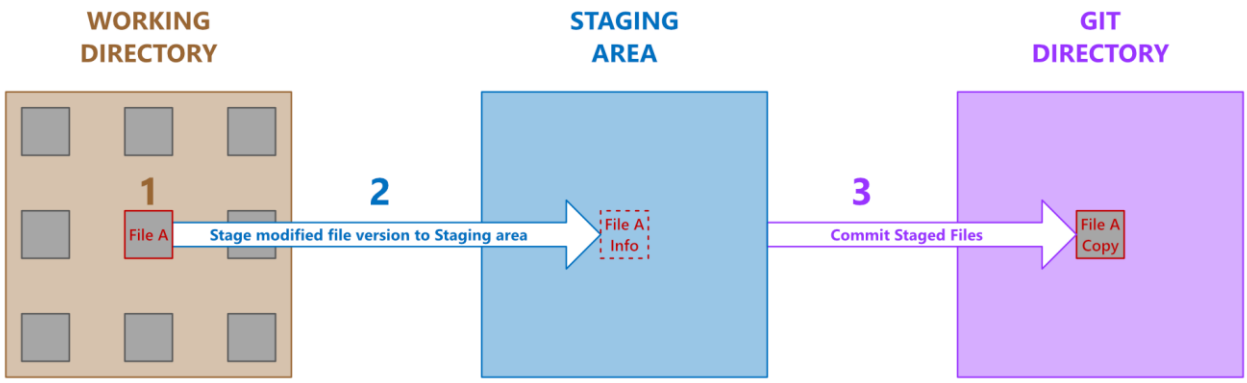
El directorio de trabajo es una carpeta local para los archivos de un proyecto. Esto significa que cualquier carpeta creada en cualquier lugar en un sistema es un directorio de trabajo.

Nota:

- Los archivos en el estado modificado residen dentro del directorio de trabajo.
- El directorio de trabajo es distinto al directorio .git. Es decir, tu creas un directorio de trabajo mientras que Git crea un directorio .git.

10. El flujo de trabajo básico de Git

Trabajar con el Sistema de Control de Versiones Git se ve algo así:



1. Modificar archivos en el directorio de trabajo. Observe que cualquier archivo que modifiques se convierte en un archivo en el ***estado modificado*.**
2. Prepare selectivamente los archivos que quieras confirmar al directorio .git.
   Observe que cualquier archivo que prepares (agregues) a la zona de preparación se convierte en un archivo en el ***estado preparado*.**
   También tenga en cuenta que los archivos preparados todavía no están en la base de datos .git.
   Preparar significa que la información sobre el archivo preparado se incluye en un archivo (llamado "index") en el repositorio .git.
3. Confirme el/los archivos que has preparado en el directorio .git. Esto es, guardar de manera permanente una instantánea de los archivos preparados en la base de datos .git.

Observe que cualquier versión del archivo que confirmes al directorio .git se convierte en un archivo en el *estado confirmado*.

11. Para alojar (o compartir) un repositorio Git en GigHub, siga los siguientes pasos:

Paso 1: Registrar una cuenta de GitHub

El primer paso para comenzar con el alojamiento en GitHub es crear una cuenta personal. Visite la página de registro oficial para registrarse.

Paso 2: Crear un repositorio remoto en GitHub

Después de registrarse, crear un home (un repositorio) en GitHub para el repositorio Git que quieres compartir.

Paso 3: Conectar el directorio Git del proyecto con el repositorio remoto

Una vez que hayas creado un repositorio remoto para tu proyecto, tienes que vincular el directorio .git del proyecto – ubicado localmente en tu sistema – con el repositorio remoto en GitHub.

Para conectar con el repositorio remoto, debes **ubicarte en el directorio raíz** del proyecto que quieras compartir, utilizando tu terminal local, y ejecuta este comando:

```
git remote add origin https://github.com/tuUsuario/tuRepositorio.git
```

Nota:

- Reemplaza tuUsuario en el código de arriba con tu nombre de usuario de GitHub.
  También reemplaza tuRepositorio con el nombre del repositorio remoto al que te quieres conectar.
- El comando de arriba implica que *git* debe *agregar* al proyecto la *URL* especificada como una referencia remota con la cual el directorio local .git puede interactuar.
- La opción origin en el comando de arriba es el nombre predeterminado (un nombre corto) que Git le otorga al servidor que

aloja tu repositorio remoto.
Es decir, **Git** utiliza el nombre corto **origin** en vez de la URL del servidor.

- No es obligatorio quedarse con el nombre predeterminado del servidor. Si prefieres otro nombre que origin, simplemente sustituya el nombre origin en el comando git remote add de arriba por cualquier nombre que prefieras.
- Siempre recuerda que un nombre corto del servidor (por ejemplo, origin) no es nada especial! Sólo existe – localmente – para ayudarte a referenciar fácilmente la URL del servidor. Por lo tanto, siéntete libre de cambiarlo a un nombre corto que puedas referenciar fácilmente.
- Para renombrar cualquier URL remota que exista, utilice el comando git remote rename de esta manera:

  git remote rename nombreURLactual nuevoNombreURL
- Cada vez que clones (descargues) cualquier repo remota, Git automáticamente le pone nombre origin a la URL de la repo. Sin embargo, le puedes especificar un nombre distinto con el comando git clone -o tuNombrePreferido.
- Para ver la URL exacta guardada para los nombres como origin, ejecuta el comando git remote -v.

Paso 4: Confirmar la conexión

Una vez que hayas conectado tu directorio Git con el repositorio remoto, verifica si la conexión fue exitosa ejecutando el comando git remote -v.

Después verifica la impresión para confirmar que la *URL mostrada* sea la misma que la *URL remota* que intentas conectarte.

Nota:

- Ver el artículo "Conectar con SSH" si quieres conectar utilizando la URL SSH en lugar de la URL HTTPS.

- Sin embargo, si no estás seguro de la URL remota que vas a utilizar, echa un vistazo al artículo "¿Qué URL remota debería utilizar?".
- ¿Quieres cambiar tu URL remota? Cambiar la URL de un remoto es una excelente guía.

Paso 5: Subir un repo Git local a un repo remoto

Después de conectar exitosamente tu directorio local con el repositorio remoto puedes comenzar a subir (cargar) tu proyecto local **upstream**.

Cuando estés listo para compartir tu proyecto en otra parte, en cualquier repo remoto, simplemente le dices a Git que suba todos tus confirmaciones, ramas y archivos en tu directorio .git local al repositorio remoto.

La sintaxis del código utilizado para subir (push) un directorio local Git a un repositorio remoto es git push -u nombreRemoto nombreRama.

Esto es, para subir tu directorio local .git y suponiendo que el nombre corto de la URL remota es "origin", ejecuta:

git push -u origin master

Nota:

- El comando de arriba implica que *git* debe *subir* tu rama *master* local a la rama *master* remota ubicada en la URL con nombre *origin*.
- Técnicamente puedes sustituir la opción origin con la URL del repositorio remoto. Recuerda, la opción origin es solo un nombre de la URL que hayas registrado en tu directorio .git local.
- El indicador -u (indicador de referencia upstream/seguimiento) automáticamente vincula la rama local del directorio .git con la rama remota. Esto te permite usar git pull sin ningún argumento.

Paso 6: Confirmar la subida

Por último, vuelve a tu página de repositorio GitHub para confirmar que Git haya subido exitosamente tu directorio Git local al repositorio remoto.

Nota:

- Tal vez tengas que refrescar la página del repositorio remoto para reflejar los cambios.
- GitHub también tiene un servicio gratuito opcional para convertir tu repositorio remoto en una página web funcional. Abajo vamos a ver "cómo" hacerlo.-

Publica tu página web con GitHub pages

Después de subir tu proyecto al repositorio remoto, fácilmente puedes publicarlo en la web de esta forma:

1. Asegurate que el nombre del archivo principal HTML de tu proyecto sea index.html.
2. En el sitio web de la plataforma GitHub ingresa al repositorio del proyecto que quieres publicar y haz click en la **pestaña settings**.
3. Desplaza hacia la sección **GitHub Pages** y cambia la rama **Source** de **none** a **master**.
4. Después aparecerá una notificación que dice "Your site is published at *https://your-username.github.io/your-github-repo-name/*".
5. Ahora puedes ver – y publicitar – tu proyecto en la URL especificada.

   Esta sección solo ha comenzado a tratar el tema de publicar tu proyecto con GitHub. Para aprender más sobre GitHub pages, echa un vistazo a esta documentación "Trabajar con páginas de GitHub Pages".

En resumen

GitHub es una plataforma en línea de alojamiento (o para compartir) repositorios de Git. Nos ayuda crear una avenida para colaborar fácilmente con cualquier persona, en cualquier lugar, en cualquier momento.

Cinco diferencias claves entre Git y GitHub.

Diferencia 1: Git vs. GitHub — Función principal

Git es un sistema de control de versiones distribuido que registra las distintas versiones de un archivo (o conjunto de archivos). Permite a los usuarios acceder, comparar, actualizar, y distribuir cualquiera de las versiones registradas en cualquier momento.

Sin embargo, **GitHub** principalmente es una plataforma de alojamiento para albergar tus repositorios Git en la web. Esto permite a los usuarios mantener sus repositorios remotos privados o abiertos para esfuerzos colaborativos.

Diferencia 2: Git vs. GitHub — Plataforma de operación

Los usuarios instalan y ejecutan Git en sus equipos locales. Esto significa que la mayoría de las operaciones de Git se pueden lograr sin una conexión a internet.

Sin embargo, GitHub es un servicio basado en la web que opera solamente en línea. Esto significa que necesitas estar conectado para hacer cualquier cosa en GitHub.

Diferencia 3: Git vs. GitHub — Creadores

Linus Torvalds comenzó Git en Abril del 2005.

Chris Wanstrath, P. J. Hyett, Tom Preston-Werner, y Scott Chacon fundaron GitHub.com en Febrero 2008.

Diferencia 4: Git vs. GitHub — Mantenedores

En Julio 2005, Linus Torvalds entregó el mantenimiento de Git a Junio C. Hamano — quien ha sido el principal mantenedor desde entonces.

En Octubre 2018, Microsoft compró GitHub.

Diferencia 5: Git vs. GitHub — Competidores

Algunas alternativas populares para Git son Mercurial, Team Foundation Version Control (TFVC), Perforce Helix Core, Apache Subversion, y IBM Rational ClearCase.

Los competidores más cercanos a GitHub son GitLab, Bitbucket, SourceForge, Cloud Source Repositories, y AWS CodeCommit.

Bibliografía:

https://www.freecodecamp.org/espanol/news/git-vs-github-what-is-version-control-and-how-does-it-work/

https://www.overant.com/blog/sistemas-de-control-de-versiones/

Vídeos tutorial:

🌟 ¿Qué es GIT? ¿Qué son los REPOSITORIOS? 2021 🗄 | EXPLICACIÓN FÁCIL 🚀 | Introducción a GIT #1 (youtube.com)

🌟 ¿Cómo instalar GIT? Crea tu primer repositorio 2021 🗄 |EXPLICACIÓN FÁCIL 🚀 | Introducción a GIT #2 (youtube.com)

🚀¿COMO CREAR UNA CUENTA + REPOSITORIO EN GITHUB? 🌟| CONFIGURACIÓN FÁCIL 2021 | Introducción a GIT #3 (youtube.com)

🚀GIT ADD + GIT COMMIT + GIT PUSH TUTORIAL 🌟|CONFIGURACIÓN FÁCIL 2021| Introducción a GIT y GITHUB #4 (youtube.com)