

Control de versiones del código fuente

El control de versiones es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas que ayudan a los equipos de desarrollo a gestionar los cambios en el código fuente a lo largo del tiempo. Este realiza un seguimiento de todas las modificaciones en el código en una base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

Los desarrolladores de software que trabajan en equipos están escribiendo continuamente nuevo código fuente y cambiando el que ya existe. El código de un proyecto, una aplicación o un componente de software normalmente se organiza en una estructura de carpetas o árbol de archivos. Un desarrollador del equipo podría estar trabajando en una nueva función mientras otro desarrollador soluciona un error no relacionado cambiando código. Cada desarrollador podría hacer sus cambios en varias partes del árbol de archivos. Los cambios realizados en una parte del software pueden ser incompatibles con los que ha hecho otro desarrollador que está trabajando al mismo tiempo. Este problema debería detectarse y solucionarse de manera ordenada sin bloquear el trabajo del resto del equipo. El control de versiones ayuda a los equipos a resolver este tipo de problemas al realizar un seguimiento de todos los cambios individuales de cada colaborador y evitar que el trabajo concurrente entre en conflicto.

Para la mayoría de equipos de software, el código fuente es un repositorio del conocimiento de valor incalculable, dado que comprende información sobre el dominio del problema que los desarrolladores han recopilado y perfeccionado con un esfuerzo cuidadoso. El control de versiones protege el código fuente tanto de las catástrofes como del deterioro casual de los errores humanos y las consecuencias accidentales.

El software de control de versiones es una parte esencial del día a día de las prácticas profesionales del equipo de software moderno. Los desarrolladores de software individuales que están acostumbrados a trabajar con un sistema de control de versiones en sus equipos suelen reconocer el valor que el control de versiones les brinda, incluso en los proyectos pequeños en los que trabajan solos.

>Talentos Digitales_



Figura 1. Encuesta de las herramientas utilizadas para el control de versiones en Stack Overflow.

Existen numerosas herramientas para el control de versiones, sin embargo, a lo largo de este curso nos centraremos específicamente en Git dado que hoy en día, es con diferencia, el sistema de control de versiones moderno más utilizado (ver Figuras 1 y 2). Un gran número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Además, este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE.

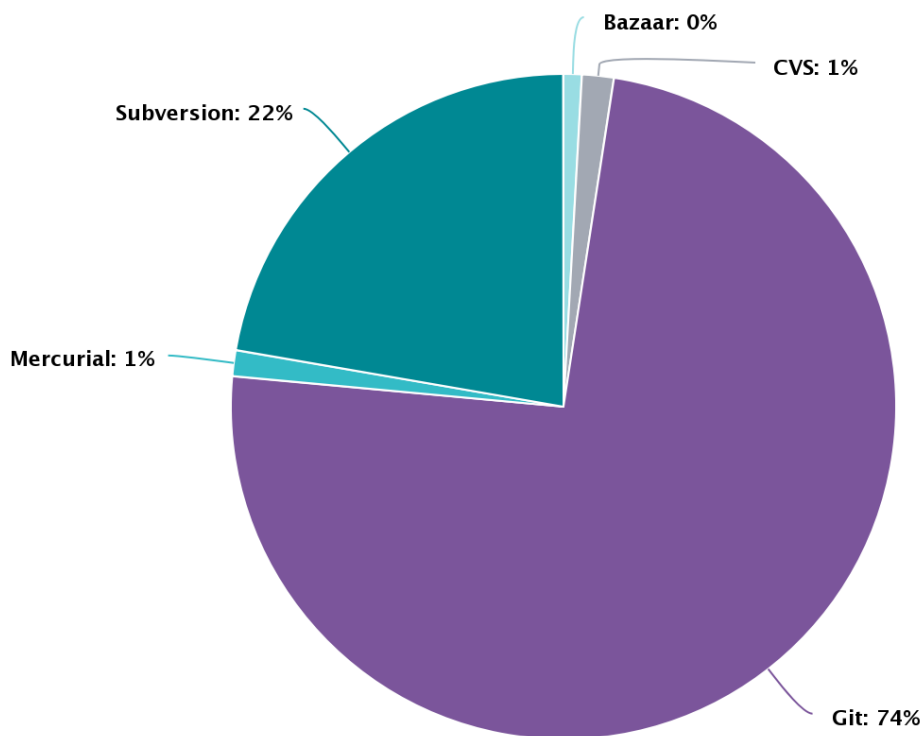


Figura 2. Uso de control de versiones en Ohloh.

Git presenta una arquitectura distribuida, donde en lugar de tener un único espacio para todo el historial de versiones del software, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

Historia de Git

Linus Torvalds, creador de Linux, junto con otros miembros de la comunidad de desarrollo de Linux, crearon y lanzaron Git en 2005. Empezaron el proyecto porque faltaban sistemas de control de versiones gratuitos y de código abierto que pudieran cumplir con sus requisitos para el desarrollo del kernel de Linux. Necesitaban un sistema que pudiera respaldar un esfuerzo de colaboración a gran escala y brindar el rendimiento necesario para llevar a cabo varias tareas al mismo tiempo.

Git entregó ese requerimiento al proporcionar una solución que no dependía de un repositorio centralizado. En su lugar, admitía un flujo de trabajo distribuido, al tiempo que protegía contra la corrupción de archivos. Git también ofreció un diseño simple que admitía el desarrollo de soluciones no lineal.

Fundamentos de Git

Git maneja sus datos como un conjunto de copias instantáneas o fotografías de un sistema de archivos. Cada vez que se confirma un cambio, o se guarda el estado del proyecto, se toma una foto del aspecto de todos los archivos en ese momento y se guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado no se almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya almacenado. Git maneja sus datos como una secuencia de copias instantáneas como muestra la Figura 3. En esta se puede observar la manera en que se guardan los archivos A, B y C en cada una de las versiones del proyecto.

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar, no se necesita información de ningún otro computador de la red. Por ejemplo, para navegar por la historia del proyecto, no es necesario conectarse al servidor para obtener la historia y mostrarla, simplemente se la lee de la base de datos local reduciendo de manera considerable los tiempos de respuesta.

>Talentos Digitales_

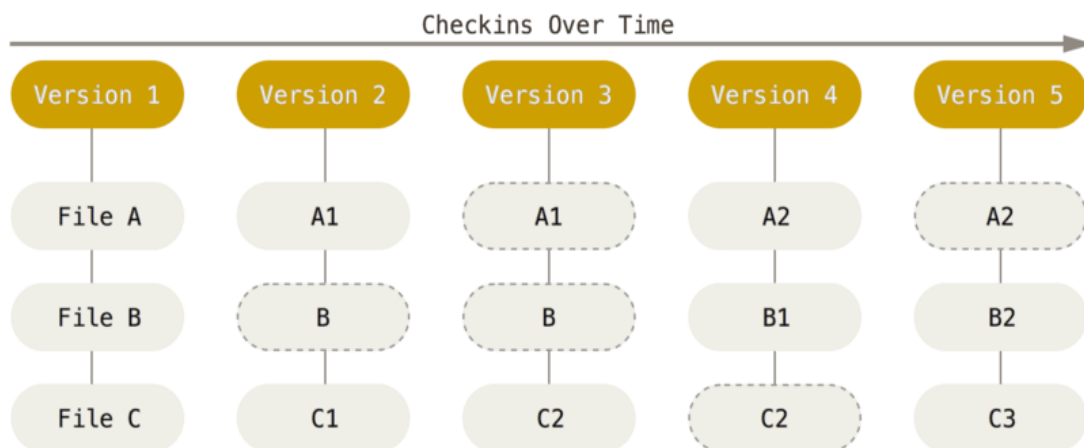


Figura 3. Almacenamiento de datos como instantáneas del proyecto a través del tiempo.

Mecanismo para mantener integridad de los archivos

Todo en Git es verificado mediante una suma de comprobación (checksum en inglés) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo registre. No se puede perder información durante su transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo.

El mecanismo que se usa para generar esta suma de comprobación se conoce como hash SHA-1. Se trata de una cadena de 40 caracteres hexadecimales (0-9 y a-f), y se calcula con base en los contenidos del archivo o estructura del directorio. Un hash SHA-1 se ve de la siguiente forma:

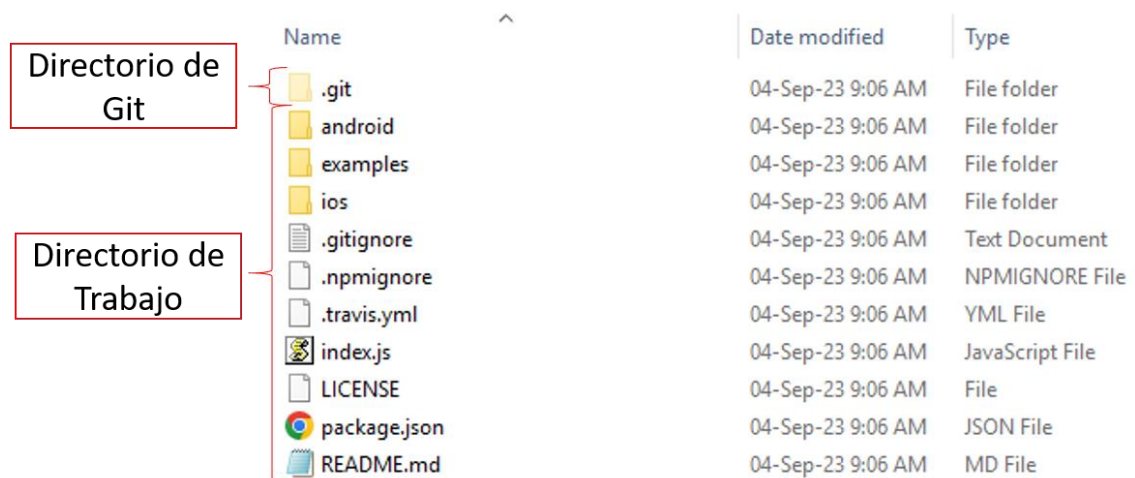
24b9da6552252987aa493b52f8696cd6d3b00373

Estos valores hash se ven por todos lados en Git, porque son usados con mucha frecuencia dado que Git guarda todo no por nombre de archivo sino por el valor hash de sus contenidos.

Partes de un proyecto Git

Un proyecto Git se divide en tres secciones principales: el directorio de Git, el directorio de trabajo y el área de preparación:

- El directorio de Git (Git directory) es donde se almacenan los metadatos y la base de datos para el proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otra computadora.
- El directorio de trabajo (working directory): es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos en el directorio de Git, y se colocan en el disco para que los puedas usar o modificar.
- El área de preparación (staging area): es un archivo, generalmente contenido en el directorio de Git, que almacena información acerca de lo que va a ir en la próxima confirmación.



El diagrama muestra la estructura de un proyecto Git. A la izquierda, hay dos recuadros rojos que etiquetan las carpetas: 'Directorio de Git' para la carpeta '.git' y 'Directorio de Trabajo' para el resto de archivos y carpetas. A la derecha, una tabla resume los detalles de cada ítem.

Name	Date modified	Type
.git	04-Sep-23 9:06 AM	File folder
android	04-Sep-23 9:06 AM	File folder
examples	04-Sep-23 9:06 AM	File folder
ios	04-Sep-23 9:06 AM	File folder
.gitignore	04-Sep-23 9:06 AM	Text Document
.npmignore	04-Sep-23 9:06 AM	NPMIGNORE File
.travis.yml	04-Sep-23 9:06 AM	YML File
index.js	04-Sep-23 9:06 AM	JavaScript File
LICENSE	04-Sep-23 9:06 AM	File
package.json	04-Sep-23 9:06 AM	JSON File
README.md	04-Sep-23 9:06 AM	MD File

Figura 4. Directorios principales de un proyecto Git.

En la Figura 4, se pueden visualizar el directorio de Git y el directorio de trabajo en un proyecto real. La carpeta `.git`, (normalmente se encuentra oculta) es a lo que se llama directorio de Git, la misma contiene la base de datos de los cambios realizado en el proyecto. El resto de archivos y carpetas constituyen el directorio de trabajo.

>Talentos Digitales_

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   .gitignore

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md
```

Figura 5. Directorios principales de un proyecto Git.

La Figura 5 por su parte, muestra el contenido del área de preparación. Como ya se describió anteriormente, esta área es un archivo que se encuentra dentro del directorio de Git. La misma detalla los cambios que serán almacenados en la próxima confirmación o commit (verde), y lo excluido para el commit (rojo).

Estados de un archivo en un proyecto

Los archivos dentro de un proyecto se pueden encontrar en tres estados principales:

- Confirmado (committed): significa que los datos están almacenados de manera segura en la base de datos local.
- Modificado (modified): significa que se ha modificado el archivo, pero todavía no se ha confirmado en la base de datos
- Preparado (staged). significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

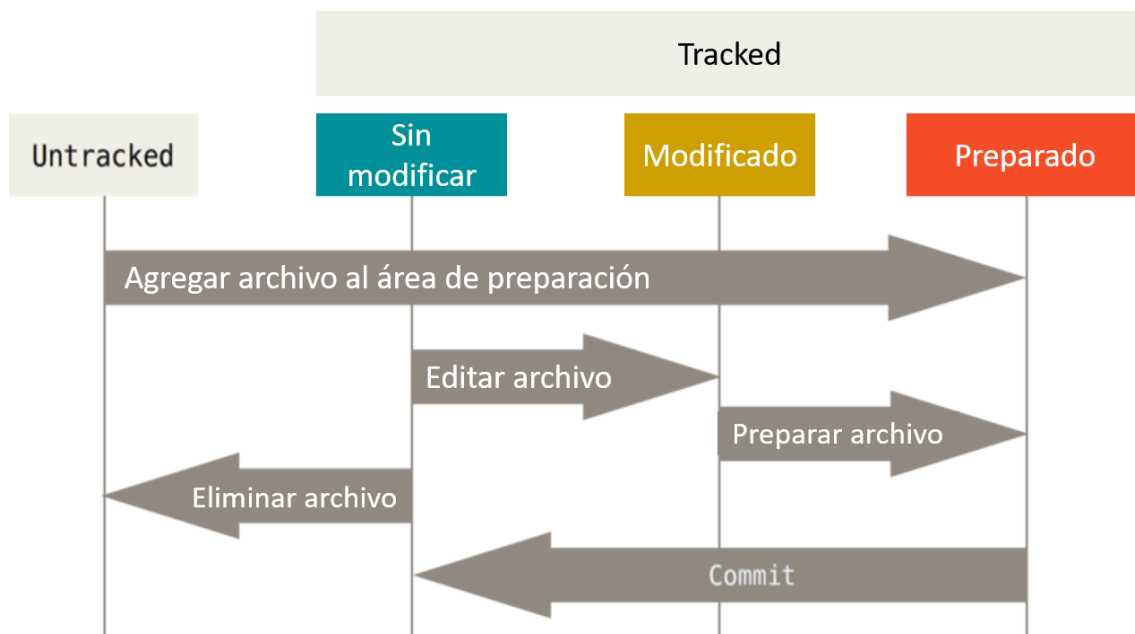


Figura 6. El ciclo de vida del estado de los archivos.

Además de estos tres estados principales, cabe mencionar que antes de realizar el commit cada archivo del repositorio también tiene dos estados adicionales: rastreados y sin rastrear. Los rastreados (tracked files) son todos aquellos archivos que están en la última instantánea del proyecto; ya sea sin modificar, modificados o preparados. Los archivos sin rastrear (untracked files) son todos los demás, es decir, cualquier otro archivo en el directorio de trabajo que no está en la última instantánea y que no esté en el área de preparación (staging area). La Figura 6 muestra las acciones que se pueden realizar con un archivo de acuerdo a su estado. Finalmente, una vez que hayan sido movidos todos los cambios al área de preparación, se procede con la confirmación de los cambios.

Guardando cambios en el Repositorio

Para conservar cambios dentro del proyecto, que implica la creación y almacenamiento de la foto o instantánea de los cambios efectuados, es necesario realizar una confirmación o commit. Se le llama commit al conjunto de cambios que se desea almacenar en la base de datos del proyecto. Para ello, se debe mover los archivos sin rastrear y modificados al área de preparación (staging area) y posteriormente confirmarlos.

El proceso de confirmación debe incluir un mensaje que describa de manera sintética cuáles fueron los cambios realizados. Por ejemplo, si se modificaron archivos .css y html para presentar fuentes distintas entre la página de inicio y la de Quiénes somos, un mensaje podría ser: *Actualización fuentes página de inicio y Quiénes somos.*

>Talentos Digitales_

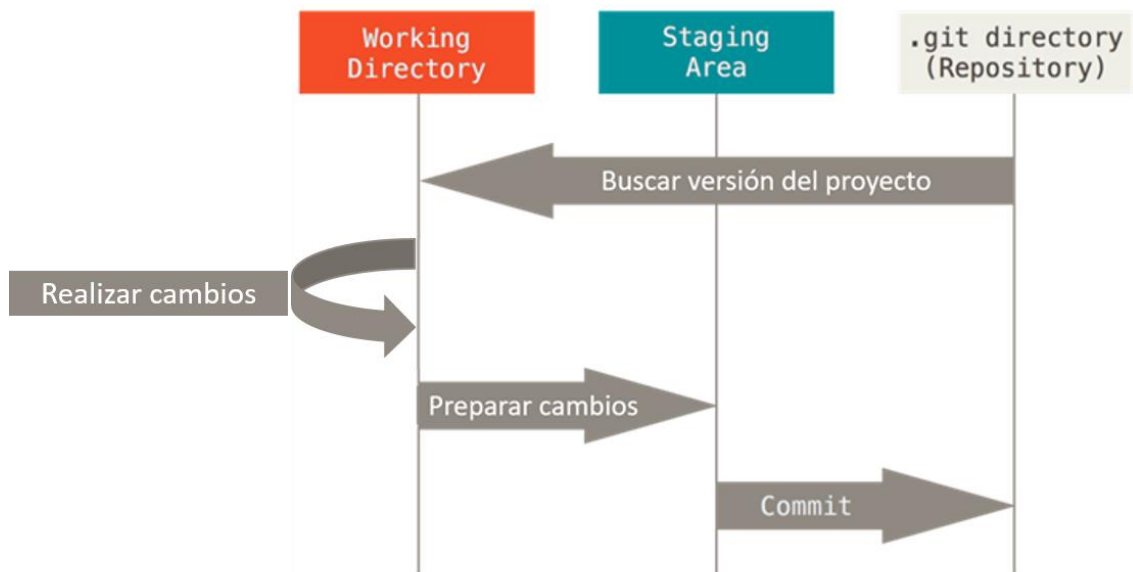


Figura 7. Flujo de trabajo en Git.

Resumiendo, se puede utilizar la Figura 7 para describir un flujo de trabajo básico en Git de la siguiente manera:

1. Modificar una serie de archivos en el directorio de trabajo.
2. Preparar los archivos, añadiéndolos al área de preparación.
3. Confirmar los cambios, lo que implica tomar los archivos tal y como están en el área de preparación y almacenar esa copia instantánea de manera permanente en el directorio de Git.

Repositorios remotos

Prácticamente todo el trabajo relacionado con el control de versiones ocurre en el repositorio local: preparación (staging), confirmación (commit), visualización del estado o del registro/historial, etc. Además, si uno es la única persona que trabaja en el proyecto es probable que nunca se necesite configurar un repositorio remoto, sólo cuando se trata de compartir datos con compañeros de equipo es donde entra en juego un repositorio remoto. Los repositorios remotos son versiones del proyecto que están hospedadas en Internet.

Técnicamente, un repositorio remoto no se diferencia de uno local: contiene ramas, commits y etiquetas como un repositorio local. A diferencia de un repositorio local, uno remoto no tiene dicho directorio de trabajo, solo consta de la carpeta del repositorio *.git*.

Es importante enfatizar que el trabajo real en su proyecto solo ocurre en su repositorio local, todas las modificaciones deben realizarse y confirmarse localmente. Luego, esos cambios se pueden cargar en un repositorio remoto para poder compartirlos con el equipo. Los repositorios remotos sólo están pensados como un medio para compartir e intercambiar código entre desarrolladores, no para trabajar realmente con archivos.

Para poder colaborar en cualquier proyecto Git se necesita saber cómo gestionar repositorios remotos. Puedes tener varios de ellos, y en cada uno tendrás generalmente permisos de solo lectura o de lectura y escritura. Colaborar con otras personas implica gestionar estos repositorios remotos enviando y trayendo datos de ellos cada vez que se necesite compartir el trabajo. Gestionar repositorios remotos incluye saber cómo añadir un repositorio remoto, eliminar los remotos que ya no son válidos, gestionar cambios remotos, entre otros. La Figura 8 muestra las operaciones Git utilizadas para interactuar con un repositorio remoto.

>Talentos Digitales_

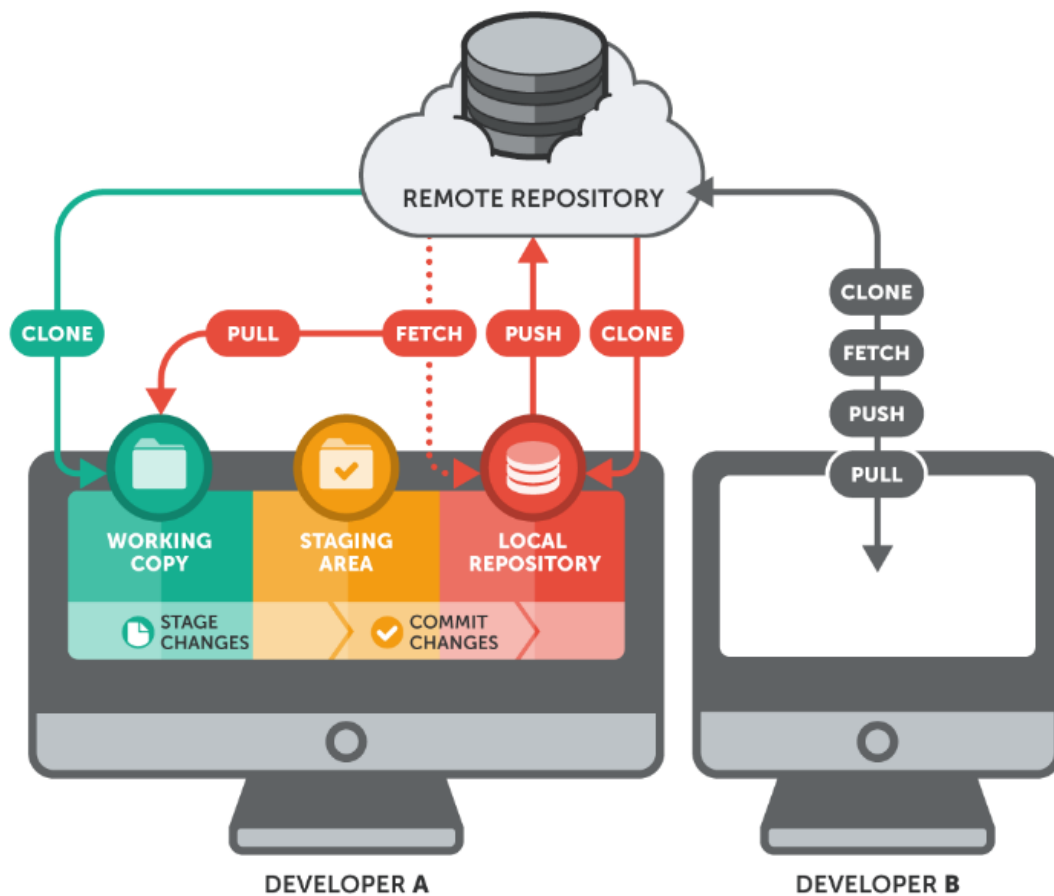


Figura 8. Operaciones Git de un repositorio remoto.

Bibliografía

- <https://www.atlassian.com/es/git/tutorials/what-is-version-control>
- <https://git-scm.com/book/es/v2>.
- <https://www.techtarget.com/searchitoperations/definition/Git>
- <https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/>
- <https://openhub.net/repositories/compare>
- <https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction>