# Control de versiones

El control de versiones en Git se refiere a la práctica de rastrear y administrar cambios en archivos y códigos en un entorno de desarrollo de software colaborativo. Git es un sistema de control de versiones distribuido que permite que varios desarrolladores trabajen en el mismo proyecto simultáneamente mientras mantiene un historial de todos los cambios realizados en el código base.

A continuación se muestran algunos conceptos y características clave del control de versiones en Git:

1. \*\*Repositorio (Repo)\*\*: Un repositorio Git es una colección de archivos y directorios junto con el historial completo de cambios realizados en esos archivos. Hay repositorios locales en las máquinas de los desarrolladores individuales y hay repositorios remotos alojados en plataformas como GitHub, GitLab o Bitbucket.

2. \*\*Commits\*\*: una confirmación es una instantánea de los cambios realizados en un conjunto de archivos en un momento específico. Cada confirmación tiene un identificador único llamado hash e incluye un mensaje de confirmación que describe los cambios.

3. \*\*Sucursales\*\*: Git permite a los desarrolladores trabajar en diferentes funciones o corregir errores simultáneamente mediante la creación de ramas. Cada rama es una línea de desarrollo independiente y los cambios realizados en una rama no afectan a las demás hasta que se fusionan.

4. \*\*Fusionar\*\*: Fusionar es el proceso de combinar cambios de una rama en otra. Se usa comúnmente para integrar nuevas funciones o corregir errores en la rama principal (generalmente llamada "maestra" o "principal").

5. \*\*Solicitudes de extracción (PR)\*\*: en un entorno colaborativo, los desarrolladores suelen crear solicitudes de extracción para proponer cambios en la rama principal. Una solicitud de extracción permite a otros miembros del equipo revisar el código, discutir los cambios y, finalmente, fusionar los cambios en la rama principal.

6. \*\*Resolución de conflictos\*\*: Git proporciona mecanismos para resolver conflictos que ocurren cuando dos o más desarrolladores realizan cambios conflictivos en la misma parte de un archivo. Los desarrolladores deben resolver estos conflictos antes de fusionar sus cambios.

7. \*\*Historial\*\*: Git mantiene un historial completo de todas las confirmaciones y cambios realizados en un proyecto. Este historial permite rastrear quién realizó cambios, cuándo se realizaron y por qué se realizaron. Esto puede resultar invaluable para depurar, auditar y comprender la evolución del proyecto a lo largo del tiempo.

8. \*\*Distribuido\*\*: Git es un sistema de control de versiones distribuido, lo que significa que cada desarrollador tiene su propia copia local de todo el repositorio. Esto permite a los desarrolladores trabajar sin conexión y de forma independiente, lo que lo convierte en un sistema de colaboración sólido y flexible.

Git se utiliza ampliamente en la industria del desarrollo de software y es una herramienta esencial para gestionar el código de forma colaborativa y organizada. Ayuda a los equipos a trabajar de manera más eficiente, reducir los conflictos y mantener un historial completo de sus proyectos.

# Repositorios

En Git, un repositorio, a menudo abreviado como "repo", se refiere a una estructura de datos que almacena todos los archivos, directorios y metadatos de un proyecto específico, junto con el historial completo de cambios realizados en esos archivos. Se puede considerar un repositorio Git como un contenedor o una base de datos que realiza un seguimiento de cada versión de cada archivo dentro del proyecto, lo que permite a los desarrolladores colaborar en el proyecto, realizar un seguimiento de los cambios y gestionar el proceso de desarrollo de forma eficaz.

Aquí hay una definición más detallada de un repositorio Git:

Un repositorio Git es un directorio o carpeta en una computadora local o remota que contiene los siguientes componentes:

1. \*\*Archivos versionados\*\*: El repositorio almacena todos los archivos y directorios relacionados con el proyecto. Git realiza un seguimiento de cada archivo dentro del repositorio y cualquier cambio realizado en estos archivos se registra como versiones diferentes. Git utiliza un sistema direccionable por contenido para almacenar y administrar eficientemente estas versiones.

2. \*\*Historial de confirmaciones\*\*: Git mantiene un historial detallado de todos los cambios realizados en los archivos en el repositorio. Cada cambio se registra como una "confirmación", que incluye un identificador único (hash), el nombre y la dirección de correo electrónico del autor, una marca de tiempo y un mensaje de confirmación que describe el propósito del cambio.

3. \*\*Sucursales\*\*: los repositorios de Git suelen tener varias ramas, que son líneas de desarrollo separadas. Las ramas permiten a los desarrolladores trabajar en diferentes funciones o correcciones de forma independiente. La rama principal, a menudo denominada "maestra" o "principal", normalmente representa la versión estable y lista para producción del proyecto.

4. \*\*Conexiones remotas\*\*: Además de los repositorios locales, los repositorios Git se pueden conectar a repositorios remotos alojados en servicios como GitHub, GitLab o Bitbucket. Esto permite que varios desarrolladores colaboren en un proyecto, compartan sus cambios y sincronicen su trabajo.

5. \*\*Archivos de configuración\*\*: Los repositorios de Git pueden incluir archivos de configuración como `.gitignore` (para especificar qué archivos deben excluirse del control de versiones) y `.gitattributes` (para definir el manejo de texto y archivos binarios).

6. \*\*Hooks\*\*: Git permite a los desarrolladores definir scripts o ganchos personalizados que se pueden ejecutar antes o después de ciertas acciones de Git, como confirmaciones o fusiones. Estos ganchos se pueden utilizar para diversos fines, incluidas las pruebas y la implementación automatizadas.

Un repositorio Git es fundamental para el proceso de control de versiones en Git, ya que proporciona una ubicación centralizada para almacenar, rastrear y administrar todo el historial y los archivos del proyecto. Los desarrolladores pueden clonar repositorios en sus máquinas locales, realizar cambios, crear nuevas ramas y compartir su trabajo con otros a través del repositorio. Este enfoque colaborativo y organizado para el control de versiones es una de las razones clave de la popularidad de Git en la comunidad de desarrollo de software.

# Ramas

En Git, una rama es una línea paralela de desarrollo dentro de un repositorio Git. Representa un espacio de trabajo separado y aislado donde los desarrolladores pueden realizar cambios en el código base independientemente de la rama principal o de otras ramas. Cada rama tiene su propio historial de confirmaciones y los cambios realizados en una rama no afectan a otras ramas hasta que se fusionan. Las ramas son un concepto fundamental y poderoso en Git, que permite el desarrollo de software colaborativo y organizado. Aquí hay una definición más detallada de una rama en Git:

Una rama en Git es:

1. \*\*Un puntero a una confirmación\*\*: en Git, una rama es esencialmente un puntero o referencia móvil que apunta a una confirmación específica en el historial de confirmaciones del repositorio. Esta confirmación se conoce como "punta" o "cabeza" de la rama. La confirmación a la que se apunta representa el último estado del código base en esa rama.

2. \*\*Espacio de trabajo aislado\*\*: cada rama proporciona un espacio de trabajo aislado para realizar cambios en los archivos y el código del proyecto. Los desarrolladores pueden trabajar en sus propias ramas independientemente de otros miembros del equipo, lo que ayuda a prevenir conflictos y permite el desarrollo paralelo de diferentes funciones o correcciones de errores.

3. \*\*Historial de confirmaciones independiente\*\*: cada rama mantiene su propio historial de confirmaciones independiente. Esto significa que la serie de confirmaciones realizadas en una rama están separadas de las de otra rama. El historial de cada rama incluye las confirmaciones exclusivas de esa rama, junto con las confirmaciones que hereda de su rama principal cuando se creó.

4. \*\*Creación y cambio\*\*: los desarrolladores pueden crear nuevas ramas a partir de una rama existente, a menudo denominada rama "principal". Una vez que se crea una rama, los desarrolladores pueden cambiar a ella para comenzar a trabajar en sus cambios. Git proporciona comandos como `git branch` y `git checkout` (o `git switch` en versiones más nuevas de Git) para administrar ramas y cambiar entre ellas.

5. \*\*Fusión\*\*: las ramas se utilizan normalmente para desarrollar nuevas funciones, corregir errores o trabajar en cambios experimentales. Una vez que los cambios en una rama se completan y prueban, se pueden fusionar nuevamente en la rama principal o en otra rama de destino. Git proporciona varias estrategias de fusión, como fusiones de avance rápido o confirmaciones de fusión, para combinar los cambios de una rama en otra.

6. \*\*Convención de nomenclatura\*\*: las sucursales suelen recibir nombres descriptivos para indicar su propósito. Las convenciones de nomenclatura comunes incluyen "función/nombre de rama" para el desarrollo de funciones, "corrección de errores/nombre de rama" para correcciones de errores y "revisión/nombre de rama" para correcciones críticas en una versión estable.

Las sucursales en Git brindan una forma flexible y organizada para que los equipos colaboren en proyectos de desarrollo de software. Permiten trabajo simultáneo en diferentes aspectos de un proyecto, control de versiones de funciones o correcciones y la capacidad de experimentar con cambios sin afectar el código base principal.

# Estados de Git

En Git, "estado" se refiere al estado o condición actual de los archivos dentro de un repositorio de Git. El comando de estado de Git se utiliza para mostrar información sobre el estado de los archivos rastreados y no rastreados en su directorio de trabajo en relación con el sistema de control de versiones de Git. Proporciona un resumen de los cambios que se han realizado pero que aún no se han comprometido. El comando de estado es una herramienta útil para comprender el estado de su proyecto y qué acciones se deben tomar. Aquí hay un desglose de los estados clave que puede encontrar en Git:

1. \*\*Sin seguimiento\*\*: estos son archivos en su directorio de trabajo que actualmente no están siendo rastreados por Git. Git no tiene conocimiento de ningún cambio realizado en archivos sin seguimiento. Para comenzar a rastrear estos archivos, puede usar el comando `git add`.

2. \*\*Modificado\*\*: Los archivos modificados son archivos que se han modificado desde la última confirmación. Git está al tanto de estos cambios, pero aún no se han confirmado. Puede preparar y confirmar estos cambios utilizando los comandos `git add` y `git commit`, respectivamente.

3. \*\*Preparados\*\*: Los archivos preparados son archivos modificados que usted ha marcado explícitamente para incluirlos en la siguiente confirmación. Organiza archivos usando el comando `git add`. Una vez preparados, estos cambios están listos para cometerse.

4. \*\*Adelante de la sucursal\*\*: este estado indica que su sucursal local tiene confirmaciones que aún no se han enviado al repositorio remoto. Significa que su sucursal está por delante de la sucursal correspondiente en el control remoto.

5. \*\*Detrás de la sucursal\*\*: por el contrario, este estado significa que su sucursal local está detrás de la sucursal correspondiente en el repositorio remoto. Implica que hay confirmaciones en la sucursal remota que no ha obtenido ni fusionado en su sucursal local.

6. \*\*Rama divergente\*\*: si su rama tiene confirmaciones delante y detrás de la rama remota correspondiente, se considera que ha divergido de la rama remota.

7. \*\*Aún no hay confirmaciones\*\*: en un nuevo repositorio o rama, es posible que veas este estado cuando aún no se hayan realizado confirmaciones. Es una indicación de que no hay historial de confirmaciones en la rama actual.

Para verificar el estado de su repositorio Git, puede usar el comando `git status`. Cuando ejecuta `git status`, Git le proporcionará un resumen del estado actual de su directorio de trabajo, indicando qué archivos se modifican, se preparan o no se rastrean, y si su rama está por delante, por detrás o se ha desviado del control remoto. rama.

Comprender el estado de Git es esencial para administrar su trabajo, ya que le ayuda a realizar un seguimiento de los cambios, decidir qué archivos enviar y saber cuándo enviar cambios a un repositorio remoto o extraer cambios de él para mantenerse sincronizado con sus colaboradores.

# Ventajas de usar Git

Git, como sistema de control de versiones distribuido, ofrece numerosas ventajas que lo convierten en una opción popular para gestionar el código fuente y colaborar en proyectos de desarrollo de software. Estas son algunas de las ventajas clave de Git:

1. \*\*Desarrollo distribuido\*\*: Git permite que cada desarrollador tenga su propia copia de todo el repositorio del proyecto, incluido su historial completo. Esto permite a los desarrolladores trabajar de forma independiente, incluso sin conexión, y colaborar de forma eficaz, al tiempo que reduce el riesgo de pérdida de datos.

2. \*\*Bifurcación y fusión\*\*: Git hace que la ramificación y la fusión sean sencillas. Los desarrolladores pueden crear ramas para diferentes funciones, correcciones de errores o experimentos, trabajar en ellas por separado y luego fusionar sus cambios nuevamente en la rama principal. Esto promueve el desarrollo paralelo y minimiza los conflictos.

3. \*\*Velocidad y eficiencia\*\*: Git es increíblemente rápido y operaciones como confirmar, bifurcar y fusionar generalmente se completan en segundos. Sus mecanismos de compresión y almacenamiento de datos lo hacen eficiente, incluso para proyectos grandes.

4. \*\*Integridad de los datos\*\*: Git utiliza una función hash criptográfica (SHA-1) para asignar un identificador único a cada confirmación. Esto garantiza la integridad de los datos, ya que cualquier cambio en el contenido de la confirmación da como resultado un hash diferente, lo que hace prácticamente imposible alterar las confirmaciones históricas sin detección.

5. \*\*Colaboración\*\*: Git simplifica la colaboración entre desarrolladores. Admite repositorios remotos alojados en plataformas como GitHub, GitLab y Bitbucket, lo que permite a los equipos compartir código, colaborar en proyectos y gestionar el control de acceso fácilmente.

6. \*\*Historial y control de versiones\*\*: Git mantiene un historial detallado de todos los cambios realizados en el proyecto, lo que facilita el seguimiento de quién realizó qué cambios, cuándo se realizaron y por qué. Esto es invaluable para depurar, auditar y comprender la evolución de un proyecto.

7. \*\*Deshacer y revertir\*\*: Git le permite revertir cambios y retroceder fácilmente a estados anteriores del código base. Esto proporciona una red de seguridad para experimentar con nuevas funciones o corregir errores sin perder datos.

8. \*\*Área de preparación (índice)\*\*: el área de preparación de Git le permite preparar cambios de forma selectiva antes de confirmarlos. Este control detallado le permite confirmar solo los cambios que desea, lo que hace que sus confirmaciones sean más enfocadas y comprensibles.

9. \*\*Código abierto y ampliamente adoptado\*\*: Git es de código abierto, lo que significa que está disponible gratuitamente y tiene una comunidad de usuarios grande y activa. Se adopta ampliamente en la industria del desarrollo de software y tiene un rico ecosistema de herramientas e integraciones.

10. \*\*Flujos de trabajo personalizables\*\*: Git es muy flexible y admite varias estrategias de flujo de trabajo, incluidos Gitflow, GitHub Flow y GitLab Flow. Esta adaptabilidad permite a los equipos adaptar su proceso de desarrollo a sus necesidades específicas.

11. \*\*Seguridad\*\*: Git proporciona sólidas funciones de seguridad, que incluyen cifrado de datos, autenticación y mecanismos de control de acceso. Esto lo hace adecuado para proyectos con código confidencial o propietario.

12. \*\*Extensibilidad\*\*: Git se puede ampliar mediante el uso de scripts personalizados, enlaces y herramientas de terceros. Esta extensibilidad le permite automatizar tareas e integrar Git con otras herramientas de desarrollo y CI/CD.

En resumen, las ventajas de Git se derivan de su eficiencia, flexibilidad, naturaleza distribuida y sólidas capacidades de control de versiones. Se ha convertido en un estándar de la industria para administrar el código fuente y colaborar en proyectos de software, y sus características permiten a los desarrolladores trabajar de manera más efectiva y eficiente.

# Comandos basicos de Git

Los comandos básicos en Git son las operaciones fundamentales que necesitas para trabajar con Git y administrar el control de versiones en tus proyectos. Estos comandos forman la base de la funcionalidad de Git. Estos son los comandos básicos esenciales de Git:

1. \*\*git init\*\*: Inicializa un nuevo repositorio Git en el directorio actual, creando un directorio `.git` oculto para almacenar los datos del repositorio.

2. \*\*git clone\*\*: crea una copia de un repositorio remoto en su máquina local, lo que le permite trabajar con el código localmente.

3. \*\*git add\*\*: cambios en etapas para la confirmación. Puede usar `git add` para especificar qué cambios (archivos o partes de archivos) desea incluir en la próxima confirmación.

4. \*\*git commit\*\*: registra los cambios preparados como una nueva confirmación en el historial de Git, junto con un mensaje de confirmación que describe los cambios.

5. \*\*git status\*\*: muestra el estado actual de su directorio de trabajo, indicando qué archivos se modifican, se preparan o no se rastrean.

6. \*\*git diff\*\*: muestra las diferencias entre el directorio de trabajo y la última versión confirmada.

7. \*\*git log\*\*: muestra el historial de confirmaciones y muestra información sobre cada confirmación, como el hash de confirmación, el autor, la fecha y el mensaje de confirmación.

8. \*\*git branch\*\*: enumera todas las ramas en el repositorio e indica la rama actualmente desprotegida con un asterisco. También puede crear, eliminar y cambiar de rama usando este comando.

9. \*\*git checkout\*\*: cambia entre ramas y confirmaciones. Le permite navegar por el historial de su proyecto y crear nuevas ramas.

10. \*\*git merge\*\*: combina cambios de una rama a otra, normalmente se utiliza para integrar ramas de funciones en la rama principal.

11. \*\*git pull\*\*: recupera cambios de un repositorio remoto y los fusiona en la rama actual.

12. \*\*git push\*\*: carga cambios locales a un repositorio remoto, actualizando su rama con sus últimas confirmaciones.

Estos comandos básicos de Git cubren las acciones esenciales que realizará cuando trabaje con Git en el día a día. Le permiten crear, modificar, confirmar y compartir su código de manera efectiva mientras realiza un seguimiento de los cambios y colabora con otros. A medida que se familiarice con Git, podrá explorar comandos y opciones adicionales para tareas y flujos de trabajo más avanzados.