GIT & GITHUB

1.1. Introducción a Git

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor.

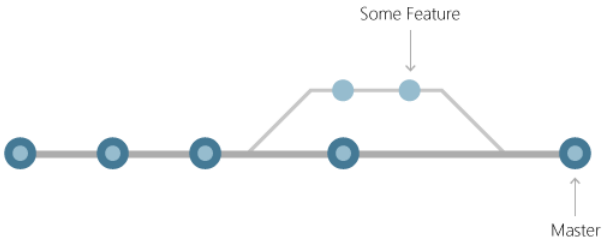
La flexibilidad y popularidad de Git lo convierten en una excelente opción para cualquier equipo. Muchos desarrolladores y graduados universitarios ya saben cómo usar Git. La comunidad de usuarios de Git ha creado recursos para entrenar a los desarrolladores y la popularidad de Git facilita recibir ayuda cuando se necesita. Casi todos los entornos de desarrollo tienen compatibilidad con Git y las herramientas de línea de comandos de Git implementadas en todos los sistemas operativos principales.

Aspectos básicos de Git

Cada vez que se guarda el trabajo, Git crea una confirmación. Una confirmación es una instantánea de todos los archivos en un momento dado. Si un archivo no ha cambiado de una confirmación a la siguiente, Git usa el archivo almacenado anteriormente. Este diseño difiere de otros sistemas que almacenan una versión inicial de un archivo y mantienen un registro de las diferencias a lo largo del tiempo.

Las confirmaciones crean vínculos a otras confirmaciones, formando un gráfico del historial de desarrollo. Es posible revertir el código a una confirmación anterior, inspeccionar cómo cambian los archivos de una confirmación a la siguiente y revisar información como dónde y cuándo se realizaron los cambios. Las confirmaciones se identifican en Git mediante un hash criptográfico único del contenido de la confirmación. Dado que todo tiene hash, es imposible realizar cambios, perder la información o dañar los archivos sin que Git lo detecte.

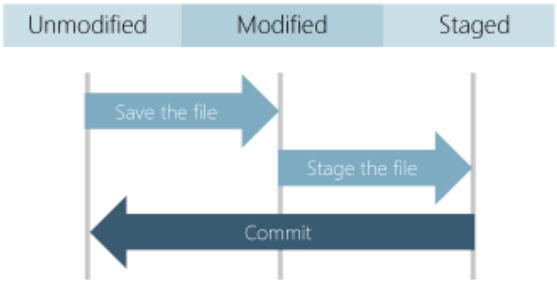
Ramas



Cada desarrollador guarda los cambios en su propio repositorio de código local. Como resultado, puede haber muchos cambios diferentes basados en la misma confirmación. Git proporciona herramientas para aislar los cambios y volver a combinarlos posteriormente. Las ramas, que son punteros ligeros para el trabajo en curso, administran esta separación. Una vez finalizado el trabajo creado en una rama, se puede combinar de nuevo en la rama principal (o troncal) del equipo.

Archivos y confirmaciones

Los archivos de Git se encuentran en uno de estos tres estados: modificados, almacenados provisionalmente o confirmados. Cuando se modifica un archivo por primera vez, los cambios solo existen en el directorio de trabajo. Todavía no forman parte de una confirmación ni del historial de desarrollo. El desarrollador debe almacenar provisionalmente los archivos modificados que se incluirán en la confirmación. El área de almacenamiento provisional contiene todos los cambios que se incluirán en la siguiente confirmación. Una vez que el desarrollador esté satisfecho con los archivos almacenados provisionalmente, los archivos se empaquetan como una confirmación con un mensaje que describe lo que ha cambiado. Esta confirmación pasa a formar parte del historial de desarrollo.



El almacenamiento provisional permite a los desarrolladores elegir qué cambios de archivo se guardarán en una confirmación para desglosar los cambios grandes en una serie de confirmaciones más pequeñas. Al reducir el ámbito de las confirmaciones, es más fácil revisar el historial de confirmaciones para buscar cambios de archivo específicos.

Ventajas de Git

Desarrollo simultáneo

Todos los usuarios tienen su propia copia local de código y pueden trabajar simultáneamente en sus propias ramas. Git funciona sin conexión, ya que casi todas las operaciones son locales.

Versiones de lanzamiento más rápidas

Las ramas permiten un desarrollo flexible y simultáneo. La rama principal contiene código estable y de alta calidad desde el que publica. Las ramas de características contienen trabajo en curso y se combinan con la rama principal tras la finalización. Al separar la rama de versión del desarrollo en curso, es más fácil administrar código estable y enviar actualizaciones más rápidamente.

Integración incorporada

Debido a su popularidad, Git se integra en la mayoría de las herramientas y productos. Todos los IDE principales tienen compatibilidad integrada con Git y muchas herramientas admiten la integración y la implementación continuas, las pruebas automatizadas, el seguimiento de los elementos de trabajo, las métricas y la integración de características de informes con Git. Esta integración simplifica el flujo de trabajo diario.

Sólido soporte técnico de la comunidad

Git es de código abierto y se ha convertido en el estándar de facto para el control de versiones. No hay escasez de herramientas y recursos disponibles para que los equipos aprovechen. El volumen de soporte técnico de la comunidad para Git en comparación con otros sistemas de control de versiones facilita recibir ayuda cuando se necesita

Git funciona con cualquier equipo

Utilizar Git con una herramienta de administración de código fuente aumenta la productividad de un equipo al fomentar la colaboración, aplicar directivas, automatizar procesos y mejorar la visibilidad y la rastreabilidad del trabajo. El equipo puede decidirse por herramientas individuales para el control de versiones, el seguimiento de los elementos de trabajo y la integración e implementación continuas. O bien, pueden elegir una solución como GitHub o Azure DevOps que admita todas estas tareas en un solo lugar.

1.2. Comandos básicos de Git

clear—Limpia pantalla de la terminal git bash.
git status—Muestra estado de la rama actual.
git log --oneline–Muestra commits en formateado a una línea por commit.
git add . or specific files—Añade todo o archivos especificos al staging area.
git commit -m "message"—Crear commit con mensaje.
git branch—Muestra todas las ramas.
git branch branch_name—Crea una nueva rama.
git checkout branch_name—Cambiar a otra rama.
git checkout -b branch_name—Crea y cambia a una rama nueva.
git switch branch_name —Cambiar a otra rama.
git commit -am "message"—añade todos los archivos y crea un nuevo commit con mensaje.
git switch -c branch_name— Crea y cambia a una rama nueva.
git ls-files–Ver archivos que están en el staging area.
git rm file_name.ext– Eliminar archivo.
git checkout file_to_undo_changes.ext–regresa a la última modificación de un archivo.
git restore file_name.ext or .–Restaura archivo o archivos especificos.
git stash –Guardar cambios sin haber hecho commit
git stash list–muestra stashes.
git stash push -m "message"–Poner mensaje a Stash.
git stash drop n_stash–Elmina n_stash de lista de stashes.
git stash clear–Elimina todos los stashes.

1.3. Repositorios en GitHub (Ramas, Uniones de ramas, Conflictos)

git remote add origin URL .—Conecta repositorio remoto a uno local.
git push origin branch_name –Actualiza repositorio remoto, desde rama local.
git remote—Muestra las direcciones remotas conectadas al repositorio local.
git branch -a—Muestra todas las ramas incluyendo las remotas
git branch -r—Muestra todas las ramas remotas
git remote show origin—Muestra detalles de un origen remoto.
git fetch origin—Trae los cambios remotos al staging area del repositorio local.
git pull origin branch_name—Actualiza rama local, git fetch y git merge.
git clone URL—clonar repositorio remoto en un repositorio local.
git merge branch_name—Une una rama con la rama en la que te encuentras.