**Introducción to GIT**

GIT es un sistema de control de versiones utilizado principalmente para el seguimiento de cambios en archivos de código fuente y colaboración en proyectos de software.

GIT fue creado por Linus Torvalds en 2005, el mismo creador del kernel de Linux. Surgió como respuesta a la necesidad de un sistema de control de versiones eficiente para el desarrollo de Linux.

El control de versiones significa que GIT registra y gestiona todos los cambios realizados en los archivos de un proyecto a lo largo del tiempo. Permitiendo a los desarrolladores trabajar de manera colaborativa evitando perder el trabajo realizado o sobrescribir cambios importantes.

Una de las características clave de GIT es su naturaleza distribuida, es decir, cada usuario tiene una copia completa del repositorio en su propio sistema, facilitando el trabajo offline, la colaboración remota y la independencia de ubicación.

Un repositorio en GIT, es donde se almacenan todos los archivos y el historial de cambios de un proyecto. Puede ser local (en un ordenador propio) o remoto (en un servidor como GITHUB, GITLAB o BITBUCKET) estos permiten la sincronización de código.

Un **commit** en GIT, es la unidad básica de trabajo y representa un conjunto de cambios realizados en los archivos de proyecto. Cada commit tiene un identificador único, un autor y un mensaje descriptivo que explica los cambios realizados

GIT permite crear ramas para trabajar en funcionalidades o correcciones de forma independiente sin afectar la rama principal (master o main), Las ramas facilitan el desarrollo paralelo, la implementación de nuevas características y corrección de errores.

Cuando se completa el trabajo en una rama, los cambios pueden fusionarse de nuevo a la rama principal mediante un proceso llamado **merge.** Las pull request son solicitudes de extracción, donde los desarrolladores proponen cambios para su revisión antes de implementarse en la rama principal.

También, GIT admite una variedad de control de flujos de trabajo, como GIT FLOW, GITHUB FLOW y GITLAB FLOW, que ofrece estructuras y pautas para organizar el desarrollo, las pruebas y la implementación de código en equipos y proyectos.

**Basic commands in GIT.**

**1.-** **git init:** inicializa un nuevo repositorio GIT en el directorio actual. Este comando se utiliza una sola vez al comienzo de un proyecto para convertir un directorio normal en un repositorio GIT.

**2.- git clone [URL]:** clona un repositorio GIT existente desde una URL remota (como GITHUB, GITLAB, BITBUCKET) en un ordenador local. Este comando crea una copia local del repositorio completo.

**3.- git status:** muestra el estado actual del repositorio, incluyendo los archivos modificados, los archivos del área de preparación y cualquier archivo sin seguimiento.

**4.- git add:** agregar archivos al área de preparación (staging área) para ser incluidos en el próximo commit. Se pueden agregar archivos específicos (git add nombre \_archivo) o todos los archivos modificados (git add).

**5.- git commit –m “mensaje”:** crea un nuevo commit con los archivos previamente agregados al área de preparación. El mensaje muestra una breve descripción de los cambios realizados en el commit.

**6.-git log:** muestra el historial de commits en el repositorio, incluyendo información como el autor, fecha y el mensaje de cada commit.

**7.- git branch:** lista todas las ramas presentes en el repositorio, la rama actual es señalada por un asterisco (\*).

**8.- git checkout [nombre\_rama o hash\_commit]:** Cambia entre ramas o revisa un commit específico. Se puede utilizar el nombre de una rama o el hash del commit para moverte a un estado en particular en la historia del proyecto.

**9.- git push [nombre\_remoto][nombre\_rama\_local]:** sube los commits locales a un repositorio remoto. Especifica el nombre del repositorio remoto (como “origin” por defecto en la mayoría de los casos) y la rama local que deseas enviar.

**10.- git merge [nombre\_rama]:** fusiona los cambios de una rama específica en la rama actual. Se utiliza después de haber revisado y confirmado que los cambios son correctos.

**11.-** **git pull [nombre\_remoto][nombre\_rama]:** descarga los cambios desde un repositorio remoto y los fusiona automáticamente en la rama local especificada.Es una combinación de los comandos **git fetch y git merge**

**12.- git remote:** muestra los repositorios remotos configurados en el repositorio local. Se pueden observar las URL de los repositorios remotos asociados con nombre como “origin” u otros nombres personalizados.

**13.- git fetch [nombre\_remoto]:** descarga los cambios y referencias (ramas, Tags, etc) desde un repositorio remoto, pero no fusiona los cambios en la rama actual. Es útil para obtener actualizaciones sin aplicarlas de inmediato.

**14.- git reset [opciones][nombre\_archivo(s)]:** permite deshacer cambios en el área de preparación o en el commit actual. Se pueden usar diferentes opciones para restablecer archivos a versiones anteriores.

**15.- git diff:** muestra las diferencias entre el estado actual del directorio de trabajo y el área de preparación. Es útil para revisar los cambios antes de realizar commit.

**16: git ckeckout –b [nombre\_nueva\_rama]:** crea una nueva rama y se mueve a ella en un solo paso. Es útil para comenzar a trabajar en una nueva funcionalidad o corrección de errores.

**Working with repositories on GIT (branches, merge y conflicts**

Para inicializar un nuevo repositorio local en git, debemos abrir una terminal y posicionarnos en el directorio deseado. Luego se utiliza el comando **git init** para inicializar un nuevo repositorio en ese directorio.

Para añadir archivos a nuestro repositorio utilizamos el comando **git add** en el área de preparación o **git add nombre\_archivo** para seleccionar un archivo en específico.

Posteriormente debemos realizar el primer **commit** para guardar los archivos o cambios agregados en la historia del repositorio con el comando **git commit –m “mensaje”.**

**Branches**

Recordemos que las ramas en GIT son líneas independientes que permiten trabajar en características nuevas o correcciones de errores sin afectar la rama principal (**main o master).** Se crean con el comando **git chechout –b nueba\_rama.** Una vez en ella todos los cambios que realicemos se guardarán en esta nueva rama sin afectar la rama principal. Esto es útil para organizar el trabajo y permitir el desarrollo paralelo.

**Merge**

Después de haber completado el trabajo en una rama se procede a integrar esos cambios en la rama principal o en otra rama, se logra fusionando la rama especificada utilizando el comando **git merge.**

**Merrge conflicts**

Los conflictos de fusión ocurren cuando GIT no puede fusionar automáticamente los cambios de dos ramas debido a modificaciones conflictivas en el mismo archivo o líneas de código. GIT marca estos conflictos en los archivos y solicitará resolverlos manualmente, se mostrara un mensaje en la terminal con los archivos en conflicto. Se deberán abrir esos archivos en un editor de texto y resolver los conflictos manualmente, manteniendo las partes relevantes y eliminando las partes conflictivas.

Se pueden resolver los conflictos eliminando las secciones de delimitadores, se puede mantener una versión, eliminar la otra o cambiar ambas versiones.

Una vez resolviendo los conflictos se ejecuta el comando **git add nombre\_archivo** y luego hacer el commit para completar la fusión.

**Advanced commands**

**1.-Pull request (solicitud de extracción):** en GitHub podemos solicitar la fusión de los cambios en una rama concreta a través de un **pull request.** Esto permite a otros colaboradores en el proyecto revisar los cambios antes de fusionarlos en la rama principal.

**2.- Fork:** este comando realiza la copia independiente de un repositorio de la cuenta en la plataforma de alojamiento (GiuHub). Es muy útil cuando se quiere contribuir en un proyecto ajeno o experimentar con modificaciones sin afectar el repositorio original.

**3.- Rebase:** es una forma de combinar cambios de una rama a otra de manera más lineal que una fusión. Se puede reorganizar y reescribir el historial de commits. Se debe tener en cuenta que el rebase reescribe el historial, por lo que debe usarse con precaución en ramas compartidas.

**4.- Stash:** no permite guardar temporalmente los cambios en tu directorio de trabajo sin hacer commit. Es útil cuando se necesita cambiar de rama pero no quieres hacer commit de cambios incompletos o no relacionados.

**5.- Clean:** permite eliminar archivos no rastreados y no versionados en el directorio de trabajo de forma permanente.

**6.- Cherry pick**: permite aplicar un commit específico de una rama a otra. Esto es útil cuando se requiere traer cambios específicos de una rama sin fusionar toda la rama.

**7.- Reflog:** muestra un registro detallado de las operaciones de referencia realizadas en tu repositorio. Puede ser útil para recuperar cambios perdidos o deshacer operaciones equivocadas.

**8.- Bisect:** es una herramienta avanzada para encontrar un commit específico que introdujo un problema en el código. Permite realizar una búsqueda binaria a través del historial de commits para encontrar el culpable.