Git & Github

Guía de estudio

Academia JAVA

XIDERAL

Icono

Descripción generada automáticamenteIcono

Descripción generada automáticamente

# Introduccion a Git y Github

## GIT

Git es un sistema de control de versiones que se utiliza principalmente para el desarrollo de software. Es de distribución gratuita y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes con velocidad y eficiencia. Esto permite a los desarrolladores realizar un seguimiento de los cambios en el código fuente durante el tiempo, coordinar el trabajo entre múltiples personas, y revertir a versiones anteriores del código si es necesario.

Existen otros servicios de control de versiones sin embargo GIT se destaca entre ellos por:

* Icono

  Descripción generada automáticamentePopularidad y adopción generalizada
* Velocidad y eficiencia
* Distribución y descentralización
* Potencia y flexibilidad
* Soporte por la misma comunidad

## GITHUB

GitHub es una plataforma de alojamiento de código que utiliza el sistema de control de versiones Git.

Los usuarios pueden alojar repositorios de código públicos o privados en GitHub y colaborar con otros desarrolladores a través de funciones como solicitudes de extracción, problemas, wikis y más.

Hay otras plataformas de alojamiento de código como GitLab Bitbucket o SourceForge sin embargo algunas de las ventajas que presenta Github sobre estos tres son:

* Icono

  Descripción generada automáticamentela integración de herramientas de terceros
* su facilidad de uso
* soporte por Microsoft
* comunidad
* el ecosistema y herramientas del desarrollo que posee.

# Instalación

Para la instalación nos dirigimos directamente al siguiente enlace que es la pagina oficial de descaras de GIT para Windows: https://git-scm.com/download/win

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Aquí seleccionaremos la opción que necesitemos esta la instalación, la versión portable y la instalación desde PowerShell usando la herramienta de winget

En este ejemplo usare la descarga para la instalación de git en el equipo, al abrir el archivo solo configuraremos todo como viene por defecto.

* Aceptamos la licencia
* Seleccionamos la carpeta de destino
* Seleccionamos los componentes que fueramos a instalar
* Seleccionamo el nombre y ruta de la ubicación del folder principal que va a usar GIT
* Elegimos el editor por defecto que va a usar GIT
* Aquí dejaremos que GIT decid el nombre de los repositorios iniciales
* Dejamo sesta parte tal cual esta para que GIT ajuste el PATH
* Seleccionamos la opcion de Use bundled OpenSSh
* Usamos igual el OpenSSL library
* Dejamos esta parte igual y damos clicl en next
* Seleccionamos la opcion de MinTYY para que git use su propia consola
* Dejamos esta parte como esta
* Usamos la credencial de Git Manager
* Damos clic en siguiente
* Dejamos esta parte como esta y siguiente
* Aquí dejamos como esta y siguiente
* Finalmente damos clic en la opcion de Launch
* Y debemos de ver la consola de Git
* Para asegurarnos que todo esta bien escribimos Git version, nos debe salir algo asi.

Texto

Descripción generada automáticamente

# Comandos Básicos de Git

Antes de ver los comandos de GIT es importante conocer como es el flujo de trabajo en GIT

Escala de tiempo

Descripción generada automáticamente

Tratando de explicar la imagen: Tenemos nuestro directorio local (una carpeta en nuestro pc) con muchos archivos, Git nos irá registrando los cambios de archivos o códigos cuando nosotros le indiquemos, así podremos viajar en el tiempo retrocediendo cambios o restaurando versiones de código, ya sea en Local o de forma Remota (servidor externo).

Sabiendo esto entonces podemos presentar algunos de los comandos básicos de GIT

1. **git init** Inicia un nuevo repositorio
2. **git status** para ver que archivos no han sido registrados
3. **git add [nombre(s)]** añade archivos al área de preparación (staging) para ser incluidos en el próximo commit.
4. **git commit -m "mensaje"** crea un nuevo commit con los cambios añadidos al área de preparación, acompañado de un mensaje que describe los cambios realizados.
5. **git push** se utiliza para enviar los commits locales a un repositorio remoto. Por ejemplo: git push origin main, donde origin es el nombre del repositorio remoto y main es el nombre de la rama.
6. **git fetch** Este comando se utiliza para descargar commits, ramas y etiquetas desde un repositorio remoto a tu repositorio local. No fusiona los cambios en tu rama actual, solo los trae para que puedas revisarlos y fusionarlos manualmente si lo deseas.
7. **git checkout** Se utiliza para cambiar entre ramas o para restaurar archivos del árbol de trabajo a un estado específico. Por ejemplo: git checkout main cambiará a la rama main, o git checkout -- archivo.txt restaurará archivo.txt al estado en el último commit.
8. **git checkout** Se utiliza para cambiar entre ramas o para restaurar archivos del árbol de trabajo a un estado específico. Por ejemplo: git checkout main cambiará a la rama main, o git checkout -- archivo.txt restaurará archivo.txt al estado en el último commit.
9. **git log** Muestra la lista de commit del más reciente al más antiguo.
10. **git revert** Esto creará un nuevo commit que deshace los cambios introducidos por el commit que deseas revertir.

Son los comandos más básicos, aquí hay algunos otros que suelen ser muy útiles

* **git log --oneline** muestra en una línea los commit realizados
* **git log --oneline --decorate --all –graph** muestra en una línea los commit realizados pero más elegante
* **git status -s** solo muestra los archivos modificados
* **git clone [URL]** clona un repositorio Git existente desde una URL remota a tu máquina local.

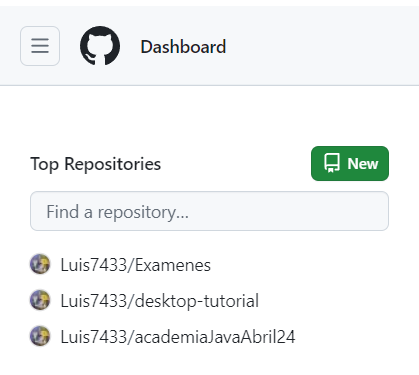
Con estos últimos podemos notar que algunos usan el guion doble – algunos usan solo un guion.

La diferencia es que -- hace referencia a una palabra y – hace referencia al comando o a varios comandos por ejeplo:

Texto

Descripción generada automáticamente con confianza media

# Trabajando con repositorios en github (Branches Merge, Conflicts)

Un repositorio es el elemento más básico de GitHub. Es un lugar donde puedes **almacenar el código, los archivos y el historial** de revisiones de cada archivo. Los repositorios pueden contar con **múltiples colaboradores** y pueden ser **públicos como privados**.

Puedes ser propietario de repositorios individualmente o puedes compartir la propiedad de los repositorios con otras personas en una organización.

En cualquier caso, los permisos administran el acceso a los repositorios.

Cuando creas un repositorio, puedes elegir si lo quieres hacer público o privado.

* Los repositorios públicos son accesibles para todo el mundo en Internet.
* Solo tú, las personas con las que compartes el acceso explícitamente y, para los repositorios de organizaciones, algunos miembros de la organización, pueden acceder a los repositorios privados.

Creación de un repositorio con GITHUB:

En la esquina superior izquierda de la aplicación selecciona y luego haz clic en Nuevo repositorio.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

A continuación, aparecerá un formulario que debe de llenar, los campos incluyen

* Interfaz de usuario gráfica, Texto, Aplicación

  Descripción generada automáticamenteName: Nombre del repositorio, no introduzca espacios o caracteres especiales
* Description: Le da la opción de añadir una descripción del repositorio
* Local path: es la carpeta dentro de su equipo a la cual desea usar como repositorio, la aplicación le da la oportunidad de navegar dentro del explorador de archivos de su equipo para seleccionar la carpeta
* Seleccione Initialize this repository with a README (Inicializar este repositorio con un archivo Léame).
* La aplicación le da la exclusividad de seleccionar que tipo de archivos desea ignorar para que no se copien
* Finalmente decida si ncesita una licencia
* Por ultimo da clic en Create repository

## Branches

Los branches, o ramas en español, son versiones paralelas del código en un repositorio de control de versiones.

Cada vez que creas una rama en un repositorio, estás creando una copia exacta del código en ese momento, en una especie de línea de desarrollo separada.

Los branches son útiles por varias razones:

* **Desarrollo paralelo:** Permiten a múltiples personas trabajar en diferentes características o arreglos de errores al mismo tiempo sin interferir entre sí.
* **Pruebas aisladas:** Puedes crear una rama para probar una nueva característica o realizar cambios importantes sin afectar la rama principal del proyecto hasta que estés seguro de que funciona correctamente.
* **Experimentación:** Los branches ofrecen la libertad de experimentar con nuevas ideas sin afectar la estabilidad del código principal.
* **Versionado y control de cambios:** Cada rama puede tener su propio historial de cambios, lo que permite un seguimiento detallado de las modificaciones específicas realizadas en cada función o arreglo de errores.

Se pueden crear branches a través de GITHUB seleccionando la opción Branch y dando clic en New Branch

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

A continuación aparece el formulario que se debe de llenar para crear el Branch

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Recordemos que un Branch es una rama que incluso puede estar dentro de otras ramas

Para la creación y uso de algunos comandos de las Branch podemos hacer lo siguiente:

Crear una rama

* **git branch <nombre\_de\_la\_rama>** Crea una nueva rama con el nombre especificado, pero no cambia a esa rama

Cambiar una rama existente

* **git checkout <nombre\_de\_la\_rama>** Cambia al branch especificado.
* **git switch <nombre\_de\_la\_rama>** Cambia al branch especificado. (a partir de Git 2.23)

Crear una rama y cambiar a ella

* **git checkout -b <nombre\_de\_la\_rama>** Crea una nueva rama con el nombre especificado y cambia a ella.
* **git switch -c <nombre\_de\_la\_rama>** Crea una nueva rama con el nombre especificado y cambia a ella. (a partir de Git 2.23)

Listar todas las ramas

* **git branch** Muestra todas las ramas en el repositorio local. La rama actual se indica con un asterisco (\*).

Eliminar una rama

* **git branch -d <nombre\_de\_la\_rama>** Elimina la rama especificada si ya ha sido fusionada en otra rama.
* **git branch -D <nombre\_de\_la\_rama>** Elimina la rama especificada, incluso si no ha sido fusionada en otra rama.

Renombrar una rama

* **git branch -m <nombre\_nuevo>** se puede renombrar una rama moviendo la anterior y creando una nueva
* **git branch -m <nombre\_antiguo> <nombre\_nuevo>** renombrado tradicional

Fusionar ramas

* **git merge <nombre\_de\_la\_rama>** Fusiona la rama especificada en la rama actual.

Ver historial de ramas fusionadas

* **git branch --merged** Muestra una lista de ramas que han sido fusionadas en la rama actual.

Ver historial de ramas no fusionadas

* **git branch --no-merged** Muestra una lista de ramas que aún no han sido fusionadas en la rama actual.

## Conflicts

Los conflictos (conflicts) en Git ocurren cuando hay divergencias entre las diferentes versiones de un archivo, especialmente cuando intentas fusionar cambios de una rama a otra. Estos conflictos pueden surgir cuando dos ramas contienen cambios en las mismas líneas de código o cuando una rama elimina un archivo que otra rama está modificando, entre otras situaciones.

Cuando Git detecta un conflicto durante una fusión, marca el archivo en conflicto y te solicita que lo resuelvas manualmente. Git **no puede determinar automáticamente** cuál de las versiones en conflicto debe prevalecer, ya que no puede adivinar la intención del desarrollador.

En cuanto a los tipos de conflictos, se pueden clasificar en función de las diferencias encontradas en los archivos en conflicto. Algunos tipos comunes de conflictos incluyen:

* **Conflictos de contenido:** Ocurren cuando dos ramas modifican las mismas líneas de un archivo y Git no puede determinar automáticamente cuál es la versión correcta. Esto puede suceder cuando dos personas editan las mismas partes del código o cuando se reorganizan líneas de código de manera diferente en diferentes ramas.
* **Conflictos de eliminación/modificación:** Ocurren cuando una rama elimina un archivo que otra rama ha modificado. Git no puede determinar si debe conservar los cambios o eliminar el archivo.
* **Conflictos de renombramiento/movimiento:** Pueden ocurrir cuando un archivo es renombrado o movido en una rama mientras que en otra rama se realizan cambios en el mismo archivo, lo que puede generar conflictos de fusión.
* **Conflictos de tipo de archivo:** A veces, las diferencias entre las ramas pueden ser tan significativas que Git no puede determinar cómo fusionar los archivos. Esto puede ocurrir cuando se realizan cambios significativos en la estructura de archivos o en el tipo de datos.

# Comandos avanzados(Pull Request, Fork, Rebase, StaSh, Clean, Cherry-pick, etc)

## git rebase

Reescribe la historia de la rama actual moviendo, combinando o eliminando commits para que se apliquen sobre la rama base

**git rebase <rama\_base>**

## git cherry-pick

Aplica un commit específico desde una rama a otra.

**git cherry-pick <hash\_del\_commit>**

## git reflog

Muestra un registro detallado de todas las operaciones realizadas en el repositorio, incluidos los cambios de rama y los rebase.

**git reflog**

## git bisect

Ayuda a encontrar el commit que introdujo un error mediante una búsqueda binaria entre los commits.

**git bisect start**

**git bisect bad**

**git bisect good <hash\_del\_commit\_bueno>**

## git filter-branch

Reescribe la historia del repositorio aplicando filtros a los commits.

**git filter-branch --commit-filter <filtro> <rama>**

## git stash

Guarda temporalmente los cambios en un stash para que puedas trabajar en otra cosa.

**git stash**

## git submodule

Administra submódulos dentro de tu repositorio para incluir dependencias externas.

**git submodule add <URL\_del\_submodulo>**

## git worktree

Permite trabajar en múltiples árboles de trabajo (directorios de trabajo) desde una sola copia de tu repositorio.

**git worktree add <ruta\_del\_nuevo\_arbol\_de\_trabajo> <rama>**

## git revert

Deshace un commit existente creando un nuevo commit que revierte los cambios introducidos por el commit original.

**git revert <hash\_del\_commit>**

## git blame

Muestra quién modificó cada línea de un archivo y en qué commit se realizaron los cambios.

**git blame <nombre\_del\_archivo>**

## Git clean

El comando git clean se utiliza para eliminar archivos no rastreados en el directorio de trabajo. Estos son archivos que no están bajo control de versión (no han sido añadidos al índice) y no están presentes en el último commit confirmado en la rama actual.

* **git clean** es el uso básico del comando clean
* **git clean -d** eliminará irreversiblemente todos los archivos no rastreados en el directorio de trabajo. Sin embargo, git clean no eliminará directorios no rastreados de forma predeterminada. Para eliminar directorios no rastreados, usa la opción -d
* **git clean -n** para previsualizar qué archivos se eliminarían sin realmente eliminarlos, puedes usar la opción -n o --dry-run
* **git clean -df** si deseas eliminar también los directorios no rastreados de forma segura (es decir, hacer una limpieza completa), puedes combinar las opciones -d y -f (force):

## Folk

Un fork es una copia de un repositorio en un servidor remoto (como GitHub) que se guarda en tu propia cuenta de usuario. Básicamente, es una bifurcación del proyecto original en la que puedes trabajar de forma independiente.

Para añadir un repositorio a tu cuenta personal desde la aplicación web puedes dirigirte a la sección fork: Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Una vez que has creado un fork, puedes clonarlo en tu máquina local utilizando el comando **git clone <URL\_del\_fork>**. Luego, puedes hacer cambios en tu fork como lo harías con cualquier otro repositorio Git.

Para tener tu fork actualizado con los cambios que se realizan en el repositorio original puedes hacerlo configurando un remoto adicional que apunte al repositorio original (**git remote add upstream <URL\_del\_repositorio\_original>)** y luego haciendo **git fetch upstream** seguido de **git merge upstream/main** (o el nombre de la rama principal del repositorio original) para traer los cambios del repositorio original a tu fork.

Por ultimo si quieres contribuir a un fork con cambios al proyecto original puedes enviar un pull request desde tu fork al repositorio original

## Pull Request

### ¿Qué es un Pull Request (PR)?

Es una solicitud para que los cambios realizados en una rama de tu repositorio (generalmente en tu fork) se fusionen en otra rama (generalmente la rama principal) de otro repositorio (generalmente el repositorio original). Es una forma de notificar a los propietarios del repositorio sobre los cambios que has realizado y solicitar que sean revisados y fusionados en la rama principal del proyecto.

### Propósito de un Pull Request

Los pull requests son esenciales en el desarrollo colaborativo, ya que permiten a los desarrolladores discutir, revisar y validar los cambios antes de que se integren en el proyecto principal. También proporcionan transparencia en el proceso de contribución y permiten un seguimiento claro de quién hizo qué cambios y cuándo.

### Creación de un Pull Request

Para crear un pull request, primero necesitas hacer cambios en una rama de tu repositorio local. Luego, puedes enviar esos cambios a tu repositorio remoto (por ejemplo, en GitHub) y desde allí crear un pull request. En GitHub, puedes hacer esto seleccionando la opción "New pull request" en la página de tu repositorio y seleccionando las ramas entre las cuales deseas fusionar los cambios.

### Revisión y Discusión

Una vez que has creado un pull request, los revisores pueden revisar tus cambios, dejar comentarios y discutir sobre ellos. Esto permite una colaboración efectiva y asegura que los cambios sean de alta calidad y estén bien entendidos antes de ser fusionados.

### Integración del Pull Request

Después de que tus cambios han sido revisados y aprobados, un colaborador con permisos de escritura en el repositorio original puede fusionar tu pull request. Esto aplica tus cambios en la rama principal del proyecto y los hace parte oficial del repositorio.

### Actualizaciones y Seguimiento

A menudo, los revisores pueden solicitar cambios adicionales o sugerir mejoras en tu pull request. En este caso, puedes realizar las modificaciones necesarias en tu rama y agregar nuevos commits. Estos cambios se reflejarán automáticamente en tu pull request existente, lo que permite una iteración continua hasta que los cambios sean aceptados.