

# ➤ Fundamentos de GIT y GITHUB

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

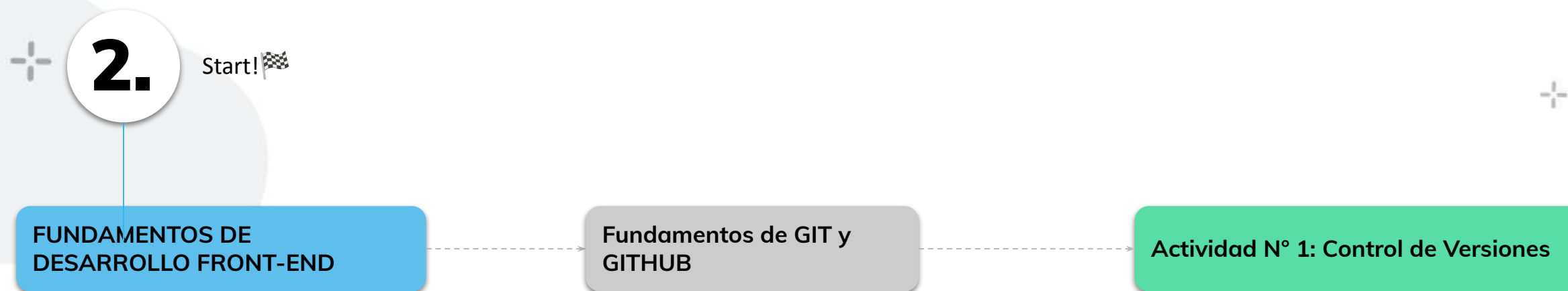
En la clase anterior trabajamos :

- ✓ Comprender el uso de plugins Bootstrap-jQuery en el desarrollo web.
- ✓ Identificar cuándo y por qué utilizar un plugin en un proyecto web.
- ✓ Aprender a incorporar y configurar plugins bootstrap-jQuery



# LEARNING PATHWAY

¿Sobre qué temas trabajaremos?



Este curso se enfoca en los fundamentos esenciales de Git, una herramienta vital para el control de versiones en el desarrollo de software. Aprenderás desde la necesidad de un repositorio hasta comandos básicos y avanzados, incluyendo commits, cambios de nombres, ramas, conflictos y etiquetas.



# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



**Comprender la importancia del control de versiones y la necesidad de un repositorio de código fuente.**



**Instalar y configurar Git en un entorno de desarrollo.**



**Dominar los comandos básicos de Git para crear, clonar y gestionar repositorios.**





# Rompehielo

## Consigna:



1. En una habitación hay cuatro gatos, cada gato ve a tres gatos. ¿Cuántos gatos hay en la habitación?
2. A las 12 del día tengo que tomarme mi primera de cuatro, debo tomarme cada hora una, ¿A qué hora me debo tomar la última?
3. Siempre mirando a la izquierda, el más chiquito no soy, me arqueó un poquito y redondito soy, mi cuerpo como un patito, pero animal no soy. ¿Quién soy?



# ➤ Fundamentos de GIT y GITHUB

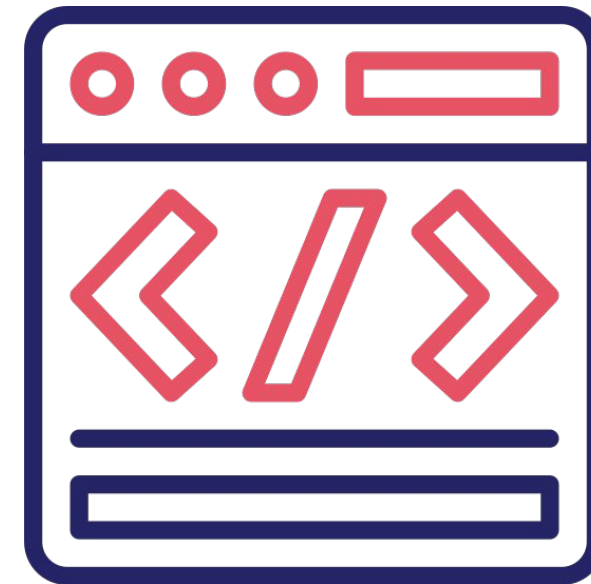


# Fundamentos de GIT

Es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar y colaborar en proyectos de software de manera eficiente, manteniendo un historial completo de los cambios realizados en los archivos del proyecto.

Se utiliza a través de la línea de comandos, pero también existen interfaces gráficas y herramientas de terceros que facilitan su uso.

Algunas de las plataformas populares que aprovechan Git para la gestión de proyectos son GitHub, GitLab y Bitbucket.



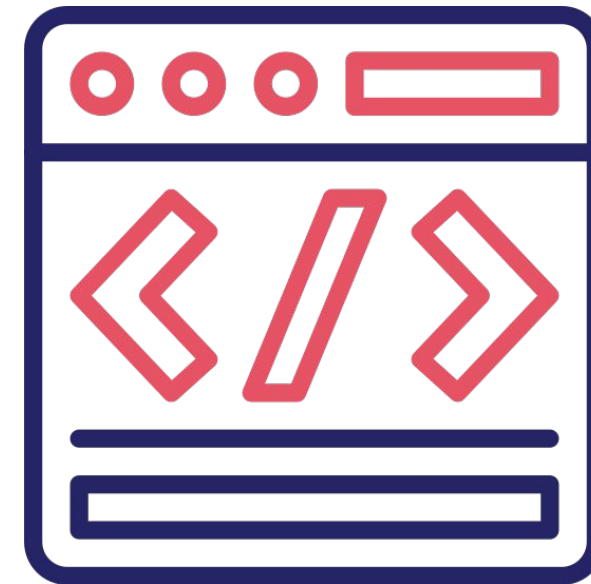




# Fundamentos de GITHUB

GitHub es una plataforma web basada en Git que facilita la colaboración y el almacenamiento de proyectos de desarrollo de software.

Es uno de los servicios más populares para alojar repositorios Git y proporciona herramientas adicionales para la gestión de proyectos y la colaboración en equipo.



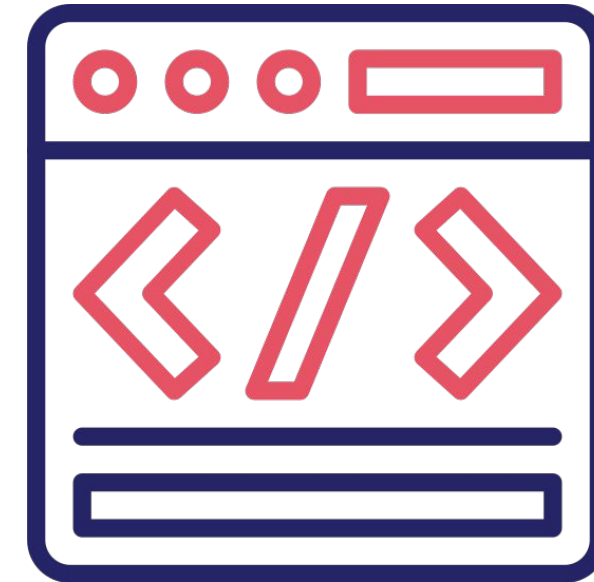


# Fundamentos de GIT y GITHUB

## Algunas de las características clave de GitHub:

### Control de versiones

GitHub aprovecha la potencia de Git para proporcionar un historial completo de cambios en los archivos de un proyecto, así como la capacidad de crear ramas, fusionar cambios y revertir revisiones.



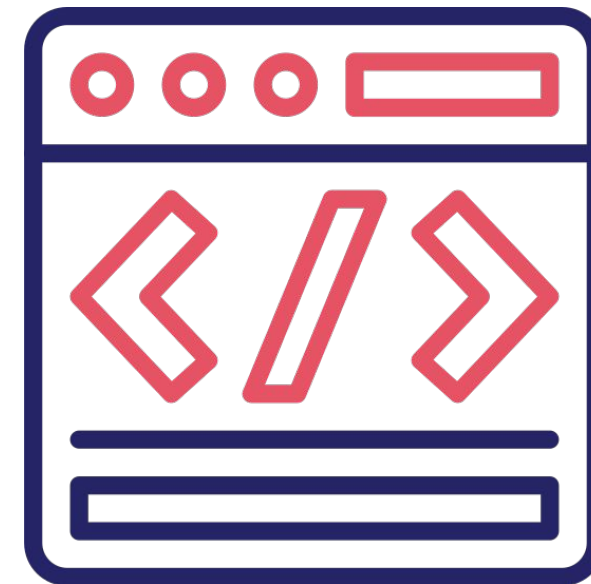


# Fundamentos de GIT y GITHUB

## Algunas de las características clave de GitHub:

### Colaboración

Los desarrolladores pueden colaborar en proyectos compartiendo sus repositorios con otros y permitiendo la contribución de código a través de solicitudes de extracción (pull requests). Esto facilita el trabajo en equipo y la revisión de cambios antes de fusionarlos en el proyecto principal.



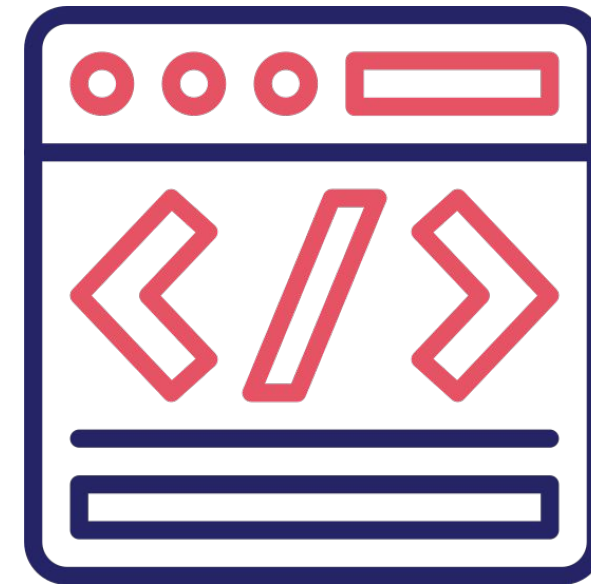


# Fundamentos de GIT y GITHUB

## Algunas de las características clave de GitHub:

### Problemas y seguimiento de tareas

GitHub permite la creación de problemas (issues) para realizar un seguimiento de errores, solicitudes de nuevas características y otras tareas relacionadas con el proyecto. Los problemas pueden asignarse a miembros del equipo, etiquetarse y comentarse para facilitar la comunicación.



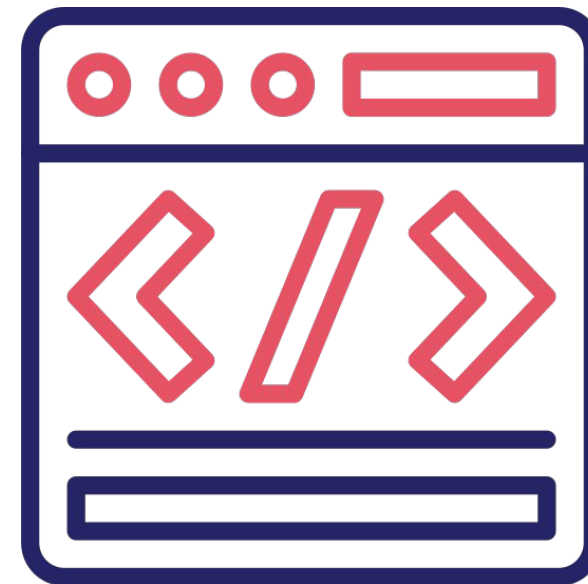


# Fundamentos de GIT y GITHUB

## Algunas de las características clave de GitHub:

### Integración continua

GitHub proporciona integración con herramientas de integración continua (CI) como Travis CI, CircleCI o Jenkins. Esto permite automatizar la ejecución de pruebas, compilación de código y otras tareas durante el desarrollo del proyecto.



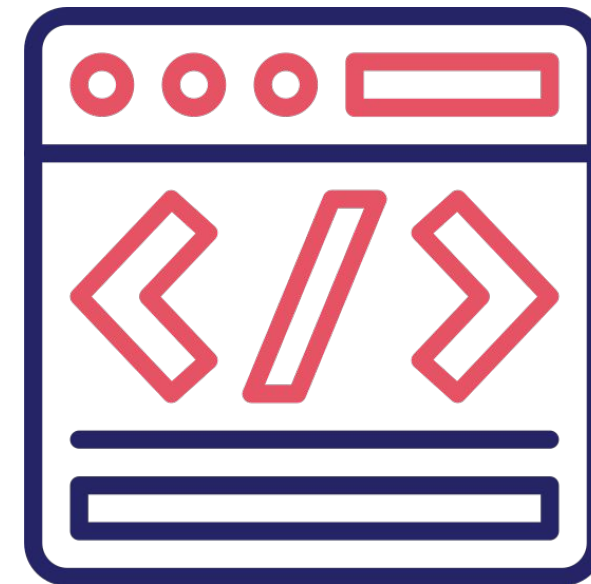


# Fundamentos de GIT y GITHUB

## Algunas de las características clave de GitHub:

### Despliegue

Con las funcionalidades de GitHub Actions, se pueden automatizar también las tareas de despliegue del software en diferentes entornos, como servidores de producción o plataformas de servicios en la nube.



# › Instalación, configuración y comandos básicos



# Instalación



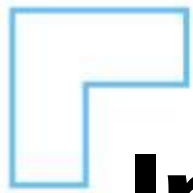
## GITHUB

Para poder vincular un repositorio de [gitHub](https://github.com), comencemos creando uno:

1. Primero Inicia sesión en tu cuenta de GitHub o crea una nueva si no tienes una. Haz clic en el botón "New" o "Nuevo" para crear un nuevo repositorio.
2. Ahora, asigna un nombre al repositorio, opcionalmente añade una descripción y elige si quieres que sea público o privado.
3. Haz clic en "Create repository" o "Crear repositorio" para crearlo.







# Instalación



## GIT

**Paso 1:** Descargar el instalador de Git Bash:

Ve al sitio web oficial de Git (<https://git-scm.com/>) y busca el botón de descarga. Git es compatible con diferentes sistemas operativos, así que asegúrate de seleccionar la versión correcta para tu sistema.



**Paso 2:** Ejecutar el instalador:

Una vez que hayas descargado el instalador, ábrelo haciendo doble clic en el archivo descargado.





# Instalación



## GIT

**Paso 3:** Configurar el instalador:

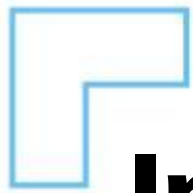
Aparecerá un asistente de configuración para la instalación de Git. Asegúrate de leer y aceptar los términos de licencia.



**Paso 4:** Seleccionar componentes:

En esta etapa, puedes elegir los componentes que deseas instalar. Asegúrate de seleccionar "Git Bash Here" o "Git Bash Here & GUI" para asegurarte de que Git Bash esté disponible en el menú contextual del Explorador de archivos (Windows).





# Instalación



## GIT

**Paso 5:** Elegir la ubicación de instalación:

Elige la ubicación donde deseas instalar Git. La configuración predeterminada suele ser adecuada para la mayoría de los usuarios.

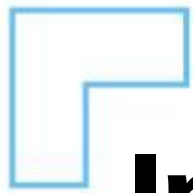


**Paso 6:** Seleccionar el editor de texto (opcional):

El instalador puede pedirte que elijas un editor de texto para usar con Git Bash.

Podes seleccionar el que prefieras o dejar el predeterminado.





# Instalación



## GIT

### **Paso 7:** Añadir Git al PATH:

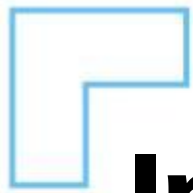
Esta opción permitirá que puedas acceder a Git desde la línea de comandos (cmd) o desde otras aplicaciones sin tener que especificar la ruta completa. Es recomendable seleccionar "Git from the command line and also from 3rd-party software".



### **Paso 8:** Configurar la terminal predeterminada:

Podes elegir la terminal que prefieras utilizar con Git Bash. Si tienes Visual Studio Code instalado, puedes seleccionar "Use Windows' default console window" o "Use MinTTY" para utilizar una terminal más avanzada.





# Instalación



## **GIT**

### **Paso 9:** Configuración adicional:

En esta etapa, se te pedirá que configures el comportamiento de finalización de línea. La configuración predeterminada suele ser adecuada para la mayoría de los usuarios.

### **Paso 10:** Instalación:

Una vez que hayas revisado todas las configuraciones, haz clic en "Install" para iniciar la instalación.

### **Paso 11:** Finalizar la instalación:

Una vez que se haya completado la instalación, haz clic en "Finish" para cerrar el asistente de configuración.

# Configurar Git en nuestra consola de GIT BASH

## En Windows:

1. Ve al menú Inicio o a la pantalla de inicio de Windows.
2. Busca "Git Bash" en el cuadro de búsqueda.
3. Haz clic en el icono "Git Bash" que aparece en los resultados de búsqueda.

## En macOS:

1. Abre Finder.
2. Navega a la carpeta "Aplicaciones" (Applications).
3. Busca la carpeta "Utilities" y ábrela.
4. Busca el programa "Terminal" y ábrelo.
5. Una vez que la Terminal esté abierta, Git Bash debería estar disponible si ya has instalado Git.





# Pasos iniciales



## 1. Verificar la instalación de Git:

Para asegurarte de que Git se haya instalado correctamente, puedes ejecutar el siguiente comando en Git Bash:



```
git --version
```

Si la instalación es exitosa, verás la versión de Git que tienes instalada.





# Pasos iniciales



## 2. Configurar tu nombre de usuario y dirección de correo electrónico:

Es importante configurar tu nombre de usuario y dirección de correo electrónico en Git para que tus commits reflejen correctamente tu identidad. Utiliza los siguientes comandos, reemplazando "Tu Nombre" y "tu@email.com" con tus datos reales:



```
git config --global user.name "Tu Nombre"  
git config --global user.email "tu@email.com"
```







# Pasos iniciales



## 3. Clonar un repositorio desde GitHub:

Podes clonar un repositorio existente desde GitHub en tu computadora utilizando Git Bash. Ve a la página del repositorio en GitHub y copia la URL del repositorio. Luego, en Git Bash, navega a la ubicación donde deseas clonar el repositorio y ejecuta el siguiente comando, reemplazando "URL\_del\_repositorio" con la URL que copiaste:



```
git clone URL_del_repositorio
```





# Pasos iniciales



## 4. Crear un nuevo repositorio desde cero:

Si deseas crear un nuevo repositorio en tu computadora desde cero, simplemente navega a la ubicación donde deseas que se encuentre el repositorio y ejecuta los siguientes comandos:



```
git init
```





# Pasos iniciales



## 5. Realizar el primer commit:

Para lograrlo, añada todos los archivos modificados y nuevos al área de preparación con el siguiente comando:



```
git add .
```

Realiza el commit con el siguiente comando:

```
git commit -m "Primer commit"
```





# Pasos iniciales



## 6. Subir los cambios al repositorio remoto:

Es hora de subir tus cambios al repositorio remoto en GitHub con el siguiente comando:

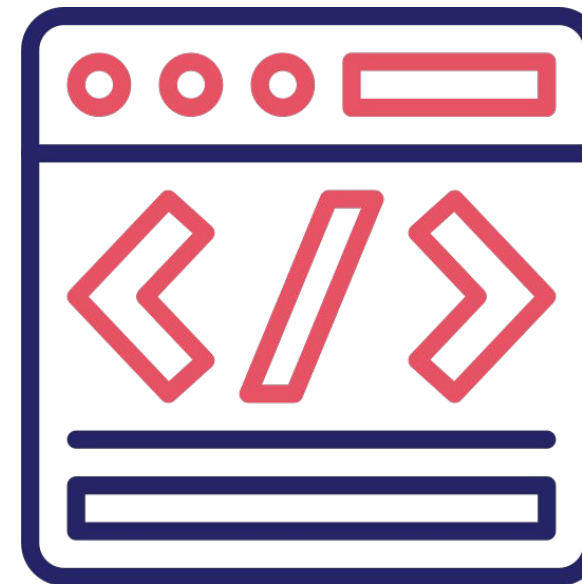
```
git push -u origin master
```

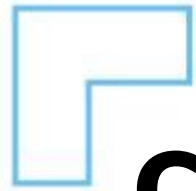




# Commits y restauración de archivos

Los commits en Git son utilizados para guardar y registrar los cambios realizados en el repositorio. Te mostraré cómo realizar commits y restaurar archivos en Git.





# Commits y restauración de archivos



## Veamos los pasos para realizar un commits

1. Verificar el estado de los archivos: Antes de realizar un commit, puedes utilizar el siguiente comando para ver los cambios realizados en los archivos del repositorio:



```
git status
```





# Commits y restauración de archivos



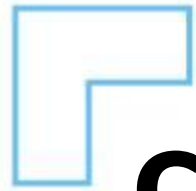
## Veamos los pasos para realizar un commits

2. Agregar los archivos modificados al área de preparación: Utiliza el siguiente comando para agregar los archivos modificados al área de preparación (staging area):



```
git add nombre_de_tu_archivo
```





# Commits y restauración de archivos



## Veamos los pasos para realizar un commits

3. Realizar el commit: Una vez que los archivos están en el área de preparación, puedes realizar el commit con el siguiente comando: (Cambia “tu descripción” Por una muy breve de los cambios realizados)



```
git commit -m "Tu descripción"
```



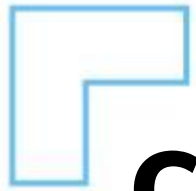


# Commits y restauración de archivos

## Deshacer cambios en el directorio de trabajo:

1. Si deseas deshacer los cambios realizados en un archivo y volver a la versión anterior almacenada en el último commit, utiliza el siguiente comando:

```
git checkout -- nombre_de_tu_archivo
```



# Commits y restauración de archivos



## Restaurar archivos desde el área de preparación:

2. Si deseas quitar los archivos del área de preparación y restaurarlos a su estado anterior, utiliza el siguiente comando:



```
git reset HEAD nombre_de_tu_archivo
```





# Commits y restauración de archivos



## Restaurar un commit anterior:

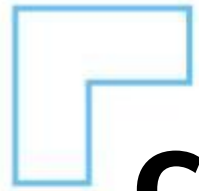
2. Si necesitas deshacer un commit y restaurar los archivos a una versión anterior, puedes utilizar el siguiente comando:



```
git revert hash_del_commit
```

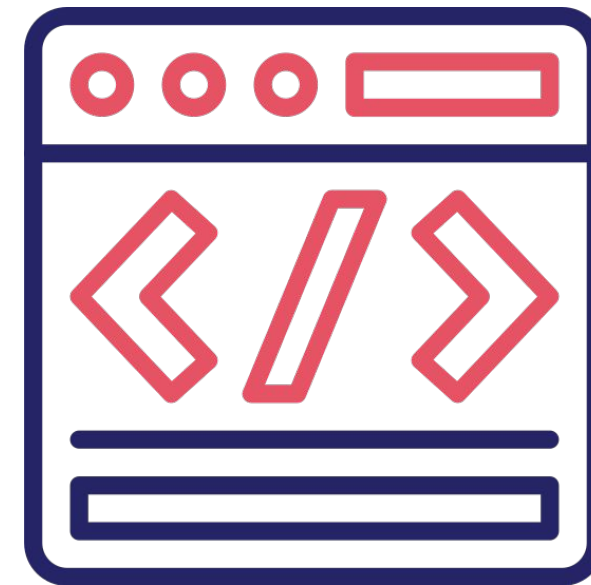
Por último, reemplaza **hash\_del\_commit** con el identificador único (hash) del commit al que deseas regresar. Esto creará un nuevo commit que deshace los cambios introducidos por el commit especificado.

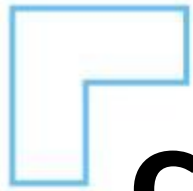




# Cambios de nombres

Uno de los cambios que podemos hacer desde git, es hacer un cambio de nombres en los archivos.





# Cambios de nombres



1. Verificar el estado actual de los archivos, con el siguiente comando:

```
git status
```



2. Renombrar el archivo utilizando el comando git mv seguido del nombre actual del archivo y el nuevo nombre que deseas asignarle. Por ejemplo, si quieres cambiar el nombre de "archivo\_viejo.txt" a "archivo\_nuevo.txt", ejecuta el siguiente comando:

```
git mv archivo_antiguo.txt archivo_nuevo.txt
```





# Cambios de nombres



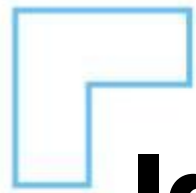
3. puedes verificar los cambios realizados utilizando nuevamente el comando git status para verificar los cambios realizados:

```
git status
```

4. Realizar un commit para agregar los cambios al área de preparación, y entre las comillas agrega una descripción del mismo:

```
git add .  
git commit -m "mje descriptivo para el commit"
```

Después de seguir estos pasos, el archivo se habrá renombrado en tu repositorio Git. Hay que tener en cuenta que el cambio de nombre se verá reflejado en todos los commits posteriores y en las versiones futuras del archivo.



# Ignorando archivos



Otra acción disponible en Git, es la de ignorar archivos, ya que algunos pueden no ser necesarios para el proyecto.

Para ignorar archivos en Git, puedes crear un archivo llamado `.gitignore` en el directorio raíz de tu repositorio y agregar los patrones de archivos que deseas ignorar. Estos patrones ayudan a especificar los archivos o tipos de archivos que no deseas que se rastreen o incluyan en el control de versiones.





# Ignorando archivos



1. Crea un archivo `.gitignore` en el directorio de tu proyecto. Podes hacerlo a través de la línea de comandos o utilizando el explorador de archivos del sistema operativo.
2. Abre el archivo `.gitignore` en un editor de texto y agrega los patrones de archivos que deseas ignorar. Cada uno de estos patrones se colocan en una línea separada. Podes utilizar comodines y reglas específicas para los patrones. }

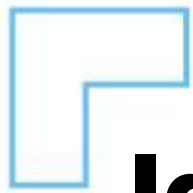


Por ejemplo, si deseas ignorar todos los archivos `.txt` y el directorio `logs`, podes agregar estas líneas al archivo `.gitignore`:

```
*.txt  
logs/
```







# Ignorando archivos



Esto evitará que se rastreen todos los archivos con extensión .txt y el directorio logs.

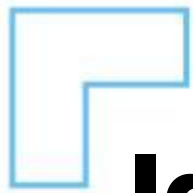
**3.** Guarda el archivo .gitignore y ciérralo.

Asegúrate de no haber realizado el seguimiento de los archivos o directorios que desees ignorar antes de agregar el archivo .gitignore. Podes utilizar el siguiente comando para verificar el estado actual de los archivos:



```
git status
```





# Ignorando archivos



4. Realiza un commit que incluya el archivo .gitignore:

```
git add .gitignore  
git commit -m "Agregado archivo .gitignore"
```

Una vez completos estos pasos, Git comenzará a ignorar los archivos y directorios especificados en el archivo .gitignore. Estos archivos no serán rastreados y no se incluirán en los commits futuros.



# › Ramas, uniones, conflictos y tags



# Ramas, uniones, conflictos y tags



Las ramas “branches”, las fusiones “merges”, los conflictos y las etiquetas “tags” son conceptos importantes en Git para administrar el flujo de trabajo y organizar los commits.






# Ramas, uniones, conflictos y tags



## Ramas (branches)

Las ramas en Git son líneas de desarrollo independientes que permiten trabajar en diferentes características o solucionar problemas sin afectar la rama principal, la cual, mayormente se nombra como master.



Cada rama tiene su historial de commits y cambios específicos, lo que facilita el desarrollo paralelo y la organización del trabajo en equipo.





# Ramas, uniones, conflictos y tags



## Para crear una rama en git:

1. Comenzaremos llamando a la nueva rama, en nuestro caso se llama “nombre-rama”:

```
git revert hash_del_commit
```

2. Ahora, nos movemos hacia la nueva rama:

```
git checkout nombre-rama
```



# Ramas, uniones, conflictos y tags

## Para crear una rama en git:

3. Haz los cambios necesarios en los archivos de tu proyecto mientras te encuentras en la rama "nueva-rama". Por ejemplo, puedes agregar un nuevo archivo o modificar el código existente.
4. Realiza un commit en la nueva rama creada:

```
git add .  
git commit -m "Agregado archivo y cambios en la nueva rama"
```



# Ramas, uniones, conflictos y tags



## Para crear una rama en git:

5. Vuelve a la rama principal (por ejemplo, "master"):

```
git checkout master
```

6. Por último, es momento de fusionar la rama "nombre-rama" en la rama principal:

```
git merge nombre-rama
```







# Ramas, uniones, conflictos y tags



## Fusiones (merges)

Las fusiones se utilizan para combinar los cambios de una rama en otra rama, generalmente para unir una rama de desarrollo a la rama principal.

Podes notar, que en el ejemplo anterior, utilizamos una fusión en el último paso, fusionando la nueva rama creada con el menú principal. Por lo que ya tienes la demostración del funcionamiento.





# Ramas, uniones, conflictos y tags



## Conflictos

Los conflictos ocurren cuando Git no puede combinar automáticamente los cambios de dos ramas debido a modificaciones conflictivas en las mismas líneas de código.

Git indicará los archivos con conflictos y te mostrará los cambios en conflicto. Debes resolver los conflictos manualmente editando los archivos afectados y eligiendo qué cambios mantener.



Una vez que se resuelven los conflictos, debes realizar un commit para finalizar la fusión.





# Ramas, uniones, conflictos y tags



## Etiquetas (tags):

Las etiquetas son punteros inmutables a un commit específico en la historia del repositorio. Los cuales se utilizan para marcar versiones o hitos importantes.

Y son muy útiles para marcar versiones estables de tu proyecto.





# Ramas, uniones, conflictos y tags



## El modo de trabajar con etiquetas:

Antes que nada, debemos verificar los commits existentes. Para eso utiliza el siguiente comando para ver el historial de commits en tu repositorio:



```
git log --oneline
```

Esto mostrará una lista de commits con sus identificadores únicos “hash” y mensajes descriptivos.





# Ramas, uniones, conflictos y tags



## El modo de trabajar con etiquetas:

Es momento de crear una etiqueta ligera (lightweight tag):

Podes crear una etiqueta ligera en un commit específico utilizando el siguiente comando:



```
git tag nombre_etiqueta commit_hash
```

Cambia “nombre\_etiqueta” por el nombre que quieras usar y agrega commit\_hash con el identificador único del commit al que deseas etiquetar.

```
git tag v1.0 abcdef123456
```

Esto creará una etiqueta llamada "v1.0" en el commit con el identificador "abcdef123456".

# Evaluación Integradora ✨



## Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase

[CLICK AQUÍ](#)

para ver la consigna completa



# **Ejercicio**

# Control de Versiones



# Control de Versiones

Breve descripción

## Contexto: 🙌

Continuamos trabajando en el proyecto para la wallet digital. El objetivo es utilizar Git para el control de versiones.

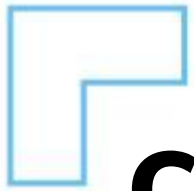
## Consigna: 📝

Usar los conocimientos vistos en la clase de hoy para capturar el estado de nuestro proyecto

**Tiempo 🕒: 20 minutos**







# Control de Versiones

Breve descripción

**Paso a paso:** 

**Uso de Git:**

- Realiza un commit inicial para capturar el estado inicial del proyecto.

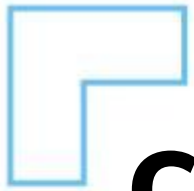
**Commits y Control de Versiones:**

- Utiliza commits de Git de manera regular y significativa mientras desarrollas la wallet digital. Cada commit debe representar un conjunto lógico de cambios.

**Ramas de Desarrollo:**

- Crea y utiliza al menos una rama de desarrollo para trabajar en nuevas características o mejoras..





# Control de Versiones

Breve descripción

**Paso a paso:** 

## **Resolución de Conflictos:**

- Si trabajas en equipo, simula un conflicto en Git y resuélvelo correctamente.

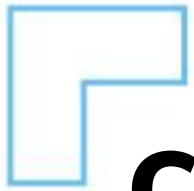
## **Etiquetas (Tags):**

- Crea una etiqueta (tag) en Git para marcar una versión específica de tu proyecto.

## **Stash y Rebase:**

- Utiliza Git stash para guardar temporalmente cambios no comprometidos cuando sea necesario.
- Realiza un rebase interactivo en Git para organizar y limpiar la historia de commits de tu proyecto.





# Control de Versiones

Breve descripción

**Paso a paso:** 

## Entrega y Comentarios:

- Al finalizar el proyecto, realiza una última confirmación y etiqueta la versión final.
- Agrega un archivo README.md que explique cómo clonar, ejecutar y contribuir al proyecto.
- Comparte el enlace al repositorio de Git con tu instructor para su revisión.



○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender la importancia del control de versiones y la necesidad de un repositorio de código fuente.**
- ✓ **Instalar y configurar Git en un entorno de desarrollo.**
- ✓ **Dominar los comandos básicos de Git para crear, clonar y gestionar repositorios.**



# #WorkingTime

Continuemos ejercitando

**¡Antes de cerrar la clase!** Te invitamos a: 📌 📌 📌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. Material 1: Lección 7: Fundamentos de GIT y GITHUB: 1-13
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento: ✚

# Time-out!

🕒 5 min.

