

Testing Automation Y Performance Turno Tarde



La importancia de Git



Para comprender mejor la necesidad de uso de Git y su impacto en el control de versiones, iniciaremos con la siguiente actividad práctica que tiene como objetivo cambiar el color de fondo de tu página web en cinco ocasiones diferentes.



¡Manos a la obra!

!Recuerda primero crear una nueva carpeta con el nombre de "Clase 1 - Introducción a Git" y hacer una copia de los documentos sobre los que venías trabajando en el curso de "Programación Web desde Cero".

- 1.** Abrir la carpeta en Visual Studio Code.
- 2.** Ir al archivo donde están definidos los estilos CSS, como por ejemplo "styles.css".
- 3.** Encontrar la propiedad que establece el color de fondo del sitio ("background-color").
- 4.** Cambiar su valor a un color diferente. Por ejemplo, puedes cambiarlo a #FF0000 para obtener un fondo rojo.
- 5.** Guardar los cambios.
- 6.** Repetir estos pasos cuatro veces más para realizar un total de cinco cambios de color de fondo en tu página web. **Deberás mantener una copia de cada versión.**
- 7.** Elegir la versión que consideres conveniente para tu página web.

¡Fin de la actividad!



¿Qué ha sucedido hasta ahora? ¿Tienes distintas carpetas con nombres que indiquen el color que has usado en cada uno? ¿Has quintuplicado tu código para poder hacer los cambios?



Como verás, esto no es muy conveniente a la hora de trabajar con versiones. Imagina un proyecto mucho más complejo y con varios colaboradores, habría cientos de carpetas por lo que esta metodología resultaría poco práctica y confusa.



Para resolver esto, ¡existe Git!

Git es un sistema de control de versiones que permite realizar un seguimiento de los cambios en el código fuente u otros archivos de texto. Funciona como un diario de bitácora, registrando todas las modificaciones que tú y tu equipo realicen.

Si alguna vez has trabajado con documentos en Google Drive donde puedes ver las colaboraciones que han hecho otros, entenderás rápidamente la importancia de este sistema.

Pasemos entonces a ver cómo tener un control más eficiente sobre las versiones de nuestro proyecto pero antes, para ello, deberemos instalar la herramienta de Git.

< >





Instalación de Git

< >

Ahora que hemos comprendido la importancia de Git, es momento de instalar esta poderosa herramienta en tu computadora para poder comenzar a utilizarlo en tu proyecto.

A continuación, te compartimos el video de instalación, o bien, puedes optar por seguir el paso a paso descrito abajo del mismo:



Pasos para instalar Git:

1. Ingresar al sitio web oficial de Git <https://git-scm.com/downloads>.
2. Seleccionar la versión de Git correspondiente a tu sistema operativo (Windows, MacOS o Linux).
3. Hacer clic en el enlace de descarga para iniciar la descarga del instalador. (Por ejemplo, para Windows deberás hacer clic en el primer link de “Click here to download”).
4. Una vez que la descarga haya finalizado, ejecutar el archivo de instalación.
5. Hacer clic en “Next” y elegir la carpeta donde ubicar los archivos de Git (como por ejemplo, podría ser la carpeta de “program files”).
6. Al momento de elegir los componentes que deseas instalar, dejar los que ya están seleccionados y hacer clic en “Next”.
7. Continuar haciendo clic en “Next” hasta llegar al botón de “Install”.
8. Hacer clic en “Install” y, por último en “Finish”.

¡Listo! 🎉 Una vez que la instalación haya finalizado, podemos verificar si Git se ha instalado correctamente abriendo la *terminal de Git Bash* y ejecutando el siguiente comando: `git --version`.

Esto debería mostrar la versión de Git instalada en tu sistema:



Testing Automation Y Performance Turno Tarde ⓘ

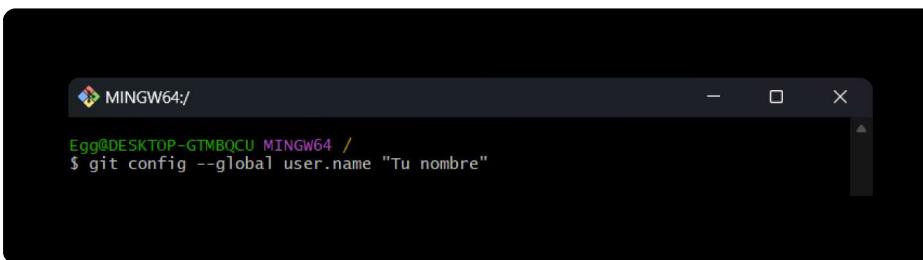
```
$ git --version  
git version 2.40.1.windows.1  
Egg@DESKTOP-GTMBQCU MINGW64 ~  
$ |
```

Configuración inicial de Git

Después de instalar Git, es importante realizar una configuración inicial utilizando el comando **“git config”**. Esta configuración es necesaria ya que Git la requerirá para poder identificar al autor de los cambios y asegurar la trazabilidad de los cambios.

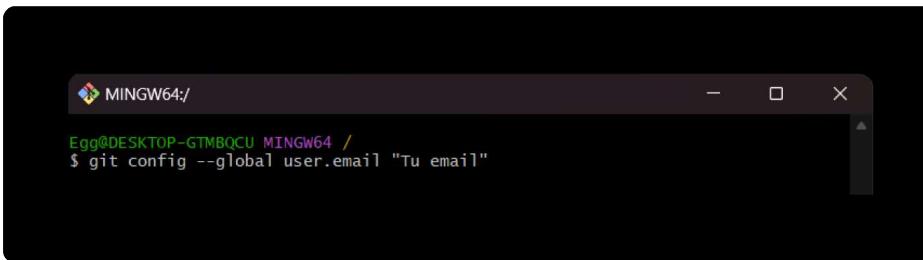
La configuración inicial consiste en establecer tu nombre de usuario y tu dirección de correo desde la **terminal de Git Bash**:

- **Nombre de usuario** → Usaremos el comando de **git config --global user.name “Tu nombre”**, como por ejemplo: *git config --global user.name “Julián Álvarez”*.



```
Egg@DESKTOP-GTMBQCU MINGW64 /  
$ git config --global user.name "Tu nombre"
```

- **Dirección de correo** → Usaremos el comando de **git config --global user.email “Tu email”**, como por ejemplo: *git config --global user.email “julian@egglive.com”*.



```
Egg@DESKTOP-GTMBQCU MINGW64 /  
$ git config --global user.email "Tu email"
```

Ahora que has realizado la configuración inicial, estás list@ para comenzar a gestionar tus proyectos en Git. ¡Adelante!



Repositorio de Git local

< >

Teniendo ya instalado y configurado Git en nuestro ordenador, pasaremos a entender cómo crear un repositorio local.

Un **repositorio** es un espacio centralizado donde se almacena, organiza y mantiene la información.

Vendría a ser “la carpeta” o **espacio donde guardaremos el proyecto** para más adelante compartirlo con otros colaboradores a través de un repositorio en la nube (como por ejemplo, en Github).

¡Manos a la obra!

1. Crear una nueva carpeta en tu ordenador con el nombre de *“Actividad de Git”*, hacer clic derecho sobre la misma y seleccionar la opción de **“Git Bash Here”**.

2. Escribir en la terminal el siguiente comando: **git init** y dar un **“enter”**.

```
MINGW64:/c/Users/Egg/Desktop/Actividad de Git
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git
$ git init
Initialized empty Git repository in C:/Users/Egg/Desktop/Actividad de Git/.git/
```

Al ejecutar **“git init”**, **estás creando un nuevo repositorio Git vacío** (o reiniciándolo si estarías trabajando sobre uno existente). Deberías ver un mensaje como este: *“Initialized empty Git repository in /ruta/a/tu/directorio/.git/”*.

3. Ahora, ejecutar el comando de: **git status**.

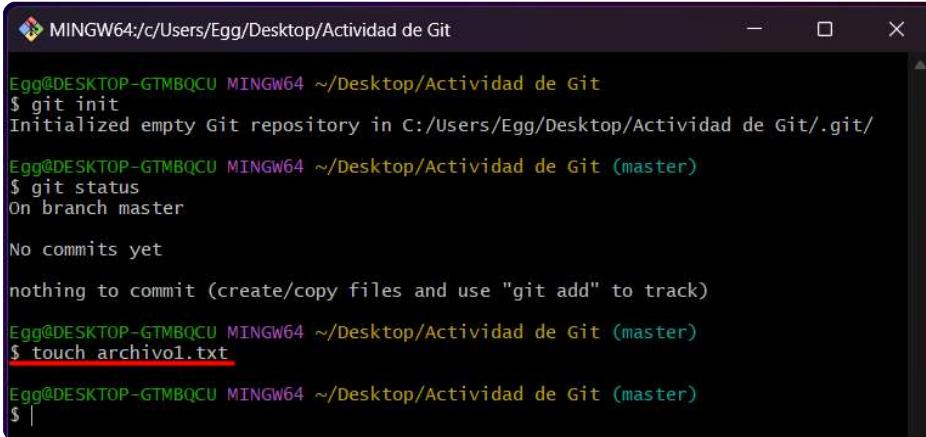
```
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git (master)
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

Con **“git status”** estamos chequeando nuestro repositorio. Es decir, nos devolverá un mensaje con el estado actual del mismo.

En esta instancia vamos a poder observar que aún solo **estamos en nuestro directorio de trabajo** (*“working directory”*), posicionados en la rama (*“branch”*) principal del mismo (*“master”* o también puede figurar como *“main”*).

4. Crear un nuevo archivo en el *“directorio de trabajo”*. Puedes hacer esto con un editor de texto o desde la línea de comandos usando, por ejemplo, el comando de: **touch archivolt.txt**.





```
MINGW64:/c/Users/Egg/Desktop/Actividad de Git
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git
$ git init
Initialized empty Git repository in C:/Users/Egg/Desktop/Actividad de Git/.git/
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git (master)
$ touch archivo1.txt

Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git (master)
$ |
```

Este comando crea un nuevo archivo llamado "archivo1.txt" el cual deberías ahora poder ver dentro de la carpeta de "Actividad de Git".

En este punto, **este archivo está en tu área de trabajo. Eso significa que Git sabe que el archivo está ahí, pero no está siguiendo los cambios del mismo todavía.**

5. Hacer que Git comience a rastrear los posibles cambios que hagamos en el archivo de "archivo1.txt". Para esto, usar el comando de: **git add archivo1.txt**.

Testing Automation Y Performance Turno Tarde ⓘ

Ahora, el "archivo1.txt" está en el **área de preparación** ("staging area"). **Git sabe que quieres seguir los cambios que realices en este archivo.**

💡 Para verificar si funcionó podemos volver a usar el comando de "git status". El archivo debería figurar debajo de "Changes to be committed".

6. Finalmente, para hacer un commit de los cambios que agregaste al **área de preparación**, usar el siguiente comando: **git commit -m "Mi primer commit"**.

```
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Actividad de Git (master)
$ git commit -m "Mi primer commit"
[master (root-commit) fb66096] Mi primer commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 archivo1.txt
```

Este comando toma todos los cambios que están en el área de preparación y los guarda en el repositorio. En este punto, Git sabe que este es un punto de cambio que quieras conservar y al que puedes volver si es necesario.

Recuerda que si quieres verificar el estado de tu repositorio en cualquier momento, puedes usar el comando **git status**. Y si quieres ver un registro de todos tus commits, puedes usar **git log**.

💡 Hecho el primer commit, puedes seguir realizando nuevos commits para registrar los cambios adicionales que realices en el proyecto. **Ten presente que antes de realizar un nuevo commit, es necesario agregar nuevamente los archivos modificados al área de preparación utilizando el comando "git add". Una vez que los archivos están en el área de preparación, puedes ejecutar el**

comando: `git commit -m "Mensaje descriptivo"` (para crear el nuevo commit con un mensaje que describa los cambios realizados).



< >

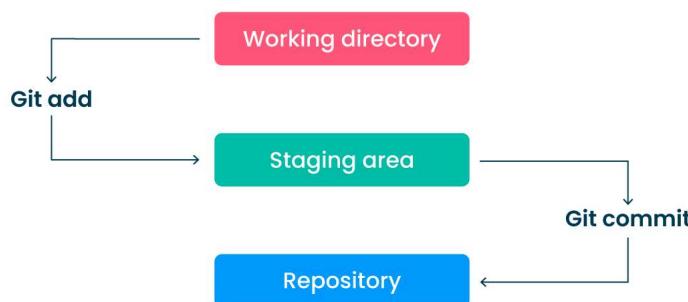




Conceptos esenciales de Git

< >

Git maneja tu código en tres "espacios" o "áreas" distintas, que te permiten organizar y controlar tus cambios de manera eficiente. Estos son: el **Working Directory** (Directorio de trabajo), el **Staging Area** (Área de Preparación) y el **Repository** (Repositorio).



Utilizaremos la siguiente metáfora para comprender mejor estas áreas. Imagina que estás escribiendo un libro:

- **Working Directory:** Tu escritorio está lleno de notas, borradores y tazas de café. Aquí es donde estás haciendo los cambios. En términos de Git, el *Working Directory es el lugar donde se hacen las modificaciones en los archivos*. Los cambios en los archivos en este directorio aún no se han preparado para un "commit". Podemos utilizar el comando **"git status"** para ver los cambios que has realizado en el *Working Directory* pero que aún no has preparado para un commit. Esto mostrará una lista de los archivos modificados.
- **Staging Area:** Una vez que estás conforme con un capítulo de tu libro, lo apartas en una carpeta aparte, listo para ser enviado a tu editor. En Git, este es el concepto del *Staging Area*, que es *donde se "preparan" los cambios antes de hacer un commit*. Para *mover archivos desde el Working Directory al Staging Area*, se utiliza el comando **"git add"**. Por ejemplo, para agregar todos los archivos modificados al Staging Area, puedes utilizar **"git add ."** (el punto hace que se añadan todos los archivos modificados).
- **Repository:** Por último, una vez que tu editor ha revisado y aprobado un capítulo, lo publica. En Git, esto se llama hacer un *"commit"*. Los cambios que se han "commitido" están ahora en el repositorio y forman parte del historial de tu proyecto. Para hacer un commit de los cambios en el Staging Area, se utiliza el comando: **git commit -m**

Testing Automation Y Performance Turno Tarde ⓘ

INFORMACIÓN RELEVANTE.

Entonces, en resumen:

- **git status** → Muestra los archivos que han cambiado desde el último commit y los cambios preparados para el próximo commit.
- **git add** → Mueve los cambios desde el *Working Directory* al *Staging Area*, preparándolos para el próximo commit.



Hola, MARTINA
EQUIPO




**Por favor, espera unos n
para que los equipos
conformados**

Mientras tanto puedes acc
videollamada desde el
"Encuentro virtual"

• • •

- **git commit** → Mueve los cambios del *Staging Area* al *Repository*, actualizando el historial de tu proyecto.
- **git log** → Permite ver el historial de commits.

Estos conceptos de Git son fundamentales para comprender cómo manejar tus cambios. A partir de los mismos, podrás explorar características más avanzadas como las ramas (branches) con mayor facilidad.





Desafío de Git

< >

Ya comprendimos que crear un repositorio nos permitirá tener un control más eficiente sobre las versiones de nuestros proyectos. Por esto, **el siguiente desafío tiene como objetivo crear un repositorio para tu página web.**

De esta manera, prepararemos el terreno para realizar las modificaciones necesarias sin la necesidad de estar creando múltiples carpetas como experimentamos en la primera actividad del día de hoy.

Recomendamos que una persona de la mesa de trabajo pueda compartir pantalla para que lo vayan resolviendo en conjunto.

¡A resolver!

Primero vamos a eliminar todos los archivos de la carpeta "Clase 1 -

Testing Automation Y Performance Turno Tardé ⓘ

1. Sobre la carpeta "Clase 1 - Introducción a Git" hacer clic derecho y seleccionar la opción "**Git Bash Here**".
2. En la terminal, **inicializar un nuevo repositorio Git** para esa carpeta. *Recuerda usar el comando "git status" para verificar el estado del repositorio.*
3. Desde Visual Studio Code, hacer un cambio que deseas en el *archivo HTML* y otro cambio en el *archivo CSS*.
4. Agregar todos los archivos de la carpeta al "**área de preparación**". Esto preparará los archivos para el próximo commit.
5. Realizar el **commit** de los cambios con una breve descripción de los mismos. Debería aparecer un mensaje confirmando la aplicación de los cambios.
6. Ejecutar el comando de **git log** para verificar que el commit se haya efectuado correctamente.

¡Desafío terminado! 🎉 ¿Lograron crear el repositorio y realizar el commit?

Una vez hecho el ejercicio, puedes ver siguiente video por si quedaron dudas de cómo resolverlo:



Desafío de Git | Introducción a Git | Sistema de control de versiones | Egg



En los próximos pasos, profundizaremos en conceptos más avanzados y útiles para el control de versiones. ¡Buen trabajo!

< >





Ramas o "branches" en Git

< >

Muy bien, ya entendimos cómo funciona esencialmente Git y hemos creado nuestro repositorio.

Ahora prepárate para cambiar nuevamente los colores de fondo de tu sitio web, pero esta vez sin crear carpetas distintas para cada cambio. Utilizaremos las poderosas ramas de Git para mantener tus modificaciones organizadas y controladas.

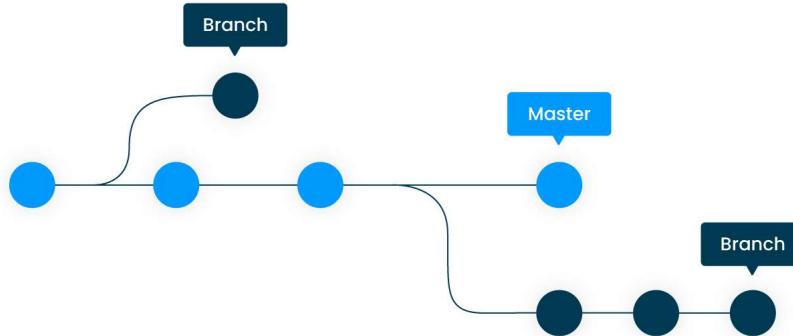
En la terminal de Git Bash, puedes colocar el comando **clear** para limpiar la pantalla y eliminar el historial de comandos y resultados anteriores.

¡Manos a la obra!

1. Dentro de la terminal de Git Bash, chequear de estar dentro de la carpeta "*Clase 1 - Introducción a Git*". En caso de no estarlo, navegar al directorio de la carpeta utilizando el comando: **cd "Ruta de la carpeta"** ("change directory"), como por ejemplo: **cd "Desktop | Clase 1 - Introducción a Git"**.
2. Verificar de estar en la rama principal (por lo general, se llama "master" o "main") ejecutando el comando: **git branch**. Si no estás en la rama principal, cambia a ella usando el comando: **git switch main** (o el nombre de tu rama principal).
3. Crear una nueva rama para realizar los cambios de color de fondo utilizando el comando: **git branch nombre-de-la-rama**. Por ejemplo, puedes usar "**git branch cambios-fondo-1**" para crear una nueva rama llamada "*cambios-fondo-1*".
4. Cambiar a la nueva rama que has creado utilizando el comando: **git switch nombre-de-la-rama**. Por ejemplo, "**git switch cambios-fondo-1**".
5. Ir al archivo CSS de tu página web en Visual Studio Code.
6. Encontrar la propiedad que establece el color de fondo ("background-color") y cambiar su valor a otro color de tu elección.
7. Guardar el archivo CSS con el cambio realizado.
8. Registrar los cambios en la rama utilizando los comandos: **git add .** (para agregar los archivos modificados) y **git commit -m "Mensaje descriptivo"** (para realizar un commit con un mensaje que describa los cambios que has realizado). Por ejemplo, puedes usar: **git commit -m "Cambiar color de fondo en la rama cambios-fondo-1"**.
9. Repetir 4 veces más los pasos del 5 al 8 para realizar cambios de color de fondo adicionales en la misma rama.

¡Excelente! 🎉 Hasta este punto, has logrado crear una "rama" en la cual has aplicado cinco cambios de color de fondo diferentes (**No cierres la terminal de Git ya que luego continuaremos avanzando sobre este ejercicio**).

Ahora bien, profundicemos en: ¿qué es una rama? En Git, **una rama es como un camino paralelo en el desarrollo de tu proyecto**



Imagina que estás trabajando en una línea de tiempo principal, llamada "*master*" o "*main*", que contiene la versión estable de tu proyecto. Ahora, supongamos que quieras probar una nueva característica o arreglar un error sin afectar la línea de tiempo principal. En este caso, probaste distintos colores de fondo. Aquí es donde las ramas entran en juego.

Puedes pensar en las ramas como caminos separados en una carretera. Cuando creas una rama, estás haciendo una copia del proyecto en ese punto específico de la línea de tiempo.

Puedes realizar cambios, experimentar e incluso agregar nuevas características en esa rama sin afectar la línea de tiempo principal. Una vez que estés conforme con los cambios en la rama, puedes fusionarla ("merge") de nuevo con la línea de tiempo principal para incorporar esos cambios en la versión estable del proyecto.

¡Es como un "Back to the future" pero aplicado a la programación!

En resumen, **las ramas te permiten trabajar en diferentes características o correcciones de manera aislada, facilitando la colaboración y el mantenimiento de un proyecto.**

Te compartimos el siguiente video donde repasamos el proceso de creación de una rama, junto con otros consejos importantes a tener en cuenta:

Branches en Git | Introducción a Git | Sistema de control de versiones | Egg



< >

Testing Automation Y Performance Turno Tardé ⓘ



Git Merge



En el ejercicio anterior, logramos crear una rama donde aplicamos cinco cambios de color de fondo diferentes. Ahora es el momento de aprender cómo incorporar esos cambios en la línea de tiempo principal (*"main"*) de nuestro proyecto.



Recapitulemos... En el punto 8 fuimos realizando los **commits** para cada cambio de color. Estos cambios fueron guardados en la rama en la que estábamos trabajando (como por ejemplo, la rama *cambios-fondo-1*).



Ahora chequeemos nuevamente la rama *main*, **¿los cambios se han aplicado?** (Recuerda que puedes hacerlo ejecutando el comando *"git switch main"* para cambiar a la rama principal y luego revisar si los cambios están presentes en la página web).

Probablemente los cambios no estén aplicados. Esto sucede porque los has aprobado en la rama pero esto no está reemplazando el código de la rama *main*.

Entonces, ¿cómo hacemos entonces para que los cambios se vean en la rama principal?

Para esto, vamos a tener que fusionar la rama actual con la rama principal utilizando el comando: **git merge**.

Para realizar esta fusión usando *git merge* es necesario seguir estos pasos:

1. Cambiar a la rama en la que deseas fusionar los cambios (generalmente la rama principal, como *"main"* o *"master"*) con el comando: **git switch "nombre-de-la-rama-destino"**. En este caso sería: *git switch main*.
2. Ejecutar el comando *git merge* especificando la rama que deseas fusionar: **git merge "nombre-de-la-rama-fuente"**, como por ejemplo: *git merge cambios-fondo-1*.

```
Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Clase 1 - Introducción a Git (cambios-fondo-1)
$ git switch main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Egg@DESKTOP-GTMBQCU MINGW64 ~/Desktop/Clase 1 - Introducción a Git (main)
$ git merge cambios-fondo-1
Updating 0a0620e..05148d7
Fast-forward
  style.css | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
```

Git intentará fusionar automáticamente los cambios de *"rama-fuente"* en *"rama-destino"*. Si hay conflictos que Git no puede resolver automáticamente deberás solucionarlos manualmente antes de completar el proceso de fusión. Esto lo veremos en detalle en la próxima clase.

¡Felicitaciones! Has fusionado con éxito los cambios de color de fondo en la línea de tiempo principal utilizando el recurso de las *"ramas"*.

Utilizando el comando de **git log** deberías poder ver los diferentes commits que fuiste realizando para cambiar los colores.

Por último, te compartimos el video donde aplicamos el comando de git merge que acabamos de aprender:

Git Merge | Introducción a Git | Sistema de control de versiones | Egg

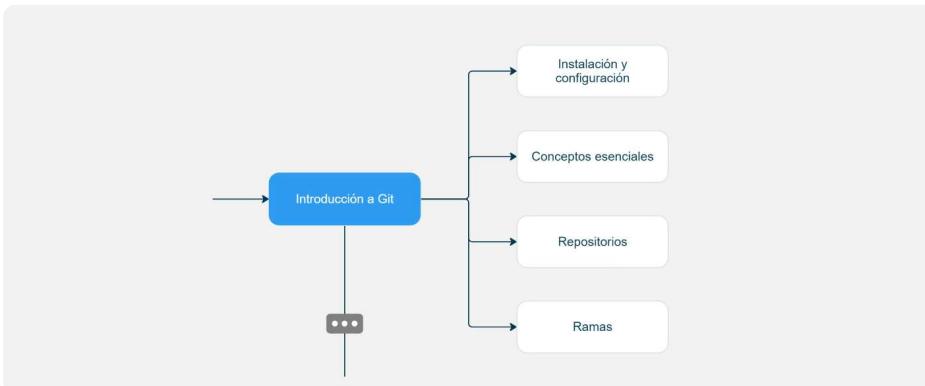


< >



Testing Automation Y Performance Turno Tardé ⓘ

Mapa de conceptos



En la clase de hoy has visto cómo instalar y configurar Git en un entorno local. Aprendiste por qué es necesario realizar cambios en el código en distintas versiones, creaste tu repositorio local y aplicaste cambios en tu sitio web utilizando el recurso de las ramas.

Recuerda que la práctica hace al maestro así que sigue ejecutando cambios con Git y aplicando lo que aprendiste hoy. No dudes en volver a consultar esta clase si necesitas refrescar tus conocimientos o repasar algún concepto.

¡Hasta la próxima! 🌟

