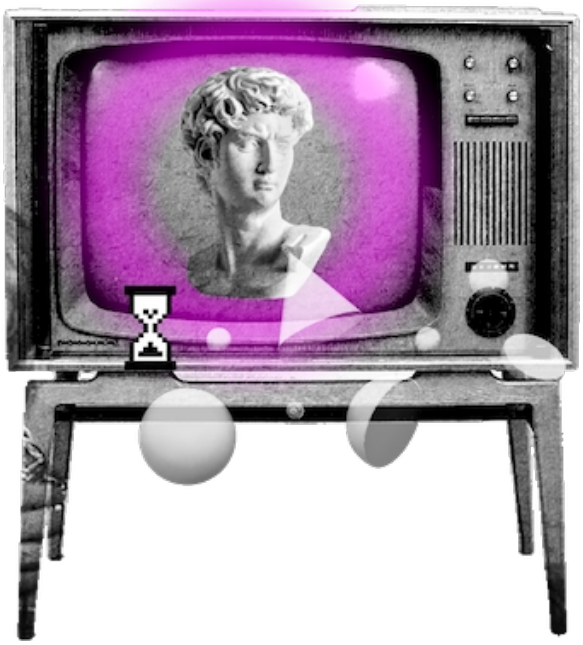
NUCBA

DISEÑO WEB

GIT + GITHUB I



APUNTES

Sistema de control de versiones: Git + Github

¿Qué es la consola?

La consola es una **interfaz de línea de comandos** (CLI, por sus siglas en inglés) que permite a los usuarios **interactuar con el sistema operativo** o con **un programa** en particular. En lugar de utilizar una interfaz gráfica de usuario (GUI), los usuarios escriben comandos en la consola para realizar tareas específicas.

La consola es una herramienta muy útil para los desarrolladores y administradores de sistemas, ya que les permite realizar tareas de forma más **eficiente y automatizada** que con una interfaz gráfica de usuario. Además, muchos programas y herramientas solo están disponibles a través de la consola, lo que la hace esencial para muchas tareas técnicas.

La consola es una característica común en sistemas operativos como Windows, macOS y Linux, y se puede acceder a través de la terminal o la línea de comandos en cada sistema operativo.

¿Qué es un sistema de control de versiones?

En el desarrollo web, un **sistema de control de versiones** es una herramienta que permite llevar un **registro de los cambios** realizados en el código fuente de una aplicación o sitio web. Esto permite a los desarrolladores colaborar de manera efectiva, rastrear y revertir cambios, y mantener diferentes versiones y ramas del proyecto.

¿Por qué es necesario un sistema de control de versiones?

Hay varias razones por las que un sistema de control de versiones es necesario en el desarrollo web:

Colaboración: Permite a múltiples desarrolladores trabajar en el mismo proyecto de manera coordinada y efectiva.

Historial de cambios: Registra todos los cambios realizados en el código y permite revertirlos si es necesario.

Manejo de ramas: Permite la creación y gestión de ramas separadas del proyecto para el desarrollo de nuevas funcionalidades y el mantenimiento de versiones estables.

Control de calidad: Ayuda a garantizar la calidad del código al permitir la revisión y aprobación de cambios antes de que se integren en la rama principal.

Documentación: Ofrece una visión detallada del desarrollo del proyecto y permite una comprensión más clara del código.

En definitiva, un sistema de control de versiones es una herramienta esencial para el desarrollo web que permite una gestión más efectiva y colaborativa de nuestros proyectos.

¿Qué es GIT?

Git es un sistema de control de versiones creado por Linus Torvalds en 2005 con el objetivo de desarrollar el kernel de Linux de manera más eficiente.

Permite a los desarrolladores llevar un **registro** de todos los **cambios** realizados en el código, **trabajar en equipo** en proyectos simultáneamente, **crear y gestionar ramas** separadas del proyecto y **revertir cambios** si es necesario. Además, Git es una herramienta ampliamente utilizada en el desarrollo de software y se integra con una amplia variedad de servicios y herramientas en línea, lo que lo convierte en una de las **herramientas de control de versiones más populares y utilizadas** en la actualidad.

Gitbash - La terminal de línea de comandos de Git

GitBash es una **terminal de línea de comandos** diseñada para trabajar con **Git.**

Proporciona una **interfaz** en la línea de comandos para realizar tareas relacionadas con Git, como clonar repositorios, crear ramas, fusionar cambios y hacer push y pull de tus cambios a un repositorio remoto.

Es además una **aplicación multiplataforma** que se ejecuta en Windows, macOS y Linux, lo que lo hace una herramienta ideal para desarrolladores que trabajan en diferentes sistemas operativos.

Recomendamos utilizar **Gitbash** para todas aquellas tareas en las cuales debamos utilizar la consola durante el transcurso de la carrera.

¿Qué es GITHUB?

GitHub es una plataforma en línea que brinda servicios de **alojamiento de control de versiones** y colaboración para proyectos de software. GitHub es esencialmente un servicio basado en la web para **alojar y compartir repositorios de Git**, lo que permite a los desarrolladores trabajar juntos en proyectos de software de manera más eficiente.

Además de alojar repositorios Git, **GitHub** ofrece una amplia variedad de herramientas y características para colaborar en proyectos, incluyendo **seguimiento de problemas, pull requests, wikis, estadísticas y más**. **GitHub** es una plataforma ampliamente utilizada en la comunidad de desarrollo de software y es uno de los servicios de control de versiones **más populares** en la actualidad.

¿Qué es un repositorio?

En el contexto de sistemas de control de versiones, como **Git**, un **repositorio** es una **colección de archivos que están bajo el control de versiones** y que se utiliza para almacenar y gestionar el historial de cambios en un proyecto.

Un **repositorio** puede incluir cualquier tipo de archivo, incluyendo código fuente, documentación, imágenes, etc.

Creando un repositorio

Para crear un nuevo repositorio en **GITHUB** , debemos seguir los siguientes pasos:

1. **Inicia sesión en GitHub**: Inicia sesión en tu cuenta de GitHub o crea una nueva cuenta si aún no tienes una.

2. **Iniciar la creación**: Haz clic en el botón "New repository" en la página principal de tu perfil de GitHub.

3. **Configura el repositorio**: Da un nombre y descripción al repositorio y selecciona la visibilidad del repositorio **(público o privado)**. También puedes elegir inicializar el repositorio con un archivo README ya creado.

4. **Crear repositorio**: Haz clic en el botón "Create repository" para crear el repositorio.

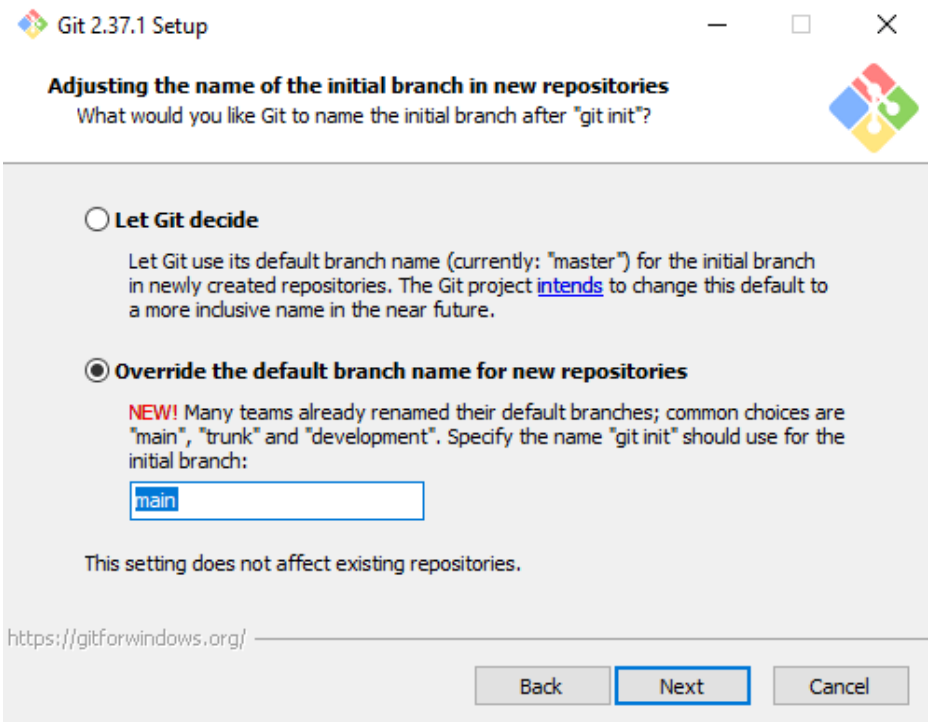
Configurando GIT

Una vez realizados los pasos anteriormente mencionados, ya tenemos nuestro repositorio listo para alojar un proyecto.
Veamos cuales son los pasos a realizar para empezar a subir archivos desde nuestra área de trabajo local a dicho repositorio remoto:

1. Descargar e instalar GIT

Descargamos e instalamos la última versión de Git para nuestro sistema operativo desde la página oficial de Git **(https://git-scm.com/downloads)**.

IMPORTANTE *: Durante la instalación nos aseguramos de que se designe la rama* ***"main"*** *como la rama principal de nuestro proyecto. Para ello, verificamos que al instalar , en el paso que corresponde al* ***ajuste de la rama inicial de un proyecto****, figure de la siguiente manera:*



2. Configurar identidad

Configura tu nombre de usuario y dirección de correo electrónico, que serán utilizados para identificar tus cambios en el control de versiones.

```
git config --global user.name "Tu nombre"
git config --global user.email "Tu mail que usaste en github"
```

Es importante que **el email que utilices sea el mismo con el cual te registraste en GITHUB.**

3. Configurar rama principal

En caso de que no lo hayas realizado en la instalación, puedes colocar el siguiente comando en la consola para poder configurar la rama **main** como la rama principal de todos los proyectos que inicialices de ahora en más en tu área de trabajo local.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git config --global init.defaultBranch main
```

Una vez realizados estos 3 pasos, ya podemos comenzar a ver los comandos que nos permitirán subir un proyecto que tengamos en una carpeta local al almacenamiento en la nube que nos otorga **Github**.

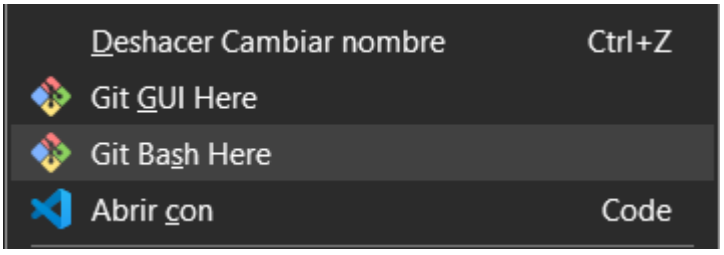
Subir un proyecto a Github

Al iniciar un proyecto de cero, normalmente lo primero que haremos es **crear una carpeta en nuestro disco duro** en la cual vayamos a contener todos los archivos necesarios de nuestro proyecto.
Por otro lado, tendremos que crear un repositorio en **GITHUB** en el cual podamos almacenar nuestro proyecto de forma remota.

1. Posicionarse en la carpeta del proyecto

Para poder subir proyectos a la nube vamos a estar utilizando mucho la consola. En este caso, tal como mencionamos anteriormente, utilizaremos **Git Bash.**

La consola no sabe donde nosotros queremos inicializar un repositorio con **GIT** , por lo tanto, es nuestro deber asegurarnos de que dentro de la consola estamos posicionados dentro de la carpeta que queremos manejar mediante nuestro sistema de versiones. La mejor manera de asegurarnos de esto es abriendo en el explorador de archivos nuestra carpeta , hacer click derecho y presionar la opción **“Git Bash here”**.



Esto hará que se abra la aplicación de Gitbash y que la misma comience a trabajar a partir del directorio de nuestro proyecto.

2. GIT INIT

Este comando convertirá nuestro almacenamiento local en un repositorio de git. Creará una carpeta **“.git”** dentro de la carpeta de nuestro proyecto (Es una carpeta que por defecto se encuentra oculta, si quieren verla deberán habilitar la opción de ver las carpetas ocultas en su explorador de archivos).

Este comando se utiliza sólo la primera vez que se quiere subir el repositorio a la nube.

3. GIT ADD

Este comando se utiliza para **preparar archivos** para la etapa de confirmación (commit) en el repositorio local.

Se puede agregar un archivo a este etapa de manera individual o bien podemos agregar múltiples archivos a la vez, pero normalmente vamos a estar agregando todos los archivos directamente a esta etapa de preparación, conocida también como **stage.**

```
PC@DESKTOP-JSA96B6 MINGW64 ~/Desktop
$ git add *nombre-del-archivo*
```

Subida de un solo archivo a la etapa de preparación.

```
PC@DESKTOP-JSA96B6 MINGW64 ~/Desktop
$ git add *archivo1* *archivo2*
```

Subida de múltiples archivos a la etapa de preparación.

```
PC@DESKTOP-JSA96B6 MINGW64 ~/Desktop
$ git add .
```

Subida de todos los archivos a la etapa de preparación.

Es importante mencionar que una vez que un archivo se haya subido a la etapa de preparación, en caso de hacerle una modificación, se deberá volver a añadir a esta etapa.

4. GIT COMMIT

Este comando es una de las herramientas más importantes en el flujo de trabajo de Git. Es el comando que utilizamos para **confirmar los cambios realizados** en el **repositorio local** y **registrarlos** en la **historia de versiones de Git**.

Al ejecutarlo, Git toma una **instantánea** de todos los archivos agregados a la **zona de preparación (staging area**) y los **registra en la historia del repositorio**. Cada **commit** es un **punto de referencia** en la historia del proyecto que se puede consultar en cualquier momento.

Además, al realizar un **commit**, es importante **incluir un mensaje** que **describa** los **cambios realizados** en el repositorio. Esto ayuda a los demás desarrolladores a comprender que es lo que se realizo en dicho commit.

```
PC@DESKTOP-JSA96B6 MINGW64 ~/Desktop
$ git commit -m 'añadido el footer del proyecto'
```

Commit que hace referencia al hecho de haber añadido un footer a un proyecto.

5. GIT REMOTE ADD

Este comando lo utilizaremos para **agregar** un **repositorio remoto** a un **repositorio local** de **GIT**, por lo tanto, este comando es el que nos permitirá establecer un vínculo entre ambos , de manera tal que se puedan **sincronizar** los cambios entre ambos repositorios, **subir** los cambios locales al repositorio remoto, **descargar** los cambios del remoto a local , etc.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git remote add origin *link del remoto*
```

Este comando se utiliza sólo la primera vez que se quiere subir el repositorio a la nube.

6. GIT PUSH

Este comando se utiliza para enviar los cambios de un repositorio local a otro en línea, como lo es **Github.**

Se utiliza luego de haber hecho un **commit** a los cambios que estaban en la **zona de preparación** con el fin de transferir dichos cambios al repositorio remoto.

Al utilizarlo por primera vez en un repositorio, debemos colocarlo de la siguiente manera:

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git push -u origin main
```

Luego, al subir más cambios al mismo repositorio, bastará con colocar **git push**.

Este es el último paso a realizar para subir un proyecto a nuestro repositorio remoto.

A partir de este punto, podemos seguir realizando cambios a nuestro repositorio local y subiendo dichos cambios a **github** utilizando los mismos comandos con la excepción de **git init** y **git remote add** , ya que solo son necesarios la primera vez que se realice la subida.

Extra. GIT STATUS

Si bien no es un paso obligatorio para subir nuestro repositorio a **Github**, es importante conocer este comando.

El mismo nos permite **ver el estado actual del repositorio local.**

Cuando se ejecuta nos muestra una lista de archivos que han sido modificados o agregados al repositorio local **desde el último commit**.

También nos muestra si existen archivos que **aún no han sido agregados a la zona de preparación** y si hay **conflictos entre las versiones local y remota** del repositorio.

Por último, también nos indica si hay **cambios que aún no se han confirmado** (No se ha realizado un commit) en el repositorio local.

GIT CLONE - Descargar un proyecto en local

Muchas veces vamos a tener que traernos un proyecto desde la nube a nuestra computadora.

Si bien Github nos permite descargar el proyecto directamente con un botón de descarga, la mejor opción es utilizar el comando **git clone**, que se utiliza para **crear una copia exacta de un repositorio de Git en el ámbito de trabajo local**.

Este comando descarga todo el **historial de versiones, archivos y metadatos del repositorio remoto** a la máquina local, lo que permite a los desarrolladores trabajar en el proyecto de forma local y actualizar el proyecto con los cambios que se realicen **(en caso de tener los permisos para hacerlo)**.

Para utilizarlo, debemos tener la url que apunta al repositorio del remoto que se quiere clonar, que lo podemos sacar clickeando en el botón verde **<> code** de nuestro repositorio en Github y copiando el link que nos aparece en el desplegable.

Finalmente, copiamos ese link luego de colocar **git clone** en la línea de comandos:

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git clone https://github.com/Nucba-Projects/coworking-js.git
```

Importante: Si utilizamos la opción “Download Zip” se nos decargará el proyecto en local pero no mantendrá la vinculación al repositorio, lo cual implica que no podremos hacer ningún tipo de modificación al proyecto y subirla a la nube **(en caso de que tuvieramos acceso y permiso para hacerlo)**

GIT PULL - Actualizando en local desde la nube

Puede suceder también, en especial si estamos trabajando entre varias personas en un mismo repositorio, que tengamos **desactualizado** el repositorio local vinculado a ese remoto que tiene **cambios nuevos.**

Para poder traernos esos cambios nuevos y no volver a descargar el repositorio, arriesgándonos a perder los cambios que nosotros hicimos en local, debemos utilizar el comando **git pull.**

Este comando se utiliza para **descargar y fusionar** cambios de un repositorio remoto en un repositorio local. Es una combinación entre el comando **git fetch** (que se utiliza para descargar los cambios del repositorio remoto) y **git merge** (se utiliza para fusionar los cambios del proyecto en la nube con el proyecto local).

Es una herramienta muy útil cuando se trabaja en un proyecto con varios desarrolladores y es necesario asegurarse de **tener la versión más actualizada del código antes de empezar a trabajar en él**.

Nucba tip: *Sigan investigando sobre los temas aquí expuestos. Practiquen, apoyense en la documentación oficial y sobre todo mantengan la constancia y la curiosidad a medida que vayan aprendiendo cosas nuevas.*

#HappyCoding 🚀