# Esta clase va a ser

**• grabada**

Certificados oficialmente por

PedidosYa

CODERHOUSE

**Clase 10.** DESARROLLO WEB

# GIT

Certificados oficialmente por


CODERHOUSE

# Temario

09

**Animaciones,transformaciones y transiciones**

- ✓ Gradientes
- ✓ Transformaciones
- ✓ Transiciones
- ✓ Animaciones

10

**GIT**

- ✓ Git
- ✓ Creando repositorios
- ✓ Ramas

11

**GitHub**

- ✓ GitHub
- ✓ Creación de cuenta
- ✓ Creación de repositorio

CODERHOUSE

# Objetivos de la clase

- **Conocer** Git.
- **Aprender** cómo manejar la terminal.
- **Usar** los comandos básicos de Git.

CLASE N°09

# Glosario

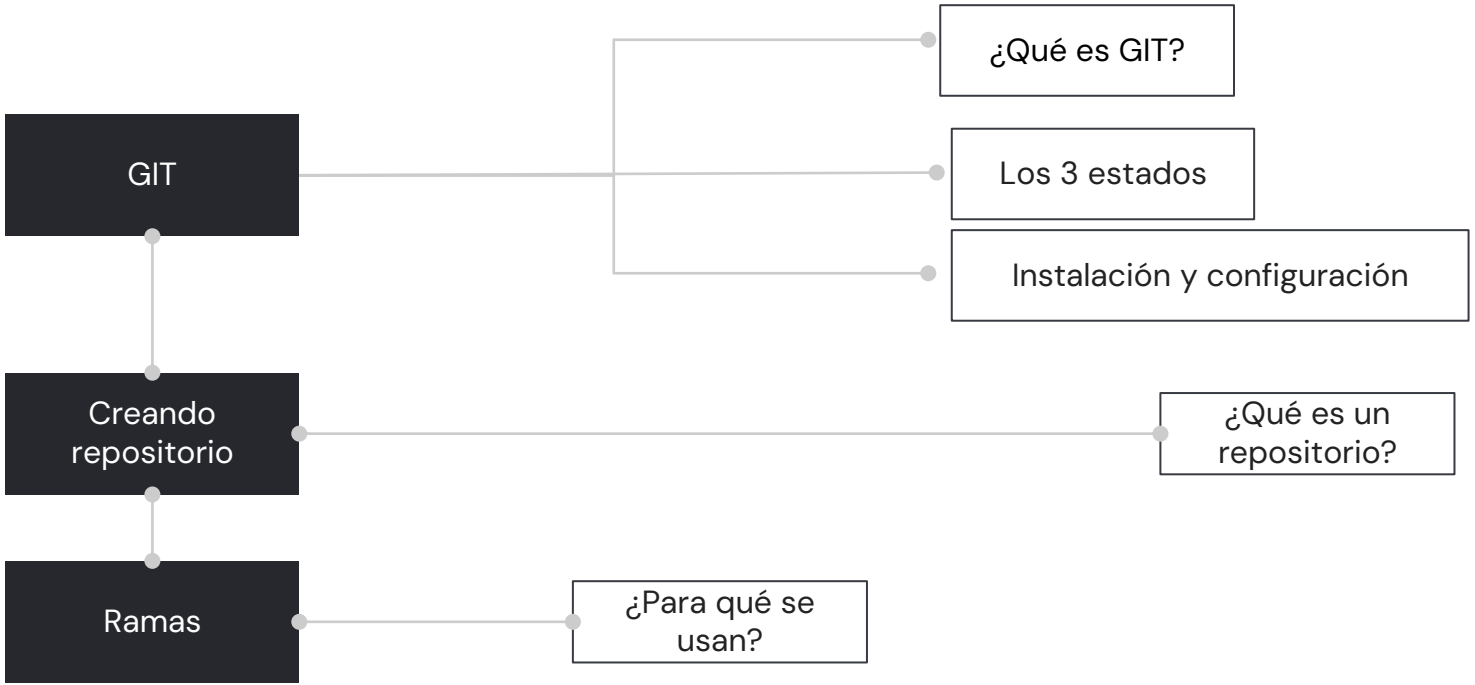
**Framework:** Es un marco de trabajo, un conjunto de herramientas y código para trabajar de acuerdo con una metodología, utilizando determinados patrones.

**Bootstrap:** Es un framework que permite crear interfaces web con CSS y JavaScript. Adapta la interfaz del sitio web al tamaño del dispositivo en que se visualice logrando sitios responsive de forma sencilla.

**CDN:** *Content Delivery Network*, una red que aloja los archivos css y js de Bootstrap, a la cual podemos acceder a través de una dirección URL para agregar Bootstrap en nuestro proyecto.

**Breakpoint:** Medidas de anchura utilizadas para el diseño responsivo. Determinan cómo se verá el diseño responsive en cada tamaño de dispositivo o viewport.

# MAPA DE CONCEPTOS



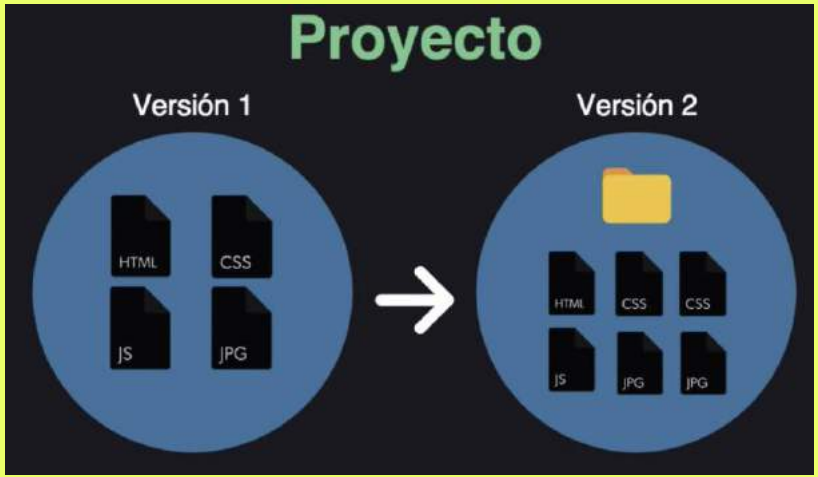
# GIT

# ¿Qué es GIT?

Git es un sistema de control de versiones gratuito y de código abierto, diseñado para manejar desde pequeños a grandes proyectos de manera rápida y eficaz. Se entiende como control de versiones a todas las herramientas que nos permiten mantener un registro de las modificaciones en nuestro proyecto. Este sistema registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.

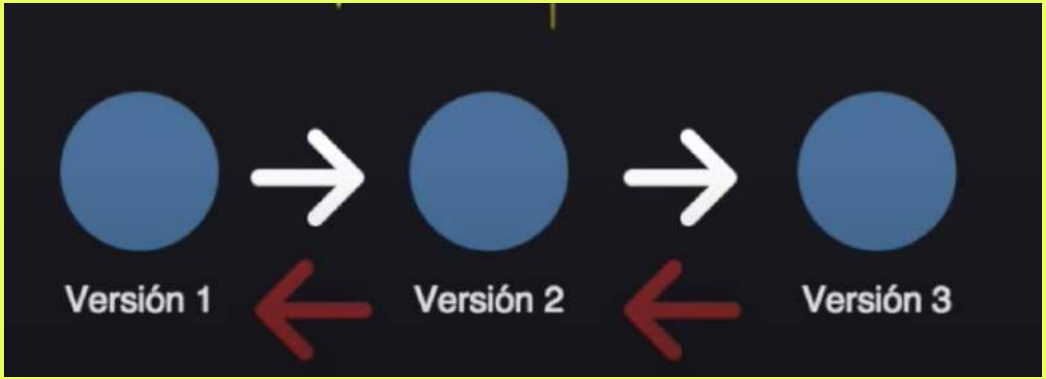

Fuente: https://git-scm.com/book/es/v2/

# ¿Qué es GIT?




GIT es una herramienta que nos permite, en cualquier momento que decidamos, obtener una “instantánea” del estado de cada elemento de nuestro proyecto. Cada una se llama “versión”.

# ¿Qué es GIT?



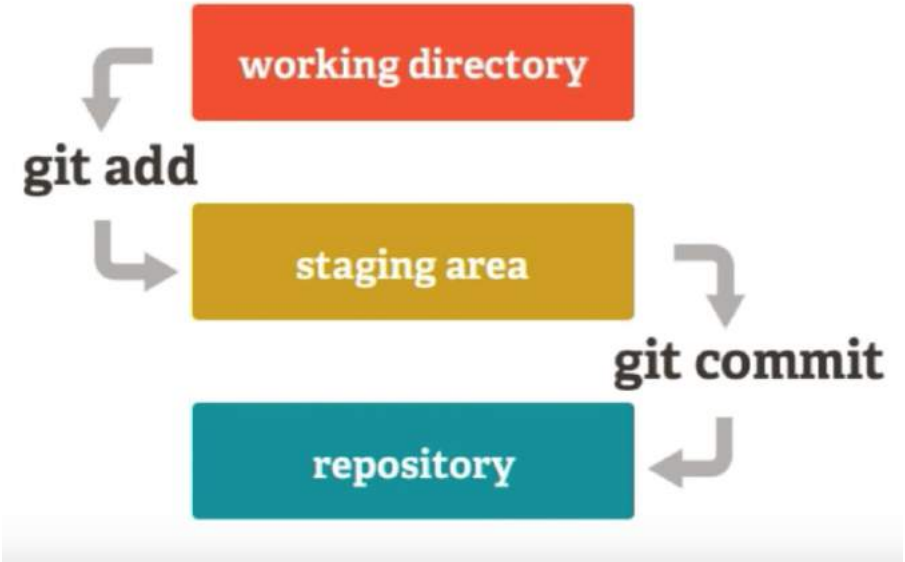

Con GIT, podemos ir a versiones anteriores, una característica muy útil para poder corregir errores, y para la organización.

# GIT los 3 estados

👉1er estado (el que trabajamos) “preparamos las cajas”.

👉2do estado (archivos listos) “agregamos las cajas listas”.

👉3er estado (registro de todos los archivos) “lote listo”.



# ¿Qué es GIT?

# Git ≠ GitHub

- ✓ Git es uno de los sistemas de control de versiones más populares entre los desarrolladores.
- ✓ GitHub es un servicio de alojamiento para los repositorios generados con el sistema Git.

¡Lo veremos la próxima clase!

# Instalación y configuración de GIT

CODERHOUSE

# Instalación y configuración de GIT

**Lo primero es lo primero: tienes que instalarlo**

Puedes obtenerlo de varias maneras, las dos principales son instalarlo desde código fuente, o instalar un paquete existente para tu plataforma. En nuestro caso, usaremos la segunda opción, tanto para Windows como para Mac.

# Instalación y configuración de GIT en Windows

CODERHOUSE

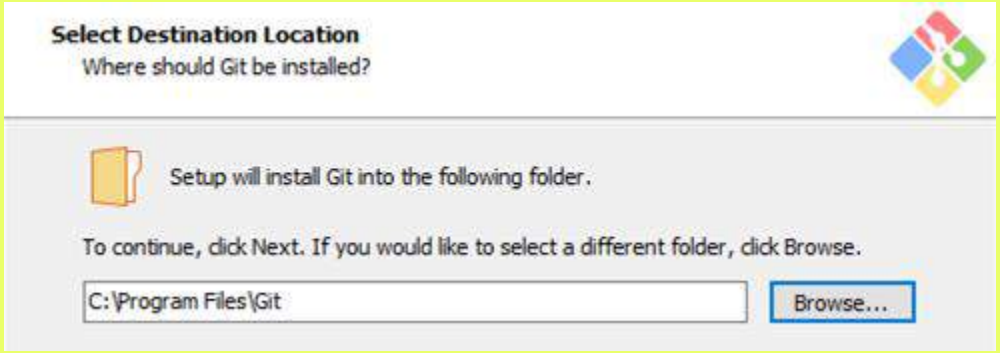
# Instalación de GIT (WINDOWS)

El primer paso es dirigirse a la página oficial de Git. A continuación vamos a descargar el archivo instalador, de acuerdo a la versión de Windows (32-bit o 64-bit) que tenga nuestra computadora.
En este momento, la versión más nueva de Git es la 2.38, pero siempre debemos prestar atención a descargar la que nos sugiera la web oficial.


CODERHOUSE

# Instalación de GIT (WINDOWS)

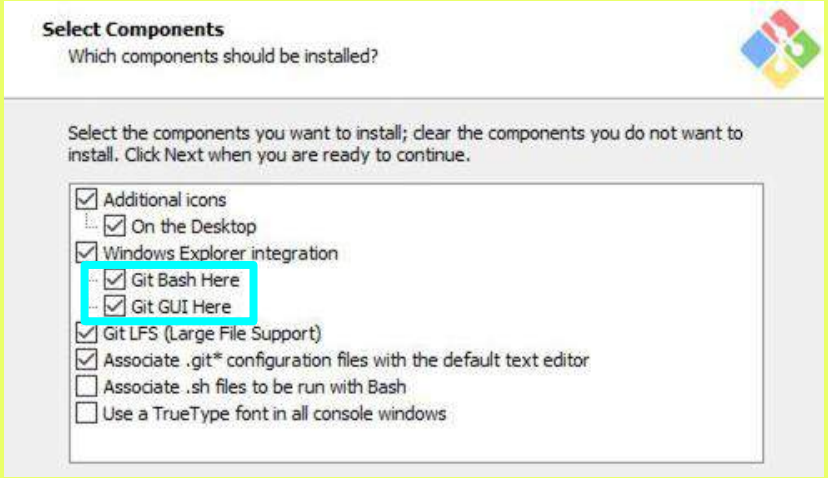
Ahora debes ejecutar el archivo descargado, y elegir la carpeta donde ubicar los archivos de Git. Si tienes dudas, te recomendamos dejar la opción que viene predeterminada.




CODERHOUSE

# Instalación de GIT (WINDOWS)

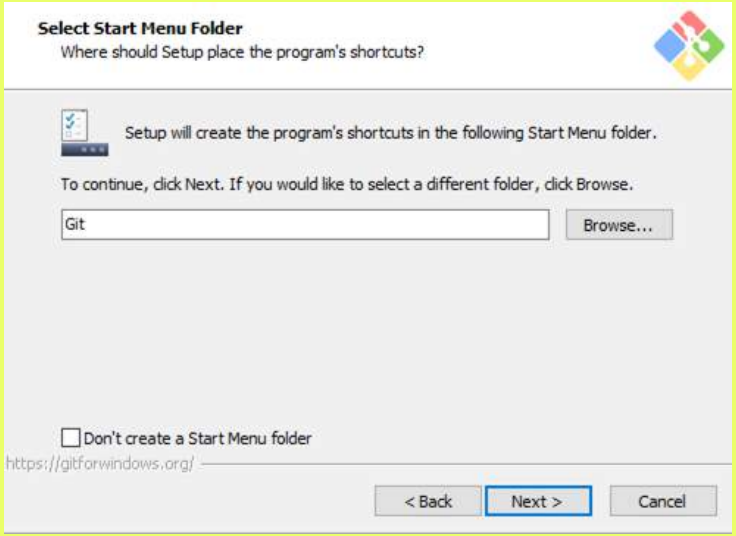
Asegúrate de tener seleccionada git bash, que es la herramienta principal con la que trabajaremos. Con esto se terminará la instalación.




CODERHOUSE

# Instalación de GIT (WINDOWS)

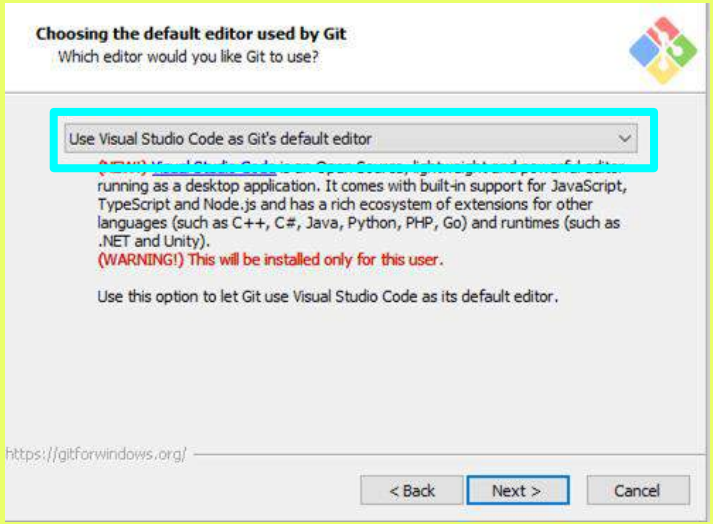
Continúa haciendo clic en “next”.




CODERHOUSE

# Instalación de GIT (WINDOWS)

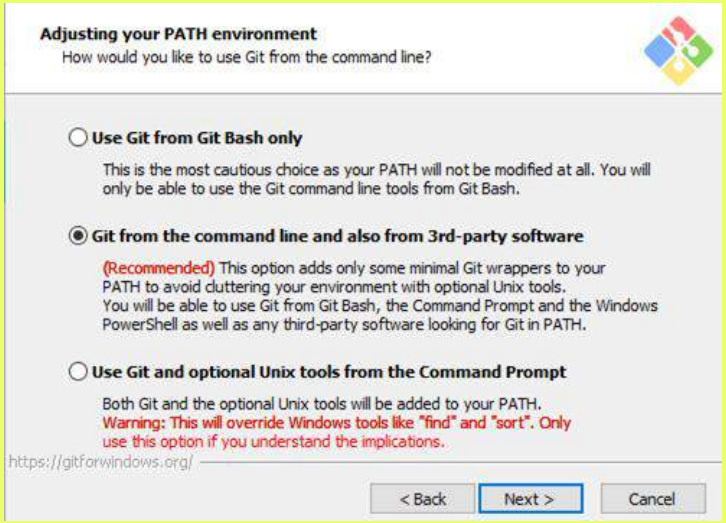
En esta pantalla, elige de la lista “use visual studio code as Git’s default editor”




CODERHOUSE

# Instalación de GIT (WINDOWS)

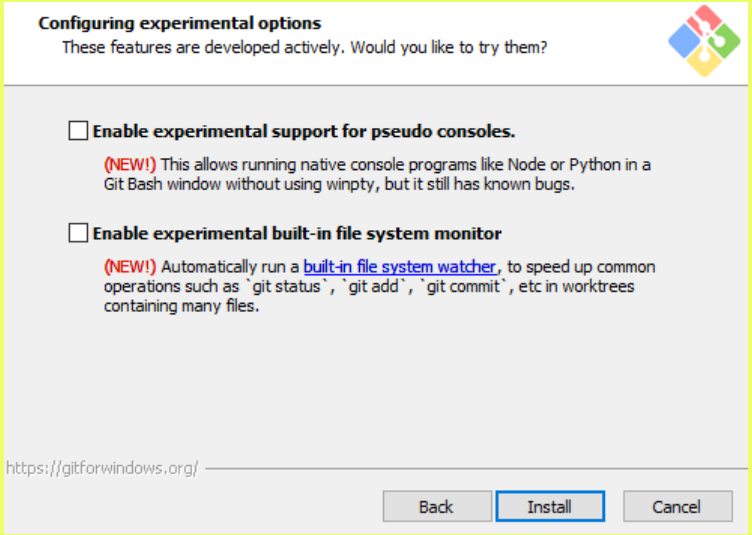
Continúa haciendo click en “next” hasta que se completen todas las opciones, dejando las configuraciones por defecto.




CODERHOUSE

# Instalación de GIT (WINDOWS)

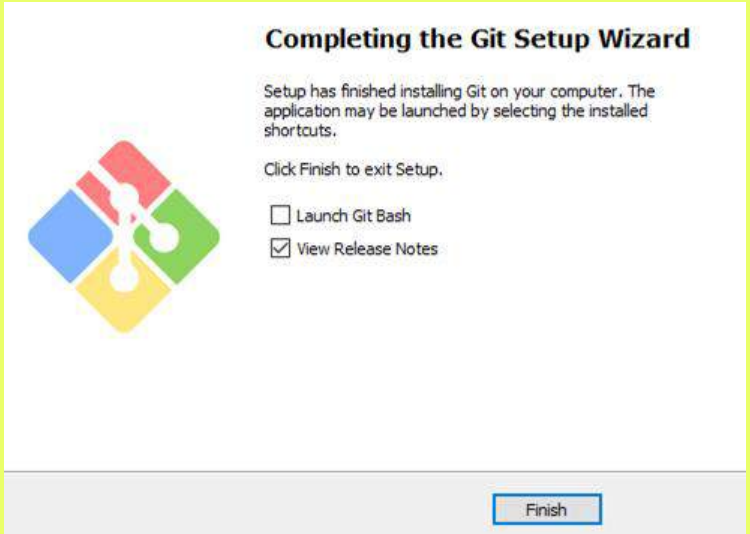
Finalmente, haz click en el botón “install”.




CODERHOUSE

# Instalación de GIT (WINDOWS)

Al finalizar el trabajo del instalador, se mostrará esta pantalla.
¡Felicitaciones! Ya tienes disponible Git en tu computadora.

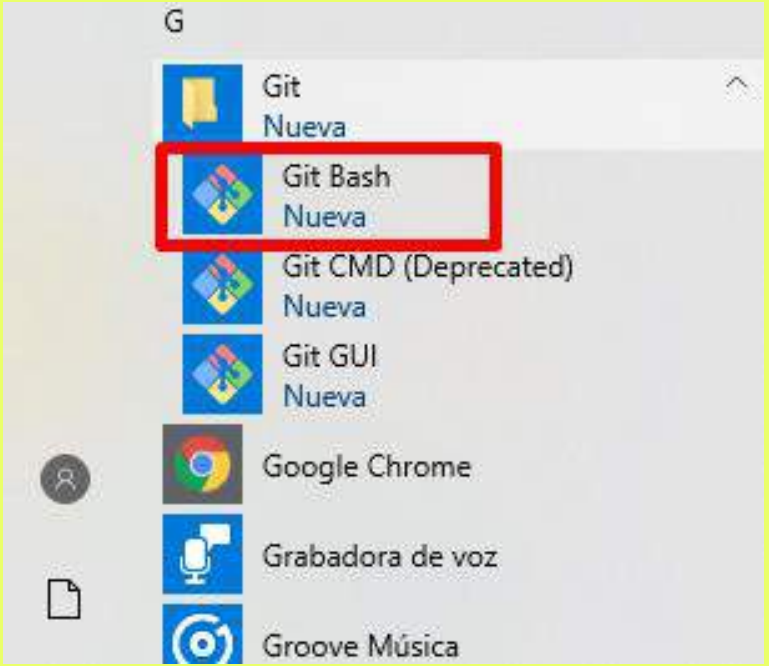



CODERHOUSE

# Instalación de GIT (WINDOWS)

Ahora tendrás disponible Git Bash desde tu lista de programas. Aquí es donde trabajaremos con Git.

Alternativa: Git GUI, para tener una interfaz más amigable.




CODERHOUSE

# Instalación y configuración de GIT en Mac

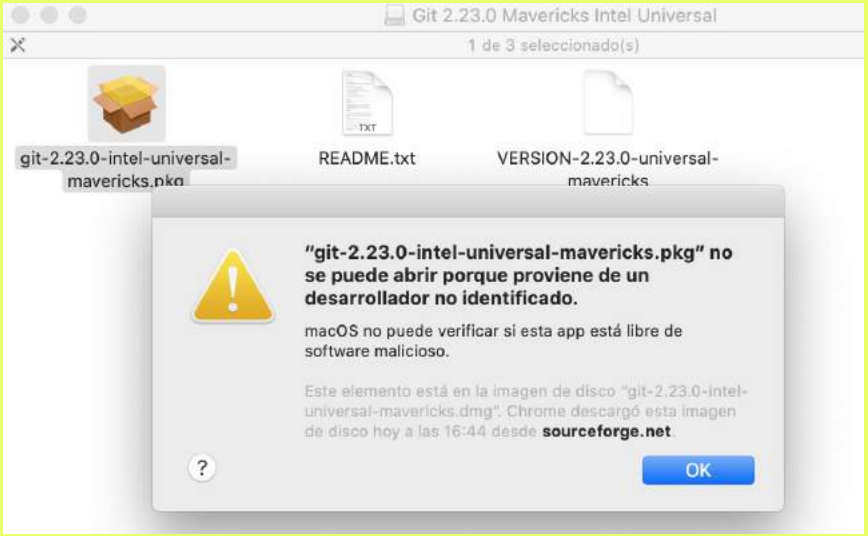
CODERHOUSE

# Instalación de GIT (MAC)

El primer paso es dirigirse a la página de SourceForge, donde se aloja el instalador de Git para Mac. Damos click en download y aguardamos a que finalice la desarga.

# Instalación de GIT (MAC)

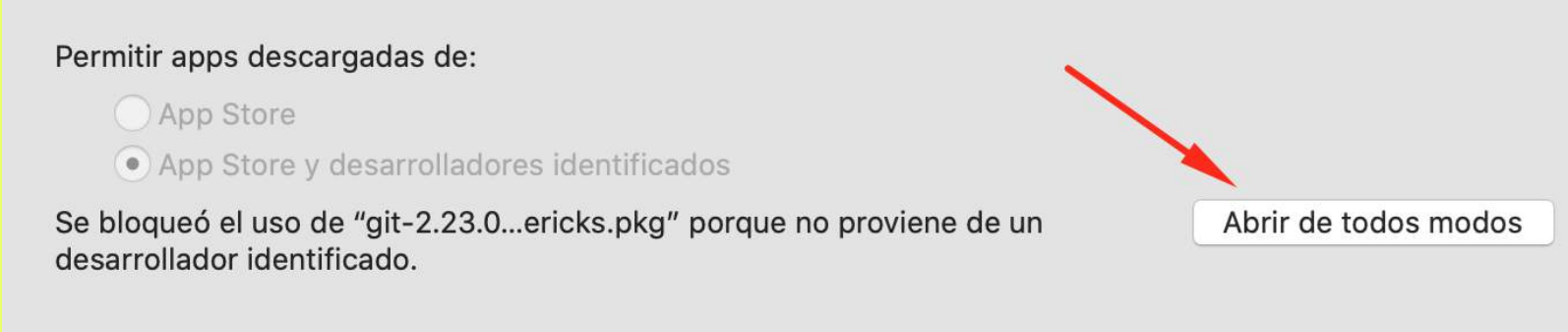
Posiblemente te salga este cartel de seguridad al intentar ejecutar el archivo descargado.




CODERHOUSE

# Instalación de GIT (MAC)

Ve a “preferencias de sistema” → “seguridad” y encontrarás lo siguiente. Haz clic en “abrir de todos modos”




CODERHOUSE

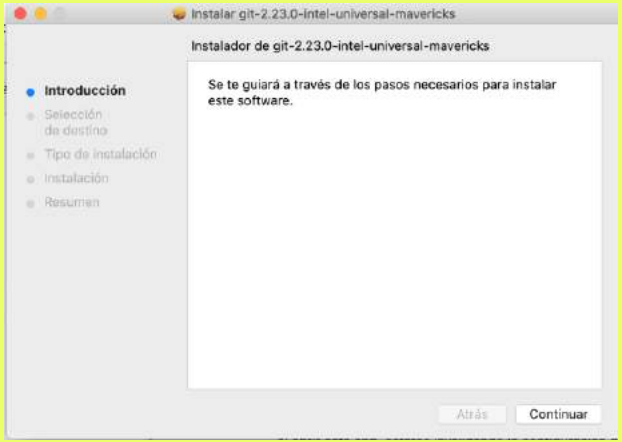
# Instalación de GIT (MAC)

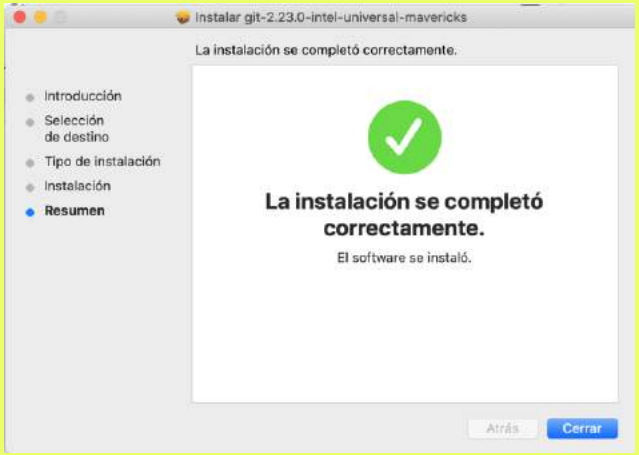
Un cartel más, y clic en “abrir”.


CODERHOUSE

# Instalación de GIT (MAC)

Una vez abierto el archivo instalador, haz clic en continuar hasta que diga “la instalación se completó”.
¡Felicitaciones! Ya tienes disponible Git en tu computadora.




CODERHOUSE

# Break

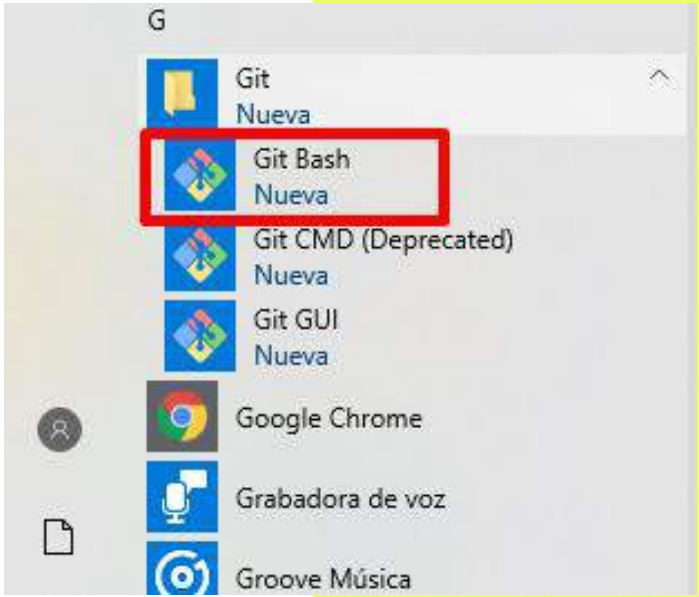
¡10 minutos y volvemos!

# Empecemos con GIT

CODERHOUSE

# Empecemos con GIT

Buscar en su menú el Git Bash, para abrir la terminal e iniciar con los comandos.




CODERHOUSE

# Verificando versión de GIT

Escribe git --version y presiona Enter.

```
john@MyShopSolutions: ~$ git --version
git version 2.38.1.windows.1
john@MyShopSolutions: ~$
```

# Configurando GIT por primera vez

Tu identidad

Lo primero que deberías hacer cuando instalas Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque las confirmaciones de cambios (commits) en Git usan esta información, y es introducida de manera inmutable en los commits que envías.

# Configurando GIT por primera vez

1

Elige un nombre de usuario que recuerdes fácil, y el email que en la próxima clase usarás en Github.

2

Establece el nombre con el comando: git config --global user.name "Nombre Apellido".

3

Establece el correo a usar con el comando. git config --global user.email johndoe@example.com

# Configurando GIT por primera vez

```
/* Paso 2*/
john@MyShopSolutions: ~$ git config --global user.name
"John Doe" /*Aquí tu nombre*/
/* Paso 3*/
john@MyShopSolutions:~$ git config --global user.email
johndoe@example.com /*Aquí tu email*/
```

# Comprobando tu configuración

Vamos a comprobar si guardamos bien el usuario usando el comando: git config --list

```
john@MyShopSolutions: ~$ git config --list
/* Se puede ver el usuario, el email y otros parámetros que
dependerán de cada sistema operativo */
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
/*Si la pantalla queda con un END al final, presiona q para
salir de esa lista y volver a escribir comandos*/
```

# Comprobando tu configuración

Puedes también comprobar qué valor tiene la clave nombre en Git ejecutando:
git config user.name

```
john@MyShopSolutions: ~$ git config user.name
John Doe
```

Puedes consultar de la misma manera user.email

# Obteniendo ayuda

Si alguna vez necesitas ayuda usando Git, hay varias formas de ver la página web del manual (manpage) para cualquier comando de Git (recuerda reemplazar <comando> por el comando para el cual requieres ayuda:

```
/*Ambos comandos disparan la ayuda de Git*/
john@MyShopSolutions: ~$ git help <comando>
john@MyShopSolutions: ~$ git <comando>--help

/*Ejemplo*/
john@MyShopSolutions: ~$ git help config
```

# Resumen

Deberías tener un conocimiento básico de qué es Git.
También deberías tener funcionando en tu sistema una versión de Git configurada con tu identidad.

Es el momento de aprender algunos fundamentos de Git.

# Creando repositorios

# ¿Qué es un repositorio?

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

- ✓ Será “la carpeta” o espacio donde guardarás tu proyecto, para más adelante compartirlo con el equipo a través de un repositorio en la nube (en internet, por ejemplo en GitHub).

# GIT INIT

Este comando se usa para crear un nuevo repositorio en Git. Nos crea un repositorio de manera local y lo hará en la carpeta donde estamos posicionados. También se le puede pasar [nombre_de_la_carpeta] y creará una con ese nombre.

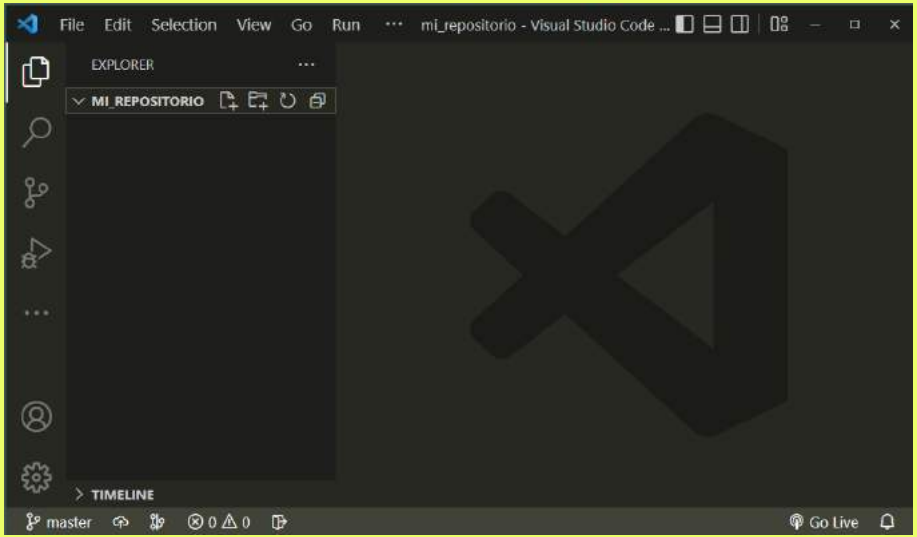
# GIT INIT

```
/* Paso 1: Me ubico en la carpeta donde quiero crear mi proyecto, usando el comando
de consola cd y la ruta relativa desde donde estoy y hacia donde quiero ir*/
john@MyShopSolutions :~$ cd Documents/Proyectos_Coder/
/* Paso 2: Ya dentro de la carpeta inicio el proyecto con el nombre que le asigne a
mi repositorio*/
john@MyShopSolutions :~/Documents/Proyectos_Coder/$ git init mi_repositorio
/* Arrojará el siguiente mensaje */
Initialized empty Git repository in
/home/usuario/Documents/Proyectos_Coder/mi_repositorio/.git/
/* Paso 3: Comprobamos que el repositorio se creó usando el comando dir para listar
el contenido de /Documents/Proyectos Coder*/
john@MyShopSolutions :~/Documents/Proyectos_Coder/$ dir
mi_repositorio
/* Paso 4: Me ubico en mi repositorio con el comando cd */
john@MyShopSolutions :~/Documents/Proyectos_Coder/$ cd mi_repositorio
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$
```

# GIT STATUS

Ya hemos visto cómo inicializar un repositorio localmente utilizando git init. Ahora nos toca crear los archivos que vamos a usar en este repositorio.

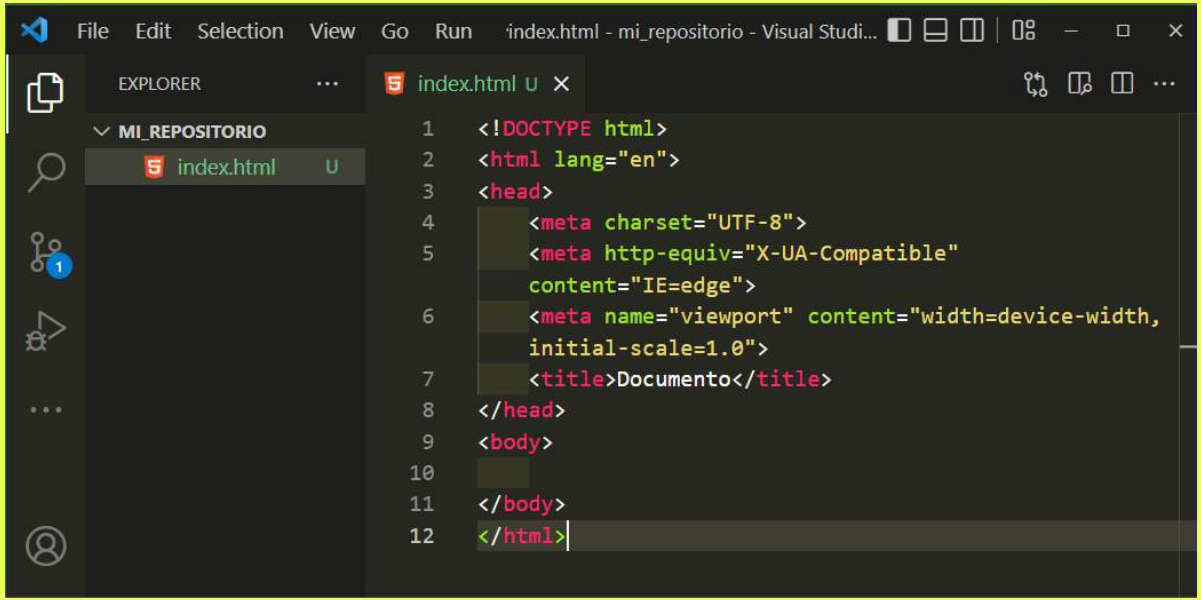
Trabajaremos con Visual Studio Code.
Lo primero será abrir el repositorio que acabamos de crear




CODERHOUSE

# GIT STATUS

Luego creamos un archivo index.html que se guardará en el repositorio




CODERHOUSE

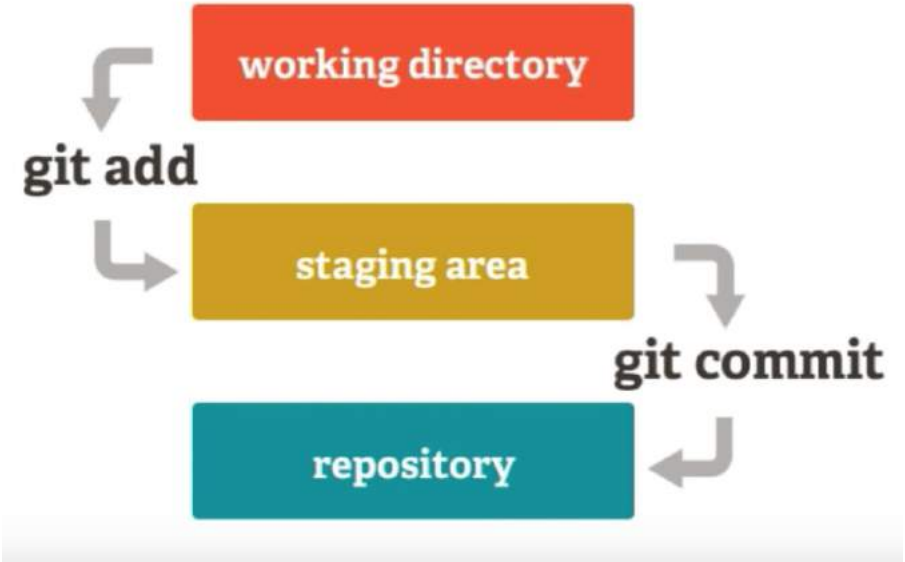
# GIT STATUS

Vamos a la terminal de VSC y con el comando git status chequeamos el estado de nuestro repositorio

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.html

nothing added to commit but untracked files present (use "git add" to track)
```

# ¿Recuerdan los 3 estados?

Estamos aquí con el index.html creado.



# GIT IGNORE

Ahora se necesita agregar el o los archivos al Staging Area. Pero, en ocasiones, puede haber ciertos archivos en el proyecto que no deben ser agregados (archivos de configuración, archivos del sistema operativo Mac, etc). Para lograrlo, vamos a usar ***.gitignore***

El archivo .gitignore, es un archivo de texto que le dice a Git qué archivos o carpetas ignorar en un proyecto. El archivo .gitignore se coloca en el directorio raíz del proyecto.
El procedimiento es simple: crea un archivo de texto y asígnale el nombre ".gitignore" (recuerda incluir el . al principio). Luego, edita este archivo según lo requiera tu proyecto. Cada nueva línea debe incluir un archivo o carpeta adicional que quieras que Git lo ignore.

# GIT IGNORE

Las entradas de este archivo también pueden seguir un patrón coincidente:

* se utiliza como una coincidencia comodín.
/ se usa para ignorar las rutas relativas al archivo .gitignore.
# es usado para agregar comentarios

Ejemplos de elementos que se pueden agregar en .gitignore

```
# Ignora archivos del sistema Mac
.DS_store

# Ignora la carpeta img_cache
img_cache

# Ignora todos los archivos de texto
*.txt

# Ignora el archivo background.jpg
background.jpg
```

# GIT ADD

Ahora se necesita agregar el o los archivos al Staging Area.

En nuestro caso, para el index.html vamos a usar el comando git add + el nombre del archivo, lo cual permite adherir el archivo para subirlo luego al repositorio. También se puede usar ***git add .*** que adhiere todos los archivos nuevos.

Para verificar si funcionó, nuevamente utilizamos git status.

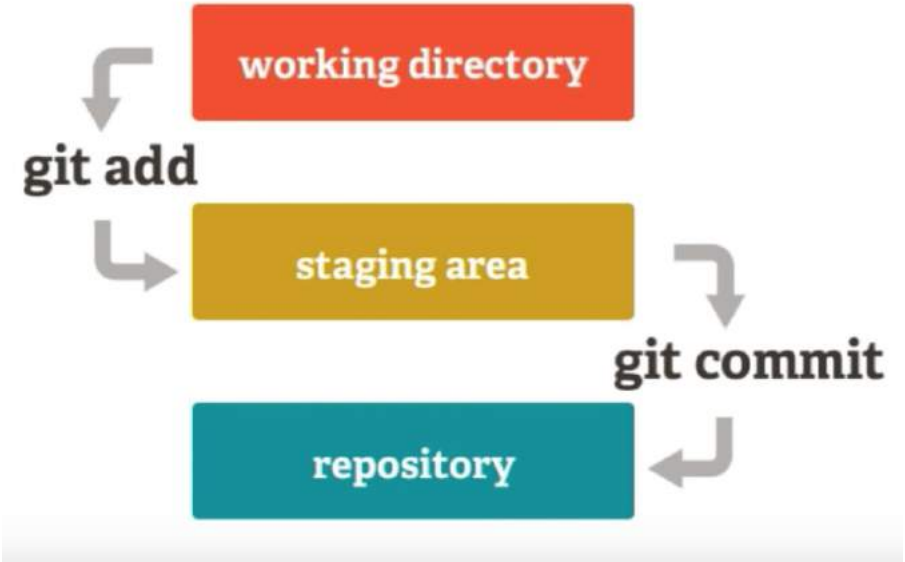
# GIT ADD

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git add index.html
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html
```

CODERHOUSE

# ¿Recuerdan los 3 estados?

Estamos aquí con el index.html adherido.



# GIT COMMIT

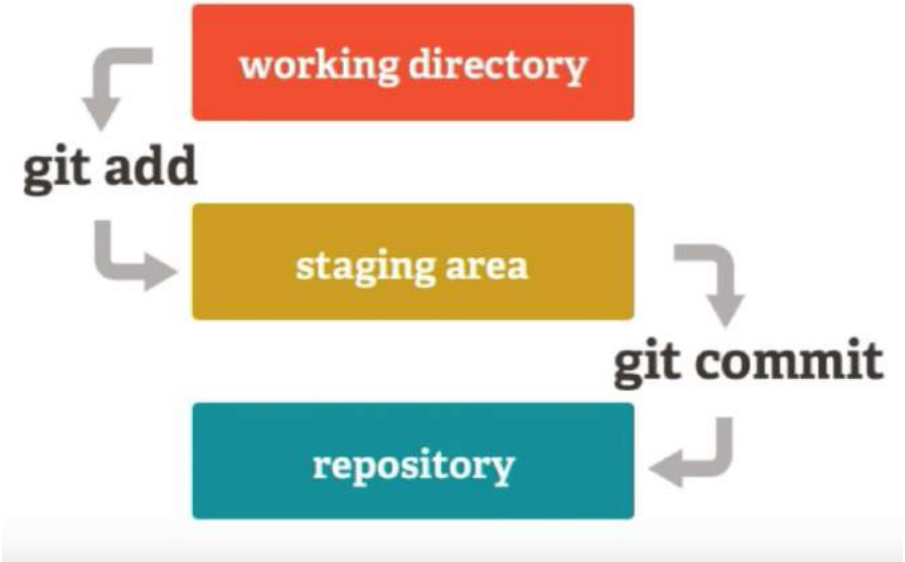
Una vez que nuestros archivos están en el Staging Area debemos pasarlos a nuestro repositorio local y para eso debemos usar el git commit, que es el comando que nos va a permitir comprometer nuestros archivos.

Es decir, que lo subirá al repositorio que se ha creado. El comando es el siguiente:
git commit -m “Comentario de qué se trata el commit que se está realizando”

# GIT COMMIT

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git
commit -m "Primer archivo del repositorio"
/* Esta sería el resultado del comando, el número siempre será
distinto */
[master (root-commit) 1734915] Primer archivo del repositorio
 1 file changed, 12 insertions(+)
 create mode 100644 index.html
```

# ¿Recuerdan los 3 estados?




Estamos aquí con el index.html comprometido para el repositorio

# GIT LOG

```
/* Con git log podemos ver los logs (historial) de lo que ha pasado
en el repositorio */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$
git log
commit 96e7e598115aeb3cb7901d3b311b29cc2ab7f392 (HEAD -> master)
Author: John Doe <johndoe@example.com>
Date:   Wed Nov 16 17:10:20 2022 -0300

    Primer archivo del repositorio
```

La documentación de *git log* es extensa, y puedes revisarla en su totalidad desde aquí: Git-scm

# Ramas

CoderTips

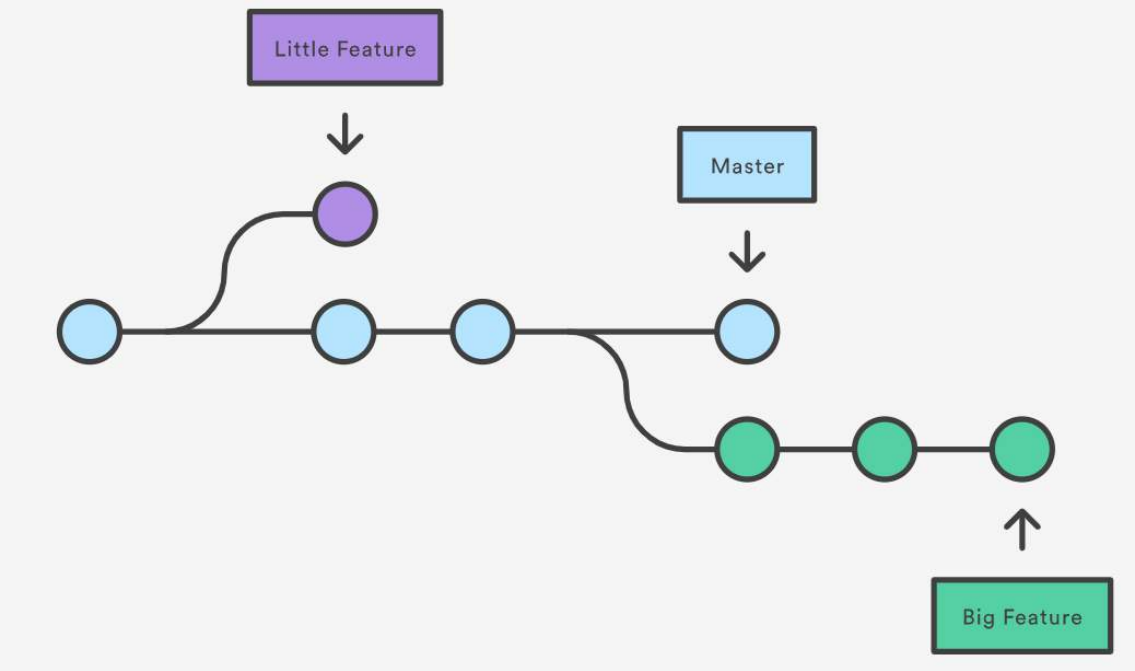
# Ramas

Cuando quieres añadir una nueva función o solucionar un error (sin importar su tamaño), generas una nueva rama para alojar estos cambios.

Esto te da la oportunidad de organizarte mejor con los cambios o correcciones experimentales.

Podemos crear una rama escribiendo
“git branch mi-rama”.

# Ramas




CODERHOUSE

# GIT BRANCH: CREANDO RAMAS

```
/* Paso 1: Verifico en cuál rama estoy */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git branch
*master
/* Paso 2. Creo la rama que voy a usar para el cambio */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$  git branch
mi_rama
/* Paso 3: Verifico que se creó la rama */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git branch -l
*master
mi_rama
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$
```

# GIT CHECKOUT: MOVERNOS ENTRE RAMAS

```
/* Para moverme a la rama que cree uso el comando de git checkout */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git
checkout mi_rama
Switched to branch 'mi_rama'
/* Verifico nuevamente que me movi de rama */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git
branch -l
master
*mi_rama
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$
```

# GIT BRANCH –D: BORRANDO RAMAS

```
/* Paso 1: Me muevo a la rama principal “master” */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git checkout
master
/* Paso 2: Verificar que se está en la rama de master */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git branch
*master
mi_rama
/* Paso 3: Procedo a borrar la rama que ya no voy a usar */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git branch -D
mi_rama
Deleted branch mi_rama (was 96e7e59)
/* Paso 4: Verificar que se borró la rama*/
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git branch
*master
```

# GIT CHECKOUT: LISTAR COMMITS

Así como nos movemos entre ramas, nos podemos mover entre commits.

Recuerden que al hacer cambios, adherirlos y comitearlos, se crea un historial de dichos cambios, los logs.

La posibilidad de volver a un commit en específico es una ventaja de los controladores de versiones, que permiten volver a un estado anterior si se presenta un problema, error o cambio inesperado

# GIT CHECKOUT: LISTAR COMMITS

## Vamos a crear nuevamente una rama...

1. Crea una rama con git branch nueva_rama
2. Cambia de rama con git checkout nueva_rama
3. Verifica que cambiaste de rama con git branch -l
4. Agrega al index.html un texto nuevo.

# GIT CHECKOUT: LISTAR COMMITS

## Vamos a crear nuevamente una rama...

5

Verifica que hubo un cambio en el index.html con git status

6

Adhiere el cambio con Git Add.

7

Comitea el cambio con git commit -m “Agregando texto al html”

8

Agrega un título al index.html y repite los pasos para poder comitear el cambio.

# GIT CHECKOUT: LISTAR COMMITS

```
/* Para ver los commits realizados, los listamos con el
comando git log --oneline para verlos en una sola línea*/
john@MyShopSolutions
:~/Documents/Proyectos_Coder/mi_repositorio$ git log --oneline
/* Se listan todos los cambios que se han realizado sobre el
index.html */
fc59b88 (HEAD -> nueva_rama) Ahora agregamos un título
6bcff19 Agregar un texto al index.html
41e6121 (master) Primer archivo del repositorio
john@MyShopSolutions
:~/Documents/Proyectos_Coder/mi_repositorio$
```

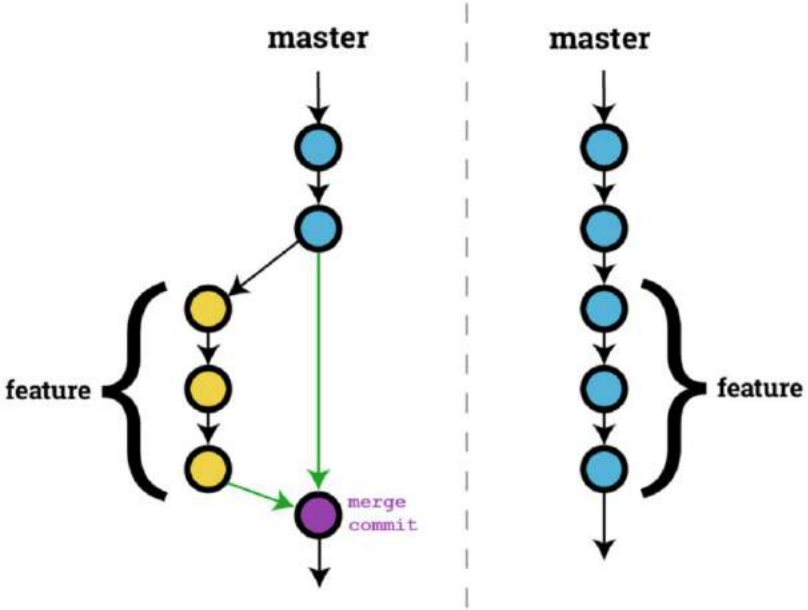
# GIT CHECKOUT: MOVERNOS A UN COMMIT

```
/* Supongamos que me equivoqué en agregar el título, quiero volver al punto
anterior del texto, busco el número de commit y me muevo hacia ese punto */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git checkout
6bcff19
Note: checking out 6bcff19.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 6bcff19... Agregar un texto al index.html
```

# GIT CHECKOUT: MOVERNOS A UN COMMIT

```
/* Si verifico donde estoy parado con git branch, se
puede observar que se está en el commit y el index.html
ha cambiado*/
john@MyShopSolutions
:~/Documents/Proyectos_Coder/mi_repositorio$ git branch
* (HEAD detached at 6bcff19)
  master
  nueva_rama
john@MyShopSolutions
:~/Documents/Proyectos_Coder/mi_repositorio$
```

# GIT MERGE




Una vez que tenemos una rama (o más), podemos experimentar características nuevas, y luego FUSIONARLAS con la rama MASTER.


CODERHOUSE

# GIT MERGE

```
/* Paso 1: Ubicarse en la rama master, que es a donde quiero fusionar los cambios usando el comando de git checkout master. */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$  git checkout master
/* Paso 2: Verificar que estoy en master con git branch. Se puede observar en el archivo de index.html que no tiene ni título ni texto. */
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$  git branch
*master
Nueva_rama
/* Paso 3: Realizar la fusión. Hacer el merge con el comando git merge nueva_rama*/
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$  git merge
nueva_rama
Updating 41e6121..fc59b88
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

# ¿Preguntas?

**#CoderTip:** Ingresa al siguiente link y revisa el material interactivo que preparamos sobre **Preguntas Frecuentes,** estamos seguros de que allí encontrarás algunas respuestas.

# Ejemplo en vivo

¡Vamos a practicar lo visto!

# GIT: resumen

- ✓ **git init**: indicarle que en ese directorio, donde ejecutamos este comando, será usado con GIT.
- ✓ **git add .**: agregar todos los archivos creados, modificados, eliminados al estado 2 (stage)
- ✓ **git commit -m "Mensaje"**: mensaje obligatorio para indicar que hemos cambiado por ejemplo, al estado 3.
- ✓ **Git log --online**: para conocer los códigos de los commits realizados.
- ✓ **Git checkout rama**: para cambiar de rama o ir a un commit específico (debemos conocer su código anteriormente) .
- ✓ **git merge rama**: debemos estar en MASTER para fusionar.
- ✓ **git branch rama**: creación de una rama (si queremos eliminar una rama ponemos git branch -D nombre-rama).

PARA RECORDAR

# ¡Con lo visto hoy, podemos manejar GIT de forma básica y tener nuestro repositorio local!

# Clientes gráficos para GIT

- ✓ Git-gui
- ✓ GitHub Desktop
- ✓ GitKraken
- ✓ SmartGit
- ✓ SourceTree


CODERHOUSE

# Segunda pre-entrega

En la clase que viene se presentará la segunda parte del Proyecto Final,  que nuclea temas vistos entre las clases 5 y 11.
Recuerda que tendrás 7 días para subirla en la plataforma.

CODERHOUSE

# ¿Preguntas?

# Resumen
# de la clase hoy

- ✓ Conocer Git.
- ✓ Aprender a usar la terminal.
- ✓ Usar los comandos básicos de Git.

# Opina y valora esta clase

CODERHOUSE

# Muchas gracias.

#DemocratizandoLaEducación

CODERHOUSE