

***RECUERDA PONER A GRABAR LA  
CLASE***





**Clase 09.** DESARROLLO WEB  
***GIT***



## ***OBJETIVOS DE LA CLASE***

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

# GLOSARIO:

## Clase 8

**Servidor:** cuando hablamos de servidores, nos referimos al ordenador que pone recursos a disposición a través de una red, o al programa que funciona en dicho ordenador.

En consecuencia, aparecen dos definiciones de servidor: hardware y software.

**Servidor hardware:** es una máquina física integrada en una red informática en la que, además del sistema operativo, funcionan uno o varios servidores basados en software.

**Servidor software:** es un programa que ofrece un servicio especial que otros programas, denominados clientes ("clients"), pueden usar a nivel local o a través de una red. El tipo de servicio depende del tipo de software del servidor.

**Seguridad:** es la acción o práctica de proteger sitios web del acceso, uso, modificación, destrucción o interrupción no autorizados.

**SVG:** es una imagen vectorial en formato XML, que funciona muy bien para los logotipos, iconos, texto e imágenes sencillas.

**SEO:** significa "Search Engine Optimization", es decir, Optimización de Motores de Búsqueda. El objetivo general del SEO es mejorar la posición de un sitio web, en los motores de búsqueda.

# ***GLOSARIO:***

## ***Clase 8***

**Metaetiquetas de descripción:** ofrece a Google y a otros buscadores un resumen del contenido de la misma.

**Ruta de exploración:** es una fila de enlaces internos, situada en la parte superior o inferior de una página, que permite a los visitantes volver rápidamente a una sección anterior, o a la página de inicio.

**Mapa de sitio:** es una página sencilla de un sitio web que muestra su estructura, y normalmente consiste en una lista jerárquica de las páginas que se incluyen en éste.

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 9

¡Para  
recordar!



# ***CRONOGRAMA DEL CURSO***

## Clase 8



### **Animaciones, transformaciones y transiciones**



PRÁCTICAS DE LO  
VISTO EN CLASE



APLICANDO GRIDS



FULL RESPONSIVE

## Clase 9



### **Git**



PRÁCTICAS DE LO  
VISTO EN CLASE

## Clase 10



### **GitHub**



PRÁCTICAS DE LO  
VISTO EN CLASE



CREAR REPOSITORIO DE  
GITHUB





# ***GUIÓN DE LA CLASE***

Accede al material complementario [aquí](#).



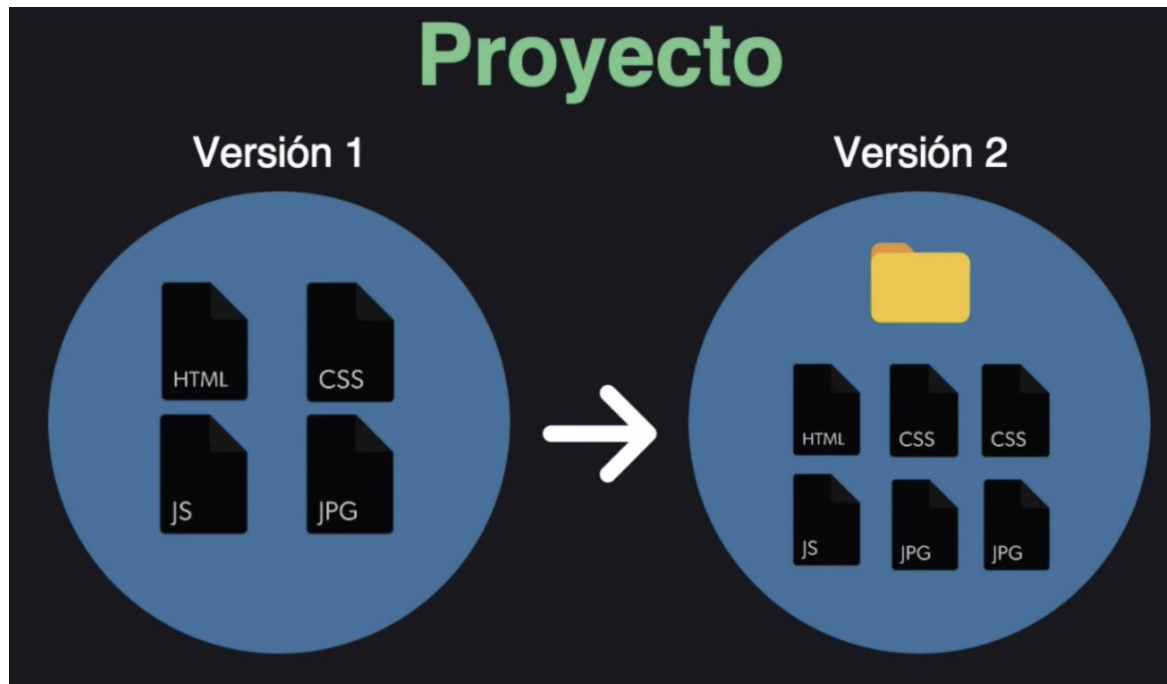
***GIT***

# ¿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 hacer modificaciones en nuestro proyecto. Este sistema registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.

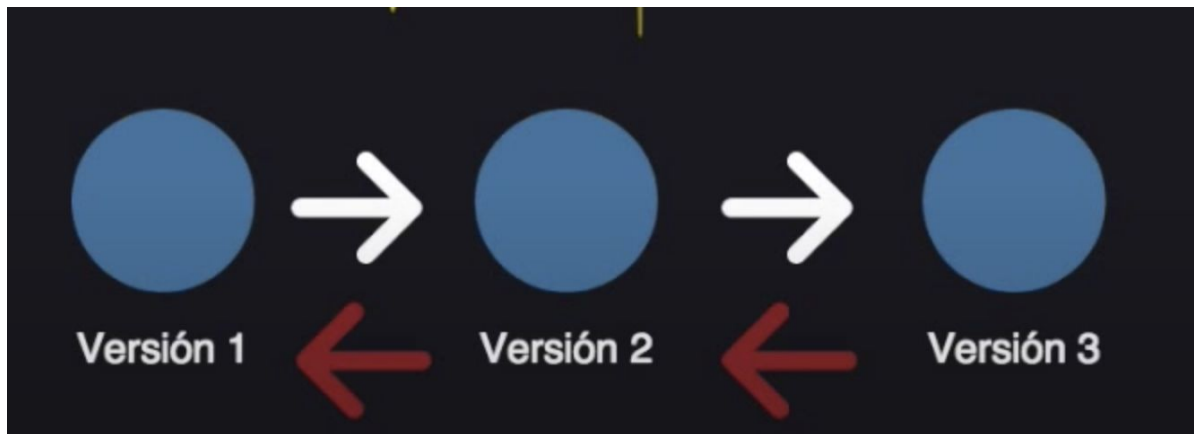


# ***GIT: ¿QUÉ ES?***



# ***GIT: ¿QUÉ ES?***

Con GIT, **podemos ir a versiones anteriores**, muy útil para errores, y para la organización.

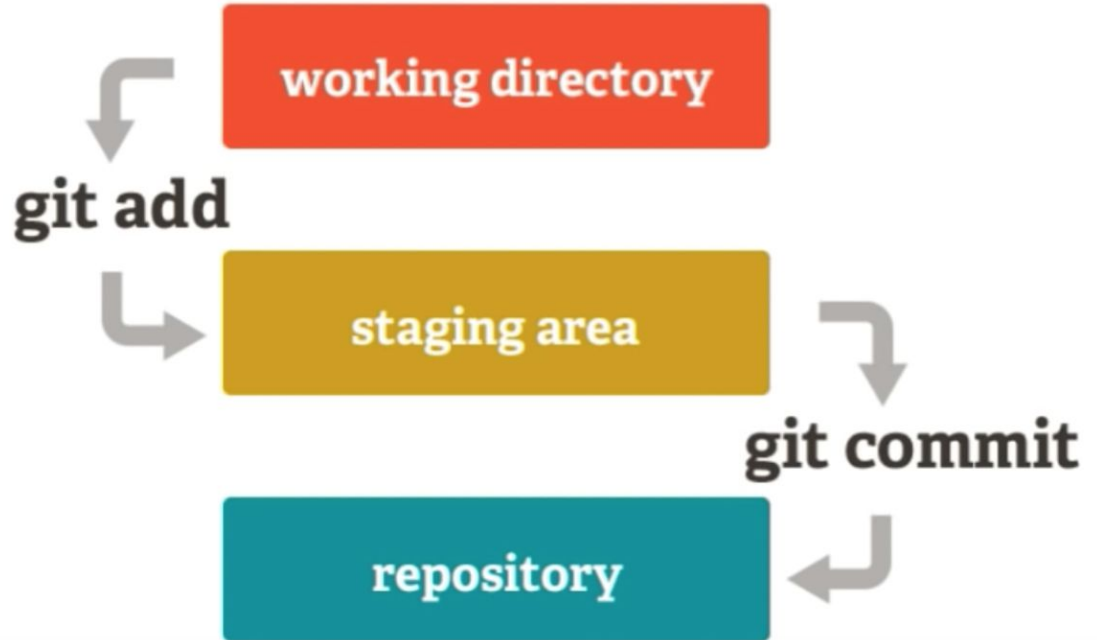


# ***GIT: LOS 3 ESTADOS***

**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 GITHUB?***



**git**



**github**  
SOCIAL CODING

# ***GITHUB***

👉 **Git** es uno de los sistemas de control de versiones más populares entre los desarrolladores.

👉 Parte de su popularidad se la debe a **GitHub, un excelente servicio de alojamiento de repositorios de software con este sistema.**

¡Lo veremos la próxima clase!



# ***INSTALACIÓN Y CONFIGURACIÓN DE GIT***

# ***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**.

# INSTALACIÓN DE GIT (WINDOWS)

👉 Ve a <https://git-scm.com/> y descarga el paquete de instalación “download (nro versión) for Windows” (o si tienes otro sistema operativo, dirá Mac o Linux).



The image shows a screenshot of the Git website's navigation menu and download section. The navigation menu is on the left, with four items: 'About' (gear icon), 'Documentation' (book icon), 'Downloads' (downward arrow icon), and 'Community' (speech bubble icon). The 'Downloads' section is highlighted, showing the latest source release '2.15.1' and a button to 'Download 2.15.1 for Windows'.

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

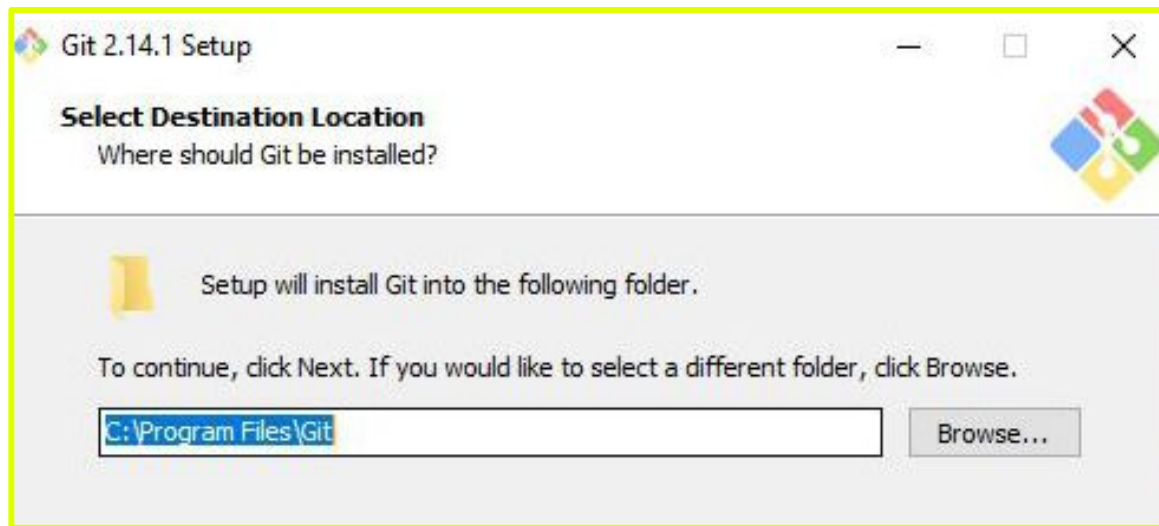
**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release  
**2.15.1**  
Release Notes (2017-11-28)  
Download 2.15.1 for Windows

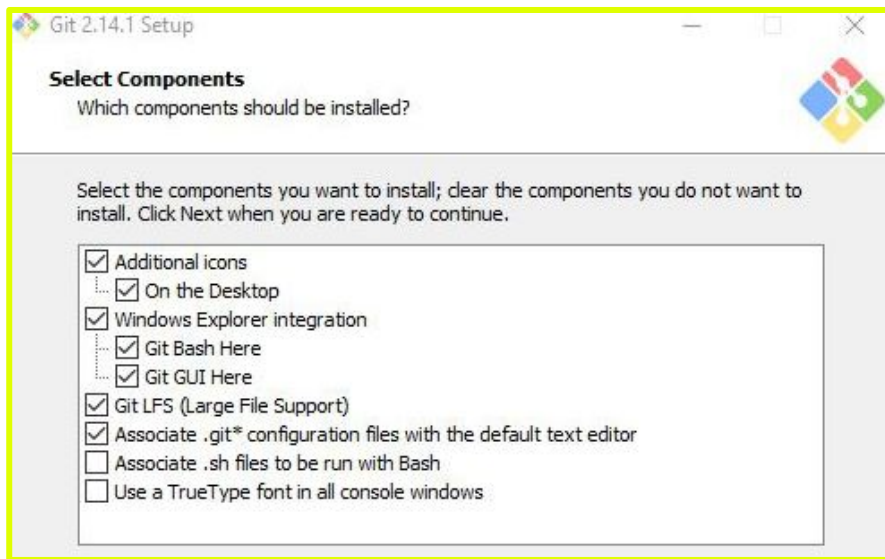
# ***INSTALACIÓN DE GIT (WINDOWS)***

👉 Ahora debes **ejecutar el archivo descargado**, y elegir la carpeta donde ubicar los archivos de Git



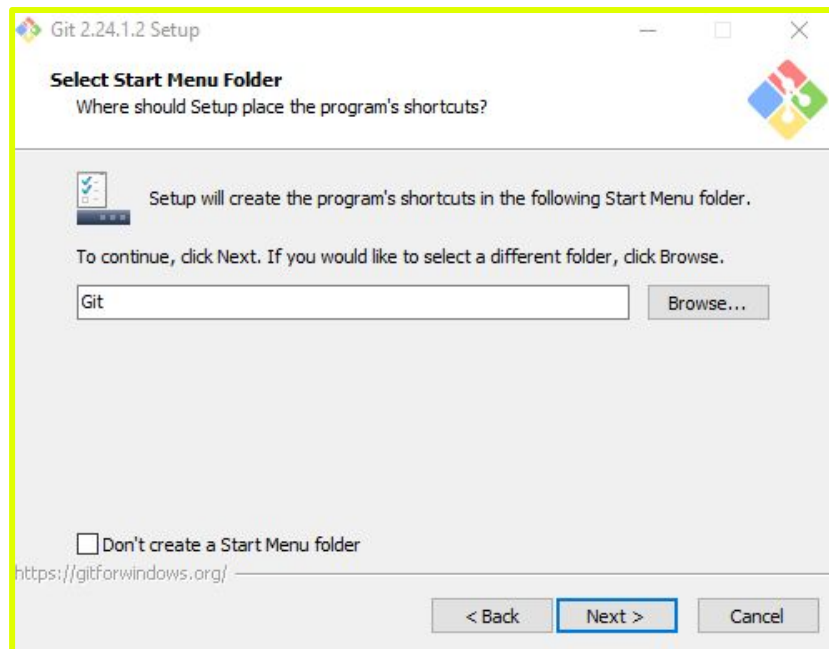
# INSTALACIÓN DE GIT (WINDOWS)

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



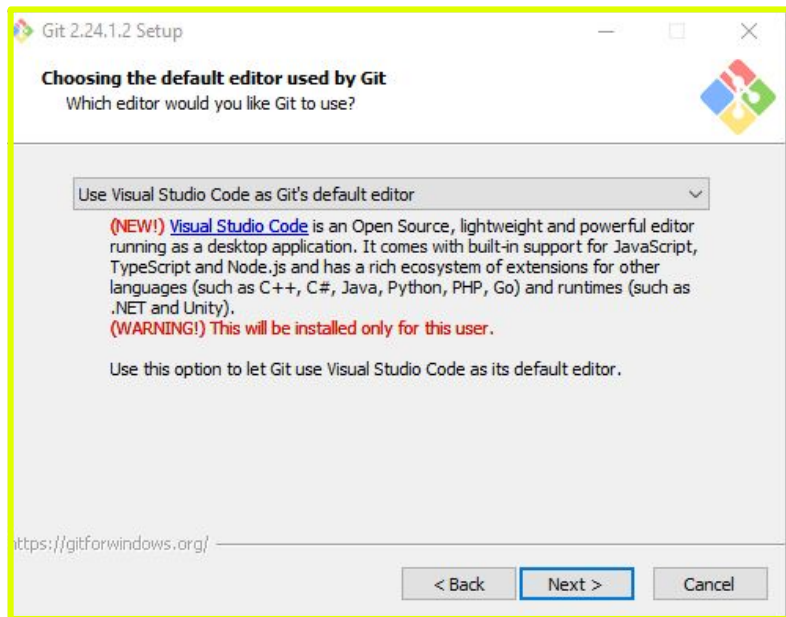
# ***INSTALACIÓN DE GIT (WINDOWS)***

👉 Continúa haciendo clic en “next”.



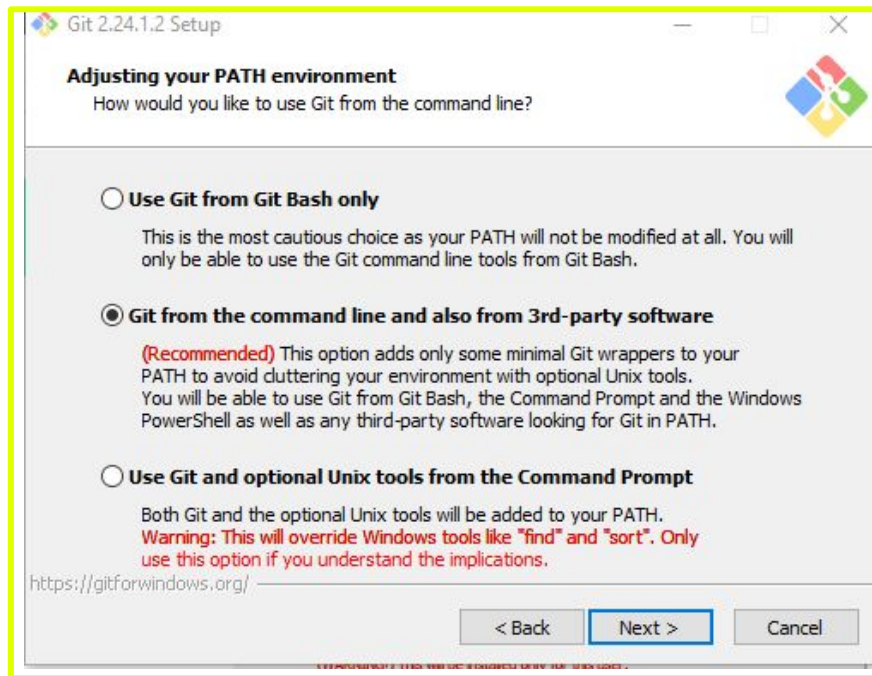
# INSTALACIÓN DE GIT (WINDOWS)

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



# INSTALACIÓN DE GIT (WINDOWS)

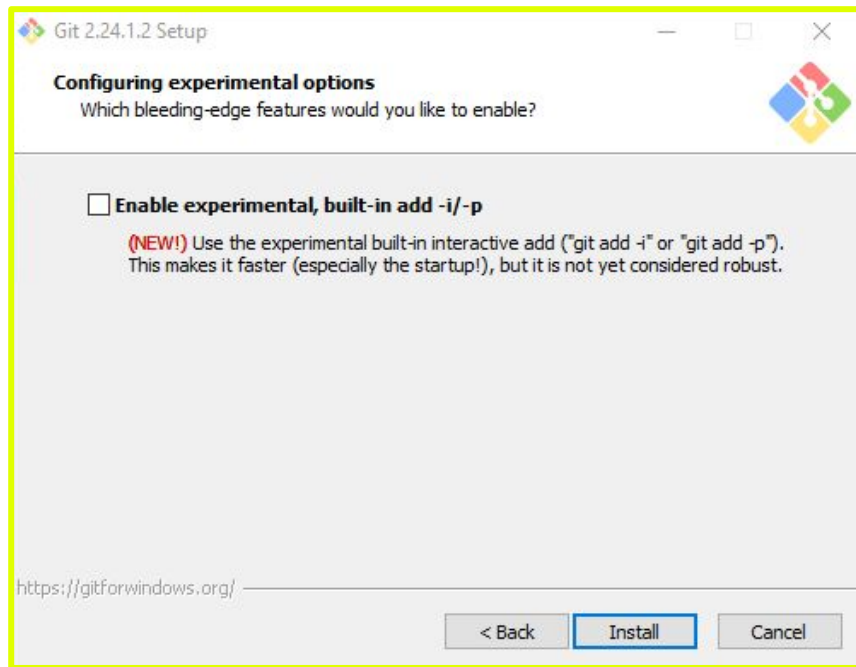
👉 Haz click en **“next”** hasta finalizar la instalación, dejando las configuraciones por defecto.





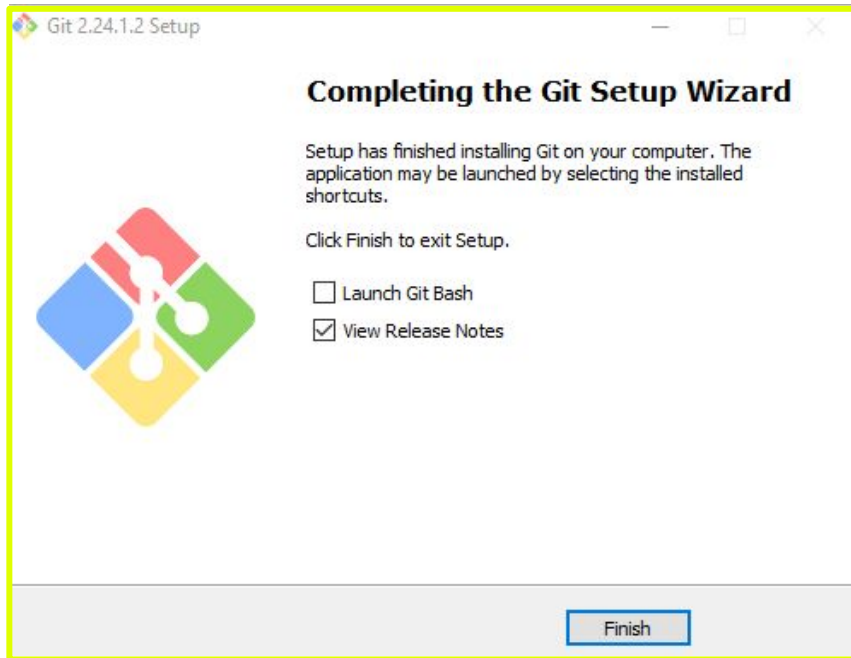
# ***INSTALACIÓN DE GIT (WINDOWS)***

👉 Finalmente aparece el botón “install”.



# ***INSTALACIÓN DE GIT (WINDOWS)***

👉 Instalación finalizada.

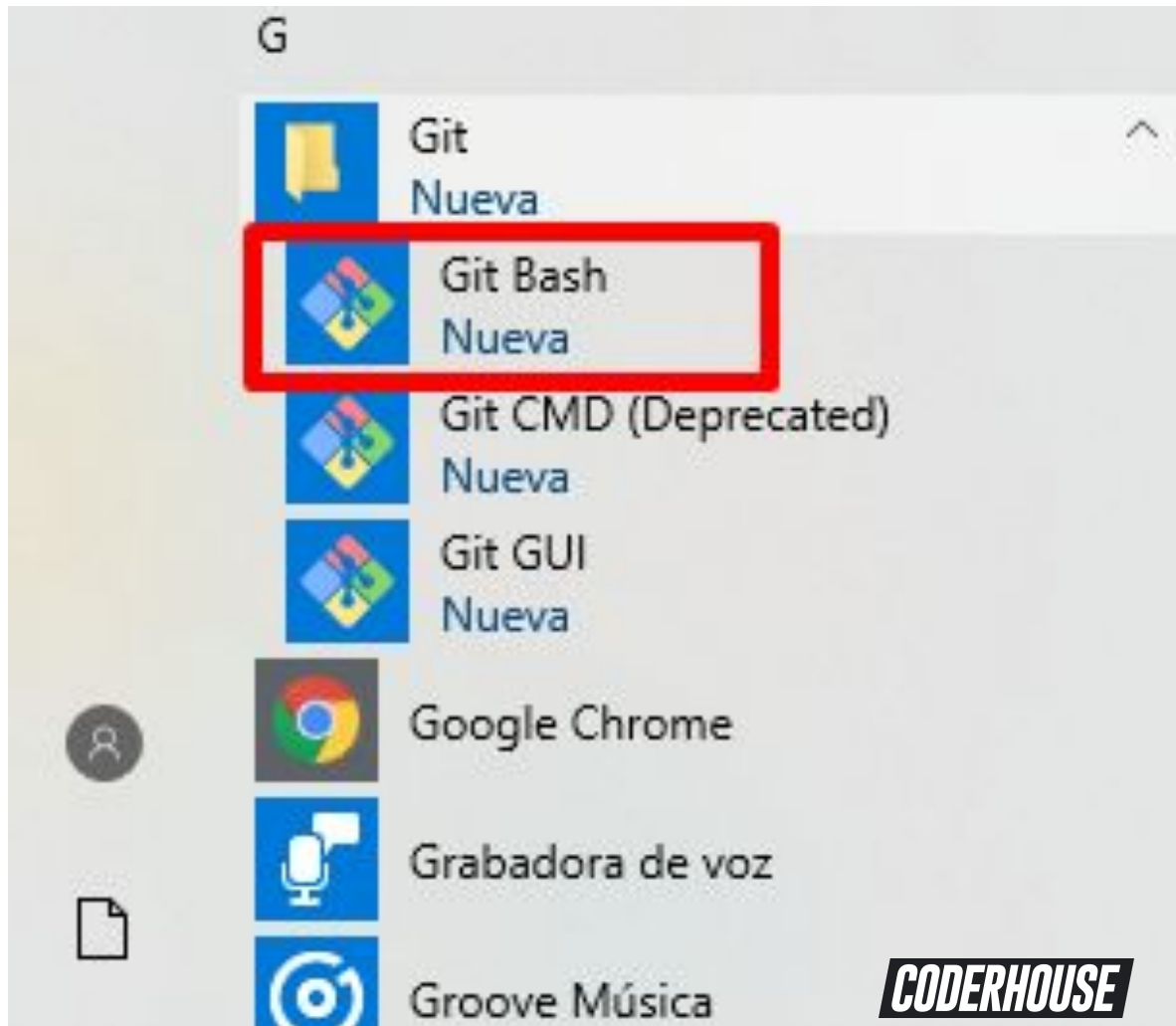


**CODERHOUSE**

# ***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.



# INSTALACIÓN DE GIT (MAC)

👉 Ve a <https://git-scm.com/> y descarga el paquete de instalación “download (nro versión) for Mac” (o si tienes otro sistema operativo, dirá Windows o Linux).



The screenshot shows the Git website layout. On the left, there are four circular icons with corresponding text: a gear for 'About', a book for 'Documentation', a download arrow for 'Downloads', and speech bubbles for 'Community'. On the right, a computer monitor displays the 'Latest source Release' section, highlighting version 2.24.1 and providing a download link for Mac.



### About

The advantages of Git compared to other source control systems.



### Documentation

Command reference pages, Pro Git book content, videos and other material.



### Downloads

GUI clients and binary releases for all major platforms.



### Community

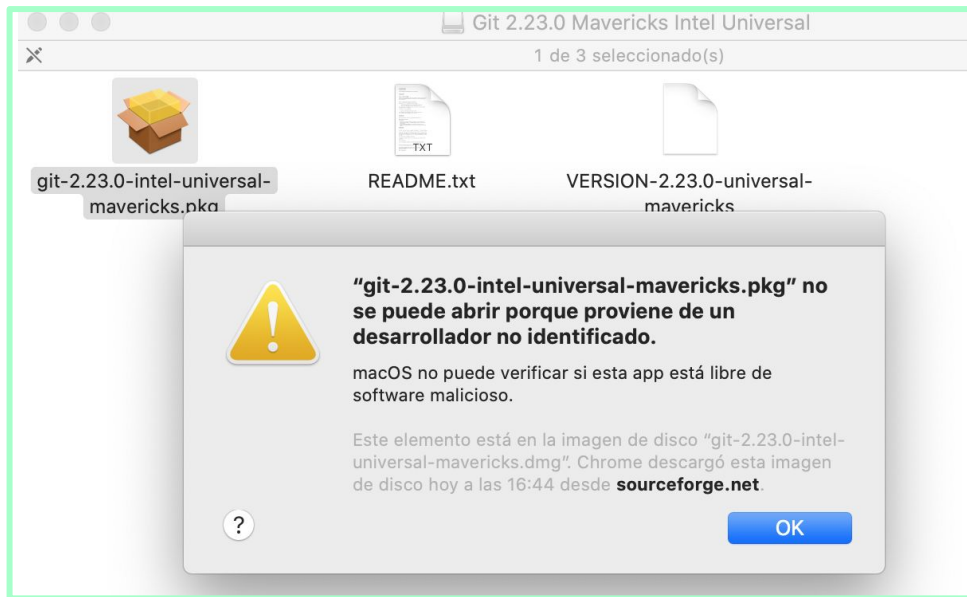
Get involved! Bug reporting, mailing list, chat, development and more.



Latest source Release  
**2.24.1**  
Release Notes (2019-12-06)  
Download 2.23.0 for Mac

# ***INSTALACIÓN DE GIT (MAC)***

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



# INSTALACIÓN DE GIT (MAC)

👉 Ve a “*preferencias de sistema*” → “*seguridad*” y encontrarán lo siguiente. Hacen clic en “***abrir de todos modos***”

Permitir apps descargadas de:

- ☐ App Store
- ☒ App Store y desarrolladores identificados

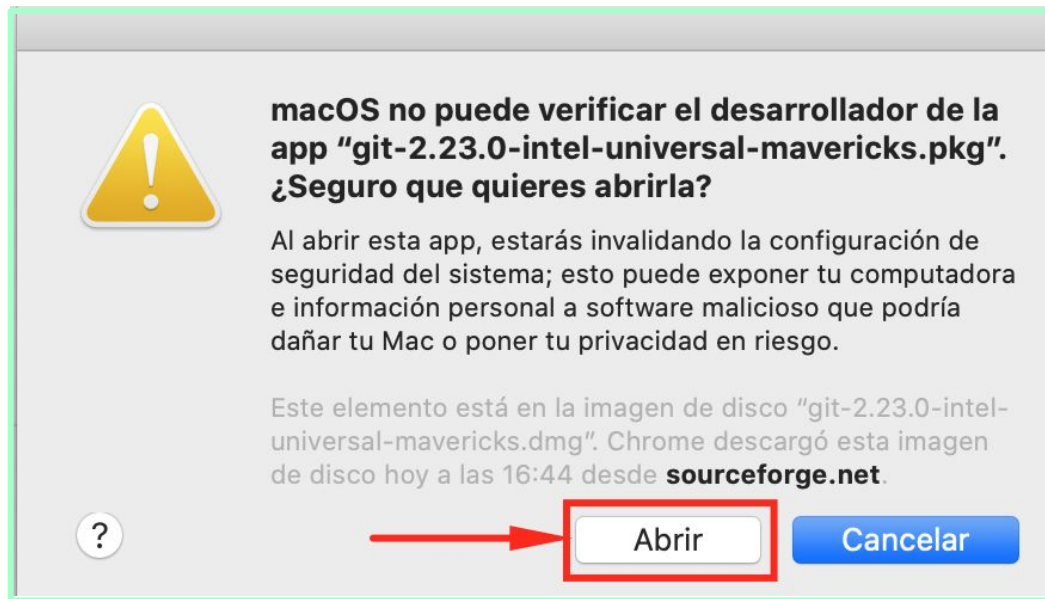
Se bloqueó el uso de “git-2.23.0...ericks.pkg” porque no proviene de un desarrollador identificado.



Abrir de todos modos

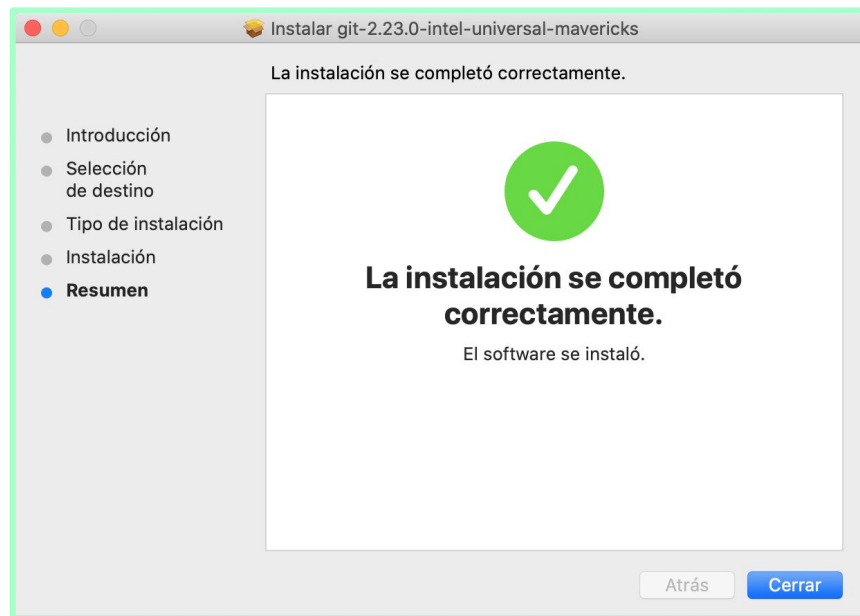
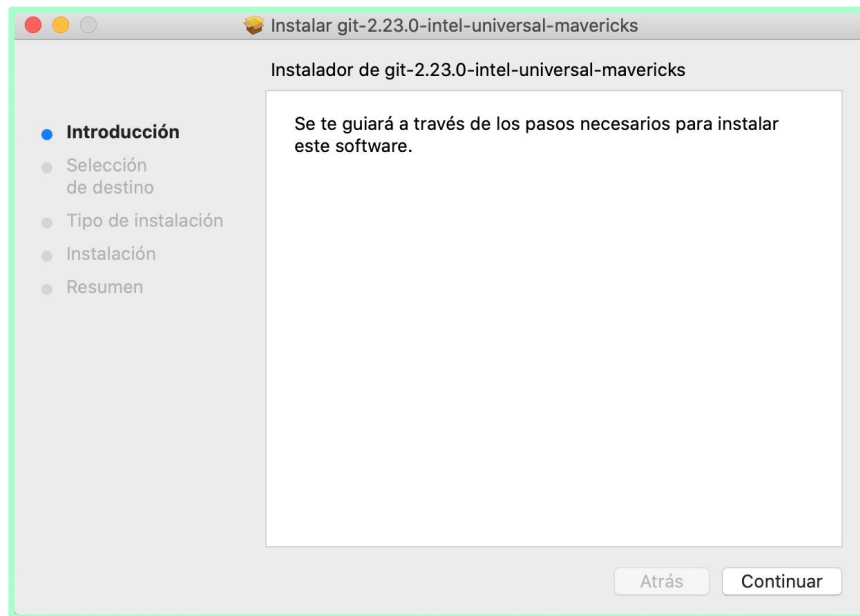
# INSTALACIÓN DE GIT (MAC)

👉 Un cartel más, y clic en “**abrir**”.



# ***INSTALACIÓN DE GIT (MAC)***

👉 Instalado, haz clic en continuar hasta que diga ***“la instalación se completó”***.

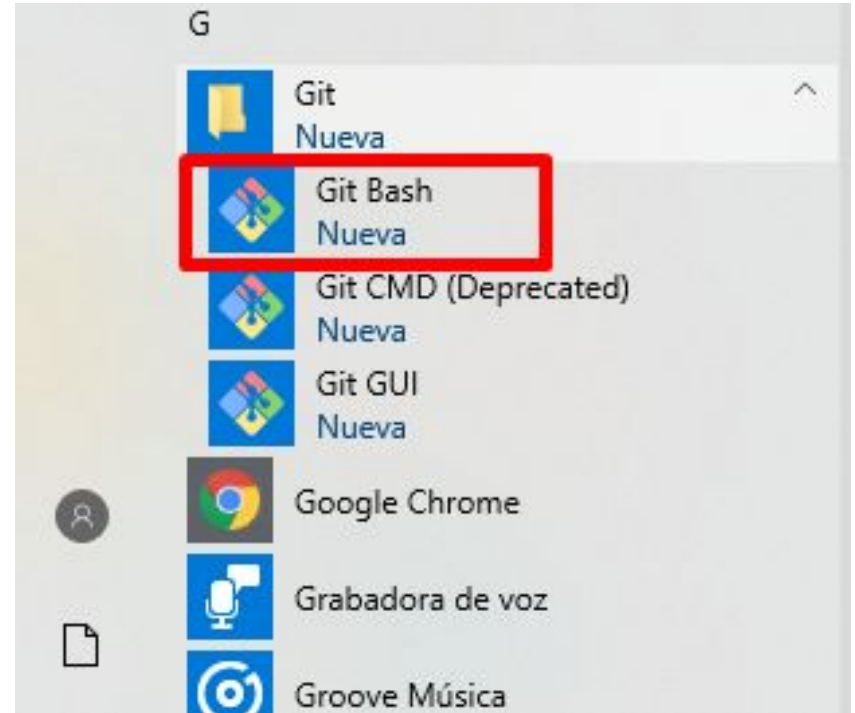




***EMPECEMOS CON GIT***

# EMPECEMOS CON GIT

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



# ***VERIFICANDO VERSIÓN DE GIT***



Escribe *git --version* y presiona Enter.

```
john@MyShopSolutions: ~$ git --version  
git version 2.17.1  
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"
```

```
/* Paso 3*/
```

```
john@MyShopSolutions:~$ git config --global user.email johndoe@example.com
```

# 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
```

# ***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 tres formas de ver la página del manual (manpage) para cualquier comando de Git:

```
/* Los tres comandos que disparan la ayuda de Git */
```

```
john@MyShopSolutions: ~$ git help config
```

```
john@MyShopSolutions: ~$ git config --help
```

```
john@MyShopSolutions: ~$ man git-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.**

**¿DUDAS?**



# ***COMANDOS BÁSICOS DE LA TERMINAL***

# ***¿CÓMO ABRIR LA TERMINAL?***

Para abrir la línea de comandos de Windows o símbolo del sistema, tan sólo tienes que ir a **Inicio > Ejecutar o Buscar > CMD.exe** y se abrirá una pequeña ventana que te recordará al antiguo MS-DOS, o a **Inicio->Git->Git Bash**.

Para abrir la terminal de Mac OS haz clic en el icono "**Finder**" situado en el Dock, luego selecciona "**Aplicaciones > Utilidades**", y finalmente dale doble clic al icono "**Terminal**".

# COMANDOS BÁSICOS

¡Para  
recordar!



👉 **/?:** si quieres saber más de un comando, añade **/?** para ver la ayuda relacionada. Te será muy útil para ver las muchas opciones de cada comando.

👉 **HELP:** te mostrará una lista de comandos disponibles.

👉 **DIR:** es el comando más conocido de DOS y sirve para ver el contenido de una carpeta (en MAC-OS usar LS).

👉 **CD:** sirve para entrar en una carpeta o salir de ella (CD...).

👉 **CLEAR:** limpia la consola.

# COMANDOS BÁSICOS

¡Para  
recordar!



👉 **MKDIR:** con este comando crearás una carpeta nueva. Con RMDIR podrás eliminarla.

👉 **MOVE** y **COPY:** son los comandos para mover y copiar archivos respectivamente. Deberás indicar el nombre del archivo con su ruta (si está en otra carpeta en la que te encuentras) y la ruta de destino.

👉 **RENAME:** sirve para renombrar un archivo o carpeta. Hay que indicar el nombre original y el definitivo.

# COMANDOS BÁSICOS

¡Para  
recordar!



👉 **DEL:** es el comando para eliminar un archivo. Recuerda que no irá a la Papelera, así que piensa muy bien antes de borrar algo. Y para eliminar carpeta usa el comando RD (en MAC-OS usar RM para archivos / para eliminar carpetas RM -RF).

👉 **EXIT:** cierra la ventana de la línea de comandos o símbolo del sistema.

👉 **COPY CON:** crear archivos (en MAC-OS usar TOUCH).

# ***CREANDO REPOSITARIOS***



# ¿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.

Por ejemplo...

# ***GIT INIT***



/\* Paso 1: Me ubico en la carpeta donde quiero crear mi proyecto \*/

```
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 \*/

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/$ dir  
mi_repositorio
```

/\* Paso 4: Me ubico en mi repositorio \*/

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/$ cd mi_repositorio
```

```
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$
```

# ***GIT STATUS***

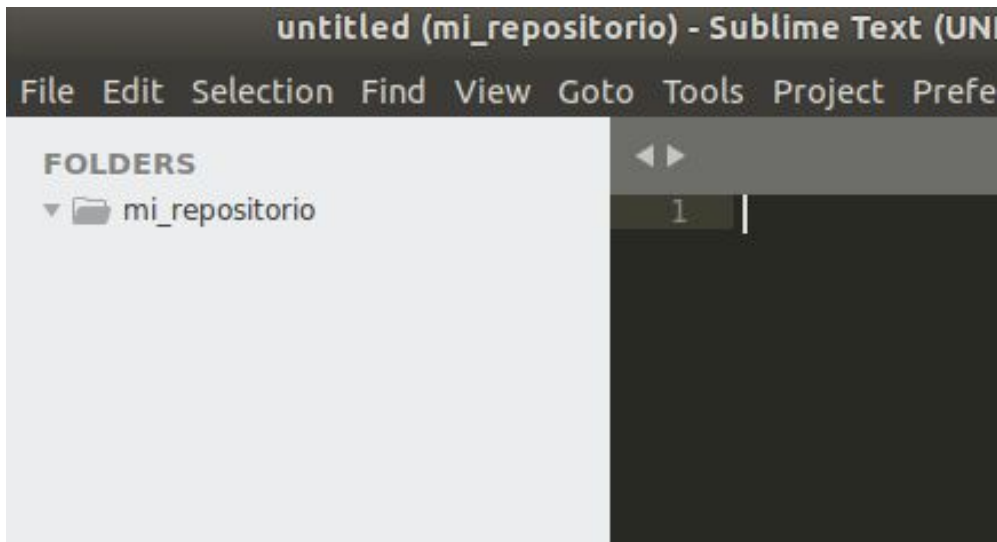


Ya hemos visto cómo inicializar un repositorio localmente utilizando **git init**. Ahora nos toca crear los archivos que vamos a usar en este repositorio.

Vamos a **Sublime Text**:



Primero, buscamos el repositorio creado



# ***GIT STATUS***



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

The screenshot shows a code editor window with the title bar indicating the path: `~/Documents/CODERHOUSE/mi_repositorio/ind`. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. On the left, a 'FOLDERS' sidebar shows a tree view with 'mi\_repositorio' expanded, containing 'index.html'. The main editor area displays the content of 'index.html' with line numbers 1 through 10. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8
9 </body>
10</html>
```

# ***GIT STATUS***



👁️ Vamos a la terminal y con **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:
```

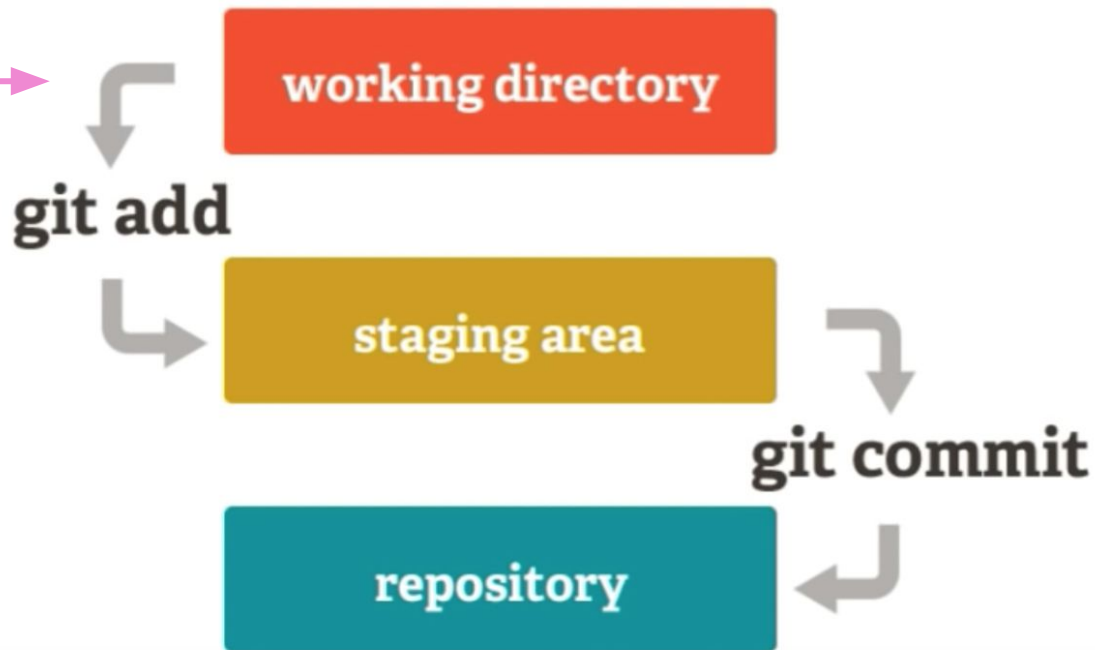
```
(use "git add <file>..." to include in what will be committed)
```

```
index.html
```

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

# ***GIT: ¿RECUERDAN LOS 3 ESTADOS?***

Estamos aquí con  
el ***index.html***  
creado.



# ***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 funciona, 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

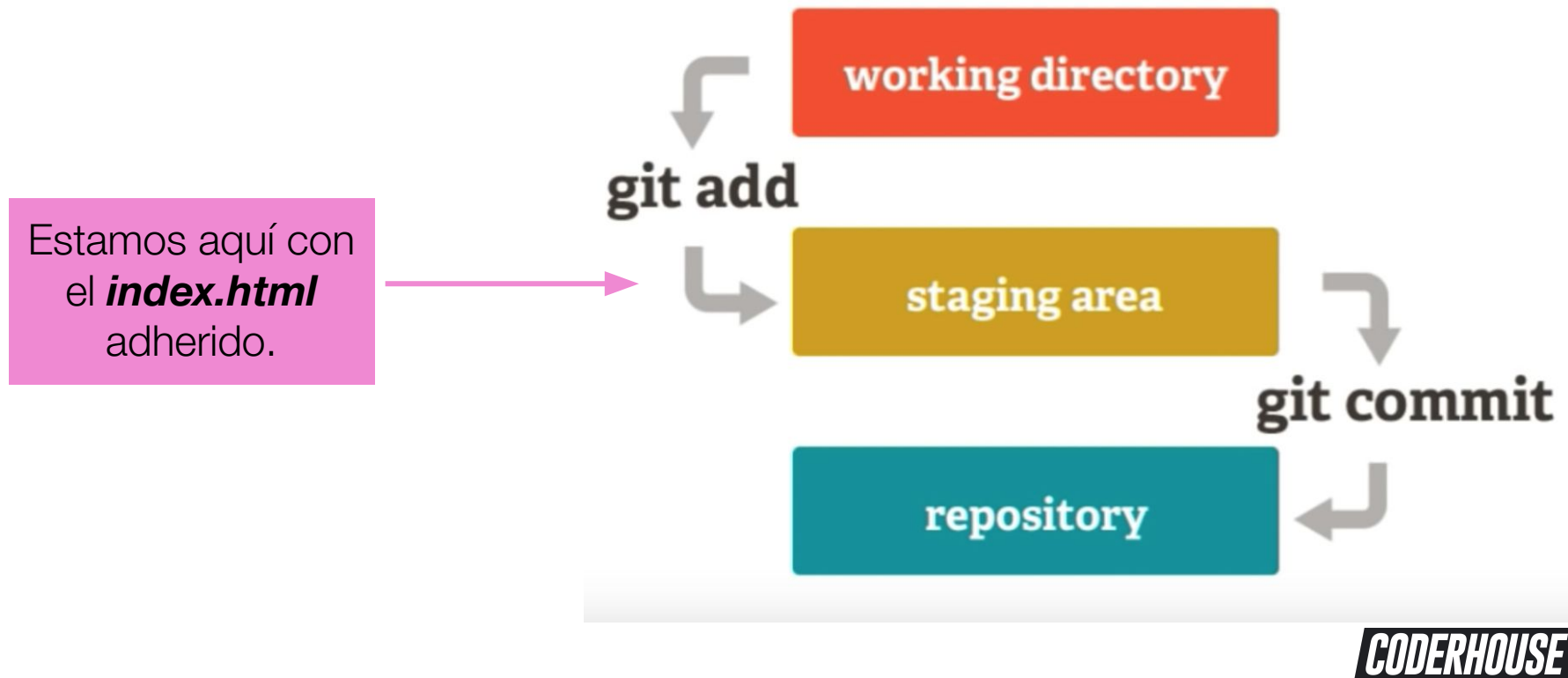
No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: index.html

# ***GIT: ¿RECUERDAN LOS 3 ESTADOS?***



# ***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:

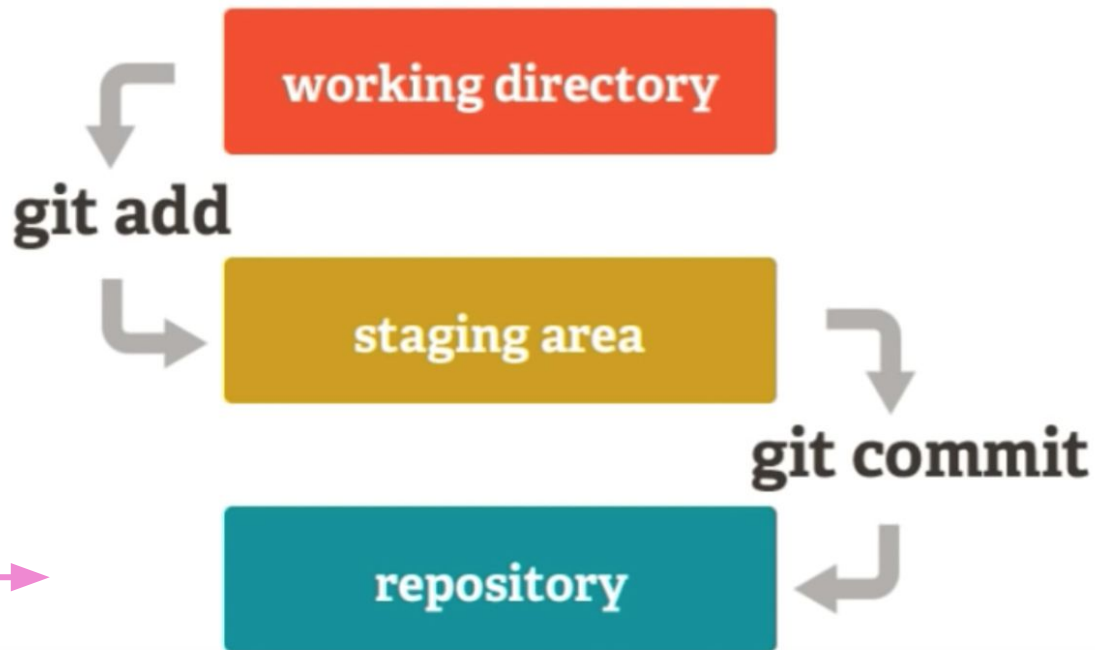
```
git commit -m "Comentario de qué se trata el commit que se está realizando"
```

# ***GIT COMMIT***



```
john@MyShopSolutions :~/Documents/Proyectos_Coder/mi_repositorio$ git commit
-m "Primer archivo del repositorio"
/* Esta sería el resultado del comando */
[master (root-commit) 1734915] nuevo archivo
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
```

# ***GIT: ¿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 1734915470ce9983f703b77807a68e42166b47dd (HEAD ->  
master)
```

```
Author: John Doe <johndoe@example.com>
```

```
Date: Sat May 22 18:53:24 2020 -0300
```

Primer archivo del repositorio

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

<https://git-scm.com/book/es/v1/Fundamentos-de-Git-Viendo-el-hist%C3%B3rico-de-confirmaciones>

***RAMAS***

# ***RAMAS***

CoderTips



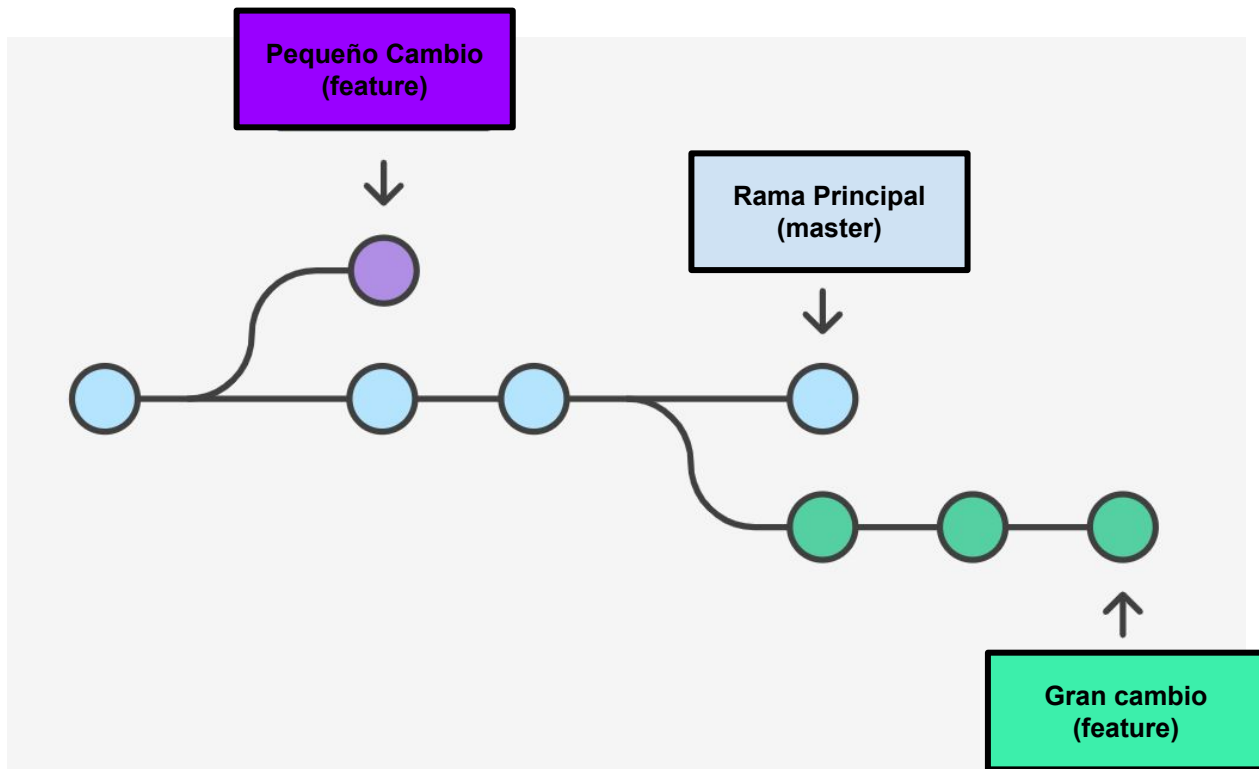
Cuando quieres añadir **una nueva función o solucionar un error** (sin importar su tamaño), generas **una nueva rama para alojar estos cambios**.

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

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



# ***RAMAS***



# ***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 **BRANCH**: 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 6d6c28c)
/* 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 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 co 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)
```

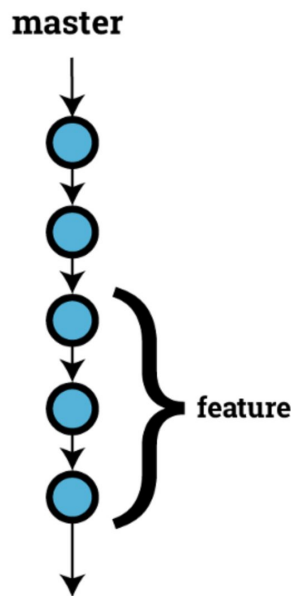
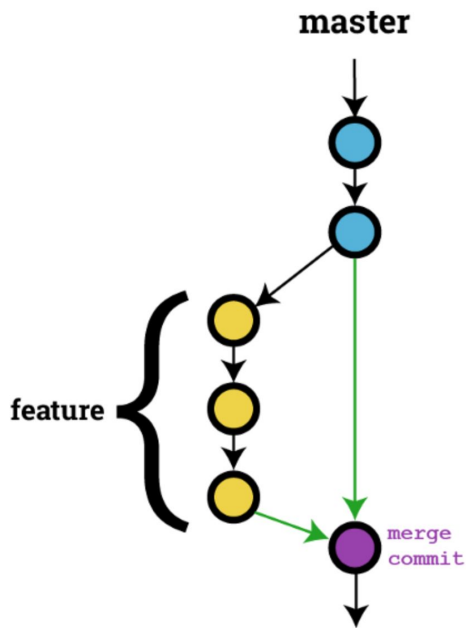
```
master
```

```
nueva_rama
```

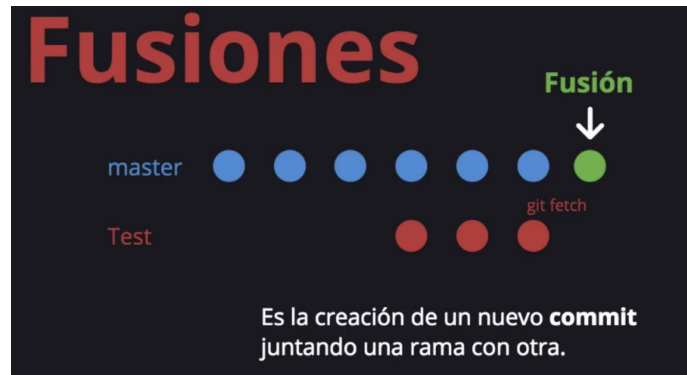
```
john@MyShopSolutions
```

```
:~/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**.



# ***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(+)
```



# ***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**



**PILAR**  
MUNICIPIO

**conectados**  
con las *oportunidades*

***CODERHOUSE***



# ¿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*)



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



The image shows a GitHub commit history page. A large red diamond with a white 'i' icon is overlaid on the left side. The word 'git' is written in large, bold, brown letters across the center. The background shows a list of commits with details like branch names, commit messages, and authors.

Branch	Commit Message	Author	Time
master	Add missing documentation about 'config.ac'	yuuji.yaginuma	13 hours ago
origin/master	Merge pull request #29373 from untidy-hair/store_accessor_enhan	Ryuta Kamizono	2 days ago
	Merge pull request #331 from bogdandvlviv/update-active_sup	Ryuta Kamizono	2 days ago
	Update "Active Store Extensions" Guide	bogdandvlviv	2 days ago
	Add ability to configure notifications info	Eileen Uchitelle	2 days ago
	Update README and allow options for store accessor	Yoshitaka	2 days ago
	Merge pull request #331 from joshuapeters/patches-1	Joshua Peters	2 days ago
	Update README to describe the initializer	Joshua Peters	2 days ago
	Merge pull request #331 from simeonst/active_storage	Ryuta Kamizono	3 days ago
	Update README to describe the initializer	Ryuta Kamizono	3 days ago
	Merge pull request #331 from bogdandvlviv/add-change-log-for	Ryuta Kamizono	3 days ago
	Add CHANGELOG for #32	Ryuta Kamizono	3 days ago
	Merge pull request #331 from bogdandvlviv/fix-active	Ryuta Kamizono	3 days ago
	Fix active _ methods [ci skip]	bogdandvlviv	3 days ago
	Merge pull request #33119 from utilium/project_name	utilium	3 days ago
	Merge pull request #33118 from dan-jensen/fix-eager-load-1	Rafael França	3 days ago
	Remove backticks around project names	utilium	3 days ago

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

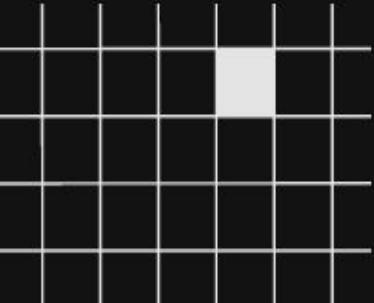
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

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

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODERHOUSE***