



Certified



Corporation®

## MÓDULO 2: Fundamentos de Desarrollo Front-End.

---

### Parte 7: Fundamentos de Git/ GitHub.



## Aprendizajes Esperados

Gestionar el código fuente utilizando GitHub para mantener un repositorio de código remoto seguro y permitir trabajo concurrente.

# Contenidos

## I.- Fundamentos de GIT.

1. Necesidad de un repositorio de código fuente .
2. Instalación, configuración y comandos básicos.
3. Commits y restauración de archivos.
4. Cambios de nombres.
5. Ignorando archivos.
6. Ramas, uniones, conflictos y tags.
7. Stash y Rebase.

## II.- Fundamentos de GitHub.

1. Repositorios remotos, Push y Pull.
2. Fetch v/s Pull.
3. Clonando un repositorio.
4. Documentando un proyecto con Markdown.
5. Administrando Pull Request.
6. Flujos de trabajo con GitHub.

# I. Fundamentos de GIT.

## 1.- Necesidad de un repositorio de código fuente.

La necesidad de un repositorio de código fuente se produce cuando:

- ☐ El producto desarrollado o por desarrollar consta de una gran cantidad de archivos y configuraciones.
- ☐ El equipo de desarrollo es muy grande.
- ☐ Distribución del proyecto que se esta desarrollando a los distintos participantes del equipo.
- ☐ Unión de diferentes módulos o partes del producto que desarrollan los participantes del equipo.



# I. Fundamentos de GIT.

## 2.- Instalación, configuración y comandos básicos.

Para instalar GIT, se debe descargar el ejecutable desde la siguiente URL:

<http://git-scm.com/download/win>

Descargar la versión según sistema operativo a utilizar.

Ejecutar el archivo descargado y esperar la instalación.

## Downloading Git



### Your download is starting...

You are downloading the latest (2.24.0) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released **about 1 month ago**, on 2019-11-06.

If your download hasn't started, [click here to download manually](#).

### Other Git for Windows downloads

Git for Windows Setup

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup](#).

Git for Windows Portable ("thumbdrive edition")

[32-bit Git for Windows Portable](#).

[64-bit Git for Windows Portable](#).

The current source code release is version 2.24.0. If you want the newer version, you can build it from [the source code](#).

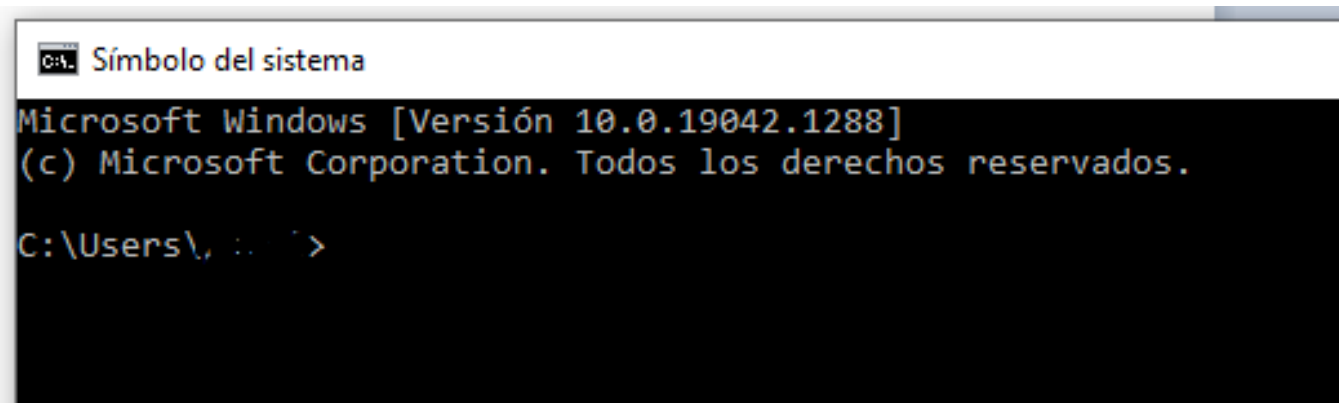
# I. Fundamentos de GIT.

## 2.- Instalación, configuración y comandos básicos.

Luego debes establecer nombre de usuario y correo. Para esto, requieres ejecutar algunos de los siguientes comandos en la consola.

Para acceder a la consola presiona la tecla de Windows y **Símbolo de sistema**.

Aparecerá la consola como muestra la siguiente imagen:



```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.19042.1288]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\>
```

# I. Fundamentos de GIT.

2.- Instalación, configuración y comandos básicos.

Escribir los siguientes comandos:

```
>git config --global user.name "Maria Ignacia"
```

```
>git config --global user.email mimail@servidor.com
```

Esta información es para asociar todas la actualizaciones de forma global a el usuario especificado.

# I. Fundamentos de GIT.

## 3.- Commits y restauración de archivos.

Los Commits son la confirmación de los cambios realizados sobre los archivos.

Para realizar esta confirmación, se utiliza el siguiente comando:

```
>git commit -m "cambios realizados con éxito"
```

Este commit creara una instantánea del proyecto el cual se puede usar para comparar o volver a la versión previa.

Si no todos los archivos fueron subidos, ejecutar el siguiente comando previamente:

```
>git add .
```

Para ver un listado de los commit del mas nuevo al más antiguo utilizar:

```
>git log
```



# I. Fundamentos de GIT.

## 3.- Commits y restauración de archivos.

Para restaurar o volver a una versión x de un archivo, se debe conocer el código del commit.

Para ver el listado, usar `>git log`

Identificar el código o id para obtener la versión guardada y utilizar el siguiente comando:

```
>git checkout a1fefbA
```

Donde a1fefbA, es el id registrado en algún commit realizado previamente.

Para volver al estado actual del proyecto, ejecutar:

```
>git checkout master
```

# I. Fundamentos de GIT.

## 4.- Cambios de nombres.

Para cambiar nombre o renombrar un archivo, utilizar el siguiente comando:

```
o>git mv nombreOriginal.html nombreNuevo.html
```

# I. Fundamentos de GIT.

## 5.- Ignorando archivos.

Para ahorrar tiempo en subida o descarga de los archivos, se puede crear un archivo de configuración que permitirá ignorar a los archivos o carpetas nombrados en el.

Los pasos para realizar esta configuración son:

1.- Crear un archivo en la carpeta raíz del proyecto con nombre:

`.gitignore`

Puede utilizar algun editor, por ejemplo: NotePad, VS Code o similar.

2.- Escribir los archivos y/o carpetas que necesitemos ignorar y guardar cambios.

Ejemplo de archivos

```
*.bat
codigoEjemplos/
archivo.html
```

# I. Fundamentos de GIT.

## 6.- Ramas, uniones, conflictos y tags.

Las ramas, permiten realizar trabajos diferentes sobre la misma base, pero no afectando la base desde donde se creo. Un uso por ejemplo, es probar una librería nueva que se quiere agregar al proyecto.

El atributo de GIT, es que su creación es rápida con respecto a otros sistema de respaldo/repositorio.

Los siguientes comando, permiten:

Crear una nueva rama : `>git branch nuevaRama`

Mostrar la rama actual : `>git branch`

Para cambiarnos de rama : `>git checkout nombreDeLaRama`

Eliminar la rama señalada : `>git branch -d nombreRamaParaBorrar`

# I. Fundamentos de GIT.

6.- Ramas, uniones, conflictos y tags.

Existe la posibilidad de unir 2 ramas. Para esto, se debe utilizar el siguiente comando:

```
>git merge ramaParaUnirConMaster
```

# I. Fundamentos de GIT.

## 6.- Ramas, uniones, conflictos y tags.

Los conflictos surgen cuando se intenta unir 2 ramas y sus respectivos archivos. En el caso de que 2 archivos no tengan el mismo contenido, GIT, no resuelve el conflicto de forma automática.

Para resolver, se deben abrir los archivos que están en conflicto y dejar la porción de código idénticos en ambos.

Git entrega un reporte señalando las líneas de contenido o códigos que tienen conflictos.

Ejemplo de conflicto y su solución (ambos archivos tendrán el mismo código).

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```



```
<div id="footer">
  please contact us at email.support@github.com
</div>
```

# I. Fundamentos de GIT.

6.- Ramas, uniones, conflictos y tags.

Los proyectos pueden ser etiquetados (tags), para marcar los avances importantes o de liberación de una versión para producción.

Para crear una etiqueta, se puede utilizar los siguientes códigos:

```
>git tag v1.0.12
```

```
>git tag v12312 -m "Primera versión producción"
```

Para obtener el listado de etiquetas registradas y su información, utilizar:

```
>git tag -l 'versionMinimaParaVer'
```

```
>git show nombreTag
```

# I. Fundamentos de GIT.

## 7.- Stash y Rebase

**Stash**, permite crear un directorio limpio cuando se requiere volver a un punto anterior del proyecto, sin eliminar los cambios realizados posterior al **commit**.

Estos cambios no deben ser confirmados para que al momento de volver al punto anterior y regresar al trabajo (cambios posteriores) no se vean afectados.

El comando para realizar esta acción es: `>git stash`

Para volver al trabajo incompleto por el cambio a la versión anterior, se utiliza:

```
>git stash pop
```

Los **stash** realizados, generan un historial, el cual puedes acceder a ver con el comando:

```
>git stash list
```



# I. Fundamentos de GIT.

## 7.- Stash y Rebase

- a. Rebase, permite mover una rama completa a un extremo de otra rama, formando una sola rama extensa.
- b. Reescritura del historial al fusionar los commit de ambas ramas.
- c. El historial es lineal, permitiendo una ir directo al inicio sin desviaciones.
- d. La navegación es mas sencilla dentro del proyecto.
- e. Tiene como desventaja la perdida de trazabilidad y seguridad.

Para reorganizar una rama se debe posicionar en la rama a fusionar y ejecutar lo siguiente:

```
>git checkout ramaParaFusionar
```

```
>git rebase master
```

# I. Fundamentos de GitHub.

## 1.- Repositorios remotos, Push y Pull.

Los repositorios remotos son servidores que están disponibles para alojar repositorios GIT. Para este curso, se utilizará GitHub como proveedor de alojamiento git.

Para iniciar el uso de repositorio remoto en GitHub, se debe crear una cuenta en la URL:

<https://github.com>

Y seguir los pasos solicitados en dicha pagina.

**Push** se utiliza para subir los commits o confirmaciones de cambios al repositorio remotos en la rama actual local.

**Pull** se utilizar para descargar las actualizaciones disponibles en el repositorio remoto y las fusiona con el trabajo local. Como recomendación, usar commits antes de usar pull.

# I. Fundamentos de GitHub.

1.- Repositorios remotos, Push y Pull.

Ejemplos:

```
>git push usuario@host:/ruta/repositorio miRama
```

```
>git pull usuario@host:/ruta/repositorio miRama
```

# I. Fundamentos de GitHub.

## 2.- Fetch v/s Pull.

**Fetch** permite recuperar un trabajo nuevo realizado por otras personas, donde se obtiene todas las etiquetas y ramas de seguimiento remoto **sin fusionar** estos cambios en tus propias ramas.

A diferencia del **Pull** que realiza la fusión de ambos trabajos.

Ejemplo Fetch:

```
>git fetch usuario@host:/ruta/repositorio
```

# I. Fundamentos de GitHub.

## 3.- Clonando un repositorio.

Clonar un repositorio es copiar tu repositorio remoto a uno local, con la finalidad de sincronizarlos.

Ejemplo:

```
>git clone usuario@host:/ruta/repositorio
```

Salida esperada:

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
> Cloning into `Spoon-Knife`...
> remote: Counting objects: 10, done.
> remote: Compressing objects: 100% (8/8), done.
> remove: Total 10 (delta 1), reused 10 (delta 1)
> Unpacking objects: 100% (10/10), done.
```

# I. Fundamentos de GitHub.

4.- Documentando un proyecto con Markdown.

Markdown es una forma de estilo de texto en la web.

Controla la visualización del documento como por ejemplo:

1. Formatear palabras como negrita o cursiva.
2. Agregar imágenes.
3. Crear listas.
4. Entre otros.

La sintaxis de toda la escritura y formatos se pueden encontrar en la documentación oficial de GitHub. LaUrl es:

<https://help.github.com/es/github/writing-on-github/basic-writing-and-formatting-syntax>

# I. Fundamentos de GitHub.

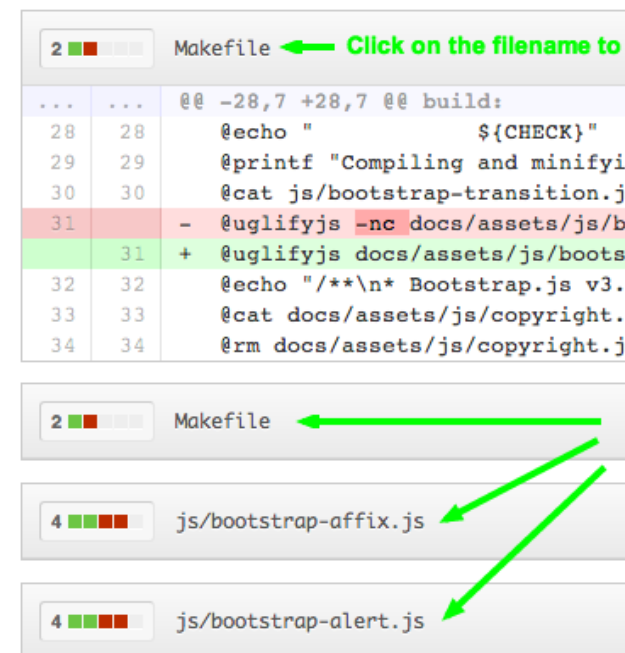
## 5.- Administrando Pull Request.

Un Pull Request es una solicitud realizada al repositorio cuando se envía nuevos códigos con cambios y donde el encargado o dueño del repositorio, se encarga de revisar este código y decide si es apto para ser integrado.

Esta funcionalidad promueve la colaboración de los distintos integrantes en un proyecto.

También es un foro para debatir, publicar bug o errores, entregar feedback, modificar el código propuesto, entre otras tareas.

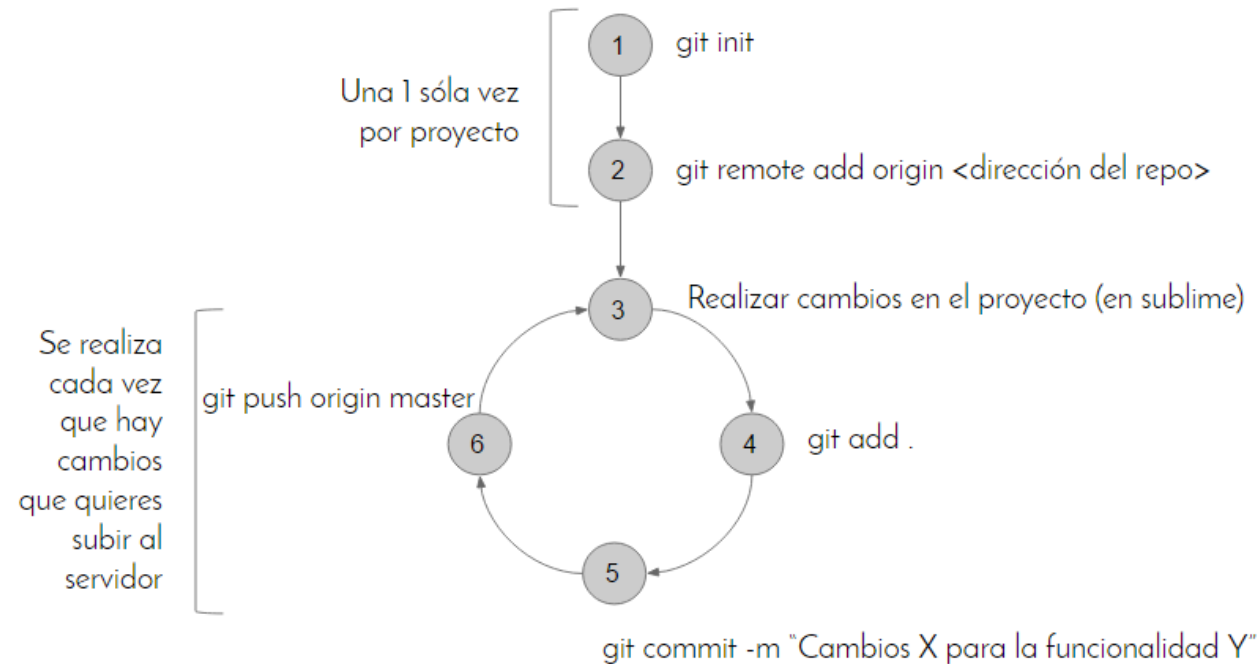
Los Pull Request, permiten comparar los códigos del archivo original con la propuesta realizada, mostrando en forma grafica el nuevo código o sus modificaciones.



# I. Fundamentos de GitHub.

## 6.- Flujos de trabajo con GitHub.

A continuación, se muestra un diagrama con el flujo de trabajo sugerido cuando se utiliza GitHub como repositorio de los proyectos desarrollados.





# Actividad.

