



3. Trabajando con Git

Inicio o init.

El comando `git init` se utiliza para **inicializar un nuevo repositorio de Git** en un directorio.

Convierte una carpeta normal en un repositorio de Git, creando una subcarpeta oculta llamada `.git` que almacena toda la información del control de versiones.

- Se usa una sola vez al comenzar un nuevo proyecto con Git.
- Crea el repositorio de forma local en el ordenador.
- No afecta los archivos existentes, solo agrega la estructura de Git.

Ejemplo. Tenemos una carpeta llamada `mi_proyecto`.

```
cd mi_proyecto # Entramos en la carpeta del proyecto
git init # Inicializamos el repositorio de Git
```

Después de ejecutar este comando, Git comienza a rastrear los archivos dentro de esta carpeta.

```
Git status # Muestra el estado de los archivos en el repositorio
```

```
On branch main

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)
```

Seguimiento o track.

Proceso para **indicarle a Git qué archivos debe monitorear** dentro del repositorio. Esto se hace con el comando `git add` → mueve los archivos al **staging area** (área de preparación).

- Un archivo no rastreado es aquel que aún no ha sido agregado a Git.
- Una vez agregado con `git add`, Git empezará a hacer seguimiento de sus cambios.
- Si un archivo no está en seguimiento, no se incluirá en los commits.

Ejemplo, añadimos `index.html` para que Git comience a rastrearlo.

```
git add index.html # Agrega el archivo al área de preparación
```



Si queremos agregar todos los archivos nuevos o modificados a la vez:

```
git add . # Agrega todos los archivos al área de preparación
```

Se pueden agregar varios archivos separando los nombres con un espacio.

Comprobamos el estado: `Git status`

```
On branch main

No commits yet

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

Publicar o commit

Un **commit** en Git, guarda los cambios de los archivos en el historial de Git de manera permanente.

- Cada commit tiene un mensaje descriptivo sobre los cambios realizados.
- Permite volver a versiones anteriores si me interesa.
- No sube los cambios a GitHub, solo los guarda en el repositorio local

Ejemplo, después de agregar los archivos con `git add`, se debe confirmar los cambios con un commit:

```
git commit -m "Agregar página de inicio"
```

Si queremos hacer un commit que incluya todos los archivos modificados:

```
git commit -am "Corrección de errores en el CSS"
```

Revisión o versión (Hash ID)

Cada commit en Git tiene un identificador único llamado **Hash ID**. Este identificador es un código alfanumérico generado por Git usando el algoritmo SHA-1 y sirve para referenciar un commit específico dentro del historial del repositorio.

Ejemplo, para ver el historial de commits y sus Hash IDs:

```
git log
```

```
commit 85a92e6b043132cc38210a24db0efe4630a99274 (HEAD -> main)
Author: rocio <rociolopez@iesgregorioprieto.com>
Date:   Sun Mar 23 18:11:32 2025 +0100

    agregar página de inicio
```



Se pueden usar los primeros 7 caracteres del Hash ID en lugar de todo código.

Head

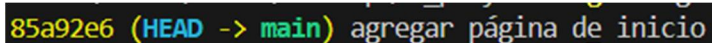
Es un puntero especial que indica el commit actual en el que te encuentras dentro de un repositorio Git. Por lo general, HEAD apunta a la última versión confirmada en la rama activa.

- Se puede mover a un commit anterior para revisar cambios previos.
- Se usa en comandos como `git checkout` y `git reset`.

Ejemplos:

Para ver a qué commit está apuntando HEAD

```
git log --oneline
```



Para mover HEAD a un commit anterior para revisar el estado del proyecto en ese punto

```
git checkout 1a2b3c4 # Mueve HEAD a un commit anterior
```

Para regresar a la última versión de la rama principal

```
git checkout main
```

Etiquetar o tag.

Es una referencia a un commit específico, generalmente utilizada para marcar versiones importantes del proyecto (versiones estables), como lanzamientos de software (v1.0, v2.0, etc.).

Puede ser **ligero** (solo referencia un commit) o **anotado** (con metadatos como autor y fecha).

Ejemplos:

Para crear un tag ligero en el commit actual:

```
git tag v1.0
```

Para crear un tag anotado con descripción:

```
git tag -a v1.0 -m "Primera versión estable del proyecto"
```



Para listar todas las etiquetas en el repositorio:

```
git tag
```

Para compartir los tags en un repositorio remoto (como GitHub):

```
git push origin --tags
```

Ramas o branch

Es una versión independiente del código donde se puede trabajar sin afectar la rama principal. Se usan para desarrollar nuevas características, corregir errores o experimentar sin modificar el código en producción.

- Permiten desarrollar cambios sin afectar la rama principal.
- Se pueden crear varias ramas para trabajar en diferentes funciones a la vez.
- Se pueden fusionar con la rama principal una vez que el desarrollo esté listo.

Ejemplos:

Para ver las ramas disponibles en un repositorio:

```
git branch
```

Para crear una nueva rama llamada `nueva-funcionalidad`:

```
git branch nueva-funcionalidad
```

Para cambiar a esa rama:

```
git checkout nueva-funcionalidad # Método antiguo
```

```
git switch nueva-funcionalidad # Método más reciente
```

Master/Main Branch

La rama **master** o **main** es la rama principal del repositorio. Es la versión estable del código.

- Es la rama por defecto cuando se crea un repositorio en Git.
- Las nuevas funcionalidades se desarrollan en ramas separadas y luego se fusionan con `main`.

Ejemplos:

Para asegurarte de que estás en la rama principal antes de fusionar cambios:

```
git checkout main # O git switch main
```



Para cambiar el nombre de `master` a `main` en un repositorio existente:

```
git branch -m master main
```

Fusionar o Merge

Merge combina los cambios de una rama con otra. Se usa principalmente para integrar el trabajo de una rama de desarrollo con la rama principal (`main`). Se usa después de terminar el desarrollo en una rama secundaria.

Ejemplo, después de trabajar en `nueva-funcionalidad`, se vuelve a `main` y se fusiona:

```
git checkout main
```

```
git merge nueva-funcionalidad
```

Una vez hecho esto, podemos eliminar la rama con:

```
Git branch -d nueva-funcionalidad
```

Este vídeo muestra un ejemplo de conflicto y cómo resolverlo:

<https://www.youtube.com/watch?v=tFr0Vg1q9Eg>

Ejercicio:

- Crea una nueva rama llamada `estilos`. Cambiate a esta rama.
- Si no lo tienes todavía, crea un fichero de hoja de estilos (`estilos.css`).
- Modifica el fichero `index.html` para que trabaje con tu hoja de estilos.
- Añade los ficheros al área de preparación.
- Confirma los cambios en la rama.
- Fusiona los cambios con la rama principal

Revertir o Revert

El comando `git revert` **crea un nuevo commit que deshace los cambios de un commit anterior**, sin eliminar el historial del repositorio. Es útil cuando necesitas deshacer cambios sin perder el registro de lo que se hizo.

- No borra commits, solo genera uno nuevo que revierte los cambios.
- Es seguro para usar en repositorios compartidos.



Ejemplos:

Para revertir un commit específico, primero se obtiene su Hash ID:

```
git log --oneline
```

Para revertir el commit 1a2b3c4:

```
git revert 1a2b3c4
```

Esto generará un nuevo commit que deshace los cambios del commit seleccionado.

Para eliminar commits y cambios permanentemente en el historial de Git → git reset --hard

```
git reset --hard 1a2b3c4
```