

Introducción a Git y GitHub

1. Conceptos Básicos de Control de Versiones

¿Qué es el control de versiones?

El control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, permitiéndote:

- Recuperar versiones específicas más adelante
- Comparar cambios a lo largo del tiempo
- Ver quién modificó por última vez algo
- Trabajar colaborativamente sin conflictos

¿Por qué necesitamos control de versiones?

Imagina esta situación: Estás trabajando en un sitio web y guardas diferentes versiones:

```
proyecto-final.html  
proyecto-final-v2.html  
proyecto-final-ESTE-SÍ.html  
proyecto-final-DEFINITIVO.html  
proyecto-final-DEFINITIVO-AHORA-SÍ.html
```

Este método es:

- Confuso (¿cuál es la versión más reciente?)
- Propenso a errores (puedes editar la versión incorrecta)
- Ineficiente para colaborar con otros

Git resuelve estos problemas de manera elegante.

¿Qué es Git?

Git es un **sistema de control de versiones distribuido** creado por Linus Torvalds (el creador de Linux) que:

- Registra historial completo de cambios
- Permite trabajar sin conexión a internet
- Facilita la colaboración entre múltiples personas
- Es rápido y eficiente

¿Qué es GitHub?

GitHub es una plataforma en línea que:

- Almacena repositorios Git en la nube
- Proporciona una interfaz web para gestionar proyectos

- Facilita la colaboración entre desarrolladores
- Ofrece herramientas adicionales (seguimiento de problemas, wikis, etc.)

Terminología básica

Antes de comenzar con los comandos, familiaricémonos con algunos términos:

- **Repositorio:** Carpeta donde Git almacena todas las versiones y metadatos de tu proyecto
- **Commit:** Una "fotografía" de tu código en un momento específico
- **Rama (branch):** Una línea independiente de desarrollo
- **Repositorio remoto:** Copia del repositorio alojada en un servidor (como GitHub)
- **Clonar:** Crear una copia local de un repositorio remoto
- **Pull:** Descargar cambios del repositorio remoto
- **Push:** Subir cambios locales al repositorio remoto
- **Área de preparación (staging area):** Zona intermedia donde colocamos cambios antes de confirmarlos

2. Instalación y Configuración Inicial

Instalación de Git

En Windows:

1. Descarga el instalador desde git-scm.com
2. Ejecuta el instalador y sigue las instrucciones (puedes dejar las opciones predeterminadas)

En Mac:

1. Si tienes Homebrew: `brew install git`
2. O descarga desde git-scm.com

En Linux:

- Debian/Ubuntu: `sudo apt-get install git`
- Fedora: `sudo dnf install git`

Configuración básica

Una vez instalado, configura tu identidad (esto quedará registrado en tus commits):

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu.email@ejemplo.com"
```

Puedes verificar la configuración con:

```
git config --list
```

3. Comandos Principales

git init

Este comando crea un nuevo repositorio Git en tu directorio actual.

```
mkdir mi-primer-proyecto  
cd mi-primer-proyecto  
git init
```

Esto creará una carpeta oculta llamada `.git` que contendrá toda la información necesaria para el control de versiones.

git status

Te muestra el estado actual de tu repositorio:

- Qué archivos han sido modificados
- Qué cambios están preparados para commit
- Qué archivos no están siendo rastreados por Git

```
git status
```

git add

Este comando añade archivos al "área de preparación" (staging area), marcándolos para ser incluidos en el próximo commit.

```
# Añadir un archivo específico  
git add index.html  
  
# Añadir todos los archivos HTML  
git add *.html  
  
# Añadir todos los archivos y carpetas  
git add .
```

git commit

Guarda los cambios en el repositorio con un mensaje descriptivo.

```
git commit -m "Creación de página principal"
```

Es muy importante escribir mensajes de commit claros y descriptivos que expliquen **qué** cambios realizaste y **por qué**.

git log

Muestra el historial de commits.

```
git log
```

Para una versión más compacta:

```
git log --oneline
```

git remote

Permite gestionar conexiones con repositorios remotos (como GitHub).

Para añadir un repositorio remoto:

```
git remote add origin https://github.com/tu-usuario/tu-repositorio.git
```

Para ver los repositorios remotos configurados:

```
git remote -v
```

git push

Envía tus commits locales al repositorio remoto.

```
# La primera vez
git push -u origin main

# Después
git push
```

git pull

Descarga y fusiona los cambios del repositorio remoto en tu copia local.

```
git pull
```

git clone

Crea una copia local de un repositorio remoto existente.

```
git clone https://github.com/usuario/repositorio.git
```

4. Creación de Cuenta en GitHub

Paso a paso para crear una cuenta

1. Ve a github.com
2. Haz clic en "Sign up" (Registrarse)
3. Proporciona:
 - Tu correo electrónico
 - Una contraseña segura
 - Un nombre de usuario único
4. Completa la verificación de seguridad
5. Haz clic en "Create account" (Crear cuenta)
6. Verifica tu dirección de correo electrónico siguiendo el enlace enviado

Configuración del perfil

1. Haz clic en tu avatar en la esquina superior derecha
2. Selecciona "Settings" (Configuración)
3. Puedes:
 - Añadir una foto de perfil
 - Completar tu nombre y biografía
 - Ajustar preferencias de notificación
 - Configurar opciones de seguridad

Creación de tu primer repositorio en GitHub

1. Haz clic en el signo "+" en la esquina superior derecha
2. Selecciona "New repository" (Nuevo repositorio)
3. Completa el formulario:
 - Nombre del repositorio
 - Descripción (opcional)
 - Visibilidad: público o privado
 - Opción para inicializar con README (recomendado para principiantes)
4. Haz clic en "Create repository" (Crear repositorio)

5. Subir tu Proyecto HTML a GitHub

Método 1: Subir un proyecto existente

Si ya tienes un proyecto HTML en tu computadora:

1. Navega a la carpeta de tu proyecto en la terminal

```
cd ruta/a/tu/proyecto
```

2. Inicializa Git

```
git init
```

3. Añade todos los archivos

```
git add .
```

4. Haz tu primer commit

```
git commit -m "Versión inicial del proyecto"
```

5. Conecta con tu repositorio remoto

```
git remote add origin https://github.com/tu-usuario/tu-repositorio.git
```

6. Sube los cambios

```
git push -u origin main
```

Nota: En versiones más recientes de Git, la rama principal se llama "main". En versiones anteriores, se llamaba "master". Si Git te muestra un error, intenta:

```
git push -u origin master
```

Método 2: Clonar un repositorio vacío y añadir archivos

1. Crea un repositorio vacío en GitHub

2. Clona el repositorio en tu computadora

```
git clone https://github.com/tu-usuario/tu-repositorio.git
```

3. Copia tus archivos HTML en la carpeta clonada

4. Añade los archivos, haz commit y push

```
git add .  
git commit -m "Añadidos archivos iniciales"  
git push
```

Actualizar tu proyecto en GitHub

Cada vez que realices cambios en tu proyecto local:

1. Verifica qué archivos han cambiado

```
git status
```

2. Añade los cambios

```
git add .
```

3. Haz un commit con un mensaje descriptivo

```
git commit -m "Actualizada la página de contacto"
```

4. Sube los cambios

```
git push
```

6. Flujo de Trabajo Básico

Un flujo de trabajo típico:

1. **Actualiza tu repositorio local** (si trabajas con otros)

```
git pull
```

2. **Trabaja en tus archivos** utilizando tu editor de código

3. **Revisa los cambios**

```
git status
git diff
```

4. Prepara los cambios

```
git add .
```

5. Confirma los cambios

```
git commit -m "Mensaje descriptivo"
```

6. Comparte los cambios

```
git push
```

7. Ejercicios Prácticos

Ejercicio 1: Tu primera página web en GitHub

1. Crea un repositorio llamado "mi-primer-web" en GitHub
2. Clona el repositorio en tu computadora
3. Crea un archivo `index.html` con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
  />
  <title>Mi Primera Página</title>
</head>
<body>
  <h1>iHola Mundo!</h1>
  <p>Esta es mi primera página web con control de versiones.</p>
</body>
</html>
```

4. Añade, haz commit y push de este archivo
5. Verifica en GitHub que tu archivo aparezca correctamente

Ejercicio 2: Realizar modificaciones

1. Modifica tu archivo `index.html` para añadir:

- Un encabezado h2
- Una lista de tus 3 comidas favoritas
- Un pie de página con tu nombre

2. Añade, haz commit y push de estos cambios

3. Verifica el historial de commits en GitHub

Ejercicio 3: Explorar versiones anteriores

1. Realiza al menos 3 cambios diferentes en tu página, haciendo un commit después de cada cambio
2. En GitHub, explora la pestaña "Commits"
3. Haz clic en diferentes commits para ver cómo era tu código en cada etapa





8. Consejos y Buenas Prácticas

Mensajes de commit efectivos

Un buen mensaje de commit debe:

- Ser claro y conciso
- Explicar QUÉ cambió y POR QUÉ
- Usar el imperativo ("Añade función" en lugar de "Añadida función")

Ejemplos:

-  "Añade formulario de contacto"
-  "Corrige error de validación en el formulario"
-  "Cambios"
-  "Actualización"

Archivo README.md

Todo buen repositorio tiene un archivo README.md que explica:

- Qué es el proyecto
- Cómo instalarlo/usarlo
- Quién lo ha creado
- Cómo contribuir (si es abierto)

GitHub mostrará automáticamente el contenido de este archivo en la página principal del repositorio.

Ejemplo básico de README.md:

```
# Mi Proyecto Web
```

```
Este es un proyecto para aprender HTML y control de versiones con Git y GitHub.
```

Contenido

- Página principal con información sobre mí
- Lista de mis intereses
- Formulario de contacto

Autor

Creado por [Tu Nombre]

Ignorar archivos

No todos los archivos deben incluirse en tu repositorio (archivos temporales, configuraciones personales, etc.).

Crea un archivo **.gitignore** en la raíz de tu proyecto para especificar qué archivos o carpetas ignorar:

```
# Archivos de sistema
.DS_Store
Thumbs.db

# Directorios de dependencias
/node_modules/

# Archivos temporales
*.tmp
*.log
```

9. Resolución de Problemas Comunes

"No puedo hacer push: rechazo de non-fast-forward"

Significa que hay cambios en el repositorio remoto que no tienes localmente.

Solución:

```
git pull
# Resuelve cualquier conflicto si es necesario
git push
```

"Conflicto de fusión (merge conflict)"

Ocurre cuando Git no puede combinar automáticamente los cambios.

Solución:

1. Abre los archivos con conflictos (marcados con <<<<<<, =====, >>>>>>)
2. Edita los archivos para resolver los conflictos manualmente

3. Guarda los archivos
4. Añade los archivos resueltos

```
git add .
```

5. Completa el merge

```
git commit
```

"He cometido un error en mi último commit"

Si aún no has hecho push:

```
# Modificar el último commit
git commit --amend -m "Mensaje correcto"

# O si solo quieres añadir más cambios sin cambiar el mensaje
git add archivo-olvidado.html
git commit --amend --no-edit
```

10. Recursos Adicionales

Para seguir aprendiendo

- [Documentación oficial de Git](#)
- [Guía de GitHub](#)
- [Git Cheat Sheet \(PDF\)](#)
- [Learn Git Branching](#) - Tutorial interactivo
- [Oh Shit, Git!?!](#) - Soluciones a problemas comunes

Herramientas gráficas

Para quienes prefieren no usar la línea de comandos:

- [GitHub Desktop](#)
- [GitKraken](#)
- [Sourcetree](#)
- Extensiones para editores como Visual Studio Code