💻 Terminal y Control de Versiones con Git

Tu primer paso hacia el desarrollo profesional

Fundamentos de la Web - Clase 11

🎯 Objetivos de Hoy

¿Qué aprenderemos?

- 💻 Dominar comandos esenciales de la terminal
- 📁 Navegar por el sistema de archivos sin mouse
- 🔧 Crear, mover y eliminar archivos desde la terminal
- 🌳 Entender Git y el control de versiones
- 📦 Crear repositorios y hacer commits
- 🌐 Conectar proyectos locales con GitHub
- 🚀 Usar Git desde Visual Studio Code
- 📄 Publicar sitios web con GitHub Pages

La Terminal

Tu nuevo superpoder

💻 ¿Qué es la Terminal?

¿Por qué necesito esto?

¡Es lo que usan los desarrolladores profesionales todos los días!

¿Qué puedes hacer con la terminal?

- 📂 Navegar por carpetas y archivos
- ⚙️ Configurar servidores en la nube
- 🧪 Ejecutar pruebas automáticas
- 💾 Administrar bases de datos
- 🚀 Desplegar aplicaciones

🖥️ Terminal vs Interfaz Gráfica

Dos formas de hacer lo mismo:

Interfaz Gráfica (GUI)	Terminal (CLI)
🖱️ Clic en iconos	⌨️ Escribir comandos
👁️ Visual e intuitiva	💻 Texto puro
🐢 Más lenta para tareas repetitivas	⚡ Súper rápida
🎨 Bonita y colorida	🔤 Minimalista
👶 Fácil para principiantes	🚀 Poderosa para profesionales

Como desarrolladores, necesitamos dominar ambas 💪

🖥️ Cómo Abrir la Terminal

🍎 En Mac:

1. Presiona `Comando + Barra espaciadora`
2. Escribe "Terminal"
3. Presiona Enter

🪟 En Windows:

1. Presiona `Windows + R`
2. Escribe "cmd"
3. Presiona Enter

💡 **Tip:** También puedes usar **CMDER** en Windows (una terminal mejorada)

🪟 CMDER para Windows

¿Qué es CMDER?

Una terminal más amigable y potente para Windows que hace tu vida más fácil.

¿Por qué usarlo?

- ✅ Comandos de Linux funcionan en Windows
- ✅ Interfaz más moderna y colorida
- ✅ Múltiples pestañas
- ✅ Copiar y pegar funciona bien

📥 **Descarga:** github.com/cmderdev/cmder / https://cmder.app/

🎯 **Instrucciones:** Descarga → Descomprime → Ancla a la barra de tareas

💻 Anatomía de la Terminal

¿Qué ves cuando la abres?

```
C:\Users\TuNombre>_
```

Desglosando cada parte:

- `C:\Users\TuNombre` → **Ruta actual:** Dónde estás ubicado
- `>` → **Prompt:** Indica que la terminal está lista
- `_` → **Cursor:** Donde escribes tus comandos

Después de escribir un comando y presionar Enter:

→ La terminal ejecuta la acción

→ Muestra el resultado

→ Te devuelve el prompt para el siguiente comando

📍 Comandos Básicos

Navegando por tu computadora

📍 pwd - ¿Dónde estoy?

Present Working Directory (Directorio de Trabajo Actual)

¿Cuándo lo necesito?

Cuando no estés seguro de dónde te encuentras en tu computadora.

```
pwd
```

Salida:

```
/Users/TuNombre/Desktop
```

💡 **En Windows (cmd):** Usa `cd` sin argumentos en vez de `pwd`

¿Por qué es útil?

- ✅ Siempre sabes tu ubicación
- ✅ Evitas errores al crear o eliminar archivos
- ✅ Te orienta antes de ejecutar comandos importantes

📄 ls - ¿Qué hay aquí?

List (Listar archivos y carpetas)

¿Cuándo lo necesito?

Para ver qué archivos y carpetas hay en tu ubicación actual.

```
ls
```

Salida:

```
Documentos    Descargas    Imágenes    proyectos
```

💡 **En Windows (cmd):** Usa `dir` en lugar de `ls`

Variaciones útiles:

- `ls -l` → Lista detallada (permisos, tamaño, fecha)
- `ls -a` → Muestra archivos ocultos (los que empiezan con `.`)
- `ls -la` → Combina ambas opciones

💻 Ejemplo: Explorando con ls

Veamos en detalle:

```
# Lista simple
ls
Documentos    Descargas    proyectos

# Lista detallada
ls -l
drwxr-xr-x  5 usuario  staff   160 Oct 20 10:30 Documentos
drwxr-xr-x  3 usuario  staff    96 Oct 19 14:22 Descargas
drwxr-xr-x  8 usuario  staff   256 Oct 21 09:15 proyectos

# Mostrar archivos ocultos
ls -a
.           ..          .gitconfig  Documentos  Descargas
```

Nota: Los archivos que empiezan con `.` están ocultos por defecto

🔀 cd - Cambiar de Directorio

Change Directory (Cambiar Carpeta)

¿Cuándo lo necesito?

Para moverte entre carpetas.

```
# Ir a una carpeta específica
cd Documentos
# Subir un nivel (ir a la carpeta padre)
cd ..
# Ir al directorio raíz
cd /
# Ir a tu carpeta de usuario
cd ~
```

💡 **Tip:** Usa `Tab` para autocompletar nombres de carpetas

💻 Ejemplo: Navegando con cd

Paso a paso:

```
# Ver dónde estoy
pwd
/Users/TuNombre
# Ver qué hay aquí
ls
Documentos    Descargas    proyectos
# Entrar a proyectos
cd proyectos
# Confirmar ubicación
pwd
/Users/TuNombre/proyectos
# Volver atrás
cd ..
# Confirmar que volví
pwd
/Users/TuNombre
```

❓ Pregunta Rápida

Si estoy en esta ubicación:

```
pwd
/Users/TuNombre/Documentos/proyectos/web
```

¿Qué comando me lleva a `/Users/TuNombre`?

A) `cd ~`

B) `cd ..`

C) `cd ../../..`

D) Tanto A como C son correctas

Piensa unos segundos... 🤔

✅ Respuesta

```
pwd
/Users/TuNombre/Documentos/proyectos/web
```

Respuesta correcta: D) Tanto A como C ¿Por qué?

Opción A: `cd ~`

- `~` es un atajo que significa "mi carpeta de usuario"
- Te lleva directo a `/Users/TuNombre`

Opción C: `cd ../../..`

- `..` = subir un nivel
- `../../..` = subir 3 niveles (web → proyectos → Documentos → TuNombre)

¡Excelente si acertaste! 🎉

+ Creando y Modificando Archivos

➕ mkdir - Crear Carpetas

Make Directory (Crear Directorio)

¿Cuándo lo necesito?

Para organizar tus proyectos en carpetas.

```
# Crear una carpeta
mkdir mi-proyecto

# Crear varias carpetas a la vez
mkdir proyecto1 proyecto2 proyecto3

# Crear carpetas anidadas
mkdir -p proyectos/web/sitio1
```

💡 **Tip:** La opción `-p` crea todas las carpetas intermedias necesarias

📝 touch - Crear Archivos

Crear archivos vacíos

¿Cuándo lo necesito?

Para crear archivos rápidamente desde la terminal.

```
# Crear un archivo
touch index.html
# Crear múltiples archivos
touch index.html style.css script.js
# Crear archivo en otra ubicación
touch /ruta/completa/archivo.txt
```

💡 **En Windows (cmd):** Usa `type nul > archivo.txt`

¿Por qué es útil? ¡Mucho más rápido que crear con clic derecho!

💻 Ejemplo: Creando Estructura de Proyecto

Vamos a crear un proyecto web completo:

```
# Paso 1: Crear carpeta del proyecto
mkdir mi-sitio-web
# Paso 2: Entrar a la carpeta
cd mi-sitio-web
# Paso 3: Crear archivos principales
touch index.html style.css script.js
# Paso 4: Crear carpetas para recursos
mkdir images css js
# Paso 5: Ver el resultado
ls
css/    images/    index.html    js/    script.js    style.css
```

✅ **¡Proyecto listo en segundos!**

🗑️ rm - Eliminar Archivos

Remove (Eliminar)

```
# Eliminar un archivo
rm archivo.txt
# Eliminar una carpeta vacía
rmdir carpeta_vacia
# Eliminar carpeta con contenido
rm -rf carpeta_con_archivos
```

⚠️ **ADVERTENCIA:**

- No hay papelera de reciclaje
- Los archivos se eliminan permanentemente
- ¡Usa con cuidado!

💡 **Tip:** Siempre verifica con `ls` antes de eliminar

✏️ mv - Mover y Renombrar

Move (Mover)

¿Cuándo lo necesito?

Para mover archivos o cambiarles el nombre.

```
# Renombrar un archivo
mv viejo.txt nuevo.txt
# Mover a otra carpeta
mv archivo.txt /ruta/destino/
# Mover y renombrar al mismo tiempo
mv archivo.txt /ruta/destino/nuevo_nombre.txt
# Mover varios archivos
mv archivo1.txt archivo2.txt /carpeta/
```

¿Por qué es útil? ¡No necesitas copiar y pegar con el mouse!

📄 cp - Copiar Archivos

Copy (Copiar)

¿Cuándo lo necesito?

Para hacer duplicados de archivos o carpetas.

```
# Copiar un archivo
cp original.txt copia.txt
# Copiar a otra ubicación
cp archivo.txt /otra/carpeta/
# Copiar una carpeta completa
cp -r carpeta_original carpeta_copia
# Copiar múltiples archivos
cp archivo1.txt archivo2.txt /destino/
```

💡 **Nota:** La opción `-r` (recursive) es necesaria para copiar carpetas

👀 cat - Ver Contenido de Archivos

Concatenate (Mostrar contenido)

¿Cuándo lo necesito? Para leer archivos de texto rápidamente.

```
# Ver contenido de un archivo
cat index.html
# Ver múltiples archivos
cat archivo1.txt archivo2.txt
# Ver con números de línea
cat -n archivo.txt
```

Salida:

```
<!DOCTYPE html>
<html>
<head>
    <title>Mi Sitio</title>
</head>
...
```

💪 Ejercicio: Comandos Básicos

¡Hora de practicar! Crea la siguiente estructura de proyecto:

```
mi-portafolio/
├── index.html
├── sobre-mi.html
├── css/
│   └── styles.css
├── js/
│   └── app.js
└── images/
```

Pistas:

- Usa `mkdir` para crear carpetas
- Usa `touch` para crear archivos
- Usa `cd` para navegar
- Usa `ls` para verificar

⏱️ **Tiempo:** 5 minutos

🔐 sudo - Superpoderes en la Terminal

Super User Do (Ejecutar como administrador)

```
# Instalar un programa
sudo apt-get install git
# Crear carpeta en zona protegida
sudo mkdir /usr/local/mi-carpeta
# Editar archivo del sistema
sudo nano /etc/hosts
```

💡 **Importante:**

- Te pedirá tu contraseña
- Úsalo con cuidado
- No lo necesitas para archivos en tu carpeta personal

Analogía: Es como tener una llave maestra 🔑

❓ Pregunta Rápida

¿Qué hace este comando?

```
mkdir -p proyectos/web/sitio1 && cd proyectos/web/sitio1 && touch index.html
```

A) Crea carpetas, entra a sitio1 y crea index.html

B) Solo crea las carpetas

C) Da error porque faltan comandos

D) Crea index.html en la carpeta actual

Piensa unos segundos... 🤔

✅ Respuesta

```
mkdir -p proyectos/web/sitio1 && cd proyectos/web/sitio1 && touch index.html
```

Respuesta correcta: A)

¿Por qué?

- `mkdir -p proyectos/web/sitio1` → Crea las carpetas
- `&&` → "Y después ejecuta esto"
- `cd proyectos/web/sitio1` → Entra a la carpeta
- `&&` → "Y después ejecuta esto"
- `touch index.html` → Crea el archivo

El símbolo `&&` encadena comandos para ejecutarlos uno tras otro.

¡Perfecto si acertaste! 🎉

🌳 Git - Control de Versiones

Tu máquina del tiempo para código

🌳 ¿Qué es Git?

¿Por qué necesito esto?

Imagina que estás escribiendo un libro. Git es como tener una máquina del tiempo que:

- 📸 Toma "fotos" de tu proyecto en diferentes momentos
- ⏮️ Te permite volver atrás si algo sale mal
- 🔀 Prueba ideas nuevas sin romper lo que funciona
- 👥 Trabaja en equipo sin sobrescribir el trabajo de otros

Definición:

Git es un **sistema de control de versiones** que rastrea cambios en tus archivos.

¿Quién lo usa?

¡Todos los desarrolladores del mundo! 🌍

🌳 ¿Por qué Git es importante?

Problemas sin Git:

❌ "proyecto_final.zip"
❌ "proyecto_final_v2.zip"
❌ "proyecto_final_AHORA_SI.zip"
❌ "proyecto_final_ESTE_ES_EL_BUENO.zip"

Soluciones con Git:

- ✅ Historial completo de cambios
- ✅ Saber quién cambió qué y cuándo
- ✅ Volver a versiones anteriores fácilmente
- ✅ Trabajar en equipo sin conflictos
- ✅ Experimentar sin miedo a romper cosas
- ✅ Mantener respaldo en la nube

🛠️ Instalando Git

Paso 1: Descargar

🌐 Visita: git-scm.com/download

Descarga para tu sistema operativo:

- 🍎 Mac
- 🪟 Windows
- 🐧 Linux

Paso 2: Instalar

Windows: Ejecuta el instalador

- ✅ Selecciona **Visual Studio Code** como editor
- ✅ Deja las demás opciones por defecto

Mac: Puede venir preinstalado

- Verifica con: `git --version`

🛠️ Configurando Git

Paso 3: Identificarte

Git necesita saber quién eres para registrar tus cambios.

```
# Configurar tu nombre
git config --global user.name "Tu Nombre"

# Configurar tu email
git config --global user.email "tu@email.com"

# Verificar configuración
git config --global --list
```

Salida:

```
user.name=Tu Nombre
user.email=tu@email.com
```

💡 **Importante:** Esta información aparece en cada commit que hagas

📦 Conceptos Clave de Git

1. 📦 **Repositorio (Repo)**
    - Carpeta especial donde Git guarda el historial
    - Como un "archivador organizado" de tu proyecto
2. 📸 **Commit**
    - Una "foto" del estado de tu proyecto en un momento
    - Incluye un mensaje describiendo los cambios
3. 🌿 **Branch (Rama)**
    - Una línea de desarrollo independiente
    - Por defecto existe la rama `main` o `master`

📦 Creando tu Primer Repositorio

git init - Inicializar repositorio

```
# Paso 1: Crear carpeta del proyecto
mkdir mi-proyecto
cd mi-proyecto
# Paso 2: Inicializar Git
git init
# Salida:
Initialized empty Git repository in /ruta/mi-proyecto/.git/
```

¿Qué pasó?

- ✅ Se creó una carpeta oculta `.git`
- ✅ Git ahora puede rastrear cambios en esta carpeta
- ✅ Tu proyecto es ahora un repositorio Git

Ver qué está pasando

¿Cuándo lo necesito?

Todo el tiempo. Es el comando más usado.

```
git status
```

Salida:

```
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html
        style.css

nothing added to commit but untracked files present
```

Te muestra:

- 📍 En qué rama estás
- 📄 Archivos nuevos (untracked)

➕ git add - Preparar Cambios

Staging Area (Área de Preparación) Antes de hacer un commit, debes decirle a Git qué archivos incluir.

```
# Agregar un archivo específico
git add index.html
# Agregar varios archivos
git add index.html style.css
# Agregar todos los archivos
git add .
# Verificar
git status
```

Analogía:

Es como poner cosas en una caja antes de enviarla por correo 📦

💾 git commit - Guardar Cambios

Crear un punto en la historia

¿Cuándo lo necesito?

Cuando quieres guardar permanentemente un conjunto de cambios.

```
# Hacer commit con mensaje
git commit -m "Agregué la página principal"

# Salida:
[main a3f5k2] Agregué la página principal
 2 files changed, 45 insertions(+)
 create mode 100644 index.html
 create mode 100644 style.css
```

Buenas prácticas para mensajes:

- ✅ "Agregué formulario de contacto"
- ✅ "Corregí error en el menú"
- ❌ "cambios"
- ❌ "asdf"

💻 Ejemplo: Primer Commit Completo

```
# Paso 1: Crear proyecto
mkdir mi-sitio
cd mi-sitio
# Paso 2: Inicializar Git
git init
# Paso 3: Crear archivos
touch index.html style.css
# Paso 4: Ver estado
git status
# Untracked files: index.html, style.css
# Paso 5: Agregar archivos
git add .
# Paso 6: Hacer commit
git commit -m "Primer commit: estructura inicial"
```

✅ ¡Tu primer commit está listo!

📜 git log - Ver Historial

Ver todos los commits

```
git log
```

Salida:

```
commit a3f5k29j3n4l5m6n7o8p9q0r1s2t3u4v (HEAD -> main)
Author: Tu Nombre <tu@email.com>
Date:   Tue Oct 21 19:30:00 2025 -0300

    Agregué formulario de contacto

commit b2e4h18i2m3k4l5m6n7o8p9q0r1s2t3u
Author: Tu Nombre <tu@email.com>
Date:   Tue Oct 21 18:15:00 2025 -0300

    Primer commit: estructura inicial
```

❓ Pregunta Rápida

¿En qué orden van estos comandos?

A) `git add .` → `git commit -m "..."` → `git init`

B) `git init` → `git add .` → `git commit -m "..."`

C) `git commit -m "..."` → `git add .` → `git init`

D) `git init` → `git commit -m "..."` → `git add .`

Piensa unos segundos... 🤔

✅ Respuesta

Respuesta correcta: B)

```
# 1. Primero: Inicializar el repositorio
git init

# 2. Segundo: Preparar los archivos
git add .

# 3. Tercero: Guardar los cambios
git commit -m "Mi mensaje"
```

¿Por qué?

1. **git init** → Crea el repositorio
2. **git add** → Prepara qué cambios incluir
3. **git commit** → Guarda los cambios permanentemente

Analogía: Primero creas la caja → Pones cosas dentro → Sellas la caja

¡Genial si acertaste! 🎉

Tu respaldo en la nube

🌐 ¿Qué es GitHub?

¿Por qué necesito esto?

Git = Herramienta en tu computadora

GitHub = Servicio en la nube para alojar repositorios Git

¿Para qué sirve GitHub?

- ☁️ Respaldo de tu código en la nube
- 👥 Colaboración con otros desarrolladores
- 🌍 Compartir proyectos con el mundo
- 💼 Portafolio profesional
- 📱 Acceso desde cualquier lugar

Otros servicios similares:

- GitLab
- BitBucket
- Gitea

🌐 Creando Cuenta en GitHub

Paso 1: Registro

🌐 Visita: github.com

Completa:

- 👤 Nombre de usuario (será tu URL: github.com/tunombre)
- 📧 Email
- 🔒 Contraseña

Paso 2: Verificación

- ✅ Verifica tu email
- ✅ Completa el tutorial inicial (opcional)

💡 **Tip:** Elige un nombre de usuario profesional, lo verán futuros empleadores

➕ Creando un Repositorio en GitHub

Paso 1: Nuevo Repositorio

1. Clic en el botón + (esquina superior derecha)
2. Selecciona "**New repository**"

Paso 2: Configuración

- **Repository name:** mi-primer-proyecto
- **Description:** Mi primer proyecto con Git
- **Public/Private:** Public (para que sea visible)
- ❌ **NO** marcar "Initialize with README"

Paso 3: Crear

Clic en "**Create repository**"

🔗 Conectar Local con GitHub

Enlazar tu proyecto local con GitHub

Después de crear el repositorio en GitHub, verás instrucciones:

```
# Conectar con el repositorio remoto
git remote add origin https://github.com/tu-usuario/mi-proyecto.git

# Renombrar rama a main (si es necesario)
git branch -M main

# Enviar cambios a GitHub
git push -u origin main
```

¿Qué hace cada comando?

- **git remote add origin** → Conecta con GitHub
- **git branch -M main** → Asegura que la rama se llama "main"
- **git push** → Envía tus commits a GitHub

📤 git push - Enviar Cambios

Subir commits a GitHub

¿Cuándo lo necesito?

Cuando quieres que tus cambios locales se reflejen en GitHub.

```
# Primera vez (configurando upstream)
git push -u origin main

# Siguientes veces
git push
```

¿Qué pasa?

- 📤 Tus commits se copian a GitHub
- ☁️ Quedan respaldados en la nube
- 🌍 Otros pueden verlos

💡 **Flujo típico:**

Cambios → `git add` → `git commit` → `git push`

📥 git pull - Traer Cambios

Descargar cambios desde GitHub

¿Cuándo lo necesito?

- Cuando trabajas desde otra computadora
- Cuando tu equipo hizo cambios
- Para mantener tu copia local actualizada

```
git pull
```

¿Qué pasa?

- 📥 Descarga commits desde GitHub
- 🔄 Actualiza tu proyecto local
- ✅ Sincroniza todo

💡 **Buena práctica:** Siempre haz `git pull` antes de empezar a trabajar

💻 Ejemplo: Flujo Completo Local → GitHub

```
# Paso 1: Crear proyecto local
mkdir mi-proyecto
cd mi-proyecto
git init
# Paso 2: Crear archivos
touch index.html
echo "<h1>Hola Mundo</h1>" > index.html
# Paso 3: Primer commit
git add .
git commit -m "Primer commit"
# Paso 4: Conectar con GitHub (reemplaza con tu URL)
git remote add origin https://github.com/tu-usuario/mi-proyecto.git
# Paso 5: Enviar a GitHub
git branch -M main
git push -u origin main
```

✅ ¡Tu proyecto está en GitHub!

🧑‍💻 Ejercicio: Tu Primer Proyecto en GitHub

¡Hora de practicar!

Crea un proyecto completo y súbelo a GitHub:

1. Crea un repositorio local llamado `hola-mundo`
2. Crea un archivo `index.html` con un mensaje
3. Haz un commit
4. Crea un repositorio en GitHub
5. Conecta y sube tu proyecto

Pistas:

- `git init` para iniciar
- `git add .` para preparar
- `git commit -m "..."` para guardar
- `git push` para subir

💻 Git en Visual Studio Code

La forma más fácil de usar Git

💻 Git en VS Code

¿Por qué usar VS Code para Git?

Ventajas:

- 👁️ Interfaz visual e intuitiva
- 🎨 Ver cambios con colores
- ✅ Hacer commits con clics
- 🔀 Manejar branches visualmente
- 📊 Ver historial gráfico

¿Significa que no usaremos la terminal?

No. Aprenderemos ambas formas:

- Terminal → Más control y velocidad
- VS Code → Más visual y fácil

🔍 Icono de Control de Código Fuente

El panel de Git en VS Code

Ubicación:

Barra lateral izquierda → Icono de rama (🔀)

¿Qué muestra?

- 📄 Archivos modificados
- ➕ Archivos nuevos
- 🗑️ Archivos eliminados
- ✅ Archivos en staging

Números en el icono:

- Indica cuántos archivos tienen cambios

➕ Inicializar Repositorio en VS Code

Paso 1: Abrir carpeta

`File → Open Folder` → Selecciona tu proyecto

Paso 2: Inicializar Git

1. Clic en el icono de Control de Código Fuente
2. Clic en "**Initialize Repository**"

¿Qué pasó?

- ✅ Se ejecutó `git init` automáticamente
- ✅ Se creó la carpeta `.git`
- ✅ VS Code ahora rastrea cambios

Equivalente en terminal:

```
git init
```

📝 Ver Cambios en VS Code

Colores en los archivos:

- 🟢 **Verde (U)** → Archivo nuevo (Untracked)
- 🟡 **Amarillo (M)** → Archivo modificado (Modified)
- 🔴 **Rojo (D)** → Archivo eliminado (Deleted)

Ver diferencias:

Clic en un archivo → Ver lado a lado:

- ◀️ Izquierda: Versión anterior
- ▶️ Derecha: Versión actual
- 🟢 Verde: Líneas agregadas
- 🔴 Rojo: Líneas eliminadas

💡 **Tip:** Esto te ayuda a revisar antes de hacer commit

➕ Staging en VS Code

Preparar archivos para commit

Método 1: Archivo individual

- Pasa el mouse sobre el archivo
- Clic en el + que aparece

Método 2: Todos los archivos

- Clic en el + junto a "Changes"

Deshacer staging:

- Clic en el - junto al archivo

Equivalente en terminal:

```
git add index.html          # Archivo individual
git add .                   # Todos los archivos
```

💾 Commit en VS Code

Guardar cambios

Paso 1: Preparar archivos

- Agrega archivos al staging (clic en +)

Paso 2: Escribir mensaje

- En el campo de texto arriba, escribe el mensaje
- Ejemplo: "Agregué página de contacto"

Paso 3: Hacer commit

- Clic en el ✓ (checkmark)
- O presiona `Ctrl + Enter` (Windows) / `Cmd + Enter` (Mac)

Equivalente en terminal:

```
git add .
git commit -m "Agregué página de contacto"
```

🌐 Conectar con GitHub desde VS Code

Método 1: Con extensión GitHub

1. Instala la extensión "GitHub Pull Requests and Issues"
2. Clic en "Publish to GitHub" en el panel de Git
3. Selecciona si quieres repositorio público o privado
4. ✅ ¡Listo! Se crea automáticamente

Método 2: Manual

1. Crea el repositorio en GitHub
2. Copia la URL
3. En VS Code, abre la terminal integrada
4. Ejecuta:

```
git remote add origin URL_DE_TU_REPO
git push -u origin main
```

Push y Pull en VS Code

Sincronizar con GitHub

Enviar cambios (Push):

- Clic en los **...** (menú) en el panel Git
- Selecciona "**Push**"
- O clic en el icono de sincronización (↻) en la barra inferior

Traer cambios (Pull):

- Clic en los **...** (menú)
- Selecciona "**Pull**"

Sincronización automática:
VS Code muestra indicadores cuando:

- ↑ Tienes commits para enviar
- ↓ Hay commits para descargar

💻 Ejemplo: Flujo Completo en VS Code

Paso a paso:

1. **Abrir carpeta en VS Code**
2. **Inicializar repositorio** (icono Git → Initialize)
3. **Crear archivos** (index.html, style.css)
4. **Ver cambios** (archivos en verde)
5. **Stage todos** (clic en + general)
6. **Commit** (mensaje + checkmark)
7. **Publish to GitHub** (botón azul)
8. **Seleccionar público/privado**

✅ **¡Proyecto en GitHub en menos de 2 minutos!**

💪 Ejercicio: Proyecto con VS Code

¡Hora de practicar!

Crea un mini-portafolio y súbelo a GitHub usando VS Code:

1. Crea carpeta `mi-portafolio`
2. Abre en VS Code
3. Inicializa Git desde VS Code
4. Crea `index.html` con tu nombre
5. Crea `style.css` con estilos básicos
6. Haz commit desde VS Code
7. Publica en GitHub desde VS Code

¡Todo desde la interfaz de VS Code!

⏱️ **Tiempo:** 10 minutos

❓ Pregunta Rápida

En VS Code, ves un archivo con 🟡 (M). ¿Qué significa?

A) Es un archivo nuevo que nunca has commiteado

B) Es un archivo que modificaste desde el último commit

C) Es un archivo que está en staging

D) Es un archivo que Git está ignorando

Piensa unos segundos... 🤔

✅ Respuesta

Respuesta correcta: B)

¿Por qué?

- 🟡 **(M)** = Modified (Modificado)
- El archivo existía en el commit anterior
- Pero ahora tiene cambios nuevos
- Aún no está en staging (necesita el +)

Otros códigos:

- 🟢 **(U)** = Untracked (Nuevo)
- 🟢 **(A)** = Added (En staging)
- 🔴 **(D)** = Deleted (Eliminado)

¡Perfecto si acertaste! 🎉

GitHub Pages

Publica tu sitio web gratis

📄 ¿Qué es GitHub Pages?

¿Por qué necesito esto?

GitHub Pages te permite alojar sitios web estáticos **gratis** directamente desde tus repositorios de GitHub.

¿Qué puedo publicar?

- 🌐 Portafolios personales
- 📝 Blogs
- 📄 Documentación de proyectos
- 🎨 Sitios de muestra
- 🚀 Landing pages

Ventajas:

- ✅ Completamente gratis
- ✅ Dominio incluido (tunombre.github.io)
- ✅ HTTPS automático
- ✅ Actualización automática al hacer push
- ✅ Súper fácil de configurar

🌟 ¿Por qué GitHub Pages?

Es la forma más simple de publicar un sitio web

Ventajas:

- 🎯 **Cero configuración** - Solo activas y funciona
- ⚡ **Súper rápido** - Deploy en segundos
- 🌍 **URL automática** - username.github.io/proyecto
- 🔒 **Seguro por defecto** - HTTPS incluido
- 📱 **CDN global** - Rápido en todo el mundo
- 💰 **100% gratis** - Sin límites

Ideal para:

- Tu primer sitio web
- Portafolio profesional
- Proyectos del curso
- Documentación de código

🚀 Activar GitHub Pages

Método 1: Desde el repositorio

1. Ve a tu repositorio en GitHub
2. Clic en **Settings** (⚙️)
3. En el menú lateral: **Pages**
4. En **Source**, selecciona:
   - Branch: `main`
   - Folder: `/ (root)`
5. Clic en **Save**
6. ✅ ¡Listo! Tu sitio estará en `https://tunombre.github.io/repo`

Tarda ~1 minuto en estar disponible 🚀

Estructura de Proyecto para GitHub Pages

¿Cómo debe ser tu proyecto?

```
mi-sitio-web/
├── index.html          ← Tu página principal (obligatorio)
├── style.css           ← Tus estilos
├── script.js           ← Tu JavaScript
└── images/             ← Tus imágenes
    └── logo.png
```

Reglas importantes:

- ✅ Tu página principal DEBE llamarse `index.html`
- ✅ Debe estar en la raíz del repositorio
- ✅ Usa rutas relativas para CSS/JS/imágenes
- ✅ No necesitas archivos de configuración especiales

¡Es así de simple! No necesitas `.gitlab-ci.yml` ni nada extra

💻 Ejemplo: Publicar Sitio en GitHub Pages

Paso a paso completo:

Paso 1: Crea tu proyecto local

```
mkdir mi-sitio
cd mi-sitio
git init
```

Paso 2: Crea tu HTML

```
touch index.html
```

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Mi Primer Sitio</title>
</head>
<body>
    <h1>¡Hola Mundo desde GitHub Pages!</h1>
    <p>Mi primer sitio publicado 🚀</p>
</body>
</html>
```

💻 Ejemplo: Publicar Sitio en GitHub Pages (2)

Paso 3: Commit local

```
git add .
git commit -m "Sitio inicial"
```

Paso 4: Crear repo en GitHub

- Ve a GitHub → New repository
- Nombre: `mi-sitio`
- Public
- NO marcar "Initialize with README"
- Create repository

💻 Ejemplo: Publicar Sitio en GitHub Pages (3)

Paso 5: Conectar y push

```
# Reemplaza con tu URL de GitHub
git remote add origin https://github.com/tu-usuario/mi-sitio.git
git branch -M main
git push -u origin main
```

Paso 6: Activar GitHub Pages

1. En tu repo → **Settings**
2. **Pages** (menú lateral)
3. Source: `main` branch, `/ (root)`
4. **Save**

Paso 7: Esperar ~1 minuto

✅ ¡Tu sitio estará en `https://tu-usuario.github.io/mi-sitio`!

🔍 Verificar Despliegue en GitHub Pages

¿Cómo sé si funcionó?

Método 1: Mensaje de éxito

- Después de activar Pages, verás:

  > "Your site is published at https://tunombre.github.io/proyecto"

Método 2: Visit site

- En **Settings** → **Pages** aparece un botón **"Visit site"**
- Clic ahí y verás tu sitio

Método 3: Verificación manual

- Espera 1-2 minutos
- Visita: `https://tunombre.github.io/nombre-repo`
- ¡Verás tu sitio publicado! 🎉

💡 **Tip:** Si ves 404, espera un minuto más y recarga

🔄 Actualizar tu Sitio en GitHub Pages

¿Cómo hago cambios?

Proceso súper simple:

1. **Edita archivos localmente** (HTML, CSS, JS)
2. **Guarda cambios**
3. **Commit:**

```
git add .
git commit -m "Actualicé el diseño"
```

4. **Push:**

```
git push
```

5. **Espera 30-60 segundos**
6. **Refresca tu sitio** → ¡Cambios visibles!

Es automático: Cada push actualiza tu sitio automáticamente 🚀

¡Mucho más rápido que GitLab Pages!

📋 Checklist: GitHub Pages

Verifica estos puntos:

- ✅ Tienes cuenta en GitHub
- ✅ Creaste un repositorio público
- ✅ Tu `index.html` está en la raíz
- ✅ Hiciste commit de todos los archivos
- ✅ Hiciste push a GitHub
- ✅ Activaste GitHub Pages en Settings
- ✅ Seleccionaste branch `main` y folder `/ (root)`
- ✅ Esperaste 1-2 minutos
- ✅ Ves tu sitio en `tunombre.github.io/repo`

💪 Ejercicio: Publica tu Portafolio

Crea y publica un portafolio personal en GitHub Pages:

1. Crea un proyecto local `mi-portafolio`
2. Crea `index.html` con:
    - Tu nombre
    - Una breve descripción
    - Tres secciones: Sobre mí, Habilidades, Contacto
3. Agrega estilos CSS básicos
4. Crea un `README.md` describiendo tu sitio
5. Sube todo a GitHub
6. Activa GitHub Pages
7. Verifica que se publique correctamente

⏱️ **Tiempo:** 15 minutos

🎯 **Comparte tu URL:** Comparte el enlace de tu sitio publicado

Resumen y Recursos

📝 Conceptos Clave de Hoy

Lo más importante:

✅ **Terminal:** Interfaz de texto para controlar tu computadora

✅ **Comandos esenciales:** `pwd`, `ls`, `cd`, `mkdir`, `touch`, `rm`

✅ **Git:** Sistema de control de versiones

✅ **Flujo Git:** `init` → `add` → `commit` → `push`

✅ **GitHub/GitLab:** Plataformas para alojar repositorios

✅ **VS Code:** Interfaz visual para usar Git

✅ **GitHub Pages:** Publicar sitios web gratis

🎯 Lo Más Importante

Recuerda:

- 💡 **La terminal es tu amiga** - Con práctica se vuelve natural
- 💡 **Git te protege** - Siempre puedes volver atrás
- 💡 **Commit frecuente** - Guarda cambios cada vez que completes algo
- 💡 **Mensajes claros** - Tu yo del futuro te lo agradecerá
- 💡 **Push regularmente** - Tu código estará respaldado
- 💡 **No tengas miedo** - No puedes romper nada permanentemente

📋 Comandos Terminal - Cheat Sheet

Navegación:

```
pwd           # Ver ubicación actual
ls            # Listar archivos
cd carpeta    # Entrar a carpeta
cd ..         # Subir un nivel
cd ~          # Ir a carpeta usuario
```

Archivos y carpetas:

```
mkdir nombre    # Crear carpeta
touch archivo   # Crear archivo
rm archivo      # Eliminar archivo
rm -rf carpeta  # Eliminar carpeta
mv origen dest  # Mover/renombrar
cp origen dest  # Copiar
```

📋 Comandos Git - Cheat Sheet

Configuración inicial:

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

Comandos básicos:

```
git init                    # Iniciar repositorio
git status                  # Ver estado
git add .                   # Preparar todos los archivos
git commit -m "mensaje"     # Guardar cambios
git log                     # Ver historial
```

Trabajar con GitHub/GitLab:

```
git remote add origin URL   # Conectar con remoto
git push -u origin main     # Enviar cambios
git pull                    # Traer cambios
```

📚 Recursos para Seguir Aprendiendo

Documentación y herramientas:

- 🎮 Terminus MIT - ¡Juego para aprender terminal!
- 📖 Git Book Oficial - Guía completa en español
- 📚 GitHub Learning Lab - Tutoriales interactivos
- 🎓 Learn Git Branching - Visualización interactiva
- 📄 GitHub Pages Docs - Documentación oficial
- 💻 Oh My Git! - Juego para aprender Git

🚀 Próximos Pasos

¿Qué viene después?

En las próximas clases veremos:

- 🌿 **Branches en Git** - Trabajar en múltiples versiones
- 🔀 **Merge y conflictos** - Fusionar cambios
- 👥 **Colaboración** - Trabajar en equipo
- 🔄 **Pull Requests** - Contribuir a proyectos
- 🏷️ **Tags y releases** - Versionar proyectos
- 🚀 **Deploy automático** - CI/CD avanzado

¡Git es la base para todo esto!

💪 Práctica para Casa

Proyecto sugerido: Tu Portafolio Web

Crea un sitio personal y publícalo:

1. 📝 **Página sobre ti** con tu foto y descripción
2. 💼 **Proyectos** que has hecho (aunque sean del curso)
3. 🛠️ **Habilidades** que estás aprendiendo
4. 📧 **Contacto** (email, LinkedIn, GitHub)
5. 🎨 **Estilo propio** con CSS

Objetivo:

- ✅ Practicar Git (mínimo 5 commits)
- ✅ Publicar en GitLab Pages
- ✅ Compartir URL en el foro del curso

💡 Consejos Finales

Buenas prácticas:

✅ **Commit frecuente** - Mejor muchos commits pequeños que uno grande

✅ **Mensajes descriptivos** - Explica QUÉ y POR QUÉ cambiaste

✅ **Pull antes de push** - Actualiza antes de subir cambios

✅ **No subas contraseñas** - Nunca hagas commit de información sensible

✅ **Usa .gitignore** - Ignora archivos que no deben estar en Git

✅ **Practica diariamente** - 10 minutos al día es mejor que 2 horas una vez

✅ **No tengas miedo** - Git está diseñado para protegerte

¿Qué archivos NO deberías subir a Git?

Crea un archivo `.gitignore` en la raíz de tu proyecto:

```
# Dependencias
node_modules/
venv/

# Archivos del sistema
.DS_Store
Thumbs.db

# Configuración local
.env
config.local.js

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


¿Qué hace?

❓ Pregunta Rápida Final

¿Cuál es el flujo correcto de trabajo?

A) Cambios → `commit` → `add` → `push`

B) Cambios → `push` → `add` → `commit`

C) Cambios → `add` → `commit` → `push`

D) `push` → Cambios → `add` → `commit`

Piensa unos segundos... 🤔

Respuesta correcta: C)

```
# 1. Hacer cambios en los archivos
# (editas en VS Code)

# 2. Preparar cambios
git add .

# 3. Guardar cambios localmente
git commit -m "Descripción de cambios"

# 4. Enviar a GitHub/GitLab
git push
```

Analogía del correo:

1. 📝 Escribir carta (Cambios)
2. 📦 Poner en sobre (add)
3. ✉️ Sellar sobre (commit)
4. 📮 Echar al buzón (push)

🎉 ¡Felicidades!

Ya sabes usar la terminal y Git

Ahora puedes trabajar como un desarrollador profesional 🚀

? Preguntas

¿Alguna duda sobre la terminal, Git o GitLab Pages?

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase!

No olvides completar todos los ejercicios 📝

Y publicar tu portafolio en GitHub Pages 🌐