Tabla de contenidos

1. Fundamentos de Git

1. ¿Qué es el Control de Versiones?

Definición y Propósito

El control de versiones es un sistema que registra los cambios realizados en archivos a lo largo del tiempo, permitiendo recuperar versiones específicas más adelante.

```
# PROBLEMA SIN CONTROL DE VERSIONES
proyecto_final.html
proyecto_final_v2.html
proyecto_final_v2_corregido.html
proyecto_final_DEFINITIVO.html
proyecto_final_DEFINITIVO_2.html
proyecto_final_AHORA_SI_FINAL.html
```

Beneficios del Control de Versiones

- ✅ **Historial completo de cambios:** Quién cambió qué y cuándo.
- ✅ **Colaboración eficiente:** Múltiples personas trabajando juntas.
- ✅ **Respaldo automático:** Nunca perder trabajo.
- ✅ **Experimentación segura:** Crear ramas para probar funcionalidades.
- ✅ **Trazabilidad:** Identificar cuándo se introdujo un error.

2. ¿Qué es Git?

Git es un sistema de control de versiones distribuido, creado por Linus Torvalds en 2005. Es rápido, eficiente y el estándar de la industria.

Git vs GitHub vs GitLab

- 🔧 **Git:** Es el software que instalas en tu computadora.
- 🌐 **GitHub:** Es una plataforma online que usa Git para alojar repositorios.
- 🏢 **GitLab:** Es una alternativa a GitHub, a menudo usada en empresas.

3. Instalación y Configuración

Para usar Git, primero debes instalarlo y configurarlo.

Verificar Instalación

```
git --version
# Salida esperada: git version 2.x.x
```

Configuración Inicial (Obligatoria)

```
# Configurar nombre de usuario
git config --global user.name "Tu Nombre Completo"

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

# Verificar configuración
git config --list
```

4. Conceptos Fundamentales

- **Repositorio (Repository):** Un directorio que contiene tu proyecto y todo el historial de cambios.
- **Commit:** Una "instantánea" de tu proyecto en un momento específico.
- **Branch (Rama):** Una línea independiente de desarrollo para trabajar en diferentes funcionalidades.

5. Flujo Básico de Trabajo

El flujo de trabajo típico en Git sigue estos pasos:

1. **Modificar archivos:** Haces cambios en tu proyecto.
2. **Preparar cambios (Staging):** Usas `git add` para seleccionar qué cambios incluirás en el próximo commit.
3. **Confirmar cambios (Commit):** Usas `git commit` para guardar los cambios en el historial.

Comandos Esenciales

```
# 1. Crear un nuevo repositorio
git init

# 2. Clonar un repositorio existente
git clone [URL_DEL_REPOSITORIO]

# 3. Verificar estado actual
git status

# 4. Preparar cambios
git add .       # Agregar todos los archivos
git add [archivo] # Agregar un archivo específico

# 5. Confirmar cambios
git commit -m "Mensaje descriptivo del commit"

# 6. Ver historial
git log
```

6. Resumen del Capítulo

Conceptos Vistos

- ✅ **Control de Versiones:** Sistema para registrar cambios.
- ✅ **Git:** Sistema distribuido, rápido y eficiente.

- ✅ **Repositorio:** Directorio con tu proyecto e historial.
- ✅ **Commit:** Instantánea de tu proyecto.
- ✅ **Branch:** Línea de desarrollo independiente.

2. Comandos Básicos Git

1. Comandos de Información

git status - Estado del Repositorio

El comando `git status` es tu mejor amigo para entender qué está pasando en tu repositorio.

```
# Ver estado completo
git status

# Ver estado resumido
git status -s
```

git log - Historial de Commits

El comando `git log` muestra el historial de commits del repositorio.

```
# Log básico (completo)
git log

# Log resumido en una línea por commit
git log --oneline

# Log con gráfico de branches
git log --graph --oneline --all
```

git diff - Ver Diferencias

El comando `git diff` muestra las diferencias entre diferentes estados del repositorio.

```
# Diferencias en working directory (no staged)
git diff

# Diferencias en staging area
git diff --staged

# Diferencias entre dos commits
git diff commit1 commit2
```

2. Comandos de Manipulación

git add - Preparar Archivos

El comando `git add` prepara archivos para el próximo commit.

```
# Agregar archivo específico
git add index.html

# Agregar directorio completo
git add css/

# Agregar todo (archivos nuevos y modificados)
git add .
```

git commit - Confirmar Cambios

El comando `git commit` confirma los cambios preparados en el staging area.

```
# Commit básico con mensaje
git commit -m "Agregar página de contacto"

# Modificar el último commit
git commit --amend -m "Nuevo mensaje corregido"
```

3. Comandos de Deshacer Cambios

git checkout - Cambiar Estado de Archivos

El comando `git checkout` puede deshacer cambios en archivos del working directory.

```
# Deshacer cambios en un archivo específico
git checkout -- archivo.txt

# ⚠️ CUIDADO: Estos cambios son irreversibles
```

git reset - Deshacer Commits y Staging

El comando `git reset` puede deshacer cambios en diferentes niveles.

```
# Quitar archivo específico del staging
git reset archivo.txt

# Deshacer el último commit (manteniendo los cambios)
git reset --soft HEAD~1
```

4. Archivo .gitignore

El archivo `.gitignore` especifica qué archivos Git debe ignorar y no rastrear.

```
# Ignorar archivos específicos
config.txt
debug.log

# Ignorar por patrones
*.log
*.tmp

# Ignorar directorios
node_modules/
dist/
```

5. Resumen del Capítulo

Comandos Dominados

- ✅ **git status**: Ver estado actual del repositorio.
- ✅ **git log**: Historial de commits.
- ✅ **git diff**: Ver diferencias entre estados.
- ✅ **git add**: Preparar archivos para commit.
- ✅ **git commit**: Confirmar cambios.
- ✅ **git checkout**: Deshacer cambios en working directory.
- ✅ **git reset**: Deshacer cambios en staging y commits.

3. Branches y Merging

1. ¿Qué son las Branches?

Una branch (rama) es una línea independiente de desarrollo que permite trabajar en diferentes funcionalidades sin afectar la rama principal (generalmente llamada `main` o `master`).

Ventajas de Usar Branches

- ✅ **Desarrollo Paralelo:** Trabajar en múltiples funcionalidades a la vez.
- ✅ **Aislamiento:** Los cambios en una rama no afectan a otras.
- ✅ **Colaboración:** Varios desarrolladores pueden trabajar en sus propias ramas.
- ✅ **Experimentación Segura:** Probar ideas sin riesgo de romper el código principal.

2. Creación y Cambio de Ramas

git branch - Gestión de Ramas

```
# Ver ramas locales
git branch

# Crear nueva rama
git branch nueva-funcionalidad

# Eliminar rama
git branch -d rama-a-eliminar
```

git checkout - Cambiar de Rama

```
# Cambiar a rama existente
git checkout nueva-funcionalidad

# Crear y cambiar en un solo comando
git checkout -b otra-funcionalidad
```

3. Merge - Fusionar Ramas

El comando `git merge` fusiona los cambios de una rama en otra.

Flujo de Trabajo Típico

1. **Crear una rama:** `git checkout -b feature-login`
2. **Trabajar y hacer commits:** `git commit -m "feat: add login form"`
3. **Cambiar a la rama principal:** `git checkout main`
4. **Fusionar la rama de funcionalidad:** `git merge feature-login`
5. **Eliminar la rama (opcional):** `git branch -d feature-login`

4. Resolución de Conflictos (Merge Conflicts)

Los conflictos ocurren cuando Git no puede fusionar automáticamente los cambios porque las mismas líneas fueron modificadas en ambas ramas.

Pasos para Resolver Conflictos

1. **Identificar archivos con conflicto:** Usa `git status`.
2. **Editar los archivos manualmente:** Busca los marcadores `<<<<<<<`, `=======`, `>>>>>>>` y decide qué código mantener.
3. **Marcar como resuelto:** Usa `git add [archivo-resuelto]`.
4. **Completar el merge:** Usa `git commit`.

5. Tags - Etiquetado de Versiones

Los tags son marcadores que señalan commits específicos, generalmente para marcar releases o versiones importantes.

Comandos de Tags

```
# Crear tag anotado (recomendado)
git tag -a v1.0 -m "Versión 1.0 - Lanzamiento inicial"

# Listar todos los tags
git tag
```

6. Resumen del Capítulo

Comandos Vistos

- ✅ **git branch**: Crear, listar y eliminar ramas.
- ✅ **git checkout**: Cambiar entre ramas.
- ✅ **git merge**: Fusionar cambios entre ramas.
- ✅ **Resolución de conflictos**: Proceso manual para resolver conflictos de merge.
- ✅ **git tag**: Etiquetar versiones importantes del proyecto.

4. GitHub

1. Repositorios Remotos - Introducción a GitHub

¿Qué es GitHub?

GitHub es una plataforma de alojamiento de código basada en Git que permite el almacenamiento en la nube, la colaboración, la gestión de proyectos y la automatización de flujos de trabajo.

Configurar GitHub

Es esencial configurar tu identidad en Git para que coincida con tu cuenta de GitHub y configurar la autenticación a través de HTTPS con un Token de Acceso Personal o mediante Claves SSH.

Crear Repositorio en GitHub

Puedes crear un repositorio directamente en GitHub y luego clonarlo, o crear un proyecto localmente y luego conectarlo a un nuevo repositorio remoto en GitHub.

2. Push y Pull - Sincronización Remota

Git Push - Subir Cambios

El comando `git push` se utiliza para subir tus commits locales al repositorio remoto.

```
# Primer push (configurar upstream)
git push -u origin main

# Push posteriores
git push
```

Git Pull - Descargar Cambios

El comando `git pull` se utiliza para descargar y fusionar los cambios desde el repositorio remoto a tu repositorio local.

```
# Descargar y fusionar cambios
git pull

# Descargar cambios sin fusionar (más control)
git fetch origin
git merge origin/main
```

3. Clonación de Repositorios

Git Clone - Descargar Proyectos

El comando `git clone` se utiliza para crear una copia local de un repositorio remoto existente.

```
# Clonar con HTTPS
git clone https://github.com/usuario/proyecto.git

# Clonar con SSH
git clone git@github.com:usuario/proyecto.git
```

4. Fork y Pull Requests

¿Qué es un Fork?

Un fork es una copia personal de un repositorio que te permite experimentar con cambios sin afectar el proyecto original. Es fundamental para contribuir a proyectos de código abierto.

Pull Requests (PRs)

Un Pull Request es una solicitud para que tus cambios sean fusionados en el repositorio original. Es el mecanismo principal de colaboración y revisión de código en GitHub.

5. Workflows Colaborativos Profesionales

GitHub Flow

Un flujo de trabajo simple y efectivo donde la rama `main` siempre está lista para ser desplegada. Las nuevas funcionalidades se desarrollan en ramas separadas y se fusionan a través de Pull Requests.

Git Flow

Un flujo de trabajo más estructurado con ramas dedicadas para `main` (producción), `develop` (integración), features, releases y hotfixes. Ideal para proyectos grandes y complejos.

6. Resumen del Capítulo

Conceptos Dominados

- ✅ Configuración de repositorios remotos y autenticación en GitHub.
- ✅ Sincronización con `git push` y `git pull`.
- ✅ Colaboración mediante `git clone`, forks y Pull Requests.
- ✅ Comprensión de workflows colaborativos como GitHub Flow y Git Flow.