

Task 1: Instalación y Configuración Inicial de Git (15 minutos)

La instalación y configuración de Git representa el **primer paso crítico** para establecer un entorno de desarrollo profesional. Más allá de simplemente "instalar software", esta configuración establece las bases para colaboración efectiva y trazabilidad completa del trabajo.

La Importancia de una Instalación Correcta

Git no es solo una herramienta, sino una **infraestructura fundamental** para el desarrollo moderno. Una instalación incorrecta puede crear problemas sutiles que se manifiestan como **errores intermitentes, problemas de encoding, o dificultades de colaboración** que afectan la productividad del equipo.

Versiones compatibles: Git evoluciona constantemente, y usar una versión demasiado antigua puede limitar funcionalidades avanzadas como git worktree, git switch, o mejoras en performance. Una versión actual asegura acceso a **características modernas** que facilitan el trabajo diario.

Configuración de línea de comandos: Git está diseñado para trabajar desde la terminal, lo que puede ser intimidante inicialmente pero ofrece **control preciso y automatización poderosa**. Aprender estos comandos establece un **foundation sólido** para herramientas más avanzadas.

Integración con el sistema operativo: Una instalación correcta asegura que Git se integre perfectamente con el explorador de archivos, editores

de código, y otras herramientas de desarrollo, creando un **ecosistema coherente**.

Configuración Esencial: Identidad y Preferencias

La configuración inicial de Git va más allá de datos personales; establece **preferencias fundamentales** que afectan cómo Git se comporta en tu entorno.

Identidad del autor: Los campos `user.name` y `user.email` no son solo metadata, sino **identificadores críticos** para la autoría de commits. En entornos profesionales, estos datos sirven para **auditoría, atribución de trabajo, y responsabilidad técnica**. Un email incorrecto puede crear problemas de **verificación en plataformas** como GitHub.

Editor por defecto: La configuración `core.editor` determina qué editor se abre para mensajes de commit. Esta elección afecta la **fluidez del workflow** diario. Un editor familiar acelera el proceso de escritura de commits descriptivos.

Comportamiento de fin de línea: La configuración `core.autocrlf` maneja diferencias entre sistemas operativos (Windows vs Unix). Una configuración incorrecta puede crear **commits innecesarios** por cambios invisibles en finales de línea, contaminando el historial con ruido.

Configuración de push por defecto: `push.default` determina qué sucede cuando ejecutas `git push` sin especificar rama. Esta configuración previene **errores accidentales** que pueden afectar repositorios compartidos.

Verificación de Instalación

Verificar que Git esté correctamente instalado no es opcional, sino **crítico** para evitar problemas futuros. Los comandos de verificación sirven como **diagnóstico preventivo** que identifica problemas antes de que afecten el trabajo real.

Task 2: El Ciclo Fundamental: Init, Add, Commit, Status (15 minutos)

El ciclo básico de Git representa el **workflow atómico** de control de versiones. Estos cuatro comandos forman el **núcleo operativo** de Git y understanding su interacción es fundamental para cualquier trabajo profesional con código.

Git Init: Nacimiento de un Repositorio

git init no es simplemente "crear una carpeta .git". Es el **acto fundacional** que transforma un directorio ordinario en un **sistema de control de versiones completo**.

Transformación conceptual: Antes de git init, tienes archivos. Después, tienes un **historial vivo** que puede rastrear cada cambio, crear ramas, fusionar trabajo, y colaborar con otros.

Archivos ocultos críticos: El comando crea el directorio .git que contiene toda la "magia" de Git: el database de objetos, configuración, hooks, referencias a ramas, etc.

Configuración automática: Git establece configuraciones por defecto inteligentes que funcionan para la mayoría de casos de uso.

Primer commit pendiente: Después de git init, todos los archivos existen en el **working directory**, esperando ser agregados al sistema de control de versiones.

Git Add: Preparación para el Commit

git add representa la **decisión consciente** de qué cambios incluir en el próximo commit. Es el puente entre el **working directory** (tu trabajo diario) y el **staging area** (cambios preparados).

Selección granular: Puedes agregar archivos específicos, patrones con wildcards, o incluso porciones específicas de archivos con *git add -p*. Esta granularidad permite **commits enfocados** que agrupan cambios lógicos relacionados.

Revisión antes del commit: El staging area actúa como un **banco de pruebas** donde puedes revisar exactamente qué cambios se incluirán antes de hacerlos permanentes.

Separación de concerns: Permite preparar múltiples cambios lógicos separadamente, creando commits que cuentan una **historia clara** de la evolución del código.

Git Status: Estado del Sistema en Tiempo Real

git status es el **dashboard vivo** de tu repositorio. Más que un comando informativo, es una **herramienta de diagnóstico** que te mantiene orientado en el workflow.

Tres secciones críticas:

- **Changes to be committed:** Archivos en staging area (verdes)
- **Changes not staged for commit:** Archivos modificados pero no preparados (rojos)
- **Untracked files:** Archivos nuevos que Git no está rastreando

***Orientación contextual:** Los mensajes de git status cambian según el estado del repositorio, proporcionando **guía contextual** sobre qué hacer a continuación.*

Git Commit: Registro Permanente del Cambio

*git commit transforma los cambios preparados en un **registro inmutable** del historial del proyecto.*

***Mensaje descriptivo obligatorio:** Git fuerza escribir un mensaje que explique qué cambió y por qué. Este mensaje se convierte en **documentación viva** del proyecto.*

***Hash único generado:** Cada commit recibe un identificador SHA-1 único que permite referenciarlo permanentemente, crear ramas desde él, revertirlo, etc.*

***Estructura de datos:** El commit contiene no solo los cambios, sino también referencias a commits padres, creando la **cadena histórica** que hace Git tan poderoso.*

El Workflow Diario

*Estos comandos forman un **ciclo virtuoso**:*

- 1. Trabajas en tus archivos (working directory)*
- 2. Ves qué cambió (git status)*
- 3. Preparas cambios específicos (git add)*
- 4. Confirmas con mensaje descriptivo (git commit)*
- 5. Repites el ciclo*

Este workflow se convierte en *segunda naturaleza*, permitiendo desarrollo fluido con *control total* sobre qué se registra y cuándo.

****Ejercicio****: Configura Git y crea tu primer repositorio local

Ejercicio práctico para aplicar los conceptos aprendidos.

Instalación

```sh

En macOS: `brew install git`

En Ubuntu/Debian: `sudo apt install git`

Verifica: `git --version`

```

Configuración Inicial

```sh

`git config --global user.name "Tu Nombre Completo"`

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

```
git config --global core.editor "code --wait" # Para VS Code
```

```
git config --global init.defaultBranch main
```

```
```
```

Verificación de configuración:

```
```sh
```

```
git config --list --show-origin
```

```
```
```

Crear primer repositorio:

```
```sh
```

```
mkdir mi-primer-repo
```

```
cd mi-primer-repo
```

```
git init
```

```
echo "# Mi primer repositorio" > README.md
```

```
git status
```

```
git add README.md
```

*git status*

*git commit -m "Initial commit: Agregar README básico"*

*git log --oneline*

'''