



Historia de Git

Git fue creado por Linus Torvalds en abril de 2005. El nombre tiene varias versiones divertidas: Torvalds bromeó que "git" es jerga británica para una persona desagradable, comentando sarcásticamente "Soy un bastardo egocéntrico y nombro todos mis proyectos por mí mismo".

El Nacimiento Rápido de Git

1

3 de abril 2005

Inicio del desarrollo

2

6 de abril

Anuncio del proyecto

3

7 de abril

Auto-hospedado

4

18 de abril

Primer merge múltiple

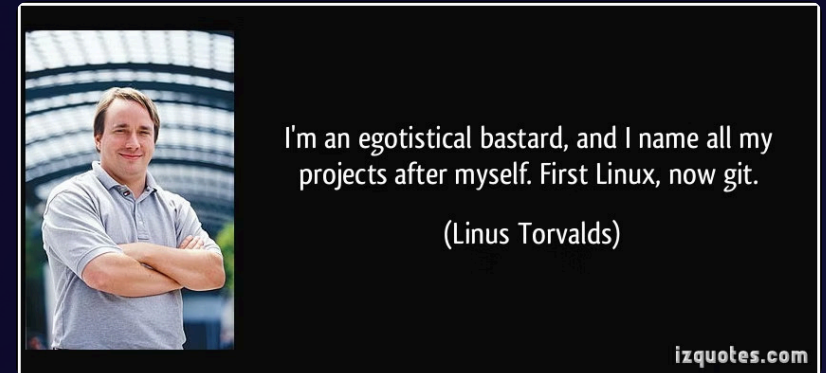
La urgencia surgió porque BitKeeper revocó su licencia gratuita, dejando al desarrollo del kernel de Linux sin herramientas. Git fue diseñado en C y scripts de shell. GitHub nació en abril de 2008.



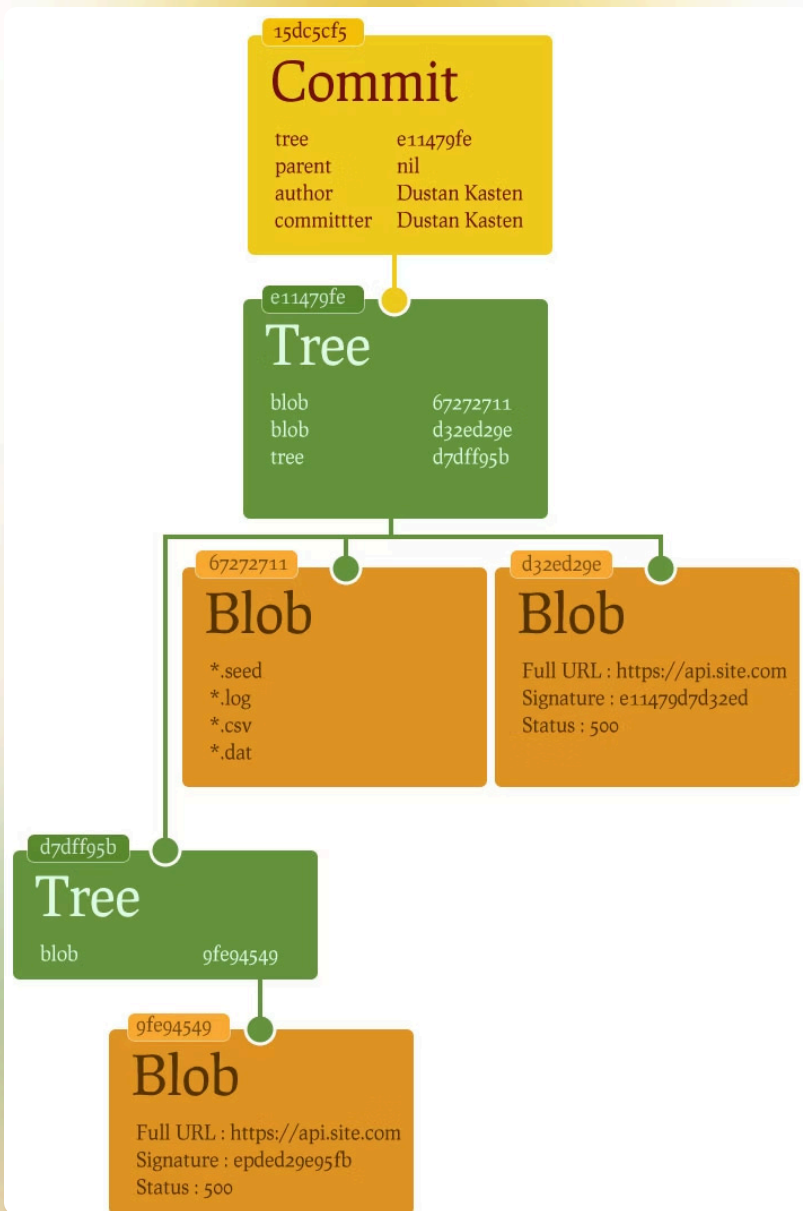
¿Qué es Git Realmente?

Git es un **sistema de control de versiones distribuido (DVCS)**. Actúa como una "máquina del tiempo para tu código".

Imagina guardar tu progreso en un videojuego, pero manteniendo todos los estados anteriores, viendo exactamente qué cambió, pudiendo volver a cualquier punto, y creando "realidades alternativas" para experimentar.



Git rastrea cambios en archivos a lo largo del tiempo, permitiendo que múltiples desarrolladores trabajen simultáneamente, manteniendo un historial completo de modificaciones.



La Magia Detrás de Git



Sistema de Objetos

Cuatro tipos: blobs (archivos), trees (directorios), commits (instantáneas) y tags (etiquetas)



Hashes SHA-1

Huella digital única para cada archivo. Si no cambia, Git apunta al mismo hash



Branches Ligeros

Solo punteros a commits. Git reconstruye tu directorio según el commit

Todo está guardado en la carpeta oculta `.git`. Dentro de `.git/objects` están todos los objetos comprimidos. Es como un almacén donde cada versión se guarda una sola vez, y Git arma el "set de decorado" correcto cada vez que cambias de branch.

Commits: Guardando Tu Progreso

¿Qué es un Commit?

Como guardar progreso en un videojuego. Cada checkpoint incluye:

- Qué cambió exactamente
- Quién hizo los cambios
- Cuándo se hicieron
- Por qué se hicieron

Comandos Básicos

```
git add archivo.js  
# Preparar archivo
```

```
git commit -m "Mensaje"  
# Crear checkpoint
```



Las Tres Áreas Fundamentales



Working Directory

Tu carpeta de proyecto donde editas archivos. Como tu escritorio de trabajo.



Staging Area

Espacio intermedio donde preparas cambios. Como preparar una maleta para un viaje.



Repository

Donde Git guarda permanentemente tus commits en la carpeta .git



Flujo de trabajo: Modificas archivo → `git add` (staging) → `git commit` (repository)

Local vs Remoto

Repositorio Local

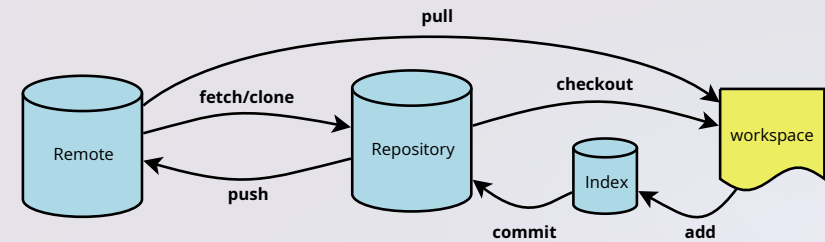
Copia completa del proyecto en tu computadora con todo su historial.
Trabajas aquí sin necesidad de internet.

```
git clone [url]
# Clonar repositorio

git remote -v
# Ver remotos configurados
```

Repositorio Remoto

Copia del proyecto en un servidor (GitHub, GitLab). Sirve como punto central para que el equipo comparta código.



Branches: Realidades Paralelas

Los branches son como crear líneas temporales alternativas. Imagina escribir una historia y probar dos finales diferentes sin perder el original.



```
git branch nueva-funcionalidad  
# Crear rama
```

```
git branch -a  
# Ver todas las ramas
```

```
git branch -d rama-vieja  
# Eliminar rama
```

Checkout, Push y Pull

01

Checkout: Tu Máquina del Tiempo

Te permite moverte entre branches o volver a commits antiguos.

```
git checkout nombre-rama  
git switch nombre-rama #  
Recomendado
```

02

Push: Enviar al Remoto

Envías tus commits locales al repositorio remoto. Como subir tu trabajo a la nube.

```
git push origin main
```

03

Pull: Traer del Remoto

Traes cambios del repositorio remoto. Como descargar el trabajo de tus compañeros.

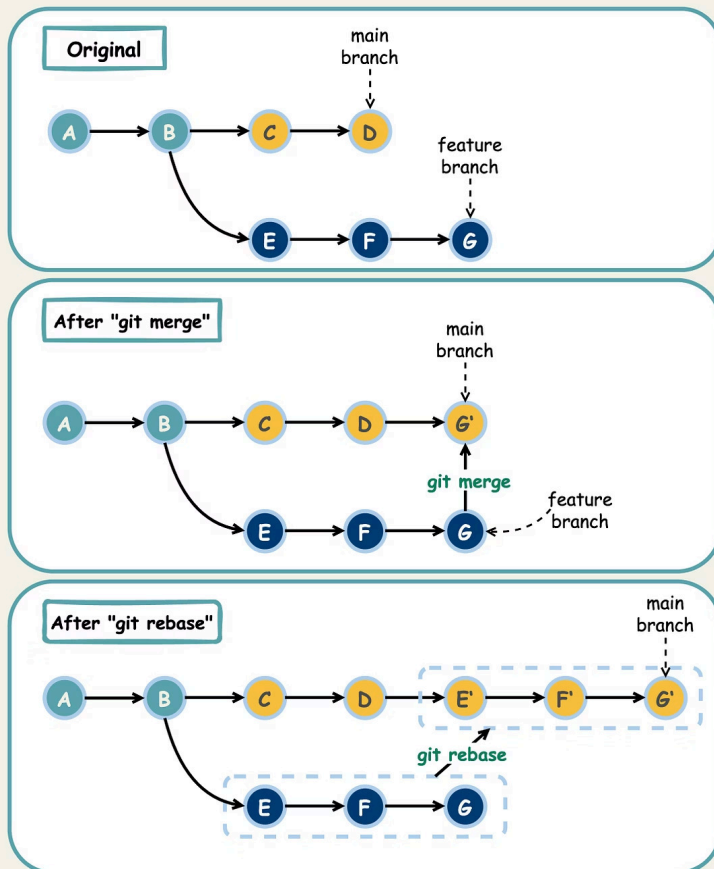
```
git pull origin main
```

📌 **Diferencia clave:** `git fetch` solo descarga (más seguro) | `git pull` descarga y aplica automáticamente (fetch + merge)

Merge vs Rebase

Git Merge vs. Git Rebase

 blog.bytebytego.com



MERGE

Une dos branches creando un nuevo commit de fusión.
Mantiene toda la historia.

```
git checkout main
git merge nueva-
funcionalidad
```

✓ Historia completa y honesta

❌ Historial "ruidoso"



REBASE

Reescribe la historia colocando tus commits encima de otra rama.

```
git checkout nueva-
funcionalidad
git rebase main
```

✓ Historia lineal y limpia

❌ Reescribe historia

Regla de oro: Usa **merge** para branches públicos, **rebase** solo para tu trabajo local privado antes de hacer push.

Conceptos adicionales

1. `.gitignore` Archivo que le dice a Git qué archivos ignorar (node_modules, archivos de configuración local, etc.)
2. `git status` Tu mejor amigo. Te muestra el estado actual: qué archivos cambiaron, qué está en staging, en qué branch estás.
3. `git log` Muestra el historial de commits. Úsalo con `--oneline --graph --all` para ver un árbol visual hermoso.
4. **Conflicts (Conflictos)** Cuando dos personas modifican la misma línea de código, Git no puede decidir automáticamente qué cambio mantener. Debes resolverlo manualmente.
5. `git stash` (Stash = Esconder / Guardar temporalmente) Guarda temporalmente cambios sin hacer commit, útil cuando necesitas cambiar de branch urgentemente.
6. **Fork** En GitHub, un fork es tu propia copia de un repositorio de otra persona. Puedes experimentar libremente sin afectar el original.
7. `git init` Inicializa un nuevo repositorio Git en una carpeta.
8. `git remote add origin [url]` Conecta tu repositorio local con un repositorio remoto.



illustrated-git.readthedocs.io



GIT Illustrated Cheatsheet — Illustrated GIT 1.0 documentation

COMANDOS BÁSICOS DE TERMINAL

```
pwd          # Mostrar directorio actual (Print Working Directory)
ls           # Listar archivos
ls -la       # Listar todos los archivos incluyendo ocultos
cd carpeta   # Cambiar de directorio (Change Directory)
cd ..        # Subir un nivel
mkdir nueva-carpeta # Crear directorio (Make Directory)
touch archivo.txt  # Crear archivo vacío
rm archivo.txt     # Eliminar archivo
rm -rf carpeta     # Eliminar carpeta y su contenido
cat archivo.txt    # Ver contenido de archivo
clear            # Limpiar terminal
```

3 PROYECTOS OPEN SOURCE RECOMENDADOS

1. First Contributions (<https://github.com/firstcontributions/first-contributions>) Este proyecto está específicamente diseñado para que principiantes hagan su primera contribución. Tiene más de 30,000 forks y una historia de commits muy activa. Tus estudiantes podrán ver miles de pull requests de personas aprendiendo Git por primera vez, lo cual es muy motivador. El historial muestra perfectamente cómo funciona el flujo de trabajo colaborativo.
2. freeCodeCamp (<https://github.com/freeCodeCamp/freeCodeCamp>) Es uno de los proyectos educativos más grandes en GitHub. Tiene un historial de commits impresionante con miles de contribuidores. Los estudiantes podrán ver cómo un proyecto masivo se gestiona con branches, issues bien organizados, y pull requests con revisiones detalladas. Es perfecto para mostrar colaboración a gran escala.
3. The Odin Project (<https://github.com/TheOdinProject/curriculum>) Otro proyecto educativo con excelente documentación de issues y pull requests. Tiene una comunidad muy activa y sus commits son fáciles de entender porque muchos son mejoras de contenido educativo, lo cual será familiar para tus estudiantes de fullstack.