

# Semántica de Commits

En la filosofía DevOps, el historial de Git no es solo un backup, es **documentación cronológica**. Adoptar un estándar como *Conventional Commits* facilita la trazabilidad y la automatización.

## 1. Estructura del Mensaje

Un commit profesional debe seguir esta estructura: `tipo(alcance): descripción breve en minúsculas`

### Tipos principales (Types)

Tipo	Cuándo usarlo	Ejemplo
feat	Nueva funcionalidad o <b>nuevo artículo/nota</b> .	<code>feat(linux): add guide for cron scheduling</code>
docs	Cambios exclusivos en documentación (corregir typos, gramática).	<code>docs(multimedia): fix typo in ImageMagick guide</code>
fix	Corregir un error técnico (link roto, error de MDX).	<code>fix(search): fix invalid css selector in modal</code>
style	Cambios que no afectan la lógica (CSS, espacios, formato).	<code>style(ui): update terminal cursor blink speed</code>
chore	Tareas de mantenimiento (actualizar dependencias, .gitignore).	<code>chore: update docusaurus to v3.2</code>
refactor	Cambio de código que ni corrige error ni añade feature.	<code>refactor(kb): restructure folders for better navigation</code>

## 2. Commits Simples vs. Multilínea

### Commits Simples

Para cambios atómicos y directos.

```
git commit -m "feat(linux): add stat command reference"
```

### Commits Multilínea

Útiles cuando el cambio requiere una explicación del "por qué" o una lista de cambios realizados.

```
git commit -m "feat(ui): implement terminal identity" -m "- Add $ prompt via CSS  
- Implement blinking cursor animation  
- Force green color palette on search modal"
```

Nota: Al usar múltiples `-m`, Git los interpreta como párrafos separados.



#### LA REGLA DE ORO

El mensaje debe completar la frase: "Si aplico este commit, yo estoy... [mensaje]".

- Ejemplo: Si aplico este commit, yo estoy... **actualizando el estilo del buscador.**