Git y GitHub: Guía Resumida de Supervivencia Ampliada

Autor: **Juan Garcia**

November 20, 2025

Abstract

Este documento es una guía de referencia exhaustiva y concisa para Git y GitHub. Cubre la instalación, el flujo de trabajo completo, la gestión de ramas y versiones, y una sección detallada sobre cómo navegar y modificar el historial del repositorio (`log` , `reset` , `revert`).

1 Fundamentos y Configuración Inicial

1.1 Conceptos Clave

- **Git:** Herramienta de **control de versiones distribuido** (local). Diseñada para velocidad y manejo de grandes proyectos (como el kernel de Linux).
- **GitHub:** Plataforma web que **aloja repositorios Git** de forma remota, facilitando la colaboración y sirviendo como *escaparate del programador*.
- **Flujo de Trabajo Git:** Implica tres áreas: **Directorio de Trabajo** $\rightarrow$ **Área de Staging (Preparación)** $\rightarrow$ **Repositorio Local (Confirmado)**.

1.2 Configuración Esencial

- **Nombre y Correo Global:** `git config --global user.name "Juan Garcia"` y `git config --global`
- **Editor por Defecto:** `git config --global core.editor "vim"` (Ejemplo con Vim)
- **Cacheo de Contraseña:** `git config --global credential.helper 'cache --timeout=3600'`
- **Ver Configuración:** `git config --list`

2 Ciclo de Vida del Código: Operaciones Remotas y Locales

2.1 Inicio y Sincronización

- **Crear Repositorio Local:** `git init` (Crea la carpeta oculta `.git`)
- **Clonar (Bajar por primera vez):** `git clone [URL]`
- **Enlazar Local con Remoto (si iniciaste localmente):** `git remote add origin [URL]`
- **Actualizar Copia Local (Traer y Fusionar):** `git pull`

2.2 El Flujo Básico (Add, Commit, Push)

1. **Preparar Cambios (`add` / Aperitivo):** `git add [archivo]` o `git add .` (o `--all`)
2. **Confirmar Cambios (`commit` / Comida):** `git commit -m "Mensaje descriptivo"`
3. **Enviar Cambios al Remoto (`push` / Postre):** `git push`

2.3 Archivos Clave

- `.gitignore`**:** Lista de archivos y directorios que Git debe **ignorar** (ej., binarios `*.class`, logs, temporales `*.temp`).
- `README.md`**:** Documento de presentación del proyecto en formato **Markdown**.
- `HEAD`**:** Puntero interno que indica el **último commit** en la rama de trabajo actual.

3 Control de Historial y Viaje en el Tiempo

3.1 Navegación e Inspección

- **Ver Historial Completo:** `git log`
- **Ver Historial Conciso:** `git log --oneline`
- **Ver Historial Conciso por Archivo:** `git log --oneline [archivo]`
- **Ver Ramas y Merges Gráficamente:** `git log --graph --oneline --all`

3.2 Deshacer Cambios (*Uncommitted Changes*)

- **Quitar del Staging (*Unstage*):** Mantiene las modificaciones en el archivo, pero lo saca del área de preparación. `git reset HEAD [archivo]`
- **Descartar Modificaciones Locales (*Discard*): Borra permanentemente** los cambios sin confirmar en el directorio de trabajo y restaura el archivo a su último commit. `git checkout -- [arch`

3.3 Deshacer Commits (*Committed Changes*)

- **Restaurar Versión Anterior (por archivo):** Restaura un archivo a una versión específica de su historial. `git checkout HEAD 1 [archivo]` (Vuelve 1 commit atrás)
- **Revertir (Seguro - Mantiene Historial):** Crea un **nuevo commit** que anula los cambios de un commit anterior. El historial problemático se mantiene, pero neutralizado. `git revert [hash_comm`
- **Reset (Peligroso - Reescribe Historial):** Mueve la `HEAD` y la rama a un commit anterior. **Elimina permanentemente** el historial posterior, lo cual es peligroso si ya se hizo `push`.
  `git reset --hard [hash_commit]`

4 Desarrollo con Ramas y Versiones

4.1 Gestión de Ramas

- **Crear y Moverse:** `git checkout -b [nueva_rama]`
- **Cambiar de Rama:** `git checkout [rama_existente]`
- **Fusionar:** Fusiona una rama (*branch*) en la rama actual. `git merge [rama_a_fusionar]`

4.2 Resolución de Conflictos

- **Detección:** Ocurre cuando dos ramas modifican la **misma línea** del mismo archivo y se intenta un `pull` o `merge`. Git avisa con `CONFLICT (content)`.
- **Proceso:**
  1. Abrir el archivo y editarlo, eliminando los marcadores `<<<<<<< HEAD`, `=======` y `>>>>>>> [commi`
  2. Elegir la versión final correcta del código.
  3. **Marcar como resuelto:** `git add [archivo_conflictivo]`
  4. **Finalizar el Merge:** `git commit -m "Solución de conflicto"`

4.3 Etiquetado de Versiones (Tags)

- **Etiquetar un Lanzamiento:** `git tag -a v2.0 -m "Versión 2.0: Función Potencia Añadida"`
- **Viajar a una Versión (Temporal):** `git checkout v2.0`
- **Enviar Etiquetas a GitHub (Necesario):** `git push origin [nombre_etiqueta]` o `git push --tags`