Viajes en el Tiempo

Historial, Diferencias y Restauración

Clase 3 · Viajes en el Tiempo

GLUD — Grupo GNU/Linux Universidad Distrital

Control de Versiones y Desarrollo Colaborativo

Mapa de ruta

1. Introducción
2. Explorando el Historial
3. Comparando Cambios
4. Restaurar y Recuperar
5. Etiquetas y Versiones
6. Buenas Prácticas

1

Introducción

¿Por qué Viajar en el Tiempo?

El poder de Git

Git no solo guarda cambios, te permite **explorar el pasado**, **comparar versiones** y **recuperar estados anteriores**.

¿Por qué Viajar en el Tiempo?

El poder de Git

Git no solo guarda cambios, te permite **explorar el pasado**, **comparar versiones** y **recuperar estados anteriores**.

Escenarios comunes:

- ¿Cuándo se introdujo este bug?
- ¿Qué cambió entre versiones?
- ¿Quién modificó esta línea?
- Necesito volver a una versión estable

¿Por qué Viajar en el Tiempo?

El poder de Git

Git no solo guarda cambios, te permite **explorar el pasado**, **comparar versiones** y **recuperar estados anteriores**.

Escenarios comunes:

- ¿Cuándo se introdujo este bug?
- ¿Qué cambió entre versiones?
- ¿Quién modificó esta línea?
- Necesito volver a una versión estable

Herramientas para navegar:

- `git log` — Ver historial
- `git diff` — Comparar cambios
- `git restore` — Recuperar archivos
- `git tag` — Marcar versiones

“

Git is a time machine for your code.

— GitHub Guides

2

Explorando el Historial

git log: Tu Máquina del Tiempo

Definición

`git log` muestra el historial completo de commits del repositorio.

git log: Tu Máquina del Tiempo

Definición

`git log` muestra el historial completo de commits del repositorio.

Uso básico

```
git log
```

git log: Tu Máquina del Tiempo

Definición

`git log` muestra el historial completo de commits del repositorio.

Uso básico

```
git log
```

Salida típica

```
commit a3f5b2c8d1e4f6a7b8c9d0e1f2a3b4c5d6e7f8a9
Author:  Tu Nombre <tu.email@ejemplo.com>
Date:  Mon Jan 6 10:30:00 2026 -0500

feat(auth):  implementar login con JWT
```

Formatos de git log

Una línea por commit

```
git log --oneline
```

Formatos de git log

Una línea por commit

```
git log --oneline
```

Salida compacta

```
a3f5b2c feat(auth):  login JWT
f2a3b4c fix(api):  error 500
c5d6e7f docs:  actualizar README
```

Formatos de git log

Una línea por commit

```
git log --oneline
```

Con gráfico

```
git log --oneline --graph
```

Salida compacta

```
a3f5b2c feat(auth):  login JWT
f2a3b4c fix(api):  error 500
c5d6e7f docs:  actualizar README
```

Formatos de git log

Una línea por commit

```
git log --oneline
```

Con gráfico

```
git log --oneline --graph
```

Salida compacta

```
a3f5b2c feat(auth):  login JWT
f2a3b4c fix(api):  error 500
c5d6e7f docs:  actualizar README
```

Salida visual

```
* a3f5b2c feat(auth):  login
* f2a3b4c fix(api):  error
* c5d6e7f docs:  README
* b4c5d6e Initial commit
```

Formatos de git log

Una línea por commit

```
git log --oneline
```

Con gráfico

```
git log --oneline --graph
```

Salida compacta

```
a3f5b2c feat(auth): login JWT
f2a3b4c fix(api): error 500
c5d6e7f docs: actualizar README
```

Salida visual

```
* a3f5b2c feat(auth): login
* f2a3b4c fix(api): error
* c5d6e7f docs: README
* b4c5d6e Initial commit
```

`--oneline` es tu mejor aliado para visualizar rápidamente

Filtrar el Historial

1 Últimos N commits

```
git log -n 5       # Muestra últimos 5 commits
```

Filtrar el Historial

1. Últimos N commits

```
git log -n 5       # Muestra últimos 5 commits
```

2. Por autor

```
git log --author="Tu Nombre"
```

Filtrar el Historial

1. Últimos N commits

```
git log -n 5       # Muestra últimos 5 commits
```

2. Por autor

```
git log --author="Tu Nombre"
```

3. Por fecha

```
git log --since="2026-01-01" --until="2026-01-06"
```

Filtrar el Historial

1. Últimos N commits

```
git log -n 5       # Muestra últimos 5 commits
```

2. Por autor

```
git log --author="Tu Nombre"
```

3. Por fecha

```
git log --since="2026-01-01" --until="2026-01-06"
```

4. Por mensaje

```
git log --grep="feat"
```

git log Avanzado

Ver archivos modificados

```
git log --stat
```

git log Avanzado

Ver archivos modificados

```
git log --stat
```

Ver cambios completos

```
git log -p
```

git log Avanzado

Ver archivos modificados

```
git log --stat
```

Ver cambios completos

```
git log -p
```

Formato personalizado

```
git log --pretty=format:"%h - %an, %ar :  %s"
```

git log Avanzado

Ver archivos modificados

```
git log --stat
```

Ver cambios completos

```
git log -p
```

Formato personalizado

```
git log --pretty=format:"%h - %an, %ar :  %s"
```

Salida

```
a3f5b2c - Tu Nombre, 2 hours ago :  feat(auth):  implementar login
f2a3b4c - Tu Nombre, 1 day ago :  fix(api):  resolver error 500
```

Buscar en el Historial

git log -S

Busca commits que agregaron o eliminaron una cadena específica.

Buscar en el Historial

git log -S

Busca commits que agregaron o eliminaron una cadena específica.

Buscar "calculateTotal"

```
git log -S "calculateTotal"
```

Buscar en el Historial

git log -S

Busca commits que agregaron o eliminaron una cadena específica.

Buscar "calculateTotal"

```
git log -S "calculateTotal"
```

Ver cambios en archivo específico

```
git log - - src/Main.java
```

3

Comparando Cambios

git diff: Comparar Versiones

Recordatorio de Clase 1

Ya vimos `git diff` básico. Ahora exploraremos usos avanzados.

git diff: Comparar Versiones

Recordatorio de Clase 1

Ya vimos `git diff` básico. Ahora exploraremos usos avanzados.

Working vs Staged

```
git diff
```

Cambios no agregados

git diff: Comparar Versiones

Recordatorio de Clase 1

Ya vimos `git diff` básico. Ahora exploraremos usos avanzados.

Working vs Staged

```
git diff
```

Cambios no agregados

Staged vs Committed

```
git diff --staged
```

Cambios en staging

Comparar entre Commits

1. Entre dos commits específicos

```
git diff a3f5b2c f2a3b4c
```

Comparar entre Commits

1. Entre dos commits específicos

```
git diff a3f5b2c f2a3b4c
```

2. Commit actual vs anterior

```
git diff HEAD~1 HEAD
```

Comparar entre Commits

1. Entre dos commits específicos

```
git diff a3f5b2c f2a3b4c
```

2. Commit actual vs anterior

```
git diff HEAD~1 HEAD
```

3. Últimos N commits

```
git diff HEAD~3 HEAD
```

Referencias Relativas

Definición

Las **referencias relativas** permiten navegar el historial desde un punto.

Referencias Relativas

Definición

Las **referencias relativas** permiten navegar el historial desde un punto.

Notación ~

- `HEAD~1` — 1 commit atrás
- `HEAD~2` — 2 commits atrás
- `HEAD~5` — 5 commits atrás

Referencias Relativas

Definición

Las **referencias relativas** permiten navegar el historial desde un punto.

Notación ~

- `HEAD~1` — 1 commit atrás
- `HEAD~2` — 2 commits atrás
- `HEAD~5` — 5 commits atrás

Notación ^

- `HEAD^` — Padre directo
- `HEAD^^` — Abuelo
- `HEAD^2` — Segundo padre (merge)

Referencias Relativas

Definición

Las **referencias relativas** permiten navegar el historial desde un punto.

Notación ~

- `HEAD~1` — 1 commit atrás
- `HEAD~2` — 2 commits atrás
- `HEAD~5` — 5 commits atrás

Notación ^

- `HEAD^` — Padre directo
- `HEAD^^` — Abuelo
- `HEAD^2` — Segundo padre (merge)

Para linearidad simple, ~ es más claro

diff en Archivos Específicos

Diferencias en un archivo

```
git diff HEAD~1 HEAD - src/Main.java
```

diff en Archivos Específicos

Diferencias en un archivo

```
git diff HEAD~1 HEAD - src/Main.java
```

Salida típica

```
diff -git a/src/Main.java b/src/Main.java
index a3f5b2c..f2a3b4c 100644
-- a/src/Main.java
+++ b/src/Main.java
@@ -10,7 +10,7 @@ public class Main {
public static void main(String[] args) {
-           System.out.println("Hola Mundo");
+           System.out.println("Hola GLUD");
}
```

diff en Archivos Específicos

Diferencias en un archivo

```
git diff HEAD~1 HEAD - src/Main.java
```

Salida típica

```
diff -git a/src/Main.java b/src/Main.java
index a3f5b2c..f2a3b4c 100644
-- a/src/Main.java
+++ b/src/Main.java
@@ -10,7 +10,7 @@ public class Main {
public static void main(String[] args) {
-           System.out.println("Hola Mundo");
+           System.out.println("Hola GLUD");
}
```

– línea eliminada, + línea agregada

Herramientas Visuales para diff

git difftool

Lanza una herramienta gráfica para visualizar diferencias.

Herramientas Visuales para diff

git difftool

Lanza una herramienta gráfica para visualizar diferencias.

Configurar difftool

```
git config --global diff.tool vimdiff
# Alternativas:  meld, kdiff3, vscode
```

Herramientas Visuales para diff

git difftool

Lanza una herramienta gráfica para visualizar diferencias.

Configurar difftool

```
git config --global diff.tool vimdiff
# Alternativas:  meld, kdiff3, vscode
```

Usar difftool

```
git difftool HEAD~1 HEAD
```

Herramientas Visuales para diff

git difftool

Lanza una herramienta gráfica para visualizar diferencias.

Configurar difftool

```
git config --global diff.tool vimdiff
# Alternativas:  meld, kdiff3, vscode
```

Usar difftool

```
git difftool HEAD~1 HEAD
```

Recomendado para cambios complejos con múltiples archivos

4

Restaurar y Recuperar

git restore: Recuperar Archivos

Definición

`git restore` recupera archivos desde el staging o desde commits anteriores.

git restore: Recuperar Archivos

Definición

`git restore` recupera archivos desde el staging o desde commits anteriores.

Descartar cambios en working directory

```
git restore archivo.txt
```

git restore: Recuperar Archivos

Definición

`git restore` recupera archivos desde el staging o desde commits anteriores.

Descartar cambios en working directory

```
git restore archivo.txt
```

Advertencia

¡Esto elimina tus cambios locales permanentemente!

git restore: Recuperar Archivos

Definición

`git restore` recupera archivos desde el staging o desde commits anteriores.

Descartar cambios en working directory

```
git restore archivo.txt
```

Advertencia

¡Esto elimina tus cambios locales permanentemente!

Úsalo solo cuando estés seguro de descartar cambios

Quitar Archivos del Staging

Escenario

```
git add archivo.txt
# ¡Ups!  No quería agregarlo todavía
```

Quitar Archivos del Staging

Escenario

```
git add archivo.txt
# ¡Ups!  No quería agregarlo todavía
```

Solución

```
git restore --staged archivo.txt
```

Quitar Archivos del Staging

Escenario

```
git add archivo.txt
# ¡Ups!  No quería agregarlo todavía
```

Solución

```
git restore --staged archivo.txt
```

Resultado

El archivo vuelve a Working Directory. Tus cambios **se mantienen**.

Quitar Archivos del Staging

Escenario

```
git add archivo.txt
# ¡Ups!  No quería agregarlo todavía
```

Solución

```
git restore --staged archivo.txt
```

Resultado

El archivo vuelve a Working Directory. Tus cambios **se mantienen**.

Restaurar desde Commit Específico

Recuperar versión antigua

```
git restore --source=HEAD~3 archivo.txt
```

Restaurar desde Commit Específico

Recuperar versión antigua

```
git restore --source=HEAD~3 archivo.txt
```

Con hash de commit

```
git restore --source=a3f5b2c archivo.txt
```

Restaurar desde Commit Específico

Recuperar versión antigua

```
git restore --source=HEAD~3 archivo.txt
```

Con hash de commit

```
git restore --source=a3f5b2c archivo.txt
```

Uso típico

Recuperar un archivo que funcionaba 3 commits atrás sin afectar otros archivos.

git checkout vs git restore

Historia

`git checkout` hacía muchas cosas. Git 2.23+ lo dividió en comandos especializados.

git checkout vs git restore

Historia

`git checkout` hacía muchas cosas. Git 2.23+ lo dividió en comandos especializados.

Forma antigua:

```
git checkout archivo.txt
git checkout -- archivo.txt
git checkout HEAD archivo.txt
```

git checkout vs git restore

Historia

`git checkout` hacía muchas cosas. Git 2.23+ lo dividió en comandos especializados.

Forma antigua:

```
git checkout archivo.txt
git checkout -- archivo.txt
git checkout HEAD archivo.txt
```

Forma moderna:

```
git restore archivo.txt
git restore archivo.txt
git restore --source=HEAD archivo.txt
```

git checkout vs git restore

Historia

`git checkout` hacía muchas cosas. Git 2.23+ lo dividió en comandos especializados.

Forma antigua:

```
git checkout archivo.txt
git checkout -- archivo.txt
git checkout HEAD archivo.txt
```

Forma moderna:

```
git restore archivo.txt
git restore archivo.txt
git restore --source=HEAD archivo.txt
```

Usa `git restore`, es más claro y específico

Recuperar Archivos Eliminados

Escenario: Eliminaste un archivo

```
rm src/Config.java
git status       # Muestra: deleted: src/Config.java
```

Recuperar Archivos Eliminados

Escenario: Eliminaste un archivo

```
rm src/Config.java
git status       # Muestra: deleted: src/Config.java
```

Recuperar desde último commit

```
git restore src/Config.java
```

Recuperar Archivos Eliminados

Escenario: Eliminaste un archivo

```
rm src/Config.java
git status       # Muestra: deleted: src/Config.java
```

Recuperar desde último commit

```
git restore src/Config.java
```

Resultado

El archivo reaparece con el contenido del último commit.

5

Etiquetas y Versiones

¿Qué son las Etiquetas (Tags)?

Definición

Un **tag** es un marcador permanente que apunta a un commit específico, típicamente usado para versiones.

¿Qué son las Etiquetas (Tags)?

Definición

Un **tag** es un marcador permanente que apunta a un commit específico, típicamente usado para versiones.

Usos comunes:

- Marcar releases (`v1.0.0`)
- Identificar versiones estables
- Puntos de referencia importantes
- Facilitar rollbacks

¿Qué son las Etiquetas (Tags)?

Definición

Un **tag** es un marcador permanente que apunta a un commit específico, típicamente usado para versiones.

Usos comunes:

- Marcar releases (`v1.0.0`)
- Identificar versiones estables
- Puntos de referencia importantes
- Facilitar rollbacks

Tipos de tags:

- **Ligeros** — Solo un puntero
- **Anotados** — Con metadata completa

Usa tags anotados para releases

Crear Tags

1. Tag ligero

```
git tag v0.1.0-alpha
```

Crear Tags

1. Tag ligero

```
git tag v0.1.0-alpha
```

2. Tag anotado (recomendado)

```
git tag -a v0.1.0-alpha -m "Primera versión alpha"
```

Crear Tags

1. Tag ligero

```
git tag v0.1.0-alpha
```

2. Tag anotado (recomendado)

```
git tag -a v0.1.0-alpha -m "Primera versión alpha"
```

3. Tag en commit específico

```
git tag -a v0.0.9 a3f5b2c -m "Versión de prueba"
```

Listar y Ver Tags

Listar todos los tags

```
git tag
```

Listar y Ver Tags

Listar todos los tags

```
git tag
```

Buscar tags por patrón

```
git tag -l "v0.1.*"
```

Listar y Ver Tags

Listar todos los tags

```
git tag
```

Buscar tags por patrón

```
git tag -l "v0.1.*"
```

Ver información de tag anotado

```
git show v0.1.0-alpha
```

Listar y Ver Tags

Listar todos los tags

```
git tag
```

Buscar tags por patrón

```
git tag -l "v0.1.*"
```

Ver información de tag anotado

```
git show v0.1.0-alpha
```

Salida

```
tag v0.1.0-alpha
Tagger:  Tu Nombre <tu.email@ejemplo.com>
Date:  Mon Jan 6 15:00:00 2026 -0500
```

Versionamiento Semántico (SemVer)

Formato

`vMAYOR.MENOR.PARCHE-sufijo`

Versionamiento Semántico (SemVer)

Formato

`vMAYOR.MENOR.PARCHE-sufijo`

MAYOR
Cambios incompatibles

`1.x.x` → `2.0.0`

MENOR
Nuevas funcionalidades

`1.2.x` → `1.3.0`

PARCHE
Correcciones de bugs

`1.2.3` → `1.2.4`

Versionamiento Semántico (SemVer)

Formato

`vMAYOR.MENOR.PARCHE-sufijo`

MAYOR
Cambios incompatibles

1.x.x → 2.0.0

MENOR
Nuevas funcionalidades

1.2.x → 1.3.0

PARCHE
Correcciones de bugs

1.2.3 → 1.2.4

Sufijos: `-alpha`, `-beta`, `-rc1`, `-rc2`

Ejemplos: `v0.1.0-alpha`, `v1.0.0-beta`, `v2.3.1`

Crear Versión v0.1.0-alpha

Práctica guiada

```
# Asegúrate de estar en un estado limpio
git status

# Crea el tag anotado
git tag -a v0.1.0-alpha -m "Release:  Primera versión alpha"

# Verifica
git tag
git show v0.1.0-alpha
```

Crear Versión v0.1.0-alpha

Práctica guiada

```
# Asegúrate de estar en un estado limpio
git status

# Crea el tag anotado
git tag -a v0.1.0-alpha -m "Release:  Primera versión alpha"

# Verifica
git tag
git show v0.1.0-alpha
```

Este tag marca tu primer hito en el proyecto

Compartir Tags

Importante

Por defecto, `git push` **NO** envía tags al remoto.

Compartir Tags

Importante

Por defecto, `git push` **NO** envía tags al remoto.

Enviar un tag específico

```
git push origin v0.1.0-alpha
```

Compartir Tags

Importante

Por defecto, `git push` **NO** envía tags al remoto.

Enviar un tag específico

```
git push origin v0.1.0-alpha
```

Enviar todos los tags

```
git push origin --tags
```

Compartir Tags

Importante

Por defecto, `git push` **NO** envía tags al remoto.

Enviar un tag específico

```
git push origin v0.1.0-alpha
```

Enviar todos los tags

```
git push origin --tags
```

En GitHub/GitLab, los tags aparecen en la sección "Releases"

Eliminar Tags

Eliminar tag local

```
git tag -d v0.1.0-alpha
```

Eliminar Tags

Eliminar tag local

```
git tag -d v0.1.0-alpha
```

Eliminar tag remoto

```
git push origin --delete v0.1.0-alpha
```

Eliminar Tags

Eliminar tag local

```
git tag -d v0.1.0-alpha
```

Eliminar tag remoto

```
git push origin --delete v0.1.0-alpha
```

Advertencia

Ten cuidado al eliminar tags públicos. Otros pueden estar usándolos.

Navegar a un Tag

Ver estado en un tag

```
git checkout v0.1.0-alpha
```

Navegar a un Tag

Ver estado en un tag

```
git checkout v0.1.0-alpha
```

Estado "detached HEAD"

Estarás en modo lectura. Para trabajar, crea una rama desde el tag.

Navegar a un Tag

Ver estado en un tag

```
git checkout v0.1.0-alpha
```

Estado "detached HEAD"

Estarás en modo lectura. Para trabajar, crea una rama desde el tag.

Crear rama desde tag

```
git checkout -b bugfix-v0.1 v0.1.0-alpha
```

Navegar a un Tag

Ver estado en un tag

```
git checkout v0.1.0-alpha
```

Estado "detached HEAD"

Estarás en modo lectura. Para trabajar, crea una rama desde el tag.

Crear rama desde tag

```
git checkout -b bugfix-v0.1 v0.1.0-alpha
```

Útil para crear hotfixes sobre versiones específicas

6

Buenas Prácticas

Cuándo Usar Cada Comando

git log

- Revisar historial
- Buscar cuándo cambió algo
- Encontrar autor de cambios
- Auditar el proyecto

git diff

- Revisar antes de commit
- Comparar versiones
- Entender cambios ajenos

Cuándo Usar Cada Comando

git log

- Revisar historial
- Buscar cuándo cambió algo
- Encontrar autor de cambios
- Auditar el proyecto

git diff

- Revisar antes de commit
- Comparar versiones
- Entender cambios ajenos

git restore

- Descartar cambios locales
- Quitar archivos de staging
- Recuperar archivos eliminados
- Probar código antiguo

git tag

- Marcar releases
- Identificar versiones estables
- Crear puntos de restauración

Flujo de Trabajo Recomendado

1 Antes de cada commit

```
git status        # ¿Qué cambió?
git diff          # ¿Cómo cambió?
git add archivo     # Agregar selectivamente
```

Flujo de Trabajo Recomendado

1 Antes de cada commit

```
git status       # ¿Qué cambió?
git diff         # ¿Cómo cambió?
git add archivo    # Agregar selectivamente
```

2 Después de varios commits

```
git log --oneline    # Revisar historial
git diff HEAD~5 HEAD  # ¿Qué cambió en total?
```

Flujo de Trabajo Recomendado

3 Al alcanzar un hito

```
git tag -a v0.1.0-alpha -m "Primera versión funcional"
```

Errores Comunes

✗ No hagas esto

- Usar `git restore` sin verificar cambios
- Eliminar tags públicos sin comunicar
- Ignorar `git diff` antes de commit
- Crear tags sin mensajes descriptivos
- No usar versionamiento semántico

Errores Comunes

✓ Haz esto

- Siempre revisa `git diff` antes de commit
- Usa tags anotados para releases
- Sigue SemVer consistentemente
- Documenta cambios importantes en tags

> *The best code is well-versioned code.*
>
> — GitHub Guides

7

Resumen

Lo que Aprendimos

Historial y Navegación

- `git log` y sus variantes
- Filtros por autor, fecha, mensaje
- Referencias relativas (`HEAD~N`)
- Buscar cambios específicos

Comparación

- `git diff` entre commits
- Diferencias en archivos específicos
- Herramientas visuales (difftool)

Lo que Aprendimos

Historial y Navegación

- `git log` y sus variantes
- Filtros por autor, fecha, mensaje
- Referencias relativas (`HEAD~N`)
- Buscar cambios específicos

Comparación

- `git diff` entre commits
- Diferencias en archivos específicos
- Herramientas visuales (difftool)

Restauración

- `git restore` para working directory
- `git restore --staged` para staging
- Recuperar desde commits específicos
- Diferencia con `git checkout`

Versionamiento

- Tags ligeros y anotados
- Versionamiento semántico
- Compartir y eliminar tags
- Crear `v0.1.0-alpha`

Comandos Clave


```
# Historial
git log --oneline --graph
git log --author="Nombre" --since="2026-01-01"

# Diferencias
git diff HEAD~1 HEAD
git diff a3f5b2c f2a3b4c -- archivo.txt

# Restauración
git restore archivo.txt
git restore --staged archivo.txt

# Tags
git tag -a v0.1.0-alpha -m "Primera versión"
git push origin --tags
```

Recursos

Git:

- https://git-scm.com/docs/git-log
- https://git-scm.com/docs/git-diff
- https://git-scm.com/docs/git-restore
- https://git-scm.com/docs/git-tag

Versionamiento:

- https://semver.org/
- Conventional Commits
- Git Visualization Tools
- GitHub Releases Guide

Práctica interactiva:

- https://learngitbranching.js.org/
- Git Immersion: https://gitimmersion.com/

Próxima Clase

Concepto de punteros y referencias. Ramas feature/* y hotfix/*. Por qué nunca trabajar directo en main.

¡Gracias!

Nos vemos en la próxima clase