Ciclo de Vida del Código

Estados del Código en Git

Clase 2 · Ciclo de Vida del Código

GLUD — Grupo GNU/Linux Universidad Distrital

Control de Versiones y Desarrollo Colaborativo

Mapa de ruta

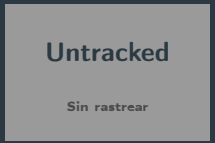
1. Estados de Archivos
2. Conventional Commits
3. Estructura de Proyecto
4. Configuración de .gitignore
5. Buenas Prácticas
6. Resumen

1

Estados de Archivos

El Ciclo de Vida de un Archivo en Git

Estados de Archivos



1 Untracked

Archivo nuevo, Git no lo conoce

2 Staged

Listo para commit

3 Committed

Guardado en historial

El Ciclo de Vida de un Archivo en Git

Estados de Archivos

1 Untracked

Archivo nuevo, Git no lo conoce

2 Staged

Listo para commit

3 Committed

Guardado en historial

El Ciclo de Vida de un Archivo en Git

Estados de Archivos

1 Untracked

Archivo nuevo, Git no lo conoce

2 Staged

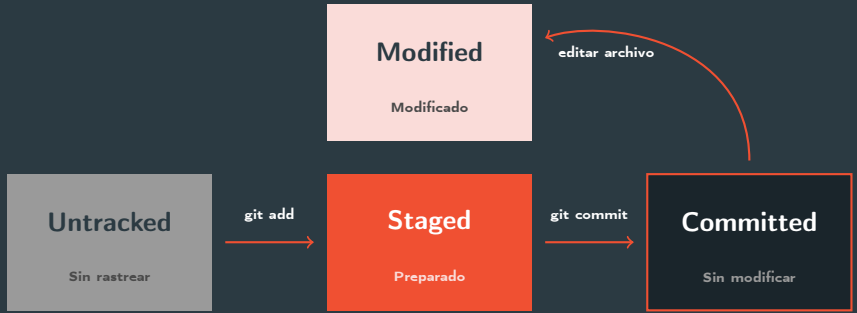
Listo para commit

3 Committed

Guardado en historial

El Ciclo de Vida de un Archivo en Git

Estados de Archivos



1 Untracked

Archivo nuevo, Git no lo conoce

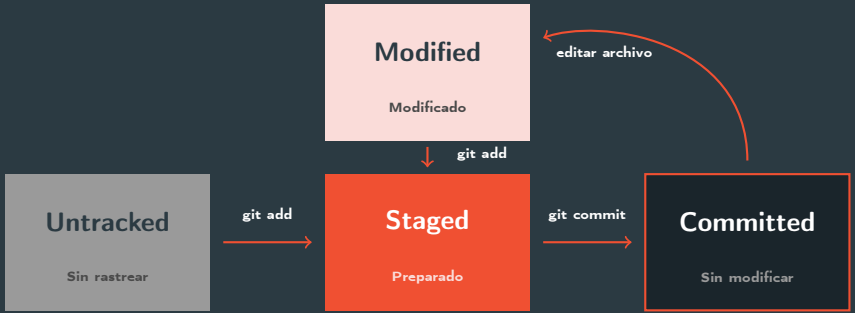
2 Staged

Listo para commit

3 Committed

Guardado en historial

El Ciclo de Vida de un Archivo en Git



1 Untracked
Archivo nuevo, Git no lo conoce

2 Staged
Listo para commit

3 Committed
Guardado en historial

Estado: Untracked

Definición

Un archivo **Untracked** es un archivo nuevo que Git aún no está rastreando.

Estado: Untracked

Definición

Un archivo **Untracked** es un archivo nuevo que Git aún no está rastreando.

Crear archivo

```
echo "Hola" > nuevo.txt
git status
```

Características:

- Git lo detecta pero no lo sigue
- Aparece en rojo en `git status`
- No se incluye en commits

Estado: Untracked

Definición

Un archivo **Untracked** es un archivo nuevo que Git aún no está rastreando.

Crear archivo

```
echo "Hola" > nuevo.txt
git status
```

Salida de git status

```
Untracked files:
(use "git add..." )

nuevo.txt
```

Características:

- Git lo detecta pero no lo sigue
- Aparece en rojo en `git status`
- No se incluye en commits

Usa `git add` para comenzar a rastrearlo

Estado: Staged

Definición

Un archivo **Staged** está en el área de preparación, listo para ser confirmado.

Estado: Staged

Definición

Un archivo **Staged** está en el área de preparación, listo para ser confirmado.

Agregar al staging

```
git add nuevo.txt
git status
```

Estado: Staged

Definición

Un archivo **Staged** está en el área de preparación, listo para ser confirmado.

Agregar al staging

```
git add nuevo.txt
git status
```

Salida de git status

```
Changes to be committed:
(use "git restore --staged..." to unstage)
new file:   nuevo.txt
```

Estado: Committed / Unmodified

Definición

Un archivo **Committed** está guardado en el historial de Git y no ha sido modificado desde el último commit.

Estado: Committed / Unmodified

Definición

Un archivo **Committed** está guardado en el historial de Git y no ha sido modificado desde el último commit.

Crear commit

```
git commit -m "feat:  agregar archivo nuevo"
git status
```

Estado: Committed / Unmodified

Definición

Un archivo **Committed** está guardado en el historial de Git y no ha sido modificado desde el último commit.

Crear commit

```
git commit -m "feat:  agregar archivo nuevo"
git status
```

Salida de git status

```
On branch main
nothing to commit, working tree clean
```

Estado: Modified

Definición

Un archivo **Modified** ha sido editado desde el último commit pero aún no está en staging.

Estado: Modified

Definición

Un archivo **Modified** ha sido editado desde el último commit pero aún no está en staging.

Modificar archivo

```
echo "Línea nueva" » nuevo.txt
git status
```

Estado: Modified

Definición

Un archivo **Modified** ha sido editado desde el último commit pero aún no está en staging.

Modificar archivo

```
echo "Línea nueva" » nuevo.txt
git status
```

Salida de git status

```
Changes not staged for commit:
(use "git add..." to update)
(use "git restore..." to discard)

modified:   nuevo.txt
```

Comandos de Control de Estado

Ver estado

```
git status
git status -s
```

Sacar de staging

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

Comandos de Control de Estado

Ver estado

```
git status
git status -s
```

Descartar cambios

```
git restore archivo.txt
```

Sacar de staging

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

Cuidado

`git restore` sin `--staged` descarta cambios permanentemente

2

Conventional Commits

¿Qué son los Conventional Commits?

Definición

Conventional Commits es una convención para escribir mensajes de commit estructurados y significativos.

¿Qué son los Conventional Commits?

Definición

Conventional Commits es una convención para escribir mensajes de commit estructurados y significativos.

Estructura

```
<tipo>[ámbito opcional]:  <descripción>

[cuerpo opcional]

[nota(s) al pie opcional(es)]
```

¿Qué son los Conventional Commits?

Definición

Conventional Commits es una convención para escribir mensajes de commit estructurados y significativos.

Estructura

```
<tipo>[ámbito opcional]:  <descripción>

[cuerpo opcional]

[nota(s) al pie opcional(es)]
```

Ventajas:

Historial legible

Automatización

Comunicación clara

Tipos de Commit

Cambios principales:

- **feat**: Nueva funcionalidad
- **fix**: Corrección de bug
- **refactor**: Cambio de código sin nueva función
- **perf**: Mejora de rendimiento

Tipos de Commit

Cambios principales:

- **feat**: Nueva funcionalidad
- **fix**: Corrección de bug
- **refactor**: Cambio de código sin nueva función
- **perf**: Mejora de rendimiento

Cambios auxiliares:

- **docs**: Documentación
- **style**: Formato (espacios, etc.)
- **test**: Agregar/modificar tests
- **chore**: Tareas de mantenimiento
- **build**: Cambios en build
- **ci**: Integración continua

Ejemplos de Conventional Commits

feat: Nueva funcionalidad

```
git commit -m "feat: agregar validación de email"
git commit -m "feat(auth): implementar login con JWT"
```

Ejemplos de Conventional Commits

feat: Nueva funcionalidad

```
git commit -m "feat: agregar validación de email"
git commit -m "feat(auth): implementar login con JWT"
```

fix: Corrección

```
git commit -m "fix: corregir cálculo de total"
git commit -m "fix(api): resolver error 500 en endpoint"
```

Ejemplos de Conventional Commits

feat: Nueva funcionalidad

```
git commit -m "feat: agregar validación de email"
git commit -m "feat(auth): implementar login con JWT"
```

fix: Corrección

```
git commit -m "fix: corregir cálculo de total"
git commit -m "fix(api): resolver error 500 en endpoint"
```

docs: Documentación

```
git commit -m "docs: actualizar README con instrucciones"
git commit -m "docs(api): documentar endpoint de usuarios"
```

Más Ejemplos

refactor: Refactorización

```
git commit -m "refactor: simplificar lógica de validación"
git commit -m "refactor(service): extraer clase helper"
```

Más Ejemplos

refactor: Refactorización

```
git commit -m "refactor: simplificar lógica de validación"
git commit -m "refactor(service): extraer clase helper"
```

chore: Tareas de mantenimiento

```
git commit -m "chore: actualizar dependencias"
git commit -m "chore: configurar .gitignore"
```

Más Ejemplos

refactor: Refactorización

```
git commit -m "refactor: simplificar lógica de validación"
git commit -m "refactor(service): extraer clase helper"
```

chore: Tareas de mantenimiento

```
git commit -m "chore: actualizar dependencias"
git commit -m "chore: configurar .gitignore"
```

test: Pruebas

```
git commit -m "test: agregar tests unitarios"
git commit -m "test(user): validar creación de usuario"
```

Commits con Breaking Changes

Breaking Change

Un **breaking change** es un cambio que rompe la compatibilidad con versiones anteriores.

Commits con Breaking Changes

Breaking Change

Un **breaking change** es un cambio que rompe la compatibilidad con versiones anteriores.

Sintaxis con !

```
git commit -m "feat!:  cambiar estructura de API"
git commit -m "refactor(core)!:  renombrar métodos principales"
```

Commits con Breaking Changes

Breaking Change

Un **breaking change** es un cambio que rompe la compatibilidad con versiones anteriores.

Sintaxis con !

```
git commit -m "feat!:  cambiar estructura de API"
git commit -m "refactor(core)!:  renombrar métodos principales"
```

Con nota al pie

```
git commit -m "feat:  cambiar endpoint de login

BREAKING CHANGE: el endpoint cambió de /auth a /api/v2/auth"
```

> *A commit message should tell a story about what changed and why.*
>
> — Linus Torvalds

3

Estructura de Proyecto

Proyecto Maven Estándar

Estructura

```
mi-proyecto/
- pom.xml
- README.md
- .gitignore
- src/
  - main/
    - java/
      - com/ejemplo/
        - Main.java
        - modelo/
  - resources/
  - test/
    - java/
```

Proyecto Maven Estándar

Estructura

```
mi-proyecto/
- pom.xml
- README.md
- .gitignore
- src/
  - main/
    - java/
      - com/ejemplo/
        - Main.java
        - modelo/
  - resources/
  - test/
    - java/
```

Archivos clave:

- `pom.xml`: Configuración Maven
- `README.md`: Documentación
- `.gitignore`: Archivos a ignorar
- `src/main/java`: Código fuente
- `src/test/java`: Pruebas

Proyecto Python con UV

Estructura

```
mi-proyecto/
- pyproject.toml
- README.md
- .gitignore
- .python-version
- src/
  - mi_proyecto/
    - __init__.py
    - main.py
    - utils.py
- tests/
  - test_main.py
  - test_utils.py
```

Proyecto Python con UV

Estructura

```
mi-proyecto/
- pyproject.toml
- README.md
- .gitignore
- .python-version
- src/
  - mi_proyecto/
    - __init__.py
    - main.py
    - utils.py
- tests/
  - test_main.py
  - test_utils.py
```

Archivos clave:

- `pyproject.toml`: Configuración UV y dependencias
- `README.md`: Documentación
- `.gitignore`: Archivos a ignorar
- `.python-version`: Versión de Python
- `src/`: Código fuente
- `tests/`: Pruebas unitarias

Crear Proyecto con UV

1. Crear proyecto desde cero

```
uv init mi-proyecto
cd mi-proyecto
```

Crear Proyecto con UV

1 Crear proyecto desde cero

```
uv init mi-proyecto
cd mi-proyecto
```

2 UV crea automáticamente

- Estructura de directorios completa
- `pyproject.toml` configurado
- Archivo `main.py` de ejemplo
- Archivo `.python-version`

Crear Proyecto con UV

1. Crear proyecto desde cero

```
uv init mi-proyecto
cd mi-proyecto
```

2. UV crea automáticamente

- Estructura de directorios completa
- `pyproject.toml` configurado
- Archivo `main.py` de ejemplo
- Archivo `.python-version`

3. Instalar dependencias

.gitignore para Python/UV

.gitignore

```
# Entorno virtual
.venv/
venv/
__pycache__/
*.pyc
*.pyo
dist/
build/
*.egg-info/
*.log
# Variables de entorno
.env
.env.local
```

Crear Estructura con Maven

1. Crear proyecto desde arqueotipo

```
mvn archetype:generate
-DgroupId=com.ejemplo
-DartifactId=mi-proyecto
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Crear Estructura con Maven

1 Crear proyecto desde arqueotipo

```
mvn archetype:generate
-DgroupId=com.ejemplo
-DartifactId=mi-proyecto
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

2 Maven crea automáticamente

- Estructura de directorios completa
- `pom.xml` configurado
- Clase `App.java` de ejemplo
- Clase de test `AppTest.java`

El archivo pom.xml

pom.xml básico

```
<project>
<modelVersion>4.0.0</modelVersion>
<groupId>com.ejemplo</groupId>
<artifactId>mi-proyecto</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
<maven.compiler.source>21</maven.compiler.source>
<maven.compiler.target>21</maven.compiler.target>
</properties>

<dependencies>
<!- tus dependencias ->
</dependencies>
</project>
```

README.md: Carta de Presentación

Importancia

El **README.md** es lo primero que ven otros desarrolladores. Debe ser claro y completo.

README.md: Carta de Presentación

Importancia

El **README.md** es lo primero que ven otros desarrolladores. Debe ser claro y completo.

Secciones recomendadas:

- Título y descripción del proyecto
- Requisitos (Java 21, Maven 3.X)
- Instrucciones de instalación
- Cómo ejecutar el proyecto
- Ejemplos de uso
- Licencia
- Contacto/contribución

Ejemplo de README.md

README.md

````
# Mi Proyecto

Descripción breve del proyecto.

## Requisitos
- Java 21+
- Maven 3.8+

## Instalación
```bash
git clone https://github.com/usuario/mi-proyecto.git
cd mi-proyecto
mvn clean install
```

## Uso
```bash
mvn exec:java -Dexec.mainClass="com.ejemplo.Main"
```
````

4

Configuración de .gitignore

¿Por qué usar .gitignore?

Propósito

El archivo **.gitignore** indica a Git qué archivos **no** debe rastrear.

¿Por qué usar .gitignore?

Propósito

El archivo **.gitignore** indica a Git qué archivos **no** debe rastrear.

Archivos a ignorar:

- Archivos compilados (`*.class`, `target/`)
- Archivos de configuración local (`.vscode/`, `.idea/`)
- Dependencias descargadas (`node_modules/`)
- Archivos de sistema (`.DS_Store`, `Thumbs.db`)
- Logs y temporales (`*.log`, `*.tmp`)
- Credenciales y secretos (`.env`, `secrets.yml`)

Nunca subas contraseñas, tokens o API keys

.gitignore para Java/Maven

.gitignore

```
# Archivos compilados
*.class
*.jar
*.war

# Directorio de Maven
target/

# IDEs
.idea/
.vscode/

# Logs
*.log
```

Usando gitignore.io

¿Qué es gitignore.io?

gitignore.io es un servicio que genera archivos .gitignore automáticamente según tu stack tecnológico.

Usando gitignore.io

¿Qué es gitignore.io?

gitignore.io es un servicio que genera archivos .gitignore automáticamente según tu stack tecnológico.

1. Visita `https://www.toptal.com/developers/gitignore`

Usando gitignore.io

¿Qué es gitignore.io?

gitignore.io es un servicio que genera archivos .gitignore automáticamente según tu stack tecnológico.

1. Visita `https://www.toptal.com/developers/gitignore`
2. Escribe: `Java`, `Maven`, `IntelliJ`, `VisualStudioCode`

Usando gitignore.io

¿Qué es gitignore.io?

gitignore.io es un servicio que genera archivos .gitignore automáticamente según tu stack tecnológico.

1. Visita `https://www.toptal.com/developers/gitignore`
2. Escribe: `Java`, `Maven`, `IntelliJ`, `VisualStudioCode`
3. Copia el contenido generado a tu `.gitignore`

También puedes usar la API o CLI: `gi java,maven > .gitignore`

Aplicar .gitignore a Archivos Rastreados

Problema

Si ya agregaste archivos a Git antes de crear .gitignore, seguirán siendo rastreados.

Aplicar .gitignore a Archivos Rastreados

Problema

Si ya agregaste archivos a Git antes de crear .gitignore, seguirán siendo rastreados.

Solución

```
# Remover del índice (sin borrar del disco)
git rm --cached -r target/

# Agregar .gitignore
git add .gitignore

# Commit
git commit -m "chore:  configurar .gitignore para Java/Maven"
```

Patrones Comunes en .gitignore

Patrones

```
# Archivo específico
config.local

# Extensión
*.log

# Directorio
build/

# Todos menos uno
*.jar
!lib/importante.jar
```

Patrones Comunes en .gitignore

Patrones

```
# Archivo específico
config.local

# Extensión
*.log

# Directorio
build/

# Todos menos uno
*.jar
!lib/importante.jar
```

Más patrones

```
# Cualquier nivel
**/temp/

# En raíz solamente
/TODO

# Comentario
# esto es un comentario

# Negación
!importante.txt
```

5

Buenas Prácticas

Commits Atómicos

Concepto

Un **commit atómico** contiene un único cambio lógico y completo.

Commits Atómicos

Concepto

Un **commit atómico** contiene un único cambio lógico y completo.

Bien

- Un bug, un commit
- Una feature, uno o más commits
- Cada commit compila
- Mensajes descriptivos

Commits Atómicos

Concepto

Un **commit atómico** contiene un único cambio lógico y completo.

Bien

- Un bug, un commit
- Una feature, uno o más commits
- Cada commit compila
- Mensajes descriptivos

Mal

- Múltiples features en un commit
- Commit con código roto
- Mensajes vagos: "fix", "cambios"
- Commits gigantes

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día
2. **Usa Conventional Commits** — Historial legible y profesional

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día
2. **Usa Conventional Commits** — Historial legible y profesional
3. **Revisa antes de commitear** — Usa `git diff` y `git status`

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día
2. **Usa Conventional Commits** — Historial legible y profesional
3. **Revisa antes de commitear** — Usa `git diff` y `git status`
4. **No subas archivos generados** — Usa .gitignore correctamente

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día
2. **Usa Conventional Commits** — Historial legible y profesional
3. **Revisa antes de commitear** — Usa `git diff` y `git status`
4. **No subas archivos generados** — Usa .gitignore correctamente
5. **Nunca subas secretos** — Contraseñas, tokens, API keys fuera

Reglas de Oro para Commits

1. **Commit frecuentemente** — No esperes al final del día
2. **Usa Conventional Commits** — Historial legible y profesional
3. **Revisa antes de commitear** — Usa `git diff` y `git status`
4. **No subas archivos generados** — Usa .gitignore correctamente
5. **Nunca subas secretos** — Contraseñas, tokens, API keys fuera
6. **Escribe mensajes claros** — Piensa en tus compañeros

Checklist Antes de Commitear

Antes de hacer commit:

- ☐ ¿El código compila? (`mvn compile`)
- ☐ ¿Revisé los cambios? (`git diff`)
- ☐ ¿Agregué solo lo necesario? (`git status`)
- ☐ ¿El mensaje es descriptivo?
- ☐ ¿Sigo Conventional Commits?
- ☐ ¿No hay archivos generados?
- ☐ ¿No hay secretos/contraseñas?

> *Commit early, commit often.*
>
> — Git Best Practices

6

Resumen

Lo que Aprendimos

Estados de archivos:

- Untracked
- Staged
- Committed / Unmodified
- Modified

Conventional Commits:

- Tipos: feat, fix, docs, etc.
- Estructura clara
- Breaking changes

Estructura de proyecto:

- Maven estándar
- pom.xml
- README.md

.gitignore:

- Ignorar compilados
- Ignorar IDEs
- Usar gitignore.io
- Patrones comunes

Comandos Clave


```
# Estados
git status

# Staging
git add archivo.txt
git restore --staged archivo.txt

# Commits
git commit -m "feat:  nueva funcionalidad"

# Diferencias
git diff
git diff --staged

# Descartar cambios
git restore archivo.txt
```

Recursos

Conventional Commits:

- https://www.conventionalcommits.org
- Especificación completa
- Ejemplos y guías

Maven:

- https://maven.apache.org/guides/
- Maven in 5 Minutes
- POM Reference

.gitignore:

- https://www.toptal.com/developers/gitignore
- Templates oficiales
- GitHub gitignore repo

Próxima Clase

Historial (git log). Diferencias (git diff avanzado). Restauración (checkout/restore). Etiquetas (tags) para versiones.

¡Gracias!

Nos vemos en la próxima clase