

# GitLab Worflow

Proyectos ST web

ST Idea

Marcos Anguís

4 de Agosto 2025

**I N D I C E**

---

1.	ABREVIATURAS Y CÓDIGOS.....	1
2.	OBJETO DEL PROYECTO.....	2
3.	ARQUITECTURA DE RAMAS.....	2
3.1.	ROLES Y RESPONSABILIDADES .....	2
3.2.	RAMAS PRINCIPALES.....	2
3.3.	RAMAS DE TRABAJO .....	3
4.	FLUJO DE DESARROLLO (DEVELOP/FEATURE) .....	3
5.	PROCESO DE APROBACIÓN .....	4
5.1.	FLUJO DE APROBACIÓN .....	4
5.2.	REGLAS DE APROBACIÓN POR RAMA .....	5
6.	REGLAS DE PROTECCIÓN.....	5
6.1.	CONFIGURACIÓN DE RAMAS PROTEGIDAS.....	5
6.2.	REGLAS DE PUSH.....	5
6.3.	EXPRESIONES REGULARES .....	6
A.	Nombres de Rama Permitidos .....	6
B.	Emails de Autor Permitidos .....	6
C.	Archivos Prohibidos.....	6
D.	Commits .....	6
7.	VERSIONADO Y COMMITS.....	7
7.1.	SISTEMA DE VERSIONADO SEMÁNTICO.....	7
7.2.	VERSIONES POR RAMA .....	7
7.3.	COMMITS VÁLIDOS .....	7
A.	Commits de Features (cambios normales de aplicativo “MINOR”) .....	7
B.	Commits de Fixes ( Cambios por fallo o bug de aplicativo “PATCH”).....	8
C.	Commits de Breaking Changes .....	8
7.4.	AUTOMATIZACIÓN DEL VERSIONADO .....	9
8.	COMANDOS GIT ESENCIALES .....	10
A.	Configuración Inicial .....	10
B.	Flujo de Trabajo Diario .....	10
C.	Comandos Útiles .....	11
9.	MEJORES PRÁCTICAS .....	12
10.	TROUBLESHOOTING.....	12
11.	DIAGRAMA DE COMPONENTES (ARQUITECTURA) .....	13

## 1. Abreviaturas y códigos

Con el fin de simplificar las referencias a personas, servicios y productos en el resto del documento, utilizaremos en adelante las siguientes abreviaturas que aquí se mencionan.

Personas:

- **LAR:** Luis Antonio Rosich
- **ML:** Mariano López
- **AG:** Alejandro Gonzalez
- **MA:** Marcos Anguís

### **Git y Repositorio**

- **MR** – Merge Request
- **PR** – Pull Request (término de GitHub, equivalente a MR)
- **SHA** – Secure Hash Algorithm (hash único de cada commit)
- **WIP** – Work in Progress
- **LGTM** – Looks Good To Me
- **HEAD** – Último commit de la rama actual
- **Rebase** – Reaplicar commits sobre otra base (no es una sigla, pero es común)

### **GitLab CI/CD y Pipelines**

- **CI** – Continuous Integration (Integración continua)
- **CD** – Continuous Delivery / Continuous Deployment (Entrega o despliegue continuo)
- **YAML** – YAML Ain't Markup Language (formato de configuración usado por GitLab CI)
- **Job** – Trabajo individual dentro de una etapa del pipeline
- **Stage** – Etapa del pipeline que agrupa varios jobs
- **Runner** – Servicio que ejecuta los jobs definidos en GitLab
- **Artifacts** – Archivos generados por los jobs (como binarios, informes, etc.)
- **Cache** – Archivos guardados para acelerar ejecuciones futuras
- **ENV** – Environment (ambiente de despliegue: producción, staging, etc.)
- **VAR** – Variable (variable de entorno)

### **Testing y Automatización**

- **TDD** – Test-Driven Development (Desarrollo guiado por pruebas)
- **BDD** – Behavior-Driven Development (Desarrollo guiado por comportamiento)
- **QA** – Quality Assurance (Aseguramiento de calidad)
- **UT** – Unit Test (Prueba unitaria)
- **IT** – Integration Test (Prueba de integración)



## 2. Objeto del proyecto

El objetivo de esta documentación es **estandarizar, centralizar y facilitar el entendimiento del flujo de trabajo en GitLab** para todos los roles involucrados en el desarrollo de los proyectos en ST. Proporciona una guía completa y accesible sobre los procesos técnicos y organizativos, incluyendo arquitectura de ramas, reglas de protección, buenas prácticas, configuración de CI/CD, y documentación visual (diagramas UML).

Está diseñada para:

- **Asegurar la coherencia del trabajo colaborativo** dentro del equipo de desarrollo.
- **Facilitar la incorporación de nuevos integrantes** mediante guías rápidas y diagramas visuales.
- **Apoyar la toma de decisiones técnicas y de calidad** mediante definiciones claras de roles, procesos y configuraciones.

En resumen, esta documentación busca **garantizar un desarrollo eficiente, organizado y alineado con los estándares del proyecto**.

## 3. Arquitectura de ramas

### 3.1. Roles y Responsabilidades

- **Desarrolladores:** Crean features, bugfixes y hotfixes
- **QA Leaders:** Aproban cambios en ramas test y release
- **Development Leaders:** Aproban cambios en rama develop
- **Product Owner:** Aproba cambios en rama main

### 3.2. Ramas principales

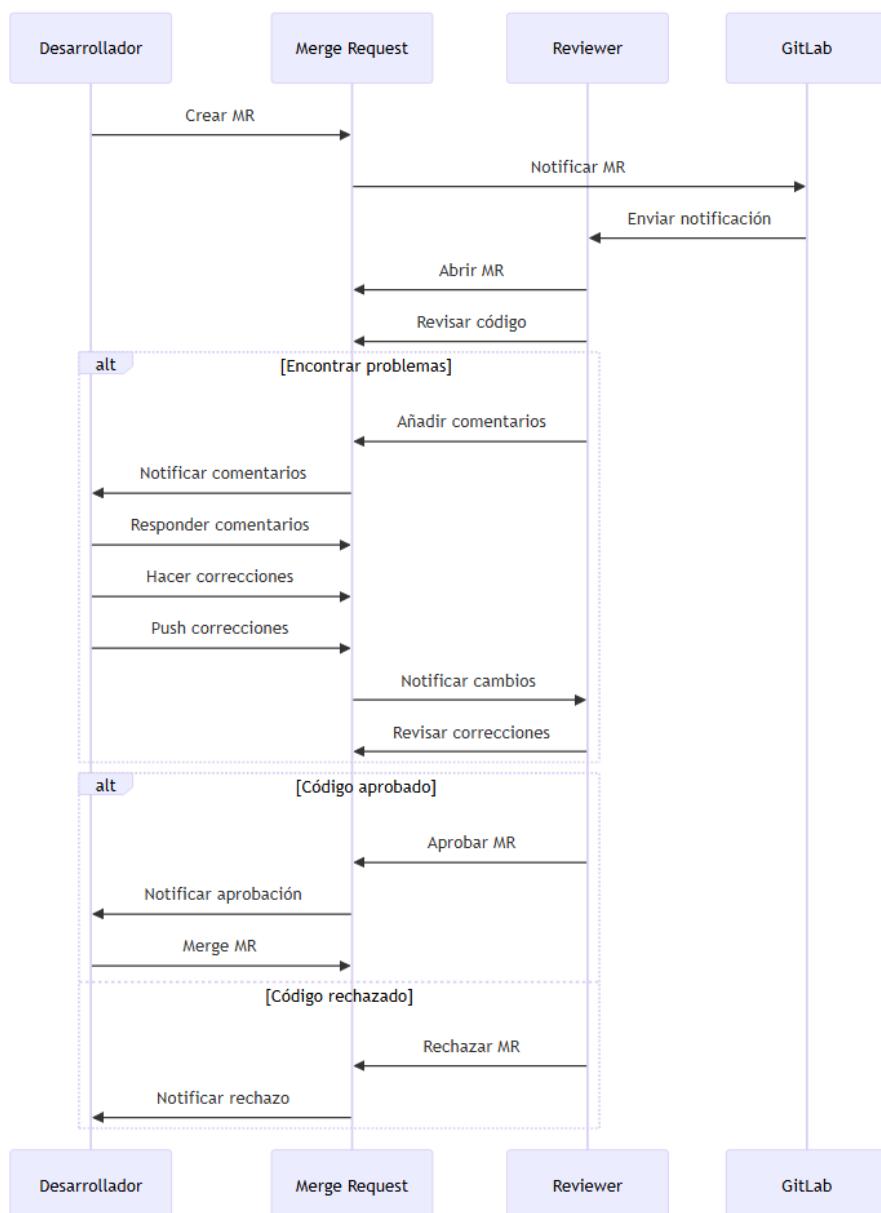
```
main (producción)
  |-- release (pre-producción)
  |-- test (testing)
  |-- develop (desarrollo)
```



### 3.3. Ramas de trabajo

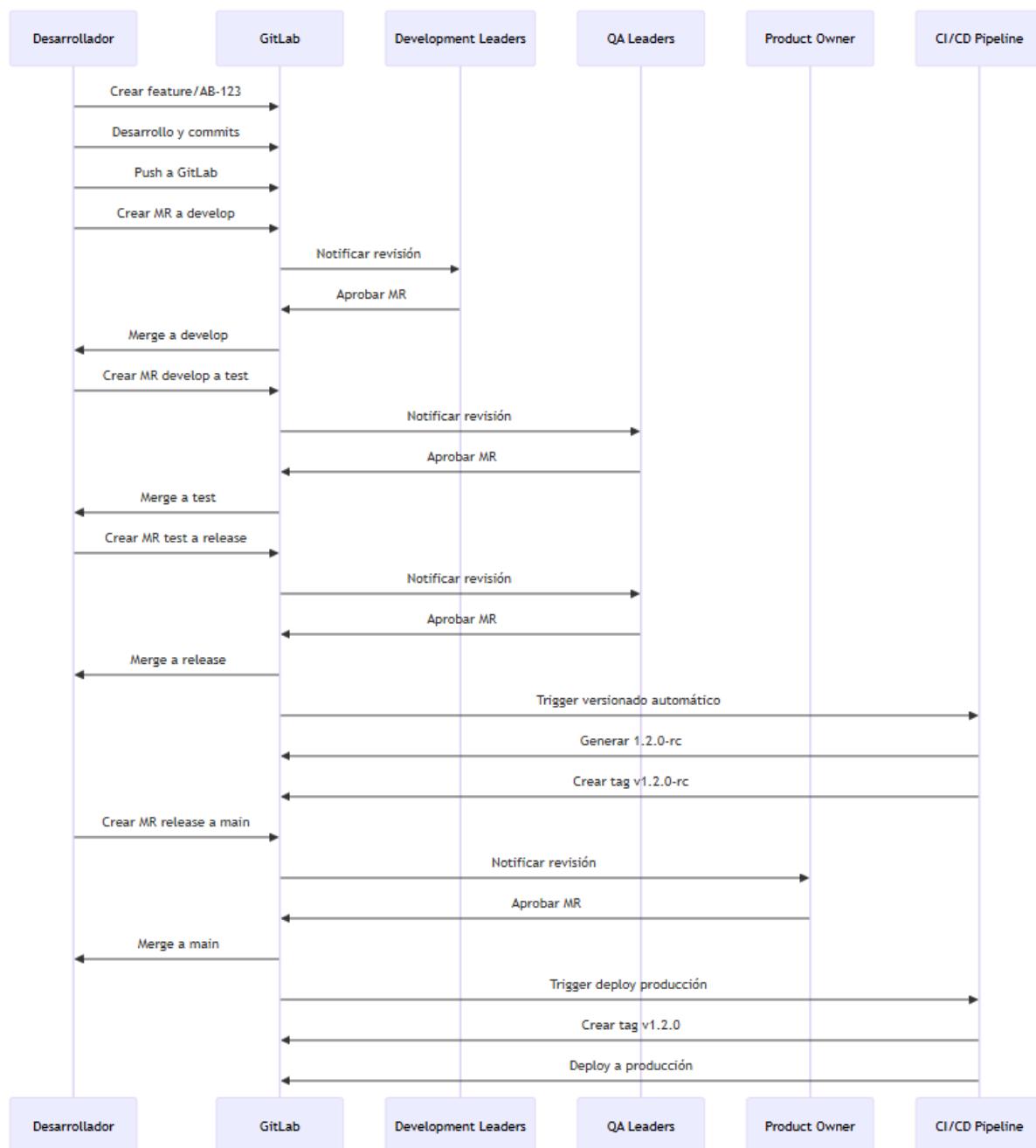
feature/*	→ develop
bugfix/*	→ develop
hotfix/*	→ main

## 4. Flujo de desarrollo (Develop/feature)



## 5. Proceso de aprobación

### 5.1. Flujo de aprobación



## **5.2. Reglas de Aprobación por Rama**

Rama	Aprobadores	Aprobaciones Requeridas
<b>develop</b>	Development Leaders	1
<b>test</b>	QA Leaders	1
<b>release</b>	QA Leaders	1
<b>main</b>	Product Owner	1

## **6. Reglas de protección**

### **6.1. Configuración de Ramas Protegidas**

Rama	Merge	Push	Force Push	Code Owner
<b>develop</b>	Developers Maintainers	No one	X	✓
<b>test</b>	Developers Maintainers	No one	X	✓
<b>release</b>	Maintainers	No one	X	✓
<b>main</b>	Maintainers	No one	X	✓

### **6.2. Reglas de Push**

- **Usuarios verificados:** Solo usuarios con emails verificados.
- **Verificar autor GitLab:** Restringir commits a usuarios GitLab existentes.
- **Prevenir archivos secretos:** Rechazar archivos que puedan contener secretos.
- **Commits no firmados:** Permitido.
- **Commits sin DCO:** Permitido.



## 6.3. Expresiones regulares

### A. Nombres de Rama Permitidos

---

(develop|test|release|main)|(feature|bugfix|hotfix)\/[A-Z][A-Z]+-\d+([-\_.]\*)?)

**Patrón:** {tipo}/{ticket}-{descripción}

**Tipos:**

- feature/ - Nuevas funcionalidades
- bugfix/ - Corrección de bugs
- hotfix/ - Correcciones urgentes

**Ejemplos:**

- feature/AB-123-nueva-pantalla-login
- bugfix/CD-456-corrige-error-autenticacion
- hotfix/EF-789-correccion-critica-produccion

### B. Emails de Autor Permitidos

---

@stidea\.com\$

### C. Archivos Prohibidos

---

(exe|msi|jar|zip|tar\gz)\$

### D. Commits

---

**Formato requerido:**

^(feat|feature|fix|bug|BREAKING|breaking|major|MAJOR)(\([a-z-]+\))?:\s+.+

**Ejemplos válidos:**

- feat: implementa nueva funcionalidad
- fix(auth): corrige validación de token
- BREAKING: cambia estructura de base de datos



## 7. Versionado y commits

### 7.1. Sistema de versionado semántico

El proyecto utiliza Semantic Versioning (SemVer) con el formato MAJOR.MINOR.PATCH:

- MAJOR: Cambios incompatibles con versiones anteriores
- MINOR: Nuevas funcionalidades compatibles
- PATCH: Correcciones de bugs compatibles

### 7.2. Versiones por rama

- develop → 1.2.0-develop-202401151430
- test → 1.2.0-beta
- release → 1.2.0-rc
- main → 1.2.0

**Variables generadas automáticamente:**

- VERSION\_TAG - Versión completa (ej: 1.2.0-develop-202401151430)
- VERSION\_TAG\_SHORT - Versión corta (ej: 1.2.0-develop)
- IMAGE\_TAG\_SRC - Hash del commit fuente
- APP\_VERSION - Versión base de la aplicación

### 7.3. Commits válidos

**Regex de validación:**

`^(feat|feature|fix|bug|BREAKING|breaking|major|MAJOR)(\([a-z]+\))?:\s+.+`

#### A. Commits de Features (cambios normales de aplicativo “MINOR”)

```
git commit -m "feat: implementa nueva pantalla de login"
git commit -m "feature: agrega validación de formularios"
git commit -m "feat(auth): mejora sistema de autenticación"
```



## **B. Commits de Fixes ( Cambios por fallo o bug de aplicativo “PATCH”)**

---

```
git commit -m "fix: corrige error en cálculo de totales"  
git commit -m "bug: resuelve problema de memoria"  
git commit -m "fix(ui): arregla alineación de botones"
```

## **C. Commits de Breaking Changes**

---

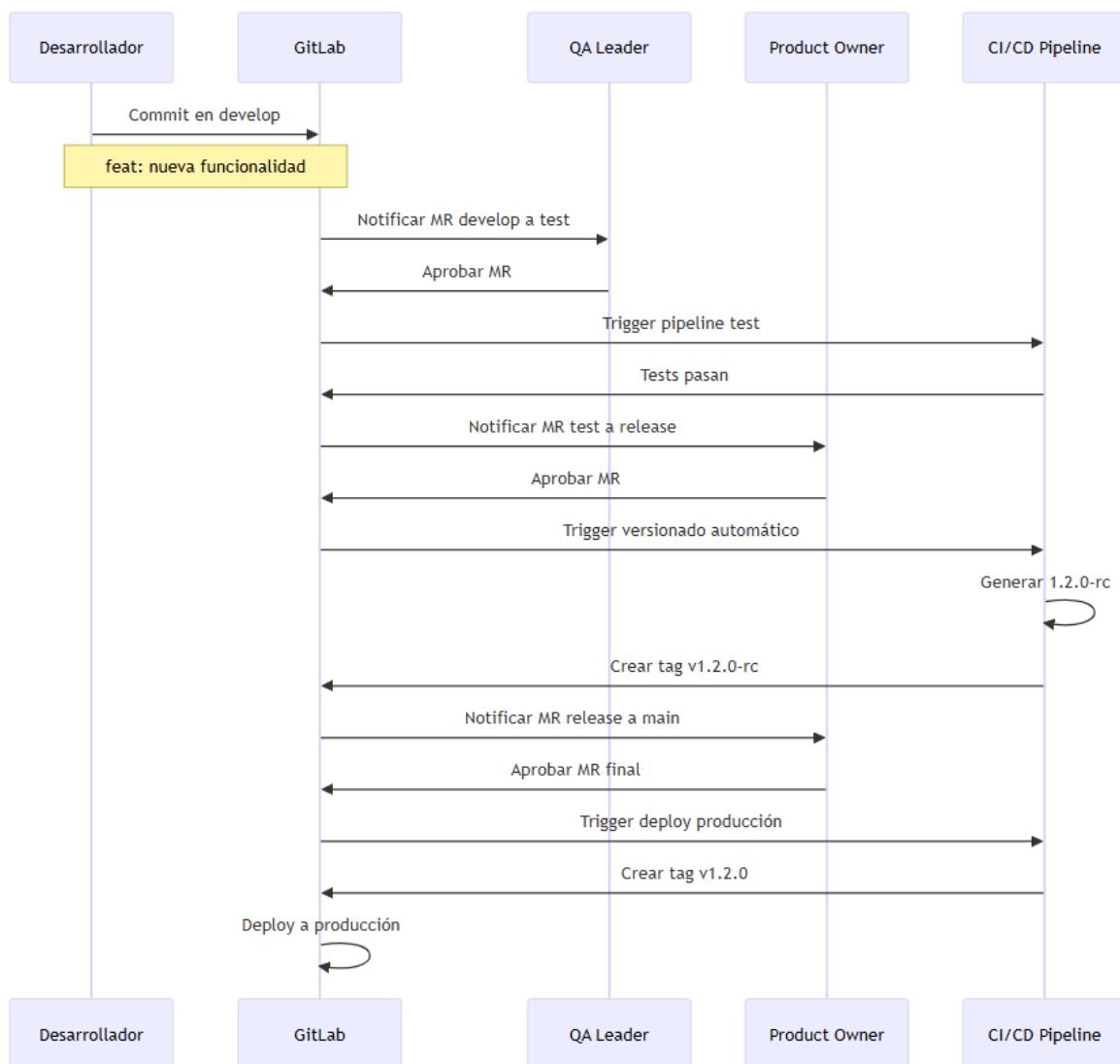
```
git commit -m "BREAKING: cambia estructura de API"  
git commit -m "breaking: modifica formato de respuesta"  
git commit -m "major: actualiza versión de framework"
```



## 7.4. Automatización del versionado

El versionado se maneja automáticamente en GitLab:

- Desarrollador → Solo hace commits en develop
- QA → Aprueba MR develop → test
- Product Owner → Aprueba MR test → release
- Sistema → Versionado automático en release
- Product Owner → Aprueba MR release → main



## 8. Comandos GIT esenciales

### A. *Configuración Inicial*

```
# Configurar usuario
git config --global user.name "Tu Nombre"
git config --global user.email "tu.email@stidea.com"

# Clonar repositorio
git clone <url-del-repositorio>
cd <nombre-del-proyecto>
```

### B. *Flujo de Trabajo Diario*

```
# 1. Actualizar rama principal
git checkout develop
git pull origin develop

# 2. Crear rama de trabajo
git checkout -b feature/AB-123-nueva-funcionalidad

# 3. Desarrollo
# ... hacer cambios ...

# 4. Commit con formato correcto
git add .
git commit -m "feat: implementa nueva funcionalidad"

# 5. Push a GitLab
git push origin feature/AB-123-nueva-funcionalidad

# 6. Crear Merge Request desde GitLab UI
# Target: develop
```



## C. Comandos Útiles

```
# Ver estado del repositorio
git status

# Ver historial de commits
git log --oneline

# Ver diferencias
git diff

# Descartar cambios
git checkout -- .

# Cambiar de rama
git checkout <nOMBRE-rama>

# Ver todas las ramas
git branch -a

# Eliminar rama local
git branch -d <nOMBRE-rama>

# Sincronizar con remoto
git fetch origin
git pull origin <rama>
```



## **9. Mejores prácticas**

### A. Para Desarrolladores

- Siempre actualiza tu rama base antes de crear un MR
- Usa mensajes de commit descriptivos y siguiendo el formato
- Revisa tu propio código antes de solicitar review
- Responde rápidamente a los comentarios de review
- Mantén los MR pequeños y enfocados

### B. Para Reviewers

- Revisa el código de manera constructiva
- Proporciona feedback específico y accionable
- Aproba solo cuando estés satisfecho con la calidad
- Usa las herramientas de GitLab para comentarios

### C. Para Maintainers

- Mantén las reglas de protección actualizadas
- Monitorea el estado de los pipelines
- Resuelve conflictos de merge cuando sea necesario
- Mantén la documentación actualizada

## **10. Troubleshooting**

### **Problemas Comunes**

#### **Error: “Branch name does not match pattern”**

- Verifica que el nombre de la rama siga el patrón: feature/AB-123

#### **Error: “Commit message does not match pattern”**

- Asegúrate de que el mensaje de commit incluya: Fixes 123.

#### **Error: “Author email not allowed”**

- Verifica que tu email termine en @stidea.com

#### **Error: “File rejected - contains secrets”**

- Revisa que no estés subiendo archivos con información sensible



## 11. Diagrama de componentes (arquitectura)

