|  |  |  |
| --- | --- | --- |
| Control de cambios: | | |
| Revisión | Fecha | Observaciones |
| 00 | 23/10/2024 | Documento inicial |

ÍNDICE

[1 Objeto: 3](#__RefHeading___Toc1514_3823752326)

[2 Campo de aplicación: 3](#__RefHeading___Toc1516_3823752326)

[3 Definiciones: 4](#__RefHeading___Toc1518_3823752326)

[4 Responsabilidades: 6](#__RefHeading___Toc1520_3823752326)

[4.1 Equipo de desarrollo 6](#__RefHeading___Toc1522_3823752326)

[5 Instrucciones 7](#__RefHeading___Toc1671_3823752326)

[5.1 Uso de Git 7](#__RefHeading___Toc1673_3823752326)

[5.1.1 Instalación y configuración de Git 7](#__RefHeading___Toc1675_3823752326)

[5.1.2 Clonar un repositorio 9](#__RefHeading___Toc1677_3823752326)

[5.1.3 Crear ramas para nuevas funcionalidades y correcciones 9](#__RefHeading___Toc1679_3823752326)

[5.1.4 Realizar commits y mantener un historial limpio 10](#__RefHeading___Toc1681_3823752326)

[5.2 Uso de GitLab 11](#__RefHeading___Toc1683_3823752326)

[5.2.1 Creación de proyectos en GitLab 11](#__RefHeading___Toc1685_3823752326)

[5.2.2 Configuración de ramas protegidas (main, develop) 12](#__RefHeading___Toc1687_3823752326)

[5.2.3 Gestión de Merge Requests 13](#__RefHeading___Toc1689_3823752326)

[5.2.4 Uso de issues para bugs y evolutivos 14](#__RefHeading___Toc1691_3823752326)

[5.2.5 Gestión de milestones y seguimiento de versiones 15](#__RefHeading___Toc1693_3823752326)

[5.2.6 Publicación de versiones con tags 16](#__RefHeading___Toc1695_3823752326)

[Anexo1: Flujo de trabajo con ramas 17](#__RefHeading___Toc1562_3823752326)

[¿Cómo trabajar con las ramas? 17](#__RefHeading___Toc1600_3823752326)

[Anexo 2. Comandos básicos 20](#__RefHeading___Toc1603_3823752326)

# Objeto:

Este documento propone una forma de establecer un sistema eficiente de control de versiones mediante el uso de Git y GitLab en la empresa. Esto permitirá gestionar de manera centralizada los proyectos de software, garantizando la integridad del código, facilitando la colaboración entre los equipos y asegurando una trazabilidad clara en la evolución del software. Con esta guía, se busca optimizar el control de versiones, la gestión de cambios, la resolución de bugs y la implementación de nuevas funcionalidades, mejorando el flujo de trabajo en el desarrollo y mantenimiento de aplicaciones.

# Campo de aplicación:

Git y GitLab se aplican en todos los proyectos de desarrollo de software de la empresa, incluyendo la implementación de nuevas funcionalidades, correcciones de errores, actualizaciones evolutivas y mantenimiento del código.

Esta herramienta será utilizada principalmente por los equipos de desarrollo y calidad de Termosalud.

# Definiciones:

* **Git:** Sistema que permite gestionar los cambios realizados en el código fuente de manera distribuida, facilitando la colaboración entre varios desarrolladores.
* **Repositorio local:** Almacenamiento de código en la máquina del desarrollador.
* **Repositorio remoto:** Copia centralizada accesible para todos los miembros del equipo.
* **Ramas (**Branches**):** master**,** develop**,** feature**,** bugfix  
  Las ramas son versiones separadas del código. La rama main contiene el código en producción, develop es el código en desarrollo, feature se usa para nuevas funcionalidades, y bugfix para la corrección de errores.
* **Commits: Acción de guardado de cambios realizados en el repositorio local que crea un punto de control en el historial del proyecto.**
* **GitLab:** plataforma web que permite gestionar repositorios de Git de forma centralizada. Ofrece herramientas para el control de versiones, la creación de Merge Requests, la gestión de issues y la planificación de milestones.
* **Merge Request/Solicitud de fusión: Petición, desde una plataforma web, de fusión de ramas que permite pasar los cambios de una rama con otra, normalmente de** feature **o** bugfix **a** develop **y de hotfix o develop a** main**, tras una revisión de código.**
* ****Tag:** Marcador que señala un estado específico del código, usado para indicar la versión oficial del proyecto (ej. v1.0.4).**
* ****Issue/Ticket:** Una tarea, bug o mejora sobre la que se debe trabajar. Las** tareas **ayudan a organizar y asignar el trabajo dentro del equipo.**
* ****Milestone/Hito:** Conjunto de** tareas **y objetivos que se agrupan bajo un *hito* específico, utilizado para planificar entregas o versiones importantes dentro de un proyecto.**

# Responsabilidades:

## Equipo de desarrollo

* **Uso correcto de Git y GitLab para versionado y trabajo en ramas  
  Seguir el flujo de trabajo establecido**
* **Realización de *commits* claros y bien documentados  
  Escribir mensajes descriptivos que indiquen claramente los cambios realizados asociándolos, si corresponde a una tarea**
* **Revisión de *Merge Requests* y validación de soluciones**  
  Revisar los Merge Requests de otros compañeros, validar que los cambios cumplan con los requisitos y que no haya errores antes de fusionar las ramas.

# Instrucciones

## Uso de Git

### Instalación y configuración de Git

Descargar desde [git-scm.com](https://git-scm.com/downloads) e instalar.

Tras la instalación debes seguir los siguientes pasos:

1. Configurar nombre y correo:

# git config --global user.name "Tu Nombre"  
# git config --global user.email "email@termosalud.com"

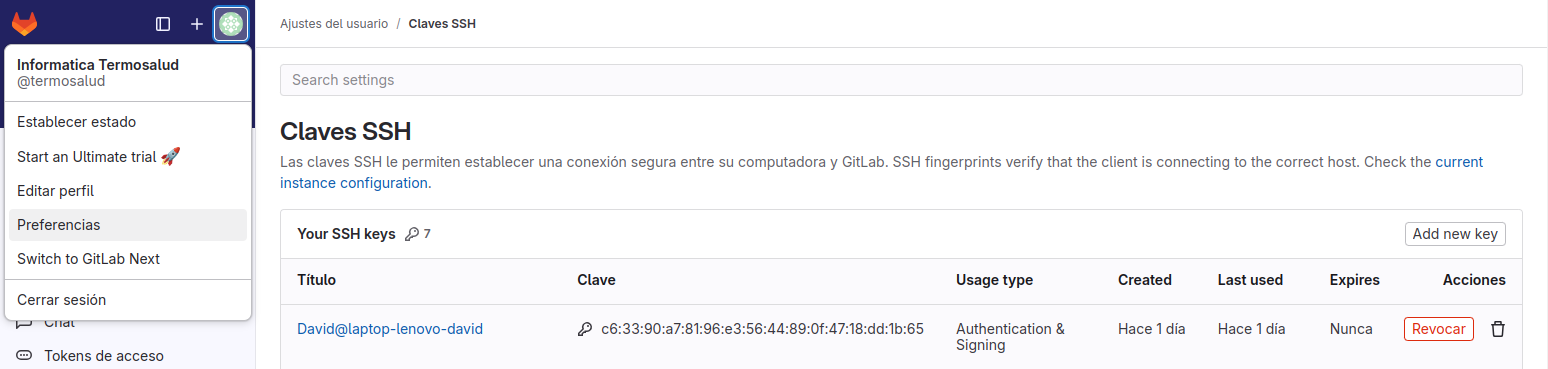
2. Crear la clave SSH:

Las claves SSH se usan para conectarse a servidores remotos de forma segura usando el método clave pública clave privada donde la primera actúa como un candado y la segunda como una llave.

# ssh-keygen

3. Añadir la clave SSH a GitLab

Para que GitLab nos permita hacer push/pull y acceder con SSH debemos añadir la clave pública a nuestro usuario de GitLab.

En la parte superior izquierda encontraremos nuestro perfil y haciendo click sobre preferencias accederemos a nuestro perfil.

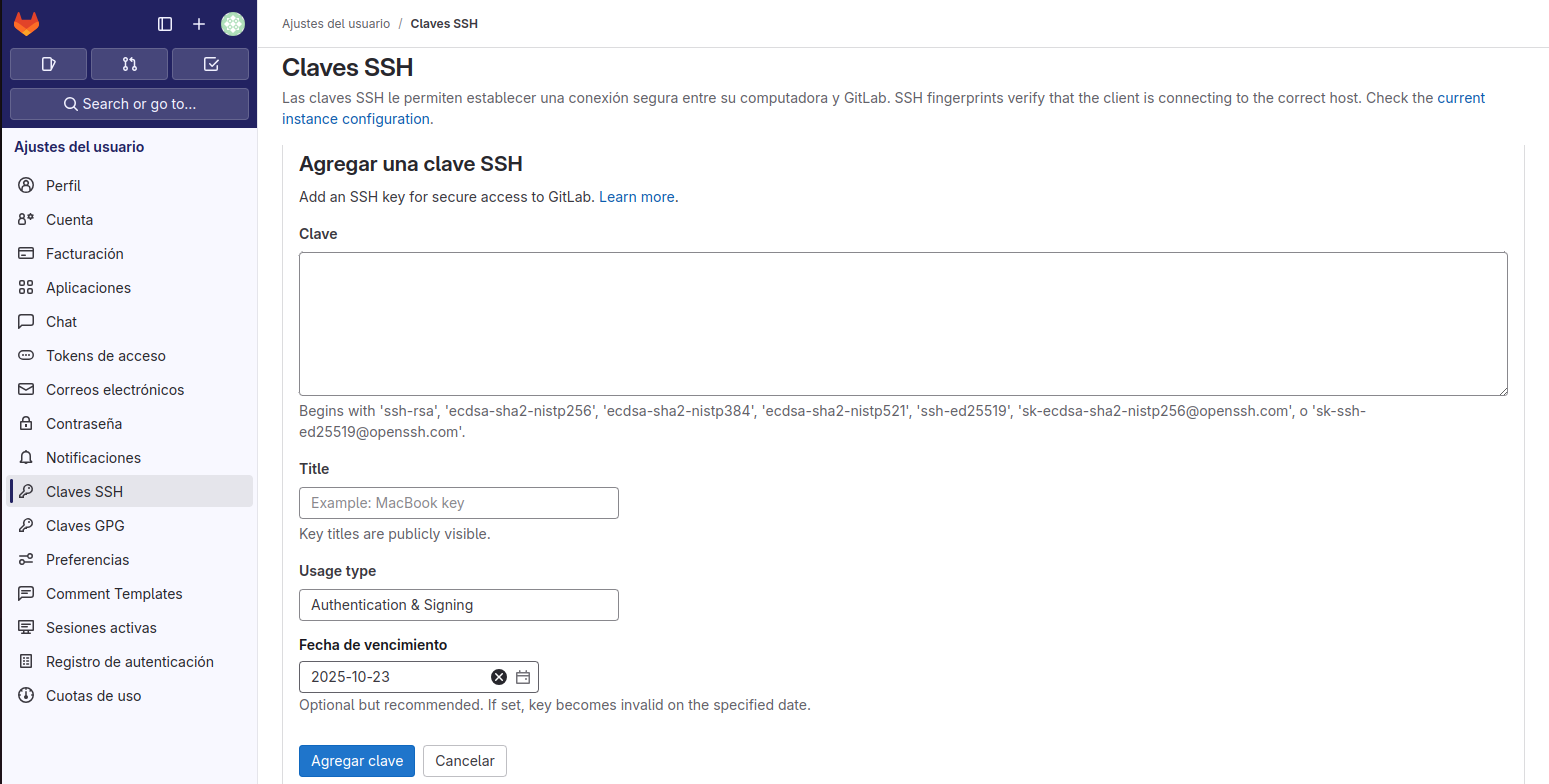
En el menú lateral izquierdo haremos click sobre Claves SSH.

Para extraer nuestra clave pública SSH deberemos ir a nuestra carpeta .ssh que estará dentro de nuestra carpeta ~ o Documentos.

# cat ~/.ssh/id\_rsa.pub

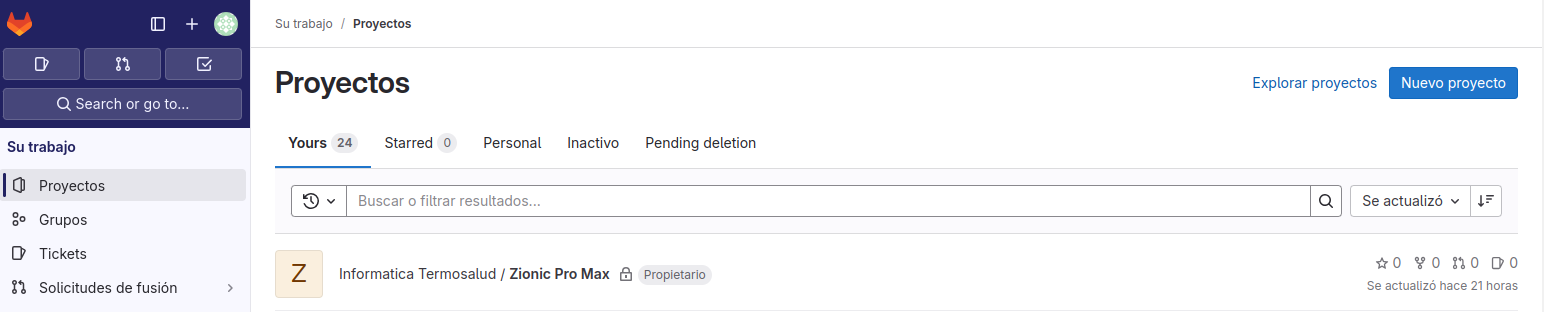
La clave puede haberse generado con otro nombre, es importante que cojamos el texto que sale en el fichero .pub.

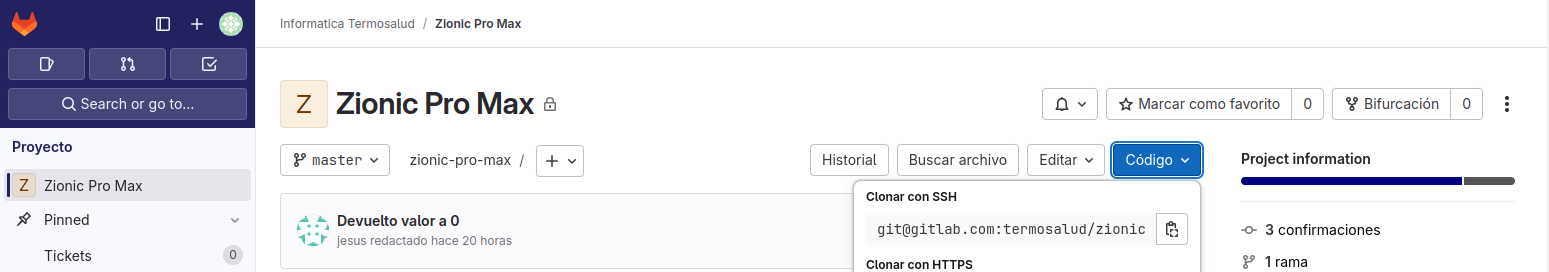
Añadiremos a la siguiente pantalla la clave y haremos click en Agregar clave

Cuando hayamos cumplimentado este formulario tendremos acceso a GitLab usando SSH y podremos actuar sobre los repositorios de los que seamos miembros.

### Clonar un repositorio

Entraremos en GitLab a buscar el repositorio

Una vez dentro del proyecto, encontraremos un botón que nos permite seleccionar la url con la que clonar el repo. Siempre usaremos ssh



Para clonar debemos colocarnos en la carpeta que queramos descargar el proyecto y desde la consola:

#git clone git@gitlab.com:termosalud/zionic-pro-max.git

### Crear ramas para nuevas funcionalidades y correcciones

Para mantener un orden y como explicaremos en el [flujo de trabajo con ramas](#_toc311) vamos a crear ramas para las nuevas funcionalidades y para la corrección de bugs.

**1. Crear una nueva rama desde** develop **para nuevas funcionalidades.**

**# git checkout develop   
# git pull origin develop   
# git checkout -b feature/<nueva-funcionalidad>**

2. Crear una nueva rama para corregir un bug desde develop o main

**# git checkout develop   
# git pull origin develop   
# git checkout -b bugfix/<correccion-bug>**

### Realizar commits y mantener un historial limpio

El historial del proyecto debe ser claro y conciso, para esto debemos evitar agrupar varios cambios por commit. Es vital que hagamos un commit por cada cambio pequeño y significativo con un comentario descriptivo.

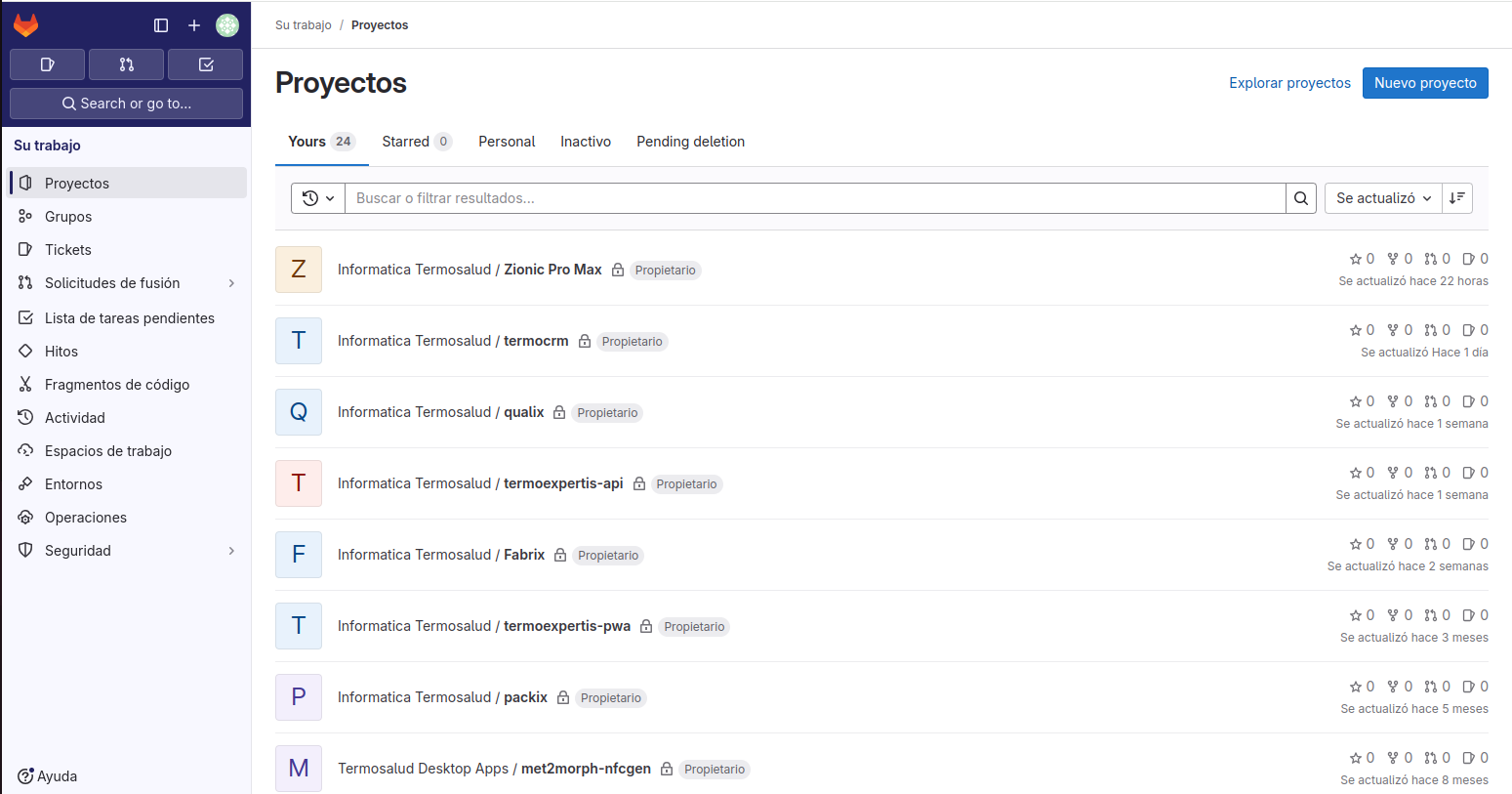
Para realizar un commit debemos añadir los cambios al *área de preparación (staging area)*

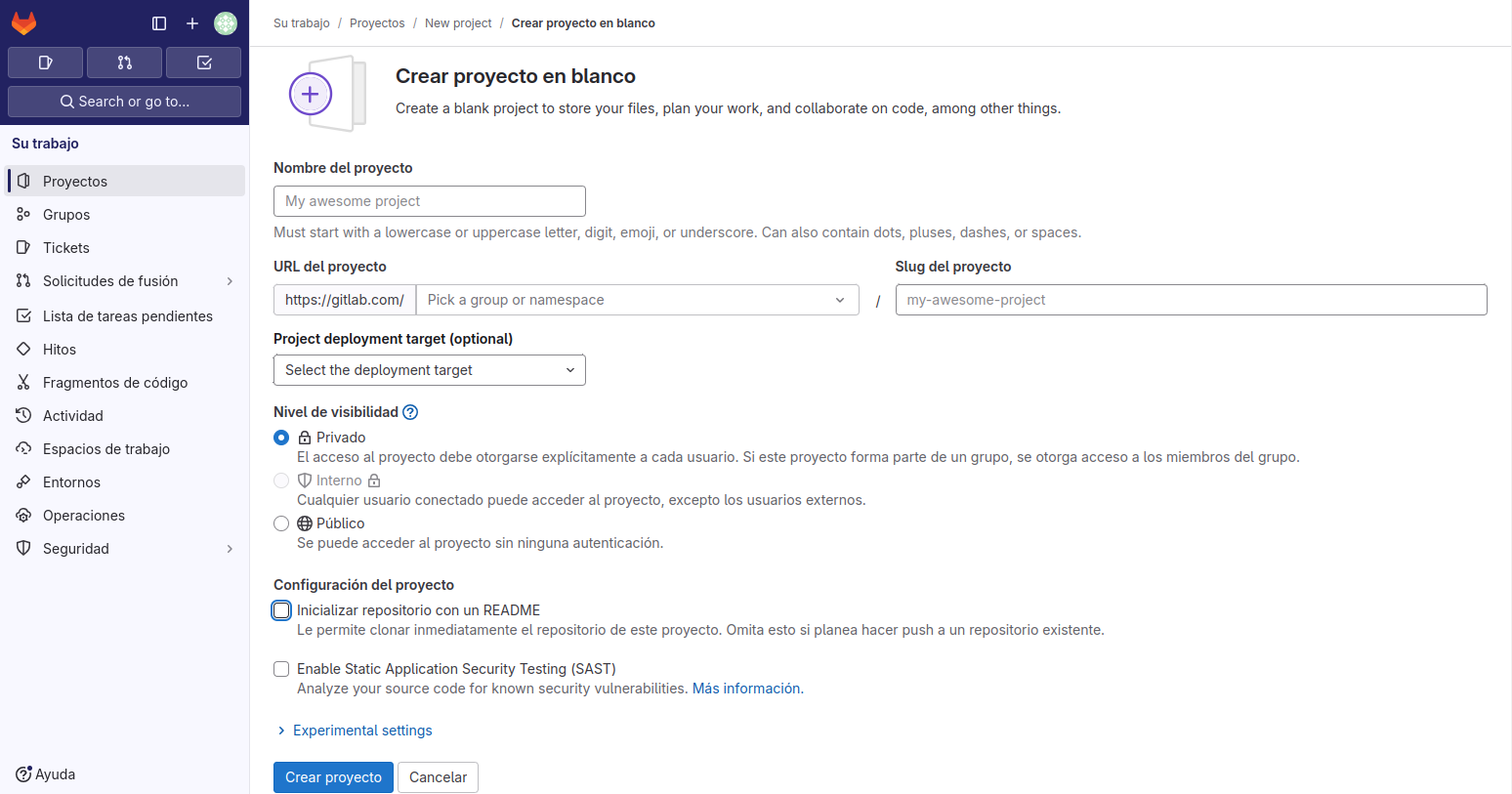
**# git add .  
# git commit -m “Descripción clara y concisa de los cambios realizados”**

## Uso de GitLab

### Creación de proyectos en GitLab

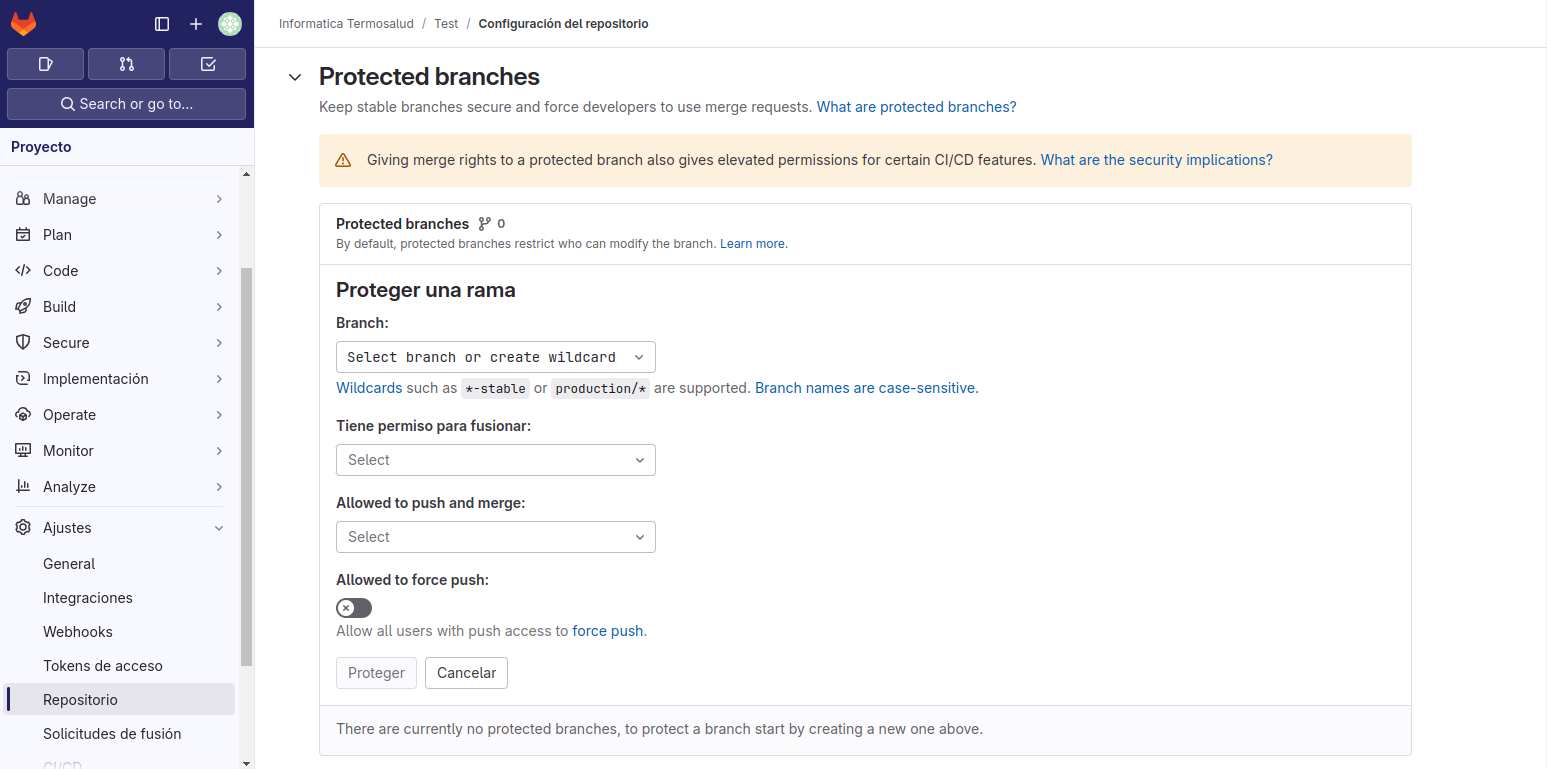
Accedemos a nuestra cuenta en GitLab, veremos nuestros proyectos:

Haciendo click sobre el botón accederemos a la página de creación de proyectos.

Aquí configuraremos el nombre del proyecto y la visibilidad del mismo. Lo normal es que todos nuestros proyectos sean privados.

### Configuración de ramas protegidas (main, develop)

Dentro del proyecto **Ajustes > Repositorios** encontramos el siguiente panel:

Aquí vamos a proteger la rama main/master para evitar *push* directo, permitiendo solo *Merge Requests*

También protegeremos la rama develop para asegurarnos de que los cambios pasen por revisiones antes de integrarse.

Asignaremos permisos a los tipos de usuarios que pueden hacer merge y push.

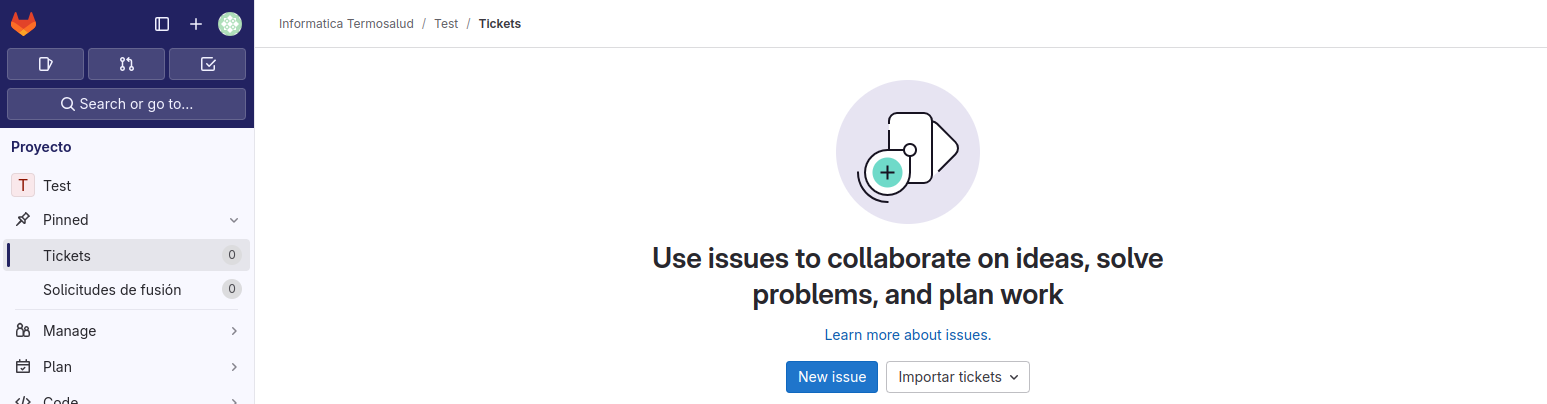
### Gestión de Merge Requests

**Crear un Merge Request (MR)**:  
Después de realizar cambios en una rama, desde el panel de GitLab, selecciona **Merge Requests** > **New Merge Request**.

* Selecciona la rama de origen (ej. feature/<nueva-funcionalidad>) y la de destino (ej. develop).
* **Asignar revisores y etiquetar**:  
  Añadiremos, si corresponde, uno o varios revisores. Describiremos el propósito de la solicitud de fusión y los issues que resuelve.
* **Resolver conflictos y hacer el merge**:  
  Si el código está aprobado y no hay conflictos, procede a hacer el merge.

### Uso de issues para bugs y evolutivos

Usaremos los issues o tickets en GitLab para gestionar tanto correcciones de bugs como nuevas funcionalidades.

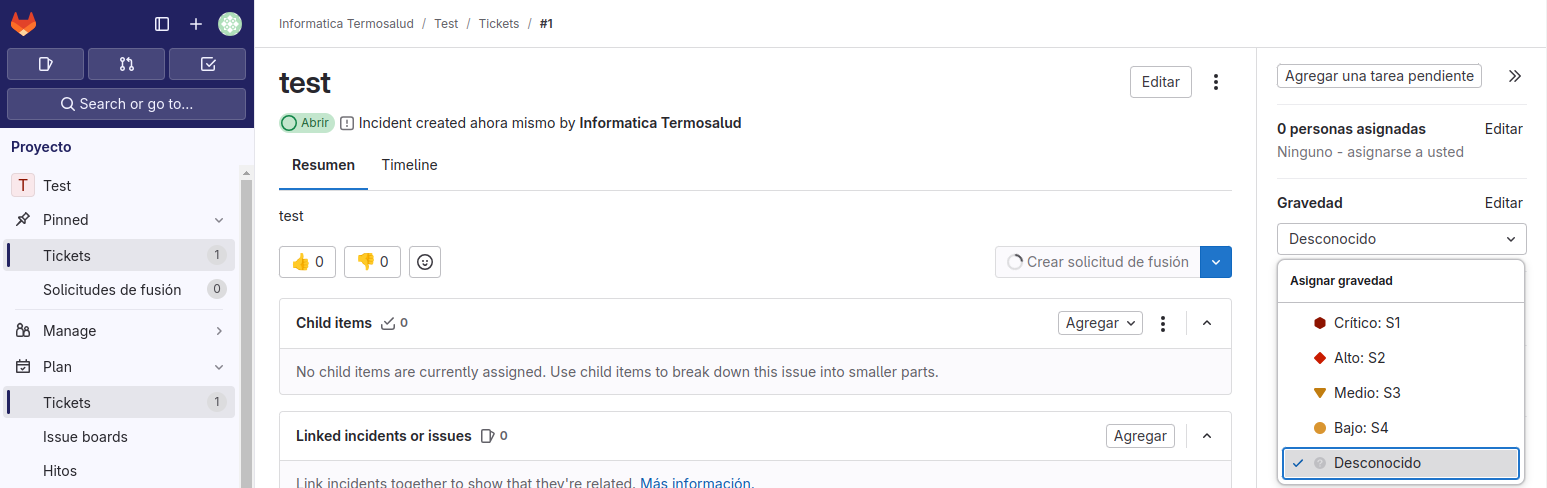


**Crear un *Issue/Ticket***

Cada bug o funcionalidad debe ser registrado como un issue detallado, incluyendo una descripción clara del error u objetivos claros del evolutivo.

**Asignar y priorizar issues**

Los issues deben ser asignados a los responsables de su resolución, y priorizados según la criticidad.

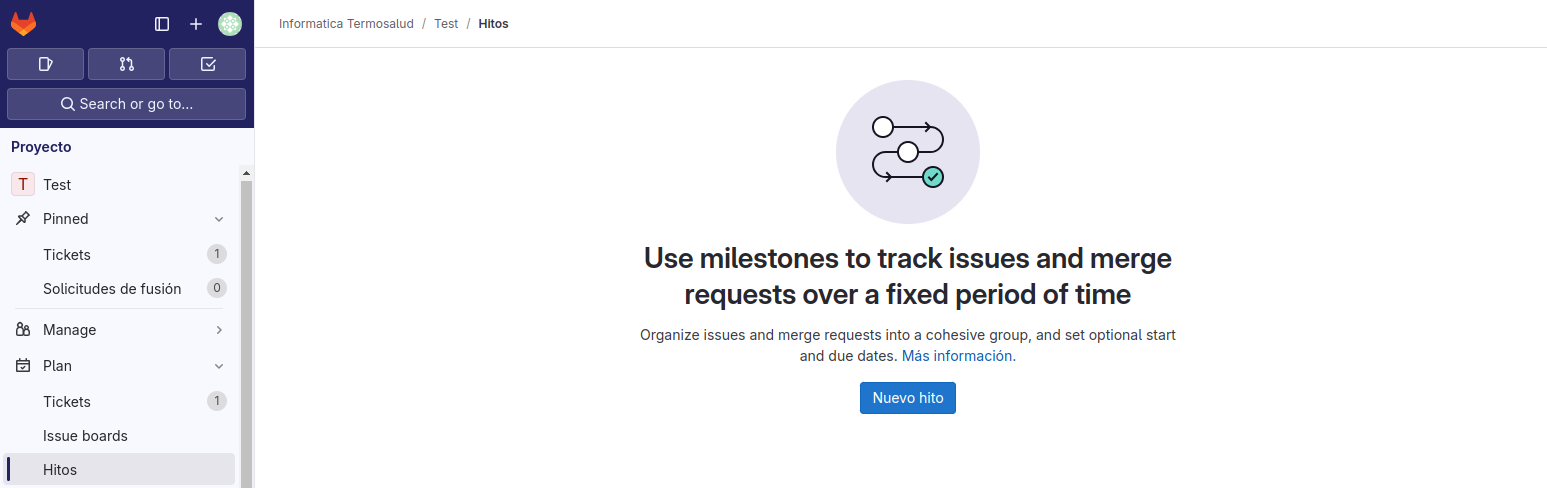


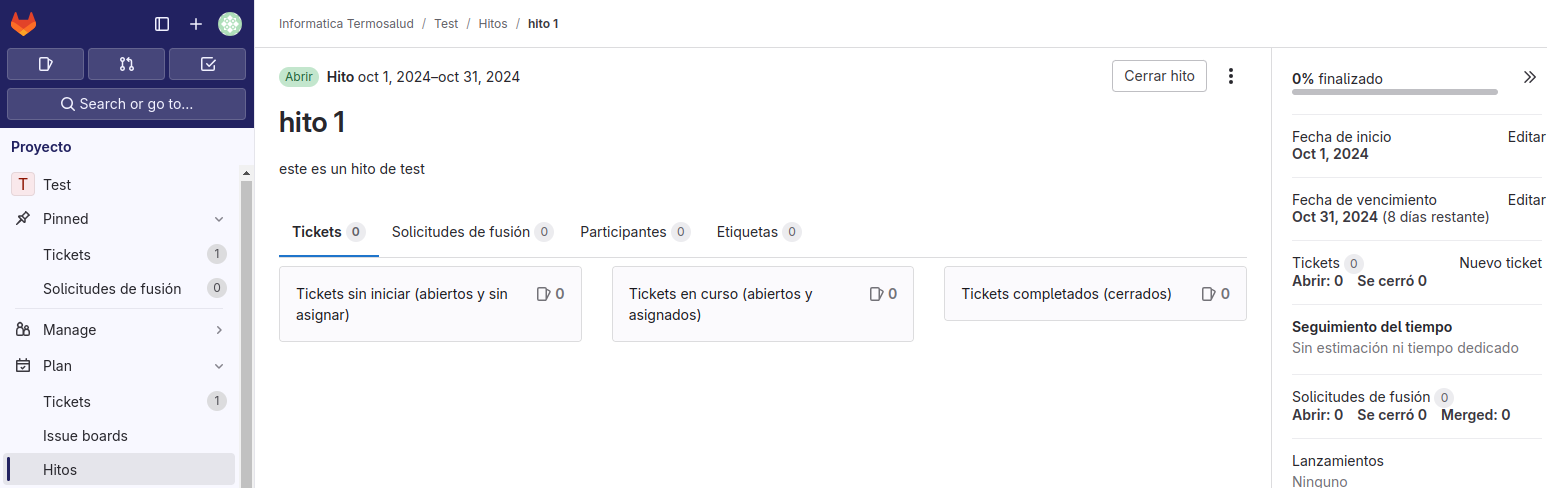
**Ramas específicas para issues**

Para cada issue, se crea una rama específica (feature/, bugfix/, hotfix/) para trabajar en su resolución o implementación, y luego se integra en develop/main mediante un *Merge Request*.

### Gestión de milestones y seguimiento de versiones

Los *Milstones/Hitos* nos permitirán agrupar varios *Issues/Tickets* entre si y asignarles un periodo de tiempo. Lo ideal sería que todos los Tickets y Merge Requests que tengan que ver con un hito estén asociados al mismo.

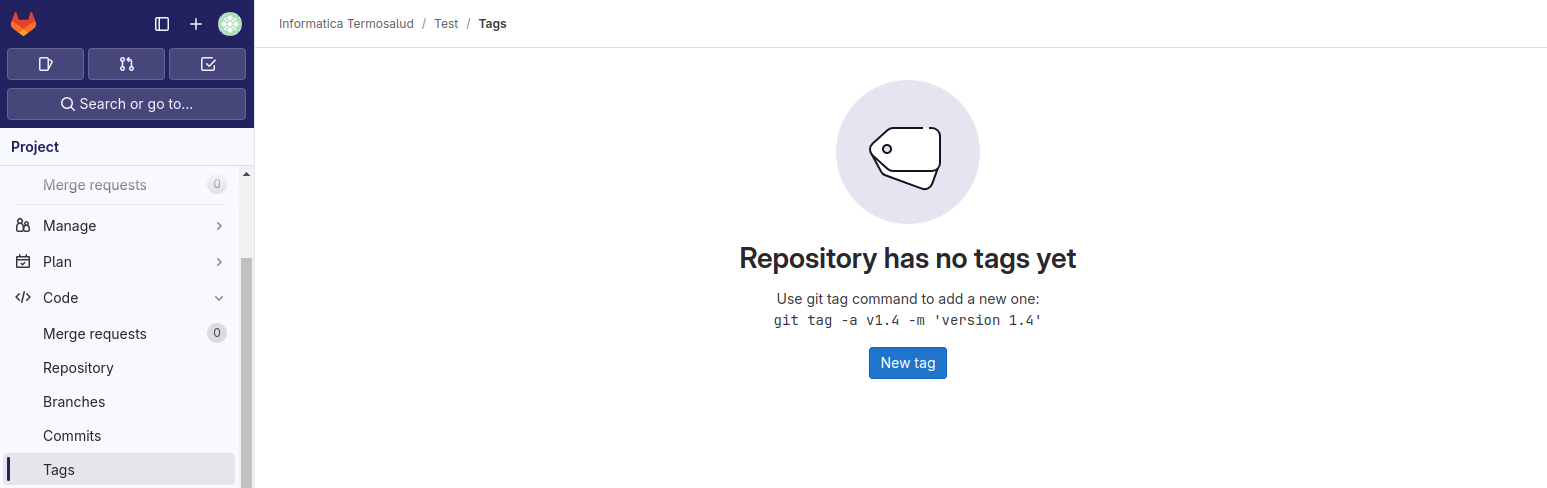
Cuando creamos un Hito, GitLab nos muestra un panel con las acciones realizadas



### Publicación de versiones con tags

**Crear un tag**

Cada vez que queramos publicar una versión debemos crear un tag. GitLab nos permite hacer una descripción sobre cada versión.

Este proceso puede hacerse desde línea de comandos de la siguiente forma

# git tag -a v1.0.0 -m "Descripción de la versión, con detalles del release"  
# git push origin v1.0.0

Debemos crear los tags después de que el *Merge Request* haya sido aprobado y fusionado a la rama principal (main), ya que el tag debe marcar el estado exacto de la versión en el código de producción.

# ****Anexo1: Flujo de trabajo con ramas****

**Rama** main (Producción):

* Código en producción, solo se hacen merges desde develop (nuevas versiones) o hotfix (correcciones urgentes).

**Rama** develop (Desarrollo):

* Rama principal de desarrollo donde se integran todas las nuevas funcionalidades y correcciones de bugs.

## ****¿Cómo trabajar con las ramas?****

1. **Crear una rama** feature/\*

Desde develop debemos crear una nueva rama feature/<nombre-de-la-tarea> para nuevas funcionalidades):

**Actualizar la rama** develop:  
git checkout develop  
git pull origin develop

**Crear una nueva rama** feature:  
git checkout -b feature/<nombre-de-la-tarea>

**Realizar cambios y hacer commits**:  
git add .  
git commit -m "Descripción del cambio"

**Subir la rama al repositorio remoto**:  
git push origin feature/<nombre-de-la-tarea>

**Crear un *Merge Request* en GitLab** y fusionar en develop tras la aprobación.

1. **Crear una rama** bugfix/\*

Cada vez que se detecte un bug debemos, siempre desde develop, crear una rama bugfix/<nombre-del-bug>.

**Actualizar la rama** develop:  
git checkout develop  
git pull origin develop

**Crear una nueva rama** bugfix:  
git checkout -b bugfix/nombre-bugfix

**Realizar cambios y hacer commits**:  
git add .  
git commit -m "Corrección del bug #ID"

**Subir la rama al repositorio remoto**:  
git push origin bugfix/nombre-bugfix

**Crear un Merge Request en GitLab** y fusionar en develop tras la aprobación.

1. **Crear una rama** hotfix/\*

Si necesitamos hacer una corrección urgente en producción usaremos el prefijo hotfix/, la diferencia con bugfix radica en que la modificación se hará sobre main y se realizará, al finalizar el cambio, un *Merge Request* sobre main y sobre develop.

**Actualizar la rama** main:  
git checkout main  
git pull origin main

**Crear una nueva rama** hotfix:  
git checkout -b hotfix/<nombre-hotfix>

**Realizar cambios y hacer commits**:  
git add .  
git commit -m "Corrección urgente en producción"

**Subir la rama al repositorio remoto**:  
git push origin hotfix/nombre-hotfix

**Crear un Merge Request** en GitLab y fusionar en main y develop tras la aprobación.

1. **Fusionar** develop **en** main **para una nueva versión de producción**

**Actualizar la rama** develop:  
git checkout develop  
git pull origin develop

**Crear un Merge Request** desde develop a main en GitLab (para una nueva versión de producción).

**Crear un** tag **para la versión**:  
git tag v1.0.0  
git push origin v1.0.0

Este flujo asegura que cada desarrollo o corrección siga el ciclo adecuado de revisión y aprobación antes de ser fusionado en las ramas principales.

# Anexo 2. Comandos básicos

* **Crear una nueva rama**: git checkout -b nombre-rama
* **Cambiar de rama**: git checkout nombre-rama
* **Actualizar una rama**: git pull origin nombre-rama
* **Hacer commits**: git add . && git commit -m "Mensaje"
* **Subir la rama al remoto**: git push origin nombre-rama
* **Crear un tag**: git tag vX.X.X && git push origin vX.X.X
* **Ver el historial de Git**: git log --oneline --decorate