MNCS - Desarrollo con GIT/Gitlab

Para dar de alta un proyecto en gitlab y que esté incluido en el sistema de despliegue hay que **poner un jira a la aplicación *dj-at-mncs***

Si tienes que **migrar desde GitLab.UM a GitLab.COM** aplica la guía Migración a gitlab.com

Repositorio remoto vs repositorio local

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Por lo general no se necesita información de ningún servidor así la mayoría de las operaciones son prácticamente instantáneas.

Por ejemplo, para navegar por los logs del proyecto, GIT simplemente lee directamente de la base de datos local. Git puede buscar el archivo hace un mes y hacer un cálculo de diferencias localmente, en lugar de tener que pedirle a un servidor remoto que lo haga, u obtener una versión antigua desde la red y hacerlo de manera local.

Los repositorios remotos son versiones del proyecto que se encuentran alojados en el servidor. Puedes haber varios, cada uno de los cuales puede ser de sólo lectura, o de lectura/escritura. Para mantenerse al día hay que gestionar estos repositorios remotos, y mandar (push) y recibir (pull) datos de ellos cuando se necesite actualizar ficheros o mandar actualizaciones.

Los tres estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos:

- **confirmado** (committed): significa que los datos están almacenados de manera segura en tu base de datos local
- **modificado** (modified): significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos
- **preparado** (staged): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación

Estructura de ramas del proyecto

El trabajo se organiza en tres ramas principales:

- **Rama master**: cualquier commit que pongamos en esta rama debe estar preparado para subir a producción
- **Rama preproducción**: rama en la que está el código que acabará en la siguiente versión del proyecto
- **Rama desarrollo**: rama en la que está el código de desarrollo

No se podrán realizar subidas manuales a ninguna de estas 3 ramas, pues se marcarán como protegidas. Todos los cambios se realizarán a través de una *merge request*

Forma de trabajo

En forma resumida, los pasos deberían ser los siguientes:

1. **Crear una rama propia** de la tarea que estemos atendiendo desde desarrollo respetando la nomenclatura definida (usualmente **jira-XXX** aunque se incorporarán más prefijos para artefactos que no sean código pero sí se incluyan en el proyecto).
2. **Trabajar** y completar la tarea.

3. **Comprobar** que no exista nada nuevo en desarrollo a incorporar en la rama de trabajo a través de la operación *rebase*.
4. **Solucionar los conflictos** de código surgidos en el punto anterior y actualizar los cambios en la rama de trabajo.
5. Generar una **Merge Request a la rama de desarrollo**
6. El equipo realiza una revisión del código y envía mensajes al desarrollador en el caso de que surja algún comentario.
7. Si hay algo que corregir, volver al punto 2.
8. Si está todo correcto, la persona con el rol de *Merger* **aceptará los cambios** y los integrará en la rama de desarrollo
9. Se harán merges request y sus correspondientes merges desde desarrollo a preproducción y de preproducción a master con los cambios que queramos ir incorporando.

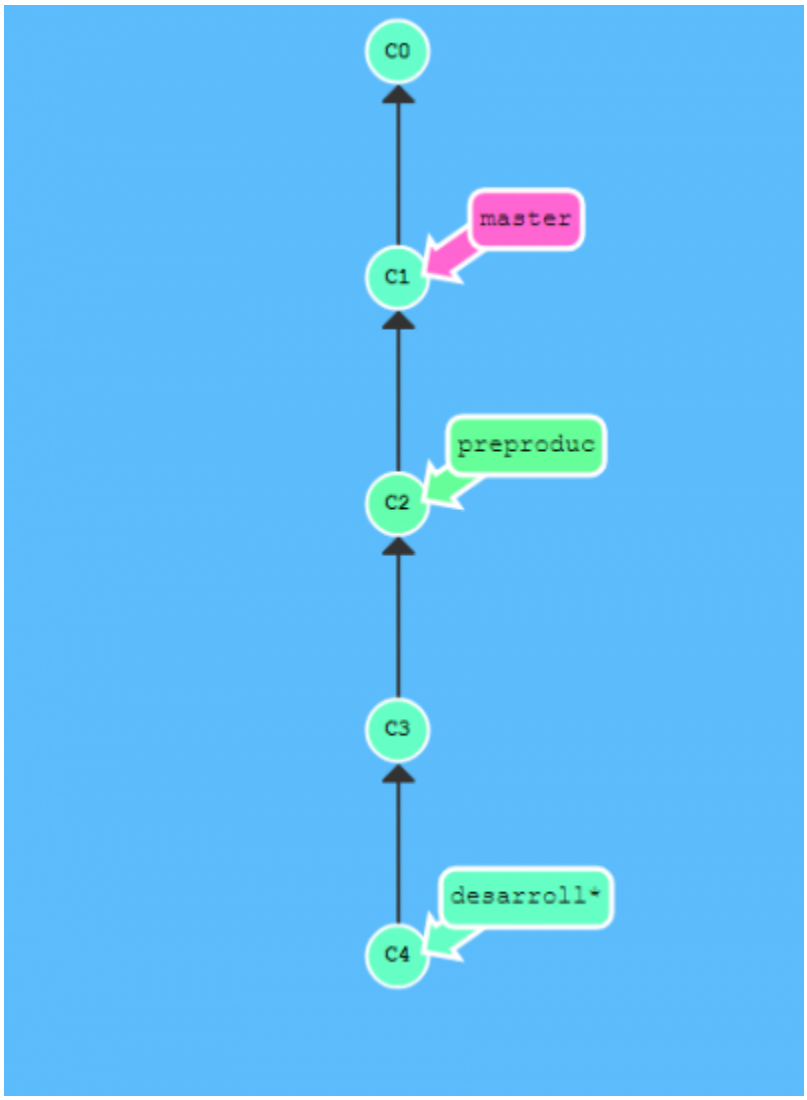
Trabajando con GIT

Vamos a explicar unos comandos básicos que deberán cubrir la mayoría de las tareas de los programadores, para ello utilizaremos la interfaz de comandos, que se puede descargar para windows aquí https://gitforwindows.org

Si queremos simular el comportamiento que tendría una acción sobre GIT, podemos simularla con https://learngitbranching.js.org/?NODEMO

También se podrá utilizar el plugin de eclipse, pero la explicación es análoga.

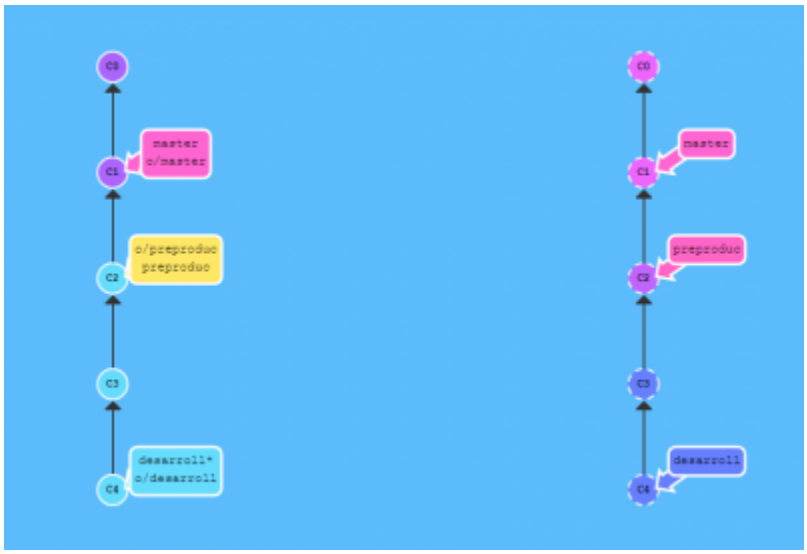
A la hora de ponernos a trabajar en un proyecto que use GIT, lo primero que deberemos hacer será **obtener el código del repositorio remoto** que tendrá un aspecto parecido a éste:



Para ello introducimos localizamos la url de nuestro proyecto en https://gitlab.um.es y utilizamos el comando **`clone`**

```
$ git clone https://gitlab.um.es/mncs/portalfundeweb.git
```

y obtendremos una copia en local del repositorio remoto



Podremos **descargar** localmente **sólo la rama que necesitamos usando el modificador `-b NOMBRERAMA`**

```
$ git clone https://gitlab.um.es/mncs/portalfundeweb.git -b desarrollo
```

O visto cómo quedaría usando el GIT BASH

```
pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2
$ git clone https://gitlab.um.es/mncs/portalfundeweb.git -b desarrollo
Cloning into 'portalfundeweb'...
remote: Enumerating objects: 555, done.
remote: Counting objects: 100% (555/555), done.
remote: Compressing objects: 100% (447/447), done.
remote: Total 555 (delta 96), reused 483 (delta 73)
Receiving objects: 100% (555/555), 2.50 MiB | 0 bytes/s, done.
Resolving deltas: 100% (96/96), done.

pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2
$ cd portalfundeweb/

pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2/portalfundeweb (desarrollo)
$
```

Podemos ver que a la derecha del todo entre paréntesis nos marca la rama en la que estamos trabajando

A continuación tenemos que crear nuestra rama de trabajo, podrá tener el nombre que queramos pero normalmente usaremos la notación **`jira-XXX` sien do XXX el identificador de jira asociado** a la tarea que estamos realizando. (Si queremos que nuestras Merge Request lancen los pipelines de desarrollo, las ramas deberán cumplir el formato anterior).

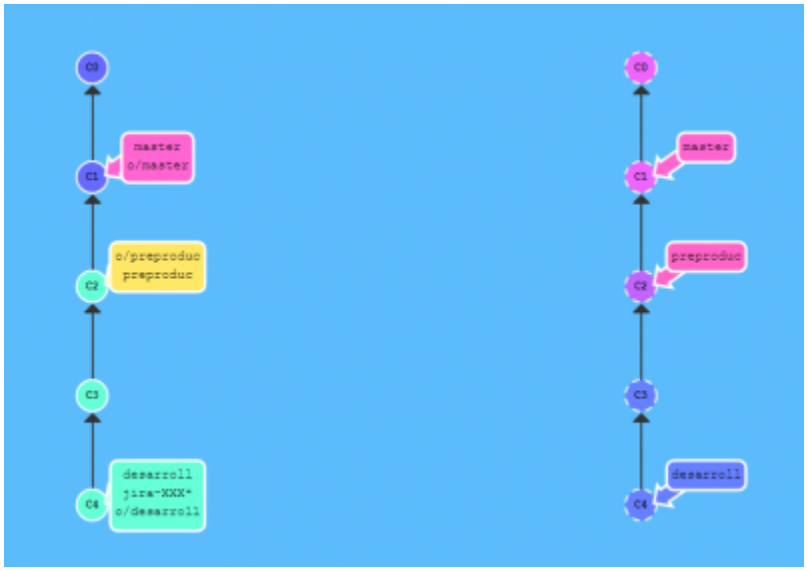
Usamos el comando checkout con el modificador `-b` para ir a la rama

```
$ git checkout -b jira-XXX
```

Podemos ver el efecto en GIT BASH

```
pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2
$ git clone https://gitlab.um.es/mncs/portalfundeweb.git -b desarrollo
Cloning into 'portalfundeweb'...
remote: Enumerating objects: 555, done.
remote: Counting objects: 100% (555/555), done.
remote: Compressing objects: 100% (447/447), done.
remote: Total 555 (delta 96), reused 483 (delta 73)
Receiving objects: 100% (555/555), 2.50 MiB | 0 bytes/s, done.
Resolving deltas: 100% (96/96), done.

pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2
$ cd portalfundeweb/

pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2/portalfundeweb (desarrollo)
$ git checkout -b jira-IDI-131
Switched to a new branch 'jira-IDI-131'

pablo.gonzalez@10C37B94616C MINGW64 /c/git-test2/portalfundeweb (jira-IDI-131)
$
```

Y nuestras ramas segurían así

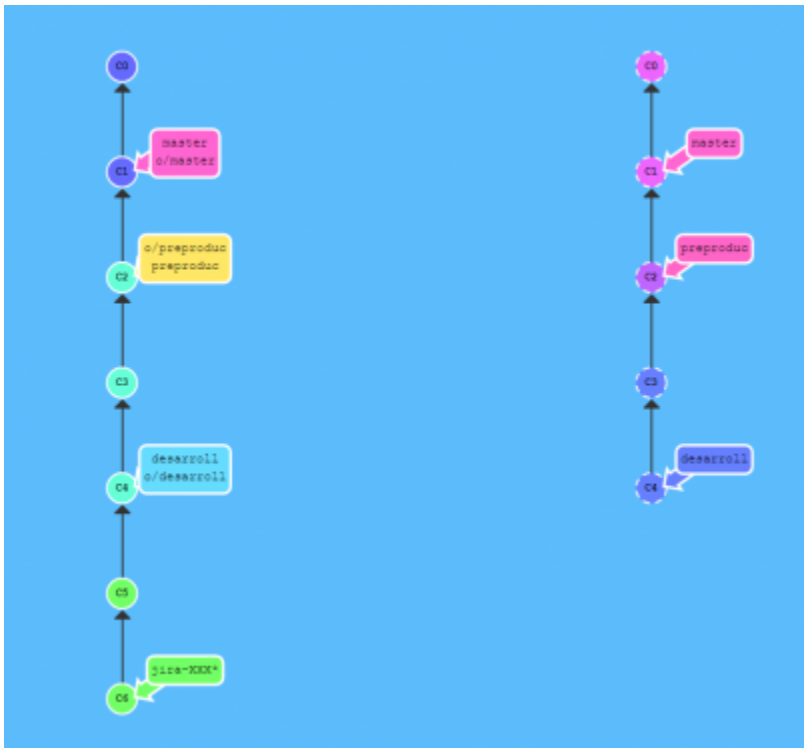


posteriormente podemos hacer nuestro trabajo y mandar varios commits al repositorio local, que se almacenarán en esta rama

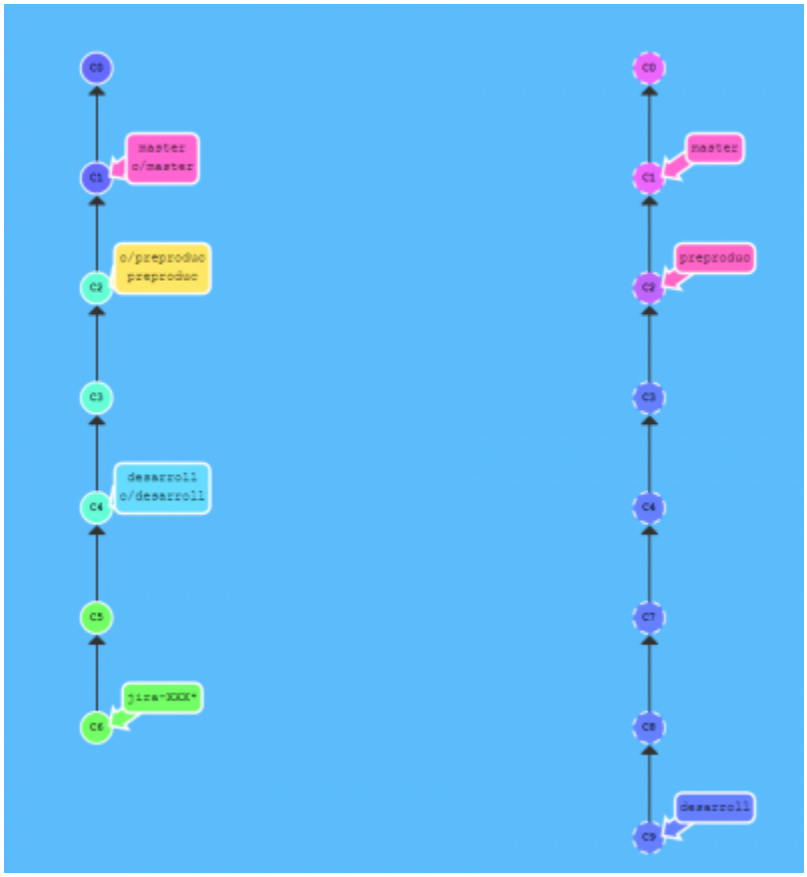
```
$ git status
$ git add .
$ git commit -m "Mi commit de ejemplo"
```

> Con git status comprobaremos el estado de los ficheros de nuestra rama. Si hemos añadido **archivos que no queremos que se incluyan** en el próximo commit podemos arreglando usando el comando **git reset <archivo>** o directamente git reset para todos los archivos pendientes de commit

El esquema de repositorios ahora quedaría así



Eso no quita para que nuestro proyecto en remoto siga teniendo actualizaciones, así que avanzaríamos en paralelo sin molestarnos



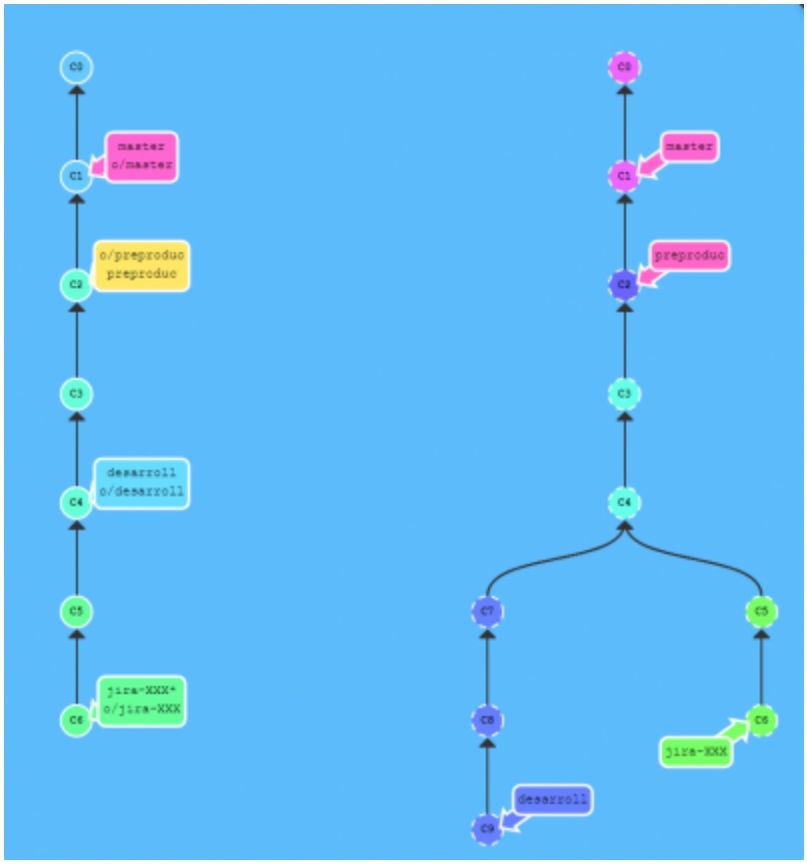
⚠️ Como podemos ver aquí **nada de nuestro trabajo está en los repositorios remotos aún**, por lo que si algo le sucede a nuestro equipo o por un descuido borramos los datos, podremos perder todo nuestro trabajo

Para evitar estos problemas, **podemos subir nuestra rama al servidor**, usamos el comando push y el repositorio remoto será origin que es el que se suele haber por defecto

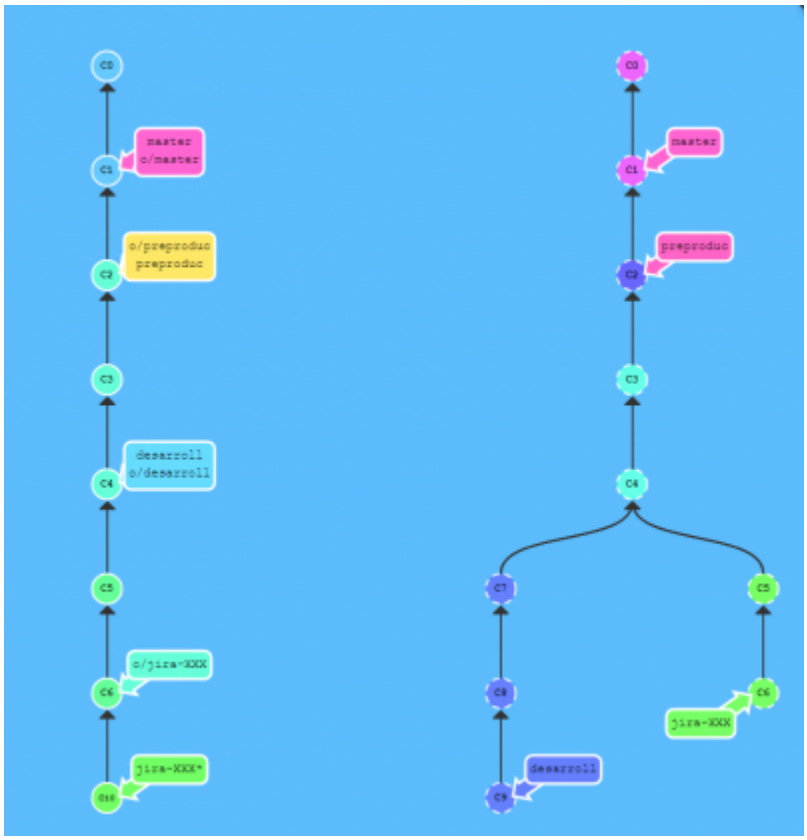
```
$ git push origin jira-XXX
```

> Es importante que la rama sobre la que vayamos a hacer push sea la que estamos trabajando

Los repositorios quedarían así, con una bifurcación en el código en el punto que creamos la rama

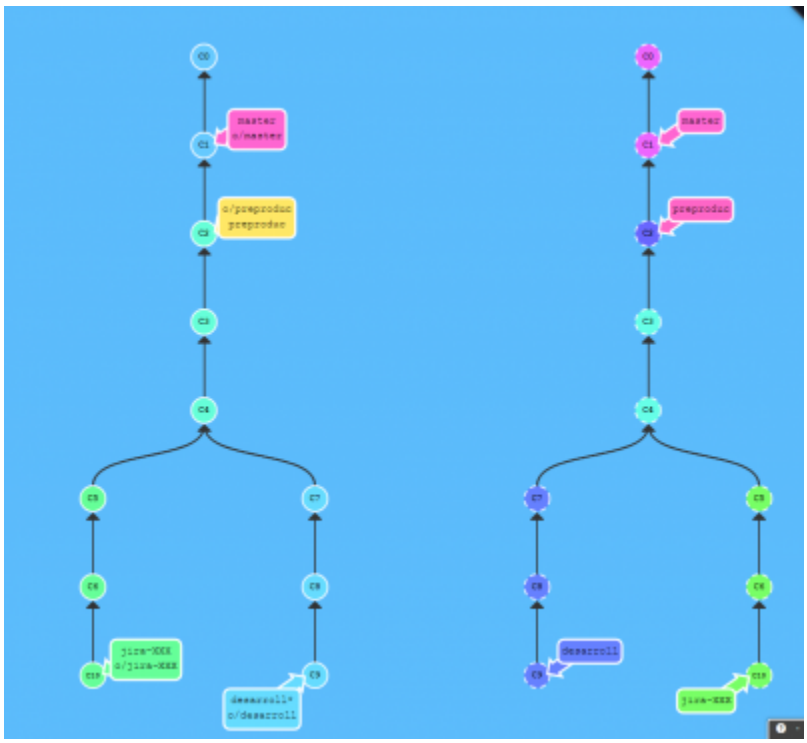


Podemos hacer más commits y vemos como en nuestro repositorio local (izquierda) la rama en el repositorio remoto `o/jira-XXX` y la del repositorio local ya no estarán sincronizadas



Si consideramos que **hemos terminado nuestro trabajo**, llega el momento de integrar con la rama de desarrollo. Primero **tendremos que traernos lo último que haya en desarrollo** para ver si hay conflictos

```
$ git checkout desarrollo
$ git pull --rebase
```



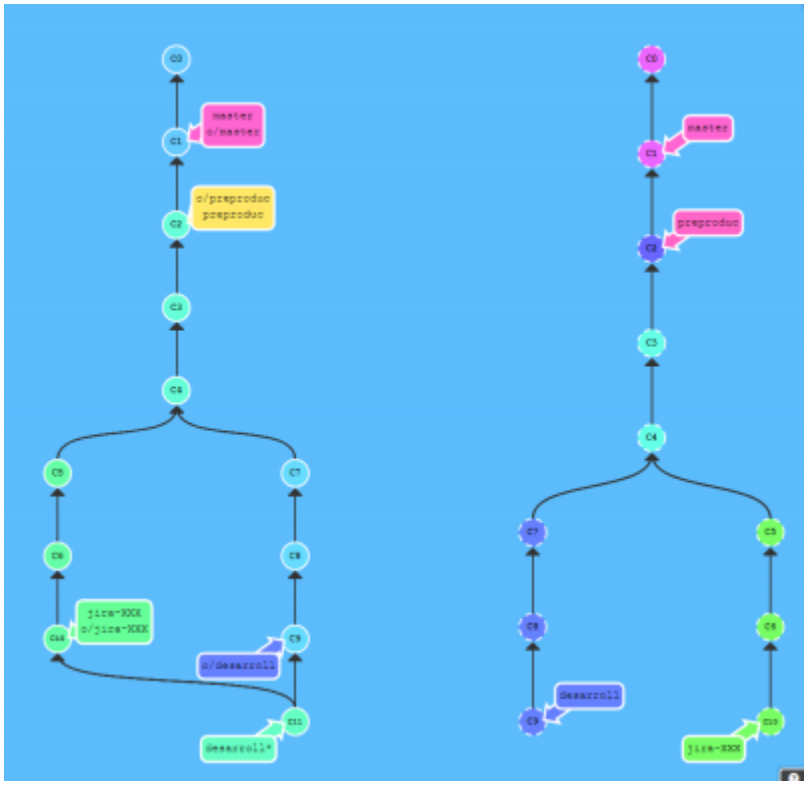
Nos situamos en la rama desarrollo con el comando checkout y a continuación nos traemos los cambios con `pull –rebase`, y volvemos a nuestra rama ya sólo nos queda hacer merge localmente de nuestra rama activa con desarrollo

```
$ git merge desarrollo
```

⚠️ Es importante decir que **este merge se realizará en local**, para producir cambios en las ramas principales (master, preproduccion y desarrollo) se hará uso de las *merge request* de gitlab

Nos queda **resolver los conflictos (si los hay) y hacer push** de los últimos cambios al repositorio remoto (en la rama jira-XXX GIT no dejará hacer push directos a master, preproduccion o desarrollo)

Y el repositorio quedaría así



Por último recomendamos hacer también merge desde desarrollo porque al empezar la siguiente tarea sacaremos la rama nueva *jira-XXX* desde esta rama y así estará actualizada antes de aceptar la merge request en el servidor

```
$ git merge jira-XXX
```

Configurando el proyecto GIT

Cuando tengamos creado el repositorio GIT deberemos ir al directorio especial *.git* y abrir el fichero *config* el contenido típico deberá ser éste

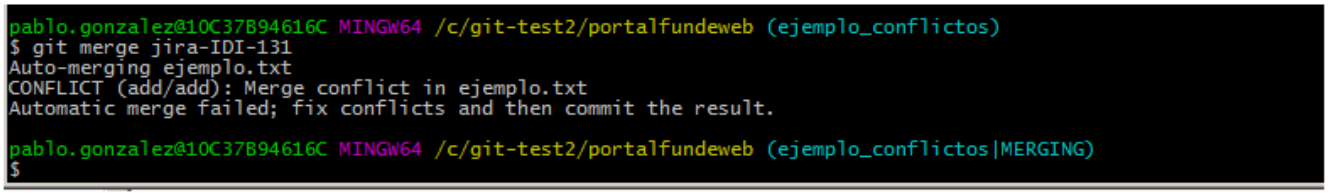
```
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "origin"]
        url = https://gitlab.um.es/mncs/portalfundeweb.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "desarrollo"]
        remote = origin
        merge = refs/heads/desarrollo
```

⚠️ **Añadiremos la propiedad *autocrlf* a *false*** *(autocrlf = false)* para **solventar problemas entre las distintas configuraciones** de los clientes de GIT

```
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
        autocrlf = false
[remote "origin"]
        url = https://gitlab.um.es/mncs/portalfundeweb.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "desarrollo"]
        remote = origin
        merge = refs/heads/desarrollo
```

Resolución de conflictos

En ocasiones al hacer el merge de una rama con otra aparecen conflictos (al estilo de SVN o de cualquier otro sistema de control de versiones) y nos vemos obligados a resolverlos para poder integrarlo



Si ejecutamos el comando

```
$ git status
```

Obtendremos más información

Aquí hemos intentado hacer merge de la rama `jira-IDI-131` con la rama `ejemplo_conflictos`. Ambos tienen el `fichero ejemplo.txt` con contenidos incompatibles y hay que resolverlo manualmente.

El contenido del fichero `ejemplo.txt` en la rama `ejemplo_conflictos` es

```
para dar un conflicto gordo
```

y en la rama `jira-IDI-131`

```
este es un ejemplo para dar conflicto
```

Si abrimos el fichero con el editor veremos los problemas

```
<<<<<<< HEAD
para dar un conflicto gordo
=======
este es un ejemplo para dar conflicto
>>>>>>> jira-IDI-131
```

Para localizar el inicio del conflicto hay que localizar el marcador «««<. El marcador ======= separa los cambios entre las 2 ramas, seguido de »»»> BR ANCH-NAME. En este ejemplo alguien escribió “para dar un conflicto gordo” en la rama base y otra persona escribió “este es un ejemplo para dar conflicto” en la rama `jira-IDI-131`

Decide qué es lo que quieres incorporar, o genera una nueva versión del fichero que incorpore ambos cambios. Elimina los marcadores de conflicto ««««<, =======, »»»> y haz los cambios finales.

```
$ git add .
$ git commit -m "Resuelto el conflicto de merges"
```

Conflictos con uno mismo

En ocasiones GIT detecta 2 commits hacia el mismo fichero y a la hora de hacer un push de la rama nos puede dar problemas

```
$ git push origin jira-XXX
```

Nos puede dar como resultado

```
To https://gitlab.um.es/mncs/portalfundeweb.git
 ! [rejected]        jira-XXX -> jira-XXX (non-fast-forward)
error: failed to push some refs to 'https://gitlab.um.es/mncs/portalfundeweb.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Para arreglarlo podemos hacer lo siguiente

```
$ git pull --rebase origin jira-XXX
```

Y si tenemos suerte será capaz de solventar el conflicto

```
From https://gitlab.um.es/mncs/portalfundeweb
 * branch            jira-XXX -> FETCH_HEAD
First, rewinding head to replay your work on top of it...
Applying: arreglamos el problema con el settings
Using index info to reconstruct a base tree...
M       .m2/settings.xml
.git/rebase-apply/patch:42: trailing whitespace.
                                <pluginRepository> <id>plugins.archiva.atica.umu.es</id>
.git/rebase-apply/patch:43: trailing whitespace.
                                    <name>ATICA  - UMU Plugins Repository</name>
.git/rebase-apply/patch:44: trailing whitespace.
                                    <url>https://archiva.um.es/archiva/repository/FundeWeb/</url>
.git/rebase-apply/patch:45: trailing whitespace.
                                        <releases>
.git/rebase-apply/patch:46: trailing whitespace.
                                            <enabled>true</enabled>
warning: squelched 5 whitespace errors
warning: 10 lines add whitespace errors.
Falling back to patching base and 3-way merge...
```

Pero puede no ser así y que nos responda algo como

```
From https://gitlab.um.es/mncs/portalfundeweb
 * branch            jira-XXX -> FETCH_HEAD
First, rewinding head to replay your work on top of it...
Applying: Añadimos las carpetas para la gestión de maven
Using index info to reconstruct a base tree...
.git/rebase-apply/patch:33: trailing whitespace.

.git/rebase-apply/patch:34: trailing whitespace.

.git/rebase-apply/patch:50: trailing whitespace.

.git/rebase-apply/patch:58: trailing whitespace.

warning: 4 lines add whitespace errors.
Falling back to patching base and 3-way merge...
Auto-merging .m2/settings.xml
CONFLICT (add/add): Merge conflict in .m2/settings.xml
error: Failed to merge in the changes.
Patch failed at 0001 Añadimos las carpetas para la gestión de maven
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".
```

Así nuestra rama de trabajo pasará a llamarse `jira-XXX|REBASE 1/2` (según el número de conflictos que detecte)

```
 ~/portalfundeweb (jira-XXX|REBASE 1/2)
$ git rebase --continue
.m2/settings.xml: needs merge
You must edit all merge conflicts and then
mark them as resolved using git add
```

editamos el fichero y lo añadimos

```
~/portalfundeweb (jira-XXX|REBASE 1/2)$ git add .m2/settings.xml
~/portalfundeweb (jira-XXX|REBASE 1/2)$ git rebase --continue
Applying: Añadimos las carpetas para la gestión de maven
Applying: arreglamos el problema con el settings
Using index info to reconstruct a base tree...
M       .m2/settings.xml
.git/rebase-apply/patch:30: trailing whitespace.
                              <pluginRepository> <id>plugins.archiva.atica.umu.es</id>
.git/rebase-apply/patch:31: trailing whitespace.
                                 <name>ATICA  - UMU Plugins Repository</name>
.git/rebase-apply/patch:32: trailing whitespace.
                                 <url>https://archiva.um.es/archiva/repository/FundeWeb/</url>
.git/rebase-apply/patch:33: trailing whitespace.
                                    <releases>
.git/rebase-apply/patch:34: trailing whitespace.
                                       <enabled>true</enabled>
warning: squelched 5 whitespace errors
warning: 10 lines add whitespace errors.
Falling back to patching base and 3-way merge...
Auto-merging .m2/settings.xml
CONFLICT (content): Merge conflict in .m2/settings.xml
error: Failed to merge in the changes.
Patch failed at 0002 arreglamos el problema con el settings
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

~/portalfundeweb (jira-XXX|REBASE 2/2) $
```

Donde nos indica que queda otro conflicto por resolver y nos ha cambiado al `REBASE 2/2` Podemos repetir la misma operación

```
~/portalfundeweb (jira-XXX|REBASE 2/2) $ git rebase --continue
.m2/settings.xml: needs merge
You must edit all merge conflicts and then
mark them as resolved using git add

~/portalfundeweb (jira-XXX|REBASE 2/2) $ git add .
~/git-pipeline/portalfundeweb (jira-XXX|REBASE 2/2) $ git rebase --continue
Applying: arreglamos el problema con el settings

~/git-pipeline/portalfundeweb (jira-XXX) $ git push origin jira-IDI-132
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 1.02 KiB | 0 bytes/s, done.
Total 8 (delta 4), reused 0 (delta 0)
remote:
remote: To create a merge request for jira-IDI-132, visit:
remote:   https://gitlab.um.es/mncs/portalfundeweb/merge_requests/new?merge_request%5Bsource_branch%5D=jira-XXX
remote:
To https://gitlab.um.es/mncs/portalfundeweb.git
   aee3109..f09c0f7  jira-XXX -> jira-IDI-XXX
```

Uso del fichero .gitignore

Hasta ahora habíamos indicado que se podían preparar los ficheros para el commit con un comando de la forma

```
$ git add .
```

Lo cual deja preparado para hacer commit a todos los archivos alcanzables desde el directorio actual y subdirectorios.

Si sólo queremos mandar un fichero al commit deberíamos usar

```
$ git add FICHERO
```

Sin embargo ésta es una labor algo tediosa y propensa a equivocaciones y olvidos cuando tenemos muchos ficheros que queremos preparar y otros que no (el resultado de compilaciones, ficheros de log, etc…).

Para facilitarnos esto crearemos en la raíz del proyecto el fichero `.gitignore`

```
$ vim .gitignore
```

Y le añadiremos contenido

```
.m2/repository/
web/target/
ear/target/
*.log
*_.java
```

Donde las 3 primeras líneas ordenan a git que ignore los cambios en estos directorios y sucesivos, después tenemos `*.log` que hace que se ignoren todos los ficheros *.log encuentren donde se encuentren en el proyecto, también podremos añadir reglas más complejas usando el modificador `!`, por ejemplo podríamos tener

```
*.log
!application.log
```

Que haría que git ignorase todos los archivos `.log` salvo el que se llame `application.log`

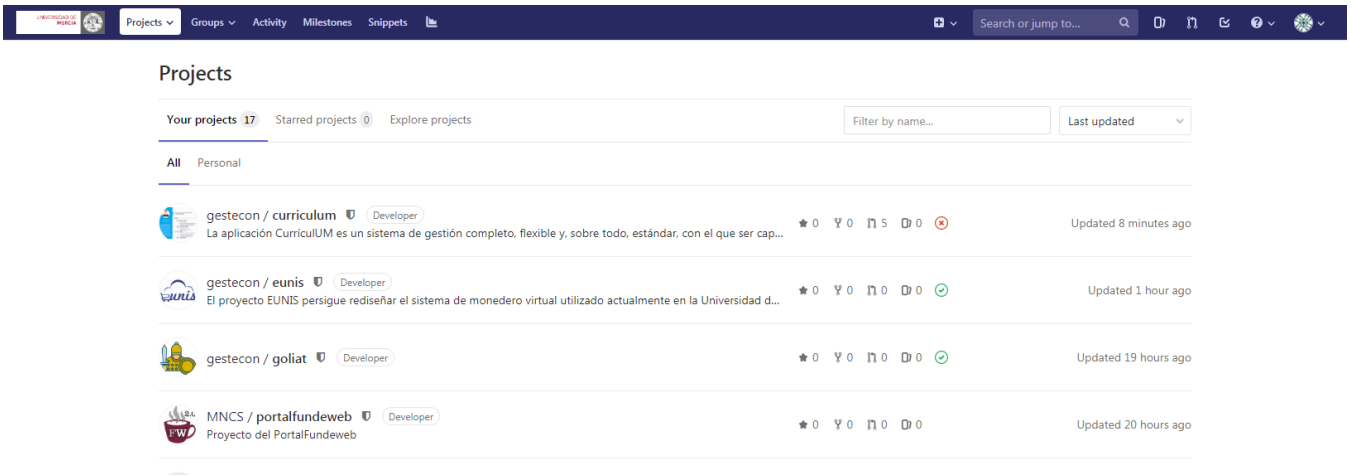
Usando GITLab

Hasta ahora hemos visto cómo trabajar con GIT de manera independiente, ahora tenemos que llevar nuestro código al proyecto principal integrándolo con la rama de desarrollo.

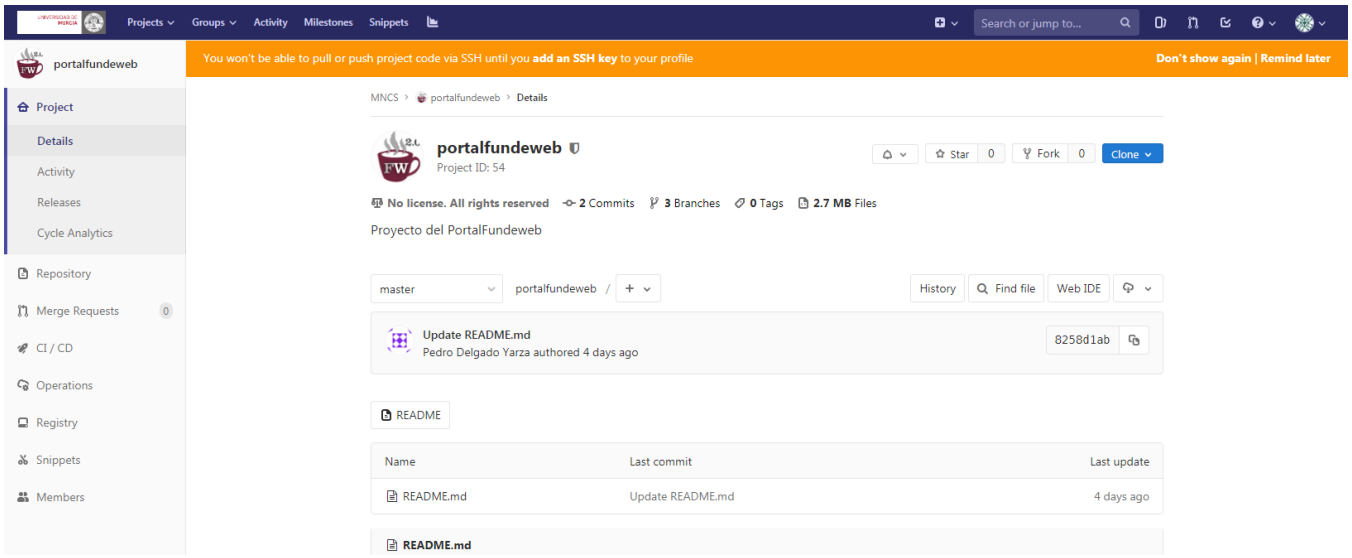
Partimos de la base que no existen conflictos o éstos se han resuelto antes de generar una *merge request*

Hacemos login en la aplicación

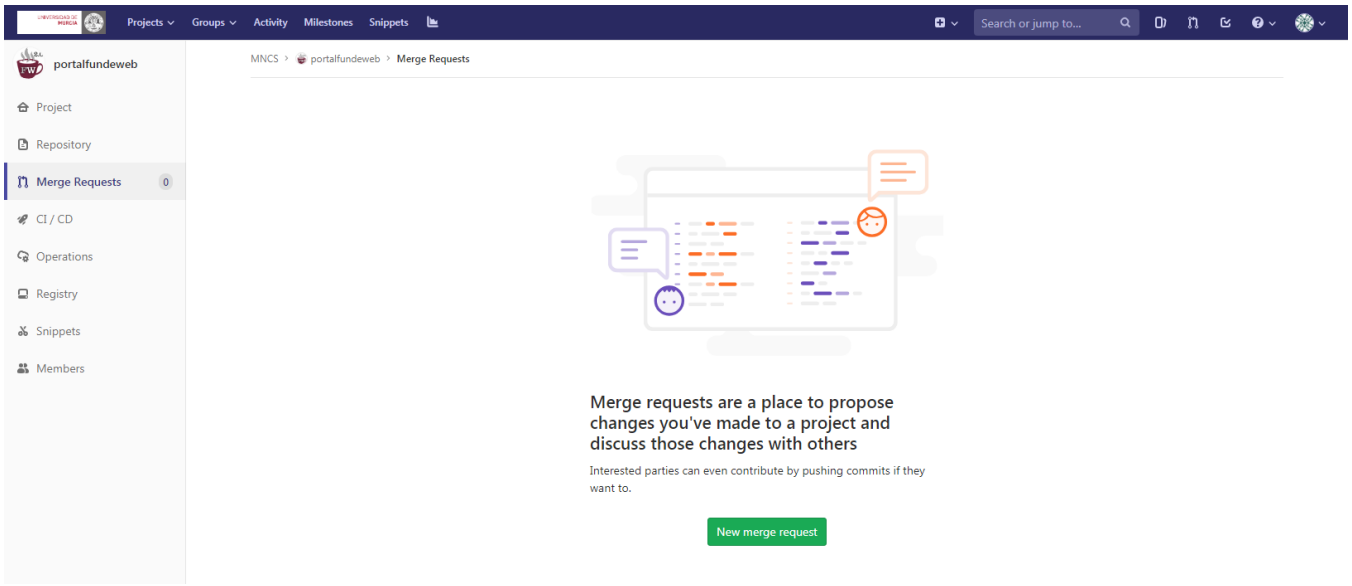
Realizando una solictud de merge



Buscamos nuestro proyecto



Buscamos la opción para generar una *merge reques*

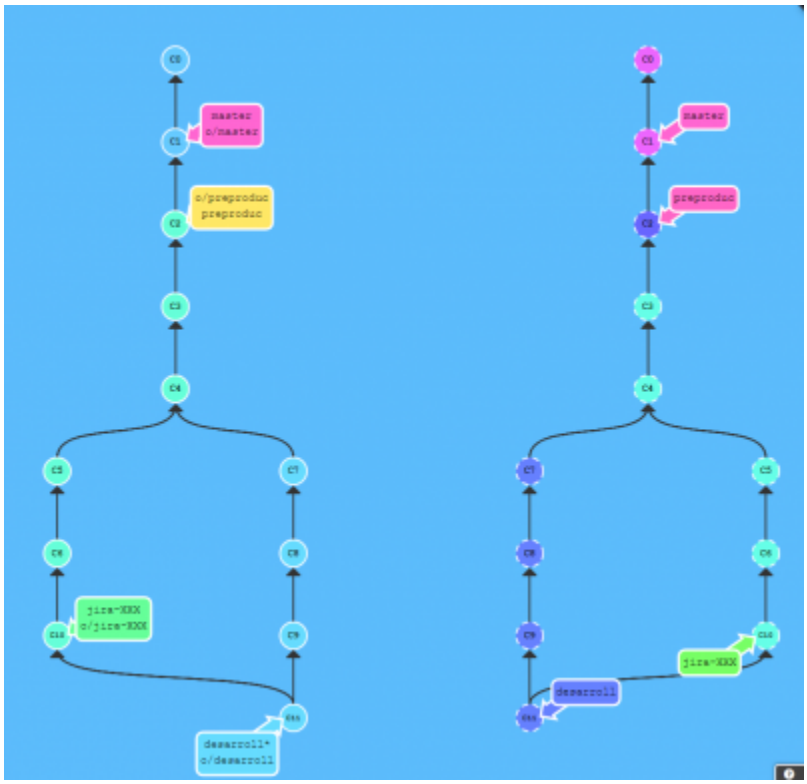


Nos preguntará qué 2 ramas queremos unir, en nuestro caso elegiremos la rama que acabamos de terminar y desarrollo

Por último rellenamos los datos de la *merge request*, podemos incluir más información para que a la hora de aceptar el merge y revisar el código se pueda usar de guía

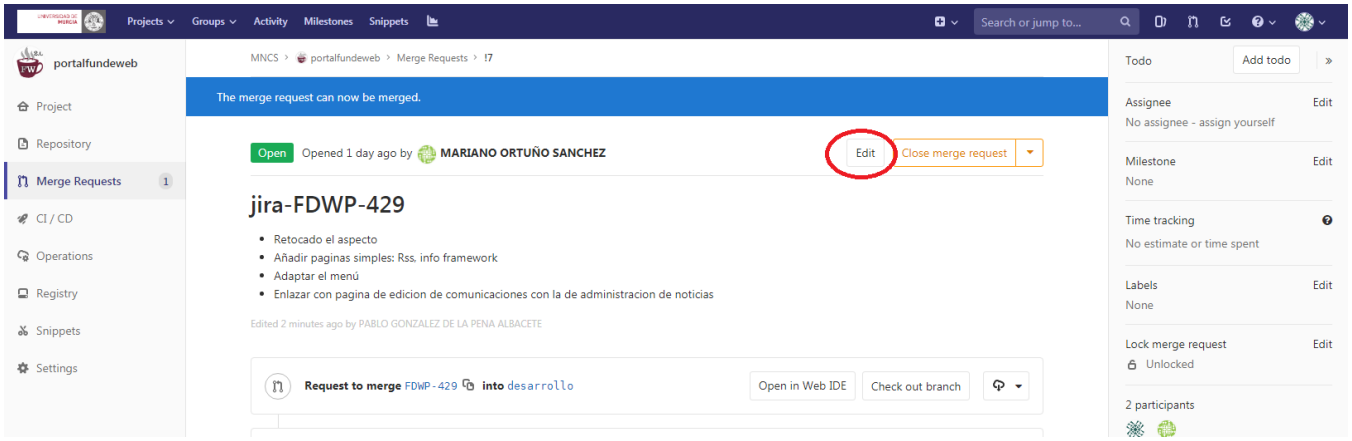
Es buena idea marcar la casilla “Delete source branch when request is accepted” esto borrará la rama del repositorio cuando el merge se haga y evitará que se acumulen ramas que no se van a usar más. Siempre la conservamos en local y podremos volver a subirla si fuera necesario.

Finalmente el repositorio quedará así

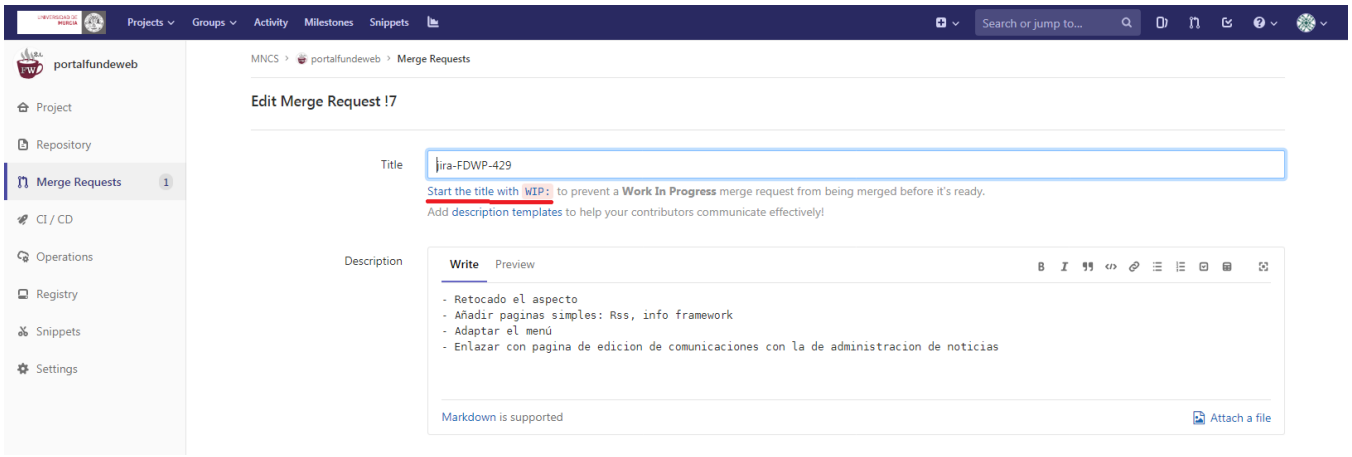


Rechazar una merge

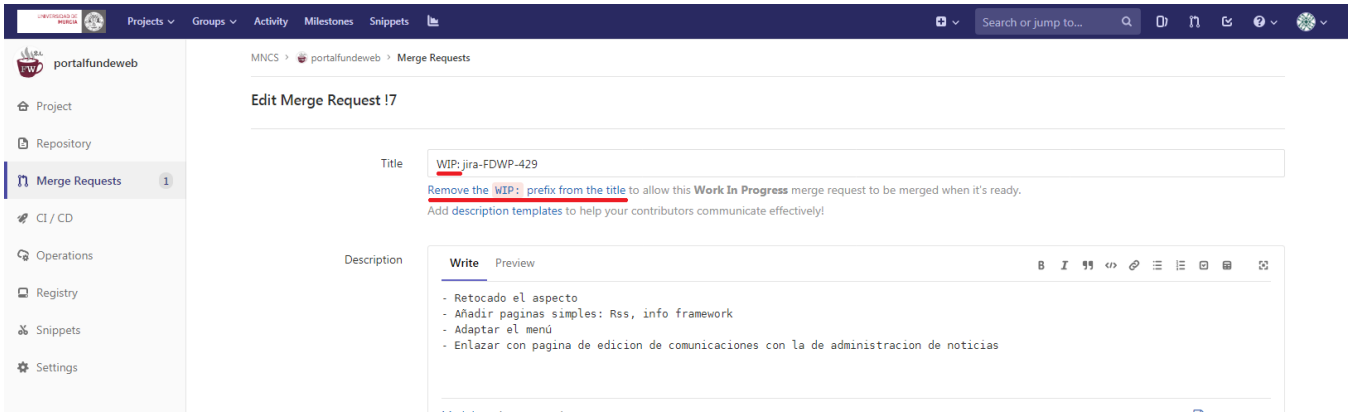
El primer paso es editar la merge



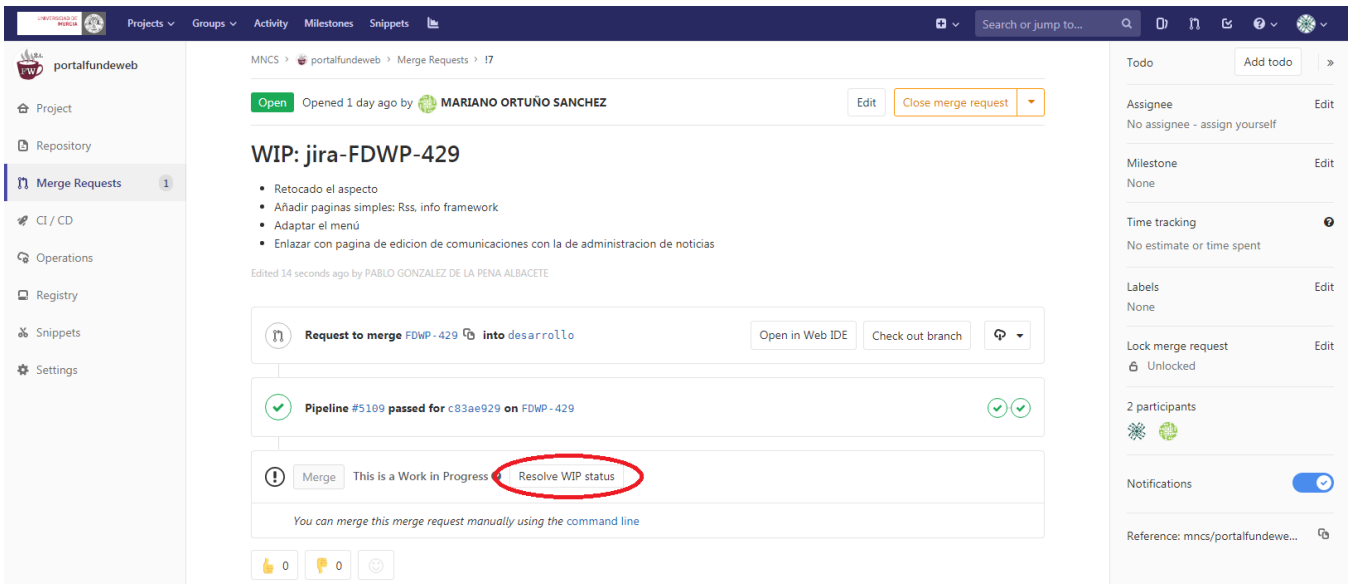
Añadimos el estado WIP y el motivo. Esto evitará que pueda hacerse merge de esta rama



Cuando el trabajo esté listo quitamos el prefijo WIP para que se pueda realizar el merge



Mientras no se quite el estado WIP la rama quedará así



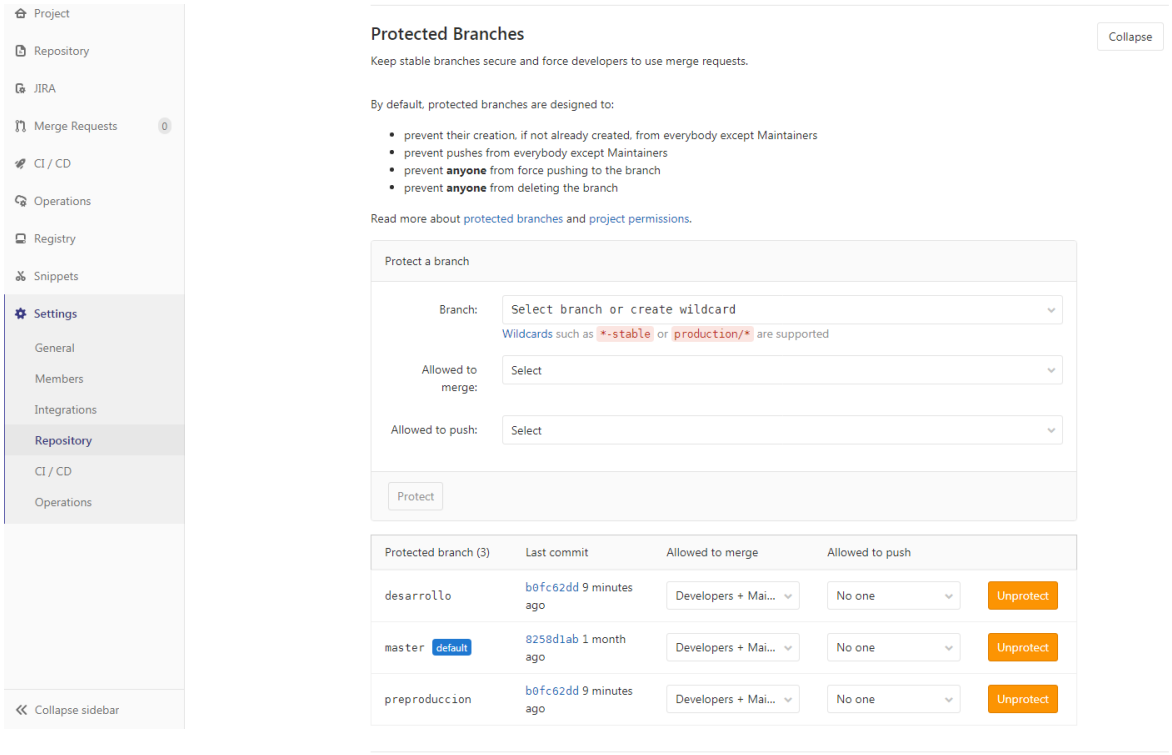
Configurando ramas

El usuario administrador tendrá que ir al apartado *Settings* y desplegar la opción de *Protected branches*

Aquí crearemos las ramas que necesitamos desarrollo y preproduccion.

En los desplegables seleccionaremos en Allowed to merge quien queremos que haga merge *Mantainers* o *Developers + Mantainers* y después en allowed to push dejaremos *None*

> Mantainer es un usuario con más responsabilidad que developer, se puede usar para que no todos los usuarios del grupo puedan hacer merge será típicamente un revisor de código.



Configurando el pipeline

Primero creamos el directorio *.m2* con el fichero *settings.xml* con la configuración maven de nuestro proyecto, típicamente:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.
xsd">

        <localRepository>.m2/repository</localRepository>
        <offline>false</offline>
        <profiles>
                <profile>
                        <id>FundeWeb-profile</id>
                        <repositories>
                                <repository>
                                        <id>archiva.atica.umu.es</id>
                                        <name>ATICA - UMU Repository</name>
                                        <url>https://archiva.um.es/archiva/repository/FundeWeb/</url>
                                        <releases>
                                                <enabled>true</enabled>
                                        </releases>
                                        <snapshots>
                                                <enabled>true</enabled>
                                        </snapshots>
                                </repository>
                        </repositories>
                        <pluginRepositories>
                                <pluginRepository> <id>plugins.archiva.atica.umu.es</id>
                                    <name>ATICA - UMU Plugins Repository</name>
                                    <url>https://archiva.um.es/archiva/repository/FundeWeb/</url>
                                        <releases>
                                            <enabled>true</enabled>
                                        </releases>
                                        <snapshots>
                                            <enabled>true</enabled>
                                        </snapshots>
                                </pluginRepository>

                        </pluginRepositories>

                </profile>
        </profiles>

        <activeProfiles>
                <activeProfile>FundeWeb-profile</activeProfile>
        </activeProfiles>

        <pluginGroups>
                <pluginGroup>com.oracle.weblogic</pluginGroup>
        </pluginGroups>

</settings>
```

Generar el fichero *.gitlab-ci.yml* con el contenido

```
variables:
  PROJECT_NAME: NombreProyecto
  SVN_BASE: https://svn.um.es/svn/NOMBREPROYECTO/
  APP_CONTEXT: nombreproyecto


include:
  - project: 'mncs/mncs-pipelines'
    file: '.gitlab-ci-template.yml'
```

o si estamos en gitlab.com

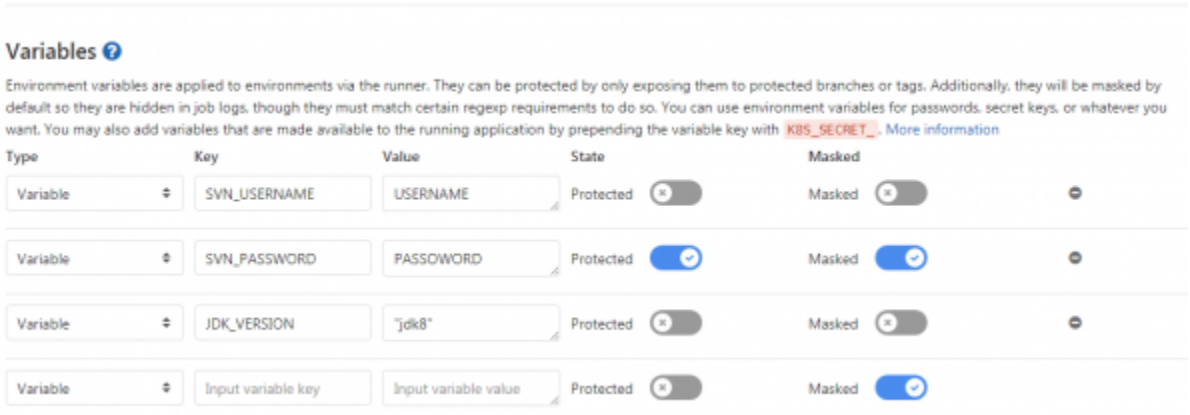
```
variables:
  PROJECT_NAME: NombreProyecto
  SVN_BASE: https://svn.um.es/svn/NOMBREPROYECTO/
  APP_CONTEXT: nombreproyecto


include:
  - project: 'umugit/atica/desarrollo/common/mncs-pipelines'
    file: '.gitlab-ci-template.yml'
```

Variables para el despliegue

Tendremos un usuario SVN por grupo de trabajo, además si usamos aplicaciones Fundeweb 2.1 tendremos que usar un perfil de compilación JDK8 todas estas cosas las especificaremos con variables dentro de GitLAB Dentro de la sección de grupo de GitLAB y por parte de un *Owner* del grupo

Le daremos al botón “Expand” y creamos las variables

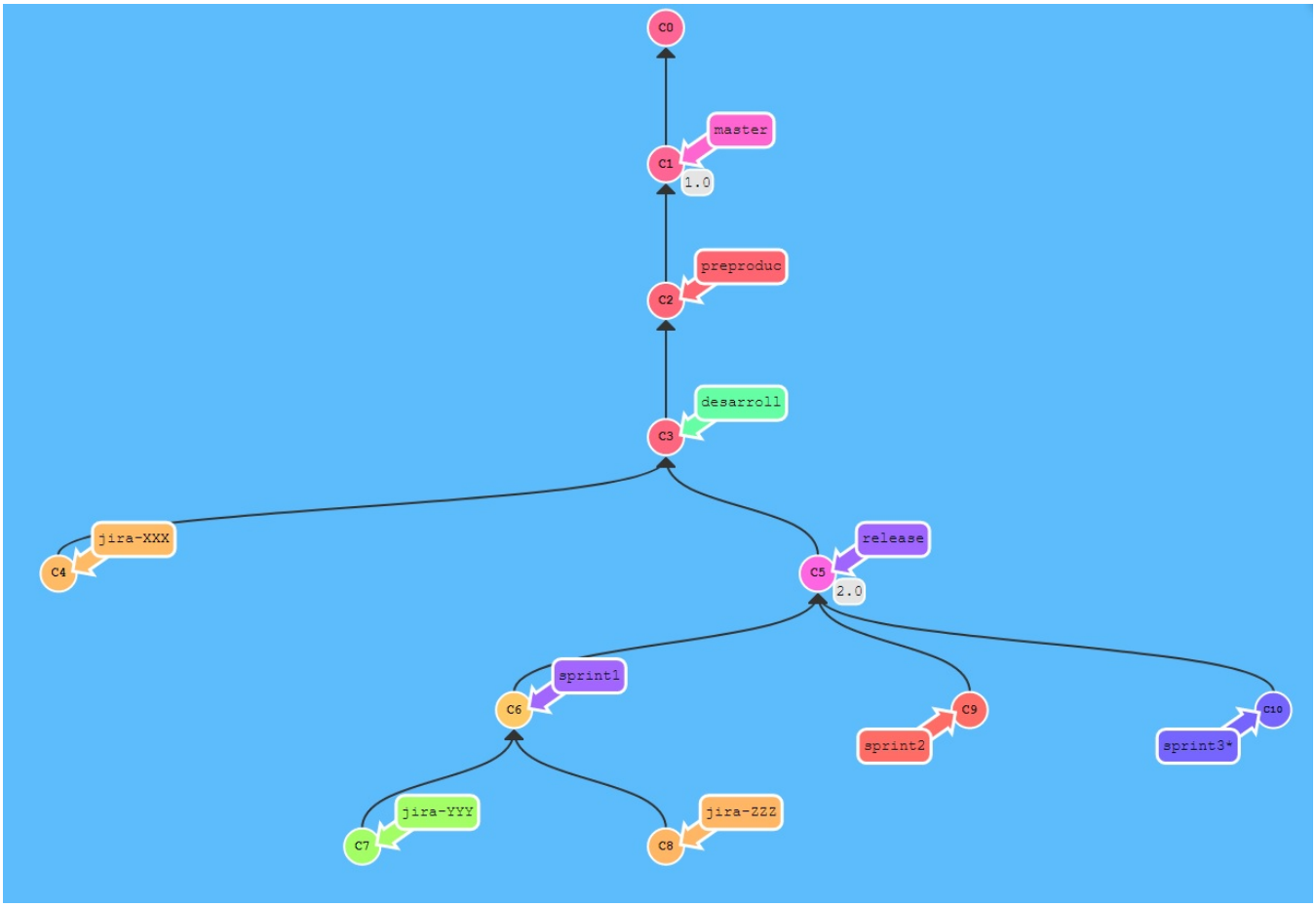
- SVN_USERNAME
- SVN_PASSWORD
- JDK_VERSION (opcional)
- SUFIJO_DESPLIEGUE (opcional, con valor fm si se despliega en servidores fm)
- PRESERVE_LIB (opcional, con valor true si queremos que se copien las librerías del package a los servidores, false por defecto). Para aplicaciones de servicios tiene que especificarse con valor *true*.
- ORIGEN_BIRTUM (opcional, con valor por defecto informes_birtum/informes/${APP_CONTEXT}), es la ruta relativa de la raíz de git de donde tiene que sacar los informes BIRT
- DESTINO_BIRTUM (opcional debe especificarse junto con el anterior, con valor por defecto informes_birtum), es la ruta relativa al SVN donde se copian los ficheros
- LISTA_SECRETS (opcional) para no almacenar secretos en claro en el código, ni el repositorio, es una lista de nombres de secrets separado por espacios. Los secrets se inyectarán en el fichero de filtros según el entorno. Tendrán que especificarse variables nuevas en gitlab *nombresecret_DESA*, *nombresecret_TEST*, *nombresecret_PROD* siendo nombresecret el literal dado de alta en la variable *LISTA_SECRETS*
- MODULO_FILTRO (opcional, valor por defecto web) el módulo al que se tienen que aplicar los secrets en los filtros
- MAVEN_CUSTOM_PROFILE (opcional) si nuestra aplicación tiene perfiles personalizados de MAVEN que hay que activar
- OLD_FUNDEWEB (opcional) si nuestra aplicación es Fundeweb 1, se lo podemos indicar para ayudar a sonarqube a detectar las rutas. El valor debe ser *true*



Las variables que se marquen como protegidas sólo serán accesibles desde ramas protegidas, por lo tanto el perfil de compilación no deberíamos tenerlo así.

También se pueden crear variables a nivel de proyecto, si tenemos aplicaciones Fundeweb 2.0 y 2.1 en nuestro grupo lo recomendado es crear la variable de perfil de compilación a este nivel y no a nivel de grupo

Gestión de releases



Etiquetas

```
# etiqueta simple
git tag mi_etiqueta

# Etiqueta anotada
git tag -a v1.0 -m 'Version 1.0'
git tag

git push origin --tags
```

Changelog

Por cada cambio que se quiera registrar hay que crear un fichero en la rama en la ruta *changelogs/unreleased/XXX.yml* con el contenido

```
---
title: Mi título de changelog
merge_request: //opcional//
author: Pablo González de la Peña
type: added|fixed|changed|deprecated|removed|security|performance|other
```

https://gitlab.um.es/help/user/project/releases/index#release-description

Utilidades

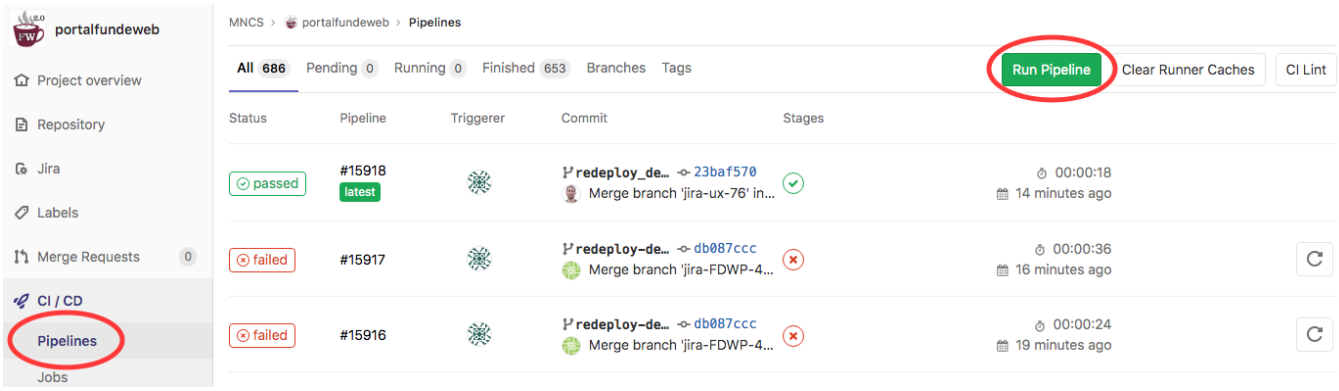
Aquí explicamos algunas mecanismos que hemos desarrollado y que os pueden ayudar en vuestras tareas ordinarias de desarrollo

Redeploys manuales

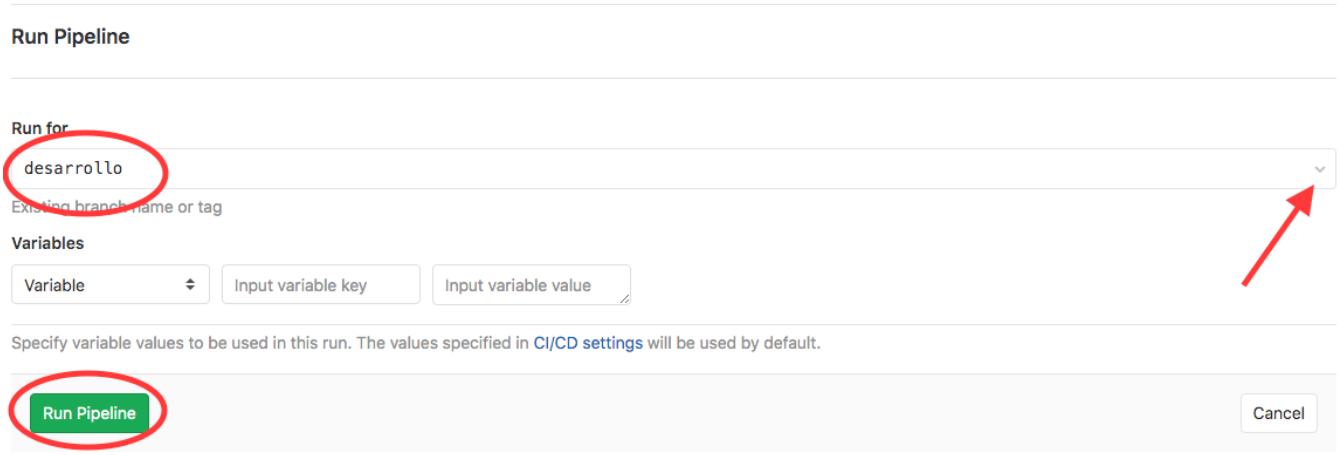
Si por alguna razón queremos hacer un redeploy de un entorno pero no tenemos ningún cambio tenemos 2 opciones

- Ejecutar el pipeline completo de la rama del entorno que corresponda

Vamos a CI/CD Pipelines



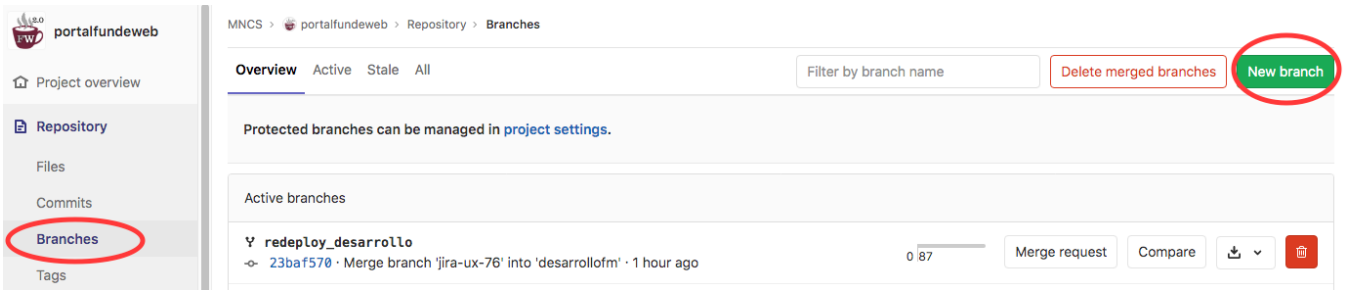
A continuación elegimos la rama en la que queremos que se ejecute y lo ejecutamos



Esto ejecutará el ciclo de vida completo, aunque si no hay cambios en el código de la rama (que es lo normal), finalmente no subirá nada nuevo al servidor y sólo se hará el redeploy.

- Lanzar sólo el cambio en el fichero redeploy (sin pasar por las etapas de build+package, etc…)

Vamos a Repository Branches

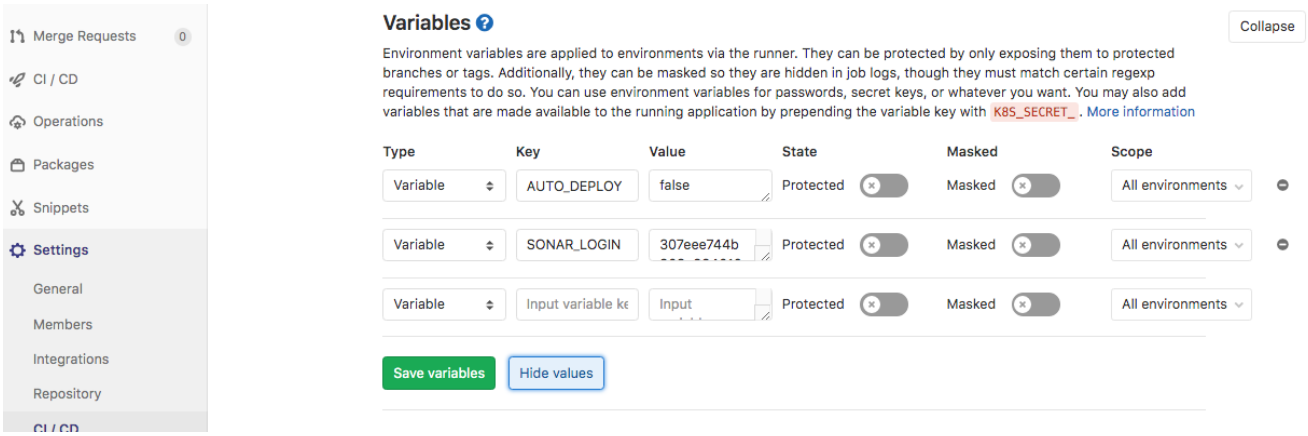


En el nombre de la rama ponemos *redeploy_desarrollo*, *redeploy_preproduccion* o *redeploy_produccion*, en la rama de origen podemos poner la del entorno que queremos redesplegar.

Ojo, porque si ya lo hemos hecho antes, es decir la rama ya existe la tendréis que borrar o seguir los pasos del punto anterior pero usando la rama *redeplo y_XXX* en lugar de la *XXX*

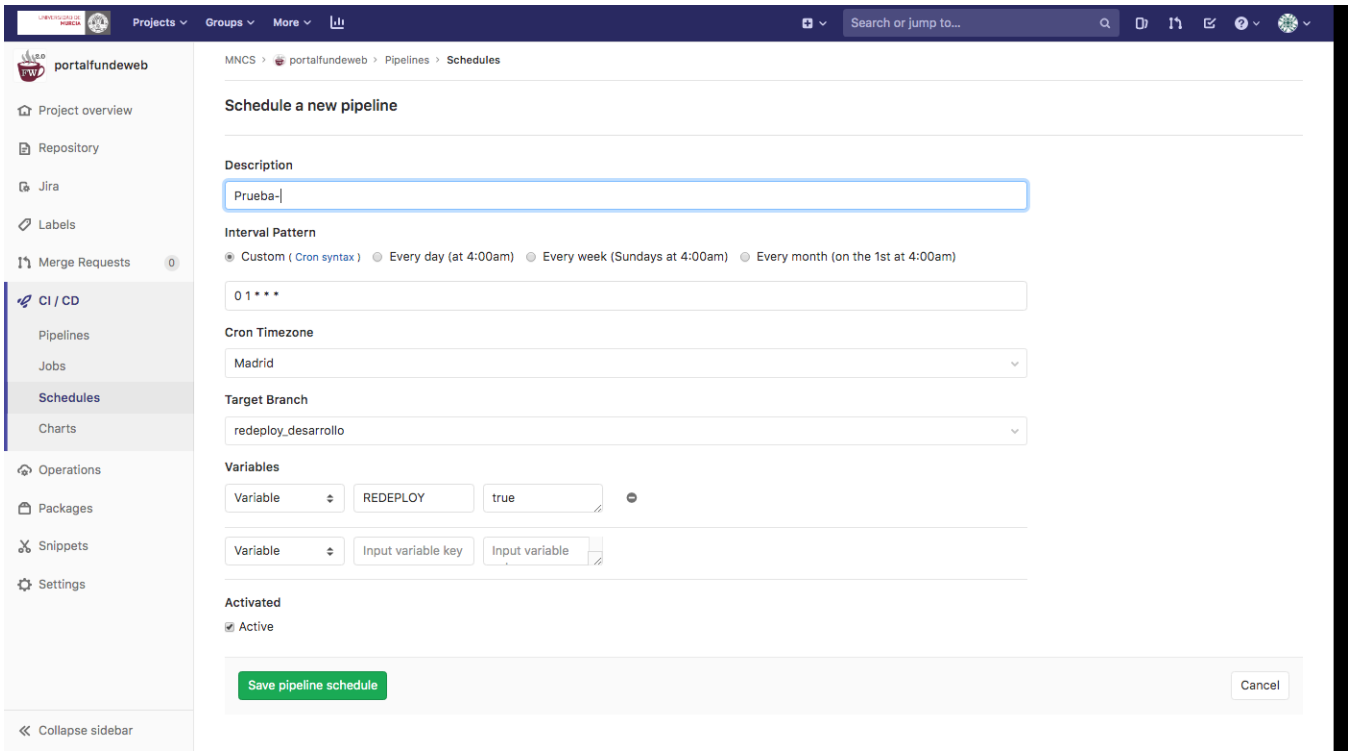
Pasos a producción programados

Lo primero que tenemos que hacer es poner la variable AUTO_DEPLOY a false a nivel de proyecto según el siguiente pantallazo Settings CI / CD , Expandir la zona de Variables y ponerle el valor false en minúsculas y sin espacios.



Esto hará que NINGÚN redespliegue se realice en producción hasta que se ejecute un pipeline de la rama master con la variable REDEPLOY a true.

Puede ser una ejecución programada como aquí



> Ojo, que el formato de programación es el de la herramienta GNU cron, hay que tener cuidado con la sintaxis para no tener ejecuciones no deseadas

O una ejecución normal o con la rama redeploy_produccion como se comenta en el apartado de redespliegues