Prácticas Profesionalizantes 3

Práctica 2: GITLAB y GITFLOW

Objetivos de la práctica:

En esta práctica se pretende conseguir:

- Crear los equipos de trabajo en GitLab
- Adaptar el flujo de trabajo en Git y GitLab al trabajo en equipo.
- Implementar GitFlow.
- Desarrollar nuevas features con GitFlow.
- Lanzamiento de una versión nueva usando GitFlow.

PARTE 1: GITLAB

Formación de equipos

En esta práctica trabajaremos en equipos de 2 o 3 personas, y cada persona debe pertenecer a por lo menos 2 equipos (el propio y uno más, no pueden ser los mismos integrantes cambiando de función).

Cada equipo trabajará con un repositorio seleccionando a uno de los miembros del equipo como dueño (owner) y al resto como desarrollador (developer). Se formará también un equipo en el grupo en el que participarán todos los miembros del equipo

Roles en el equipo (grupo)

Los miembros del equipo tendrán un papel diferente.

Dueño del grupo (owner): Encargado de gestionar el flujo de Git y de supervisar los merge requests, issues y tablero de GitLab.

Desarrollador (developer): Encargado de nuevas características o funcionalidades al código fuente existente.

Trabajando con GitLab

1 - Empezar un proyecto en GitLab. ¿Noto alguna diferencia con GitHub? Enumere las diferencias.

2 - La materia tiene un repositorio en GitLab, nuestro primer trabajo será hacer una bifurcación (fork) de ese repositorio. Nota: Los estudiantes no pueden escribir sobre ese repositorio para poder hacerlo deben pedir una solicitud de cambio “Merge Request”.

https://gitlab.com/isfdyt210/sandbox

3 - Dentro del “sandbox” encontrara un archivo README.md con una seccion “Actividades” realice la primer actividad “directorio estudiantes/” . En esta actividad deberá bifurcar un repositorio, agregar contenido y solicitar el cambio a través de un Merge Request. Cada actividad tiene un archivo README.md dentro del directorio en este archivo estan las instrucciones.

4 - Idem al ejercicio anterior pero en esta ocasión realice la segunda actividad “directorio equipos/”. En está actividad deberá crear un grupo en gitlab, solo debe estar el usuario que creó el grupo en cada equipo.

5 - Agréguese a un grupo, vea los grupos que fueron aceptados en el repositorio, y realice una petición de cambio para agregarse a uno con el formato establecido.

6 - Idem al ejercicio 3 (tres) pero en esta ocasión realice la tercera actividad “directorio cancionero/”. En esta actividad verá como es el flujo de trabajo entre distintos programadores.

PARTE 2: GITFLOW

Nuevo flujo de trabajo para los hitos(issues)

Debemos adaptar el flujo de trabajo en GitLab al trabajo en equipo. En cuanto a la gestión de los issues y tablero del proyecto cambiaremos lo siguiente:

- **Nueva rama**: El responsable seleccionado será el que abra una rama nueva para el desarrollo la subirá a GitLab.
- **Desarrollo**: Se trabaja en la rama. Cualquier compañero puede unirse a la rama y trabajar junto con el responsable.
- **Merge request**: Cuando la nueva característica o hito se ha terminado, el responsable abre un merge request en GitLab
- **Revisión de código:** Los miembros del equipo revisan el código en el merge request
- **Integración del merge request**: Generalmente el responsable del repositorio aprueba el cambio solicitado, y la tarea o hito se integra al merge request.

Para implementar el trabajo en equipo será necesario trabajar sobre ramas remotas compartidas. A continuación repasaremos con más detalle algunos aspectos comandos de Git necesarios.

Comandos Git

Veamos algunos comandos de Git relacionados con el trabajo compartido en repositorios y ramas remotas.

1.- Subir una rama al repositorio remoto:

```
$ git checkout -b nueva-rama
$ git push -u origin nueva-rama
```

2.- Descargar una rama del repositorio remoto:

```
$ git fetch
$ git checkout nueva-rama
```

El comando git fetch se descarga todos los cambios pero no los mezcla con las ramas locales. Los deja en ramas remote tracking a las que les da el nombre del servidor y la rama (origin/nueva-rama).

En el caso del comando anterior, el comando git checkout nueva-rama es equivalente a git checkout -b nueva-rama origin/nueva-rama. Se crea una rama local nueva-rama conectada a la rama origin/nueva-rama.

3.- Actualizar una rama con cambios que otros compañeros han subido al repositorio remoto:

```
$ git pull
```

El comando git pull es equivalente a un git fetch seguido de un git merge. El comando git fetch actualiza la rama remota origin/nueva-rama. El comando git pull es equivalente a hacer:

```
$ git checkout nueva-rama
$ git fetch
$ git merge origin/nueva-rama
```

Subir cambios de la rama actual:

(estando en la rama que queremos subir)

```
$ git push
```

El comando git push funcionará correctamente sin más parámetros si previamente hemos subido la rama con un git push -u.

4.- Comprobar el estado de las ramas locales y remotas:

```
$ git branch -vv
```

Este comando no accede directamente al servidor, sino que muestra la información de la última vez que se accedió a él. Si queremos la información actualizada podemos hacer un git fetch --all antes:

```
$ git fetch --all
$ git branch -vv
```

Es importante recordar que git fetch (a diferencia de git pull) no modifica los repositorios locales, sino que baja las ramas remotas cachés locales.

5.- Información de los repositorios remotos:

```
$ git remote show origin
```

Proporciona información del repositorio remoto, todas sus ramas, del local y de la conexión entre ambos.

```
$ git remote -v update
```

Proporciona información del estado de las ramas remotas y locales (si están actualizadas o hay cambios en algunas no bajadas o subidas).

6.- Borrado de ramas remotas desde el terminar:

```
$ git push origin --delete nueva-rama
$ git remote prune origin
```

Si necesitamos en la rama de feature código que se haya añadido en la rama master.

Podemos hacer un merge de la rama master en la rama de feature para incorporar los avances de código que se han hecho en master y que necesitamos en nuestra nueva rama:

```
$ git checkout nueva-rama
$ git merge master
```

7.- Solución de conflictos en un pull request:

Recordamos lo que hemos visto en teoría sobre la solución de conflictos detectados en un pull request.

Supongamos que hay un conflicto entre la nueva rama y master. GitHub detectará el conflicto en la página de pull request. Para arreglar el conflicto:

```
$ git checkout master
$ git pull
$ git checkout nueva-rama
$ git merge master
# arreglar el conflicto
$ git push
```

Trabajando con GitFlow

El flujo de trabajo GitFlow que vamos a seguir es muy similar al flujo de trabajo Git (recordar la clase de teoría).

Ramas de largo recorrido

En GitFlow se publican las distintas versiones del proyecto en la rama long-lived master y se hace el desarrollo en la rama develop. A partir de ahora **no desarrollaremos directamente en master** sino en develop.

En la página de configuración del repositorio en GitLab en Settings > Branches > Default branch se puede configurar la rama por defecto contra la que se realizarán los commits y la que aparecerá en la página del proyecto. Hay que definir **develop**.

Ramas de feature

Desde el comienzo de trabajo con Git en las prácticas 1 y 2 estamos haciendo un desarrollo basado en ramas de corto recorrido, equivalentes a las ramas de features de GitFlow.

Tal y como se comenta en GitFLow estas ramas saldrán de develop y se integrarán en develop. La diferencia es que en GitFlow estas ramas se integran con la rama de desarrollo manualmente haciendo merge, mientras que nosotros las integramos haciendo un pull request.

Ramas de release

Hasta ahora hemos hecho los releases en la rama master. A partir de ahora seguiremos la estrategia de GitFlow y haremos ramas de release que salen de develop y se integran en master y en develop.

Haremos también la integración haciendo merge request.

Pasos a seguir

1.- El responsable de GitLab de cada equipo se debe encargar de crear la rama develop y configurarla como rama principal del proyecto en GitLab. Todos los otros miembros deberán descargarla y moverse a ella en sus repositorios locales. Esta rama pasará a ser la de desarrollo principal.

2.- El desarrollador modificará algún archivo del grupo, agregar otra estrofa al cancionero, para que también se lancen los merge request en la rama develop (además de en la rama master).

3.- Hacer un Merge Request (MR) de prueba en la rama develop para comprobar que todo funciona bien.

4.- Cread tres issues distintos, simulando tres nuevas funcionalidades. Deben ser issues muy sencillos agregar más estrofas al cancionero. Cada uno de los miembros del equipo será el responsable de uno de los issues.

5.- El responsable de GitLab configurará el repositorio para obligar a que cualquier merge request tenga que tener la revisión de una persona distinta del responsable del MR. (Investigar)