PLAN DE GESTIÓN DEL CÓDIGO FUENTE

Grupo:
decide-single-juanymedio-fork-3
Miembros:
Julio Navarro Rodríguez
Pablo Martínez Valladares
Manuel Palacios Pineda

1. Introducción

El presente plan de gestión del código fuente ha sido elaborado con el propósito de establecer una estrategia clara y eficiente para el control de versiones, gestión de ramas (branching) y manejo de commits en el desarrollo de nuestro proyecto.

Este documento ofrece directrices específicas destinadas a optimizar la colaboración entre los desarrolladores, asegurando la integridad, la trazabilidad y la estabilidad del código a lo largo del ciclo de vida del software

2. Branching

Para implementar los incrementos de funcionalidad en el proyecto, se ha diseñado una estrategia de creación de ramas que nos ayudará a tener bajo control el progreso en cada tarea. A continuación se listan los distintos tipos de ramas que se pueden encontrar en nuestro repositorio:

- **master**: representa la rama principal de producción. Contiene el código estable y probado que se puede implementar en el entorno de producción.

- **develop**: en esta rama se integran todos los incrementos funcionales desarrollados por el equipo. Es el punto de partida para nuevas *features*. Se considera una rama de desarrollo estable. Se crea al empezar el proyecto a partir de la rama *master*.

- **Feat-(feature-name)**: cada nuevo incremento funcional que se vaya a integrar en el proyecto se desarrolla en su propia rama individual, derivada de la rama *develop*. Una vez se termina su desarrollo, se fusiona de nuevo con la rama *develop*.

- **Management-(management-name)**: se crean a partir de la rama *develop* para ajustar ficheros de configuración, scripts de integración continua o scripts de despliegue.

- **Fix-(feature-name)**: se crean para corregir errores detectados tras fusionar la rama *Feat-[feature-name]* con *develop*. Una vez se solucionan los errores, se vuelve a fusionar con la rama *develop*.

- **Hotfix-(version-tag)**: se crean para corregir problemas críticos en producción. Permiten solucionar errores urgentes mientras se mantiene la

estabilidad del código. Una vez se resuelve el problema, se fusiona con las ramas *master* y *develop*, para confirmar que la corrección se aplique tanto a la versión actual en producción como a la rama de desarrollo.

- **Release-(version-tag)**: Se utilizan para preparar el código para una nueva versión. Aquí se llevan a cabo tareas como pruebas finales, corrección de errores menores y ajustes de última hora. Una vez que está lista, se fusiona con la rama *master* para una nueva versión y con *develop* para incorporar esos cambios en el desarrollo en curso.

3. Commits

Para aportar claridad y comprensión en el historial de cambios del repositorio, se va a seguir una estructura fija a la hora de redactar los mensajes de los commits. Los commits deben ser atómicos, para mayor facilidad de seguimiento de los cambios. La estructura seguida en el proyecto será la siguiente:

- **Título**: (tipo-de-cambio)(tarea-asociada[opcional]):(nombre-descriptivo)

- **Descripción**: (descripción-cambios)

Como cabecera del título, existen los siguientes tipos de cambios:

- **feat**: representa la adición de una nueva *feature* para mejorar la funcionalidad del software.

- **fix**: se usa cuando se corrige un error o un problema específico en el código.

- **test**: se usa al añadir pruebas o realizar cambios relacionados con las pruebas del proyecto.

- **docs**: utilizado para cambios o actualizaciones en la documentación.

- **management:** utilizado para tareas relacionadas con despliegue o integración continua.

Un ejemplo de commit podría ser el siguiente:

fix(importar_censo): Se han modificado los ficheros de prueba de la característica.

Los ficheros temporales no pasaron las pruebas de integración continua , por lo que se va a probar con ficheros persistentes.

4. Pull Requests

Las pull request son un aspecto fundamental de la gestión de código en proyectos en equipo. Son solicitudes que hace un colaborador para integrar los cambios de una rama en otra. De esta forma, otros colaboradores del proyecto pueden revisar los cambios y dar feedback.

En nuestro proyecto los principales casos en los que se deben realizar pull requests son:

- *feature - develop*

- *develop - master*

Para que una pull requests sea aceptada, debe cumplir una serie de requisitos:

- El código ha sido probado.

- La fusión no genera conflictos.

- Cumple los requisitos de GitHub Actions.

En el caso que alguno de los requisitos anteriores no se cumplieran, la pull request no podrá ser aceptada. En este punto se presentan dos opciones:

- Cancelar la pull request.

- Hacer commits adicionales para solucionar los problemas y volver a verificar los requisitos de aceptación anteriores.