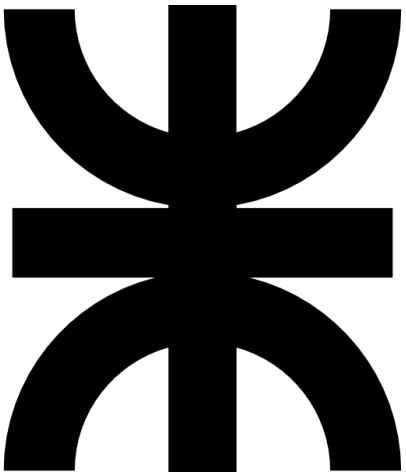

Universidad Tecnológica Nacional

Facultad Regional Santa Fe

Cátedra: Ingeniería y calidad del Software

Trabajo práctico N° 1

Alumnos:
Flores, José Ignacio
Garello, Enzo
Paredes, Sebastian
Pallotti, Francisco Fernando

b) Para no subir cambios en configuraciones locales debemos configurar el archivo .gitignore detallando los ficheros o los archivos que deseamos que no se almacenen en el repositorio, ej: ficheros de instalación de librerías externas como así también archivos de keystorage por seguridad o cualquier otro archivo.

En nuestro caso particular, vamos a configurarlo para que ignore los archivos de configuración de Visual Studio Code. Para ello nos hemos servido de una plantilla disponible en internet, que especifica los diferentes tipos de archivos que pueden contener las settings de usuario de VSCode.

e) Para llevar a entorno productivo release 1, siguiendo el esquema de git flow, primero debemos fijarnos que en release 1 tenemos todos los cambios, incluidos los de la rama main (si no los tenemos, hacemos git pull origin main). Luego, realizamos un PR de release 1 a main. Si no hay conflictos, realizamos el merge entre las ramas y luego publicamos la nueva versión creando un tag y luego un release y el número de versión.

f) Para corregirlo, tenemos que crear una una rama hotfix desde la rama main, en la cual solucionamos el error, realizamos un commit y hacemos push. Por último lo mergeamos a la rama main.

g) Creamos un pull request desde main hacía develop, para traernos la última versión “línea base” del software hacia nuestra rama de desarrollo.

j) Para llevar a producción la feature creada, generamos el PR de esta rama(feature 2h) a la rama develop, paso siguiente verificamos si no hay conflictos para luego hacer el merge sobre develop. Luego creamos una rama release 2. Para finalizar y llevar a producción estos cambios hacemos un pull request desde Release 2 hacia main, luego generamos un tag, y en este caso, como se agregaron funcionalidades mínimas, se debe actualizar el número de versión a 1.1.0.

3)
a)
Podríamos usar el archivo de readme para realizar una descripción del proyecto y los archivos que lo conforman. Adicionalmente podríamos indicar que existe un changelog del proyecto, escrito de acuerdo al estándar propuesto por la cátedra. El resultado de esta propuesta puede encontrarse en el repositorio compartido.

b) Si un externo realiza una modificación, se le pedirá que los realice, primero, en una rama separada (cuyo nombre debe comenzar con feature), luego que en los commit agregue un comentario que detalle lo realizado y genere el PR hacia develop para que se haga el code review.
En la herramienta para realizar PR en GitHub, se permiten agregar diferentes datos a este pull request a fin de facilitar al revisor la tarea de evaluar la solicitud.

A su vez, los datos que sería conveniente pedirle a quien genera el Pull request son:

- Titulo del PR: Describir en una línea el propósito general del cambio
- Descripción detallada: Debe explicar el motivo del cambio, puesto que el **cómo** se encuentra en el código.
- Referencias a Issues/ Historias de usuario/Funcionalidades implementadas: Debe contener el código del requerimiento al que atiende, o del issue que fixea.

Por otro lado, GitHub nos ofrece diferentes funcionalidades que ayudan a entender el PR y solucionar conflictos que impidan el merge, como lo son:

- Vista de cambios: Nos muestra las líneas de código que fueron modificadas, agregadas o eliminadas.
- Comentarios en el código: Los revisores pueden añadir comentarios directamente sobre las líneas específicas de código dentro del PR, para realizar aclaraciones.
- Plantillas de PR: GitHub permite configurar plantillas de PR dentro de un repositorio, que se pueden personalizar para que incluyan secciones predeterminadas, facilitando la estandarización.
- Historial de commit y log: Los commits asociados al PR muestran el historial de los cambios realizados durante la creación y ajustes del PR, lo cual permite rastrear la evolución de la propuesta.