POLÍTICA DE LA GESTIÓN DE LOS COMMITS

Fecha de Emisión: 17/10/2024
Descripción: Normativa para la creación, uso y gestión de commits en el proyecto para garantizar la colaboración estructurada y controlada.

Introducción y Propósito

Esta política tiene como objetivo proporcionar un conjunto claro de reglas y convenciones para la creación de commits dentro del proyecto. Un historial de commits limpio, coherente y bien estructurado es fundamental para un proyecto exitoso, ya que facilita el seguimiento de cambios, la revisión de código y la colaboración entre los miembros del equipo.

El uso de **commits atómicos** y el seguimiento de las convenciones de **Conventional Commits** asegura que cada cambio sea fácilmente comprensible, reversible y rastreable. De esta manera, se mejora la calidad del código, la claridad en las revisiones y la eficiencia de la integración continua.

Estructura del Mensaje de Commit

Cada commit debe seguir el formato **Conventional Commits y también mantener la atomicidad**, que tiene la siguiente estructura:

```
<tipo>(<área>): <descripción breve>

[opcional] cuerpo del mensaje
[opcional] pie de mensaje
```

Tipos de Commits

Utilizaremos los siguientes tipos de *commits* para definir el propósito de cada cambio:

- **feat**: Para la inclusión de nuevas funcionalidades.
- **fix**: Para correcciones de errores.
- **docs**: Para cambios en la documentación (solo documentación).
- **style**: Para cambios en el estilo del código (formateo, punto y coma, etc.) que no afectan la lógica.

- **refactor**: Para cambios de código que mejoran la estructura o el rendimiento, sin cambiar la funcionalidad.
- **test**: Para agregar o modificar tests.
- **chore**: Para cambios en el proceso de construcción o herramientas auxiliares y bibliotecas, como la configuración de *CI/CD*.
- **perf**: Para cambios en el código que mejoran el rendimiento.
- **build**: Para cambios que afectan el sistema de construcción o dependencias externas.
- **ci**: Para cambios en los archivos de configuración de *CI* y scripts.

Atomicidad de Commits

Un commit debe ser **atómico**, lo cual significa que:

- **Cada commit debe contener solo un cambio significativo**. Si haces varios cambios no relacionados, cada uno debe ser un commit separado.
- **No se deben combinar múltiples tipos de cambios en un solo commit** (por ejemplo, cambios de estilo y nuevas funcionalidades en un mismo commit).
- **Cada commit debe ser capaz de compilarse o funcionar de manera independiente** sin errores para que, si se necesitara, el repositorio pudiera revertirse a un commit específico sin interrupciones.

Cuerpo y Pie del Mensaje de Commit

- **Cuerpo (opcional)**: El cuerpo tiene la funcionalidad de especificar aún más el commit, si la descripción breve no es suficiente.También se puede incluir la motivación del cambio.

- **Pie del mensaje (opcional)**: Se usa para para cerrar *issues*, referenciar documentos o tareas, y proporcionar detalles específicos del cambio. Por ejemplo:

```
Closes #123
See also #456
```

Buenas Prácticas en el Mensaje del Commit

- Hay que usar el **modo imperativo** en la descripción breve del commit (e.g., “agrega” en lugar de “agregado”).
- La **descripción breve** debe tener un máximo de **50 caracteres**. Para mayor detalle se deberá usar el cuerpo del commit.
- El cuerpo deberá tener como máximo de **72 caracteres** para legibilidad.
- La descripción breve debe ser **clara y concisa**.

Ejemplos de Commits

Commit de nueva funcionalidad:

```
feat(login): implementar autenticación con Google

Se ha añadido la funcionalidad de autenticación usando el SDK
de Google para mejorar la experiencia del usuario.
Closes #45
```

Commit de corrección de error:

```
fix(api): corregir error en la validación de email

El endpoint de registro no validaba correctamente el formato
del email, lo cual causaba que usuarios se registraran con
correos no válidos.
```

Commit de estilo:

```
style(button): mejorar el padding de los botones

Se ha ajustado el padding de los botones en el formulario de
login para una mejor presentación visual.
```

Validación, Revisión y Aprobación de la Convención

Para validar que la convención de commits es correcta cada commit deberá ser avisado a los revisores del proyecto, comprobando que sigue todas y cada una de las reglas estipuladas en este documento.

Además debe ser revisado antes de su integración en ramas principales. Durante la revisión de código, se debe verificar que el mensaje siga esta convención, que los cambios sean atómicos y que el commit sea funcional de forma independiente.

Flujo de Trabajo de Commits y Pull Requests

1. Trabajar en una Rama Nueva

- Para cada nueva tarea o corrección de error, crea una rama específica siguiendo la política de gestión de ramas (por ejemplo, `feature/nueva-funcionalidad`, `fix/error-en-login`).

2. Commits Atómicos

- Realiza un commit por cada cambio completo o tarea específica. Evita mezclar múltiples cambios en un solo commit.

3. Sincronización Regular

- Antes de hacer push, sincroniza tu trabajo con la rama remota y resuelve cualquier conflicto que pueda surgir.

4. Pull Requests

- Una vez que el trabajo en tu rama esté listo, abre un pull request (PR) para fusionar los cambios en la rama de desarrollo (`develop`). El PR debe ser revisado y aprobado por al menos un miembro del equipo antes de proceder con la fusión.

5. Revisión del Código

- Los revisores deben asegurarse de que los commits sigan las convenciones y que el código pase las pruebas. El PR debe ser fusionado solo después de pasar la revisión.

6. Fusión y Eliminación de la Rama

- Después de que un pull request sea aprobado y fusionado, la rama correspondiente debe ser eliminada para mantener el repositorio limpio y organizado.

Reglas Adicionales

1. **Evitar Commits con Código Incompleto o Roto**:
    - No hagas commits con código que rompa la aplicación o con funcionalidades a medio terminar. Si es necesario, usa ramas de trabajo para mantener tu código en progreso.
2. **Pruebas Obligatorias Antes del Commit**:
    - Asegúrate de que tu código pase todas las pruebas locales antes de hacer un commit. Si existen herramientas de integración continua (CI), verifica que las pruebas automáticas no fallen antes de realizar el push.
3. **Sin Mensajes de Commit Genéricos**:
    - Mensajes como “fix” o “update” sin contexto no están permitidos. Los mensajes deben ser explícitos respecto al cambio realizado.
4. **Commits Multilínea**:
    - Si el cambio no es trivial, proporciona una descripción detallada del mismo en la segunda línea del mensaje del commit, después del título breve.

5. **Uso de Ramas Correctas**:
    - Asegúrate de hacer commits en la rama correcta según la naturaleza del cambio (por ejemplo, ramas `feature/` para nuevas funcionalidades, `fix/` para correcciones).
6. **Política de Protección de la Rama `main`**:
    - La rama `main` debe estar protegida para que no se puedan hacer commits directos. Los cambios deben realizarse mediante pull requests revisados por otro miembro del equipo.

Firma de los integrantes:

- Espinosa Naranjo, Pablo:

- Garate Fuentes, Yesica:

- Harana Mancilla, Rafael:

- Pizarro López, Eduardo:

- Portillo Sánchez, Alonso:

- Sevillano Barea, Alejandro: