**Politicas de MERGE Petly**

**9B IDGS**

Andrade Carbajal Jesús Ricardo

[20223l001005@utcv.edu.mx](mailto:20223l001005@utcv.edu.mx)

Gaspar Cruz Ismael

[20213l001052@utcv.edu.mx](mailto:20213l001052@utcv.edu.mx)

Hernández Rodríguez José Aaron

[20223l001008@utcv.edu.mx](mailto:20223l001008@utcv.edu.mx)

Ramirez Vega Iosef Yamil

[20223l001016@utcv.edu.mx](mailto:20223l001016@utcv.edu.mx)

Alvízar Martínez Alexis

[20223l001038@utcv.edu.mx](mailto:20223l001038@utcv.edu.mx)



**I. Introducción**

Este documento establece las directrices formales para realizar **merge de código** dentro del flujo de trabajo colaborativo del proyecto *Petly*. El objetivo es evitar errores críticos en la rama principal (main), garantizar la calidad del código, y mantener la estabilidad del sistema, alineado con las buenas prácticas de GitHub y la metodología **Git Flow**.

**II. Requisitos Previos al Merge**

Antes de realizar cualquier acción de merge, se deben cumplir los siguientes puntos:

1. ✅ El Pull Request (PR) fue revisado y aprobado por al menos un miembro del equipo.
2. ✅ La rama se encuentra **actualizada con main** y sin conflictos.
3. ✅ Todas las pruebas locales y visuales (UI/UX) han sido validadas por el desarrollador responsable.
4. ✅ Se ha documentado correctamente el cambio realizado en la descripción del PR.
5. ✅ El código subido no afecta otras funciones sin justificación técnica.
6. ✅ En caso de funcionalidad nueva o crítica, se notificó previamente en la daily o por mensaje.

**III. Tipos de Merge Permitidos**

Con base en el enfoque **Git Flow**, todos los desarrolladores trabajan directamente sobre la rama main a través de Pull Requests, por lo que el proceso de merge debe ser controlado:

|  |  |  |
| --- | --- | --- |
| **Tipo de merge** | **¿Cuándo usarlo?** | **Justificación** |
| **Squash and merge** | ✔️ Cambios pequeños o medianos | Ideal para limpiar el historial de commits innecesarios |
| **Merge commit** | ✔️ Funcionalidades completas que requieren preservar todos los commits | Conserva trazabilidad del desarrollo por pasos |
| **Rebase and merge** | 🚫 No permitido | Rompe la linealidad de Git Flow y puede generar conflictos en equipo |

**IV. Reglas y Responsabilidades**

* Solo pueden ejecutar merge los desarrolladores **autores del PR**, una vez aprobado.
* **No se permite el uso de push --force en ninguna rama compartida.**
* Todo merge debe hacerse en horario de trabajo definido o con consentimiento del equipo.
* El **SCRUM Master** será el responsable de resolver cualquier disputa o error relacionado con merges.

**V. Roles asignados en el proceso**

|  |  |  |
| --- | --- | --- |
| **Miembro** | **Rol** | **Responsabilidad específica** |
| **José Aaron Hernández Rodríguez** | FrontEnd | Validación visual y compatibilidad con diseño |
| **Ramírez Vega Iosef Yamil** | BackEnd/Datos | Validación de integridad de datos y consumo de API |
| **Gaspar Cruz Ismael** | Product Owner | Confirmación funcional de entregables importantes |
| **Alvízar Martínez Alexis** | Desarrollador | Asegurar que el código respeta estándares de calidad |
| **Andrade Carbajal Jesús Ricardo** | SCRUM Master | Verificación de cumplimiento de políticas y control de conflictos |

**VI. Faltas y sanciones aplicables**

La inobservancia de estas políticas pone en riesgo el estado del repositorio principal. Por ello, se aplicarán las siguientes sanciones, tomando como base la **Ley Federal del Trabajo de México**:

|  |  |  |
| --- | --- | --- |
| **Falta** | **Consecuencia** | **Fundamento legal** |
| Merge sin revisión o sin PR | Advertencia formal o eliminación del commit | Art. 47 fracc. X – "desobedecer instrucciones" |
| Merge con errores críticos por falta de pruebas | Reasignación de tareas y sanción académica interna | Art. 134 fracc. II – "cuidado del trabajo asignado" |
| Merge fuera de horario o sin autorización | Suspensión temporal de acceso al repositorio | Art. 47 fracc. II – "falta de probidad" |
| Modificación directa en main sin autorización | Registro de falta grave en bitácora del equipo | Art. 47 fracc. I – "engañar al patrón o compañeros" |

**VII. Observaciones Finales**

Las presentes políticas aplican durante todo el ciclo de desarrollo de Petly. Cualquier excepción deberá ser aprobada por el **SCRUM Master** y comunicada a todo el equipo.

Revisiones o actualizaciones a este documento deberán registrarse con fecha y motivo.

**VIII. Nomenclatura de Ramas**

La **nomenclatura de ramas** debe seguir un esquema estándar para asegurar consistencia y facilitar la navegación y colaboración dentro del repositorio. A continuación, se define el formato que se debe utilizar para la creación de ramas.

**A. Formato de las ramas**

Cada rama debe estar nombrada siguiendo el siguiente esquema:

php-template

Copiar

<tipo>/<proyecto>/<descripción-corta>

* **tipo**: Se refiere al tipo de tarea que se está realizando. Los tipos más comunes son:
  + feat: Nueva funcionalidad.
  + fix: Corrección de errores.
  + refactor: Cambios de refactorización o mejoras de código.
  + docs: Cambios relacionados con la documentación.
  + test: Nuevas pruebas o modificaciones de pruebas existentes.
  + chore: Tareas generales, como actualizaciones de dependencias o cambios de configuración.
* **proyecto**: Representa el área del proyecto en el que se está trabajando. Puede ser:
  + frontend: Si es un cambio en el frontend.
  + backend: Si es un cambio en el backend.
  + database: Si es un cambio relacionado con la base de datos.
  + devops: Si es un cambio relacionado con la infraestructura.
* **descripción-corta**: Una breve descripción del cambio realizado, escrita en minúsculas y separada por guiones si es necesario.

**B. Ejemplos de nomenclatura de ramas**

1. **Para nuevas funcionalidades (features)**:
   * feat/frontend/agregar-boton-login
   * feat/backend/integrar-api-mascotas
2. **Para correcciones de erro res (fixes)**:
   * fix/frontend/corregir-bug-login
   * fix/backend/solucionar-error-consulta-database
3. **Para cambios de refactorización (refactor)**:
   * refactor/frontend/optimizar-renderizado
   * refactor/backend/reorganizar-codigo-login
4. **Para cambios en la documentación (docs)**:
   * docs/actualizar-readme
   * docs/agregar-instrucciones-despliegue
5. **Para nuevas pruebas (test)**:
   * test/frontend/pruebas-login
   * test/backend/pruebas-api-mascotas
6. **Para tareas generales (chore)**:
   * chore/devops/actualizar-dependencias
   * chore/frontend/eliminar-codigo-obsoleto

**C. Reglas adicionales**

* Las ramas deben ser creadas **desde la rama dev** y deben tener un nombre claro que refleje el cambio que se está haciendo.
* No se deben usar nombres genéricos como dev o master para ramas de características, correcciones o tareas.
* Las ramas deben ser **cerradas** (borradas) después de realizar el merge exitoso con main.

**D. Sanciones por incumplimiento**

El incumplimiento de estas reglas de nomenclatura puede causar confusión en el equipo, generando retrasos en las entregas y errores de integración. Las consecuencias por no seguir esta nomenclatura son las siguientes:

|  |  |  |
| --- | --- | --- |
| **Falta** | **Consecuencia** | **Fundamento legal** |
| Usar nombres ambiguos o incorrectos para ramas | Llamado de atención o corrección inmediata | Art. 47 fracc. X – "desobedecer instrucciones" |
| Crear ramas directamente desde main | Reasignación de tareas y revisión de flujo de trabajo | Art. 134 fracc. II – "ejecutar con cuidado el trabajo" |
| No seguir el esquema de nomenclatura definido | Advertencia formal o sanción interna | Art. 47 fracc. II – "falta de probidad" |