**Politicas de PULL REQUEST 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**

El presente documento tiene como finalidad establecer las normas y lineamientos que deben seguir los integrantes del equipo de desarrollo del proyecto *Petly* al momento de crear y gestionar Pull Requests (PR) dentro del flujo de trabajo definido por la metodología Git Flow. Estas políticas buscan mantener la calidad del código, la organización del repositorio, la trazabilidad del desarrollo y una cultura de colaboración profesional.

**II. Estructura y Proceso para Pull Requests**

**A. Flujo base**

* Todo el desarrollo se debe realizar en **ramas individuales** creadas desde la rama principal de desarrollo (main) siguiendo la estrategia **Git Flow**.
* Al finalizar una tarea o funcionalidad, se deberá **crear un Pull Request** desde la rama individual hacia main.

**B. Nombres y convenciones**

* El nombre del PR debe seguir el formato:  
  tipo: descripción breve  
  Ejemplos:
  + feat: agregar vista de detalles de mascota
  + fix: corregir validación en formulario de registro
  + docs: actualizar README
* Los tipos válidos son:
  + feat – nueva funcionalidad
  + fix – corrección de errores
  + refactor – reestructuración de código sin cambiar funcionalidad
  + docs – cambios en documentación
  + style – ajustes de formato o estilo
  + test – pruebas

**C. Descripción obligatoria del PR**

Todo PR debe incluir:

1. Descripción clara de los cambios realizados.
2. Tarea correspondiente (si aplica).
3. Screenshots o evidencia visual cuando se trate de frontend.
4. Indicación de posibles efectos colaterales (bugs, componentes afectados).

**D. Revisión y aprobación**

* Cada PR **debe ser revisado y aprobado** por al menos **otro miembro del equipo**.
* No se puede hacer merge sin aprobación previa.
* Se deben usar los comentarios para sugerir cambios, identificar errores y resolver dudas técnicas.

**E. Buenas prácticas**

* Subir código funcional y probado localmente.
* Mantener cambios organizados y con commits claros.
* Evitar mezclar tareas diferentes en un mismo PR.
* No subir archivos innecesarios (.env, .DS\_Store, node\_modules, etc.).

**III. Roles y responsabilidades**

* **José Aaron Hernández Rodríguez** – Asegura que el diseño frontend cumpla estándares visuales antes de aprobar PR relacionados con interfaz.
* **Ramírez Vega Iosef Yamil** – Revisa integridad de datos y seguridad en PRs relacionados con backend y base de datos.
* **Gaspar Cruz Ismael** – Supervisa coherencia general del proyecto y da aprobación final como Product Owner.
* **Alvízar Martínez Alexis** – Participa activamente en revisión cruzada de funciones.
* **Andrade Carbajal Jesús Ricardo** – Como SCRUM Master, se asegura del cumplimiento de los procedimientos definidos y resuelve conflictos.

**IV. Sanciones por incumplimiento**

En caso de omitir estas políticas o afectar negativamente el flujo de trabajo del equipo, se aplicarán sanciones internas proporcionales, respaldadas por el marco de la **Ley Federal del Trabajo de México**, especialmente en contextos laborales o institucionales formales.

**Faltas y sus consecuencias:**

|  |  |  |
| --- | --- | --- |
| **Falta** | **Consecuencia** | **Referencia Legal** |
| Realizar merge directo sin revisión o PR | Llamado de atención por escrito / pérdida de puntos en evaluación | Art. 47 fracc. X – "desobedecer instrucciones" |
| Subir código con errores evidentes o sin probar | Reasignación de tareas / Evaluación negativa | Art. 134 fracc. II – "ejecutar con cuidado el trabajo" |
| Falsificar revisión o aprobación de PR | Advertencia formal / Exclusión temporal del repositorio | Art. 47 fracc. II – "falta de probidad" |
| No participar en revisiones durante dos sprints consecutivos | Reporte al coordinador académico o docente | Art. 134 fracc. III – "colaborar con otros trabajadores" |

**V. Observaciones finales**

Estas políticas están sujetas a revisión continua por el SCRUM Master y el Product Owner. Cualquier cambio deberá notificarse por escrito y discutirse en sesión de equipo.

**VI. 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" |