Normativa de Commits en el Proyecto – Proyecto Integrador I

Integrantes del Equipo

- Juan David Ocampo Martínez – A00401856
- Luna Catalina Martínez Vasquez – A00401964
- Renzo Fernando Mosquera Daza – A00401681
- Jhon Alexander Lamus Mora – A00372128
- Juan José Ramos Henao – A00294612
- Sebastian Jimenez Galvis – A00401374

---

1. Introducción

El propósito de esta normativa es garantizar que el historial de commits sea claro, consistente y detallado, facilitando la colaboración entre los miembros del equipo y la integración en procesos de CI/CD. Una nomenclatura unificada mejora la trazabilidad de los cambios y la detección de errores o mejoras en el proyecto.

---

2. Convención de Commits

Se adopta la metodología **Conventional Commits**, adaptándola a las necesidades particulares del proyecto. Cada commit **debe** estar redactado en **minúscula** y la descripción debe separar cada palabra con un **guion bajo (_)**. De esta forma, el formato será:

tipo(scope): breve_descripción_del_cambio

- **tipo**: Categoriza la naturaleza del cambio.
- **scope** (opcional): Indica el módulo o parte específica del proyecto afectada.
- **breve_descripción_del_cambio**: Resumen conciso del cambio, con las palabras separadas por guion bajo.

---

3. Tipos de Commits

Para una mayor claridad, se incluyen los siguientes tipos de commits:

- **feat**: Introducción de nuevas funcionalidades.
  Ejemplo: feat(auth): implementar_registro_de_usuarios

- **fix**: Corrección de errores y bugs en el código.
  Ejemplo: fix(api): corregir_validacion_de_datos

- **docs**: Cambios o mejoras en la documentación.
  Ejemplo: docs(readme): actualizar_instrucciones_de_instalacion

- **style**: Ajustes en el formato del código (indentación, espacios, etc.) sin modificar la lógica.
  Ejemplo: style(ui): ajustar_espaciado_y_formato

- **refactor**: Reestructuración del código sin alterar su funcionalidad.
  Ejemplo: refactor(database): reorganizar_relaciones_entre_tablas

- **perf**: Mejoras relacionadas con el rendimiento o la optimización del sistema.
  Ejemplo: perf(api): optimizar_consulta_sql

- **test**: Adición o modificación de pruebas unitarias o de integración.
  Ejemplo: test(auth): agregar_tests_para_validar_token

- **chore**: Tareas de mantenimiento, actualizaciones de dependencias o ajustes generales.
  Ejemplo: chore(deps): actualizar_versiones_de_librerias

- **build**: Cambios que afectan el sistema de construcción (build system).
  Ejemplo: build: actualizar_configuracion_de_build

- **ci**: Modificaciones en la configuración o scripts de integración continua.
  Ejemplo: ci: configurar_pipeline_de_integracion_continua

- **revert**: Revertir cambios o commits previos.
  Ejemplo: revert(ui): revertir_cambios_erroneos

- **merge**: Integración de ramas o fusiones de implementaciones.
  Ejemplo: merge: fusionar_rama_de_nuevas_funcionalidades

- **security**: Cambios enfocados en la implementación o mejora de aspectos de seguridad.
  Ejemplo: security(auth): reforzar_medidas_de_seguridad

- **i18n**: Cambios relacionados con la internacionalización y localización.
  Ejemplo: i18n(ui): agregar_traducciones_español

- **config**: Ajustes y modificaciones en archivos y parámetros de configuración.
  Ejemplo: config: actualizar_variables_de_entorno

---

4. Uso del Scope

El **scope** especifica la parte del proyecto que se ve afectada. Se recomienda definir scopes bien identificados y, en caso de afectar varias áreas, dividir el commit en partes separadas. Algunos ejemplos de scopes incluyen:

- **auth**: Funcionalidades de autenticación y autorización.

- **users**: Gestión y perfiles de usuarios.

- **database**: Cambios en la estructura de datos, migraciones o consultas.

- **api**: Modificaciones en endpoints y lógica de negocio.

- **ui**: Ajustes en la interfaz de usuario.

- **docs**: Documentación y guías de usuario.

- **tests**: Pruebas unitarias o de integración.

- **build**: Scripts de compilación y empaquetado.

- **ci**: Configuración y scripts del entorno de integración continua.

- **config**: Archivos de configuración y variables de entorno.

- **deployments**: Scripts o configuraciones para despliegues.

- **security**: Implementación o mejora de medidas de seguridad.
- **performance**: Cambios dirigidos a mejorar el rendimiento general.
- **analytics**: Modificaciones en el seguimiento o análisis de datos.
- **logging**: Ajustes y mejoras en el registro de eventos.

---

5. Reglas Adicionales

- **Minúsculas obligatorias**: Tanto el tipo como la descripción del commit deben escribirse en minúscula.
- **Guion bajo para separar palabras**: En la descripción se utilizará el guion bajo (_) para separar las palabras, evitando espacios.
- **Claridad y Concisión**:
    - Los mensajes deben ser claros, evitando expresiones genéricas como “arreglar bugs” sin especificar el contexto.
    - Cada commit debe abordar un cambio único y concreto. Si el cambio abarca varias áreas, se recomienda dividirlo en múltiples commits.
- **Uso correcto de verbos**: Iniciar la descripción con un verbo en infinitivo (implementar, corregir, actualizar, etc.).
- **Abreviaciones**: Se deben evitar abreviaturas ambiguas o no reconocidas; en caso de usar alguna, asegurarse de que sea comprensible para todo el equipo.
- **Coherencia**: Mantener la uniformidad en el estilo de escritura de los mensajes para garantizar la fácil lectura y trazabilidad en el historial de commits.