Documento de normalización de Github

—

1. Creación y nomenclatura de ramas

Para facilitar la organización y el seguimiento del proyecto, las ramas deben seguir una estructura de nombres estandarizada basada en el tipo de trabajo que se va a realizar y el alumno responsable de la tarea.

1.1 Tipos de ramas

Tipo de rama	Prefijo	Descripción
Principal	`main`	Rama principal del proyecto. Solo se fusionan cambios revisados y aprobados.
Desarrollo	`develop`	Rama donde se integran los cambios antes de pasar a `main`.
Funcionalidad	`feature`	Ramas dedicadas a nuevas funcionalidades.
Corrección de errores	`fix`	Ramas para corregir errores detectados.
Mejora	`enhancement`	Pequeñas mejoras o refactorizaciones del código.
Documentación	`docs`	Ramas para actualizar o agregar documentación.

1.2 Formato de nombres de ramas

Las ramas deben crearse siguiendo este formato:

```
[prefijo]/[descripción-corta]-[siglas-alumno]
```

Ejemplos:

- `feature/login-form-JP` → Implementación del formulario de login por Juan Pérez.
- `fix/bug-registro-LG` → Corrección de un error en el registro realizada por Luis Gómez.
- `docs/manual-usuario-AR` → Redacción del manual de usuario a cargo de Ana Rodríguez.
- `enhancement/refactor-database-MM` → Refactorización del esquema de la base de datos hecha por Marta Martínez.

2. Mensajes de commit

Para que los commits sean claros y comprensibles, deben seguir una estructura estándar:

2.1 Formato del mensaje

```
[TIPO]: [Descripción breve en presente]

[Explicación opcional en párrafo adicional]
```

2.2 Tipos de commits

Tipo de commit	Uso
`feat`	Nueva funcionalidad
`fix`	Corrección de errores
`docs`	Cambios en la documentación
`style`	Cambios de formato o estilo (espacios, comas, etc.) sin modificar lógica
`refactor`	Modificaciones en el código que no alteran su funcionalidad

2.3 Ejemplos de commits correctos

- `feat: Añadir validación en el formulario de login`
- `fix: Corregir error en la validación de email`
- `docs: Actualizar guía de instalación`
- `refactor: Optimizar función de carga de datos`

2.4 Buenas prácticas en commits

- Realizar commits pequeños y frecuentes.
- Cada commit debe contener cambios coherentes y relacionados.
- Usar el presente en los mensajes (Ejemplo: "Corrige error..." en lugar de "Corregido error...").
- Evitar mensajes genéricos como "arreglos", "actualización", "cambios".
- Comprobar que el código funciona antes de hacer un commit.
- Hacer `pull` antes de `push` para evitar conflictos.

3. Flujo de trabajo con ramas

1. Crear una nueva rama basada en `develop` siguiendo la nomenclatura establecida.
2. Desarrollar los cambios en la rama.
3. Hacer commits frecuentes con mensajes descriptivos.
4. Antes de hacer `push`, actualizar la rama con `develop` (`git pull origin develop`).
5. Subir la rama (`git push origin <nombre-rama>`).
6. Crear un **Pull Request (PR)** contra `develop` y asignar a un compañero para revisión.
7. Resolver posibles conflictos y realizar cambios según la retroalimentación.
8. Una vez aprobado, hacer `merge` en `develop`.