**Protocolo de buenas prácticas para el uso de Git y GitHub**

Este protocolo es una sugerencia para optimizar la organización y eficiencia de nuestro trabajo. Se aceptan aportes para la mejora continua del mismo.

**1. Configuración Inicial**

Configurar Git correctamente:

Configurar nombre y correo electrónico:

```
git config --global user.name "Nombre"
git config --global user.email "email@example.com"
```

**2. Estructura del repositorio**

Repositorio principal `main`:  mantener siempre estable y listo para producción. Evitar hacer commits directamente aquí.

- Rama de desarrollo (`development`):  usar para integrar características antes de fusionarlas con `main`.

- Ramas de característica (`feature`): crear una nueva rama para cada nueva funcionalidad o tarea siguiendo la estructura:

```
git checkout -b feature/nombre-descriptivo
```

Ramas de corrección rápida (`hotfix`):para corregir errores urgentes en producción:

```
git checkout -b hotfix/nombre-descriptivo
```

**3. Protección de la Rama `main`**

- Configurar protección de la rama `main`: asegurarse de que la rama `main` esté protegida mediante las siguientes configuraciones en GitHub:

1. Ir a repositorio en GitHub: navegar al repositorio en GitHub.
2. Abrir configuración: ir a la pestaña Settings (Configuraciones) del repositorio.
3. Acceder a Branches: en el menú de la izquierda, seleccionar Branches.

4. Agregar regla de protección:

- Hacer clic en Add rule en la sección Branch protection rules.

- En Branch name pattern, ingresar `main`.

- Activar las siguientes opciones:

- Require pull request reviews before merging: exigir al menos una revisión y aprobación antes de fusionar.

- Require status checks to pass before merging: asegurar que todas las pruebas automáticas se completen y pasen antes de permitir la fusión.

- Include administrators: aplicar estas reglas también a los administradores.

- Restrict who can push to matching branches: opcionalmente, limitar qué personas pueden hacer push directo a `main`.

- Hacer clic en Create o Save changes para guardar la regla.

**4. Commits**

Adoptar commits semánticos siguiendo la estructura propuesta por NC para mantener la claridad y el orden en el historial de cambios. Escribirlos en inglés y mantener una concordancia.

- Estructura Básica:

tipo(scope): descripción

- Ejemplo:

feat(auth): add user authentication using JWT

Tipos Comunes:

- feat: Nueva característica o funcionalidad (feature).

- fix: Corrección de un bug.

- docs: Cambios en la documentación.

- style: Cambios de formato o estilo (espacios, puntos y comas, etc.) que no afectan el código.

- refactor: Mejora en la estructura del código sin corregir errores ni añadir características.

- test: Añadir o corregir tests.

- Realizar commits diarios: hacer commits en un horario adecuado (antes de las 22:00 hs Arg.) para que el TL pueda realizar un seguimiento del trabajo.

5. **Flujo de Trabajo desde el repo local**

- Pasos básicos para trabajar de manera segura:

1. Cambiar a la rama de desarrollo:

   git checkout development

2. Traer los ultimos cambios del repositorio remoto:

   git fetch

3. Actualizar la rama local con los ultimos cambios:

   git pull

4. Cambiar a tu rama personal o de característica:

   git checkout nombre-rama

5. Fusionar los cambios de la rama de desarrollo a tu rama:

   git merge development

6. Realizar Commits y subir cambios al repo remoto:

   git add .

   git commit -m "mensaje del commit"

   git push

6. Pull Requests (PR)

- Revisar PRs antes de fusionar: **al menos un miembro del equipo debe revisar y aprobar el PR antes de fusionarlo. (un mimbro para back y un miembro para front)**

- Escribir mensajes de PR claros: incluir la descripción del cambio y su propósito.

- Referenciar issues: si un PR resuelve un issue, incluir “Closes #número_issue” en la descripción.

7. Resolución de conflictos

- Intentar resolver los conflictos de manera local antes de hacer un push. Pedir ayuda si es necesario para resolver conflictos complejos.

- Evitar `git push --force`: a menos que sea absolutamente necesario, ya que puede sobrescribir cambios de otros.

8. Documentación

- Mantener el README actualizado: asegurarse de que el README refleje siempre el estado actual del proyecto, incluyendo instrucciones para configurar el entorno, correr tests, etc.

- Usar Wiki : para documentar el uso del repositorio, guías de estilo, estándares de código, etc.

9. Seguridad

- Configurar protección de ramas adicionales: si es necesario, aplicar reglas de protección similares en otras ramas críticas del proyecto.

10. Comunicación

- Usar herramientas como Slack o Discord para recibir notificaciones automáticas de GitHub (PRs, issues, etc.)

- Mantener transparencia en los avances: comunicar claramente los cambios que se están realizando y los obstáculos encontrados.

11. Revisión y mejora continua

- Proporcionar retroalimentación periódica: revisar periódicamente estas prácticas y ajustarlas según las necesidades del equipo.

**Recomendaciones de NC:**

- Se hace la **main** que es la rama de producción con código estable y listo para desplegar.
- Rama **develop** que es donde se combinan las características nuevas que están listas para ser testeadas como un todo. Aquí se mergean las ramas de feat cuando están completadas.
- Se suelen hacer ramas por **feature** y lo mejor es que sean ramas de ciclo de vida corto. Se crea la rama, se desarrolla la feat, se mergea a dev y se cierra la branch.
- Después tenés las ramas de **fix** o **bugfix** que se hacen sobre develop y se mergean ahí para solucionar problemas en las features o actualizar las features.
- Después tenés las **hotfix** que se hacen sobre de main para corregir errores de producción.