



Cómo generar aportes

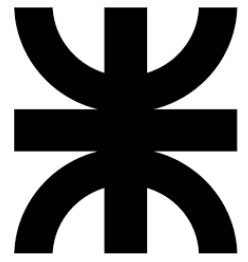
Proyecto Informática II

Autores: Gatica Marcos Raúl - José Prado

Curso: 2R4

Asignatura: Informática II.

Institución: Universidad Tecnológica Nacional - Facultad Regional de Córdoba.



U
T
N

F
R
C

1. Flujo de trabajo

En el siguiente documento se explica la metodología de trabajo para el proyecto "SISTEMA DE BLOQUEO DE ENCENDIDO VEHICULAR MEDIANTE ALCOHOLÍMETRO", con el fin de organizar y separar las diferentes áreas de desarrollo, facilitando que cada integrante o aportes voluntarios puedan subir sus modificaciones o generarlos. El motivo de esta metodología surge con el fin de evitar conflictos entre cambios no relacionados y manteniendo el historial de trabajo estructurado.

1.1. main

Rama principal del proyecto S.B.E.V.M.A. Aquí se integran los cambios aprobados y estables. Todo lo que se llega a *main* debe haber pasado por un proceso de revisión. Es importante mantener esta rama libre de errores, ya que representa la versión final y funcional del proyecto.

1.2. devel

Esta es la rama de desarrollo principal. Todo el código relacionado con el software del proyecto se trabaja aquí. El objetivo de esta rama es que cada integrante pueda hacer sus aportes y pruebas antes de integrarlo a *main*. Al trabajar en *devel*, nos permite implementar y probar nuevas funcionalidades sin afectar la estabilidad del proyecto.

1.3. hardw

Es la rama dedicada al desarrollo y documentación del hardware. Todos los cambios que afectan las especificaciones, la configuración o la implementación del hardware se gestionan en esta rama. Esto permite que el equipo de hardware pueda trabajar de forma independiente sin interferir con los desarrollos de software y documentación.

1.4. docs

Esta rama está destinada a la creación y mantenimiento de la documentación del proyecto. Los manuales, guías de instalación, y cualquier otro tipo de documentación necesaria se alojan aquí. Separar la documentación en su propia rama garantiza que los desarrolladores puedan trabajar en el código sin preocuparse por las modificaciones en los documentos, y viceversa.

2. Por qué de estas ramas

2.1. Organización

Al tener ramas dedicadas para cada parte del proyecto (software, hardware, y documentación), podemos mantener un orden claro sobre qué se está trabajando y dónde. Esto evita confusiones y reduce la probabilidad de mezclar cambios de áreas diferentes.

2.2. Flujo de trabajo eficiente

Al trabajar en ramas específicas, los colaboradores pueden implementar y probar sus cambios sin comprometer la estabilidad del proyecto en *main*. Solo cuando los cambios han sido revisados y validados, se hacen los *Pull Requests* (PRs) para integrarlos a *main*.

2.3. Control de versiones

Este esquema permite tener un registro claro de cada cambio hecho en el proyecto. Al crear PRs y hacer merges entre ramas, se mantiene un historial detallado de quién hizo cada contribución y cuándo. Esto es especialmente útil cuando hay que rastrear la fuente de un problema o realizar revisiones de código.

2.4. Colaboración y revisión

El uso de Pull Requests permite que cualquier cambio sea revisado por el equipo antes de ser integrado a la rama principal (*main*). Esto asegura que todos los aportes sean revisados y aprobados, fomentando un mejor control de calidad en el código, la documentación y el hardware.

3. Flujo típico de trabajo con estas ramas

3.1. Desarrollo separado

- Cada colaborador trabaja en su área específica en alguna de las ramas (*devel*, *hardw*, *docs*), según corresponda.
- Cada colaborador debe notificar el comienzo y fin de su trabajo.
- Se realizan los commits correspondientes para reflejar los avances.

3.2. Revisión y Pull Requests

- Una vez que los cambios están listos, se abre un PR desde la rama de trabajo (*devel*, *hardw*, *docs*), hacia la rama principal (*main*).
- El equipo principal de desarrollo (los autores de este documento) revisan los cambios propuestos, los comentarios y, si es necesario, los ajustes.

3.3. Integración con main

Una vez aprobado el PR, los cambios se fusionan a *main* asegurando que el proyecto se mantenga actualizado con los aportes más recientes.

4. Trabajo con GNU/Linux

Esta sección contiene la explicación sobre cómo trabajar con GitHub usando la terminal en sistemas operativos GNU/Linux.

4.1. Instalación de Git

Asegúrese de tener **git** instalado en su sistema. Opcionalmente puede instalar GitHub CLI en su sistema para hacer los PR desde su consola:

Sistemas basados en Debian:

```
apt update
apt full-upgrade
apt install git gh
```

Sistemas basados en Arch Linux:

```
pacman -Syu git github-cli
```

Una vez instalado en su sistema, ya puede clonar el repositorio remoto del proyecto. Recomendamos usar SSH con este link:

```
git@github.com:marcosgatica2003/
```

```
proyectoInformaticaII.git
```

Nota: va a necesitar una clave SSH en su computadora para poder clonar correctamente el repositorio y hacer sus aportes, puede generarla revisando esta documentación de GitHub:

```
https://docs.github.com/es/authentication/
connecting-to-github-with-ssh
/generating-a-new-ssh-key-and-adding-it-to-
the-ssh-agent
```

Una vez que los cambios se han subido al repositorio remoto, se puede generar un Pull Request para que el responsable de la revisión lo apruebe y combine con la rama principal. Para ello, es necesario utilizar la herramienta GitHub CLI, como se explicó al principio:

```
# Crear un PR desde la rama actual hacia main
gh pr create --base main --title
"Descripción del PR" --body
"Detalles del cambio"
```

A partir de aquí, sus aportes al proyecto ya han sido subidos y estarán a la espera de que un supervisor los revise.

4.2. Configuración

Una vez clonado el repositorio, necesita generar la configuración para poder subir correctamente sus aportes. Git utiliza el nombre de usuario y el correo electrónico configurados para asociar los commits con la identidad del desarrollador. Estos valores se pueden establecer de la siguiente manera:

```
# Establecer nombre de usuario
git config --global user.name "Tu Nombre"

# Establecer correo
git config --global user.email "Tu correo"

# Establecer un editor de texto (VIM)
git config --global user.editor vim

# Puede ver los cambios con:
git config --list
```

Después de ejecutar estos comandos, Git usará el nombre y el correo electrónico proporcionados para identificar las contribuciones en todos los repositorios locales.

4.3. Generar un PR

Antes de realizar cualquier cambio, es necesario cambiarse a la rama en la que se desea trabajar. El siguiente comando cambia a una rama existente, por ejemplo *devel*:

```
git checkout devel
```

Una vez en la rama correcta, puedes realizar cambios en los archivos del proyecto. Después de hacer las modificaciones, sigue estos pasos para agregar y confirmar los cambios:

```
# Agregar todos los cambios a preparación
git add .
```

A partir de aquí sus cambios están preparados para ser agregados a un commit, puede hacerlo con:

```
# Realizar un commit con un mensaje descriptivo
git commit -m "su mensaje breve"
```

```
# Abrir el editor VIM y escribir su mensaje
git commit
```

Para compartir los cambios con el resto del equipo, debes subir la rama de trabajo al repositorio remoto. Esto se hace con el siguiente comando (suponiendo que está trabajando en la rama *devel*):

```
git push origin devel
```