

Facultad de Ingeniería, Diseño y Ciencias

Aplicadas

Ingeniería de Sistemas

Proyecto integrador I

Profesores: Milton Sarria Paja, Claudia Castiblanco, Ingri Lorena Jojoa



Integrantes:

- **Juan José De La Pava - A00381213**
- **Rony Farid Ordoñez - A00397968**
- **David Artunduaga Penagos - A00396342**
- **Pablo Guzman Alarcon - A00399523**
- **Juan Parra Betancourt - A00398004**

Taller de Git y GitHub Parte 1

Objetivo del Taller

Información previa:

- **README.md:** Cada grupo debe actualizar el `README.md` al inicio del proyecto, incluyendo una descripción clara del mismo, instrucciones para configurarlo y cualquier otra información relevante. Esto debe realizarse cada vez que se complete una historia de usuario o una nueva funcionalidad.
- **CODESTYLE.md:** Antes de comenzar con la codificación, los equipos deben acordar un estándar de codificación y documentarlo en `CODESTYLE.md`. Todos los miembros del equipo deben seguir este estándar al escribir código, y cualquier modificación en las reglas debe ser consensuada y actualizada en este archivo.

1. Configuración Inicial y Creación de Repositorio (10%)

Actividades:

- Todos los usuarios deben haber creado
- Configurar el nombre de usuario y correo electrónico en Git utilizando:

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tuemail@ejemplo.com"
```

FOTOS CONFIGURACIÓN:

```
..royecto-gecko-team/src git +  
user.name=JuanJ0lp  
user.email=juanosedesdelapava1@gmail.com  
credential.https://github.com.helper=  
credential.https://github.com.helper=!/usr/bin/gh auth git-credential  
pull.rebase=false  
(END)
```

```
PowerShell 7.4.5  
Loading personal and system profiles took 2659ms.  
➤ git config --global --list  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
user.name=David Artunduaga  
user.email=1040871155@u.icesi.edu.co  
difftool.sourcetree.cmd=''  
mergetool.sourcetree.cmd=''  
mergetool.sourcetree.trustexitcode=true  
  
pwsh MEM: 70% | 8/11GB 241ms  
15:37 | ➤ Taller GIT
```

```
PS C:\Users\ASUS> git config --global --list  
user.email=e5jparra@gmail.com  
user.name=Juanpapb0401  
safe.directory=D:/Semestre II/Semestre II/APO 1/Projects  
difftool.sourcetree.cmd=''  
mergetool.sourcetree.cmd=''  
mergetool.sourcetree.trustexitcode=true  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
PS C:\Users\ASUS>
```

- PS D:\202402-proyecto-gecko-team> `git config --global --list`
user.email=123975198+Pableis05@users.noreply.github.com
user.name=Pablo Andres Guzman Alarcon
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true

```
➤ git config  
core.editor="C:\Users\Usuario\AppData\Local\Programs\Microsoft VS Code\bin\code" --wait  
user.name=Rony Ordoñez  
user.email=ronyfarido@gmail.com
```

FOTO CLONACIÓN:

```
ASUS@DESKTOP-NTHR NAC MINGW64 /d  
$ git clone https://github.com/Rony7v7/Taller-GIT.git  
Cloning into 'Taller-GIT'...  
remote: Enumerating objects: 3, done.  
remote: Counting objects: 100% (3/3), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)  
Receiving objects: 100% (3/3), done.
```

```
ASUS@PCJPP MINGW64 ~/Desktop/Quinto Semestre/Proyecto Integrador/TallerGit  
$ git clone https://github.com/Rony7v7/Taller-GIT.git  
Cloning into 'Taller-GIT'...  
remote: Enumerating objects: 3, done.  
remote: Counting objects: 100% (3/3), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)  
Receiving objects: 100% (3/3), done.
```

```
➤ git clone https://github.com/Rony7v7/Taller-GIT.git  
Cloning into 'Taller-GIT'...  
remote: Enumerating objects: 3, done. 3 minutes  
remote: Counting objects: 100% (3/3), done. 3 minutes  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)  
Receiving objects: 100% (3/3), done.  
  
Windows Terminal pwsh MEM: 71% | 8/11GB 3s 232ms  
15:39 Taller GIT
```

```
➤ git clone https://github.com/Rony7v7/Taller-GIT.git  
Cloning into 'Taller-GIT'...  
fatal: unable to access 'https://github.com/Rony7v7/Taller-GIT.git/': Could not resolve host: github.com  
➤ git clone https://github.com/Rony7v7/Taller-GIT.git  
Cloning into 'Taller-GIT'...  
remote: Enumerating objects: 8, done.  
remote: Counting objects: 100% (8/8), done.  
remote: Compressing objects: 100% (5/5), done.  
remote: Total 8 (delta 1), reused 8 (delta 1), pack-reused 0 (from 0)  
Receiving objects: 100% (8/8), done.  
Resolving deltas: 100% (1/1), done.  
➤
```

R/ Ya se tiene el git configurado, aqui esta una captura

The screenshot shows a GitHub repository page for 'Taller-GIT'. The repository is private. It contains one branch ('master') and one commit ('JuanJDip First commit'). The commit was made 5 hours ago and includes three files: 'codestyle.md', 'main.py', and 'readme.md', all of which were committed first. There is also a 'README' file. The repository has 1 star, 0 forks, and 1 person watching it. It has no releases or packages published.

Code

About

No description, website, or topics provided.

Activity

0 stars

1 watching

0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

✓ **Entrega:** Capturas de pantalla mostrando la configuración y clonación del repositorio.

✓ **Pregunta para reflexión:**

¿Cuál es la diferencia entre clonar un repositorio y hacer un fork? ¿En qué situaciones utilizarías cada uno?

R/ Clonar un repositorio crea una copia local vinculada al original, permitiendo sincronizar y hacer cambios directamente si tienes permisos. Hacer un fork copia el repositorio a tu cuenta en GitHub, sin afectar el original, y se usa principalmente cuando no tienes acceso directo para modificar el repositorio principal o quieres trabajar en tus propios cambios antes de proponerlos.

2. Colaboración en Equipo usando Ramas (20%)

✓ **Actividades:**

- Crear ramas específicas para cada funcionalidad o tarea asignada (Se encuentran al final de este documento) usando:

```
git checkout
```

```
Ejemplo: git checkout -b feat/crear-usuario
```

```
ASUS@DESKTOP-NTHR NAC MINGW64 /d/Taller-GIT (dev)
$ git checkout -b feat/ValidacionCombustible
Switched to a new branch 'feat/ValidacionCombustible'
```

```
ASUS@DESKTOP-NTHR NAC MINGW64 /d/Taller-GIT (feat/ValidacionCombustible)
```

```
ASUS@DESKTOP-NTHR NAC MINGW64 /d/Taller-GIT (dev)
$ git checkout -b feat/CreacionPotencia
Switched to a new branch 'feat/CreacionPotencia'
```

```
ASUS@DESKTOP-NTHR NAC MINGW64 /d/Taller-GIT (feat/CreacionPotencia)
$ git checkout feat/ValidacionCombustible
```

```
~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(dev) (0.169s)
git checkout feat/Validacion_Inicio_de_sesion
Switched to branch 'feat/Validacion_Inicio_de_sesion'
```

321

```
> git checkout -b feat/perfil-usuario
Switched to a new branch 'feat/perfil-usuario'
¿Por qué es importante seguir una convención para nombrar las ramas? ¿Qué
beneficios tiene en un equipo grande?
Windows Terminal pwsh MEM: 64% | 7/11GB 115ms
15:51 | Segu a n n Taller-GIT Las ramas es importante porque facilita
organización y comprensión del trabajo dentro de un equipo, especialmente
```

```
ASUS@PCJPP MINGW64 ~/Desktop/Quinto Semestre/Proyecto Integrador/TallerGit/Talle
r-GIT (feat/imprimir-vehiculo)
$ git checkout -b feat/imprimir-vehiculos-nuevos-atributos
Switched to a new branch 'feat/imprimir-vehiculos-nuevos-atributos'
```

```

ASUS@PCJPP MINGW64 ~/Desktop/Quinto Semestre/Proyecto Integrador/TallerGit/Talle
r-GIT (feat/imprimir-vehiculos-nuevos-atributos)
$ git checkout -b feat/imprimir-vehiculos
Switched to a new branch 'feat/imprimir-vehiculos'

→ git checkout -b feat/clase-vehiculo
• Switched to a new branch 'feat/clase-vehiculo'
  → git checkout -b feat/filtro-años
• Switched to a new branch 'feat/filtro-años'
○ → Taller-GIT ↵ (feat/filtro-años)

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(dev) (0.156s)
git checkout feat/filtro_mayor_menor
Switched to branch 'feat/filtro_mayor_menor'.
Your branch is up to date with 'origin/feat/filtro_mayor_menor'.

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(feat/filtro_mayor_menor)

```

✓ **Entrega:** Capturas de pantalla de las ramas creadas.

✓ **Tips de comandos:**

- Para listar todas las ramas locales y remotas:
git branch -a
- Para cambiar de rama:
git checkout <nombre-de-la-rama>

✓ **Pregunta para reflexión:**

¿Por qué es importante seguir una convención para nombrar las ramas? ¿Qué beneficios tiene en un equipo grande?

R/ Seguir una convención para nombrar las ramas es importante porque facilita la organización y comprensión del trabajo dentro de un equipo, especialmente en proyectos grandes. Los nombres consistentes y descriptivos permiten identificar fácilmente el propósito de cada rama (como una característica, corrección de errores o mejora), lo que mejora la comunicación, evita confusiones y reduce errores. Además, una convención clara ayuda a gestionar mejor las integraciones y revisiones de código, haciendo que el proceso de desarrollo sea más eficiente y colaborativo.

3. Gestión de Commits y Estándares de Codificación (20%)

✓ **Objetivo:** Realizar commits significativos y coherentes, siguiendo un estándar acordado por el equipo.

✓ **Actividades:**

- Hacer commits con mensajes descriptivos que reflejen los cambios realizados. Ejemplo:

```
git commit -m "feat: añadir validación al formulario de registro"
```

- Documentar el estilo de codificación utilizado en el proyecto en un archivo CODESTYLE.md

Paso 1: Definir el Estándar de Codificación: Todos los usuarios (A, B, C, D, y E) deben reunirse para discutir y acordar un estándar de codificación que todos seguirán en el proyecto (**CODESTYLE.md**). Decidan aspectos como:

- Nombres de variables y funciones (camelCase, PascalCase, etc.).
- Reglas de indentación (espacios vs. tabuladores).
- Longitud máxima de líneas de código.
- Formato de comentarios y documentación del código.

Crear el Archivo **CODESTYLE.md**:

- Un usuario (por ejemplo, el usuario A) crea el archivo **CODESTYLE.md** en la rama `main` o `develop`.
- Este archivo debe incluir todas las reglas acordadas en la reunión.
- **Commit:** El usuario A realiza un commit con un mensaje claro, por ejemplo:
 - `git add CODESTYLE.md`
 - `git commit -m "docs: Added coding style guidelines to CODESTYLE.md"`
 - `git push origin main`

```
→ git add .\code\codestyle.md
●→ git commit -m "docs: Added coding style guidelines to CODESTYLE.md"
● [master 2bb93ba] docs: Added coding style guidelines to CODESTYLE.md
  1 file changed, 63 insertions(+)
  create mode 100644 code/codestyle.md
→ git push
```

Paso 2: Realizar Cambios y Gestionar Commits: Asignar a cada usuario una tarea que implique hacer cambios en el código. Cada tarea debe realizarse en una rama separada. **Ejemplo de tareas:**

- **Usuario A:** Implementar la funcionalidad de registro de usuarios.
- **Usuario B:** Añadir validación al formulario de inicio de sesión.
- **Usuario C:** Crear la página de perfil del usuario.
- **Usuario D:** Mejorar el sistema de autenticación.
- **Usuario E:** Refactorizar el código del controlador de usuarios.

Crear una Rama para Cada Tarea:

- Cada usuario crea una nueva rama desde `main` o `develop` para trabajar en su tarea asignada.

Ejemplo:

```
git checkout -b feat/registro-usuario
```

R/

Se usará el commit standard de :

<https://www.conventionalcommits.org/en/v1.0.0/>

Paso 3: Realizar Cambios y Hacer Commits:

- Cada usuario realiza los cambios necesarios en su rama. Siguen el estándar de codificación definido en el archivo **CODESTYLE.md**.
- Los usuarios deben hacer commits pequeños y frecuentes, con mensajes

descriptivos que expliquen los cambios realizados.

- **Ejemplo de commits:**

- git add registro.js
- git commit -m "feat: Implement user registration functionality"
- git add auth.js
- git commit -m "fix: Correct validation error in login form"

- ✓ **Entrega:** Capturas de pantalla de los commits realizados y el contenido del archivo CODESTYLE.md.

```
> git add --all
> git commit -m "feat: Implementada las operaciones con vehiculos en main"
```

```
~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(dev) (0.011s)
git pull
From https://github.com/Rony7v7/Taller-GIT
 5b9cfbc..66e826b dev      -> origin/dev
Updating 5b9cfbc..66e826b
Fast-forward
 code/main.py | 39 ++++++++-----+-----+-----+-----+
 1 file changed, 39 insertions(+), 30 deletions(-)

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(dev) (2.168s)
git branch

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(dev) (0.154s)
git checkout feat/filtro_mayor_menor
Switched to branch 'feat/filtro_mayor_menor'

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(feat/filtro_mayor_menor) (1.418s)
git merge dev
Merge made by the 'ort' strategy.
 code/Vehicle.py | 49 +-----+-----+-----+-----+
 code/main.py    | 39 +-----+-----+-----+-----+
 2 files changed, 88 insertions(+), 0 deletions(-)
 create mode 100644 code/Vehicle.py

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(feat/filtro_mayor_menor) (0.116s)
git add .

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(feat/filtro_mayor_menor) $ (0.201s)
git commit -m "feature: modified filtro de mayor o menor de vehicles"
[feat/filtro_mayor_menor d0a@029] feature: modified filtro de mayor o menor de vehicles
 1 file changed, 15 insertions(+), 7 deletions(-)

~/Desktop/Universidad/IntegradorSis/Taller_de_git git:(feat/filtro_mayor_menor)
```

```
ASUS@DESKTOP-NTHRNAC MINGW64 /d/Taller-GIT (dev|MERGING)
$ git add --all

ASUS@DESKTOP-NTHRNAC MINGW64 /d/Taller-GIT (dev|MERGING)
$ git commit -m "feat: agregar correctamente combustible"
[dev e538490] feat: agregar correctamente combustible

ASUS@DESKTOP-NTHRNAC MINGW64 /d/Taller-GIT (dev)
$ git push
Enumerating objects: 28, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 16 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 1.89 KiB | 277.00 KiB/s, done.
Total 12 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 4 local objects.
To https://github.com/Rony7v7/Taller-GIT.git
 9710f81..e538490 dev -> dev
```

```
ASUS@DESKTOP-NTHRNAC MINGW64 /d/Taller-GIT (dev|MERGING)
$ git commit -m "feat: agregar correctamente combustible"
[dev f3f9ad0] feat: agregar correctamente combustible
```

```
ASUS@DESKTOP-NTHRNAC MINGW64 /d/Taller-GIT (dev)
$ git commit -m "docs: Creacion de formato de commits"
[dev 871def5] docs: Creacion de formato de commits
 1 file changed, 12 insertions(+)
```

```
ASUS@DESKTOP-NTHRNC MINGW64 /d/Taller-GIT (feat/CreacionPotencia)
$ git commit -m "develop: Crear potencia.py"
[feat/CreacionPotencia 6244e50] develop: Crear potencia.py
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 code/potencia.py
```

```
→ git add .
→ git commit -m "feat: Filtro años por rango"
[feat/filtro-años 75c0440] feat: Filtro años por rango
 1 file changed, 13 insertions(+), 3 deletions(-)
```

```
● PS C:\Users\ASUS\Desktop\Quinto Semestre\Proyecto Integrador\TallerGit\Taller-GIT> git commit -m "feat: Imprimir vehiculos"
[feat/imprimir-vehiculos a6c057e] feat: Imprimir vehiculos
 1 file changed, 17 insertions(+), 3 deletions(-)
```

```
● PS C:\Users\ASUS\Desktop\Quinto Semestre\Proyecto Integrador\TallerGit\Taller-GIT> git commit -m "feat:Nuevo print vehiculos"
[feat/imprimir-vehiculos-nuevos-atributos d157f34] feat:Nuevo print vehiculos
 2 files changed, 8 insertions(+), 40 deletions(-)
```

```
> git add --all
> git commit -m "feat: Implementada las operaciones con vehiculos en main"
```

```
└─ 17:47 | ↗ → ■ → ■ → ■ → Taller-GIT
  git commit -m "Refactor: actualización de clase Vehicle para soportar nuevos atributos"
```

✓ Tips de comandos:

- Para añadir cambios al área de preparación (staging area):
- `git add <nombre-del-archivo>`
- Para ver el historial de commits:
- `git log --oneline`

✓ Pregunta para reflexión:

¿Qué diferencia hay entre un commit estándar y uno amend? ¿Cuándo usarías cada uno?

Un commit estándar guarda los cambios en el historial del repositorio como un nuevo registro, mientras que un commit con `--amend` modifica el último commit realizado, permitiendo ajustar el mensaje o agregar cambios adicionales sin crear un nuevo commit. Usarías un commit estándar para registrar un nuevo conjunto de cambios en el proyecto, y `--amend` cuando quieras corregir o complementar el commit anterior, como en caso de errores tipográficos en el mensaje o cuando olvidaste incluir algunos archivos en el commit previo.

4. Merge y Resolución de Conflictos (30%)

✓ **Objetivo:** Realizar merges efectivos y resolver conflictos que puedan surgir al integrar cambios.

✓ Actividades:

- Hacer merge de las ramas en la rama `develop` o `main`, resolviendo conflictos si es necesario:
- `git merge <nombre-de-la-rama>`
- Utilizar un editor de texto o IDE como Visual Studio Code para resolver los conflictos manualmente.

Revisión del Código:

- Antes de realizar el merge, cada usuario revisa su código para asegurarse de que cumple con el estándar de codificación.
- Si es necesario, ajustan el código para alinearse con las pautas en `CODESTYLE.md`.

Realizar el Merge y Resolver Conflictos: Sincronización con `main` o `develop`:

- Antes de hacer un pull request (PR), cada usuario debe sincronizar su rama con `main` o `develop` para asegurarse de que están trabajando con la versión más reciente del código. Esto se hace con:

```
git fetch origin
git merge origin/main
```

Resolver Conflictos (si los hay): Si surgen conflictos durante el merge, los usuarios deben resolverlos de acuerdo con el estándar de codificación y hacer un commit de la resolución de conflictos.

Crear un Pull Request (PR): Cada usuario crea un pull request desde su rama hacia `main` o `develop`.

- El PR debe incluir una descripción clara de los cambios realizados y una referencia al estándar de codificación utilizado.
- Solicitan una revisión por parte de otro miembro del equipo.

Revisión por Pares y Aprobación:

- Otro usuario revisa el PR para asegurarse de que el código sigue el estándar de codificación y cumple con los requisitos funcionales.
- Si todo está en orden, aprueban el PR y realizan el merge.

Tips de Comandos a Utilizar:

- **Para cambiar de rama:**
git checkout <nombre-de-la-rama>
- **Para listar commits recientes:**
git log --oneline
- **Para ver las diferencias antes de un commit:**
git diff
- **Para visualizar y resolver conflictos:**
git status
- **Para fusionar una rama con main o develop:**
git merge <nombre-de-la-rama>

✓ **Entrega:** Capturas de pantalla del proceso de merge y la resolución de conflictos, junto con una explicación de los pasos seguidos.

✓ **Tips de comandos:**

- Para ver los archivos en conflicto después de un merge:
 - git status
- Para abortar un merge en caso de errores:
 - git merge --abort

✓ **Pregunta para reflexión:**

¿Qué estrategias de resolución de conflictos podrías aplicar en un proyecto con múltiples colaboradores?

En proyectos con múltiples colaboradores, la resolución de conflictos se facilita mediante estrategias como la comunicación clara sobre las áreas de trabajo, el uso de ramas independientes para cada tarea, y la fusión o rebase frecuente para mantener las ramas actualizadas. Las revisiones de código también ayudan a detectar problemas antes de integrar cambios. En caso de conflictos, los colaboradores pueden resolverlos manualmente revisando los archivos afectados, y las pruebas automatizadas aseguran que los cambios no rompan funcionalidades existentes, mejorando la estabilidad del proyecto.

5. Trabajo Final y Documentación del Proyecto (20%)

✓ **Objetivo:** Consolidar el trabajo realizado y documentar el proceso del proyecto.

✓ **Actividades:**

- Completar el proyecto y preparar un informe en formato markdown (`informe.md`) que contenga: Nombres de los participantes, Evidencias de los commits, merges y conflictos resueltos, Descripción de los estándares de codificación, Capturas de pantalla del proceso completo.

Documentar el Proceso en `informe.md`:

- Cada usuario documenta en un archivo `informe.md` cómo realizó su tarea, cómo gestionó los commits, y cómo resolvió cualquier conflicto.
- El archivo debe incluir capturas de pantalla de los commits realizados, los conflictos resueltos, y la revisión del código.

✓ **Entrega:** Subir el informe al repositorio en GitHub

✓ **Pregunta para reflexión:**

¿Cómo puedes asegurarte de que la documentación de tu proyecto sea útil para futuros colaboradores?

Para asegurar que la documentación de un proyecto sea útil para futuros colaboradores, es importante que sea clara, organizada y actualizada. Debe incluir una guía de inicio para que los nuevos miembros puedan configurar el entorno de desarrollo rápidamente, junto con explicaciones detalladas de las principales funcionalidades, arquitectura y flujo del proyecto. También es útil proporcionar ejemplos prácticos y mantener un registro de decisiones técnicas importantes. Además, una buena práctica es actualizar la documentación cada vez que se implementen cambios significativos, e incentivar a los colaboradores a contribuir con mejoras o aclaraciones.

Recomendaciones para el Taller

- **Trabajo en Equipo:** Trabajar en grupos personas para mejorar la colaboración y la gestión de proyectos.
- **Revisión por Pares:** Configurar reglas de branches para requerir revisiones antes de hacer merge a `develop` o `main`.

- **Comunicación:** En el contexto no es necesario usar herramientas porque trabajarán en clase y podrán comunicarse directamente con sus compañeros. En el desarrollo de otros proyectos pueden usar herramientas que les permitan chatear o hacer videollamadas (Teams, Slack, Discord, etc.) para mantener una comunicación constante.
-

Contexto y Tareas Por Realizar

Cada miembro del equipo trabajará en una parte de un sistema de gestión de vehículos, desarrollando clases que representen distintos aspectos como información técnica, historial de mantenimiento y gestión. A su vez, se incluirá un archivo `README.md` para documentar el uso del sistema.

Deben crear una carpeta `code` donde estarán todos los archivos que se especificarán durante la realización de los pasos y el archivo de `CODINGSTYLE.md`. Por fuera de esta carpeta debe estar el archivo `README.md` y el archivo `informe.md`.

Trabajen en la rama `develop` para crear la estructura básica del proyecto. Es importante que todos terminen la primera parte de las tareas antes de seguir con la segunda. Por cada tarea realizada actualicen el `README.md` con la estructura y funcionalidades nuevas del proyecto.

Nota: No eliminan las ramas creadas una vez finalicen su propósito.

1. Creación de estructura inicial

Usuario A:

- Crea una clase "Vehiculo" que debe incluir los atributos "marca", "modelo", "año", "kilometraje", "estado_actual" y "tipo_combustible", junto con sus respectivos métodos getter y setter.

Usuario B:

- Crea una clase "HistorialMantenimiento" que almacene información sobre las reparaciones y mantenimientos realizados a un vehículo. Debe incluir los atributos "fecha", "descripcion_servicio", "kilometraje_en_servicio", "costo", y "nombre_mecanico", junto con sus respectivos métodos getter y setter.

Usuario C:

- Implementa la clase "Main" que posea una lista de vehículos y permita agregarlos y buscarlos por año.

Usuario D:

- Implementa validaciones adicionales en la clase "Vehiculo", asegurando que el tipo de combustible solo pueda ser de una lista predefinida (ej. "Gasolina", "Diesel", "Eléctrico").

Usuario E:

- Implementa un método en la clase "Main" que permita imprimir todos los vehículos de la flota con las características de cada uno.

2. Nuevas Funcionalidades

Usuario A:

- Modifica la clase "Main", para que ahora el filtro por año permita buscar vehículos que se encuentren en un rango de años. Actualiza el `README.md` con ejemplos de cómo calcular la antigüedad de un vehículo.

Usuario B:

- Modifica la clase "Main", para que ahora el filtro por año reciba un parámetro que especifique si es mayor o menor, y que, dependiendo de este, liste los vehículos que son mayores o menores al año especificado. Actualiza el `README.md` con instrucciones sobre cómo utilizar esta funcionalidad.

Usuario C:

- Modifica la clase “Vehiculo”, para agregar un nuevo atributo “color”. Agrega los getter y setter pertinentes y actualiza el `README.md` con el nuevo atributo.

Usuario D:

- Modifica la clase “Vehiculo”, para agregar un nuevo atributo “potencia” (número que indique caballos de fuerza). Agrega los getter y setter pertinentes y actualiza el README.md con el nuevo atributo.

Usuario E:

- Modifica el método de impresión de datos de todos los vehículos para agregar los nuevos atributos definidos por el usuario D y .

Cuando el equipo termine las modificaciones y la rama `develop` esté actualizada con los últimos cambios, uno de los integrantes deberá hacer un pull request a `main` y integrantes diferentes deberán aceptarlo, de forma que la rama `main` quede actualizada también.

En caso de presentar conflictos, documéntenlos en el archivo `informe.md` con el lugar del conflicto y cómo se solucionaron.

The screenshot shows a GitHub pull request interface with two code files being reviewed:

- main.py**:
 - Line 84: `class Main:`
 - Line 85: `def __init__(self):`
 - Line 86: `self.vehicle_list = []`
 - Line 72: `def add_vehicle(self, vehicle):`
 - Line 73: `"""`
 - Line 74: `Agrega un vehículo a la lista.`
 - Line 75: `:param vehicle: Objeto de tipo Vehicle.`
 - Line 76: `:type vehicle: Vehicle`
 - Line 77: `"""`
 - Line 78: `self.vehicle_list.append(vehicle)`

Annotations for `main.py`:

 - Line 74: `Agrega un vehículo a la lista.` (Red underline, expected expression)
 - Line 75: `:param vehicle: Objeto de tipo Vehicle.` (Red underline, unexpected indentation)
 - Line 76: `:type vehicle: Vehicle` (Red underline, unexpected indentation)
 - Line 78: `self.vehicle_list.append(vehicle)` (Red underline, unexpected indentation)

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

HistorialMantenimiento.py:

Annotations for `HistorialMantenimiento.py`:

 - Line 63: `def find_vehicles_by_year(self, year, year_range):` (Red underline, expected expression)
 - Line 64: `"""`
 - Line 65: `Expected indented block`
 - Line 66: `def find_vehicles_by_year(self, year, mayor_o_menor):` (Red underline, unexpected indentation)
 - Line 67: `"""`
 - Line 68: `feat/filter/mayor menor` (Red underline, expected indented block)
 - Line 69: `"""`
 - Line 70: `Unindent not expected`
 - Line 71: `Busca vehículos por su año de fabricación.`
 - Line 72: `:param year: Año de fabricación del vehículo.`
 - Line 73: `:type year: int`
 - Line 74: `:param mayor_o_menor: Especifica si el filtro es para vehículos mayores o menores al año especificado.`
 - Line 75: `:type mayor_o_menor: str`
 - Line 76: `return: Lista de vehículos que cumplen con el criterio de año.`
 - Line 77: `:rtype: list`

Annotations for `HistorialMantenimiento.py`:

 - Line 63: `def find_vehicles_by_year(self, year, year_range):` (Red underline, expected expression)
 - Line 64: `"""`
 - Line 65: `if mayor_o_menor == 'mayor':` (Red underline, unexpected indentation)
 - Line 66: `vehicles_by_year = [vehicle for vehicle in self.vehicle_list if vehicle.year > year]` (Red underline, "self" is not defined)
 - Line 67: `elif mayor_o_menor == 'menor':` (Red underline, "mayor_o_menor" is not defined)
 - Line 68: `vehicles_by_year = [vehicle for vehicle in self.vehicle_list if vehicle.year < year]` (Red underline, "self" is not defined)
 - Line 69: `else:`
 - Line 70: `raise ValueError("El parámetro 'mayor_o_menor' debe ser 'mayor' o 'menor'.")`
 - Line 71: `return vehicles_by_year`
 - Line 72: `"""`

Annotations for `HistorialMantenimiento.py`:

 - Line 63: `def find_vehicles_by_year(self, year, year_range):` (Red underline, unindent not expected)

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

OUTLINE | **TIMELINE**