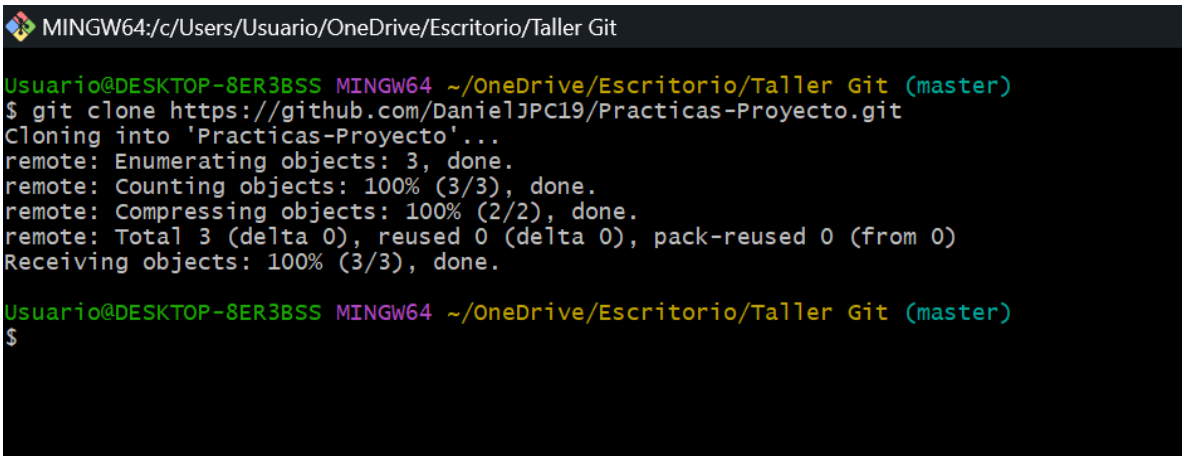
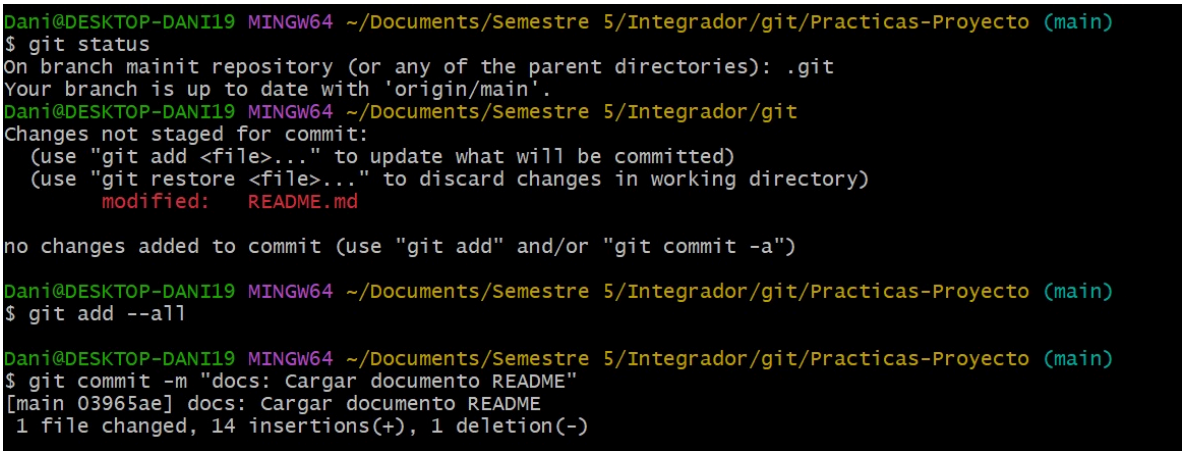
Taller Git

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

Capturas de pantalla mostrando la configuración y clonación del repositorio.





Pregunta de reflexión:

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

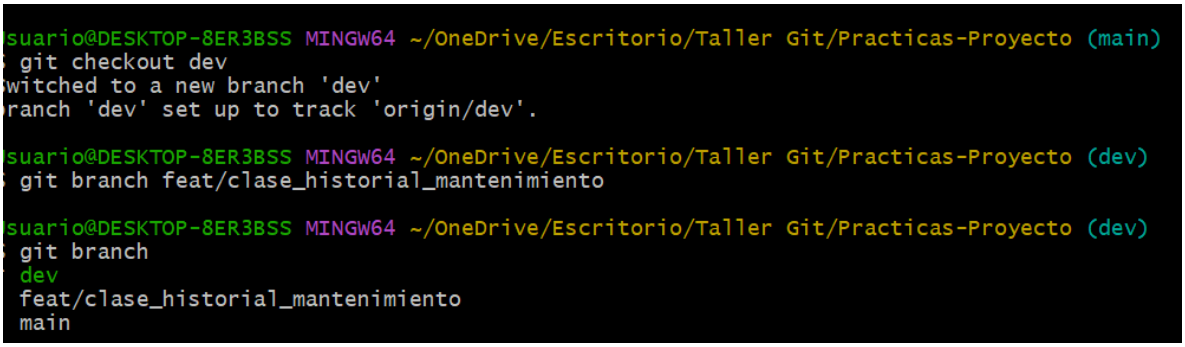
R: Clonar repositorio: Clonar un repositorio crea una copia exacta del repositorio en tu máquina local. Se hace cuando se desea trabajar localmente con el código de un repositorio y contribuir directamente a él (si tienes permisos).

Hacer fork: Hacer un fork implica crear una copia completa del repositorio original en nuestro propio espacio de GitHub/GitLab (o la plataforma que utilicemos). Se usa cuando se quieren hacer cambios al proyecto pero no tenemos permisos para hacer commits directamente en el repositorio original.

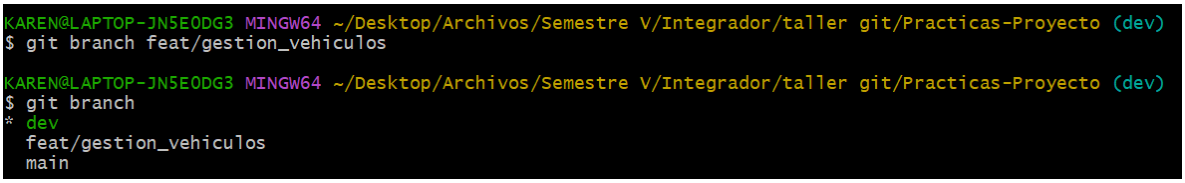
2. Colaboración en Equipo usando Ramas (20%)

- **Crear ramas específicas para cada funcionalidad o tarea asignada:**

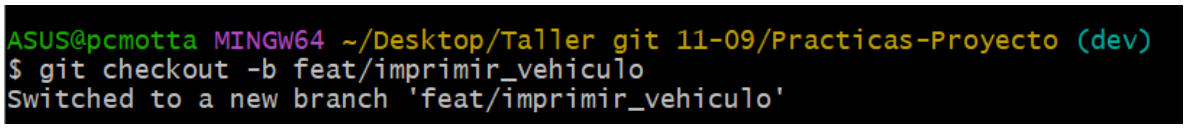
  USUARIO B:



  USUARIO C:



  USUARIO E:



- **Modificar el archivo README.md y añadir la información relacionada con la historia de usuario:**

  USUARIO B:

```
Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git status
On branch feat/filtrar_mayor_menor
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git add --all

Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git status
On branch feat/filtrar_mayor_menor
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md


Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git commit "feat: instrucciones sobre cómo utilizar esta funcionalidad."
error: pathspec 'feat: instrucciones sobre cómo utilizar esta funcionalidad.' did not match any file(s) known to git

Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git commit -m "feat: instrucciones sobre cómo utilizar esta funcionalidad."
[feat/filtrar_mayor_menor 30633ff] feat: instrucciones sobre cómo utilizar esta funcionalidad.
 1 file changed, 32 insertions(+)

Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git status
On branch feat/filtrar_mayor_menor
nothing to commit, working tree clean

Usuario@DESKTOP-8ER3BSS MINGW64 ~/OneDrive/Escritorio/Taller Git/Practicas-Proyecto (feat/filtrar_mayor_menor)
$ git push origin feat/filtrar_mayor_menor
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.06 KiB | 1.06 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
```

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 de nombres para las ramas en un equipo grande es importante porque:

> **Facilita la colaboración:** Todos entienden el propósito de cada rama de forma clara y rápida.
>
> **Organiza el trabajo:** Estructura las ramas según su tipo (funcionalidades, correcciones, versiones), mejorando la gestión del proyecto.
>
> **Evita errores y conflictos:** Nombres consistentes reducen la confusión y el riesgo de trabajar en ramas equivocadas.

En un equipo grande, los beneficios incluyen mejor organización, mayor eficiencia, diversidad de habilidades, reducción de la carga individual y fomento de la colaboración y creatividad.

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

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

  USUARIO B:

```
$ git commit -m "feat: creación clase mantenimiento"
[feat/clase_historial_mantenimiento fdb2127] feat: creación clase mantenimiento
 1 file changed, 58 insertions(+)
 create mode 100644 CarManagement/src/HistorialMantenimiento.java
```

USUARIO C:

```
KAREN@LAPTOP-JN5EODG3 MINGW64 ~/Desktop/Archivos/Semestre V/Integrador/taller git/Practicas-Proyecto (feat/gestion_vehiculos)
$ git commit -m "feat: actualizacion clase main"
[feat/gestion_vehiculos 391840a] feat: actualizacion clase main
 1 file changed, 39 insertions(+)
 create mode 100644 CarManagement/src/Main.java
```

USUARIO E:

```
ASUS@pcmotta MINGW64 ~/Desktop/Taller git 11-09/Practicas-Proyecto (feat/imprimi
r_vehiculo)
$ git commit -m "feat: creación del metodo para imprimir vehiculos"
[feat/imprimir_vehiculo 69de6fa] feat: creación del metodo para imprimir vehicul
os
 1 file changed, 6 insertions(+)
```

Pregunta para reflexión:

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

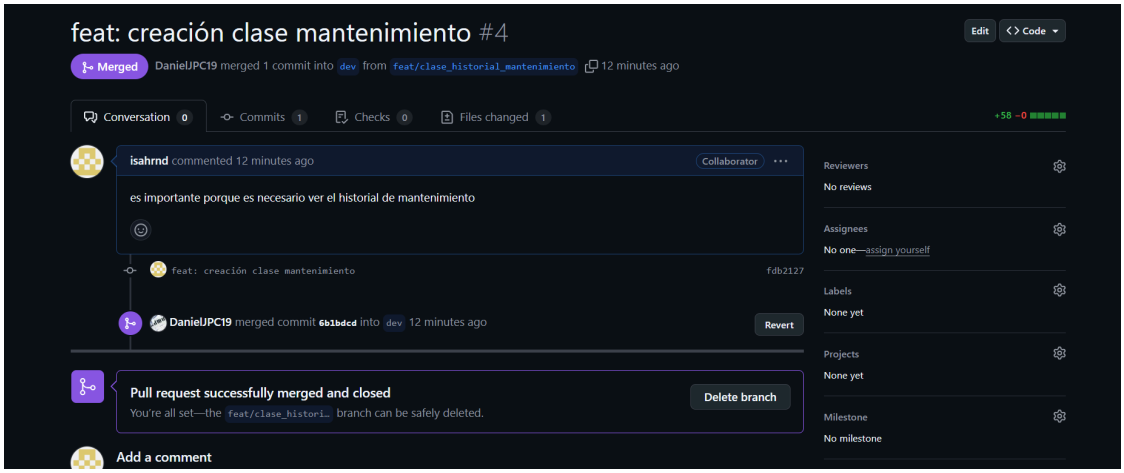
R: Commit Estándar: Registra nuevos cambios en el historial del repositorio. Usado para cada conjunto de cambios significativos.

Commit --amend: Modifica el último commit, permitiendo ajustar su contenido o mensaje. Usado para correcciones rápidas antes de compartir el commit.

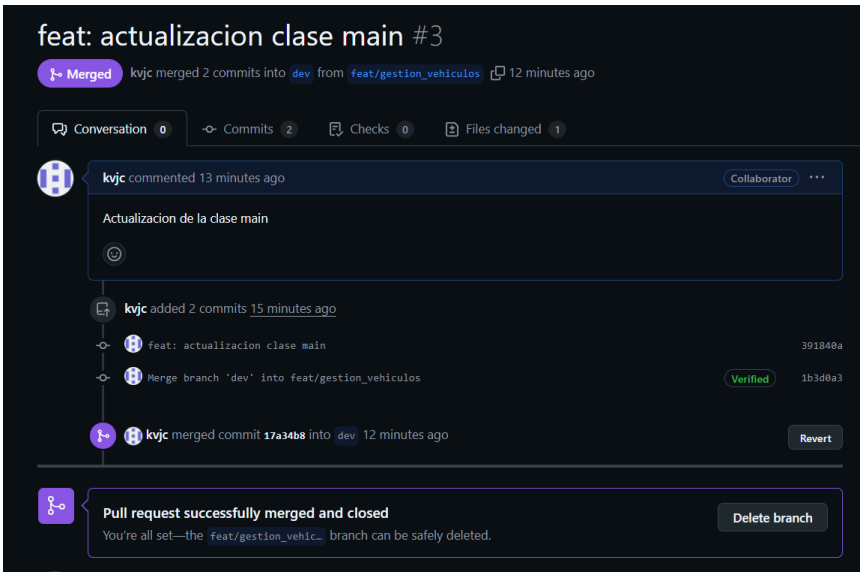
En resumen, se usa el commit estándar para cambios nuevos y --amend para ajustar el último commit antes de compartirlo.

4. Merge y Resolución de Conflictos (30%)

USUARIO B:



USUARIO C:



USUARIO E:

Pregunta para reflexión:

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

R: Para resolver conflictos en proyectos con múltiples colaboradores se pueden aplicar estas estrategias:

Revisar conflictos manualmente cuando surjan durante un merge o rebase.

Comunicación con el equipo para evitar conflictos en cambios importantes.

Hacer pull frecuentemente para mantenerse al día con los cambios.

Usar ramas temáticas para aislar cambios y minimizar conflictos.

Hacer commits pequeños y claros para facilitar el rastreo de cambios.

Utilizar herramientas de resolución de conflictos para simplificar el proceso.

PARTE 2 (Nuevas funcionalidades)

1. Creación de nuevas ramas y confirmación de los cambios:

USUARIO D:

```
nicho@LAPTOP-2PCCNVT7 MINGW64 ~/Documents/UNIVERSIDAD SEMESTRE 5/INTEGRATOR_TASK_2024/Practicas-Proyecto (feat/potencia_vehiculo_agregada)
$ git commit -m "feat:potencia_vehiculo_agregada"
[feat/potencia_vehiculo_agregada ea83d9b] feat:potencia_vehiculo_agregada
 1 file changed, 10 insertions(+), 1 deletion(-)
```

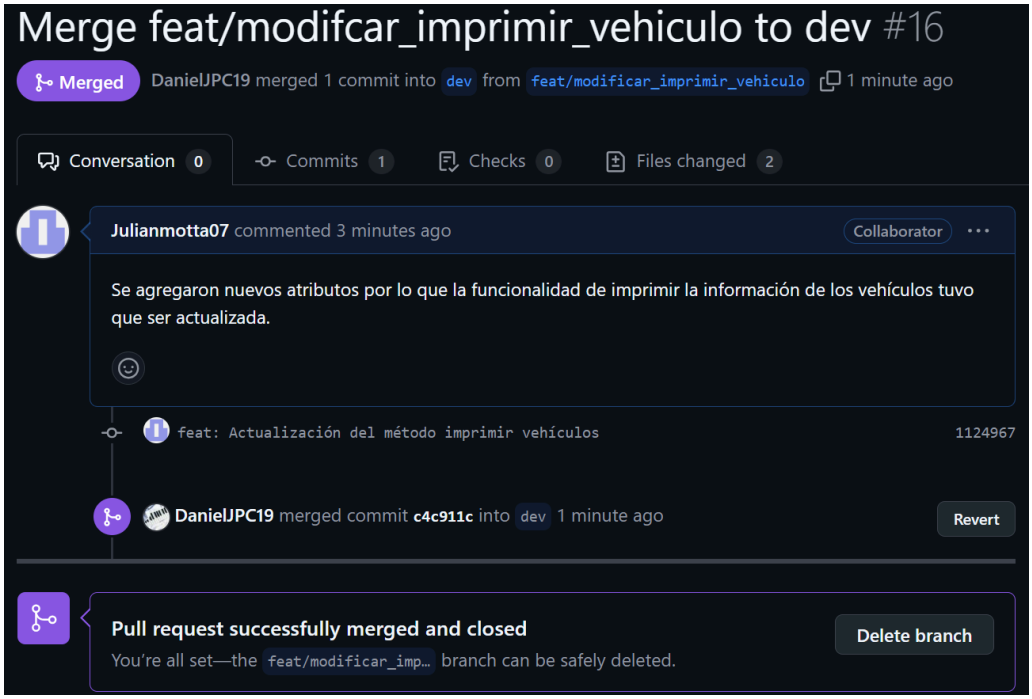
USUARIO E:

```
ASUS@pcmotta MINGW64 ~/Desktop/Taller git 11-09/Practicas-Proyecto (dev)
$ git checkout -b feat/modificar_imprimir_vehiculo
Switched to a new branch 'feat/modificar_imprimir_vehiculo'
```

```
ASUS@pcmotta MINGW64 ~/Desktop/Taller git 11-09/Practicas-Proyecto (feat/modific
ar_imprimir_vehiculo)
$ git commit -m "feat: Actualización del método imprimir vehículos"
[feat/modificar_imprimir_vehiculo 1124967] feat: Actualización del método imprim
ir vehículos
 2 files changed, 17 insertions(+)
```

2. Merge y resolución de conflictos:

USUARIO E:



3. Release:

USUARIO F:

```
Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git commit -m "docs: actualizar descripción"
[release/v1 69e056e] docs: actualizar descripción
 1 file changed, 5 insertions(+), 2 deletions(-)

Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git push origin release/v1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 739 bytes | 369.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/DanielJPC19/Practicas-Proyecto.git
   8021ff0..69e056e  release/v1 -> release/v1

Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git status
On branch release/v1
Your branch is up to date with 'origin/release/v1'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git add --all

Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git commit -m "docs: agregar trabajo por usuario en README.md"
[release/v1 387a4a3] docs: agregar trabajo por usuario en README.md
 1 file changed, 21 insertions(+), 1 deletion(-)

Dani@DESKTOP-DANI19 MINGW64 ~/Documents/Semestre 5/Integrador/git/Practicas-Proyecto (release/v1)
$ git push origin release/v1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 957 bytes | 478.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/DanielJPC19/Practicas-Proyecto.git
   69e056e..387a4a3  release/v1 -> release/v1
```