CEP CVE

Guía Completa de Git – Proyecto de Alquiler de Coches

Asignatura: Programación

Autores: Andrés Pérez Perales, Pablo Corral y Alexander Romano

Índice

1. A. Introducción a Git
2. B. Instalación y configuración
3. C. Conceptos y flujo de trabajo (local vs remoto)
4. D. Comandos esenciales con ejemplos
5. E. Ramas, merges y estrategias
6. F. Conflictos y cómo resolverlos
7. G. Trabajo en equipo: Pull Requests y revisiones
8. H. Buenas prácticas y .gitignore
9. I. Glosario y guía rápida de comandos

A. Introducción a Git

Git es un sistema de control de versiones distribuido. Permite guardar el historial de cambios de un proyecto, colaborar con otras personas, y recuperar versiones anteriores si algo sale mal. En este documento aprenderás desde cero cómo usar Git en el proyecto de alquiler de coches.

B. Instalación y configuración

1) Instalar Git:

Visita https://git-scm.com/ y descarga la versión para tu sistema operativo. En Windows instala Git Bash para usar la terminal.

2) Configurar usuario:

Comandos:

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

3) Archivos importantes:

.gitconfig: configuración global de Git.

.gitignore: lista de archivos que no subiremos al repositorio.

Ejemplo de .gitignore:

```
/node_modules *.log .env .DS_Store
```

C. Conceptos y flujo de trabajo (local vs remoto)

Resumen rápido del flujo: trabajas en tu carpeta de trabajo (working directory), añades cambios al área de preparación (staging), haces commits en tu repositorio local (.git), y finalmente subes (push) los cambios al repositorio remoto (GitHub). Otros colaboradores pueden traer esos cambios con pull/fetch.

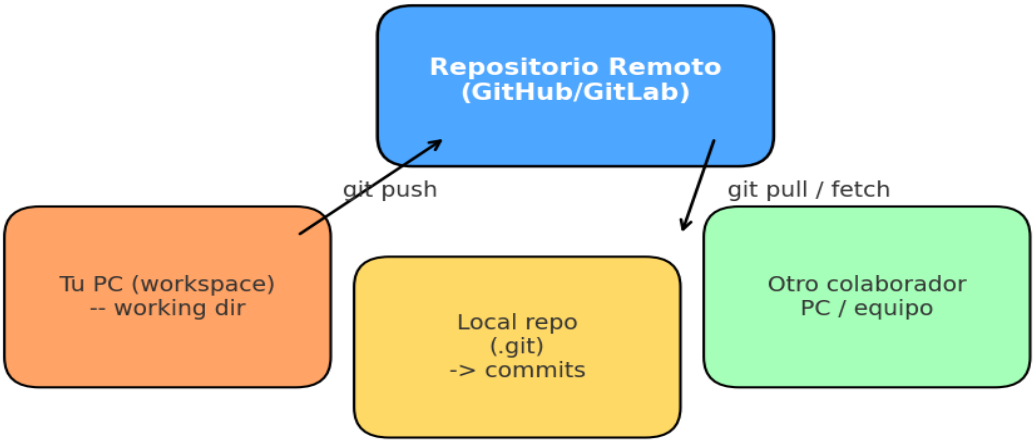

Figura: Relación entre repositorio local y remoto. Usa git push para subir y git pull para traer cambios.

D. Comandos esenciales con ejemplos

git init: Inicializa un repositorio Git en la carpeta actual.

```
Sintaxis / Ejemplo: git init
```

Resultado esperado: Crea la carpeta oculta .git y empieza a registrar cambios.

git clone [url]: Clona un repositorio remoto a tu equipo.

```
Sintaxis / Ejemplo: git clone https://github.com/usuario/alquiler-coches.git
```

Resultado esperado: Crea una copia local del repositorio remoto.

git status: Muestra el estado de archivos (modificados, staged, sin seguimiento).

```
Sintaxis / Ejemplo: git status
```

Resultado esperado: Lista archivos modificados y preparados para commit.

git add [archivo]: Añade archivos al área de preparación (staging).

```
Sintaxis / Ejemplo: git add src/app.js git add .
```

Resultado esperado: Prepara archivos para el siguiente commit.

git commit -m "mensaje": Crea un commit guardando los cambios en el historial.

```
Sintaxis / Ejemplo: git commit -m "Añadida función reservas"
```

Resultado esperado: Guarda un nuevo punto de restauración en el historial.

git log: Muestra el historial de commits.

```
Sintaxis / Ejemplo: git log --oneline --graph --decorate
```

Resultado esperado: Lista los commits con información resumida.

git diff: Muestra las diferencias entre archivos o commits.

```
Sintaxis / Ejemplo: git diff HEAD~1 HEAD
```

Resultado esperado: Muestra qué líneas cambiaron entre versiones.

git branch: Lista ramas o crea una nueva.

```
Sintaxis / Ejemplo: git branch git branch feature/reservas
```

Resultado esperado: Crea o muestra ramas.

git checkout [rama]: Cambia a otra rama o crea+salta a una nueva con -b.

```
Sintaxis / Ejemplo: git checkout -b feature/pago
```

Resultado esperado: Te sitúa en otra rama para trabajar allí.

git merge [rama]: Fusiona cambios de otra rama en la actual.

```
Sintaxis / Ejemplo: git checkout main git merge feature/reservas
```

Resultado esperado: Integra los cambios; puede generar conflictos.

git remote -v: Muestra los remotos configurados.

```
Sintaxis / Ejemplo: git remote -v
```

Resultado esperado: Muestra origin y otras URLs.

git fetch: Descarga objetos y refs del remoto sin fusionar.

```
Sintaxis / Ejemplo: git fetch origin
```

Resultado esperado: Actualiza referencias remotas pero no toca tu working dir.

git pull: git pull = git fetch + git merge (por defecto).

```
Sintaxis / Ejemplo: git pull origin main
```

Resultado esperado: Trae y fusiona cambios del remoto en tu rama actual.

git push: Sube tus commits al remoto.

```
Sintaxis / Ejemplo: git push origin feature/reservas
```

Resultado esperado: Actualiza el repositorio remoto con tus commits.

git revert [commit]: Crea un nuevo commit que deshace otro commit.

```
Sintaxis / Ejemplo: git revert a1b2c3
```

Resultado esperado: Deshace los cambios sin reescribir historial.

git reset: Mueve HEAD; tiene modos --soft, --mixed, --hard.

```
Sintaxis / Ejemplo: git reset --soft HEAD~1
```

Resultado esperado: Deshace commits locales (con precaución).

git stash: Guarda cambios temporales que no quieres commitear aún.

```
Sintaxis / Ejemplo: git stash git stash pop
```

Resultado esperado: Oculta cambios y luego los recupera.

git tag: Marca puntos en el historial (versiones).

```
Sintaxis / Ejemplo: git tag -a v1.0 -m "Release 1.0"
```

Resultado esperado: Crea una etiqueta para releases.

git cherry-pick [commit]: Aplica un commit específico en la rama actual.

```
Sintaxis / Ejemplo: git cherry-pick a1b2c3
```

Resultado esperado: Trae un commit concreto sin merge completo.

E. Ramas, merges y estrategias

Las ramas permiten desarrollar funciones separadas. Algunas estrategias comunes:

Git Flow: usa ramas 'develop', 'release', 'hotfix' y 'main' para un flujo organizado.

Feature Branching: crea ramas por cada nueva función (ej. feature/reservas).

Trunk-based development: ramas pequeñas y merges frecuentes en main.

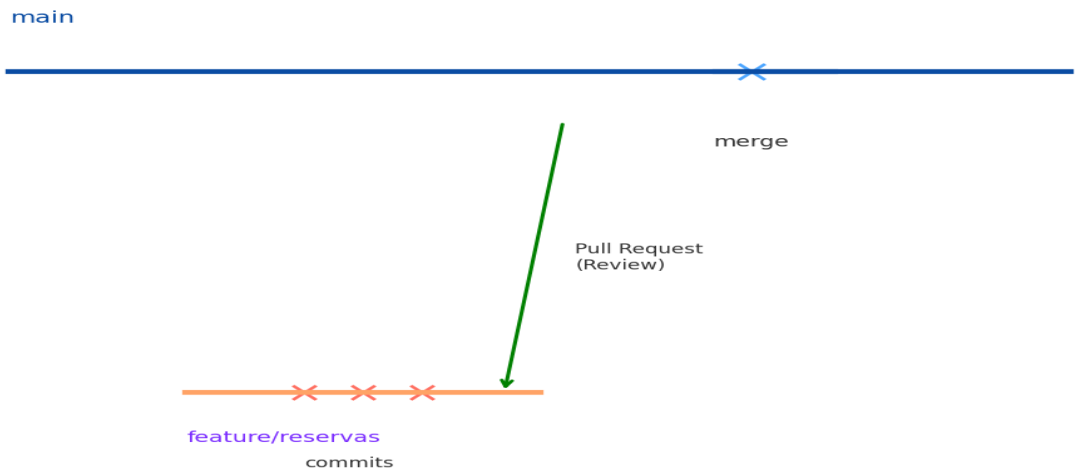

Figura: Flujo típico con una rama feature, revisión y merge a main.

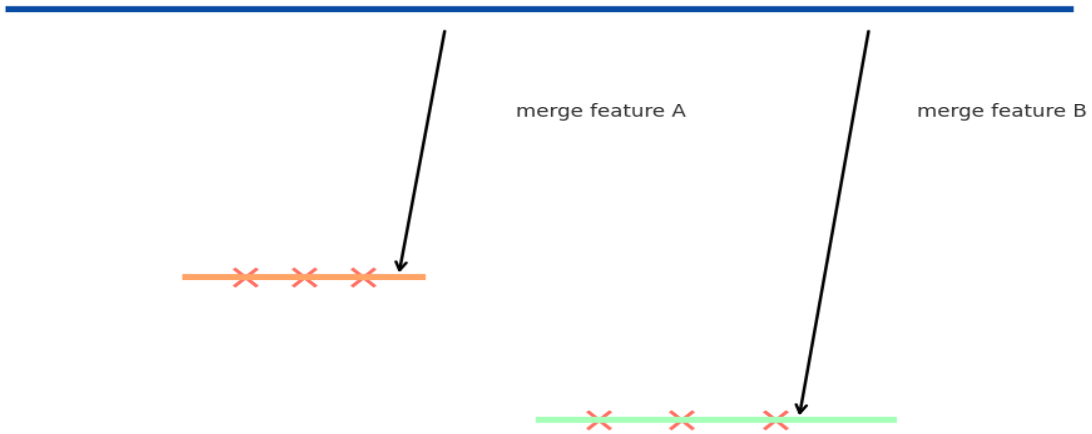

Figura: Varias ramas feature A y B que se fusionan en main.

F. Conflictos y cómo resolverlos

Los conflictos aparecen cuando dos cambios modifican las mismas líneas de un archivo y Git no puede decidir cuál mantener. Pasos para resolver un conflicto:

1. Detectar conflicto: después de git merge o git pull verás archivos 'both modified'.

2. Abrir archivo en conflicto: encontrarás marcas <<<<<<<, =======, >>>>>>> indicando las versiones.

3. Editar y elegir qué conservar (o combinar ambos cambios).

4. git add para marcar como resuelto.

5. git commit para finalizar la fusión (o git merge --continue si usas rebase).

Figura: Conflicto entre Dev A y Dev B. Resuélvelo editando el archivo y commitando la resolución.

Consejos para evitar conflictos:

- Hacer commits y pushes pequeños y frecuentes.

- Comunicar en el equipo qué archivos se van a tocar.

- Rebase con cuidado: git pull --rebase para mantener historial limpio.

G. Trabajo en equipo: Pull Requests y revisiones

Flujo típico en GitHub/GitLab: crear branch -> desarrollar -> push -> abrir Pull Request (PR) -> revisión -> merge.

Buenas prácticas en PRs:

- Escribir descripción clara del cambio.
- Hacer PR pequeños y enfocados.
- Revisar el código de compañeros y dejar comentarios constructivos.

H. Buenas prácticas y .gitignore

Buenas prácticas generales:

- 1. Mensajes de commit claros: Título corto + descripción si hace falta.
- 2. Commits pequeños y atómicos: cada commit una idea/funcionalidad.
- 3. Revisiones de código: pull requests y revisión por pares.
- 4. Proteger la rama main: usar reglas en GitHub para requerir PRs y aprobaciones.
- 5. No subir secretos: usar variables de entorno y .env en .gitignore.

Ejemplo avanzado de .gitignore para un proyecto web:

```
/node_modules .env dist/ *.log .vscode/ .DS_Store
```

I. Glosario y guía rápida

Repositorio: Carpeta con todo el historial de un proyecto (.git para local, remoto en GitHub).

Commit: Foto del proyecto en un momento concreto; lleva autor, fecha y mensaje.

Branch (rama): Línea de desarrollo independiente para nuevas funciones o correcciones.

Merge: Integración de cambios de una rama en otra.

Rebase: Reaplica commits sobre otra base, reescribe historial (usarlo con cuidado).

Staging (index): Área donde preparas cambios antes de commitear (git add).

Pull Request: Solicitud para que el equipo revise y fusione tus cambios al main.

Guía rápida - comandos útiles

Acción	Comando ejemplo
Inicializar repo	git init
Clonar remoto	git clone <url>
Ver estado	git status
Añadir	git add <archivo> | git add .
Commit	git commit -m 'mensaje'
Ver historial	git log --oneline --graph --decorate
Crear rama	git branch feature/x
Cambiar rama	git checkout feature/x | git switch feature/x
Crear y cambiar	git checkout -b feature/x
Actualizar desde remoto	git pull
Subir cambios	git push origin branch
Guardar cambios temporales	git stash
Aplicar commit específico	git cherry-pick <commit>

Consejos finales

- Practica en proyectos pequeños: crea un repo y prueba comandos sin miedo.
- Lee mensajes de git: te dicen exactamente qué pasa (conflictos, rebase, etc.).
- Usa GUI si eres más visual (GitHub Desktop, SourceTree) pero aprende la terminal.
- Mantén el main protegido y revisa los PRs antes de fusionar.