Guía práctica: usar y mantener el repo EDA en varias computadoras (Git + GitHub)

Repo remoto: `https://github.com/Garunix94/EDA.git`

> **Objetivo**: Tener un único repositorio central en GitHub y trabajar desde distintas PCs sin copiar carpetas a mano. Cada cambio se hace en una **rama** (branch), se sube a GitHub y se integra a `main` / `master` solo cuando **aprobás** un Pull Request (PR).
>
> **Nota sobre la rama principal**: GitHub suele usar `main` como rama por defecto. Si tu repo usa `master`, reemplazá `main` por `master` en los comandos.

---

1) Requisitos en cada computadora

1. **Instalar Git**:
2. Windows: https://git-scm.com/download/win (dejar opciones por defecto y que agregue Git al PATH).
3. macOS: `brew install git` (o instalar Xcode Command Line Tools).
4. Linux (Debian/Ubuntu): `sudo apt install git`
5. **Configurar identidad (solo 1ª vez en cada PC)**:

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

6. **(Opcional) Evitar problemas de saltos de línea en Windows**:

```
git config --global core.autocrlf true
```

---

2) Clonar el repositorio en una PC nueva

Elegí una carpeta de trabajo y ejecutá:

```
git clone https://github.com/Garunix94/EDA.git
cd EDA
```

Esto crea una copia local conectada al repo de GitHub (remoto `origin`).

---

3) Flujo de trabajo recomendado (branch + PR)

Para que **vos decidas** qué entra a `main` / `master`, **no trabajes directo** en la rama principal. Usá ramas por cambio/tema y un Pull Request.

A. Antes de empezar a trabajar (actualizá tu copia)

```
cd EDA
git switch main   # o: git checkout main (si tu Git es antiguo)
git pull origin main  # si tu rama principal es master, reemplazar
```

B. Crear una rama para tu cambio

Usá un nombre descriptivo (sin espacios), por ejemplo `feature/listas-enlazadas` o `fix/bug-pop`:

```
git switch -c feature/tu-cambio
```

C. Hacer cambios y ver el estado

- Editá/creá archivos normalmente.
- Revisá qué cambió:

```
git status
```

D. Preparar y guardar los cambios (commit)

1. Agregar todos los cambios staged:

   ```
   git add -A
   ```

2. Crear el commit con un mensaje claro:

   ```
   git commit -m "Implementa cola con punteros; agrega pruebas básicas"
   ```

   Tip: Commits chicos y frecuentes ayudan a revisar y resolver conflictos.

E. Subir la rama al remoto (GitHub)

```
git push -u origin feature/tu-cambio
```

La primera vez puede pedirte iniciar sesión en GitHub (Git Credential Manager en Windows facilita esto).

F. Abrir un Pull Request (PR)

1. Entrá a `https://github.com/Garunix94/EDA`.
2. GitHub suele sugerirte **Compare & pull request** para tu rama recién subida. Si no, andá a **Pull requests → New pull request** y elegí:
3. **base**: `main` (o `master`)
4. **compare**: `feature/tu-cambio`
5. Escribí un título/descripción del cambio y creá el PR.

G. Revisar y decidir (desde tu cuenta propietario)

- Mirá el **diff** del PR.
- Si todo está bien: **Merge** del PR en `main` / `master` (botón **Merge pull request** → **Confirm merge**).
- Si falta algo: dejá comentarios o **solicitá cambios**. El autor edita, hace nuevos commits en la misma rama y vuelve a subir `git push`. Se actualiza el PR automáticamente.

H. Actualizar tu copia local después del merge

Volvé a la rama principal y traé lo último:

```
git switch main
git pull origin main
```

(Recordá usar `master` si esa es tu rama principal.)

I. Limpiar ramas ya integradas (opcional)

```
git branch -d feature/tu-cambio         # borra local
git push origin --delete feature/tu-cambio  # borra en remoto
```

---

4) Resumen exprés (machete)

```
# 1) Preparar
cd EDA
git switch main && git pull origin main

# 2) Rama nueva
git switch -c feature/mi-cambio

# 3) Trabajar
# ... editar archivos ...

git add -A
git commit -m "Mensaje claro del cambio"
```

```
git push -u origin feature/mi-cambio

# 4) Abrir PR en GitHub y pedir revisión
# 5) Merge del PR → vuelve a main/master

git switch main && git pull origin main
```

---

5) Mantener todo sincronizado entre computadoras

En **cada PC** antes de trabajar:

```
cd EDA
git switch main
git pull origin main
```

Trabajá en ramas, subí tu rama y **abrí PR**. Después del merge, en las demás PCs:

```
git pull origin main
```

---

6) Buenas prácticas

- Un **PR por tema** (más fácil de revisar).
- Mensajes de commit **claros y en imperativo** ("Agrega...", "Corrige...").
- Hacé `git pull` antes de empezar y antes de crear tu rama.
- Evitá subir binarios o archivos generados; usá `.gitignore`.
- Nombrá ramas con prefijos: `feature/`, `fix/`, `chore/`.

---

7) Problemas comunes y soluciones

Conflictos al hacer pull o merge

1. Git te mostrará archivos en conflicto.
2. Abrilos y resolvé las marcas `<<<<<<<`, `=======`, `>>>>>>>`.
3. Marcá como resueltos y commiteá:

   ```
   git add -A
   git commit -m "Resuelve conflictos en X"
   ```

4. Si era en tu rama de trabajo, volvé a hacer `git push` y el PR se actualizará.

Cambié directo en main/master por error

- Creá una rama desde ahí para aislar el cambio y abrir PR:

```
git switch -c hotfix/aislar-cambio
git push -u origin hotfix/aislar-cambio
```

- Abrí PR desde esa rama y luego **revertí** o **reseteá** el commit indeseado en `main/master` si fuese necesario.

"Mi rama quedó vieja" (divergió de main/master)

```
git switch feature/tu-cambio
git fetch origin
git merge origin/main   # o origin/master
# resolvé conflictos si aparecen, luego
git add -A && git commit -m "Merge de main/master"
git push
```

---

8) (Opcional) Proteger la rama principal

Para **obligar** a que todo pase por PR y evitar pushes directos a `main/master`:

1. En GitHub: **Settings → Branches → Branch protection rules → Add rule**.
2. Rule pattern: `main` (o `master`).
3. Activá al menos:
4. **Require a pull request before merging**
5. **Restrict who can push to matching branches** (si querés)

---

9) Comandos útiles adicionales

```
git log --oneline --graph --decorate   # historial compacto
git diff                               # ver cambios sin preparar
git restore --staged <archivo>         # sacar de staging
git checkout -- <archivo>              # descartar cambios locales en un
archivo

git remote -v                          # ver remotos
git fetch --prune                      # limpiar refs a ramas borradas en
remoto
```

Checklist rápido para empezar en una PC nueva

-

¡Listo! Con este flujo, **vos siempre decidís** qué entra a la rama principal y mantenés un solo lugar centralizado para todos los códigos.