Guía Breve de GitHub para Curso Microeconomía I

Versión: 13 feb 2025

Repositorios importantes:

https://github.com/adalovelace-ec/matematicasCCS

https://github.com/adalovelace-ec/microeconomia1

Contenido

1. Crear un Nuevo Repositorio en GitHub

1. **Inicia sesión** en tu cuenta de GitHub.
2. Haz clic en la pestaña **Repositorios**.
3. Da clic en el botón verde **New** o **Nuevo repositorio** (parte superior izquierda).
4. Se abrirá la página para crear un repositorio.
5. Completa los siguientes campos:
    - **Repository name:** Escribe un nombre (preferible en minúsculas, sin caracteres especiales y sin espacios).
      Ejemplo: "mate"
    - **Descripción (opcional):** Agrega una breve descripción.
    - **Visibilidad:** Selecciona **Público** (para que otros usuarios puedan ver el repositorio) o **Privado** (solo usuarios autorizados).
    - **Importante:** No marques las opciones para inicializar el repositorio con un README, .gitignore o licencia, ya que ya cuentas con archivos en tu carpeta local.
6. Haz clic en **Create repository** (botón verde al final).
7. Copiar el código para configurar en tu VSC GitHub:

- *Ejemplo para configurar el repositorio (solo la primera vez):*

```
echo "# mate" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/nombreCuentaGit/nombreRepositorio.git
git push -u origin main
```

---

2. Crear un Directorio de Trabajo en tu PC

1. Crea una carpeta en tu computadora (por ejemplo, en el escritorio) donde trabajarás.
2. Abre el directorio en Visual Studio Code (VSC):
   - Selecciona la carpeta en VSC.
   - Acepta las opciones para **confiar en los autores**.
3. **Verifica:**
   - Que tengas instalado Python y Anaconda (y que se haya creado el entorno de trabajo).
   - Puedes consultar la guía correspondiente en nuestro repositorio **https://github.com/adalovelace-ec/matematicasCCS** o visitar:
     - Python Downloads
     - Anaconda

---

3. Subir Archivos de tu Directorio Local a GitHub

1. **Abrir Terminal en VSC:**
   - Dentro de VSC, abre una nueva terminal.
2. **Poner código para configurar el repositorio (solo la primera vez):**

```
echo "# mate" >> README.md
git init
```

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/nombreCuentaGit/nombreRepositorio.git

git push -u origin main

- **Nota:** Deberás autenticarte con tus credenciales de GitHub (clic en Authorize y coloca tu usuario/contraseña o token).

Resultado: Tus archivos ya estarán respaldados en tu repositorio de GitHub.

3. *Resultado:* Tu repositorio queda creado en GitHub.
4. **Cuando quieras agregar nuevos archivos o cambios, sigue los siguientes pasos:**
   - Navega hasta la carpeta de tu proyecto.
   - Ejecuta:

     git init
5. **Agregar y Confirmar Archivos:**
   - Agregar archivos individualmente:

     git add nombre_archivo.py
   - Para agregar todos los archivos modificados:

     git add .
   - Para poner comentarios de lo que vas a subir en tu cuenta de GitNub:

     git commit -m "agregar tus comentarios"
   - Para subir en tu cuenta de GitNub:

     git push

- **Entonces, para subir cambios adicionales al repositorio, debes usar los siguientes comando en tu terminal:**

git init

git add .

git commit -m "añadí cambios triviales"

git push

---

4. Otros Comandos Útiles para Git en la Terminal

- **Verificar el remoto actual:**

git remote -v

- **Eliminar un remoto y agregar uno nuevo:**

Eliminar el remoto actual

git remote remove origin

Agregar un nuevo remoto

git remote add origin https://github.com/DDavidVillamar/mega.git

- **Modificar la URL del remoto existente:**

git remote set-url origin https://github.com/DDavidVillamar/mega.git

---

5. Hacer una Copia (Fork) y Clonar un Repositorio de Terceros

1. **Fork del Repositorio:**
    - Desde GitHub, haz fork del repositorio de terceros para crear una copia en tu perfil.
2. **Clonar el Repositorio Forkeado:**
    - Copia la URL del repositorio fork (botón **Code**).

Remendamos hacer esto para, nuestros 2 repositorios:

https://github.com/adalovelace-ec/matematicasCCS.git

https://github.com/adalovelace-ec/microeconomia1.git

- Abre la terminal en la carpeta donde quieras el proyecto y ejecuta:

Crear la carpeta de trabajo

mkdir algoritmos

cd algoritmos

Clonar el repositorio (esto se hace la primera vez

git clone https://github.com/tu-usuario/tu-repositorio.git

para nuevas actualizaciones entras en el directorio del repositorio:

cd tu-repositorio

Para traer las últimas actualizaciones del repositorio remoto:

git pull origin main

Este comando descarga y fusiona los cambios del repositorio remoto a tu copia local sin necesidad de clonar nuevamente.

---

6. Comandos Útiles para Navegación en la Terminal

- **Ver en qué carpeta estás:**

pwd

- **Ver el contenido de la carpeta actual:**
    - En **Windows:**

dir

    - En **Linux/Mac:**

ls

- **Regresar a la carpeta anterior:**

cd ..

- **Ir a una ruta específica:**

cd C:\Users\nombre-usuario\Desktop\nombre-carpeta

- **Crear varias carpetas anidadas de una vez:**

mkdir carpeta1/carpeta2/carpeta3

- **Eliminar una carpeta:**

En Windows:

- **Para eliminar una carpeta en Terminal de VSC**

rm -r nombrearchivo -Force

En Mac:

rm -rf nombrearchivo

7.¿Qué son las ramas en GitHub y por qué son importantes?

- **Definición:**
  Una **rama** en GitHub (y en Git) es una versión paralela del código principal. Imagina que el repositorio es un árbol, y cada rama es una ramita en la que puedes trabajar de forma independiente sin afectar el tronco principal (generalmente llamado main o master).
- **Propósito:**
  - **Aislamiento:** Permiten desarrollar nuevas funcionalidades, corregir errores o probar ideas sin alterar la versión estable del código.
  - **Colaboración:** Cada desarrollador puede trabajar en su propia rama, y luego, mediante un proceso de revisión (Pull Request), integrar sus cambios al proyecto principal.
  - **Control de cambios:** Facilitan el seguimiento de qué cambios se hicieron, quién los hizo y en qué rama se originaron, ayudando a mantener un historial claro y organizado.

- **Ejemplo Práctico:**

1. **Crear una rama nueva:**

git checkout -b nueva-funcionalidad

Este comando crea la rama nueva-funcionalidad y te mueve automáticamente a ella.

Para cambiarte a una rama existente se usa:

git checkout nombre-de-la-rama

2. **Realizar cambios:**
Modifica o añade archivos en esta rama. Por ejemplo, podrías crear un archivo funcion_nueva.py con el código de la nueva funcionalidad.

3. **Confirmar y subir los cambios:**

git add funcion_nueva.py

git commit -m "Añadida nueva funcionalidad de análisis"

git push origin nueva-funcionalidad

Así, tus cambios se guardan en la rama nueva-funcionalidad en GitHub.

4. **Fusionar la rama:**

Una vez que hayas probado y revisado los cambios, puedes crear un Pull Request en GitHub para fusionar la rama nueva-funcionalidad con la rama principal (main). Esto permite que los cambios se integren de forma controlada y con posibilidad de revisión por parte del equipo.

Resumen:
Las ramas son herramientas esenciales para:

- Probar nuevas ideas sin riesgo.
- Trabajar de forma colaborativa y ordenada.
- Mantener la estabilidad del proyecto mientras se desarrollan mejoras.

Este enfoque no solo mejora la organización del código, sino que también facilita el trabajo en equipo y la revisión de cambios, aspectos fundamentales en proyectos de desarrollo y análisis en economía.

Para subir los cambios de una rama al main, existen dos métodos comunes: usando la línea de comandos o mediante un Pull Request en GitHub. Aquí te explico ambos:

Método 1: Usando la Línea de Comandos para actualizar cambios de una rama en main

1. **Asegúrate de que todos los cambios estén confirmados y subidos en tu rama.** Si aún no lo has hecho:

git add .

git commit -m "Descripción de los cambios"

git push origin nombre-de-tu-rama

2. **Cámbiate a la rama main:**

git checkout main

3. **Actualiza la rama main con los últimos cambios del repositorio remoto:**

git pull origin main

4. **Fusiona la rama con los cambios en main:**

git merge nombre-de-tu-rama

    - Si aparecen conflictos, resuélvelos manualmente, guarda los cambios y luego confirma la fusión.

5. **Sube la rama main actualizada al repositorio remoto:**

git push origin main

Método 2: Hacerlo directamente en la página de GitHub.

8. Guía de Colaboración en GitHub Usando Ramas

1. Configuración Inicial del Proyecto

a) Crear el Repositorio

1. **Inicia sesión** en GitHub.
2. Haz clic en **Repositorios** y luego en el botón **New**.
3. Completa los datos:
    - **Repository name:** Elige un nombre (por ejemplo, proyecto-colaborativo).
    - **Visibilidad:** Selecciona Público o Privado según el objetivo.
    - *Importante:* No inicialices con README, .gitignore o licencia si vas a subir archivos locales.
4. Haz clic en **Create repository**.

b) Invitar a Colaboradores

1. En el repositorio, haz clic en **Settings** (Configuración).
2. Selecciona la pestaña **Manage access**.
3. Haz clic en **Invite a collaborator** y agrega el nombre de usuario o correo electrónico del colaborador.
4. El invitado recibirá un mensaje para aceptar la invitación.

---

2. Clonar el Repositorio en Cada Computadora

Cada colaborador debe clonar el repositorio a su máquina local:

1. Abre la terminal y ejecuta:

```
git clone https://github.com/tu-usuario/proyecto-colaborativo.git
```

2. Entra en la carpeta del proyecto:

```
cd proyecto-colaborativo
```

---

3. Trabajo en Ramas Independientes

a) Crear una Nueva Rama para Cada Funcionalidad o Tarea

Cada colaborador debe crear su propia rama para trabajar en cambios sin afectar la rama principal (main).

1. Para crear y cambiar a una nueva rama:

```
git checkout -b nombre-de-la-rama
```

Ejemplo:

- Colaborador 1: git checkout -b nueva-funcionalidad
- Colaborador 2: git checkout -b correccion-bug

b) Realizar Cambios y Confirmarlos (Commits)

1. Modifica o agrega archivos en tu rama.
2. Añade los cambios:

git add .

3. Realiza un commit describiendo los cambios:

git commit -m "Descripción de los cambios realizados"

c) Subir la Rama a GitHub

1. Sube la rama al repositorio remoto:

git push origin nombre-de-la-rama

---

4. Integrar Cambios Mediante Pull Requests (PR)

a) Crear un Pull Request

1. En GitHub, ve al repositorio.
2. Verás una notificación que indica que se han subido ramas nuevas; haz clic en **Compare & pull request** para la rama en la que trabajaste.
3. Agrega un título y una descripción detallada de los cambios.
4. Envía el Pull Request.

b) Revisar y Aprobar el Pull Request

1. El otro colaborador o el equipo puede revisar el PR.
2. Comenta si es necesario para solicitar cambios o sugerencias.
3. Una vez aprobado, haz clic en **Merge pull request** para fusionar la rama con main.

c) Actualizar el Repositorio Local

1. Cambia a la rama main:

git checkout main

2. Actualiza tu copia local con:

git pull origin main

---

5. Buenas Prácticas para la Colaboración

- **Comunicación:** Coordina con tu colaborador quién trabaja en qué parte del código para evitar solapamientos.

- **Commits Claros y Frecuentes:** Realiza commits pequeños y descriptivos.
- **Uso de Ramas:** Trabaja siempre en una rama diferente a main y fusiona mediante Pull Requests.
- **Resolución de Conflictos:** Si surgen conflictos al fusionar, resuélvelos en conjunto antes de hacer el merge.
- **Mantener Actualizada la Rama Principal:** Realiza git pull regularmente para estar al día con los cambios de tus compañeros.

Estos pasos te ayudarán a establecer un flujo de trabajo colaborativo en GitHub, aprovechando las ramas para desarrollar, probar y fusionar cambios de manera ordenada y segura.