Actividad Guiada: Colaboración con Git y GitHub

En esta actividad, aprenderéis a colaborar en un mismo proyecto utilizando Git y GitHub. Seguiremos un **flujo de trabajo básico pero fundamental** para cualquier equipo de desarrollo. El objetivo es que os **familiaricéis con los comandos esenciales** y las buenas prácticas para trabajar de forma sincronizada.

Herramientas:

- **Navegador web** (para GitHub)
- **Línea de comandos de Windows (CMD)**
- **Un editor de texto** (como el Bloc de notas)

Organización: Equipo de 4 o 5 alumnos del proyecto intermodular de 2º DAM.

Parte 1: Configuración del Entorno de Trabajo

Paso 1: Alta en GitHub si aún no tenéis cuenta

Cada miembro del equipo debe tener su propia cuenta de GitHub.

1. Id a github.com.
2. Haced clic en **"Sign up"** y seguid las instrucciones para crear una cuenta gratuita.

Paso 2: Crear una Organización

La organización en GitHub servirá como el contenedor de vuestro proyecto de equipo.

1. Uno rol **Hacedor** del equipo debe crear la organización.
2. Haced clic en el icono `+` en la esquina superior derecha de GitHub y seleccionad **"New organization"**.
3. Elegid el plan gratuito ("Free").
4. Nombrad la organización siguiendo este formato: `[nombreequipo]_pi2damiesbalmis`. Por ejemplo: `equipo_alfa_pi2damiesbalmis`.
5. Completad los pasos de creación.

Paso 3: Añadir Miembros a la Organización

Ahora, el propietario de la organización debe invitar al resto de compañeros y al **profesor**.

1. Dentro de la página de la organización, id a la pestaña **"People"**.
2. Haced clic en el botón **"Invite member"**.
3. Buscad por nombre de usuario de GitHub o email a cada compañero y al profesor y enviad las invitaciones.
4. Los invitados recibirán un correo para aceptar la invitación. ¡Aceptadla!
5. Cambia is el rol del profesor a **"Owner"** para que tenga permisos completos.

Paso 4: Crear un Equipo

Los equipos ayudan a gestionar los permisos de acceso a los repositorios.

> **Nota**
>
> Crear un equipo es algo opcional en este caso, pero es una buena práctica para gestionar permisos en proyectos más grandes.

1. En la página de la organización, id a la pestaña **"Teams"**.
2. Haced clic en **"New team"**.
3. Dadle un nombre al equipo, por ejemplo, "***equipo_alfa***".
4. En el siguiente paso, añadid a todos los miembros de la organización al equipo.

Paso 5: Crear el Repositorio del Proyecto

El repositorio contendrá todos los ficheros de vuestro proyecto.

1. En la página principal de la organización, haced clic en el botón verde **"New"** para crear un nuevo repositorio.
2. **Repository name:** `pruebas`
3. **Description:** (Opcional) "Repositorio para practicar el flujo de Git".
4. Seleccionad **"Public"**.
5. **Importante:** Marcad la casilla **"Add a README file"**. Esto inicializa el repositorio con un primer fichero.
6. Haced clic en **"Create repository"**.

Parte 2: Trabajo en Local con Git

Paso 6: Clonar el Repositorio y Configuración Inicial

Cada miembro del equipo debe tener una copia local del proyecto.

1. **Configuración inicial de Git (solo la primera vez en un ordenador):**
   Abrid el CMD de Windows y configurad vuestro nombre y email. Esto es crucial para que Git sepa quién realiza cada cambio.

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

2. **Clonar el repositorio:**
   - En la página del repositorio `pruebas` en GitHub, haced clic en el botón verde `[<> Code]`.
   - Copiad la URL HTTPS (por ejemplo: `https://github.com/nombre-organizacion/pruebas.git`).
   - En el CMD, navegad a la carpeta donde queráis guardar el proyecto (ej: `cd repositorios`) y ejecutad:

```
git clone [URL_COPIADA_DEL_REPOSITORIO]
```

   - Navegad dentro de la carpeta recién creada:

```
cd pruebas
```

Paso 7: Crear y Moverse a una Rama Personal

Cada uno trabajará en su propia rama para no interferir con el trabajo de los demás. Una rama es como una línea temporal independiente del proyecto.

1. **Crear una nueva rama:** Reemplazad `[tu-nombre]` por vuestro nombre en minúsculas.

```
git branch [tu-nombre]
```

2. **Moverse a la nueva rama:**

```
git checkout [tu-nombre]
```

> **Nota**
>
> Acabáis de crear una "copia" del estado actual de la rama `main` y os habéis movido a ella para empezar a trabajar. Todos vuestros cambios a partir de ahora se harán en esta nueva rama.

Paso 8: Crear y Añadir un Fichero

Ahora, cada uno añadirá su propio fichero al proyecto.

1. **Crear un fichero Markdown:** Usad el Bloc de notas o vuestro editor preferido para crear un fichero con vuestro nombre. Por ejemplo, `juan.md`.

```
notepad juan.md
```

2. Añadid contenido al fichero. Podéis usar sintaxis Markdown, por ejemplo:

```
# Fichero de Juan

* Me gusta el desarrollo de software.
* Mi lenguaje favorito es Python.
```

3. Guardad y cerrad el fichero.
4. **Añadir el fichero al "staging area"**: Le indicamos a Git que queremos incluir este fichero en nuestro próximo "guardado" (commit).

```
git add juan.md
```

5. **Hacer commit**: Guardamos los cambios de forma permanente en el historial de nuestra rama.

```
git commit -m "Añade fichero de Juan"
```

Paso 9: Subir la Rama a GitHub

Vuestros cambios (la nueva rama y el nuevo fichero) solo existen en vuestro ordenador. Ahora los subiréis a GitHub.

```
git push --set-upstream origin [tu-nombre]
```

> **Nota**
>
> El comando `--set-upstream origin [tu-nombre]` crea un enlace entre tu rama local y una nueva rama con el mismo nombre en el repositorio remoto (GitHub), para que en futuros `push` solo necesites escribir `git push`.

Parte 3: Integración y Colaboración

Paso 10: Sincronizar y Fusionar en `main`

Ahora, cada miembro del equipo (de uno en uno, para evitar conflictos iniciales) integrará sus cambios en la rama principal `main`.

1. **Volver a la rama `main`:**

   ```
   git checkout main
   ```

2. **Traer los últimos cambios de `main` desde GitHub:** Esto es una buena práctica para asegurar que tienes la última versión antes de fusionar tus cambios.

   ```
   git pull origin main
   ```

3. **Fusionar (merge) tu rama personal en `main`:**

   ```
   git merge [tu-nombre]
   ```

   > **Nota**
   >
   > Este comando coge todos los commits de tu rama personal y los aplica sobre la rama `main` en tu ordenador. Ahora tu `main` local tiene los cambios de tus compañeros (del `pull`) y los tuyos (del `merge`).

Paso 11: Subir los Cambios de `main`

Ahora que tu `main` local está actualizada con tus cambios, súbela a GitHub para que los demás puedan verla.

```
git push origin main
```

> **Importante**
>
> El resto del equipo debe ahora ejecutar `git pull origin main` en sus ramas `main` locales para recibir los cambios que acabas de subir.

Paso 12: Actualizar tu Rama y Añadir Nuevo Contenido

Antes de seguir trabajando en tu rama, es buena idea sincronizarla con los cambios que se han añadido a `main`.

1. **Vuelve a tu rama personal:**

   ```
   git checkout [tu-nombre]
   ```

2. **Fusiona los cambios de `main` en tu rama:**

   ```
   git merge main
   ```

3. **Añade nuevo contenido a tu fichero:**

   ```
   notepad [tu-nombre].md
   ```

   Añade una nueva sección y guarda los cambios.

4. **Haz `add` y `commit` de los nuevos cambios:**

   ```
   git add [tu-nombre].md
   git commit -m "Actualiza fichero con nueva información"
   ```

Paso 13: Colaborar con un Pull Request

En lugar de subir los cambios directamente a `main`, ahora usaremos un **Pull Request (PR)**. Este es un mecanismo para proponer cambios y pedir que alguien los revise antes de integrarlos.

1. **Sube los últimos cambios de tu rama a GitHub:**

   ```
   git push origin [tu-nombre]
   ```

2. **Crea el Pull Request en GitHub:**
   - Ve a la página del repositorio en tu navegador.
   - Verás un mensaje amarillo que dice **"Your branch had recent pushes"**. Haz clic en el botón verde **"Compare & pull request"**.
   - Se abrirá un formulario. Asegúrate de que la base sea `main` y la rama a comparar sea `[tu-nombre]`.
   - Añade un título y una descripción para tu PR.
   - En la barra lateral derecha, busca la sección **"Reviewers"** y selecciona a un compañero del equipo para que revise tus cambios.
   - Haz clic en **"Create pull request"**.
3. **Revisar y Aprobar el Pull Request (acción del revisor):**
   - El compañero asignado recibirá una notificación.
   - Debe ir a la pestaña **"Pull Requests"** del repositorio y abrir tu PR.
   - En la pestaña **"Files changed"**, puede ver los cambios que has hecho.
   - Si todo está correcto, debe hacer clic en **"Review changes"**, seleccionar **"Approve"** y enviar la revisión.
   - Finalmente, el revisor debe hacer clic en el botón verde **"Merge pull request"** y confirmar la fusión. Esto integrará tus cambios en la rama `main` directamente en GitHub.

Parte 4: Gestión de Conflictos

Paso 14: Simular y Resolver un Conflicto

¿Qué pasa si dos personas modifican la misma línea del mismo fichero al mismo tiempo? Git no sabrá qué cambio conservar y se producirá un **conflicto**. Vamos a simularlo.

Preparación:

- **Dos miembros del equipo** (Persona A y Persona B) se ponen de acuerdo.

- Ambos deben tener su rama `main` local actualizada: `git checkout main` y luego `git pull origin main`.
- Van a modificar el fichero `README.md`.

Acciones Simultáneas:

1. **Persona A:**
   - Abre `README.md` (`notepad README.md`).
   - Modifica la primera línea añadiendo: `Proyecto del Equipo Alfa`.
   - Guarda el fichero.
   - Ejecuta los siguientes comandos:

     ```
     git add README.md
     git commit -m "Modificación de Persona A"
     git push origin main
     ```

   - El `push` de la Persona A tendrá éxito. ¡Ha sido más rápida!
2. **Persona B:**
   - Al mismo tiempo, abre `README.md`.
   - Modifica la **misma primera línea** añadiendo: `Pruebas de colaboración - Equipo Alfa`.
   - Guarda el fichero.
   - Ejecuta los siguientes comandos:

     ```
     git add README.md
     git commit -m "Modificación de Persona B"
     git push origin main
     ```

El Conflicto:

El `push` de la Persona B **fallará**. Git mostrará un error indicando que el repositorio remoto tiene cambios que no tienes en tu local. Git te sugerirá que hagas un `git pull` primero.

Resolución (Persona B):

1. **Intenta sincronizar con `git pull`:**

   ```
   git pull origin main
   ```

2. Git intentará fusionar los cambios remotos con los tuyos, pero fallará y mostrará un mensaje de `MERGE_CONFLICT`. Te dirá en qué fichero está el conflicto (`README.md`).
3. **Abre el fichero conflictivo:**

```
notepad README.md
```

Verás algo como esto:

```
<<<<<<< HEAD
Proyecto del Equipo Alfa
=======
Pruebas de colaboración - Equipo Alfa
>>>>>>> [hash-del-commit-remoto]
```

Explicación de los marcadores:

- `<<<<<<< HEAD`: Indica el inicio de tus cambios locales.
- `=======`: Separa tus cambios de los cambios que vienen del repositorio remoto.
- `>>>>>>> ...`: Marca el final de los cambios remotos.

4. **Resuelve el conflicto:** Debes editar el fichero manualmente para decidir cómo debe quedar la versión final. Elimina los marcadores de Git (`<<<`, `===`, `>>>`) y deja el contenido deseado. Por ejemplo, podéis decidir combinar ambas ideas:

```
# Pruebas de colaboración - Proyecto del Equipo Alfa
```

5. **Guarda el fichero** una vez resuelto.
6. **Informa a Git de que el conflicto está resuelto:**

```
git add README.md
```

7. **Completa la fusión con un commit:**

```
git commit
```

Se abrirá un editor de texto con un mensaje de commit predefinido. Simplemente guarda y cierra el editor para confirmar.

8. **Ahora sí, haz el `push` final:**

```
git push origin main
```

¡Éxito! El conflicto ha sido resuelto y los cambios están en GitHub.

Push Directo vs. Pull Request

Aunque en este ejercicio hemos hecho ambas cosas, en entornos profesionales casi siempre se prefiere el Pull Request para la rama principal.

Característica	Push Directo a `main`	Pull Request (PR)
Velocidad	**Ventaja:** Muy rápido y directo. Ideal para proyectos personales o correcciones triviales.	**Inconveniente:** Proceso más lento, requiere más pasos y la intervención de otra persona.
Calidad y Revisión	**Inconveniente:** No hay revisión de código. Es fácil introducir errores (`bugs`) en la rama principal.	**Ventaja:** Permite la **revisión por pares**. Otros pueden revisar el código, sugerir mejoras y detectar errores antes de que lleguen a `main` .
Colaboración	**Inconveniente:** Es un proceso aislado. No fomenta la discusión sobre los cambios.	**Ventaja:** Es el centro de la colaboración. Permite discutir los cambios, hacer preguntas y mantener un registro de las decisiones tomadas.
Integración Continua	**Inconveniente:** Puede romper el `build` o las pruebas automáticas si se sube código defectuoso.	**Ventaja:** Se pueden ejecutar pruebas automáticas sobre el PR. Si las pruebas fallan, se bloquea la fusión, protegiendo la rama `main` .
Historial del Proyecto	**Ventaja:** Genera un historial más lineal y simple.	**Inconveniente:** Puede generar un historial más "ruidoso" con commits de fusión, aunque existen estrategias para evitarlo (squash, rebase).

Glosario de Comandos Utilizados

Comando	Descripción en esta actividad
`git config --global user.name`	Configura tu nombre de autor para todos tus repositorios.
`git config --global user.email`	Configura tu email de autor para todos tus repositorios.
`git clone [url]`	Crea una copia local de un repositorio remoto de GitHub.
`git branch [nombre-rama]`	Crea una nueva rama.
`git checkout [nombre-rama]`	Se mueve a la rama especificada para empezar a trabajar en ella.
`git add [fichero]`	Añade un fichero modificado al "área de preparación" (staging) para incluirlo en el próximo commit.
`git commit -m "mensaje"`	Guarda una instantánea de los cambios preparados en el historial local de la rama.
`git push origin [nombre-rama]`	Sube los commits de tu rama local a la rama correspondiente en GitHub (remoto `origin`).
`git pull origin [nombre-rama]`	Descarga los cambios de una rama remota y los fusiona en tu rama local actual.
`git merge [nombre-rama]`	Fusiona el historial de la rama especificada en la rama actual.