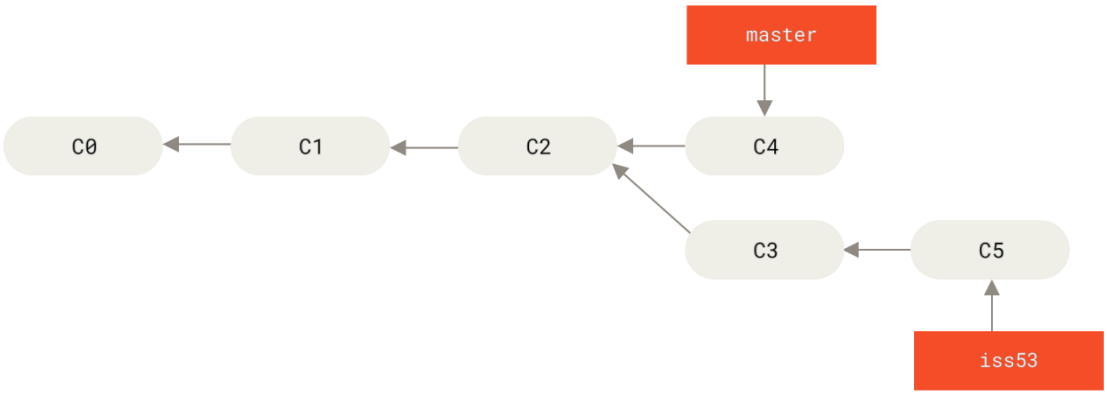
* Práctica 1.4: GitHub Básico

P1.4 - Gestión de ramas con Git y enlazar a GitHub

1. Trabajar con ramas

Antes de nada, veamos qué es una rama y los procedimientos básicos para ramificar y fusionar



Vamos a comenzar la práctica donde lo dejamos en la práctica anterior, P3.

1. Abrimos Git Bash o una terminal y nos movemos a la carpeta de nuestro proyecto. Ahora vamos a ejecutar un comando para crear una ***rama nueva*** y también nos va a situar en ella:

```
> git checkout -b desEjercicios1
```

   Ya estamos en otra rama de desarrollo, no en **main** *(la rama principal)*. En esta nueva rama vamos a tener todos los ficheros de la rama main tal y cómo están en este mismo momento *(cómo si hubiéramos sacado una foto exacta)*.

3. A continuación, creamos un directorio con el nombre `ejercicios1`, y dentro de esta nueva carpeta, creamos un programa que se llame `prueba2.py` que solicite dos números, los sume y muestre el resultado:

```
num1 = int(input("Dame un número: "))
num2 = int(input("Dame otro: "))
print("La suma " + str(num1) + " + " + str(num2) + " es igual a " + str(num1 + num2))
print(f"{num1} + {num2} = {num1+num2}")
```

8. Añadimos el fichero al área de preparación:

```
> git add .
```

10. Observad que los IDEs, tienen integrado Git y podéis hacerlo también desde el mismo IDE, ya que ha detectado que el proyecto está gestionado por Git *(En PyCharm, haciendo clic con el botón derecho del ratón encima del fichero aparecen las opciones de Git)*

11. Después hacemos Commit para pasarlo al repositorio:
```
> git commit -m "Creado el programa prueba2"
```
13. Si nos cambiamos de nuevo a la rama `main`, podemos observar cómo desaparece el fichero `prueba2.py`:
```
> git checkout master
> ls -l
> ls ejercicios1
> git status
```
18. Si cambiamos a la rama `desEjercicios1` volverá a aparecer `prueba2.py`:
```
> git checkout desEjercicios1
> ls ejercicios1
> git status
```
22. Si ya hemos acabado el trabajo en nuestra rama y queremos actualizar todas las modificaciones a la rama principal, debemos ***FUSIONAR*** los cambios de la rama `desEjercicios1` con la rama `main`:
```
> git checkout master
> git merge desEjercicios1
```
25. Si hay conflictos y falla, hacemos lo siguiente para arreglarlo:

    La forma más directa de resolver un conflicto de fusión es editar el archivo conflictivo. Abre el archivo que tiene el conflicto en el editor o el IDE. Vamos a eliminar todas las líneas divisorias de conflictos, como estas:

    ```
    <<<<<<< master
    =======
    >>>>>>> desEjercicios1
    ```

    y arreglar el archivo hasta que quede como se desea. Cuando hayas editado el archivo, utiliza `git add archivoEditado.txt` para preparar el nuevo contenido fusionado. Para finalizar la fusión, crea una nueva confirmación ejecutando lo siguiente:

    ```
    > git commit -m "merged and resolved the conflict in
    archivoEditado.txt"
    ```

    Git verá que se ha resuelto el conflicto y crea una nueva confirmación de fusión para finalizar la fusión.

26. Para ver las ramas que tengo:
```
> git branch
```
28. Si estando en la rama `main` creo un fichero nuevo o una modificación, pero me olvido de añadirlo con `add` y hacer `commit`, y cambio de rama... pues me estoy llevando todo los cambios a la rama a la que nos hemos cambiado por no haber hecho `commit`.
29. Para eliminar una rama:
```
> git branch -d nombreRama
```

2. Trabajar con `stash`

Se trata de un comando que “congela” el estado en el que se encuentra el proyecto en un momento determinado, con todos los cambios que tenemos a "sin comitear", y lo guarda en una pila provisional brindando la posibilidad de poder recuperarlo más adelante.

1. Por ejemplo, si tengo cambios y no quiero hacer Commit en ese momento porque mis cambios aún no son definitivos, podemos dejarlo en un área temporal:
2. `> git stash`
3. A partir de ese momento podré cambiar de rama ya sin problemas y al volver de nuevo a mi rama para seguir trabajando en los cambios revierto el stash:
4. `> git stash pop`
5. También puedo hacer los `stash` que queramos con:
6. `> git stash save "Primer stash"`
7. Para ver los stash que tengo:
8. `> git stash list`
9. Para recuperar uno en concreto de la lista:
10. `> git stash pop stash@{2}`

3. Gestión del proyecto enlazado con Github

GitHub es una plataforma de desarrollo colaborativo que utiliza el sistema de control de versiones Git. Git gestiona repositorios locales en tu ordenador, permitiendo el seguimiento de cambios en tu código. GitHub extiende esta funcionalidad al proporcionar repositorios remotos alojados en la nube, lo que facilita la colaboración entre equipos dispersos geográficamente. Permite a los desarrolladores subir sus cambios a un repositorio en línea y fusionarlos con el trabajo de otros. Además, GitHub ofrece herramientas como seguimiento de problemas, ramificaciones y solicitudes de extracción para facilitar la colaboración y la gestión de proyectos.

Ojo, parar conectar git y github, tendreis que tener una clave publica. Aqui se explica cómo.

1. Lo primero que debemos hacer es ir a la página web de GitHub y registrarnos con nuestro correo corporativo de `xxxxxxx@g.educaand.es`

   GitHub - Creación y configuración de la cuenta

2. Después crearemos un repositorio en GitHub (para más información podéis acceder a la documentación de GitHub, Creación de un repositorio). El nombre del repositorio puede ser el siguiente, dependiendo del curso dónde estés:
   - DAM1_ProgPhyton
   - DAW1A_ProgPython
   - DAW1B_ProgPython
3. Ahora ya tenemos nuestro proyecto gestionado por Git en local y un repositorio en la nube. Para enlazarlos y así estar tranquilos que nuestros ficheros nunca se van a perder seguiremos las instrucciones que nos proporciona GitHub al crearnos el nuevo repositorio para conectarlo con mi repo local... por ejemplo, si lo hubiéramos llamado `PruebasProgPython` y mi usuario de GitHub fuera `dcansib438`, nos generaría un comando similar al siguiente *(lo copiamos y pegamos en Git Bash o la terminal)*:
4. `> git remote add origin https://github.com/dcansib438/PruebasProgPhyton.git`
5. Para subir cambios a Github:
6. `> git push origin master`

7. Para descargar los cambios que se hayan realizado en Github, realizados desde otro ordenador u otro desarrollador distinto:
8. `> git pull origin master`

Otros enlaces

- GitHub
- Creación de un repositorio en github)
- Chuleta de comandos git
- Ayuda visual
- Guia rápida
- Documentación de referencia de git
- Libro de Git