

Proyecto Git y GitHub

A) Repositorio local

1. Explica la diferencia entre Git y GitHub.

Git es el software que te permite hacer control de version por línea de comandos, y GitHub te permite hacerlo de forma gráfica.

2. Busca el libro Git avalado por la comunidad GitHub.

Indica la url del sitio web en el que lo has encontrado: <https://git-scm.com/book/es/v2>

¿Cuál es la última versión?

¿Puedes descargarlo en diferentes formatos? Indica qué formatos.

¿Está en versión en español?

- La versión 2.1.442

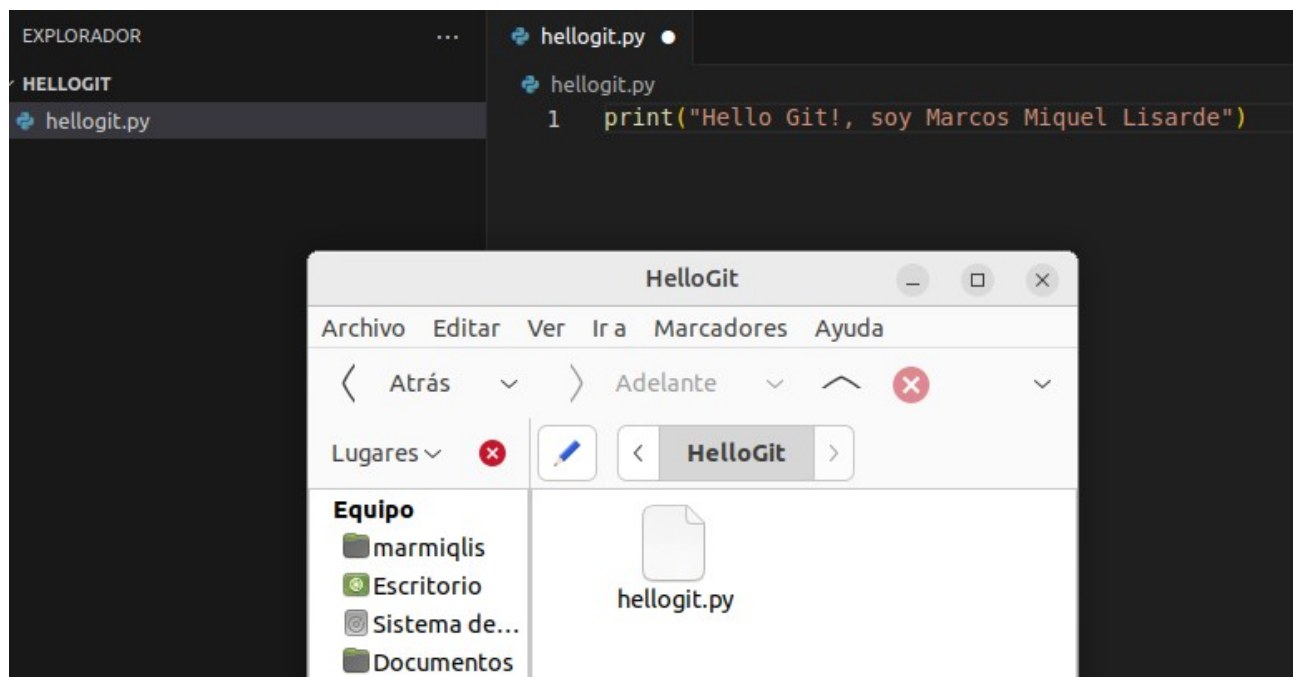
- Lo puedes descargar en .pdf, .epub y .mobi

- Si, está en español y muchos otros idiomas

3. Busca documentación detallada sobre la configuración de Git.

Indica la url del sitio web en el que lo has encontrado: <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Configurando-Git-por-primera-vez>

4. Imagina que vas a desarrollar una aplicación con diferentes funcionalidades y compuesta por varios ficheros. Vas a comenzar en tu entorno local y usando la consola, crea una carpeta o directorio con el nombre HelloGit, y dentro de ella un fichero python con el nombre hellogit.py, que contenga la instrucción `print("Hello Git!, soy TunombreyApellidos")`. Puedes usar Visual Studio Code.



5. Crea un proyecto Git: ¿Que comando Git usas para crear el repositorio Git en el directorio HelloGit, convirtiéndolo así en el directorio de trabajo?:

```
marmiqlis@INF1-03:~$ git init /media/marmiqlis/Festplatte/repositorioPrueba/HelloGit/
```

6. Un proyecto Git tiene tres secciones principales:

- Directorio Git: es donde Git almacena todo lo que necesita para poder hacer un seguimiento y control de las versiones del proyecto, ¿cual es la ruta del directorio Git de tu proyecto HelloGit?
- /media/marmiqlis/Festplatte/repositorioPrueba/HelloGit/.git
- Directorio de trabajo (o árbol de trabajo): es donde un usuario realiza cambios locales en un proyecto, ¿qué ficheros tienes en el directorio de trabajo de tu proyecto HelloGit?
Por ahora ninguno.
- Zona de "staging": es un archivo (también llamado "index", "stage" o "cache") que almacena información sobre los ficheros que se incluirán en tu próximo commit.

7. Indica los tres estados principales en los que puede estar un fichero del repositorio en un momento dado.

- *Confirmado (committed), modificado (modified) y preparado (staged)*

8. Una vez configurado usuario y email de Git, ¿qué comando Git usas para ver usuario y email configurado? Incluye captura.

```
marmiqlis@INF1-03:~$ git config --list
user.email=marcosmiquel77@gmail.com
user.name=Marcos
```

9. Haz un commit de hellogit.py: git commit -m "primer commit hellogit", ¿cual es el hash o id de este commit? Incluye captura.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git add *
```

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "primer commit hellogit"
[master (commit-raiz) 9a46dc5] primer commit hellogit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hellogit.py
```

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git log
commit 9a46dc587e1976af4dcbdd02e994a3a15bce0a7f (HEAD -> master)
Author: Marcos <marcosmiquel77@gmail.com>
Date: Mon Jan 27 08:30:06 2025 +0100

    primer commit hellogit
```

10. Crea un fichero `hellogit2.py`, que contenga la instrucción `print("Hello Git 2!, soy TuNombreyApellidos")`. Haz un commit: `git commit -m "segundo commit, incluyo hellogit2"`. ¿cual es el hash de este commit? Incluye captura.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git add *
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "segundo commit , incluyo hellogit2"
[master 79bcdc4] segundo commit , incluyo hellogit2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hellogit2.py
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git log
commit 79bcdc4ed4b37e325525db10e512575cd8d5ddd0 (HEAD -> master)
Author: Marcos <marcosmiquel77@gmail.com>
Date: Mon Jan 27 08:37:28 2025 +0100

    segundo commit , incluyo hellogit2

commit 9a46dc587e1976af4dcbdd02e994a3a15bce0a7f
Author: Marcos <marcosmiquel77@gmail.com>
Date: Mon Jan 27 08:30:06 2025 +0100

    primer commit hellogit
```

11. Para ver la rama de cambios de forma gráfica puedes usar `git log --graph --decorate --all --oneline`. Para evitar escribir estos comandos largos se pueden crear alias, crea un alias con el nombre `tree` y pruebalo. Incluye captura.

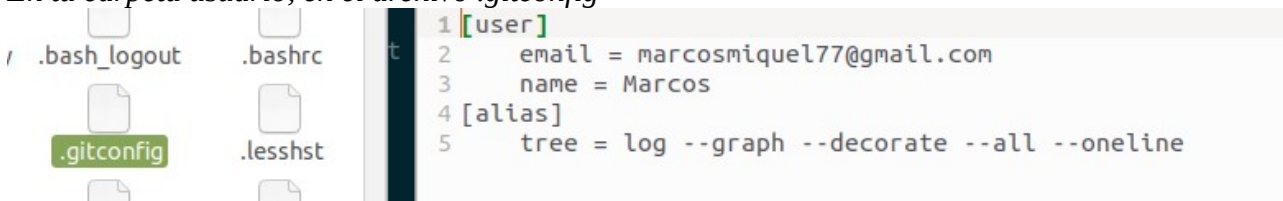
Para crearlo: `git config --global alias.tree "log --graph --decorate --all--oneline"`

Para probarlo: `git tree`

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git config --global alias.tree "log --graph --decorate --all --oneline"
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* 79bcdc4 (HEAD -> master) segundo commit , incluyo hellogit2
* 9a46dc5 primer commit hellogit
```

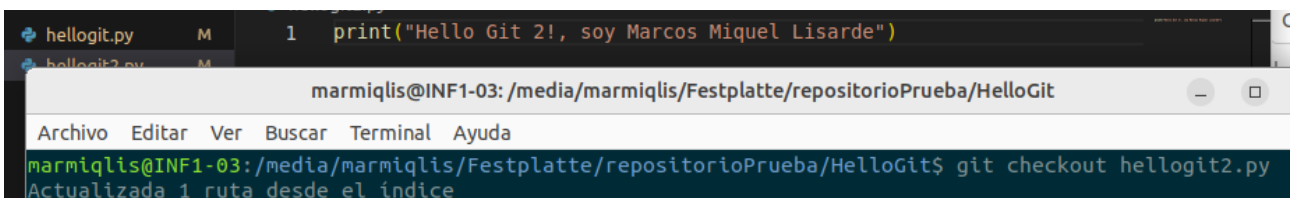
¿Sabes en que fichero de Git queda almacenado este alias?

En tu carpeta usuario, en el archivo `.gitconfig`



```
1 [user]
2     email = marcosmiquel77@gmail.com
3     name = Marcos
4 [alias]
5     tree = log --graph --decorate --all --oneline
```

12. Imagina que llevas varias horas programando. Para simular esto, cambia la línea de `hellogit.py` por `print("Hello Git!, soy TuNombreyApellidos y he hecho cambios")` y guarda, sin hacer commit. también cambia la línea de `hellogit2.py` por `print("Hello Git 2!, soy TuNombreyApellidos y he hecho MUCHOS cambios")` y guarda sin hacer commit. En `hellogit2.py` no quieres esos cambios, ¿qué comando Git usas para deshacer solo cambios realizados en el fichero `hellogit2.py` de tu directorio de trabajo? Hazlo y comprueba que se ha eliminado lo último que has añadido (la parte de la frase: y he hecho muchos cambios). Incluye captura.



```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git checkout hellogit2.py
Actualizada 1 ruta desde el índice
```

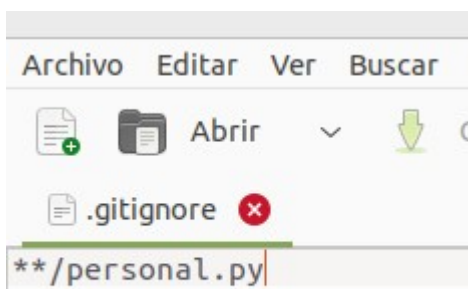
13. Cambia la línea de `hellogit2.py` por `print("Hello Git 2!, soy TuNombreyApellidos y he hecho cambios")` y guarda. Haz un commit: `git commit -m "tercer commit con cambios"`. Incluye captura.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "tercer commit con cambios"
[HEAD desacoplado 15150c2] tercer commit con cambios
2 files changed, 2 insertions(+), 1 deletion(-)
```

14. Ahora imagina que sigues trabajando y has cambiado tantas cosas que ya te has perdido y no es suficiente con deshacer cambios en un fichero, necesitas volver a un commit anterior. Modifica la línea de código de `hellogit2.py` con `print("Hello Git 2!, soy TuNombreApellido y con tantos cambios me he perdido")`. ¿Qué comando Git usa para volver al primer commit? Hazlo e incluye captura

- No hay una forma limpia de hacer esta operación. Se podría forzar forzar con un `git reset` pero se perderían casi todos los commits hechos hasta ahora.

15. Crea un fichero con el nombre `personal.py`, este fichero contendrá código de pruebas que haces en el proyecto pero no quieres que forme parte de la foto o commit. Resumiendo, quieres que Git ignore siempre este fichero en tus commits. Crea en la raíz del proyecto el fichero con el nombre `.gitignore` con la línea `**/personal.py` (el contenido del fichero `.gitignore` son todos los nombres o rutas de los ficheros a ignorar en los commits). Ahora haz `git add .gitignore` y `git commit -m "incluyo el .gitignore"`. Incluye captura.

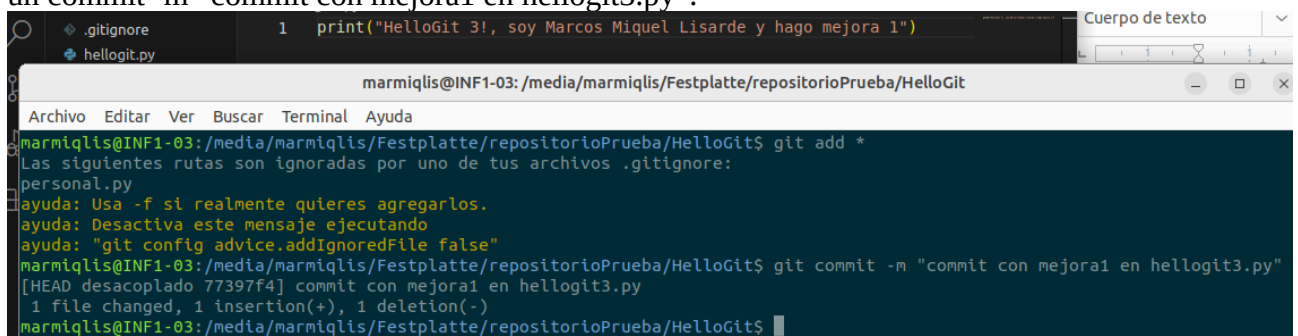


```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git add .gitignore
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "incluyo el .gitignore"
[HEAD desacoplado 0af40eb] incluyo el .gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore
```

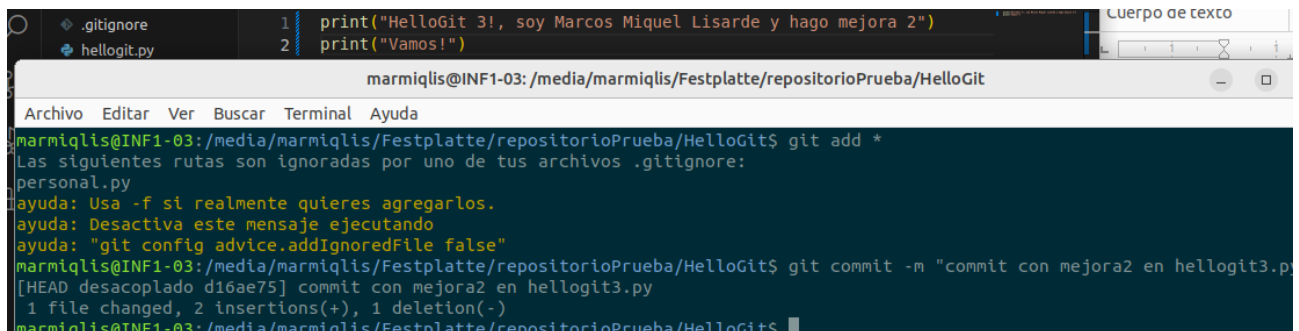
16. Crea el archivo `hellogit3.py`, con la instrucción `print("Hello Git 3!", soy TuNombreApellido)`, haz un commit `-m "commit con hellogit3.py"`.



17. Cambia en `hellogit3.py` por `print("Hello Git 3!", soy TuNombreApellido y hago mejora1)`. Haz un commit `-m "commit con mejora1 en hellogit3.py"`.



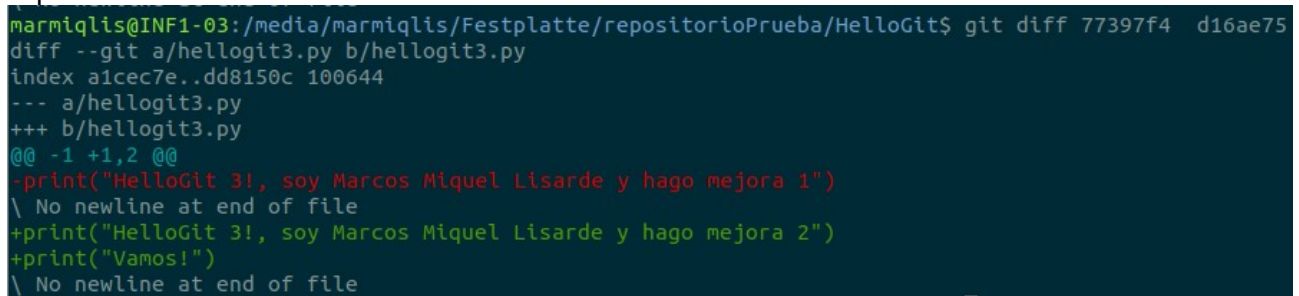
17. Cambia en `hellogit3.py` por `print("Hello Git 3!", soy TuNombreApellidos y hago mejora2)` y añade `print("¡Vamos!")`. Haz un commit -m "commit con mejora2 en `hellogit3.py`".



```
.gitignore
1 print("HelloGit 3!, soy Marcos Miquel Lissarde y hago mejora 2")
2 print("¡Vamos!")

marmiqlis@INF1-03: /media/marmiqlis/Festplatte/repositorioPrueba/HelloGit
Archivo Editar Ver Buscar Terminal Ayuda
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git add *
Las siguientes rutas son ignoradas por uno de tus archivos .gitignore:
personal.py
ayuda: Usa -f si realmente quieres agregarlos.
ayuda: Desactiva este mensaje ejecutando
ayuda: "git config advice.addIgnoredFile false"
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "commit con mejora2 en hellogit3.py"
[HEAD desacoplado d16ae75] commit con mejora2 en hellogit3.py
1 file changed, 2 insertions(+), 1 deletion(-)
```

18. Ahora imagina que te has dado cuenta que el commit que has hecho con la mejora2 tiene un error y no lo quieres, ¿con qué comando ves las diferencias con el commit anterior?. Incluye captura.



```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git diff 77397f4 d16ae75
diff --git a/hellogit3.py b/hellogit3.py
index a1cec7e..dd8150c 100644
--- a/hellogit3.py
+++ b/hellogit3.py
@@ -1,2 @@
- print("HelloGit 3!, soy Marcos Miquel Lissarde y hago mejora 1")
\ No newline at end of file
+ print("HelloGit 3!, soy Marcos Miquel Lissarde y hago mejora 2")
+ print("¡Vamos!")
\ No newline at end of file
```

19. Explica detalladamente que ocurre en el proyecto si en este momento haces `git revert` con el id de este último commit de la mejora2. Hazlo e incluye captura con `git tree` o `git log --oneline`.



```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git revert d16ae75
[HEAD desacoplado e982604] Revert "commit con mejora2 en hellogit3.py"
1 file changed, 1 insertion(+), 2 deletions(-)
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* e982604 (HEAD) Revert "commit con mejora2 en hellogit3.py"
* d16ae75 commit con mejora2 en hellogit3.py
* 77397f4 commit con mejora1 en hellogit3.py
* 828f904 commit con hellogit3.py
* 0af40eb incluyo el .gitignore
* 15150c2 tercer commit con cambios
* 9e5594f (master) segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

20. Crea una rama llamada `login` para desarrollar una nueva parte del proyecto que se encarga de hacer el login de usuario, usa el comando `git branch login`. Incluye captura con `git tree`.



```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git branch login
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* e982604 (HEAD, login) Revert "commit con mejora2 en hellogit3.py"
* d16ae75 commit con mejora2 en hellogit3.py
* 77397f4 commit con mejora1 en hellogit3.py
* 828f904 commit con hellogit3.py
* 0af40eb incluyo el .gitignore
* 15150c2 tercer commit con cambios
* 9e5594f (master) segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

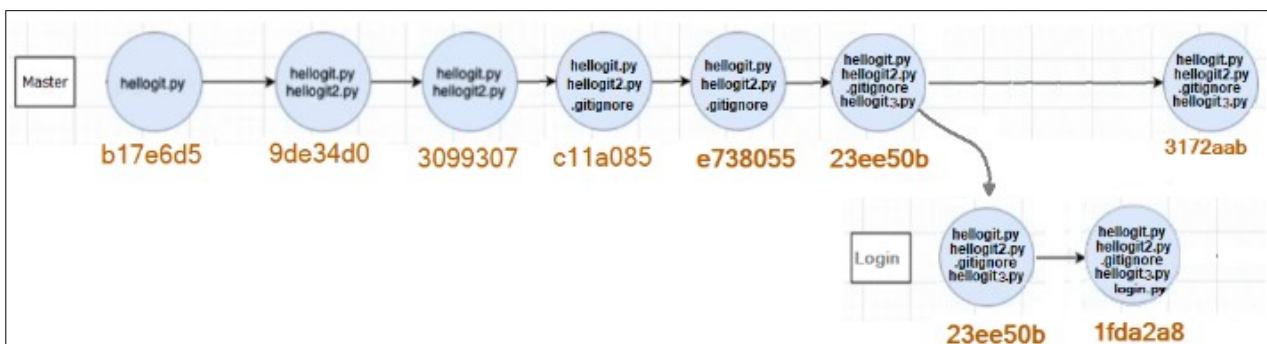
21. Puedes ver que HEAD está apuntando a la rama master, pero se ha creado una nueva rama login, ¿con qué comando te situas en esa rama login? Hazlo e incluye captura con get tree. Observa la diferencia con la captura anterior. Observa lo que ocurre en Visual Studio Code cuando pasas de una rama a otra, incluye capturas.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git switch login
Cambiado a rama 'login'
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* e982604 (HEAD -> login) Revert "commit con mejora2 en hellogit3.py"
* d16ae75 commit con mejora2 en hellogit3.py
* 77397f4 commit con mejora1 en hellogit3.py
* 828f904 commit con hellogit3.py
* 0af40eb incluyo el .gitignore
* 15150c2 tercer commit con cambios
* 9e5594f (master) segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

22. Has creado la rama login desde la rama master, Git ha incluido en la rama login todo lo de la rama master y puedes seguir trabajando en la rama login. Crea en la rama login un nuevo fichero login.py con la instrucción print("login"). Haz un commit -m "commit con login.py". Incluye captura con get tree.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git commit -m "commit con login.py"
[login b4c8e68] commit con login.py
1 file changed, 1 insertion(+)
create mode 100644 login.py
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ log tree
Orden «log» no encontrada, pero hay 16 similares.
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* b4c8e68 (HEAD -> login) commit con login.py
* e982604 Revert "commit con mejora2 en hellogit3.py"
* d16ae75 commit con mejora2 en hellogit3.py
* 77397f4 commit con mejora1 en hellogit3.py
* 828f904 commit con hellogit3.py
* 0af40eb incluyo el .gitignore
* 15150c2 tercer commit con cambios
* 9e5594f (master) segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

Un resumen gráfico de todos los commits que has hecho, que son las fotos del proyecto realizadas en un momento concreto, podría ser similar al siguiente:



23. Estas trabajando con dos ramas, master y login:

1º) Pasate a la rama master, haz un cambio en hellogit.py, con la instrucción `print("Hello Git!, soy TuNombreApellidos, cambio después de crear rama login")`. Haz un commit: `git commit -m "cambio en hellogit.py después de crear rama login"`.

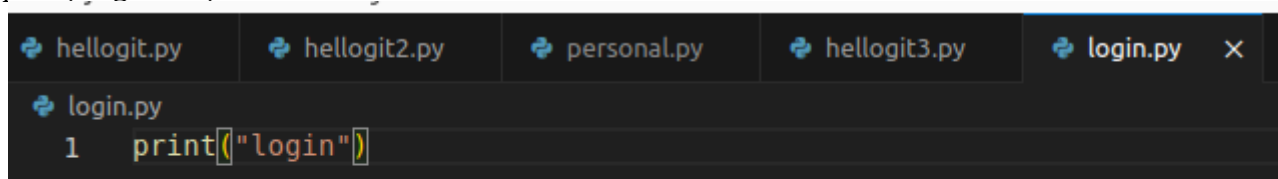
```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* ea7e56f (HEAD -> master) cambio en hellogit.py despues de crear rama login
| * b4c8e68 (login) commit con login.py
| * e982604 Revert "commit con mejora2 en hellogit3.py"
| * d16ae75 commit con mejora2 en hellogit3.py
| * 77397f4 commit con mejora1 en hellogit3.py
| * 828f904 commit con hellogit3.py
| * 0af40eb incluyo el .gitignore
| * 15150c2 tercer commit con cambios
|/
* 9e5594f segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

2º) Ese nuevo commit que has hecho, en la rama master, es posterior a la creación de la rama login, por tanto no está incluido en la rama login. Sitúate en la rama login con el comando `git switch login`. ¿Qué comando usas para traer los cambios hechos en la rama master a la rama login en la que estás situado? Hazlo e incluye captura. Observa el cambio del código en Visual Studio Code e incluye captura.

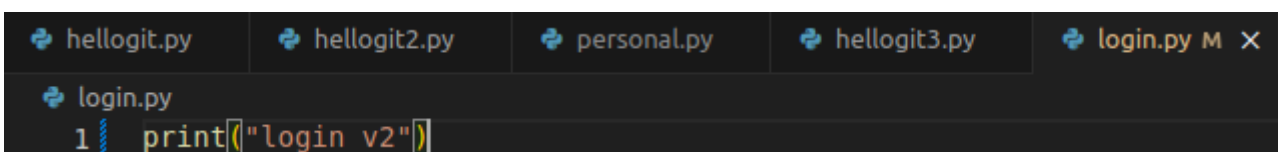
```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git merge master log
in
Ya está actualizado.
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* 8504d3b (HEAD -> login) commit obligatorio para hacer merge
| \
| * ea7e56f (master) cambio en hellogit.py despues de crear rama login
| * | b4c8e68 commit con login.py
| * | e982604 Revert "commit con mejora2 en hellogit3.py"
| * | d16ae75 commit con mejora2 en hellogit3.py
| * | 77397f4 commit con mejora1 en hellogit3.py
| * | 828f904 commit con hellogit3.py
| * | 0af40eb incluyo el .gitignore
| * | 15150c2 tercer commit con cambios
|/
|/
* 9e5594f segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```

24. Ahora cambia la línea de código de login.py `print("login")` por `print("login v2")` y guarda. Intenta cambiar a la rama master, Git te muestra aviso de que tienes cambios sin commit que podrías perder al cambiar de rama.

Para eso existe el comando **git stash**, que es un commit temporal que no se refleja en el historial de commits, solo zona local, podemos verlo con **git stash list**. Haz `git stash` y Git te dejará pasarte a la rama master. Imagina que has estado solucionando algún error de código en la rama master y has terminado, cambiate a la rama login, ¿qué instrucción ves ahora en login.py, `print("login")` o `print("login v2")`?



Usa el comando **git stash pop** para traer todo lo que dejaste en el commit temporal y seguir trabajando. También podrías usar **git stash drop** para descartar ese commit temporal. Incluye captura



25. ¿Con qué comando podemos eliminar la rama login?

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git branch -d login
```

¿Y con qué comando podemos ver las ramas que hay?

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git branch
* master
```

¿Se ha eliminado completamente la rama o se podría recuperar?

Se podría recuperar con un checkout y el código que aparece al borrar la rama

Elimina la rama login e incluye captura de consola y de Visual Studio Code.

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git branch -D login
Eliminada la rama login (era 8504d3b).
```

```
marmiqlis@INF1-03:/media/marmiqlis/Festplatte/repositorioPrueba/HelloGit$ git tree
* ea7e56f (HEAD -> master) cambio en hellogit.py despues de crear rama login
* 9e5594f segundo commit, incluyo hellogit2
* 5a5476a primer commit HelloGit
```