## COMPROBAR EL ESTADO DEL PROYECTO

git status

En *Git*, una rama (*branch*) es una versión del proyecto en el que estás trabajando; aquí puedes ver que estamos en una rama llamada master:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
        .gitignore
        hello_git.py

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Cada vez que compruebas el estado del proyecto, debería mostrar que estás en la rama máster. Entonces puedes ver que estamos a punto de hacer el *commit* inicial.
Un commit es una captura del proyecto en un momento determinado.
Git nos informa que hay archivos sin seguimiento en el proyecto porque no le hemos dicho todavía qué archivos seguir. A continuación nos dice que no hay nada agregado al commit actual, pero hay archivos sin seguimiento que podríamos querer agregar al repositorio.

## AÑADIENDO ARCHIVOS AL REPOSITORIO

Añadamos dos archivos al repositorio y comprobemos de nuevo el estado:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git add .
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master

No hay commits todavía

Cambios a ser confirmados:
  (usa "git rm --cached <archivo>..." para sacar del área de stage)
        nuevos archivos: .gitignore
        nuevos archivos: hello_git.py

usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

El comando *git add .* Añade todos los archivos de un proyecto que todavía no están siendo rastreados al repositorio. Esto no hace *commit* a los archivos, sólo le dice a *Git* que empiece a prestarles atención. Cuando ahora comprobamos el estado del proyecto, podemos ver que *Git* reconoce algunos cambios que necesita para hacer *commit*. La etiqueta *new file* significa que estos archivos se agregaron recientemente al repositorio.

## HACIENDO UN COMMIT

Hagamos nuestro primer *commit:*

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git commit -m "Started project."
[master (commit-raíz) b4b2eb4] Started project.
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello_git.py
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Emitimos el comando *git commit -m "message"* para obtener una instantánea del proyecto. El flag *-m* le dice a *Git* que guarde el mensaje siguiente ("*Started project.*") en el log del proyecto. La salida muestra que estamos en la rama master y que esos dos archivos han cambiado.
Cuando comprobamos de nuevo el estado, podemos ver que estamos en la rama máster y que tenemos un árbol de trabajo limpio. Este es el mensaje que quieres ver cada vez que haces un *commit* al estado de trabajo de tu proyecto. Si obtienes un mensaje diferente, léelo con cuidadosamente; es probable que hayas olvidado añadir un archivo antes de hacer el *commit*.

## COMPROBANDO EL LOG

Git guarda un log de todos los commits hecho al proyecto. Comprobemos el log:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log
commit b4b2eb48220b001bbf74cb89b2db34fe94789f78 (HEAD -> master)
Author: Juanjesuses <juanj.rodriguez.sanchez@gmail.com>
Date:   Tue Nov 1 08:14:09 2022 +0100

    Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Cada vez que haces un *commit, Git* genera una única referencia ID de 40 caracteres. Registra quién realizó el *commit*, cuando lo hizo y el mensaje grabado. No siempre necesitarás toda esta información, así que Git te proporciona una opción para imprimir una versión simple de las entradas del log:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log --pretty=oneline
b4b2eb48220b001bbf74cb89b2db34fe94789f78 (HEAD -> master) Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

El argumento *–pretty=oneline* las dos partes más importantes de la información: la referencia ID del commit y el mensaje guardado para el *commit*.

## El segundo commit

Para ver el auténtico poder del control de versiones necesitamos hacer un cambio en el proyecto y hacer commit a ese cambio. Aquí sólo añadiremos otra línea al hello_git.py

print(“Hello Git world!”)
print(“Hello everyone.”)

Cuando comprobemos el estado del proyecto veremos que Git ha informado que el archivo ha cambiado:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
        modificados:     hello_git.py

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Vemos la rama en la que estamos trabajando, el nombre del fichero que ha sido modificado y que no se ha hecho commit a los cambios. Hagamos commit al cambio y comprobemos de nuevo el estado:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git commit -am "Extended greeting."
[master 8f095dd] Extended greeting.
 1 file changed, 1 insertion(+)
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log --pretty=oneline
8f095dd418ae011a551568fa842859f5e951adf4 (HEAD -> master) Extended greeting.
b4b2eb48220b001bbf74cb89b2db34fe94789f78 Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Hacemos un nuevo *commit* pasándole como flag *-am* cuando usamos el comando *git commit*. El flag *-a* le dice a Git que añada todos los archivos modificados en el repositorio al commit actual. (Si creas algunos ficheros nuevos entre commits, simplemente vuelve a emitir el comando *git add* para incluir los nuevos archivos en el repositorio.) El flag *-m* le dice a *Git* que grabe el mensaje en el log para este commit.

Cuando comprobamos el estado del proyecto, vemos que de nuevo se ha limpiado el directorio de trabajo. Finalmente vemos los dos commits en el *log*.

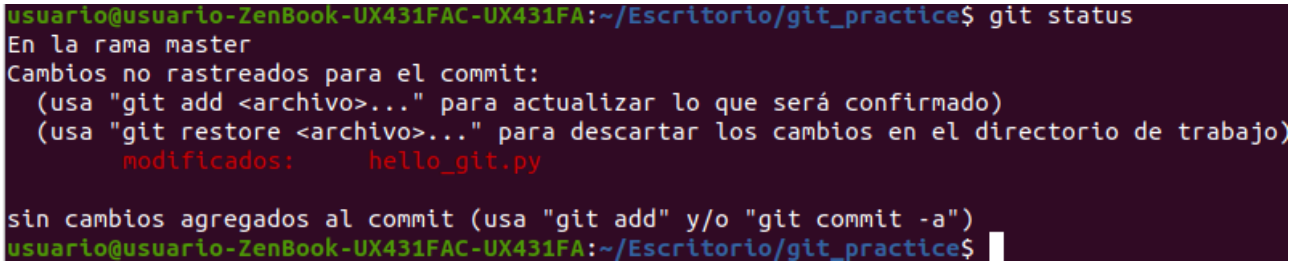
## Revirtiendo un cambio

Ahora veamos como abandonar un cambio y volver al estado de trabajo previo. Primero añadamos una nueva línea a *hello_git.py*:

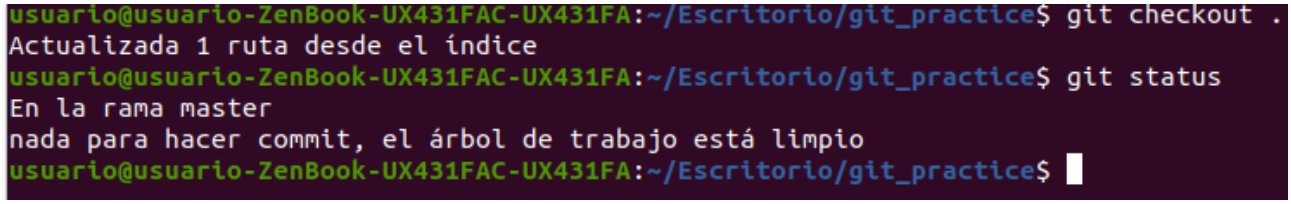
print(“Hello Git world!”)
print(“Hello everyone.”)

print(“Oh no, I broke the project!”)

Salva y ejecuta el archivo.
Comprobamos el estado y vemos que Git nos informa del cambio:




*Git* ve que hemos modificado *hello_git.py* y podemos hacer commit al cambio si queremos. Pero esta vez en lugar de hacer commit al cambio, revertiremos al último commit cuando sabíamos que nuestro proyecto funcionaba. No haremos nada con el archivo hello_git.py: no eliminaremos la línea o utilizaremos la función deshacer en el editor de texto. En su lugar, introduciremos los siguientes comandos en nuestra sesión de terminal:




A pesar de que volvemos a un estado previo podemos observar un cambio trivial en este proyecto simple, si estuviéramos trabajando en un gran proyecto con docenas de archivos modificados, todos los archivos que cambiaron desde el último commit podrían ser revertidos.
Esta funcionalidad es increíblemente útil: puedes hacer tantos cambio como quieras cuando implementas una nueva funcionalidad y si no funciona, puedes descartarlos sin que afecte a tu proyecto. No tienes que recordar estos cambios y deshacerlos manualmente. *Git* hace todo eso por ti.

Puedes refrescar tu archivo en tu editor para ver la versión anterior.

### COMPROBACIÓN DE COMMITS ANTERIORES

Puedes comprobar cualquier commit en tu log, no sólo los más recientes, incluyendo los primeros seis caracteres de la referencia ID en lugar del punto. Comprobando y revisando un commit anterior, puedes volver al último commit o abandonar tu trabajo reciente y continuar desarrollando desde el commit anterior.

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log --pretty=oneline
8f095dd418ae011a551568fa842859f5e951adf4 (HEAD -> master) Extended greeting.
b4b2eb48220b001bbf74cb89b2db34fe94789f78 Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git checkout b4b2eb
Nota: cambiando a 'b4b2eb'.

Te encuentras en estado 'detached HEAD'. Puedes revisar por aquí, hacer
cambios experimentales y hacer commits, y puedes descartar cualquier
commit que hayas hecho en este estado sin impactar a tu rama realizando
otro checkout.

Si quieres crear una nueva rama para mantener los commits que has creado,
puedes hacerlo (ahora o después) usando -c con el comando checkout. Ejemplo:

  git switch -c <nombre-de-nueva-rama>

O deshacer la operación con:

  git switch -

Desactiva este aviso poniendo la variable de config advice.detachedHead en false

HEAD está ahora en b4b2eb4 Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Cuando compruebas un commit anterior, dejas la rama máster y entras en lo que Git refiere como *detached HEAD state*. *HEAD* es el estado actual del commit de tu proyecto. Estás separado por has dejado una rama nombrada (máster en este caso). Para volver a la rama máster, comprueba esto:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git checkout master
La posición previa de HEAD era b4b2eb4 Started project.
Cambiado a rama 'master'
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Este comando te devuelve a la rama máster. A menos que quieras trabajar con algunas funcionalidades más avanzadas de Git, es mejor no hacer ningunos cambios en tu proyecto cuando compruebes un commit antiguo. Sin embargo, si eres el único trabajando en un proyecto y quieres descartar todos los commits más recientes y regresar al estado previo, puedes resetear el proyecto a un commit previo. Trabajando desde la rama máster, introduce lo siguiente:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log --pretty=oneline
8f095dd418ae011a551568fa842859f5e951adf4 (HEAD -> master) Extended greeting.
b4b2eb48220b001bbf74cb89b2db34fe94789f78 Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git reset --hard b4b2eb
HEAD está ahora en b4b2eb4 Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git log --pretty=oneline
b4b2eb48220b001bbf74cb89b2db34fe94789f78 (HEAD -> master) Started project.
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Primero comprobamos el estado para estar seguros de que estamos en la rama máster. Cuando miramos el log, vemos ambos commits. Luego lanzamos el comando git reset –hard con los primeros seis caracteres de la referencia ID del commit que queremos revertir de forma permanente.

Comprobamos de nuevo el estado y vemos que estamos en la rama máster sin nada que hacer commit. Cuando miramos de nuevo el log, vemos que estamos en el commit desde el que queríamos comenzar.

Eliminando el repositorio

A veces estropearás el historial de tu repositorio y no sabrás cómo recuperarlo. Si esto sucede, en primer lugar considera pedir ayuda utilizando los métodos debatidos en el Apéndice C. Si no puedes solucionarlo y estás trabajando en un solo proyecto puedes continuar trabajando con los archivos pero puedes deshacerte del historial del proyecto eliminado el directorio *.git*. Esto no afectará al estado actual de cualquiera de los archivos pero eliminará todos los commits, por lo que no podrás comprobar cualquier otro estado del proyecto.

Para hacer esto, o abres un archivo en el explorador y eliminas el repositorio *.git* o lo eliminas desde la línea de comandos. Después necesitas empezar con un nuevo repositorio para comenzar a registrar de nuevo los cambios. Aquí está el proceso completo como en una sesión de terminal:

```
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ rm -rf .git
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
fatal: no es un repositorio git (ni ninguno de los directorios superiores): .git
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git init
Inicializado repositorio Git vacío en /home/usuario/Escritorio/git_practice/.git/
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
        .gitignore
        hello_git.py

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git add .
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git commit -m "Starting over."
[master (commit-raíz) ce8577d] Starting over.
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello_git.py
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
usuario@usuario-ZenBook-UX431FAC-UX431FA:~/Escritorio/git_practice$
```

Primero comprobamos el estado y vemos que tenemos un árbol de trabajo limpio. Entonces utilizamos el comando *rm -rf* para eliminar el directorio *.git* (*rmdir /s .git en Windows*). Cuando comprobamos el estado después de eliminar la carpeta *.git*, nos dice que eso no es un repositorio Git. Toda la información que Git utiliza para grabar un repositorio es almacenada en la carpeta .git, así que eliminándola, borramos todo el repositorio.
Entonces somos libres de usar git init para iniciar un nuevo repositorio. Comprobamos el estado que muestra que hemos vuelto al escenario inicial esperando el nuevo commit. Comprobando el estado ahora nos muestra que estamos en la rama máster sin ningún commit.
Usar el control de versiones requiere un poco de práctica, pero una vez que empiezas a usarlo, nunca querrás trabajar sin él de nuevo.