# D Utilizar Git para el control de versiones



El software de control de versiones te permite tomar instantáneas de un proyecto siempre que esté en estado de trabajo. Cuando realizas cambios en un proyecto -por ejemplo, cuando implementas una nueva función- puedes volver a un estado de trabajo anterior si el estado actual del proyecto no funciona bien.

Utilizar un software de control de versiones te da libertad para trabajar en mejoras y cometer errores sin preocuparte de arruinar el proyecto. Esto es especialmente crítico en proyectos grandes, pero también puede ser útil en proyectos más pequeños, incluso cuando trabajas con programas contenidos en un único archivo.

En este apéndice, aprenderás a instalar Git y a utilizarlo para el control de versiones en los programas en los que estés trabajando ahora. *Git* es el software de control de versiones más popular que se utiliza hoy en día. Muchas de sus herramientas avanzadas ayudan a los equipos a colaborar en grandes proyectos, pero sus funciones más básicas también funcionan bien para desarrolladores en solitario. Git implementa el control de versiones mediante el seguimiento de los cambios realizados en cada archivo de un proyecto; si cometes un error, sólo tienes que volver a un estado previamente guardado.

## Instalación de Git

Git funciona en todos los sistemas operativos, pero hay diferentes enfoques para instalarlo en cada sistema. Las siguientes secciones proporcionan instrucciones específicas para cada sistema operativo.

Git está incluido por defecto en algunos sistemas, y a menudo se incluye con otros paquetes que puede que ya tengas instalados. Antes de intentar instalar Git, comprueba si ya está en tu sistema. Abre una nueva ventana de terminal y ejecuta el comando git --version. Si aparece un número de versión específico, Git está instalado en tu sistema. Si ves un mensaje pidiéndote que instales o actualices Git, sigue las instrucciones en pantalla.

Si no ves ninguna instrucción en pantalla y utilizas Windows o macOS, puedes descargar un instalador de [https://git-scm.com.](https://git-scm.com) Si eres usuario de Linux con un sistema compatible con apt, puedes instalar Git con el comando sudo apt install git.

### Configurar Git

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para ello, Git necesita saber tu nombre de usuario y tu correo electrónico. Debes proporcionar un nombre de usuario, pero puedes inventarte una dirección de correo electrónico falsa:

$ git config --global user.name "username"  
$ git config --global user.email "username@example.com"

Si olvidas este paso, Git te pedirá esta información cuando hagas tu primera confirmación.

También es mejor establecer el nombre por defecto para la rama principal de cada proyecto. Un buen nombre para esta rama es main:

$ git config --global init.defaultBranch main

Esta configuración significa que cada nuevo proyecto que gestiones con Git empezará con una única rama de confirmaciones llamada *main*.

## Crear un proyecto

Vamos a crear un proyecto con el que trabajar. Crea una carpeta en algún lugar de tu sistema llamada *git\_practice*. Dentro de la carpeta, crea un sencillo programa Python:

**hello\_git.py**

print("Hello Git world!")

Utilizaremos este programa para explorar la funcionalidad básica de Git.

## Ignorar archivos

Los archivos con la extensión *.pyc* se generan automáticamente a partir de los archivos *.py*, por lo que no necesitamos que Git haga un seguimiento de ellos. Estos archivos se almacenan en un directorio llamado *\_\_pycache\_\_*. Para decirle a Git que ignore este directorio, crea un archivo especial llamado *.gitignore*-con un punto al principio del nombre del archivo y sin extensión- y añádele la siguiente línea:

**.gitignore**

\_\_pycache\_\_/

Este archivo le dice a Git que ignore cualquier archivo del directorio *\_\_pycache\_\_*. Utilizar un archivo *.gitignore* mantendrá tu proyecto libre de desorden y facilitará el trabajo.

Puede que tengas que modificar la configuración de tu explorador de archivos para que se muestren los archivos ocultos (archivos cuyos nombres empiezan por un punto). En el Explorador de Windows, marca la casilla del menú Ver denominada **Elementos** ocultos. En macOS, pulsa ⌘-SHIFT-. (punto). En Linux, busca la opción Mostrar archivos ocultos.

## Nota

Si estás en macOS, añade una línea más a *.gitignore*. Añade el nombre .DS\_Store; son archivos ocultos que contienen información sobre cada directorio en macOS, y desordenarán tu proyecto si no los añades a *.gitignore*.

## Inicializar un repositorio

Ahora que tienes un directorio que contiene un archivo Python y un archivo *.gitignore*, puedes inicializar un repositorio Git. Abre un terminal, navega hasta la carpeta *git\_practice* y ejecuta el siguiente comando:

git\_practice$ git init  
Initialized empty Git repository in git\_practice/.git/  
git\_practice$

La salida muestra que Git ha inicializado un repositorio vacío en *git\_practice*. Un *repository* es el conjunto de archivos de un programa que Git está rastreando activamente. Todos los archivos que Git utiliza para gestionar el repositorio se encuentran en el directorio oculto *.git*, con el que no necesitarás trabajar en absoluto. Simplemente no borres ese directorio, o perderás el historial de tu proyecto.

## Comprobar el estado

Antes de hacer nada más, echemos un vistazo al estado del proyecto:

git\_practice$ git status  
❶ On branch main  
No commits yet  
  
❷ Untracked files:  
 (use "git add <file>..." to include in what will be committed)  
 .gitignore  
 hello\_git.py  
  
❸ nothing added to commit but untracked files present (use "git add" to track)  
git\_practice$

En Git, una *branch* es una versión del proyecto en la que estás trabajando; aquí puedes ver que estamos en una rama llamada main ❶. Cada vez que compruebes el estado de tu proyecto, debería mostrar que estás en la rama main. Entonces verás que estamos a punto de hacer la confirmación inicial. Un *commit* es una instantánea del proyecto en un momento determinado.

Git nos informa de que hay archivos sin rastrear en el proyecto ❷, porque aún no le hemos dicho qué archivos rastrear. Luego nos dice que no hay nada añadido a la confirmación actual, pero que hay archivos sin rastrear que quizá queramos añadir al repositorio ❸.

## Añadir archivos al repositorio

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

❶ git\_practice$ git add .  
❷ git\_practice$ git status  
On branch main  
No commits yet  
  
Changes to be committed:  
 (use "git rm --cached <file>..." to unstage)  
❸ new file: .gitignore  
 new file: hello\_git.py  
  
git\_practice$

El comando git add . añade al repositorio todos los archivos de un proyecto que aún no estén siendo rastreados ❶, siempre que no estén listados en *.gitignore*. No confirma los archivos; sólo le dice a Git que empiece a prestarles atención. Cuando comprobamos el estado del proyecto ahora, podemos ver que Git reconoce algunos cambios que necesitan ser confirmados ❷. La etiqueta *new file* significa que estos archivos se han añadido recientemente al repositorio ❸.

## Hacer una confirmación

Hagamos el primer commit:

❶ git\_practice$ git commit -m "Started project."  
❷ [main (root-commit) cea13dd] Started project.  
❸ 2 files changed, 5 insertions(+)  
 create mode 100644 .gitignore  
 create mode 100644 hello\_git.py  
❹ git\_practice$ git status  
On branch main  
nothing to commit, working tree clean  
git\_practice$

Emitimos el comando git commit -m "mensaje" ❶ para hacer una instantánea del proyecto. La bandera -m indica a Git que registre el mensaje que sigue (Started  proyecto.) en el registro del proyecto. La salida muestra que estamos en la rama main ❷ y que han cambiado dos archivos ❸.

Si ahora comprobamos el estado, veremos que estamos en la rama main y que tenemos un árbol de trabajo limpio ❹. Este es el mensaje que deberías ver cada vez que confirmes un estado de trabajo de tu proyecto. Si recibes un mensaje diferente, léelo atentamente; es probable que hayas olvidado añadir un archivo antes de hacer un commit.

## Comprobación del registro

Git mantiene un registro de todas las confirmaciones realizadas en el proyecto. Vamos a comprobar el registro:

git\_practice$ git log  
commit cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main)  
Author: eric <eric@example.com>  
Date: Mon Jun 6 19:37:26 2022 -0800  
  
 Started project.  
git\_practice$

Cada vez que realizas una confirmación, Git genera un ID de referencia único de 40 caracteres. Registra quién hizo la confirmación, cuándo se hizo y el mensaje registrado. No siempre necesitarás toda esta información, por lo que Git proporciona una opción para imprimir una versión más simple de las entradas del registro:

git\_practice$ git log --pretty=oneline  
cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Started project.  
git\_practice$

La bandera --pretty=oneline proporciona las dos piezas de información más importantes: el ID de referencia de la confirmación y el mensaje registrado para la confirmación.

## La segunda confirmación

Para ver el verdadero poder del control de versiones, necesitamos hacer un cambio en el proyecto y confirmar ese cambio. Aquí sólo añadiremos otra línea a *hello\_git.py*:

**hello\_git.py**

print("Hello Git world!")  
print("Hello everyone.")

Cuando comprobemos el estado del proyecto, veremos que Git se ha dado cuenta del archivo que ha cambiado:

git\_practice$ git status  
❶ On branch main  
Changes not staged for commit:  
 (use "git add <file>..." to update what will be committed)  
 (use "git restore <file>..." to discard changes in working directory)  
  
❷ modified: hello\_git.py  
  
❸ no changes added to commit (use "git add" and/or "git commit -a")  
git\_practice$

Vemos la rama en la que estamos trabajando ❶, el nombre del archivo que se modificó ❷, y que no se ha confirmado ningún cambio ❸. Confirmemos el cambio y comprobemos de nuevo el estado:

❶ git\_practice$ git commit -am "Extended greeting."  
[main 945fa13] Extended greeting.  
 1 file changed, 1 insertion(+), 1 deletion(-)  
❷ git\_practice$ git status  
On branch main  
nothing to commit, working tree clean  
❸ git\_practice$ git log --pretty=oneline  
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.  
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.  
git\_practice$

Hacemos un nuevo commit, pasando la bandera -am cuando utilicemos el comando git commit ❶. La bandera -a indica a Git que añada todos los archivos modificados en el repositorio a la confirmación actual. (Si creas algún archivo nuevo entre confirmaciones, vuelve a emitir el comando git add . para incluir los archivos nuevos en el repositorio). La bandera -m indica a Git que registre un mensaje en el registro para esta confirmación.

Cuando comprobamos el estado del proyecto, vemos que volvemos a tener un árbol de trabajo limpio ❷. Por último, vemos las dos confirmaciones en el registro ❸.

## Abandonar cambios

Veamos ahora cómo abandonar un cambio y volver al estado de trabajo anterior. Primero, añade una nueva línea a *hello\_git.py*:

**hello\_git.py**

print("Hello Git world!")  
print("Hello everyone.")  
  
print("Oh no, I broke the project!")

Guarda y ejecuta este archivo.

Comprobamos el estado y vemos que Git se da cuenta de este cambio:

git\_practice$ git status  
On branch main  
Changes not staged for commit:  
 (use "git add <file>..." to update what will be committed)  
 (use "git restore <file>..." to discard changes in working directory)  
  
❶ modified: hello\_git.py  
  
no changes added to commit (use "git add" and/or "git commit -a")  
git\_practice$

Git ve que hemos modificado *hello\_git.py* ❶, y podemos confirmar el cambio si queremos. Pero esta vez, en lugar de confirmar el cambio, volveremos a la última confirmación, cuando sabíamos que nuestro proyecto funcionaba. No haremos nada en *hello\_git.py*: no borraremos la línea ni utilizaremos la función Deshacer del editor de texto. En su lugar, introduce los siguientes comandos en tu sesión de terminal:

git\_practice$ git restore .  
git\_practice$ git status  
On branch main  
nothing to commit, working tree clean  
git\_practice$

El comando git restore nombrearchivo te permite abandonar todos los cambios desde la última confirmación en un archivo específico. El comando git restore . abandona todos los cambios realizados en todos los archivos desde la última confirmación; esta acción restaura el proyecto al último estado confirmado.

Cuando vuelvas a tu editor de texto, verás que *hello\_git.py* ha vuelto a este estado:

print("Hello Git world!")  
print("Hello everyone.")

Aunque volver a un estado anterior pueda parecer trivial en este sencillo proyecto, si estuviéramos trabajando en un proyecto grande con docenas de archivos modificados, se restaurarían todos los archivos que hubieran cambiado desde el último commit. Esta función es increíblemente útil: puedes hacer tantos cambios como quieras al implementar una nueva función, y si no funcionan, puedes descartarlos sin afectar al proyecto. No tienes que recordar esos cambios y deshacerlos manualmente. Git hace todo eso por ti.

## Nota

Puede que tengas que actualizar el archivo en tu editor para ver la versión restaurada.

## Revisar confirmaciones anteriores

Puedes revisar cualquier confirmación en tu registro, utilizando el comando checkout, utilizando los seis primeros caracteres de un ID de referencia. Tras comprobar y revisar una confirmación anterior, puedes volver a la última confirmación o abandonar tu trabajo reciente y retomar el desarrollo a partir de la confirmación anterior:

git\_practice$ git log --pretty=oneline  
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.  
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.  
git\_practice$ git checkout cea13d  
Note: switching to 'cea13d'.  
  
❶ You are in 'detached HEAD' state. You can look around, make experimental  
changes and commit them, and you can discard any commits you make in this  
state without impacting any branches by switching back to a branch.  
  
If you want to create a new branch to retain commits you create, you may  
do so (now or later) by using -c with the switch command. Example:  
  
 git switch -c <new-branch-name>  
  
❷ Or undo this operation with:  
  
 git switch -  
  
Turn off this advice by setting config variable advice.detachedHead to false  
  
HEAD is now at cea13d Started project.  
git\_practice$

Cuando revisas una confirmación anterior, abandonas la rama principal y entras en lo que Git denomina estado *detached HEAD* ❶. *HEAD* es el estado comprometido actual del proyecto; estás en *detached* porque has abandonado una rama con nombre (main, en este caso).

Para volver a la rama main, sigue la sugerencia ❷ deshacer la operación anterior:

git\_practice$ git switch -  
Previous HEAD position was cea13d Started project.  
Switched to branch 'main'  
git\_practice$

Este comando te devuelve a la rama main. A menos que quieras trabajar con algunas funciones más avanzadas de Git, es mejor que no hagas ningún cambio en tu proyecto cuando hayas comprobado una confirmación anterior. Sin embargo, si eres el único que trabaja en un proyecto y quieres descartar todas las confirmaciones más recientes y volver a un estado anterior, puedes restablecer el proyecto a una confirmación anterior. Trabajando desde la rama main, introduce lo siguiente:

❶ git\_practice$ git status  
On branch main  
nothing to commit, working directory clean  
❷ git\_practice$ git log --pretty=oneline  
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.  
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.  
❸ git\_practice$ git reset --hard cea13d  
HEAD is now at cea13dd Started project.  
❹ git\_practice$ git status  
On branch main  
nothing to commit, working directory clean  
❺ git\_practice$ git log --pretty=oneline  
cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Started project.  
git\_practice$

Primero comprobamos el estado para asegurarnos de que estamos en la rama main ❶. Cuando miramos el registro, vemos ambos commits ❷. A continuación, emitimos el comando git reset --hard con los seis primeros caracteres del ID de referencia de , el commit al que queremos volver permanentemente ❸. Comprobamos de nuevo el estado y vemos que estamos en la rama main sin nada que confirmar ❹. Cuando volvemos a mirar el registro, vemos que estamos en la confirmación desde la que queríamos volver a empezar ❺.

## Borrar el repositorio

A veces estropearás el historial de tu repositorio y no sabrás cómo recuperarlo. Si esto ocurre, primero considera la posibilidad de pedir ayuda utilizando los métodos comentados en el Apéndice C. Si no puedes arreglarlo y estás trabajando en un proyecto en solitario, puedes seguir trabajando con los archivos pero deshacerte del historial del proyecto borrando el directorio *.git*. Esto no afectará al estado actual de ninguno de los archivos, pero borrará todos los commits, por lo que no podrás consultar ningún otro estado del proyecto.

Para ello, abre un explorador de archivos y borra el repositorio *.git* o bórralo desde la línea de comandos. Después, tendrás que empezar de nuevo con un repositorio nuevo para empezar a seguir tus cambios de nuevo. Este es el aspecto de todo este proceso en una sesión de terminal:

❶ git\_practice$ git status  
On branch main  
nothing to commit, working directory clean  
❷ git\_practice$ rm -rf .git/  
❸ git\_practice$ git status  
fatal: Not a git repository (or any of the parent directories): .git  
❹ git\_practice$ git init  
Initialized empty Git repository in git\_practice/.git/  
❺ git\_practice$ git status  
On branch main  
No commits yet  
  
Untracked files:  
 (use "git add <file>..." to include in what will be committed)  
 .gitignore  
 hello\_git.py  
  
nothing added to commit but untracked files present (use "git add" to track)  
❻ git\_practice$ git add .  
git\_practice$ git commit -m "Starting over."  
[main (root-commit) 14ed9db] Starting over.  
 2 files changed, 5 insertions(+)  
 create mode 100644 .gitignore  
 create mode 100644 hello\_git.py  
❼ git\_practice$ git status  
On branch main  
nothing to commit, working tree clean  
git\_practice$

Primero comprobamos el estado y vemos que tenemos un directorio de trabajo limpio ❶. A continuación, utilizamos el comando rm -rf .git/ para borrar el directorio *.git* (del .git en Windows) ❷. Cuando comprobamos el estado después de borrar la carpeta *.git*  , nos dice que no es un repositorio Git ❸. Toda la información que Git utiliza para rastrear un repositorio se almacena en la carpeta *.git*, por lo que al eliminarla se elimina todo el repositorio.

Entonces podemos utilizar git init para iniciar un repositorio nuevo ❹. Comprobar el estado muestra que estamos de nuevo en la fase inicial, a la espera de la primera confirmación ❺. Añadimos los archivos y hacemos la primera confirmación ❻. Comprobar el estado nos muestra ahora que estamos en la nueva rama main sin nada que confirmar ❼.

Utilizar el control de versiones requiere un poco de práctica, pero una vez que empieces a usarlo, nunca querrás volver a trabajar sin él.