

DAM

# Contornos de desenvolvimento

## *UD2* **INSTALACIÓN Y USO DE ENTORNOS DE DESARROLLO: FUNDAMENTOS DE GIT**

PROFESORA | CRISTINA PUGA FDEZ

UNIDADE | UD2

TRIMESTRE | 1

MÓDULO | CD

## Tabla de contenido

1.	Introducción a Git.....	3
2.	Configuración de Git.....	3
3.	Comprobando configuración.....	3
4.	Inicializando repositorio en local .....	6
5.	Git status .....	10
6.	Git add .....	10
7.	Git commit.....	11
8.	Archivos ignorados .....	12
9.	Git log .....	16
10.	Git checkout .....	16
11.	Archivos modificados .....	19
12.	Estado abreviado .....	22
13.	Patrones para ignorar ficheros.....	23
14.	Ver cambios no preparados .....	24
15.	Eliminar archivos .....	28
16.	Cambiar el nombre archivos .....	30
17.	Opciones del historial de confirmaciones .....	30
18.	Deshacer.....	30
19.	Trabajar con remotos .....	32
	ANEXO I: Materiales .....	33

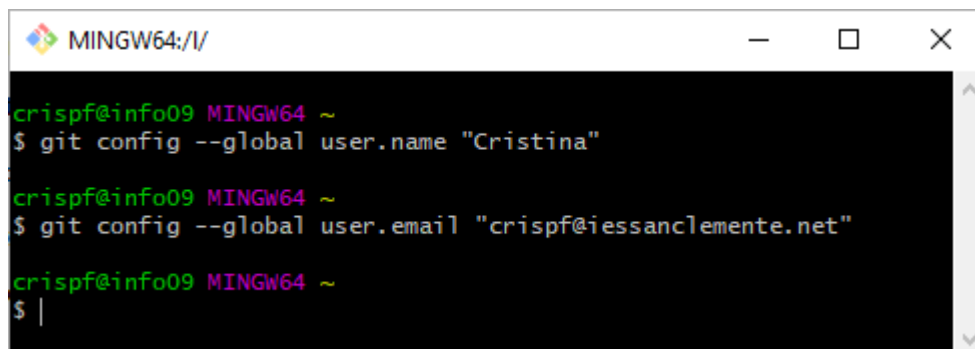
## 1. Introducción a Git

Git es un sistema que permite realizar el control de versiones. Esta funcionalidad permite registrar los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un sistema de control de versiones (VCS) también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente.

## 2. Configuración de Git

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

A screenshot of a terminal window titled 'MINGW64:/I/'. The prompt is 'crispf@info09 MINGW64 ~'. The user enters the command '\$ git config --global user.name "Cristina"'. The prompt changes to 'crispf@info09 MINGW64 ~' and the user enters '\$ git config --global user.email "crispf@iessanclemente.net"'. The prompt changes to 'crispf@info09 MINGW64 ~' and the user enters '\$ |'.

```
crispf@info09 MINGW64 ~
$ git config --global user.name "Cristina"

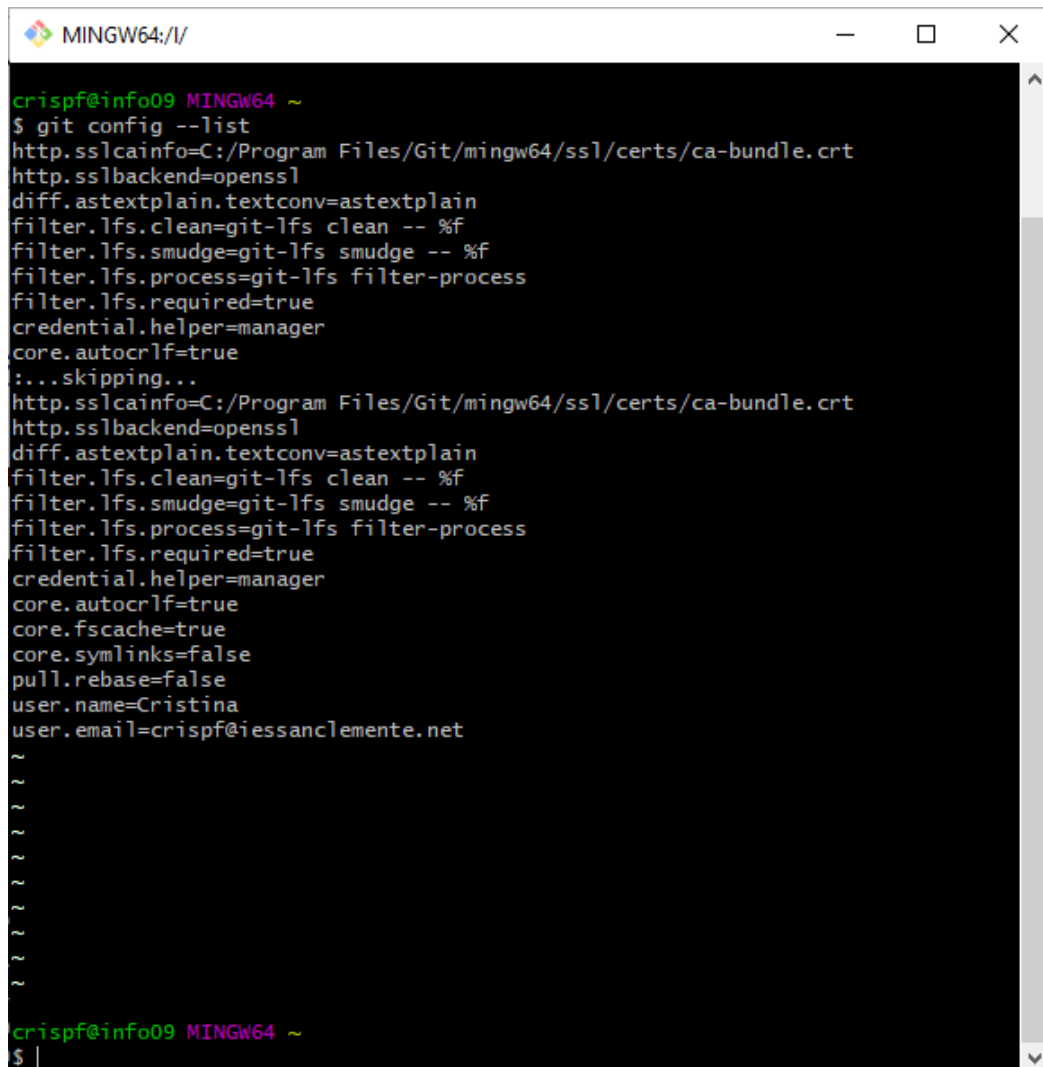
crispf@info09 MINGW64 ~
$ git config --global user.email "crispf@iessanclemente.net"

crispf@info09 MINGW64 ~
$ |
```

Sólo necesitas hacer esto una vez si especificas la opción `--global`, ya que Git siempre usará esta información para todo lo que hagas en ese sistema. Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando sin la opción `--global` cuando estés en ese proyecto.

## 3. Comprobando configuración

Si quieres comprobar tu configuración, puedes usar el comando **git config --list** para mostrar todas las propiedades que Git ha configurado:



```
crispf@info09 MINGW64 ~
$ git config --list
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
core.autocrlf=true
...skipping...
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
user.name=Cristina
user.email=crispf@iessanclemente.net
~
~
~
~
~
~
~
crispf@info09 MINGW64 ~
$ |
```

También puedes comprobar el valor que Git utilizará para una clave específica ejecutando **git config<key>**



```
crispf@info09 MINGW64 ~
$ git config user.name
Cristina

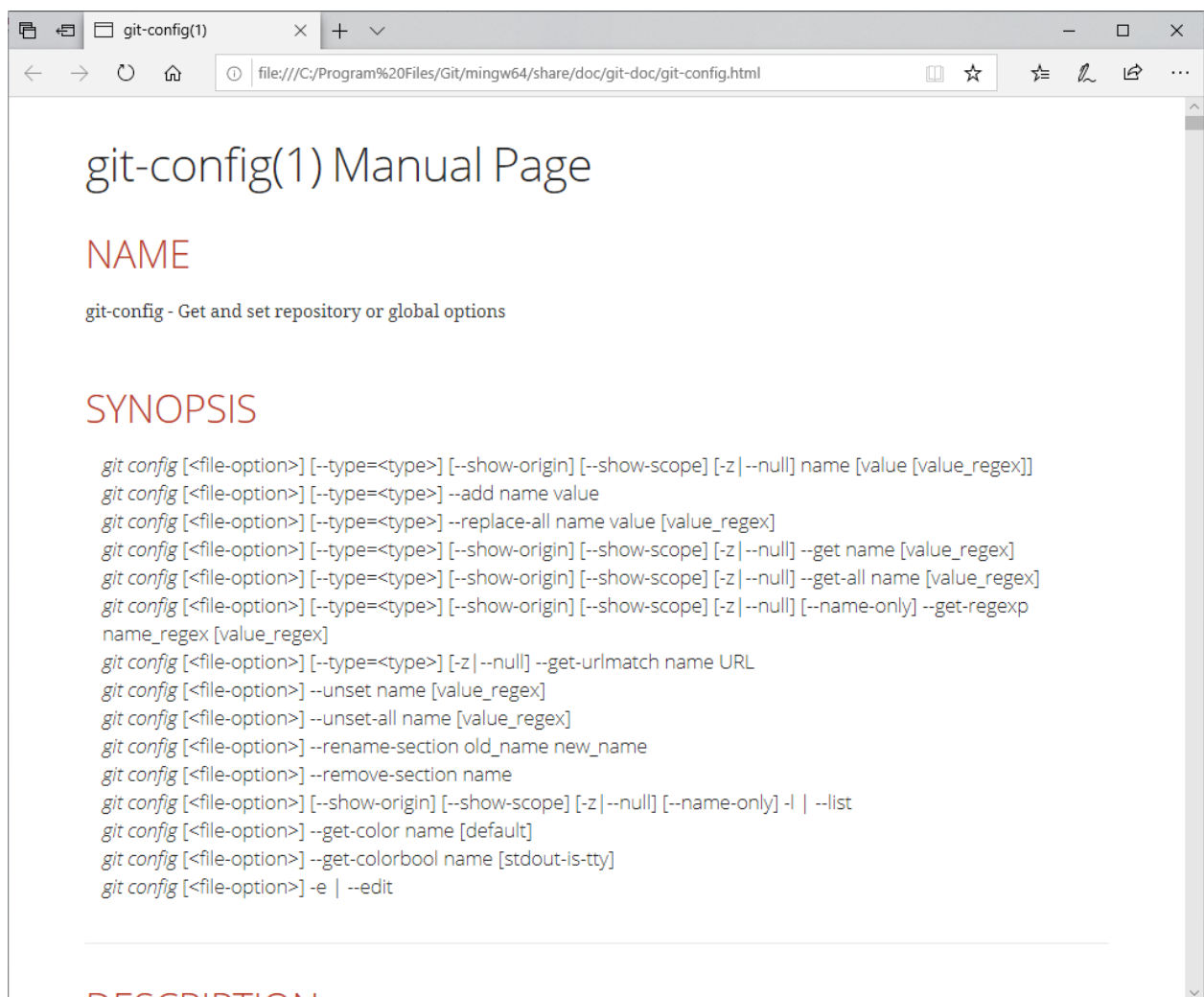
crispf@info09 MINGW64 ~
$
```

Obteniendo ayuda

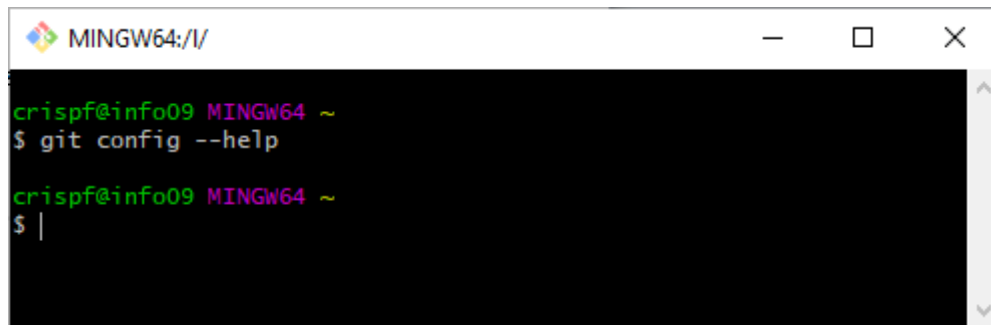
**git help <comando>**



```
crispcf@info09 MINGW64 ~  
$ git help config  
  
crispcf@info09 MINGW64 ~  
$
```



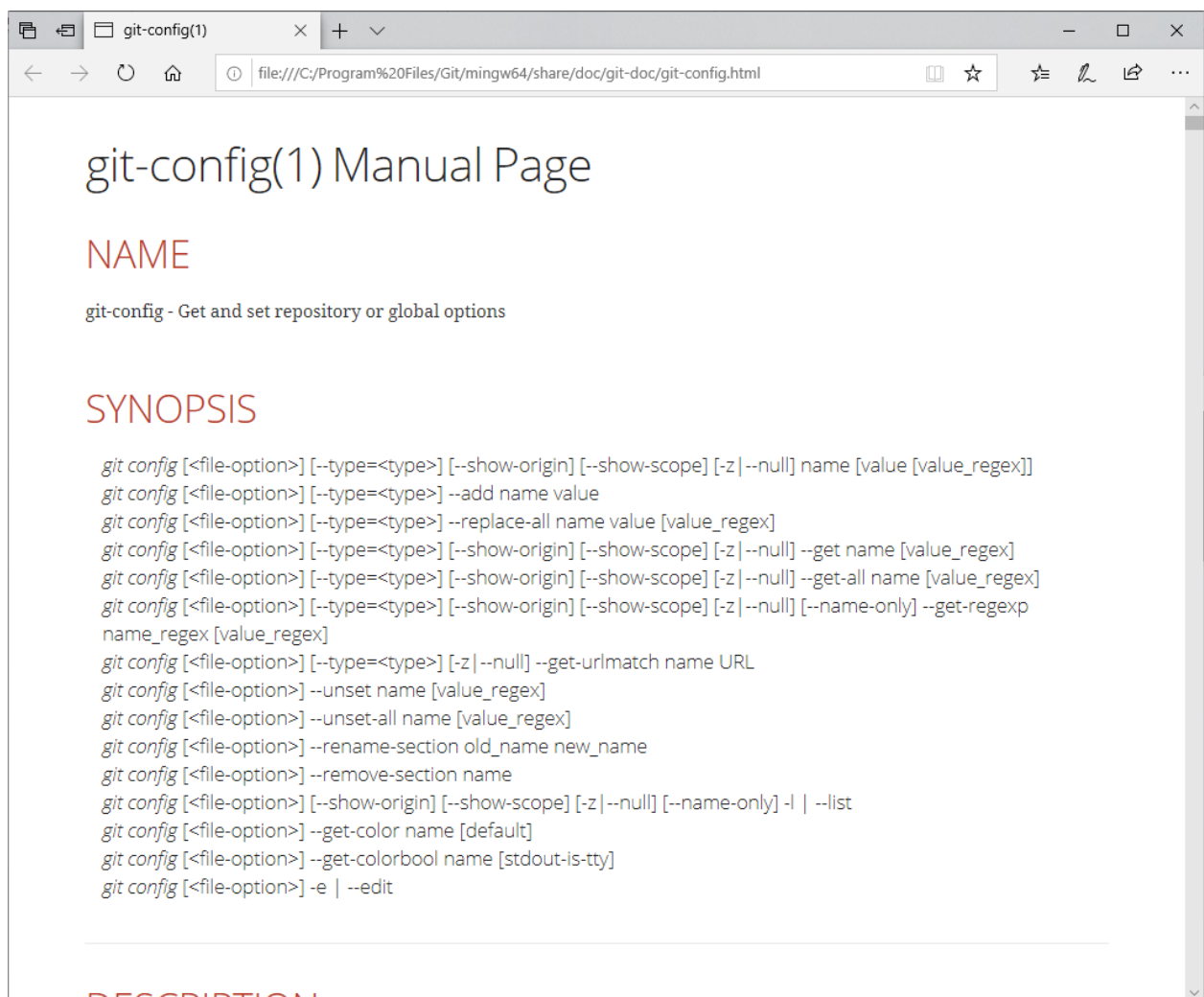
**git <comando> --help**



```
MINGW64:/I/

crispcf@info09 MINGW64 ~
$ git config --help

crispcf@info09 MINGW64 ~
$ |
```

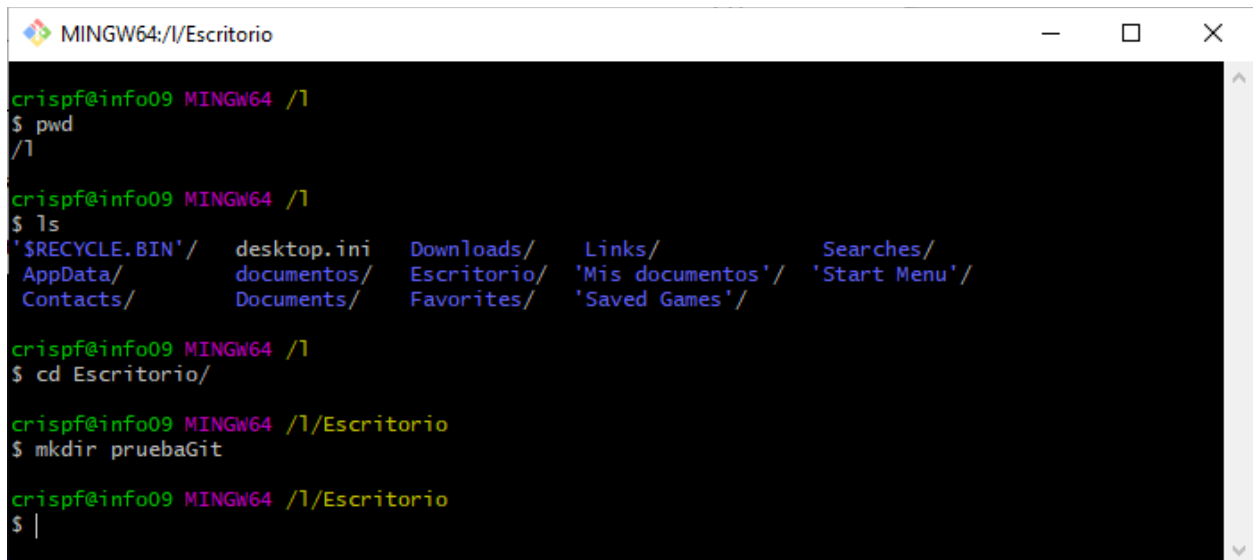


## 4. Inicializando repositorio en local

Para inicializar el repositorio en local realizamos los siguientes comandos:

- Compruebo la ruta en la que estoy: **pwd**

- Listo el contenido de ficheros y directorios de la ubicación en la que estoy: **ls**
- Me muevo hasta el escritorio de mi usuario:
  - **cd ..** : para subir de nivel
  - **cd Escritorio/** : para ubicarme en el directorio “Escritorio”
- Creo una carpeta que va a almacenar mi repositorio: **mkdir XX**



```
crispcf@info09 MINGW64 /1
$ pwd
/1

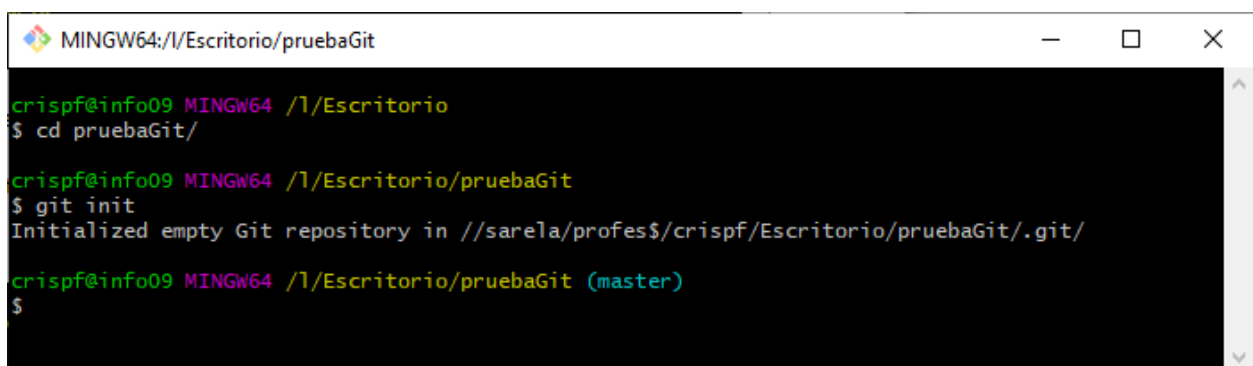
crispcf@info09 MINGW64 /1
$ ls
'$RECYCLE.BIN'/' desktop.ini Downloads/ Links/ Searches/
AppData/ documentos/ Escritorio/ 'Mis documentos/' 'Start Menu/'
Contacts/ Documents/ Favorites/ 'Saved Games/'

crispcf@info09 MINGW64 /1
$ cd Escritorio/

crispcf@info09 MINGW64 /1/Escritorio
$ mkdir pruebaGit

crispcf@info09 MINGW64 /1/Escritorio
$ |
```

- Me muevo a la carpeta que acabamos de crear: **cd pruebaGit**
- Inicializo el repositorio dentro de esa carpeta: **git init**



```
crispcf@info09 MINGW64 /1/Escritorio
$ cd pruebaGit/

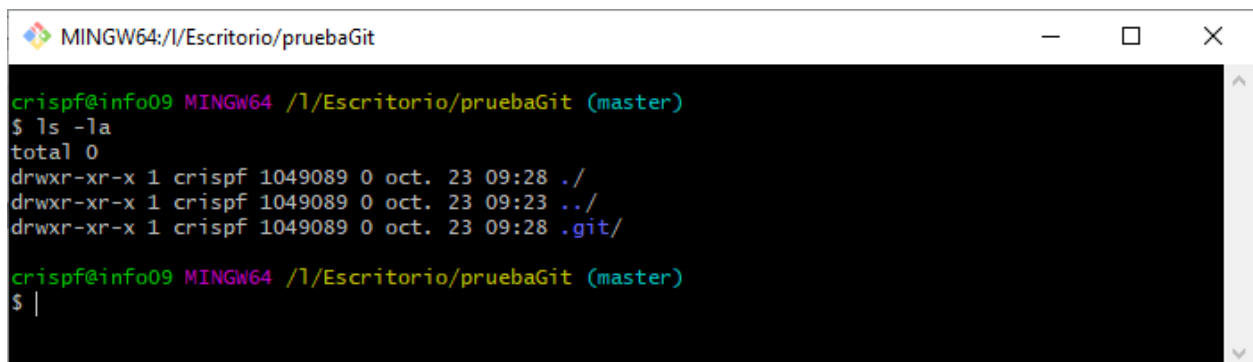
crispcf@info09 MINGW64 /1/Escritorio/pruebaGit
$ git init
Initialized empty Git repository in //sarela/profes$/crispcf/Escritorio/pruebaGit/.git/

crispcf@info09 MINGW64 /1/Escritorio/pruebaGit (master)
$
```

Nos fijamos en dos cosas, la primera que al ejecutar el comando nos dice que se ha inicializado correctamente el repositorio en la ruta elegida.

La segunda que a partir de ahora siempre que naveguemos dentro de nuestra carpeta tendremos al final de la ruta la versión en la que estamos trabajando (master) en este caso.

Si ejecutamos el comando `ls -la` nos muestra los ficheros ocultos:

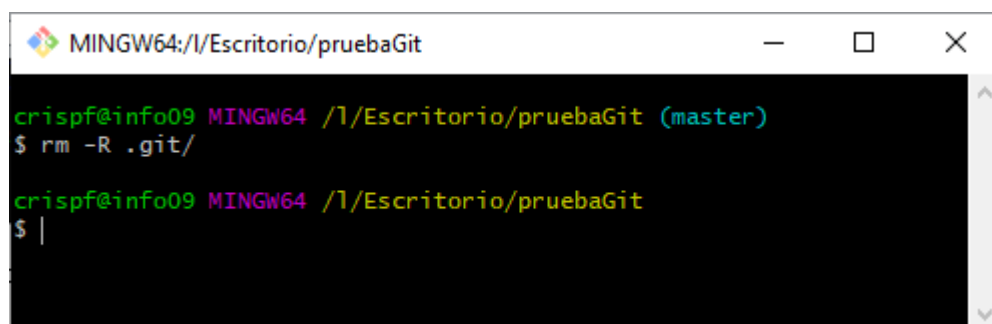


```
crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ ls -la
total 0
drwxr-xr-x 1 crispf 1049089 0 oct. 23 09:28 ./
drwxr-xr-x 1 crispf 1049089 0 oct. 23 09:23 ../
drwxr-xr-x 1 crispf 1049089 0 oct. 23 09:28 .git/

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

Comprobamos que nos crea una carpeta `.git` que será la base del repositorio.

En caso de querer borrar el repositorio, tendremos que ejecutar el comando que elimina esa carpeta: `rm -R .git/`




```
crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ rm -R .git/

crispf@info09 MINGW64 /I/Escritorio/pruebaGit
$ |
```

Una vez eliminada esa carpeta vemos que ya no nos muestra la versión en la que estamos trabajando (master).

Para poder trabajar con nuestro repositorio vamos a inicializarlo de nuevo y a crear dos ficheros con los comandos **nano** o **vim**:

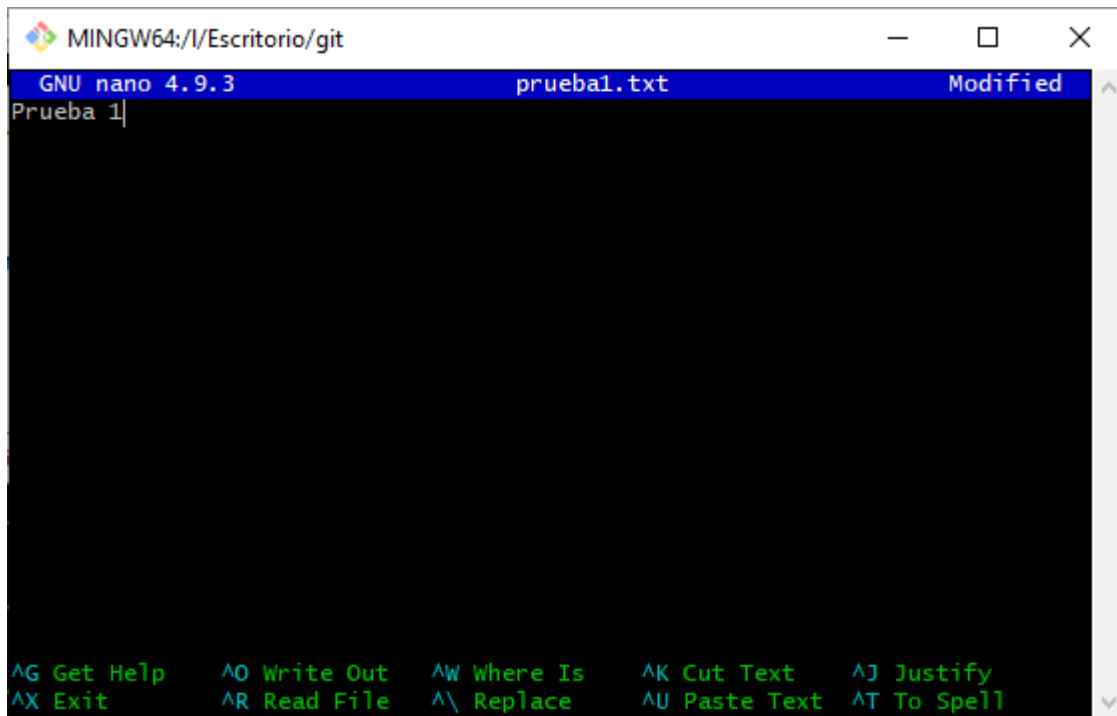


```
crispf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ nano prueba1.txt

crispf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ |
```

Si creamos el fichero con el editor nano, como se muestra en los comandos anteriores, tendremos para prueba1.txt:



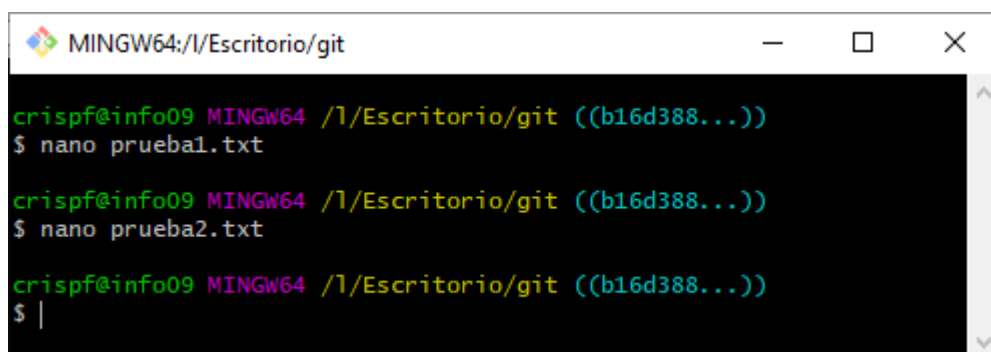


```
MINGW64:/I/Escritorio/git
GNU nano 4.9.3      prueba1.txt      Modified
Prueba 1|

^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify
^X Exit          ^R Read File    ^\ Replace     ^U Paste Text  ^T To Spell
```

Y una vez incluido el contenido, pulsamos **Ctrl+X** para salir, y a continuación pulsamos **Y** para guardar los cambios y pulsamos **Intro**.

Ahora hacemos lo mismo para el fichero prueba2.txt.

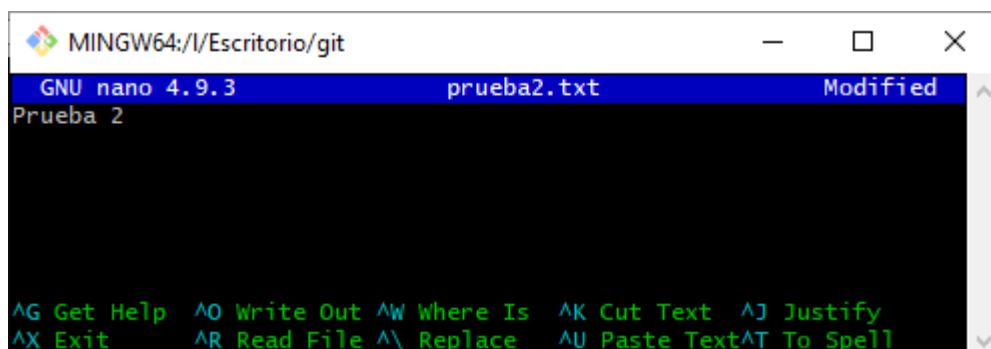


```
MINGW64:/I/Escritorio/git
crispf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ nano prueba1.txt

crispf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ nano prueba2.txt

crispf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ |
```

Editamos el fichero y posteriormente lo guardamos.

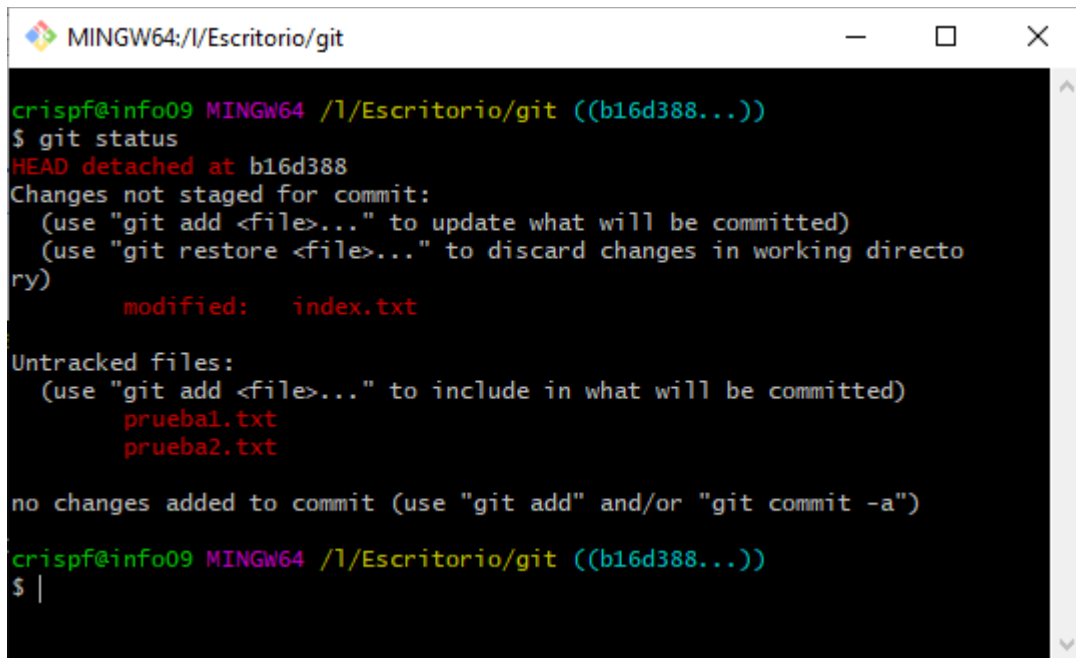


```
MINGW64:/I/Escritorio/git
GNU nano 4.9.3      prueba2.txt      Modified
Prueba 2|

^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify
^X Exit          ^R Read File    ^\ Replace     ^U Paste Text  ^T To Spell
```

## 5. Git status

El comando **git status** es la herramienta principal para determinar qué archivos están en qué estado.



```
crispcf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ git status
HEAD detached at b16d388
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)
        modified:   index.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        prueba1.txt
        prueba2.txt

no changes added to commit (use "git add" and/or "git commit -a")
crispcf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ |
```

Puedes ver que los archivos `prueba1.txt` y `prueba2.txt` están sin rastrear porque aparece debajo del encabezado “Untracked files” (“Archivos no rastreados” en inglés) en la salida.

Sin rastrear significa que Git ve archivos que no tenías en el commit anterior. Git no los incluirá en tu próximo commit a menos que se lo indiques explícitamente. Se comporta así para evitar incluir accidentalmente archivos binarios o cualquier otro archivo que no quieras. Si queremos incluir los dos ficheros en el próximo commit debemos rastrearlos.

## 6. Git add

Para comenzar a rastrear un archivo debes usar el comando **git add**. Podemos ejecutar dos comandos uno por cada archivo. O realizar un único **git add \***, que incluye todo lo que hay en la carpeta.

En nuestro caso, vamos a rastrear todos los archivos en la misma acción:

```
crispcf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$ git add *
warning: LF will be replaced by CRLF in prueba1.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in prueba2.txt.
The file will have its original line endings in your working directory

crispcf@info09 MINGW64 /I/Escritorio/git ((b16d388...))
$
```

Los warnings que nos dan son porque estamos trabajando en una consola Linux dentro del sistema operativo de Windows.

Si ahora ejecutamos un **git status** para ver el estado de los archivos tendremos algo como esto:

```
MINGW64:/I/Escritorio/pruebasGit

crispcf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   prueba1.txt
        new file:   prueba2.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$
```

Puedes ver que está siendo rastreado porque aparece luego del encabezado “Cambios a ser confirmados” (“Changes to be committed” en inglés).

Si confirmas en este punto, se guardará en el historial la versión del archivo correspondiente al instante en que ejecutaste git add (el cual inició el rastreo de archivos en tu directorio).

El comando git add puede recibir tanto una ruta de archivo como de un directorio; si es de un directorio, el comando añade recursivamente los archivos que están dentro de él.

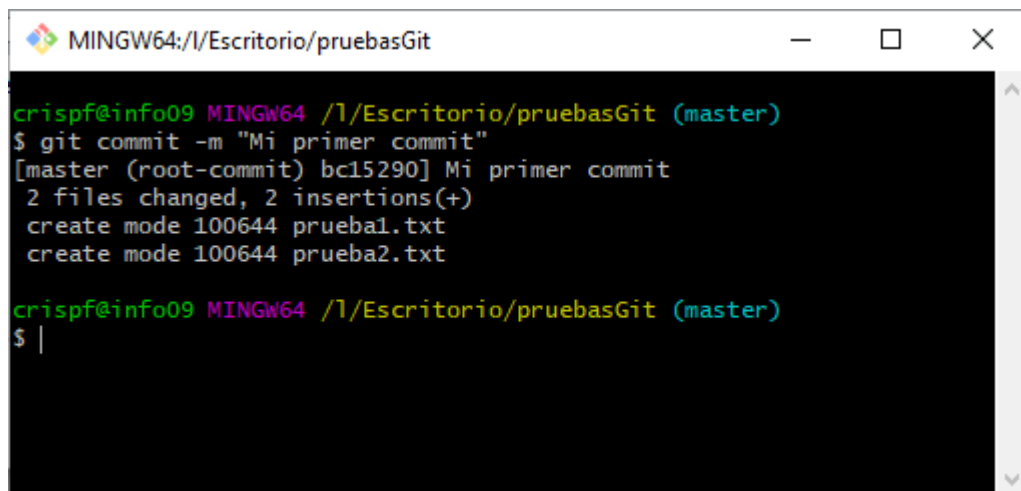
## 7. Git commit

Ahora que tu área de preparación está como quieres, puedes confirmar tus cambios. Recuerda que cualquier cosa que no esté preparada - cualquier archivo que hayas creado o modificado y que no hayas agregado con git add desde su edición - no será confirmado.

Para ello ejecutamos el comando: **git commit -m “descripción cambios”**

A través de **-m** le indicamos el mensaje que veremos después y que nos servirá para saber a qué

hace referencia este commit.

A terminal window titled 'MINGW64:/l/Escritorio/pruebasGit' showing a successful git commit. The user 'crispf@info09' is in the 'master' branch. They run '\$ git commit -m "Mi primer commit"'. The output shows the commit was successful with SHA-1 hash 'bc15290', 2 files changed, and 2 insertions. The files are 'prueba1.txt' and 'prueba2.txt'.

```
crispf@info09 MINGW64 /l/Escritorio/pruebasGit (master)
$ git commit -m "Mi primer commit"
[master (root-commit) bc15290] Mi primer commit
2 files changed, 2 insertions(+)
create mode 100644 prueba1.txt
create mode 100644 prueba2.txt

crispf@info09 MINGW64 /l/Escritorio/pruebasGit (master)
$ |
```

Puedes ver que la confirmación te devuelve una salida descriptiva: indica cuál rama has confirmado (master), que checksum SHA-1 tiene el commit (código alfanumérico), cuántos archivos han cambiado y estadísticas sobre las líneas añadidas y eliminadas en el commit.

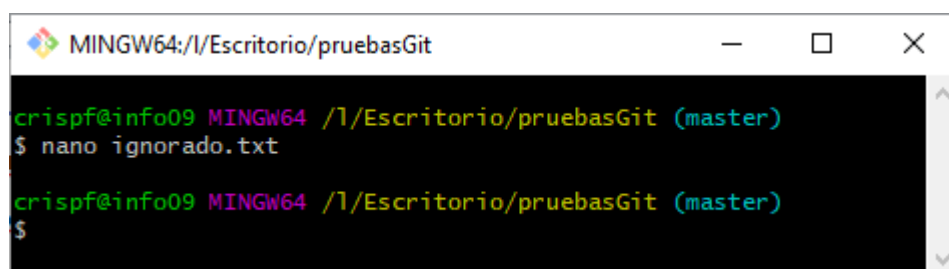
Recuerda que la confirmación guarda una instantánea de tu área de preparación. Todo lo que no hayas preparado sigue allí modificado; puedes hacer una nueva confirmación para añadirlo a tu historial. Cada vez que realizas un commit, guardas una instantánea de tu proyecto la cual puedes usar para comparar o volver a ella luego.

## 8. Archivos ignorados

A veces, tendrás algún tipo de archivo que no quieres que Git añada automáticamente o más aun, que ni siquiera quieras que aparezca como no rastreado. Este suele ser el caso de archivos generados automáticamente como trazas, archivos creados por tu sistema de compilación, etc.

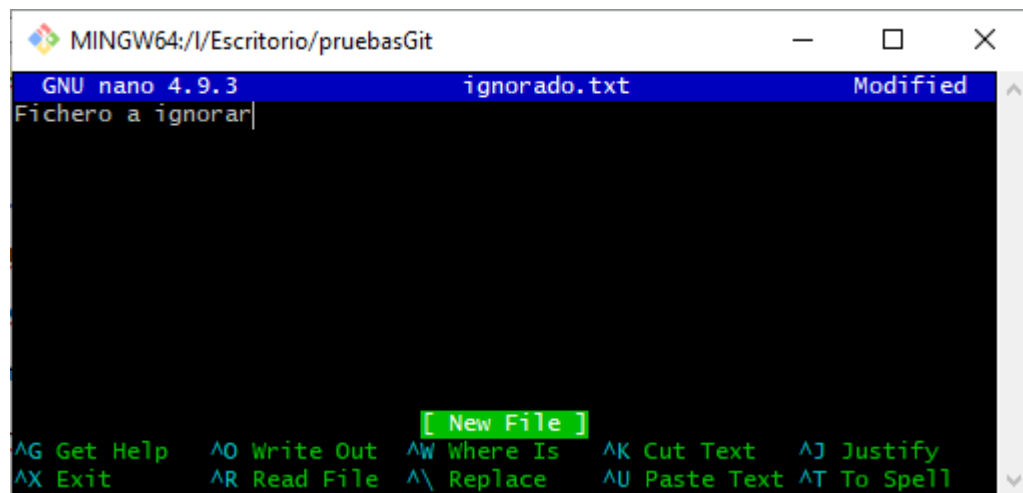
En estos casos, puedes crear un archivo llamado `.gitignore` que liste patrones o archivos a ignorar en el commit.

Para ver esto, vamos a crearnos un nuevo archivo que queremos que sea ignorado: `ignorado.txt`

A terminal window titled 'MINGW64:/l/Escritorio/pruebasGit' showing the user 'crispf@info09' running '\$ nano ignorado.txt' to create a new file. The prompt returns to '\$' after the command.

```
crispf@info09 MINGW64 /l/Escritorio/pruebasGit (master)
$ nano ignorado.txt

crispf@info09 MINGW64 /l/Escritorio/pruebasGit (master)
$
```



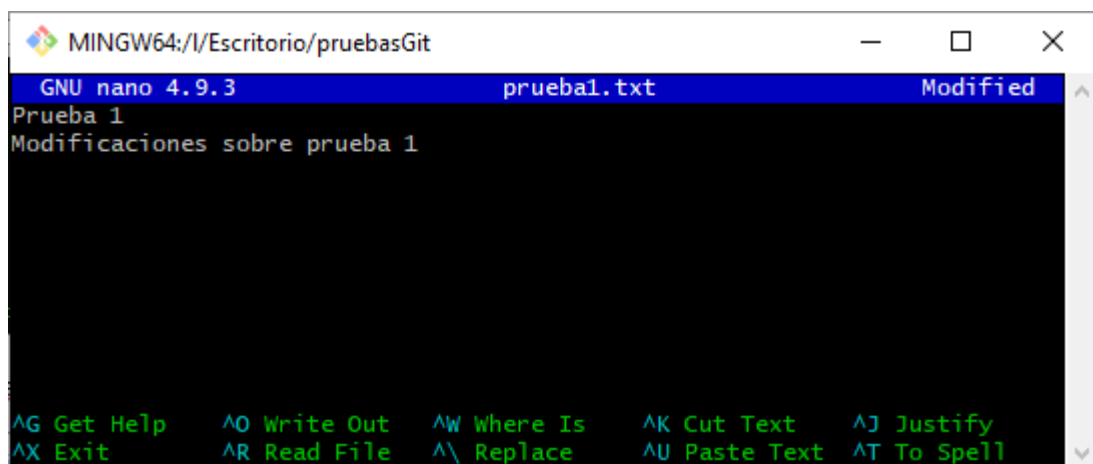
```

MINGW64:/I/Escritorio/pruebasGit
GNU nano 4.9.3      ignorado.txt      Modified
Fichero a ignorar

[ New File ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell

```

Y vamos a modificar prueba1.txt añadiendo una nueva línea:



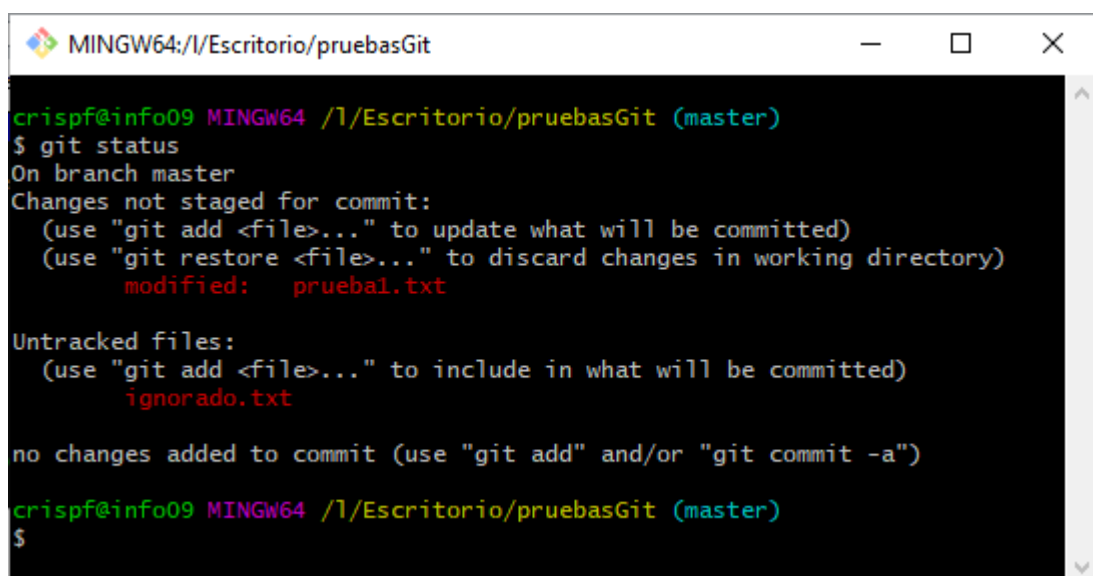
```

MINGW64:/I/Escritorio/pruebasGit
GNU nano 4.9.3      prueba1.txt      Modified
Prueba 1
Modificaciones sobre prueba 1

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell

```

Si ahora hacemos un **git status** comprobamos que tenemos dos archivos, uno modificado (prueba1.txt) y el otro sin realizar ningún seguimiento (ignorado.txt)



```

MINGW64:/I/Escritorio/pruebasGit
crisprf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git status
On branch master
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)
        modified:   prueba1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ignorado.txt

no changes added to commit (use "git add" and/or "git commit -a")

crisprf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$

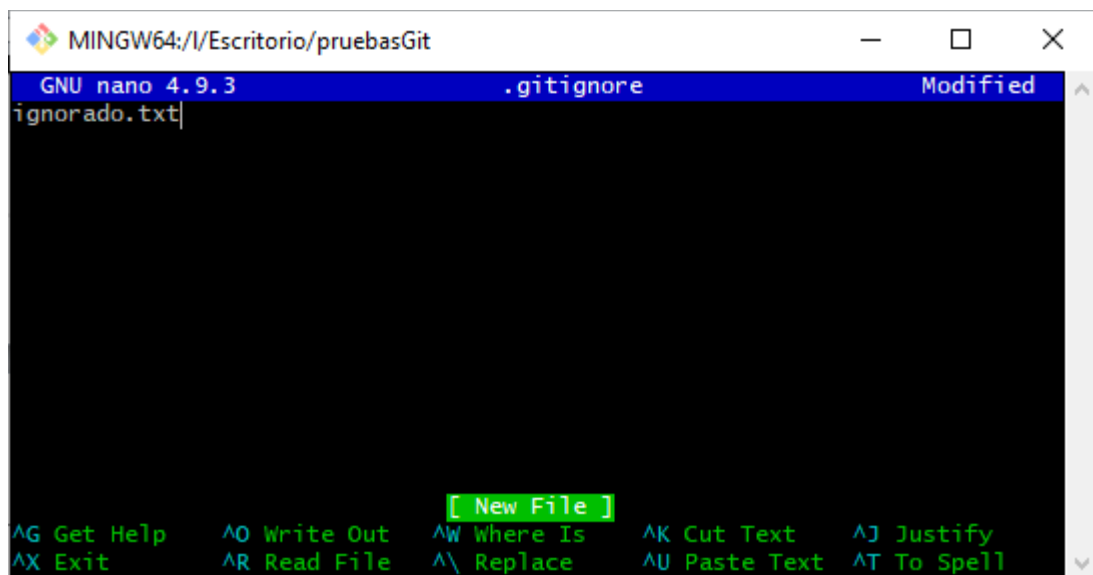
```

Lo que vamos a hacer es crear un fichero `.gitignore`:

```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ nano .gitignore

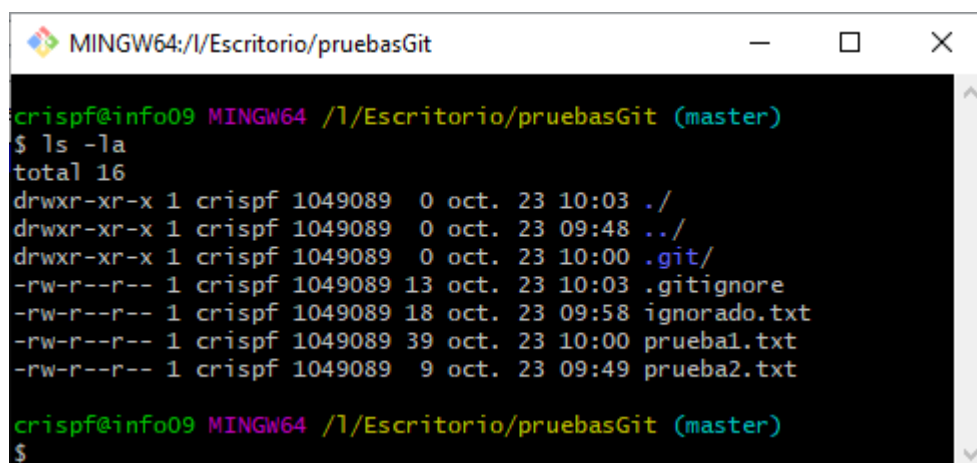
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$
```

Añadimos en el contenido del fichero el nombre de los ficheros que queremos que sean ignorados. En nuestro caso el fichero `ignorado.txt` para que no se realice ningún seguimiento sobre él, ya que no nos interesa.



The screenshot shows a terminal window titled 'MINGW64:/I/Escritorio/pruebasGit' with the GNU nano 4.9.3 editor open to the `.gitignore` file. The text `ignorado.txt` is entered on the first line. The bottom status bar shows various keyboard shortcuts like `^G Get Help`, `^O Write Out`, `^W Where Is`, `^K Cut Text`, `^J Justify`, `^X Exit`, `^R Read File`, `^_ Replace`, `^U Paste Text`, and `^T To Spell`. A `[ New File ]` button is also visible.

Ahora mismo tenemos los siguientes ficheros en nuestra carpeta:



The screenshot shows a terminal window titled 'MINGW64:/I/Escritorio/pruebasGit' with the following output from the `ls -la` command:

```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ ls -la
total 16
drwxr-xr-x 1 crispf 1049089  0 oct. 23 10:03 ./
drwxr-xr-x 1 crispf 1049089  0 oct. 23 09:48 ../
drwxr-xr-x 1 crispf 1049089  0 oct. 23 10:00 .git/
-rw-r--r-- 1 crispf 1049089 13 oct. 23 10:03 .gitignore
-rw-r--r-- 1 crispf 1049089 18 oct. 23 09:58 ignorado.txt
-rw-r--r-- 1 crispf 1049089 39 oct. 23 10:00 prueba1.txt
-rw-r--r-- 1 crispf 1049089  9 oct. 23 09:49 prueba2.txt

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$
```

Si ahora hacemos un `git status` vemos que tenemos el fichero `prueba1.txt` modificado y el `.gitignore` para realizar el seguimiento pero ya no tenemos `ignorado.txt` porque lo hemos añadido al fichero `.gitignore`:

```
MINGW64:/I/Escritorio/pruebasGit

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git status
On branch master
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)
        modified:   prueba1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

no changes added to commit (use "git add" and/or "git commit -a")

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$
```

Vamos a hacer un **git add** sobre esos dos ficheros para marcarlos:

```
MINGW64:/I/Escritorio/pruebasGit

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git add prueba1.txt
warning: LF will be replaced by CRLF in prueba1.txt.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git add .gitignore
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ |
```

En este momento tenemos los dos ficheros rastreados y listos para su confirmación:

```
MINGW64:/I/Escritorio/pruebasGit

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        modified:   prueba1.txt

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ |
```

Realizamos un commit para confirmar los cambios:



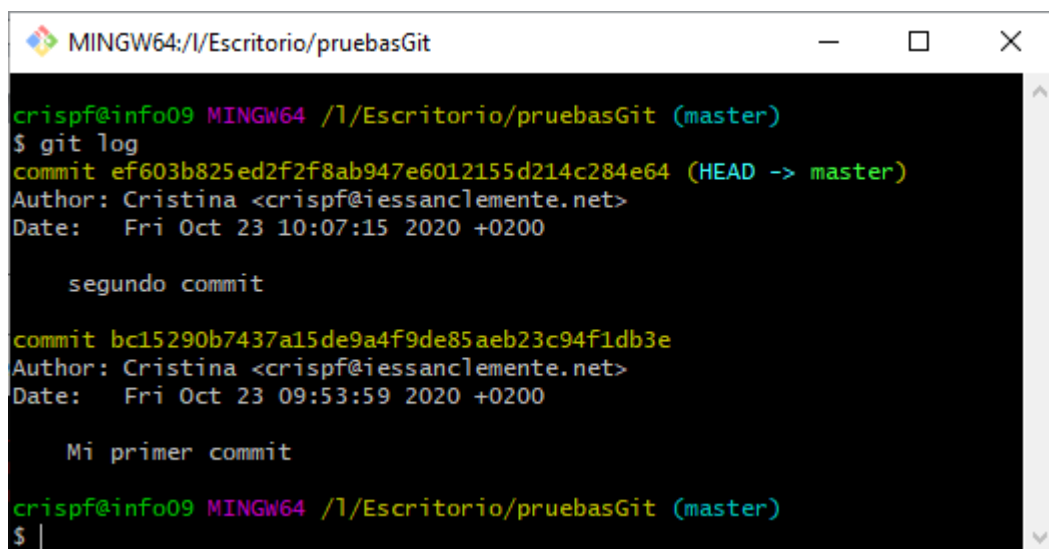
```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git commit -m "segundo commit"
[master ef603b8] segundo commit
2 files changed, 2 insertions(+)
create mode 100644 .gitignore

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ |
```

## 9. Git log

Después de haber hecho varias confirmaciones, probablemente quieras mirar atrás para ver qué modificaciones se han llevado a cabo. La herramienta más básica y potente para hacer esto es el comando **git log**.

Si hacemos un git log en este momento tendremos las siguientes líneas de confirmación:



```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git log
commit ef603b825ed2f2f8ab947e6012155d214c284e64 (HEAD -> master)
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 10:07:15 2020 +0200

    segundo commit

commit bc15290b7437a15de9a4f9de85aeb23c94f1db3e
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 09:53:59 2020 +0200

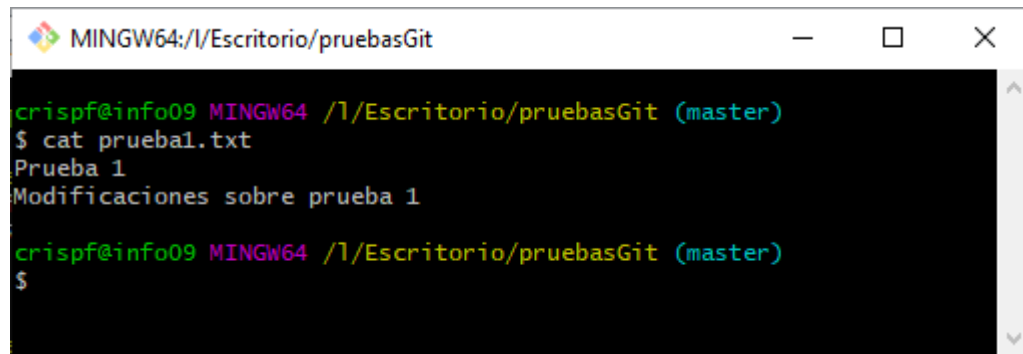
    Mi primer commit

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ |
```

## 10. Git checkout

Con el comando **checkout** podemos volver a una versión anterior de nuestro documento. En este caso vamos a revertir los cambios en prueba1.txt eliminando la línea nueva del último commit. Actualmente el archivo está como sigue (visualizamos el contenido del fichero mediante el comando **cat**):

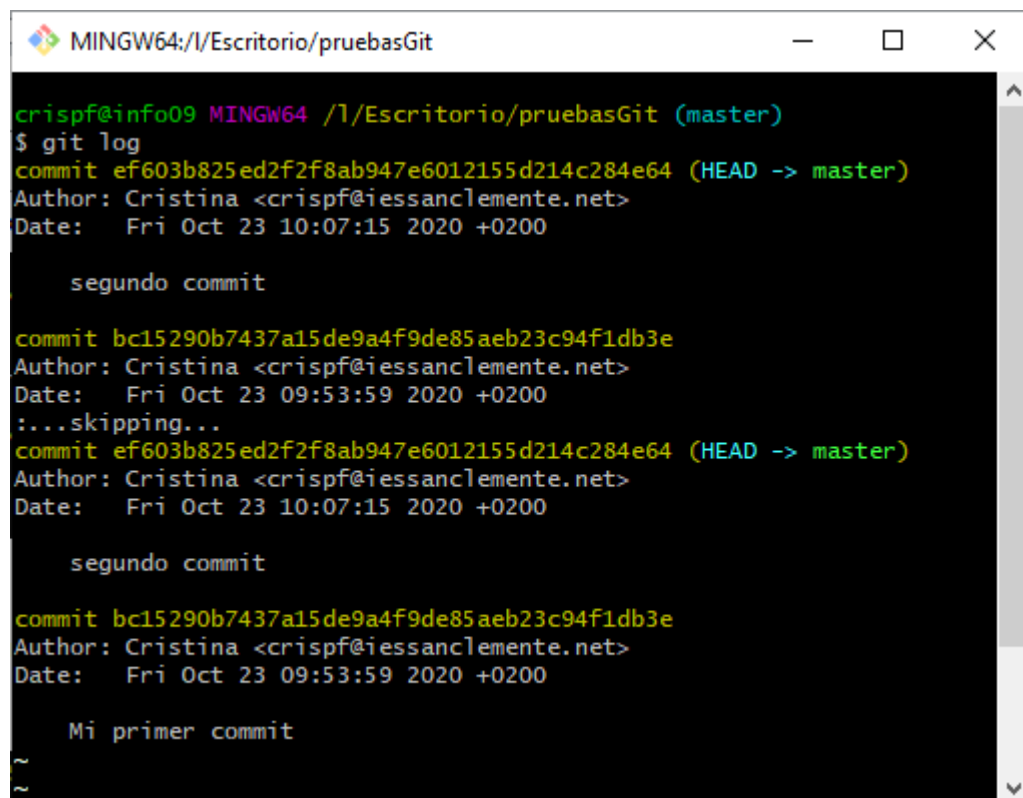




```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ cat prueba1.txt
Prueba 1
Modificaciones sobre prueba 1

crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$
```

Si queremos volver a la versión anterior en la cual sólo estaba la primera línea, lo primero que tenemos que hacer es un **git log**. Con este comando podemos ver el hash (ese número tan largo) de cada commit:



```
crispf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git log
commit ef603b825ed2f2f8ab947e6012155d214c284e64 (HEAD -> master)
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 10:07:15 2020 +0200

    segundo commit

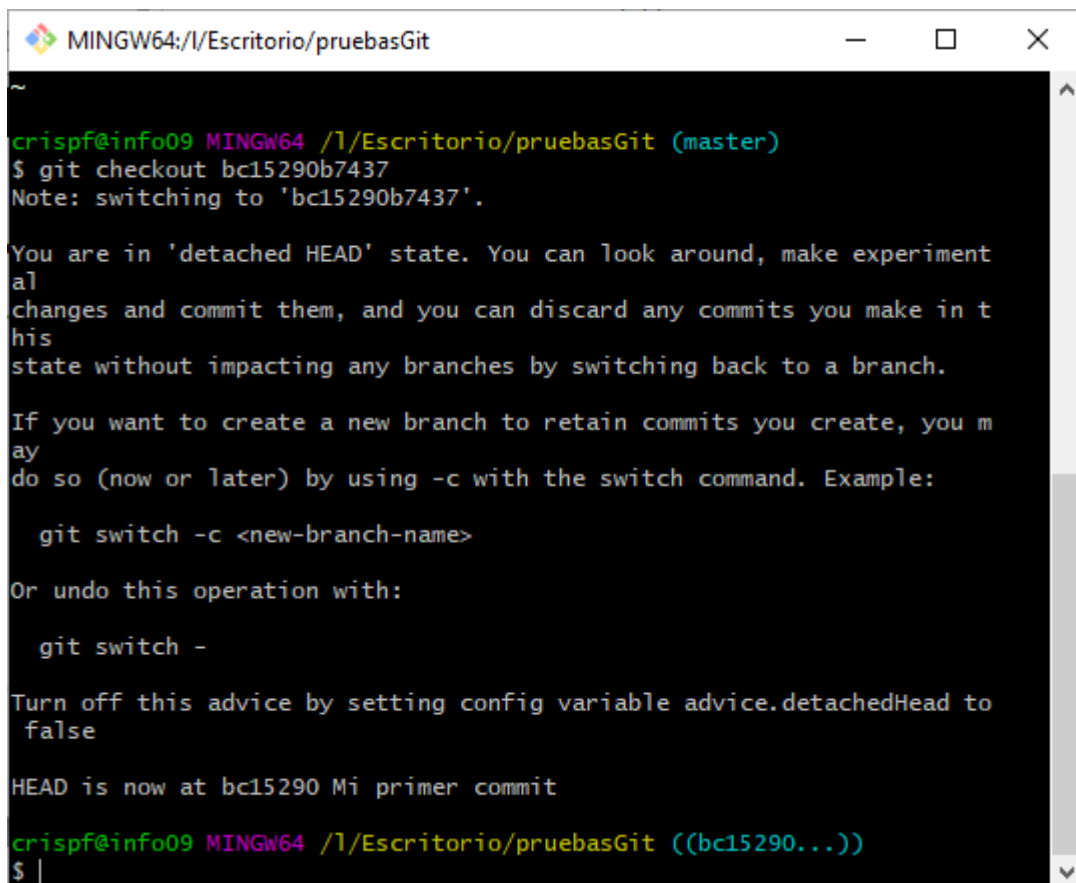
commit bc15290b7437a15de9a4f9de85aeb23c94f1db3e
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 09:53:59 2020 +0200
:...skipping...
commit ef603b825ed2f2f8ab947e6012155d214c284e64 (HEAD -> master)
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 10:07:15 2020 +0200

    segundo commit

commit bc15290b7437a15de9a4f9de85aeb23c94f1db3e
Author: Cristina <crispf@iessanclemente.net>
Date:   Fri Oct 23 09:53:59 2020 +0200

    Mi primer commit
~
~
```

Para revertir los cambios tenemos que copiar los 10 primeros alfanuméricos que identifican el commit donde está la versión de nuestro documento que nos interesa (en este caso Mi primer commit) y pegarlos junto al comando: **git checkout XXX**



```
crispcf@info09 MINGW64 /I/Escritorio/pruebasGit (master)
$ git checkout bc15290b7437
Note: switching to 'bc15290b7437'.

You are in 'detached HEAD' state. You can look around, make experiment
al changes and commit them, and you can discard any commits you make in t
his
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 m
ay
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to
false

HEAD is now at bc15290 Mi primer commit
crispcf@info09 MINGW64 /I/Escritorio/pruebasGit ((bc15290...))
$ |
```

Si después de realizar el checkout comprobamos el contenido del fichero (prueba1.txt) vemos que ha sido modificado y ya no aparece la segunda línea de texto:



```
crispcf@info09 MINGW64 /I/Escritorio/pruebasGit ((bc15290...))
$ cat prueba1.txt
Prueba 1

crispcf@info09 MINGW64 /I/Escritorio/pruebasGit ((bc15290...))
$ |
```

Además nos cambia la versión en la que estamos trabajando, ya no es la master sino una antigua. Como hemos revertido todos los cambios, si hacemos un `ls -la` ya no veremos el archivo `.gitignore`

```
MINGW64:/I/Escritorio/pruebasGit

crispf@info09 MINGW64 /I/Escritorio/pruebasGit ((bc15290...))
$ ls -la
total 12
drwxr-xr-x 1 crispf 1049089 0 oct. 23 10:13 ./
drwxr-xr-x 1 crispf 1049089 0 oct. 23 09:48 ../
drwxr-xr-x 1 crispf 1049089 0 oct. 23 10:13 .git/
-rw-r--r-- 1 crispf 1049089 18 oct. 23 09:58 ignorado.txt
-rw-r--r-- 1 crispf 1049089 10 oct. 23 10:13 prueba1.txt
-rw-r--r-- 1 crispf 1049089 9 oct. 23 09:49 prueba2.txt

crispf@info09 MINGW64 /I/Escritorio/pruebasGit ((bc15290...))
$
```

!!!Para poder continuar creamos un nuevo repositorio (eliminamos la carpeta .git y hacemos un git add \* para rastrear los tres ficheros).

```
MINGW64:/I/Escritorio/pruebaGit

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git add *
warning: LF will be replaced by CRLF in ignorado.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in prueba2.txt.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

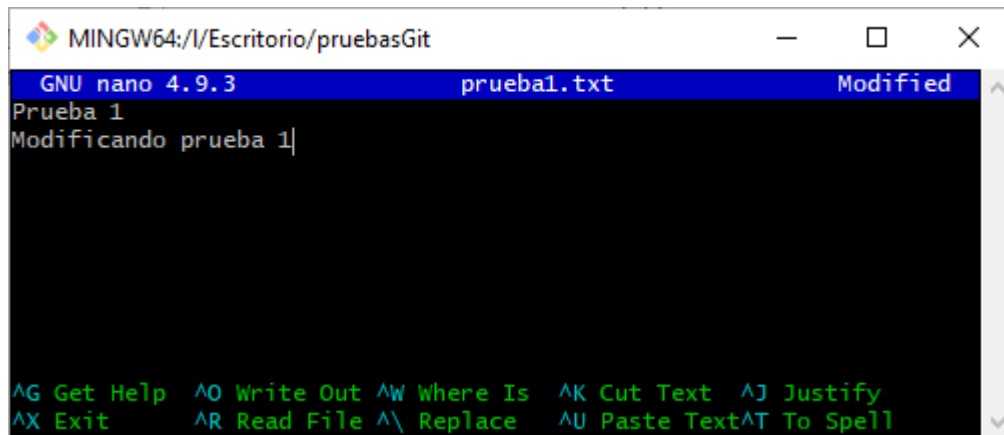
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   ignorado.txt
    new file:   prueba1.txt
    new file:   prueba2.txt

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

## 11. Archivos modificados

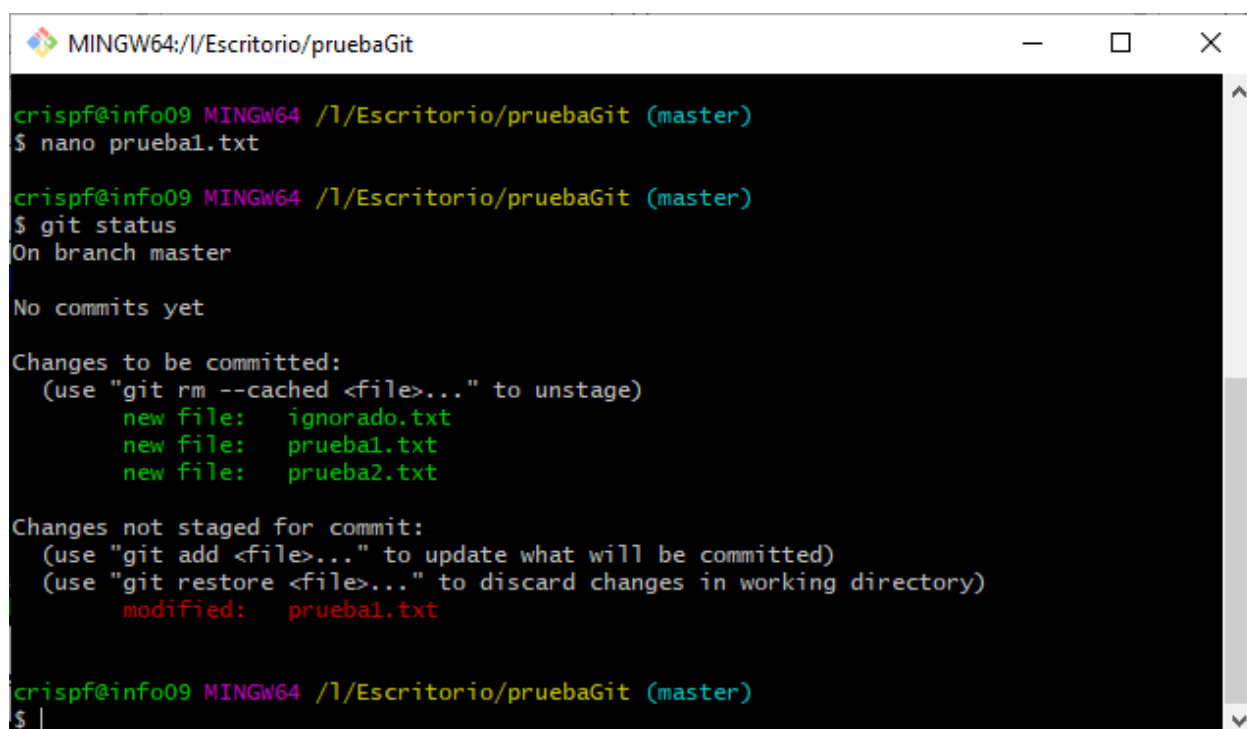
Vamos a cambiar un archivo que esté rastreado. Nos vale cualquier archivo, vamos a modificar el prueba1.txt (por ejemplo).



```
MINGW64:/I/Escritorio/pruebasGit
GNU nano 4.9.3 prueba1.txt Modified
Prueba 1
Modificando prueba 1

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell
```

Si ahora ejecutamos un **git status** veremos algo parecido a esto:



```
MINGW64:/I/Escritorio/pruebaGit
crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ nano prueba1.txt

crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

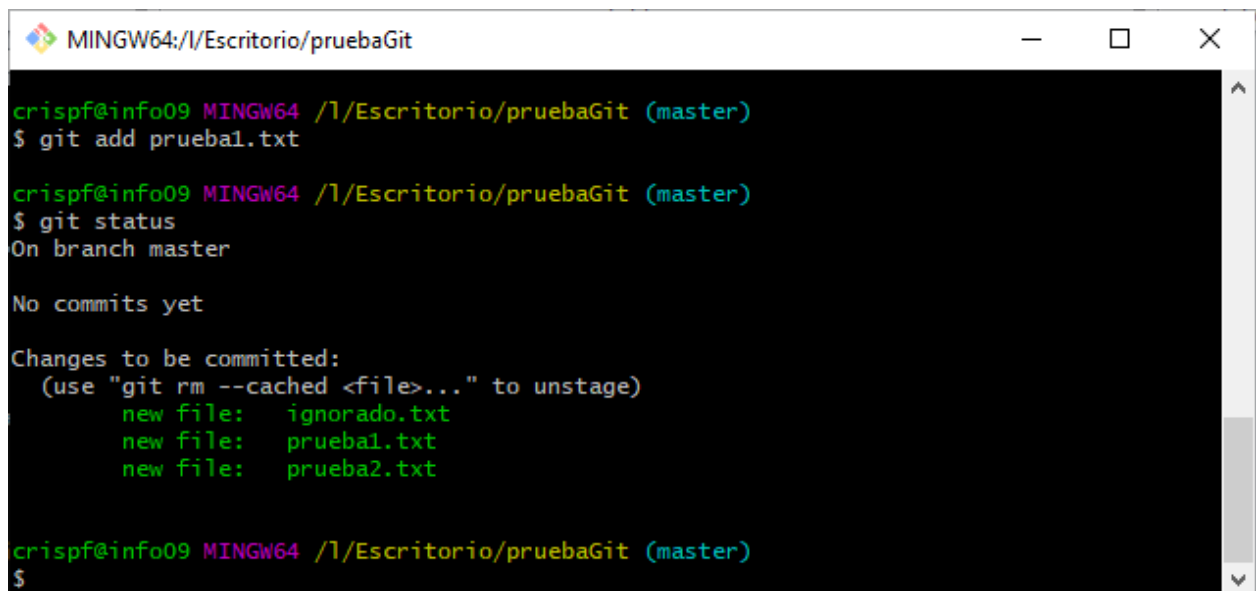
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   ignorado.txt
    new file:   prueba1.txt
    new file:   prueba2.txt

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)
    modified:   prueba1.txt

crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

El archivo “prueba1.txt” aparece en una sección llamada “Changes not staged for commit” (“Cambios no preparado para confirmar” en inglés) - lo que significa que existe un archivo rastreado que ha sido modificado en el directorio de trabajo pero que aún no está preparado. Para prepararlo, ejecutamos el comando **git add**.

Al ejecutar un **git status** comprobamos que todo está correcto:



```
MINGW64:/I/Escritorio/pruebaGit
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git add prueba1.txt

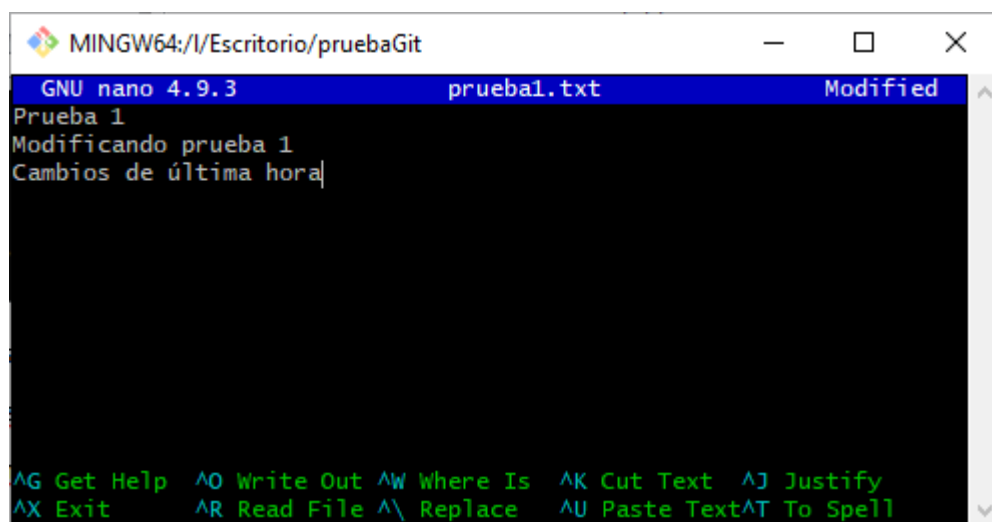
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   ignorado.txt
    new file:   prueba1.txt
    new file:   prueba2.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

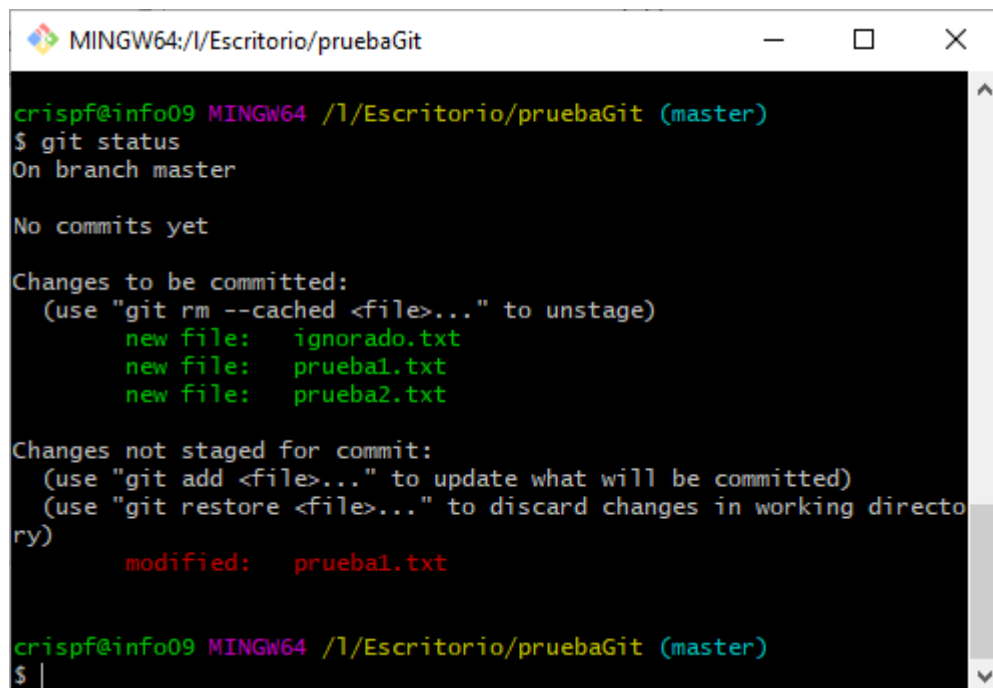
Ambos archivos están preparados y formarán parte de tu próxima confirmación. En este momento, supongamos que recuerdas que debes hacer un pequeño cambio en prueba1.txt antes de confirmarlo. Abres de nuevo el archivo, lo cambias y ahora estás listo para confirmar.



```
GNU nano 4.9.3 prueba1.txt Modified
Prueba 1
Modificando prueba 1
Cambios de última hora|

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell
```

Sin embargo, ejecutemos git status una vez más:



```
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   ignorado.txt
        new file:   prueba1.txt
        new file:   prueba2.txt

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)
        modified:   prueba1.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

Ahora prueba1.txt aparece como preparado y no preparado. Resulta que Git prepara un archivo de acuerdo al estado que tenía cuando ejecutas el comando `git add`.

Si confirmas ahora, se confirmará la versión de prueba1.txt que tenías la última vez que ejecutaste `git add` y no la versión que ves ahora en tu directorio de trabajo al ejecutar `git status`.

Por lo tanto, si modificas un archivo después de ejecutar *git add*, deberás ejecutar `git add` de nuevo para preparar la última versión del archivo.

## 12. Estado abreviado

Si bien es cierto que la salida de **git status** es bastante explícita, también es verdad que es muy extensa. Para comprobar las posibles salidas vamos a:

- Modificar un archivo (M)
- Añadir un nuevo archivo no rastreado (??)
- Preparar un archivo (A).

A screenshot of a terminal window titled 'MINGW64:/I/Escritorio/pruebaGit'. The window shows a series of commands and their outputs. The user runs 'nano prueba1.txt', then 'nano nuevoArchivo.txt', and finally 'git status --short'. The output of 'git status --short' shows: 'A ignorado.txt', 'AM prueba1.txt', 'A prueba2.txt', and '?? nuevoArchivo.txt'. The prompt '\$' is visible at the end of the last command.

```
crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ nano prueba1.txt

crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ nano nuevoArchivo.txt

crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status --short
A  ignorado.txt
AM prueba1.txt
A  prueba2.txt
?? nuevoArchivo.txt

crisprf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

## 13. Patrones para ignorar ficheros

Ya vimos el funcionamiento del archivo `.gitignore`, en el que podemos añadir los nombres de los ficheros que no queremos que sean rastreados, pero también podemos añadir patrones de rastreo, lo que nos permitirá crear reglas y no tener que añadir todos los nombres de los ficheros.

Las reglas sobre los patrones que puedes incluir en el archivo `.gitignore` son las siguientes:

- Ignorar las líneas en blanco y aquellas que comiencen con `#`.
- Aceptar patrones glob estándar.
- Los patrones pueden terminar en barra (`/`) para especificar un directorio.
- Los patrones pueden negarse si se añade al principio el signo de exclamación (`!`).

Los patrones glob son una especie de expresión regular simplificada usada por los terminales.

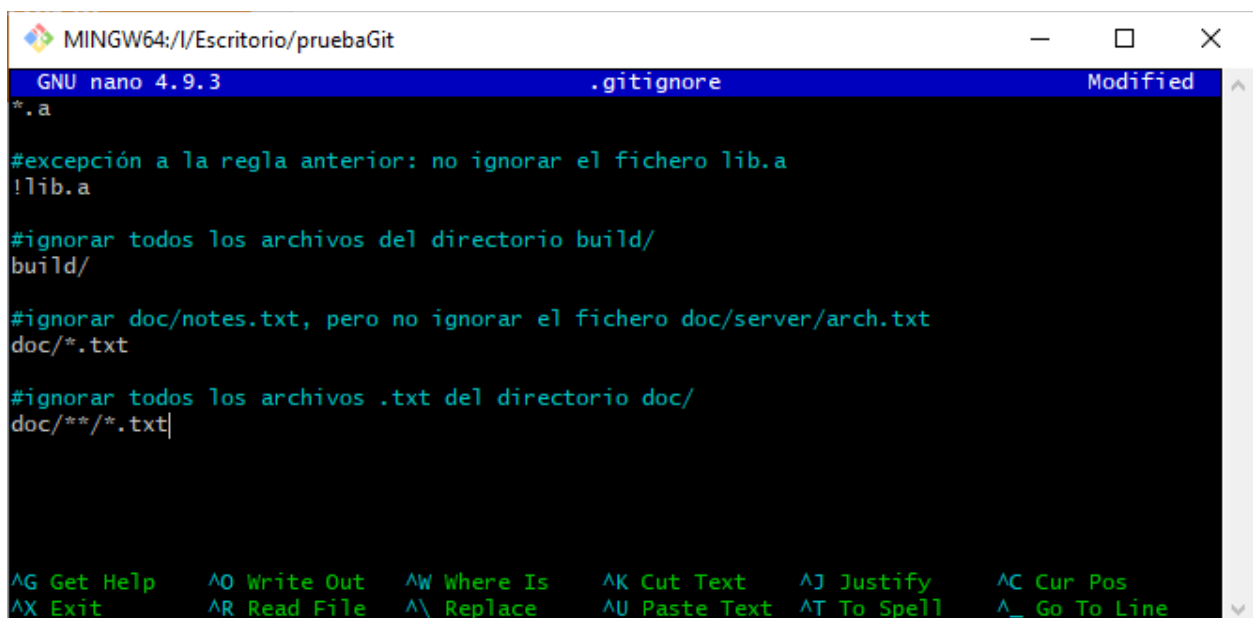
- Un asterisco (`*`) corresponde a cero o más caracteres
- `[abc]` corresponde a cualquier caracter dentro de los corchetes (en este caso a, b o c)
- el signo de interrogación (`?`) corresponde a un caracter cualquiera
- y los corchetes sobre caracteres separados por un guión (`[0-9]`) corresponde a cualquier caracter entre ellos (en este caso del 0 al 9).

- También puedes usar dos asteriscos para indicar directorios anidados; `a/**/z` coincide con `a/z`, `a/b/z`, `a/b/c/z`, etc.

!! Validador online de expresiones regulares:  
<http://www.contadordecaracteres.info/prueba-expresiones-regulares.html>

!! Ejemplos de archivos `.gitignore`:  
<https://github.com/github/gitignore>

Vamos a crear y editar nuestro fichero `.gitignore` añadiendo las siguientes reglas:



```
MINGW64:~/Escritorio/pruebaGit
GNU nano 4.9.3 .gitignore Modified
*.a

#excepción a la regla anterior: no ignorar el fichero lib.a
!lib.a

#ignorar todos los archivos del directorio build/
build/

#ignorar doc/notes.txt, pero no ignorar el fichero doc/server/arch.txt
doc/*.txt

#ignorar todos los archivos .txt del directorio doc/
doc/**/.txt|

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line
```

## 14. Ver cambios no preparados

Como habíamos modificado el archivo `prueba1.txt` y añadido los archivos `.gitignore` y `nuevoarchivo.txt` si hacemos un `git status` nos encontramos con lo siguiente:



```
MINGW64:/I/Escritorio/pruebaGit
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   ignorado.txt
    new file:   prueba1.txt
    new file:   prueba2.txt

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)
    modified:   prueba1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    nuevoArchivo.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

Para ver qué has cambiado pero aún no has preparado, escribe **git diff** sin más parámetros:

```
MINGW64:/I/Escritorio/pruebaGit
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git diff
diff --git a/prueba1.txt b/prueba1.txt
index 4996a6e..fadfa8f 100644
--- a/prueba1.txt
+++ b/prueba1.txt
@@ -1,2 +1,3 @@
 Prueba 1
 Modificando prueba 1
+Cambios de última hora en prueba1

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

Nos dice que hemos modificado el archivo prueba1.txt añadiendo la línea que aparece resaltada en verde y con un + al inicio: *"Cambios de última hora en prueba1"* .

Ahora vamos a preparar y confirmar todos los cambios:

```
MINGW64:/I/Escritorio/pruebaGit

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git add *
warning: LF will be replaced by CRLF in nuevoArchivo.txt.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   ignorado.txt
        new file:   nuevoArchivo.txt
        new file:   prueba1.txt
        new file:   prueba2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git add .gitignore
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$
```

El archivo .gitignore no lo coge con el \* por eso hacemos un nuevo git add para incluirlo.

Ahora comprobamos que tenemos todo preparado:

```
MINGW64:/I/Escritorio/pruebaGit

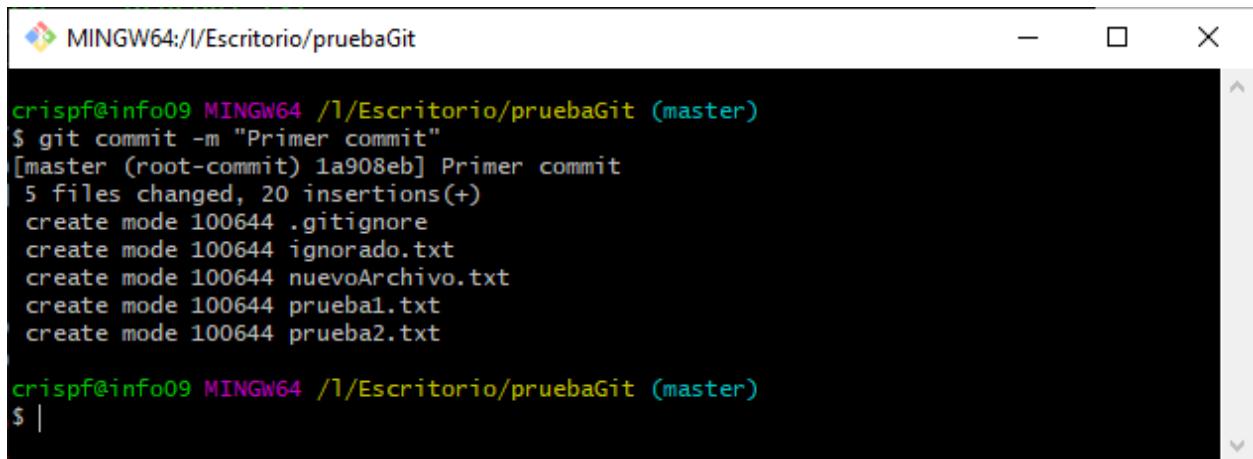
crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   ignorado.txt
        new file:   nuevoArchivo.txt
        new file:   prueba1.txt
        new file:   prueba2.txt

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

Y confirmamos:



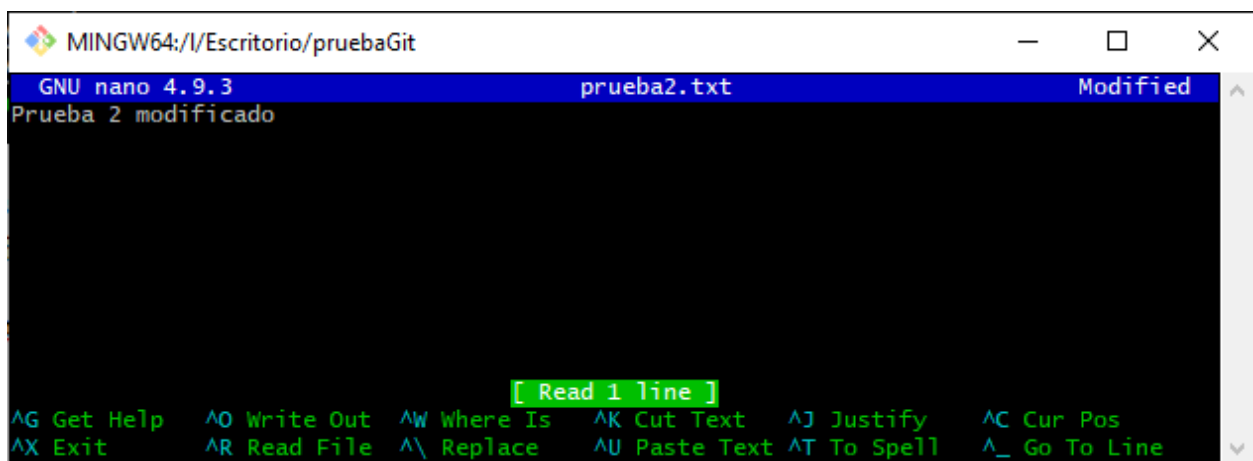
```

MINGW64:/l/Escritorio/pruebaGit
crispf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git commit -m "Primer commit"
[master (root-commit) 1a908eb] Primer commit
5 files changed, 20 insertions(+)
create mode 100644 .gitignore
create mode 100644 ignorado.txt
create mode 100644 nuevoArchivo.txt
create mode 100644 prueba1.txt
create mode 100644 prueba2.txt

crispf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ |

```

Vamos a modificar ahora el prueba2.txt:



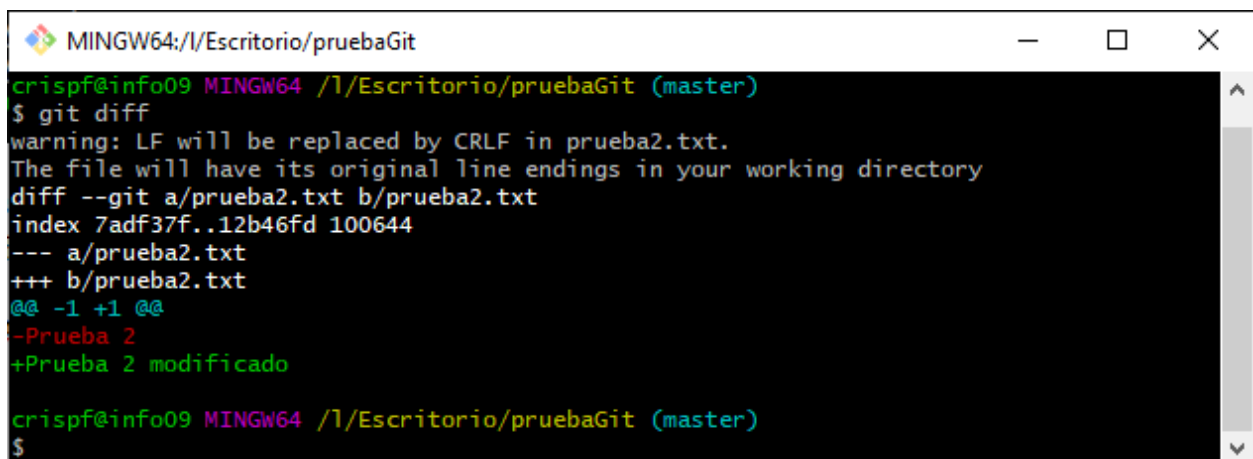
```

MINGW64:/l/Escritorio/pruebaGit
GNU nano 4.9.3 prueba2.txt Modified
Prueba 2 modificado

[ Read 1 line ]
^G Get Help  ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line

```

Si ejecutamos un **git diff** nos indica que una línea ha sido borrada (rojo) y otra ha sido añadida (verde).



```

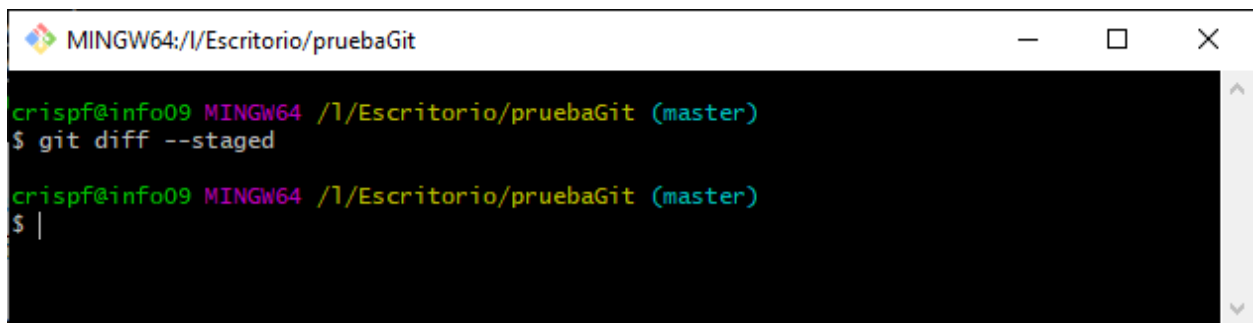
MINGW64:/l/Escritorio/pruebaGit
crispf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git diff
warning: LF will be replaced by CRLF in prueba2.txt.
The file will have its original line endings in your working directory
diff --git a/prueba2.txt b/prueba2.txt
index 7adf37f..12b46fd 100644
--- a/prueba2.txt
+++ b/prueba2.txt
@@ -1,1 @@
-Prueba 2
+Prueba 2 modificado

crispf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$

```

Si quieres ver lo que has preparado y será incluido en la próxima confirmación, puedes usar **git diff --staged**. Este comando compara tus cambios preparados con la última instantánea confirmada.

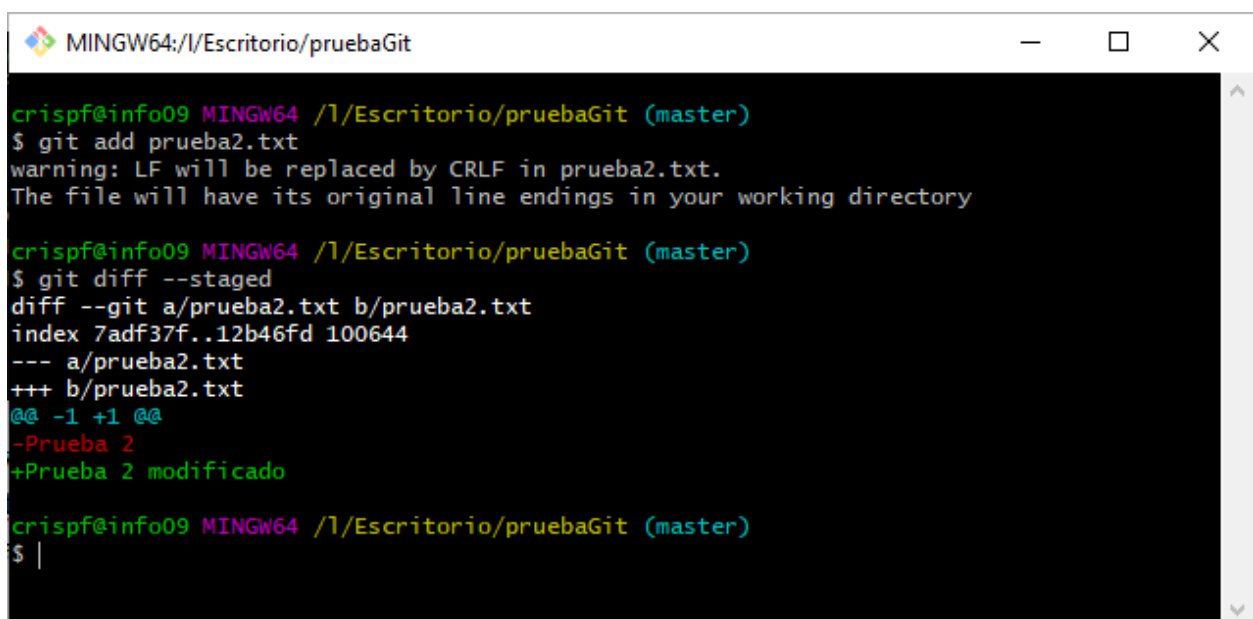
Si ejecutamos ahora `git diff --staged` no veremos nada (ya que no hay nada preparado).

A terminal window titled 'MINGW64:/I/Escritorio/pruebaGit' showing a user named 'crispf@info09' in a 'MINGW64' environment. The user is on the 'master' branch. They run the command 'git diff --staged', and the terminal shows no output, indicating no staged changes.

```
crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git diff --staged

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

Si ejecutamos `git add` con este fichero, y a continuación volvemos a ejecutar el comando `git diff --staged`, comprobaremos que ahora si que muestra información:

A terminal window titled 'MINGW64:/I/Escritorio/pruebaGit' showing a user named 'crispf@info09' in a 'MINGW64' environment. The user is on the 'master' branch. They run 'git add prueba2.txt', which shows a warning about line endings. Then they run 'git diff --staged', which shows the diff for 'prueba2.txt' with a modification.

```
crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git add prueba2.txt
warning: LF will be replaced by CRLF in prueba2.txt.
The file will have its original line endings in your working directory

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git diff --staged
diff --git a/prueba2.txt b/prueba2.txt
index 7adf37f..12b46fd 100644
--- a/prueba2.txt
+++ b/prueba2.txt
@@ -1,1 @@
-Prueba 2
+Prueba 2 modificado

crispf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

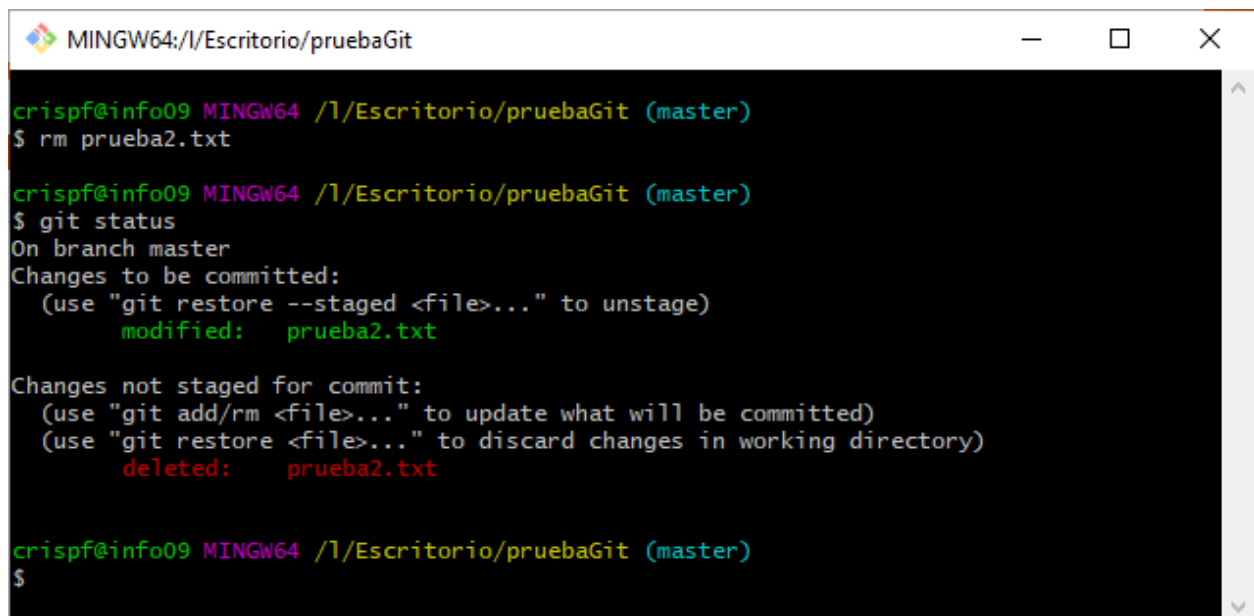
Es importante resaltar que al llamar a **git diff** sin parámetros no verás los cambios desde tu última confirmación, sólo verás los cambios que aún no están preparados. Esto puede ser confuso porque si preparas todos tus cambios, `git diff` no te devolverá ninguna salida.

## 15. Eliminar archivos

Si simplemente eliminas el archivo de tu directorio de trabajo, aparecerá en la sección “Changes not staged for commit” (esto es, sin preparar) en la salida de **git status**.

Vamos a eliminar el fichero `prueba2.txt`: **rm prueba2.txt**

Si hacemos un **git status**, comprobamos que el archivo ha sido marcado para eliminar:

A terminal window titled 'MINGW64:/l/Escritorio/pruebaGit' showing the execution of 'rm prueba2.txt' followed by 'git status'. The status output indicates that 'prueba2.txt' is deleted and staged for commit.

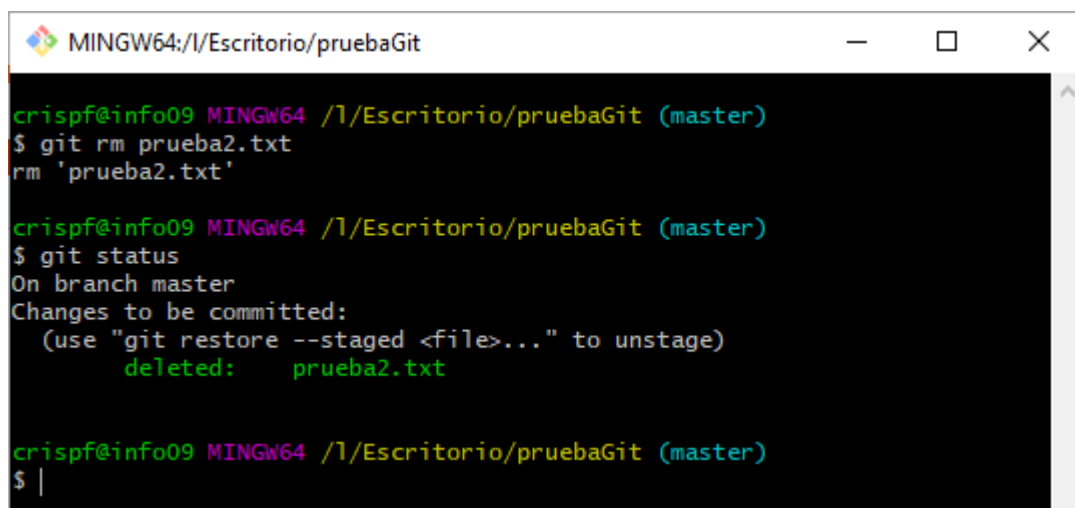
```
crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ rm prueba2.txt

crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   prueba2.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    prueba2.txt

crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$
```

Ahora, si ejecutas **git rm**, entonces se prepara la eliminación del archivo:

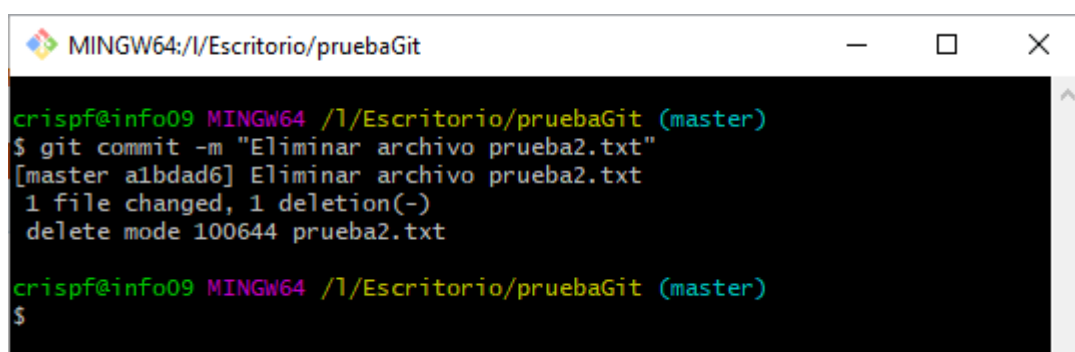
A terminal window titled 'MINGW64:/l/Escritorio/pruebaGit' showing the execution of 'git rm prueba2.txt'. The status output shows 'prueba2.txt' as deleted and staged for commit.

```
crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git rm prueba2.txt
rm 'prueba2.txt'

crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    prueba2.txt

crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ |
```

Con la próxima confirmación, el archivo habrá desaparecido y no volverá a ser rastreado.

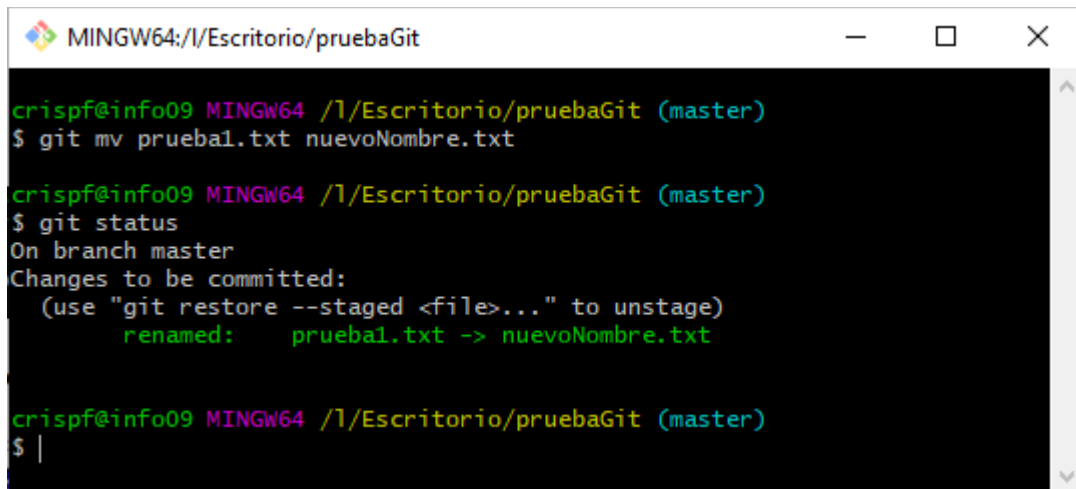
A terminal window titled 'MINGW64:/l/Escritorio/pruebaGit' showing the execution of 'git commit -m "Eliminar archivo prueba2.txt"'. The commit message and file changes are displayed.

```
crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$ git commit -m "Eliminar archivo prueba2.txt"
[master a1bdad6] Eliminar archivo prueba2.txt
1 file changed, 1 deletion(-)
delete mode 100644 prueba2.txt

crispcf@info09 MINGW64 /l/Escritorio/pruebaGit (master)
$
```

## 16. Cambiar el nombre archivos

Para cambiar el nombre de un archivo tenemos el siguiente comando: **git mv fichOrigen.txt nuevoNombre.txt**



```
crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git mv prueba1.txt nuevoNombre.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:   prueba1.txt -> nuevoNombre.txt

crispcf@info09 MINGW64 /I/Escritorio/pruebaGit (master)
$ |
```

## 17. Opciones del historial de confirmaciones

Después de haber hecho varias confirmaciones, o si has clonado un repositorio que ya tenía un histórico de confirmaciones, probablemente quieras mirar atrás para ver qué modificaciones se han llevado a cabo. La herramienta más básica y potente para hacer esto es el comando **git log**.

Se ven las confirmaciones en orden cronológico inverso.

Algunas de las opciones más útiles son:

- **-p**: que muestra las diferencias introducidas en cada confirmación.
- **-2**: que hace que se muestren únicamente las dos últimas entradas del historial

Se puede utilizar las opciones temporales como **--since** (desde) y **--until** (hasta) que resultan muy útiles. Por ejemplo, este comando lista todas las confirmaciones hechas durante las dos últimas semanas: **git log --since=2.weeks**

## 18. Deshacer

En cualquier momento puede que quieras deshacer algo. Una de las acciones más comunes a deshacer es cuando confirmas un cambio antes de tiempo y olvidas agregar algún archivo, o te equivocas en el mensaje de confirmación.

Si quieres rehacer la confirmación, puedes reconfirmar con la opción **--amend**: **git commit --**

## amend -m "Mensaje"

Si hacemos un git status tenemos el siguiente contenido:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    nuevoarchivo.txt
        renamed:    prueba1.txt -> nuevonombre.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        nuevoarchivo.txt
```

Si hacemos un commit vemos lo siguiente:

```
$ git commit -m "commit a medias"
[master 4af4e8d] commit a medias
2 files changed, 1 deletion(-)
delete mode 100644 nuevoarchivo.txt
rename prueba1.txt => nuevonombre.txt (100%)
```

Vemos que no nos incluyó el nuevoarchivo.txt porque no hicimos un add:

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        nuevoarchivo.txt
```

Preparamos el fichero:

```
$ git add nuevoarchivo.txt
warning: LF will be replaced by CRLF in nuevoarchivo.txt.
The file will have its original line endings in your working directory
```

Si queremos modificar el commit anterior y no generar uno nuevo tenemos la opción **--amend**:

```
$ git commit --amend -m "nuevo commit que modifica al anterior"
[master d31b495] nuevo commit que modifica al anterior
Date: Wed Oct 23 20:42:05 2019 +0200
2 files changed, 1 insertion(+)
rename prueba1.txt => nuevonombre.txt (100%)
```

```
$ git status
On branch master
nothing to commit, working tree clean
```

Si hacemos un **git log** para ver el historial de cambios vemos que el commit primero cuyo

mensaje era “commit a medias” no aparece.

## 19. Trabajar con remotos

Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet o en cualquier otra red. Colaborar con otras personas implica gestionar estos repositorios remotos enviando y bajando datos de ellos cada vez que necesites compartir tu trabajo.

Para ver esta funcionalidad, vamos a clonar nuestro repositorio, para lo que utilizamos el comando: **git clone**

Para ver los repositorios remotos que tenemos configurados, utilizamos el comando: **git remote**

```
$ git remote  
origin
```

Como hemos clonado el repositorio, vemos origin (origen, en inglés) que es el nombre que por defecto Git le da al servidor del que hemos clonado.

Si queremos mostrar las URLs que Git ha asociado al nombre y que serán utilizadas al leer y escribir en ese remoto, usaremos la opción -v: **git remote -v**

```
$ git remote -v  
origin https://github.com/schacon/ticgit (fetch)  
origin https://github.com/schacon/ticgit (push)
```

Para añadir un remoto nuevo y asociarlo a un nombre que se pueda diferenciar fácilmente usaremos el comando: **git remote add [nombre] [url]**

A la hora de enviar los ficheros a un remoto, deberemos utilizar el comando: **git push [nombreRemoto] [nombreRama]**

En caso de lo que necesitemos sea obtener los datos de los proyectos remotos, podremos usar el comando: **git fetch [nombreRemoto]**



## ANEXO I: Materiales

### I. Textos de apoyo o de referencia

- Manuais San Clemente:

[https://manuais.iessanclemente.net/index.php/Control de versiones con Git y GitHub](https://manuais.iessanclemente.net/index.php/Control%20de%20versiones%20con%20Git%20y%20GitHub)

- Libro Git Pro: <https://git-scm.com/book/es/v2>

### II. Recursos web

- Página oficial de Git: <https://git-scm.com>
- Página oficial de Bit bucket: <https://bitbucket.org/>

### III. Recursos didácticos

- Apuntes en el aula virtual.
- Ordenador personal, con navegador web y conexión a internet.
- Software para elaboración de documentos de texto.