



Práctica GIT

Contenido

1. Iniciando el repositorio.
2. Trabajando en el repositorio.

2
3

12 MOST COMMON GIT COMMANDS

git init  Creates a new local repository in the current directory	git clone  Copies an existing remote repository to your local machine.	git status  Shows the state of your working directory and staging area.
git add  Adds changes in your working directory to the staging area, which is a temporary area where you can prepare your next commit.	git commit  Records the changes in the staging area as a new snapshot in the local repository, along with a message describing the changes.	git push  Uploads the local changes to the remote repository usually a on a platform like GitHub or GitLab.
git pull  Downloads the latest commits from a remote repository and merges them with your local branch.	git branch  Lists, creates, renames, or deletes branches in your local repository. A branch is a pointer to a specific commit.	git checkout  Switches your working directory to a different branch or commit, discarding any uncommitted changes
git merge  Combines the changes from one branch into another branch, creating a new commit if there are no conflicts	git diff  Shows the differences between two commits, branches, files, or the working directory and the staging area.	git log  Shows the history of commits in the current branch, along with their messages, authors, and dates.



1. Iniciando el repositorio.

Para ir familiarizándonos con los comandos que podemos utilizar vamos a realizar un ejercicio práctico.



El ejercicio lo vamos a realizar con GIT Bash, la interfaz de línea de comandos de Git, ya que a pesar de que hay aplicaciones de escritorio gráficas que nos pueden ayudar, lo más importante es saber manejarse bien con los comandos.

En primer lugar, abriremos el terminal de Git (**Git Bash**) y creamos un directorio de prueba en el que vamos a trabajar. Una vez creado accedemos a él y ejecutamos el comando '**git init**'. Con este comando creamos un repositorio local en el directorio de trabajo.

```
MINGW64/c:/Users/fernando.fernandez/Prueba_Git
fernando.fernandez@L2307001 MINGW64 ~
$ mkdir Prueba_Git
fernando.fernandez@L2307001 MINGW64 ~
$ cd Prueba_Git/
fernando.fernandez@L2307001 MINGW64 ~/Prueba_Git
$ git init
Initialized empty Git repository in C:/Users/fernando.fernandez/Prueba_Git/.git/
```

La terminal nos informa de que hemos iniciado el repositorio y que ha creado en el 'fichero oculto' '.git'. Es importante mantener esta carpeta segura ya que si se pierde entonces perderemos el repositorio.

Una vez creado el repositorio, la carpeta en la que estamos sería nuestro **working directory**. Para ver el estado en el que se encuentra usamos el comando '**git status**':

```
MINGW64/c:/Users/fernando.fernandez/Prueba_Git
fernando.fernandez@L2307001 MINGW64 ~/Prueba_Git (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Este comando nos indica la rama en la que estamos trabajando y que no hemos realizado ningún **commit** hasta la fecha. Esto es lógico ya que acabamos de crear el repositorio y aún no hemos trabajado en él.



i Un **commit** representa un punto en la historia del proyecto donde hemos guardado cambios específicos. En otras palabras, es algo así como una captura de tu proyecto suele ir acompañado de una breve descripción que aclara qué nuevas modificaciones se han hecho. Este comando crea una nueva versión del proyecto en tu zona de trabajo local (**staging area**).

Cada **commit** tiene un identificador único que nos permite realizar un seguimiento de la evolución del código a lo largo del tiempo y tener un registro de los cambios realizados y por lo tanto, podríamos incluso revertir cambios.

2. Trabajando en el repositorio.

Para comenzar a trabajar en el repositorio vamos a copiar el archivo '**home.html**' en el directorio que creamos en el paso anterior. El archivo contiene la siguiente información:

```
home.html
C: > Users > Fernando > Prueba_Git > home.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE-edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Nter</title>
8  </head>
9  <body>
10     <h1>Bienvenidos a Nter</h1>
11 </body>
12 </html>
13
14
```

Una vez realizado el cambio, volvemos a comprobar el estado de nuestro repositorio ('**git status**')

```
MINGW64/c/Users/Fernando/Prueba_Git
msysgit> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
```

Vemos que Git nos avisa de que el archivo que acabamos de agregar en el directorio no está en seguimiento. Para añadir los archivos al seguimiento utilizamos el comando '**git add**':



```
MINGW64/c/Users/fernando/Prueba_Git
fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git add home.html

fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git add .

fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   home.html
```

- i** El comando **git add** puede ser utilizado principalmente de dos formas:
1. **'git add nombre_archivo'**: Para agregar un archivo específico.
 2. **'git add .'**: Para agregar todos los archivos modificados y nuevos en el directorio actual.

Utilizando este comando estaríamos 'colocando' los archivos en el área de preparación o **staging area**. Los archivos en esta área están listos para ser incluidos en el próximo commit. En otras palabras, hemos indicado a Git que deseamos incluir esos cambios específicos en nuestro historial de versiones.

Es importante tener en cuenta que los archivos en el **staging area** no se han confirmado completamente en el repositorio de Git, será necesario que realicemos un **commit** para que esos cambios se guarden permanentemente en el historial de versiones. Para ello usamos el comando **'git commit'**

Este comando coge los archivos que hemos agregado previamente con **'git add'** al área de preparación y los guarda como un nuevo **commit** en el historial de versiones de Git. El mensaje del commit proporciona una descripción breve de los cambios realizados en ese commit en particular.

```
MINGW64/c/Users/fernando/Prueba_Git
fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git commit -m 'Comenzamos el proyecto'
[master (root-commit) fe731e6] Comenzamos el proyecto
1 file changed, 12 insertions(+)
 create mode 100644 home.html
```

- i** En comando **'git commit -m "Mensaje del commit"'** el parámetro **'-m "Mensaje del commit"'** proporciona una breve descripción de los cambios realizados en ese **commit** en particular.

Nota: Si no usamos el parámetro **'-m'**, cuando ejecutemos el comando se nos abrirá un editor de texto para que veamos el cambio que vamos a realizar y podamos añadir la descripción.

¡Con esto hecho ya tenemos nuestro primer commit!



Para revisar el **commit** que acabamos de hacer usaremos el comando con '**git log**'. Este comando nos permite ver el historial de commits en el repositorio de Git. Al ejecutarlo, veremos una lista de commits junto con información relevante, como el autor, la fecha, y el mensaje asociado al commit.

```
MINGW64/c/Users/fernando/Prueba_Git
fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git log
commit fe731e6ee728326e8abd00126e261bb904a00b0b (HEAD -> master)
Author: fernando <fernando@nter.com>
Date:   Fri Jan 19 10:13:44 2024 +0100

    Comenzamos el proyecto
```

Vemos que nuestro commit tiene un código asociado a él, '**fe731e6ee728326e8abd...**'. Este código nos servirá para poder identificarlo y, como veremos más adelante, volver a ese punto si fuese necesario.

Si no queremos ver tanta información acerca de los commits, el parámetro '**--oneline**' hará que se nos muestre la información en una sola línea.

```
MINGW64/c/Users/fernando/Prueba_Git
fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git log --oneline
fe731e6 (HEAD -> master) Comenzamos el proyecto
```

Ahora que hemos realizado nuestro primer commit, vamos a empezar a hacer cambios en el repositorio y ver qué ocurre. Para ello, en primer lugar, debemos hacer alguna modificación en el archivo. Esto lo podemos hacer abriendo el archivo en cualquier editor de texto plano y por ejemplo, escribiendo 'Bienvenidos a Nter 2.0' y una vez hecho, seguimos los siguientes pasos:

```
MINGW64/c/Users/fernando/Prueba_Git
fernando@L2307001 MINGW64 ~/Prueba_Git (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:   home.html

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

fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git add .

fernando@L2307001 MINGW64 ~/Prueba_Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   home.html
```



A continuación, volvemos a modificar el archivo y hacemos lo siguiente:

```

MINGW64~/c/Users/.../Prueba_Git
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   home.html

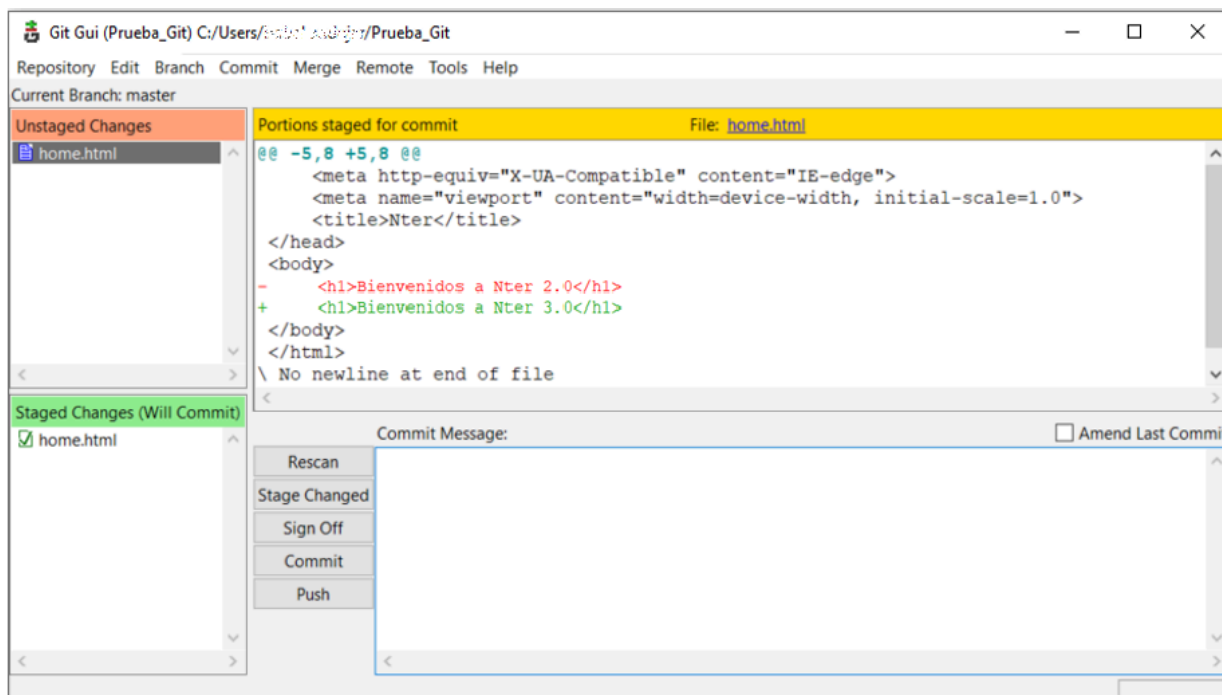
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   home.html

MINGW64~/c/Users/.../Prueba_Git
$ git diff
diff --git a/home.html b/home.html
index ea9f8aa..db77565 100644
--- a/home.html
+++ b/home.html
@@ -7,6 +7,6 @@
<title>Nter</title>
</head>
<body>
-  <h1>Bienvenidos a Nter 2.0</h1>
+  <h1>Bienvenidos a Nter 3.0</h1>
</body>
</html>
\ No newline at end of file
  
```

Como vemos tenemos cambios guardados en el **staging area** y, posteriormente se han vuelto a hacer cambios sobre el mismo fichero.

Para ver las modificaciones que se han hecho en fichero con respecto a lo que tenemos en el **staging area** podemos utilizar el comando 'git diff'. Este comando nos muestra en rojo la línea modificada que está guardada en el **staging area** y en verde la línea de la última modificación que se ha realizado.

Otra opción para ver estos cambios es usar 'Git Gui':





Si ahora hacemos un **commit**, debemos tener en cuenta que solo se almacenarán los cambios que estaban en el **staging area** y, por lo tanto, si queremos almacenar todos los cambios debemos añadir las últimas modificaciones.

```
MINGW64/c/Users/.../Prueba_Git

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   home.html

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:   home.html

$ git commit -m 'Estamos trabajando en el proyecto'
[master cf21f96] Estamos trabajando en el proyecto
1 file changed, 1 insertion(+), 1 deletion(-)
```

No obstante, si hacemos ese **commit**, tenemos la opción de agregar esos cambios al commit anterior, para ellos usamos el comando **'git commit --amend'** :

```
MINGW64/c/Users/.../Prueba_Git

$ 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:   home.html

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

$ git add .

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   home.html

$ git commit --amend
```

Una vez ejecutemos el **git commit --amend** se nos abrirá 'Vim' un editor de código en el que GIT nos va a indicar el mensaje descriptivo del commit al que vamos a agregar estos cambios de última hora, junto con los ficheros involucrados, la fecha, etc. Será algo parecido a lo siguiente:

