Práctica Git - 1ª Parte: Introducción

Configurar el Git e inicializar un proyecto.

En primer lugar vamos a crear una carpeta para hacer la práctica. Creamos la ruta ***c:\practica\lista*** y dentro de esta creamos el archivo ***alimentos.txt*** y le añadimos una primera línea: ***agua***

(En cada línea que añadamos pulsamos ***<Intro>*** al final)

Abrimos el ***Git CMD***, que es con el que vamos a trabajar en esta práctica en Windows, pero sería igual hacerlo todo con el ***Git Bash***, salvo los comandos propios del sistema. (El CMD se basa en comandos Windows y el Bash en comandos del Shell Bash de Linux). Los comandos del Git no cambian. A continuación nos trasladamos a la carpeta de la práctica (del proyecto que queremos seguir en Git).

```
> cd c:\practica\lista
```

Antes de inicializar el proyecto vamos a asegurarnos de que tenemos ya una configuración básica. Para eso usamos el comando:

```
> git config --list
```

Nos fijamos en el ***user.name*** y en el ***user.email*** y si no son correctos escribimos:

```
> git config --global user.name "nombre_que_queramos"
> git config --global user.email "correo_electronico_que_queramos"
```

Ahora ya podemos inicializar el proyecto en **Git**. Para eso escribimos:

```
> git init
```

Esto crea una carpeta oculta ***.git*** en nuestro proyecto en la que están todos los archivos y carpetas necesarios para llevar el seguimiento de este.

Ver el estado de los archivos

Si ahora hacemos:

```
> git status
```

Veremos que aparece en rojo el archivo ***alimentos.txt*** y que nos aparece la siguiente información:

```
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        alimentos.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Nos informa de que estamos en la rama maestra (la rama por defecto antes de crear ninguna otra).

Que hemos hecho el commit inicial (inicializar el git del proyecto con el comando anterior)

Que hay ficheros fuera de seguimiento que podemos añadir al área de preparación (staging area) para su posterior confirmación (commit).

Añadir y Quitar ficheros del Área de Preparación

Así que lo primero que haremos será:

```
> git add alimentos.txt
> git status
```

Ahora vemos que pone cosas diferentes:

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

Nos informa de que hemos añadido cosas (nos aparece ***alimentos.txt*** en verde) al área de preparación (staging area) y ya pueden ser confirmadas (commit), es decir, añadidas al repositorio. De momento aún no lo están.

También nos informa de que podemos sacarla fuera del área de preparación (to unstage) por si no queríamos realmente hacer un commit con ese archivo, que ha sido simplemente un error.

Vamos a hacer esto último:

```
> git rm --cached alimentos.txt
```

Si ahora hacemos un git status veremos que nos sale lo mismo que al principio. O sea que no tenemos nada en el área de preparación y por tanto no tenemos nada que confirmar.

Podemos hablar de que tenemos dos comandos para añadir y eliminar del área de preparación*:* ***git add*** y ***git rm --cached***

Hacer el primer commit y configurar editor

Vamos ahora a añadirlo de nuevo y a confirmar para añadir de una vez el archivo a nuestro repositorio:

```
> git add "alimentos.txt"
> git commit
```

Se nos abrirá el editor por defecto, el **vim** si no hemos tocado nada.

Si intentamos escribir veremos que no hace nada. Para entrar en modo edición y poder escribir hay que pulsar antes la tecla i (insert) o a (add). Ahora podemos escribir el mensaje. Ponemos por ejemplo: ***commit inicial***. Ahora para salir tenemos que pulsar primero ***<Escape>*** para volver al modo comando. A continuación pulsar la tecla dos puntos ":" y escribir wq (write and quit) para poder salir.

Como este editor es un poco engorroso si no se conoce vamos a cambiarlo. Escribimos:

```
> git config --global core.editor notepad.exe
```

Ahora la próxima vez nos abrirá el bloc de notas, mucho menos engorroso si estamos acostumbrados a Windows. Pero vamos a seguir con el último commit.

Al hacer el commit no hace falta especificar el nombre de ningún archivo porque añadirá todos los archivos que estén en el área de preparación (Staging area a partir de ahora).

La información que nos da nada más hacer el commit es:

```
[master (root-commit) 366bf25] Commit inicial
 1 file changed, 1 insertion(+)
 create mode 100644 alimentos.txt
```

Lo que más nos interesa de momento es la primera línea. En ella nos dice que hemos hecho un commit en la rama maestra y le da un nombre (366bf25) que diferirá en vuestro caso, pues es el código hash generado a partir de la fecha y hora actual del sistema y el tamaño y contenido del fichero. En realidad sólo se muestran los 6 primeros caracteres del código generado y que identificará a ese commit a partir de ahora.

Si ahora hacemos:

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

Nos está diciendo que no hay cambios en nuestro directorio de trabajo. Así que lo tenemos todo confirmado, todo limpio. Hasta que no hagamos de nuevo cambios en el fichero o añadamos nuevos ficheros no podremos añadir nada a la staging area ni, por tanto, hacer commit.

De hecho, si intentamos hacer un nuevo commit, nos volverá a decir lo mismo:

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

Si queremos cambiar el nombre del commit porque nos hemos equivocado, por el **vim** o porque nos hemos olvidado algo o por lo que sea, podemos volver a hacerlo de la siguiente forma:

```
> git commit -m "nuevo mensaje" --amend
```

De esta manera hemos cambiado el nombre del commit y nos ha generado un nuevo identificador.

Realizar y deshacer cambios en el Directorio de Trabajo

Vamos ahora a hacer un cambio, añadiendo otra línea que ponga, por ejemplo: ***peras***

Acordémonos de añadir un ***<Intro>*** al final. Y ahora hagamos:

```
> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   alimentos.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Ahora ya no aparece nada "untracked" como al principio ni pone Initial Commit. Aparece el fichero alimentos.txt de nuevo en rojo pero en estado "modified".

Otra cosa nueva es que, además de indicarnos que podemos usar add para pasar el fichero a la staging area, podemos también usar un nuevo comando: git checkout, para descartar cambios en el directorio de trabajo.

Así que, desde el momento que Git hace el seguimiento del proyecto y detecta que hay modificaciones en archivos nos permite también deshacer los últimos cambios modificados.

Vamos a hacer esto último:

```
> git checkout alimentos.txt
> git status
On branch master
nothing to commit, working tree clean
```

Vemos que vuelve a estar todo limpio. Y si abrimos el archivo veremos que la última línea añadida la ha borrado. Ha deshecho los cambios que habíamos realizado, y que el **Git** ha detectado, desde la última vez que hicimos commit. Así que, en realidad es un comando peligroso. Normalmente casi nunca querremos utilizar algo tan drástico. Es el único comando que realmente borra cambios, haciéndolos irrecuperables.

Vamos ahora a realizar de nuevo el cambio (añadir ***peras*** + **<intro>**) y a añadirlo y hacer un segundo commit:

```
> git add .
> git commit -m "Segundo commit"
[master d8219b4] Segundo commit
 1 file changed, 1 insertion(+)
```

Esta vez en lugar de especificar el nombre del archivo hemos puesto por comodidad un punto que es como decirle que añada todo lo que haya por añadir. De nuevo vuelve a ponernos en la primera línea el nombre del commit y el mensaje (la marca amarilla es para destacar que son los primeros 7 caracteres del nombre).

Ver la lista de commits y diferencias

Vamos a ver ahora los commits que tenemos de momento con otro de los comandos más usados:

```
> git log
commit d8219b49c2543c3ce8402ac65f6eb12e02b05319
Author: Nacho Rodriguez <irodriguezn@gmail.com>
Date:   Sun Nov 6 17:35:44 2016 +0100
    Segundo commit
commit 366bf252ec997c3ae5976c13bb887e3ed8cb1229
Author: Nacho Rodriguez <irodriguezn@gmail.com>
Date:   Sun Nov 6 17:13:58 2016 +0100
    Commit inicial
```

Nos aparecen los dos commits realizados en orden inverso. Es decir, el último siempre sale primero. Nos aparece una primera línea con el nombre del commit completo, autor, email, fecha del commit y el mensaje que le pusimos a este.

Veamos cómo podemos ver diferencias entre estos dos commits. Para ellos vamos a usar ***git diff*** especificando los 6 primeros caracteres del git que queremos comparar con el último (en realidad con menos también vale con un mínimo de 4):

```
> git diff 366bf2
diff --git a/alimentos.txt b/alimentos.txt
index 295ea8d..a37656a 100644
--- a/alimentos.txt
+++ b/alimentos.txt
```

```
@@ -1,2 +1,3 @@
 agua
+peras
```

Vamos a irnos acostumbrando poco a poco a ver esta manera de comparar archivos porque nos será muy útil más adelante cuando lo usemos más, aunque ahora suene un poco a chino. La parte final se entiende bien. Hemos añadido una línea (***peras***) y nos aparece en verde y con un + delante, así que parece claro. Lo va entre arrobas "@@" nos dice simplemente que hemos pasado de 2 a 3 líneas.

Ahora vamos a intercalar una línea entre las dos anteriores: ***huevos*** y, como siempre, pulsamos ***<INTRO>.***

Hacemos el add y el commit pero esta vez en una sola línea con la opción -a del commit, que lo que hace es añadir todo lo que esté para añadir a la staging area y a continuación el commit:

```
> git commit -a -m "añadido huevos entre dos líneas"
```

Vamos a ver de nuevo las diferencias. Para ello hemos de saber las primeras letras del identificador, así que vamos a usar de nuevo el ***git log***, pero con una nueva opción:

```
> git log --oneline
c8c6435 añadido huevos entre dos líneas
d8219b4 Segundo commit
366bf25 Commit inicial
```

Vemos que con la opción ***--oneline***, sale más resumido. Muestra cada commit en una línea. Suficiente para lo que queremos hacer que es volver a usar el comando diff para comparar el segundo commit con el último (poned el identificador de vuestro segundo commit que no coincidirá con el que pongo yo):

```
> git diff d821
diff --git a/alimentos.txt b/alimentos.txt
index a37656a..000f4aa 100644
--- a/alimentos.txt
+++ b/alimentos.txt
@@ -1,2 +1,3 @@
 agua
+huevos
 peras
```

Queda muy claro que se ha intercalado huevos entre agua y peras.

Añadir un alias

Como vamos a utilizar mucho el comando git log --oneline, una buena idea para trabajar más cómodo y rápido es crear un "alias". Es decir una abreviatura de un comando que usamos mucho. Lo hacemos de la siguiente manera:

```
> git config --global alias.lo "log --oneline"
> git lo
c8c6435 añadido huevos entre dos líneas
d8219b4 Segundo commit
366bf25 Commit inicial
```

Ver diferencias Staging --> Último Commit | Staging --> Area Trabajo

Vamos a hacer nuevos cambios. Borremos ahora ***agua*** y añadamos ***harina*** entre ***huevos*** y ***peras***. Y hagamos el add pero sin hacer el commit todavía. Si ahora quisiéramos ver las diferencias con respecto a la última confirmación sin necesidad de hacer un nuevo commit podemos hacerlo de la siguiente forma:

```
> git diff --staged
diff --git a/alimentos.txt b/alimentos.txt
index 74cbd13..675f6a5 100644
--- a/alimentos.txt
+++ b/alimentos.txt
@@ -1,3 +1,3 @@
-agua
 huevos
+harina
 peras
```

Lo que estamos viendo es lo mismo que veríamos si hubiésemos hecho el commit y comparásemos con el anterior.

Ahora vamos a hacer un nuevo cambio en el fichero. Añadamos ***manzanas*** a la última línea. Y hagamos simplemente:

```
> git diff
diff --git a/alimentos.txt b/alimentos.txt
index 675f6a5..45ba595 100644
--- a/alimentos.txt
+++ b/alimentos.txt
@@ -1,3 +1,4 @@
 huevos
 harina
 peras
+manzanas
```

Vemos ahora las diferencias entre lo que acabamos de cambiar y lo que tenemos ahora mismo en la staging area. Hagamos por fin el add y el commit todo de una:

```
> git commit -a -m "Borrado agua, añadido harina y manzanas"
[master a0e373e] Borrado agua, añadido harina y manzanas
 1 file changed, 2 insertions(+), 1 deletion(-)

> git lo
a0e373e Borrado agua, añadido harina y manzanas
c8c6435 añadido huevos entre dos líneas
d8219b4 Segundo commit
366bf25 Commit inicial

> git diff ef36
diff --git a/alimentos.txt b/alimentos.txt
index 74cbd13..45ba595 100644
--- a/alimentos.txt
+++ b/alimentos.txt
@@ -1,3 +1,4 @@
-agua
 huevos
+harina
 peras
+manzanas
```

Volver a un commit anterior

Supongamos ahora que nos arrepentimos de los últimos cambios. ¿Hay alguna manera de volver al commit anterior? Por supuesto, la hay. Hagamos:

```
> git reset --hard c8c6
HEAD is now at c8c6435 añadido huevos entre dos lineas
```

Si ahora miramos el archivo, veremos que realmente se han deshecho todos los cambios del último commit.

Es posible que no quisiéramos llegar tan lejos y hubiéramos preferido que no nos tocara el archivo original hasta estar seguros de qué hacer con él. En lugar de usar la opción ***--hard***, podíamos haber usado cualquiera de estas otras dos opciones:

--soft -> Deshace el commit en el repositorio pero no toca ni la staging area ni el directorio de trabajo.

--mixed -> Deshace el commit en el repositorio y en la staging area pero no el directorio de trabajo. Es la opción por defecto.

Añadir más ficheros y eliminarlos de la Staging Area

Vamos ahora a crear un segundo y un tercer ficheros: ***drogueria.txt*** y ***notas.txt***

En ***droguería.txt*** creamos un par de líneas: ***lejía*** y ***papel higiénico*** y en el otro escribimos: ***No quiero hacer seguimiento de este archivo.***

Por último, en ***alimento.txt*** añadimos ***manzanas.***

Si hacemos un git status veremos que hay dos ficheros "untracked" y uno "modified". Ahora hay dos maneras de eliminar notas.txt de la staging area:

```
> git reset HEAD notas.txt
> git rm --cached notas.txt
```

De cualquiera de las dos formas eliminamos el fichero del área de trabajo pasando a estar de nuevo "untracked".

Si lo que queremos es que no nos vuelva a pasar esto. Es decir que si hacemos un git add no nos lo vuelva a añadir tenemos que crear un archivo .gitignore e incluir el nombre del archivo (o una expresión regular para tipos de archivos parecidos).

Curiosamente, una vez escrito notas.txt en el .gitignore, si hacemos un git status veremos que ya no nos aparece en absoluto notas.txt, pero sí que nos aparece el propio .gitignore para ser añadido. Si no queremos que lo añada tenemos que incluir en una línea .gitignore en el propio fichero.

```
> type .gitignore
notas.txt
.gitignore
```