Uso básico de git

git init

Habíamos definido a un repositorio como un área de almacenamiento para archivos en el cual llevamos registro de nuestro trabajo.
Habíamos visto que el comando para convertir una simple carpeta en un repositorio, es **`git init`**. Este comando crea una carpeta "oculta" llamada **`.git`**. Puede ser interesante revisar el contenido de esa carpeta, pero **no** es necesario para utilizar git, y nunca modificaremos el contenido de esa carpeta manualmente.
Es importante entender que, en este punto 1) el repositorio es completamente local a nuestra máquina, y 2) no se inició un "servidor" ni un "proceso": simplemente se creó una carpeta.
Hagamos la prueba:
Creemos una carpeta llamada "probandoGIT". Entremos en la carpeta e iniciemos el repositorio, con los siguiente comandos:

```
mkdir probandoGIT
cd probandoGIT
git init
ls
```

El último comando **`ls`** no mostrará nada, puesto que nuestro directorio de trabajo está vacío. Lo único que hay en la carpeta es la subcarpeta oculta **`.git`**. Esta subcarpeta es nuestro repositorio, que llevará registro del trabajo.

La *staging area* y el comando git status

Este comando, como su nombre lo indica, nos muestra el estado actual del repositorio.
Un determinado archivo puede tener cuatro estados:
Sin seguimiento (untracked): Son archivos que están en nuestro directorio de trabajo, pero que no forman parte del repositorio.
Sin modificar (unmodified): Son archivos que están en nuestro repositorio, y que no han sido modificados desde la última versión (desde el último *commit*).
Modificados (modified): Son archivos que están en nuestro repositorio, pero que han cambiado desde el último *commit*.
Preparados (staged): Son archivos nuevos o modificados, que están listos para ser agregados en el próximo *commit*.

El área de staging.
Lo único que se agrega al repositorio al momento de hacer commit es lo que esté en el área de staging.
Se entiende mejor si lo probamos. Supondremos que estamos en el repositorio probandoGIT, realizado en el apartado anterior.
1) Ejecutamos el comando `git status`. Debería aparecer un mensaje parecido a este[1]:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
No hay commits todavía
no hay nada para confirmar (crea/copia archivos y usa "git add" para hacerles seguimiento)
```

2) Con nuestro editor preferido, crearemos un archivo de texto llamado saludo.txt, que diga simplemente HOLA, y lo guardaremos en la carpeta probandoGIT.
3) Ejecutamos nuevamente el comando:

1 Estas salidas corresponden a un sistema GNU/Linux. Si se utiliza git bash en Windows, las salidas serán muy similares.

```
usuario@PC ~/probandoGIT $ git status
En la rama master
No hay commits todavía
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    saludo.txt
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa
"git add" para hacerles seguimiento)
```

Esta salida nos indica que ahora nuestro directorio de trabajo contiene un archivo en estado “sin seguimiento” (untracked).

4) Para que nuestro repositorio comience a llevar el registro de este archivo, usaremos el comando git add:

```
usuario@PC ~/probandoGIT $ git add saludo.txt
```

5) Este comando no produce ninguna salida, pero veamos los cambios con el comando git status:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
No hay commits todavía
Cambios a ser confirmados:
  (usa "git rm --cached <archivo>..." para sacar del área de stage)
    nuevo archivo:  saludo.txt
```

Vemos que ahora el archivo está en la staging area, es decir, está listo para ser agregado al repositorio cuando se realice el próximo commit.

Nuestro primer commit

Habíamos indicado que hacer commit es registrar un cambio en un repositorio.

6) En nuestro caso, registraremos el nuevo archivo, con el comando git commit:

```
usuario@PC ~/probandoGIT $ git commit
```

Este comando nos abrirá un editor, pidiéndonos que ingresemos el mensaje que describirá el commit. Ingresaremos “**Nuestro primer commit**” (o cualquier otro mensaje), guardamos y salimos. (Si saliéramos del editor sin ingresar un mensaje, el commit se cancelaría). En este momento, hemos guardado la primera versión de nuestro repositorio.

Al volver a la terminal, veremos que el comando muestra la siguiente salida:

```
[master (commit-raíz) fb84183] Nuestro primer commit
 1 file changed, 1 insertion(+)
 create mode 100644 saludo.txt
```

7) Ahora volvermos a ejecutar git status:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
nada para hacer commit, el árbol de trabajo esta limpio
```

Vemos este mensaje porque nada ha cambiado desde el último commit. Todos los archivos (en este caso, uno solo), están el estado “Sin modificar” (unmodified).

8) Ahora introduciremos un cambio en el archivo saludo.txt. Abramos el archivo con un editor, agreguemos una segunda línea que diga CHAU, guardemos y salgamos.

9) Volvamos a ejecutar git status:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
    (usa "git checkout -- <archivo>..." para descartar los cambios en el
directorio de trabajo)
    modificado:     saludo.txt
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

Vemos que ahora el estado del archivo saludo.txt es “Modificado” (modified), puesto que ya no coincide con la versión que estaba guardada en el último commit, y todavía no lo hemos agregado a la *staging area*.

10) Agreguémoslo a la staging area con git add:

```
usuario@PC ~/probandoGIT $ git add saludo.txt
```

11) Volvamos a ver su nueva situación:

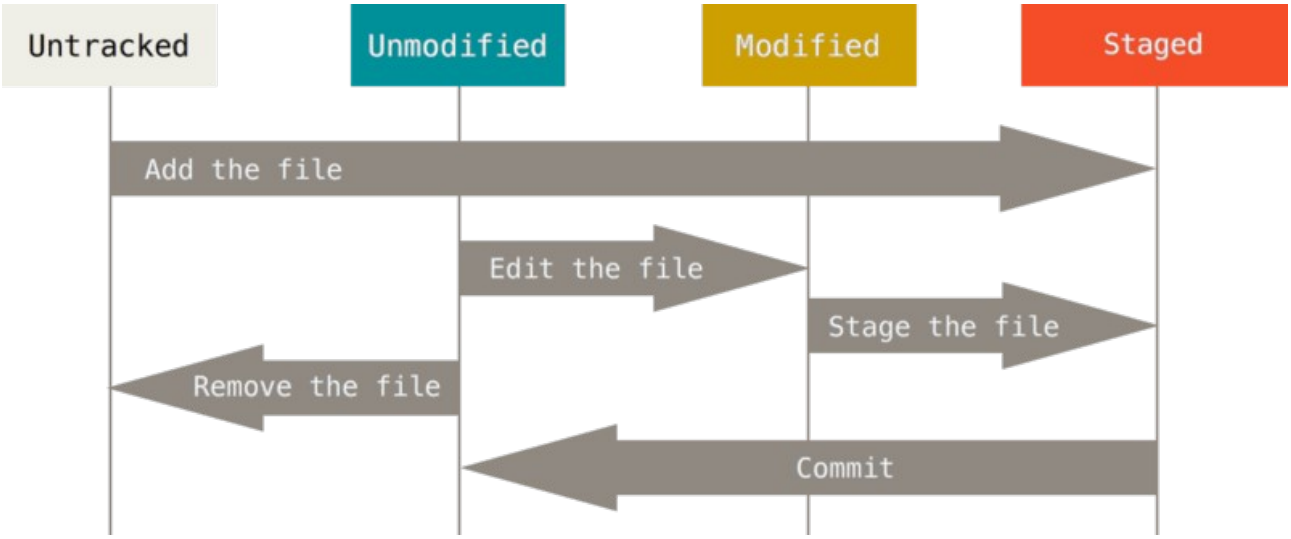
```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    modificado:     saludo.txt
```

Vemos que ahora el archivo saludo.txt, que ha sido modificado desde el último commit, está listo para ser agregado al próximo commit.

13) Por último, hagamos el segundo commit. Esta vez, utilizaremos el modificador -m, para agregar el mensaje directamente desde la línea de comandos, sin necesidad de utilizar el editor:

```
usuario@PC ~/probandoGIT $ git commit -m "Segundo commit"
[master d785346] Segundo commit
 1 file changed, 1 insertion(+)
```

El siguiente esquema[2] puede servir para comprender mejor los distintos estados en los que puede encontrarse un archivo:



Una analogía para la staging area

Podemos imaginar al repositorio como un teatro[3].

El directorio de trabajo serían los camarines, en donde los actores modifican su atuendo según las necesidades de la obra.

La *staging area* sería el momento en el que el actor está sobre el escenario, pero aún no se abrió el telón. Esto le permite hacer una última revisión antes de que los espectadores lo vean.

El momento de hacer *commit* sería cuando se abre el telón, y los espectadores ven al actor en escena.

2 Imagen tomada del libro “Pro Git” de Scott Chacon y Ben Straub. Disponible en español y con licencia libre en https://git-scm.com/book/es/v2/

3 Esta analogía está tomada del libro de Rick Umali, “Learn Git in a Month of Lunches”. https://www.manning.com/books/learn-git-in-a-month-of-lunches

Revisar el historial

Ahora que hemos hecho dos commits, tenemos ya un breve historial de trabajo.
Utilizando el comando git log, podemos ver el historial de nuestro repositorio:

```
usuario@PC ~/probandoGIT $ git log
commit d7853465e7b5b3df0c3055dd8cab05980009e9fc (HEAD -> master)
Author: fulano <fulano@correo.com>
Date:   Wed Mar 13 20:12:48 2019 -0300
    Segundo commit

commit fb8418331167d67e028b49d7a750a80181fcb16a
Author: fulano <fulano@correo.com>
Date:   Wed Mar 13 19:58:54 2019 -0300
    Nuestro primer commit
```

Vemos, en orden cronológico inverso, información importante acerca de nuestro historial: el autor del commit, el momento en que se realizó, y el mensaje asociado. Vemos también un extenso número hexadecimal, llamado SHA ID, que es un código único para cada commit.
La palabra master es la *rama* en la que nos encontramos. Git permite ramificar el flujo de trabajo, tema que se verá más adelante. Por lo pronto, nuestra única rama se llama master. La palabra HEAD, representa el commit actual. Es de esperar que sea el último commit, aunque git permite "viajar en el tiempo", ubicando HEAD en un commit anterior. Este tema también se verá más adelante.
Si queremos resumir la salida de git log, podemos utilizar el modificador --oneline:

```
usuario@PC ~/probandoGIT $ git log --oneline
d785346 (HEAD -> master) Segundo commit
fb84183 Nuestro primer commit
```

El modificador --stat, nos permite revisar **qué** cambió entre un commit y otro:

```
usuario@PC ~/probandoGIT $ git log --stat
commit d7853465e7b5b3df0c3055dd8cab05980009e9fc (HEAD -> master)
Author: fulano <fulano@correo.com>
Date:   Wed Mar 13 20:12:48 2019 -0300
    Segundo commit
 saludo.txt | 1 +
 1 file changed, 1 insertion(+)

commit fb8418331167d67e028b49d7a750a80181fcb16a
Author: fulano <fulano@correo.com>
Date:   Wed Mar 13 19:58:54 2019 -0300
    Nuestro primer commit
 saludo.txt | 1 +
 1 file changed, 1 insertion(+)
```

En ambos commits, vemos que el cambio introducido consistió en agregar una línea al archivo saludo.txt.