TUTORIAL básico “GIT”(DESDE TERMINAL)

INICIANDO/CLONANDO PROYECTO CON “GIT”

+ CONFIGURACIÓN INICIAL

Agregando NOMBRE y EMAIL para poder identificarnos como autores.
Nuestro nombre y email se configurarán para que saldrán reflejados (como autor) cada vez que hagamos un cambio desde aquí.

-En la consola de comandos:

```
//Agregamos nombre de Usuario/Autor.
$ git config --global user.name “Nuestro Nombre de Usuario”
//Agregamos email de Usuario/Autor.
$ git config --global user.email “<NuestroEmail>”
```

*PARA TRABAJAR CON “git”, TENEMOS DOS OPCIONES, O CREAMOS UN REPOSITORIO NUEVO Y LO INICIALIZAMOS PARA COMENZAR NUESTRO PROYECTO.
O BIEN CLONAMOS UN PROYECTO YA EXISTENTE Y ESTARÁ LISTO PARA TRABAJAR.*

+ CREANDO UN NUEVO PROYECTO E INICIALIZANDOLO

Para crear nuestro primer repositorio y poder empezar a trabajar con git (en caso de no tener uno creado).

-En la consola de comandos:

```
//Creamos un directorio “NUEVO” en el que trabajaremos(repositorio).
$ mkdir MiProyecto
//Accedemos a la carpeta recién creada (nuestro repositorio).
$ cd MiProyecto
//Inicializamos nuestro repositorio.
$ git init
```

+ CLONANDO UN REPOSITORIO

Nos permite hacer una copia de un repositorio que ya esté en nuestro ordenador, o descargarlo de un sitio remoto.

-En la consola de comandos:

```
//Clonando repositorio desde local,(Ej repo local en carpeta TEST).
$ git clone ~/TEST/repositorioAclonar.git

//Clonando repositorio desde remoto (Ej desde Github).
$ git clone https://github.com/azanet/Apuntes
```

TRABAJANDO CON ARCHIVOS Y EL REPOSITORIO

Para agregar un archivo al repositorio, este antes de ser “agregado” tiene que pasar por la fase de “staging”, en esta fase se van agregando todos los archivos y modificaciones para cuando ya terminemos de trabajar por ese dia/momento/motivo, realicemos un “commit” y suba todo junto a nuestro repositorio como “un solo cambio”.

+ AÑADIENDO ARCHIVOS AL “STAGING INDEX”

Esto es un “área temporal” en el que se almacenan los archivos que se han creado nuevos o realizado cambios en unos ya existentes (en la carpeta de trabajo) y hay que “prepararlos” en este área antes de subirlos a nuestro Git.

-En la consola de comandos:

```
//Para “enviar al staging” un único archivo
$ git add archivoNuevo.txt

//Para “enviar al staging” todos los archivos modificados
$ git add .
```

+ ELIMINANDO ARCHIVOS DEL “STAGING INDEX”

Para sacar un archivo de la fase “Staging” por que no queremos subirlo ya se por que nos confundimos a lo que sea realizaremos lo siguiente.

-En la consola de comandos:

```
//Para eliminar del “staging” un único archivo
$ git reset HEAD archivoAsacar.txt

//Para deshacer todos los cambios desde el último commit y no deje señal.
$ git checkout -- nombreDelArchivo_AdejarIntacto.txt
```

+ CONSULTANDO “ESTADO” DE LOS ARCHIVOS

Si no sabemos si los archivos fueron ya subidos o no, o saber en qué estado se encuentran y qué archivos, podemos utilizar.

-En la consola de comandos:

```
//Consultar el estado de los archivos de nuestra carpeta de trabajo
$ git status
```

+ EJECUTANDO COMMIT (REGISTRANDO LOS CAMBIOS EN EL REPOSITORIO)

Con este comando, ya se confirmarán los cambios realizados y terminarán de subir nuestros archivos a nuestro repositorio.
Al ejecutar este comando, se nos abrirá un editor de texto, en el que deberemos escribir los cambios que se van a realizar en el commit, ya que luego eso será nuestra guía para saber que hicimos en ese momento.

Como todo... el texto tiene unas pautas a seguir que son las siguientes:

> La primera parte de este texto, es la línea de "RESUMEN", y tendrá una extensión máxima de 50 caracteres.

> Después debe haber un salto de línea extra.

> En estas siguientes líneas escribimos el resto del mensaje con una extensión máxima de 72 caracteres por línea.

> Salimos del editor guardando, y se realizará el commit al hacer esto.

Una vez tenemos claro esto, procedemos a hacer el commit.

-En la consola de comandos:

```
//Realizamos el commit y escribimos el mensaje, corto y descriptivo
$ git commit
//Creamos el mensaje, salimos guardando y listo! Archivos Subidos!!
```

+ MOVIENDO/ELIMINANDO ARCHIVOS EN EL REPOSITORIO

Para cambiar los archivos de lugar en el repositorio, debemos cambiarlo en nuestro equipo y luego hacer un commit para realizar el cambio.

+ CONSULTANDO HISTORIAL DE REGISTRO DEL REPOSITORIO

Para consultar el historial de los commits que se han realizado en el repositorio, podemos hacer uso del comando "log", y con este, se nos mostrará el historial revelandonos parte de su hash identificativo, nombre de autor, fecha, titulo y mensajes que añadimos en su momento.

-En la consola de comandos:

```
//Nos mostrará el historial de commits realizados.
$ git log

//Ver los commits en "código corto" y con "color"(en 50 caracteres aprox.)
$ git log --oneline --decorate
```

CREANDO/TRABAJANDO CON RAMAS

Crear ramas nos sirven para trabajar en el proyecto de forma paralela por el motivo que sea.. y cuando terminemos, podemos fusionar las ramas para unificarlas y continuar trabajando con una sola rama con todos los cambios adheridos.

+ VER LAS RAMAS EXISTENTES

-En la consola de comandos:

```
//Nos mostrará las ramas que existan.
$ git branch
```

+ CREAR UNA RAMA NUEVA

-En la consola de comandos:

```
//Creará una NUEVA RAMA con el nombre que le indiquemos.
$ git branch nombreDeLaRamaNueva
```

+ CAMBIARNOS A OTRA RAMA

-En la consola de comandos:

```
//Cambiaremos a la RAMA indicada en el comando.
$ git checkout nombreDeLaRama_A_Situarnos
```

+ CREAR UNA RAMA Y CAMBIARNOS DIRECTAMENTE A ESTA

-En la consola de comandos:

```
//Crearemos y nos cambiaremos a la RAMA indicada en el comando.
$ git checkout -b nombreDeLaRama_A_Crear_Y_Situarnos
```

+ CREAR UNA RAMA Y CAMBIARNOS DIRECTAMENTE A ESTA

-En la consola de comandos:

```
//Crearemos y nos cambiaremos a la RAMA indicada en el comando.
$ git checkout -b nombreDeLaRama_A_Crear_Y_Situarnos
```

+ ELIMINAR UNA RAMA

-En la consola de comandos:

```
//Eliminaremos la RAMA indicada en el comando.
$ git branch -d nombreDeLaRama_A_ELIMINAR
```

FUSIONANDO RAMAS

Al fusionar las ramas los cambios existentes en las dos ramas, se mezclan y lo más normal es que terminen saliendo conflictos, git hará todo lo posible por arreglarlos, pero es muy probable que tengamos que arreglarlo nosotros a mano, aunque “git” nos facilitará bastante la tarea.

PARA FUSIONAR DOS RAMAS EXISTENTES!!
En primer lugar, hacemos un “checkout” y nos movemos a la rama donde queremos realizar la fusión(en la que queremos seguir definitivamente).

+ FUSIONANDO RAMAS (SIN QUE SE CREEN CONFLICTOS)

Nos movemos a la rama en la que queremos que se una todo y le pasamos el comando con el nombre de la RAMA que queremos que se inserte.
-En la consola de comandos:

```
//Fusionará las dos ramas insertará la indicada en el comando a la actual.
$ git merge NombreDeLaRama_AFusionarConLaActual
//Ahora podemos eliminar la rama que fue insertada a la principal.
```

+ FUSIONANDO RAMAS (CON CONFLICTOS)

Nos movemos a la rama en la que queremos que se una todo y le pasamos el comando con el nombre de la RAMA que queremos que se inserte.
-En la consola de comandos:

```
//Fusionará las dos ramas insertará la indicada en el comando a la actual.
$ git merge NombreDeLaRama_AFusionarConLaActual

//Podemos ver el estado y conflictos de la “fusión” con un “status”.
$ git status
```

Como dijimos en el título, en el caso de haber escrito en una línea ya existente, GIT nos alerta de que se ha producido un conflicto que el no va a reparar, porque no sabe con que linea/s nos queremos quedar, y deberemos actuar de la siguiente manera.

Al salir los conflictos, git añadirá unas marcas indicando el principio y el final de las líneas agregadas y que debemos corregir.
Eliminaremos las marcas que git ha agregado y eliminamos las líneas que NO queremos quedarnos. Una vez arreglado, lo agregamos y ya se realizará el cambio totalmente.

-En la consola de comandos:

```
//Agregamos el archivo “conflictivo” reparado.
$ git add nombreDelArchivoReparado
//El archivo subirá y la fusión se habrá terminado de realizar.
```

TRABAJANDO CON COMMITS ANTIGUOS Y TAGS

+ VOLVER A UN COMMIT ANTIGUO

Volveremos y veremos el proyecto tal y como estaba en el momento de ese commit, para dirigirnos al commit, hacemos un checkout pasandole el hash (corto o largo) del commit que queremos visitar, podremos crear otra rama desde aquí o hacer lo que queramos con el.

-En la consola de comandos:

```
//Accederemos al proyecto tal y como estaba en ese momento.
$ git checkout 0a3b561

//Para volver a la rama actual y salir de este commit Pasado.
$ git checkout NombreDeLaRama_Actual
```

+ CREAR “TAG” PARA COMMIT ACTUAL

Podemos asignar un alias/etiqueta/tag a un commit para poder trabajar con esta etiqueta y el commit en lugar de hacerlo con su HASH.

-En la consola de comandos:

```
//Creamos etiqueta para el commit Actual (podría ser la version o algo )
$ git tag NombreDeEtiqueta
```

+ CREAR “TAG” PARA COMMIT PASADO

-En la consola de comandos:

```
//Creamos etiqueta para el commit y pasamos el hash de commit antiguo
$ git tag NombreDeEtiqueta 0a3b561
```

+ VER LISTA DE “TAG” CREADOS EN EL REPOSITORIO

-En la consola de comandos:

```
//Listará los “tags” que se han creado en el repositorio
$ git tag
```

+ MOVERNOS A UN COMMIT POR SU “TAG”

-En la consola de comandos:

```
//Nos situaremos en el commit del tag indicado
$ git checkout NombreDeLaEtiqueta
```

TRABAJANDO CON REPOSITORIOS REMOTOS

Para trabajar con repositorios remotos (ej. Github) debemos en primer lugar o disponer de un repositorio clonado el cual ya estará establecido para trabajar con él en remoto, o tenemos la opción de "asociar" un repositorio con el que queremos trabajar en remoto.
El nombre con el que normalmente se agrega el repositorio remoto suele ser "origin" aunque puede ser otro

+ ASOCIANDO UN REPOSITORIO (LOCAL o REMOTO)

Podremos asociar un repositorio local, o uno remoto.
-En la consola de comandos:

```
//Agregando repositorio LOCAL
$ git remote add origin ~/TEST/miRepoLocal.git
//Agregando repositorio REMOTO
$ git remote add origin https://github/tuREPOSITORIO
```

+ RAMAS DEL REPO REMOTO

Al asociar el repositorio remoto se generarán ramas "remotas", podremos distinguirlas de las locales por ir precedidas de "/remote/origin/".
-En la consola de comandos:

```
//Para ver TODAS las RAMAS (locales y remotas)
$ git branch --all
```

+ SINCRONIZANDO CAMBIOS DE REPO "REMOTO"(origen) CON "PULL"

Para enviar los cambios a nuestro repositorio remoto, si este no está vacío, es recomendable hacer un "pull" antes de subirle nada.
Ya que si nuestros archivos no coinciden con los del Repo REMOTO, tendremos problemas.

-En la consola de comandos:

```
//Por si tenemos "una versión" muy anticuada del repositorio o tenemos cambios realizados y queremos Sincronizar sin que dañe los archivos de nuestro directorio de trabajo.
$ git pull --rebase
//Sincronizando y actualizando nuestro directorio de trabajo con el Repositorio Remoto asociado. Este sí modificará NUESTRO directorio de trabajo y lo dejará tal y como está en el repositorio.
$ git pull origin master
```

+ ENVIANDO CAMBIOS A REPO REMOTO "PUSH"

Para enviar los cambios a nuestro repositorio remoto, si este no está vacío, es recomendable hacer un "pull" antes de subirle nada.
Utilizaremos el comando PUSH, seguido del NOMBRE ASOCIADO del remoto y por último, el Nombre de la RAMA que queremos enviar al remoto.

-En la consola de comandos:

```
//Enviando cambios de RAMA "master" a Repo Remoto asociado como "origin"
$ git push origin master
```