NOTAS TÉCNICAS GITHUB

# Introducción a Git

## ¿Qué es Git?

El proyecto Git nació en 2005 de la mano de la comunidad de desarrollo del sistema operativo Linux, tras la ruptura de la sociedad entre la comunidad y un DVCS (Distributed Version Control System, un sistema para control de versiones normalmente empleado en proyectos de programación) propietario llamado BitKeeper. Algunos de los objetivos de este nuevo sistema son:

* Velocidad
* Diseño simple
* Soporte para desarrollo en paralelo
* Distribuido
* Capacidad para gestionar grandes proyectos

Lo que distingue a Git de otros sistemas de control de versiones es la manera de gestionar los datos. Los otros sistemas de control de versiones gestionan la información como los ficheros y los cambios sobre esos ficheros. Por otra parte, en Git cada vez que se sube un cambio se toma una captura de pantalla del estado de los ficheros en ese momento y almacena una referencia a la misma. Si no hay un cambio en algún fichero, Git no vuelve a almacenarlo, si no que enlaza a una versión del fichero que ya almacenada. Git plantea el sistema de datos como una sucesión de capturas.

Imagen que contiene control, monitor, vídeo, control remoto

Descripción generada automáticamente

*Ilustración 1: Control de versiones alternativo a Git*

Pantalla de celular con aplicaciones

Descripción generada automáticamente

*Ilustración 2: Control de versiones de Git*

## Características de Git

### Operaciones locales

La mayoría de las operaciones en Git sólo necesitan recursos locales. Por la forma que tiene Git de almacenar la información y dado que el histórico completo se almacena en el equipo la mayoría de operaciones se completan casi instantáneamente. Por ejemplo, se permite ver las diferencias entre versiones de un mismo archivo sin necesidad de conectarnos al servidor remoto.

### Integridad

En Git a todo se le aplica un *checksum* (número que sirve para identificar una operación) antes de almacenarse y luego es referenciada con ese *checksum.* Por tanto, significa que no es posible cambiar el contenido de un archivo sin que el sistema Git lo detecte. Git utiliza este *checksum* como identificador para almacenar su contenido.

### Git generalmente solo añade datos

Cuando se realiza una acción en Git, casi todas añaden datos a la base de datos de Git. Aún así es posible perder datos o cambios a los que no se les haya hecho un *commit*, pero una vez que se hace un *commit* al repositorio es muy complicado perder esa información de manera definitiva. Se pueden hacer experimentos y cambios sin un peligro real de realizar cambios irreversibles sobre los archivos existentes o eliminarlos.

## GitHub

Accediendo a la página web de [GitHub](https://github.com/) podemos crear una cuenta únicamente con un correo electrónico.

### Crear un repositorio

Una vez hayamos accedido a la cuenta, podemos crear un repositorio de manera sencilla. Únicamente tendremos que decidir si la cuenta es pública o privada; si queremos generar un archivo README.md, que se suele crear para contener información genérica acerca del repositorio; y si queremos generar un archivo *.gitignore*, que contendrá los archivos y directorios de los que no queremos llevar un seguimiento.

### Clonar un repositorio

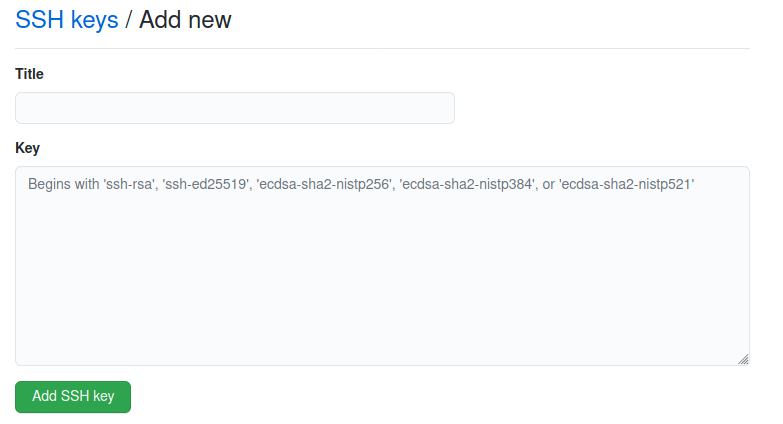
Una vez hayamos creado un repositorio, o tengamos acceso a un repositorio en el que queramos trabajar, podemos clonarlo en nuestro equipo y trabajar en él. Hay dos maneras básicas de clonar un repositorio:

* **HTTPS**. La conexión mediante la URL de HTTPS suele plantear ventajas porque simplemente requiere los detalles de la cuenta (usuario y contraseña) y permite la conexión con todos los firewalls. Plantea el problema de que si esas credenciales son robadas puede accederse a la cuenta.
* **SSH.** Requiere un poco de trabajo extra para configurar la conexión, como la configuración del SSH Keys. La ventaja de este método es que con el robo de las claves SSH, no se puede lograr el acceso completo a la cuenta. En este documento seguiremos esta vía de conexión.

#### Conexión con SO Linux

En primer lugar, hay que generar una [clave SSH](https://git-scm.com/book/es/v2/Git-en-el-Servidor-Generando-tu-clave-p%C3%BAblica-SSH) y posteriormente añadirla a la cuenta:

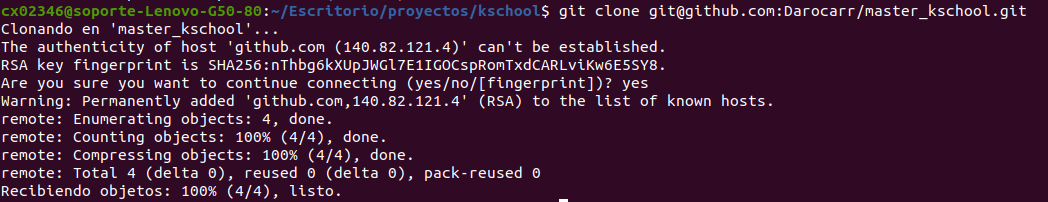




Hay que ponerle un título y copiar el texto de la clave pública. Con esto ya podremos conectarnos mediante la URL de SSH.

Para descargar el repositorio en local, debemos abrir una terminal y ejecutar el siguiente comando en el directorio en que queramos trabajar:

git clone git**@**github**.**com**:**Darocarr**/**master\_kschool**.**git



Comprobamos que ahora tenemos la carpeta con los archivos del repositorio.



Ya tenemos el entorno configurado para trabajar.

#### Conexión con SO Windows

Para generar una clave SSH en Windows podemos seguir [este tutorial](https://www.redeszone.net/tutoriales/servidores/como-crear-clave-ssh-windows-10/). Una vez generada, debemos ir a la ubicación del archivo y añadir la clave pública en GitHub como en el apartado de conexión en Linux.

Para trabajar con Git en Windows es recomendable tener una consola Git, como puede ser GitBash, que suele venir instalada por defecto. Si no estuviera instalada, puede instalarse desde [aquí](https://gitforwindows.org/).

Una vez descargado e instalado, podemos abrir la consola de GitBash y ejecutar los mismos comandos que en el apartado de Linux para realizar el clonado del repositorio y la comprobación de su correcta importación a nuestro equipo.­­­­

## Directorio de trabajo

En Git, los ficheros pueden encontrarse en dos estados básicos: registrado o no registrado.

Aquellos ficheros no registrados son de los que Git no tiene ninguna constancia, aquellos ficheros que no se encontraban en la última instantánea del estado de los ficheros. Cuando clonamos un repositorio, todos los archivos estarán registrados y sin modificar porque Git acaba de obtenerlos.

Los ficheros registrados pueden encontrarse no modificados, modificados o *staged*. Conforme modificamos y aplicamos acciones sobre los ficheros, éstos van cambiando de estado. Podemos visualizar las transiciones entre estados en el siguiente diagrama:

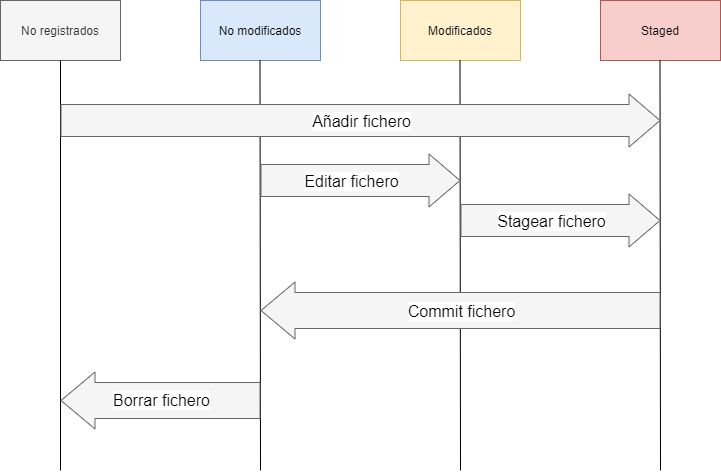


Ilustración 3: Diagrama con los estados posible de los ficheros

* En primer lugar, si no tenemos registrado un fichero, ya sea porque lo obtenemos mediante la clonación del repositorio o porque lo hemos creado nuevo desde cero, el archivo pasa a estar *staged*, listo para ser *commiteado*.
* Cuando un archivo se encuentra registrado y sin modificar, puede permanecer así si no necesitamos aplicar cambios sobre él. Si el archivo se ve modificado, debemos añadirlo al área *staged* (más adelante en la documentación se aportarán más detalles al respecto).
* Una vez que se hayan añadido al área *staged* los archivos sobre los que hayamos hecho cambios y que queramos guardar, debemos *commitear* los cambios para preservarlos y volver al estado de “no modificado”.
* Si borramos un fichero, perdemos el registro sobre él. Es posible recuperar el fichero retrocediendo a una instantánea anterior, es decir, a un *commit* anterior.

# Comandos básicos de Git

## Iniciar un repositorio desde local

Si tenemos un directorio local con los archivos con los que queremos conformar un repositorio, podemos incializarlo con el comando:

git init

Una vez hecho esto, se generará un directorio *.git* en el que se registrarán los cambios que hagamos en el repositorio. Luego hemos de añadir los archivos que queramos que queden registrados:

git add **\*.**c

git add LICENSE

git commit **-**m 'Initial project version'