UT3 - Actividad 02 – Primeros comandos GIT

Contenido

[1 ﻿ Introducción 2](#_Toc148524604)

[1.1 Ventajas de GIT 2](#_Toc148524605)

[2 Configuración 3](#_Toc148524606)

[2.1 Primeros comandos 3](#_Toc148524607)

[2.2 Configurando 4](#_Toc148524608)

[2.2.1 Establecer nombre de usuario 4](#_Toc148524609)

[2.2.2 Configurar la dirección de correo del usuario 4](#_Toc148524610)

[2.2.3 Activar colores de la interfaz 4](#_Toc148524611)

[2.2.4 Ver las configuraciones 4](#_Toc148524612)

[3 Creando el primer repositorio 4](#_Toc148524613)

[4 Desde VSCode utiliza la terminal de git 9](#_Toc148524614)

# ﻿ Introducción

**Git es un software de control de versiones** diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo **coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código**.

+Info: <https://jsamos.es/siscd/git.html>

## Ventajas de GIT

**1- Facilita el trabajo colaborativo**: Distintos programadores pueden estar editando el mismo archivo, o versiones distintas del mismo archivo, y todos los cambios serán reflejados en el documento final.

**2- Reduce considerablemente los tiempos de deploy** (despliegue) de un proyecto, al subir solamente los cambios (no los archivos cambiados, sólo los cambios!), que en Git se conoce como "diff": las diferencias entre la versión local (la que estás trabajando) y la "master" que está en el servidor central.

**3- Permite regresar a versiones anteriores de forma sencilla y muy rápida**. En caso de haber realizado cambios negativos en un proyecto en producción, volver a la última versión estable es un simple comando, que retrocede a su estado previo todos los cambios realizados en la última modificación. Esto puede hacerse hacia cualquier versión del proyecto, sin importar la cantidad o calidad de los cambios posteriores.

**4- Permite generar flujos de trabajo** que facilitan el desarrollo y mantenimiento de proyectos de gran tamaño. El ecosistema Git es increíble, y agrega un montón de herramientas a nuestra disposición para facilitarnos el trabajo, de forma robusta, rápida y profesional.

A través de los “hooks” de Git, los distintos servicios pueden detectar cambios en el historial de versiones y realizar acciones automáticas (como actualizar los archivos en el servidor o ejecutar una suite de tests y enviarnos su resultado), ¡dejándonos tiempo libre para cosas más productivas!

**5- Las "branches" o ramas, permiten trabajar con una base de código** **paralela al proyecto** en sí, donde podemos corregir bugs o desarrollar nuevas características para el producto sin afectar el "master", pero manteniendo todas las ventajas de usar un sistema de control de versiones. Una vez que estamos contentos con nuestro "branch", podemos combinarlo con el "master" o, en lenguaje Git, hacer un "merge".

**6- Empezar a trabajar desde otro entorno es tan fácil como "clonar" el proyecto** a tu nuevo entorno, trabajar sobre los archivos que se quieran, y subir los cambios al "master" o a una "branch".

**7- Sistema de etiquetas, para etiquetar las distintas versiones del proyecto**. Esto es un marcador a una versión específica del proyecto, sólo que en lugar de tener distintos backups de versiones anteriores, apuntamos a distintas versiones dentro de la misma base de código.

A continuación, completa este manual con tus capturas tras la ejecución de los comandos, sustituyendo cuando sea necesario las imágenes por las tuyas.

# Configuración

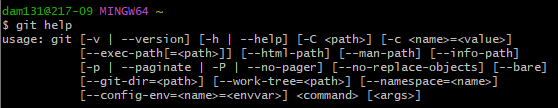
## Primeros comandos

git –version



Nos muestra la versión de git

git help



Muestra ayuda sobre los comandos

git help “*comando*”

git help

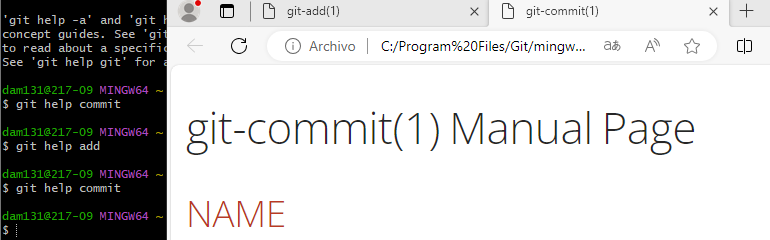
git help commit

git help add

Muestra ayuda sobre un comando concreto. Ejemplo:

git help commit

git help add



## Configurando

### Establecer nombre de usuario

git config --global user.name “victor pb”



### Configurar la dirección de correo del usuario

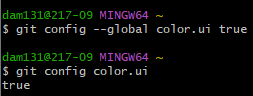
git config --global user.email [victorpb@educastur.org](mailto:victorpb@educastur.org)



Se usa para anotar que usuario realizó cada acción

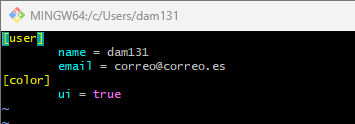
### Activar colores de la interfaz

git config --global color.ui true



### Ver las configuraciones

git config --global –e



Salir tecleando :qa

# Creando el primer repositorio

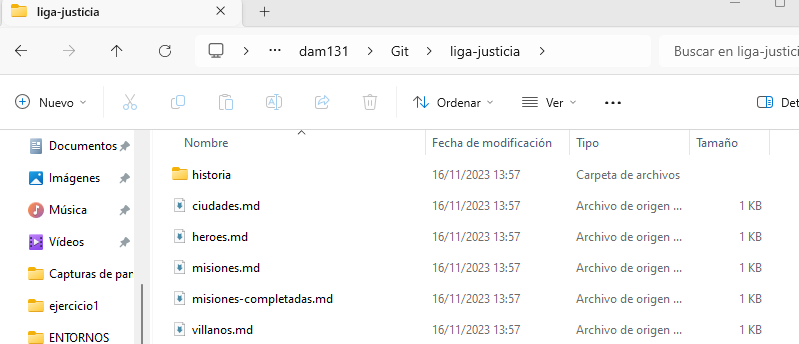
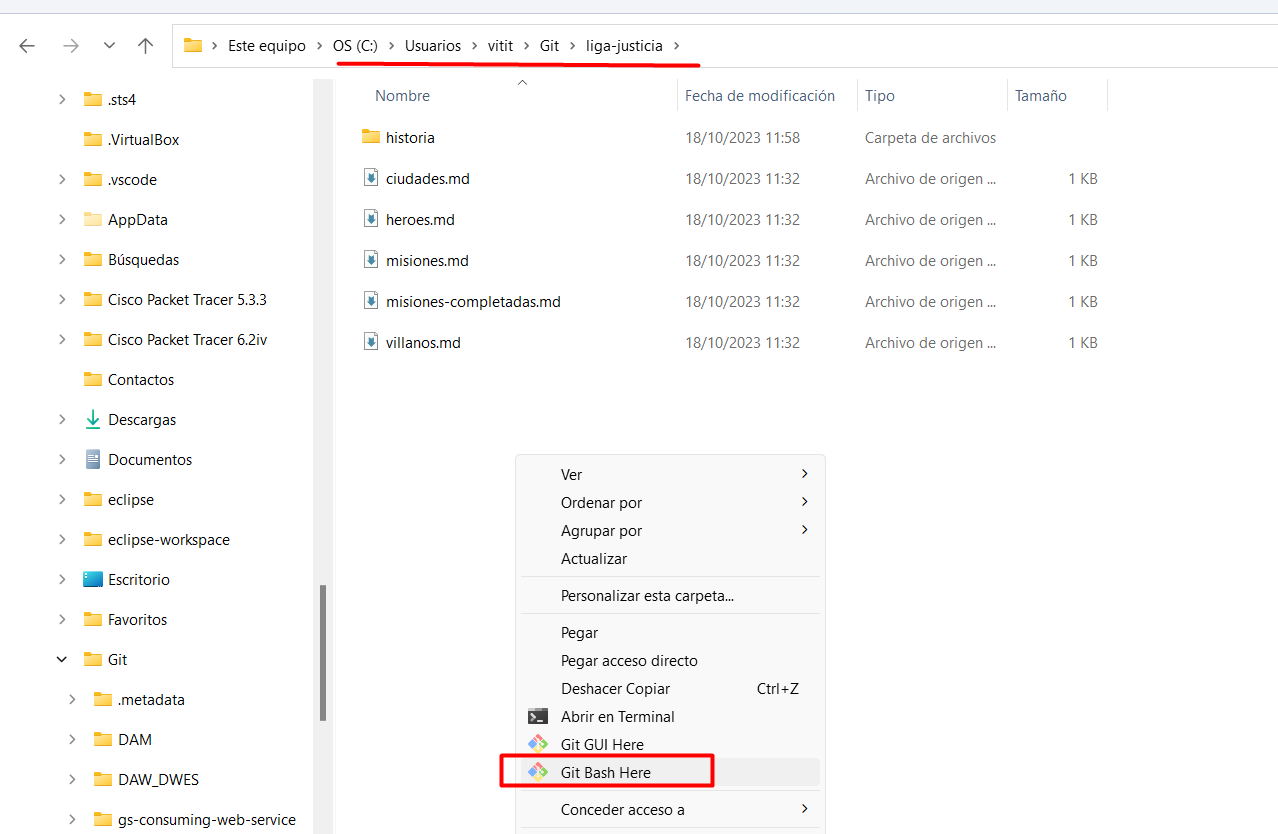
Un repositorio de Git es un almacenamiento virtual de nuestro proyecto. Permite guardar versiones del código a las que se puede acceder cuando se necesite.

**Con la instalación de Git se habrá creado una carpeta C:\users\Tu\_usuario\Git donde por defecto se crean los repositorios de código local. Descomprime el codigo**

Extraemos en el directorio anterior, el directorio “liga-justicia.zip” que contiene un conjunto de ficheros .md o archivos Markdown, son ficheros de texto plano para la descripción de elementos en un directorio. +info : <https://programminghistorian.org/es/lecciones/introduccion-a-markdown>

Desde la línea de comandos GIT BASH nos situamos en la carpeta que será nuestro repositorio:

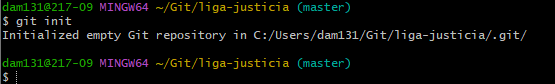
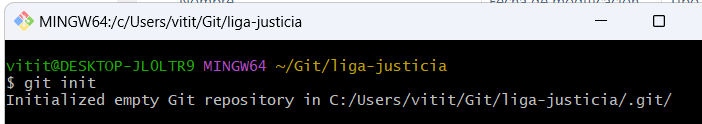
Añade una captura



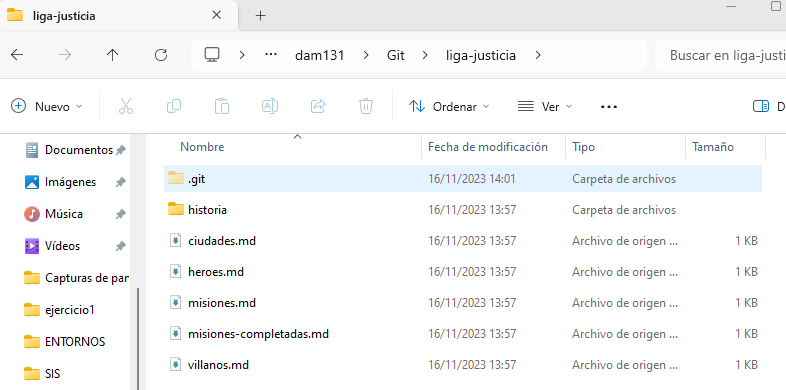
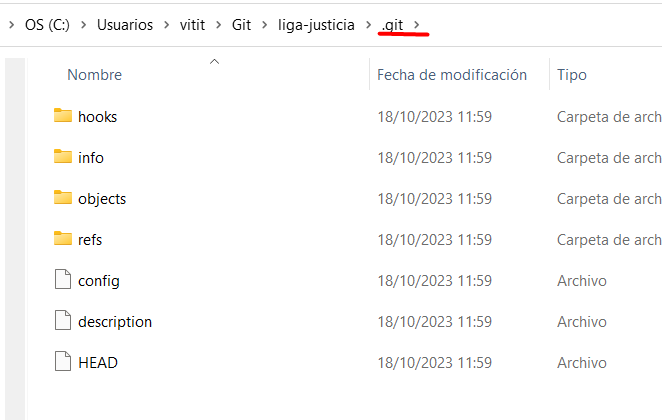
En este caso la carpeta que será nuestro repositorio es liga-justicia que es la que contiene el código fuente.

A continuación, iniciamos el repositorio con el comando:

git init



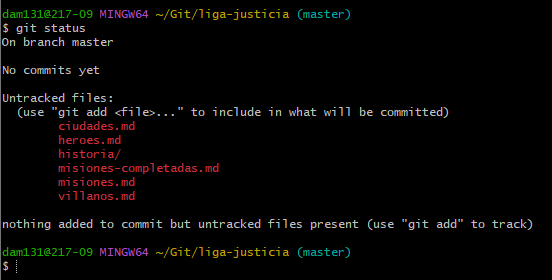
Dentro de la carpeta donde hemos creado el repositorio se crea la **carpeta oculta. git**:



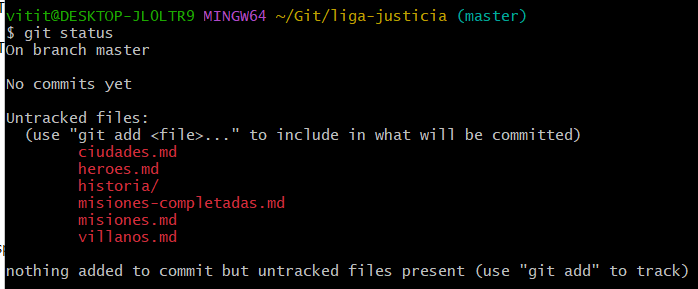
Este directorio no se debe borrar porque es el directorio de trabajo de git.

Vamos a ver el estado de nuestro repositorio:

git status



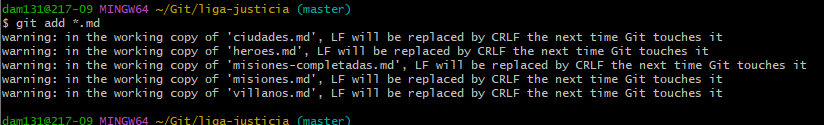
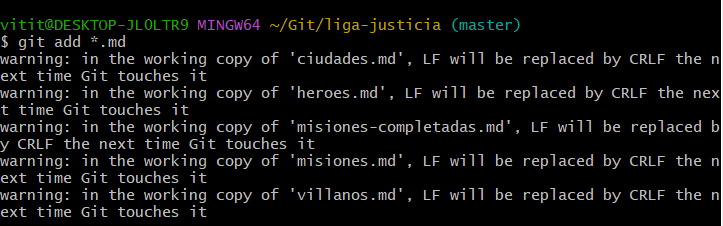
Vemos que estamos trabajando en la rama master que es la rama principal



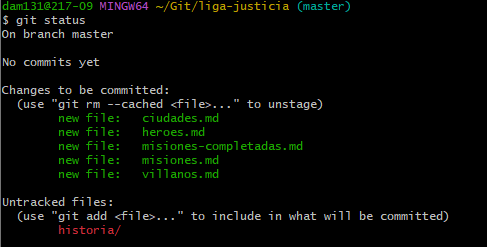
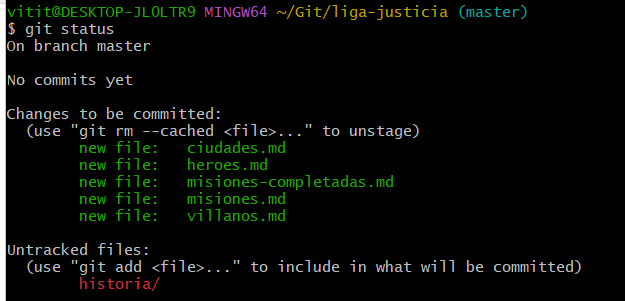
Además, git nos indica que no estamos haciendo seguimiento de algunos archivos. En la terminología de Git supone añadirlos al escenario de git **“stage”.**

Vamos a indicar a git que queremos hacer seguimiento de los archivos con extensión .md:

git add \*.md



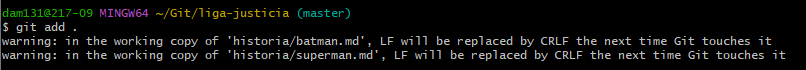
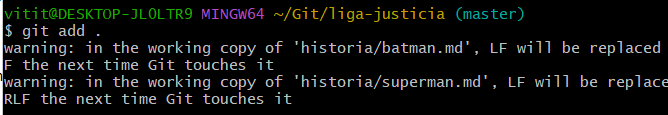
Si vemos otra vez el estado con git status:



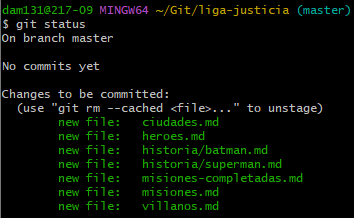
Vemos que ahora se nos muestran muchos archivos de los que hacemos seguimiento (los que están en color verde) ya que lo que hemos añadido en vez de un fichero suelto es un directorio y se han añadido todos los archivos dentro de él.

Si queremos añadir todo el contenido de la carpeta actual podemos usar el comando:

git add .



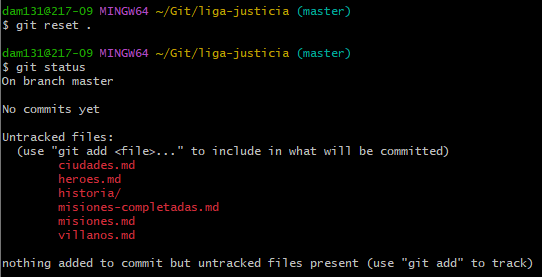
El punto indica el directorio actual.



De la misma forma que podemos añadir seguimiento de archivos podemos eliminarlo con el comando:

git reset <nombre archivo>

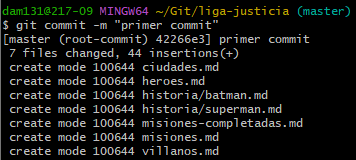
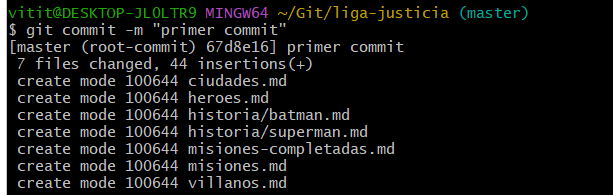
Por ejemplo los “reseteo” todos:



Cuando tengamos seleccionados los archivos de los que queremos hacer seguimiento procedemos a realizar una instantánea, es decir guardar el estado actual de los archivos. Para ello usamos el comando commit:

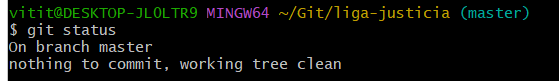
git commit –m “Primer commit”

La opción m se utiliza para indicar el mensaje identificador del commit



La opción m y el texto entre comillas sirve para identificar la instantánea tomada.

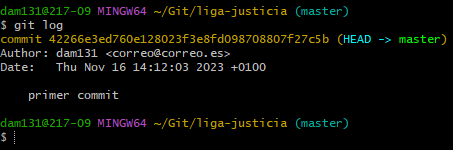
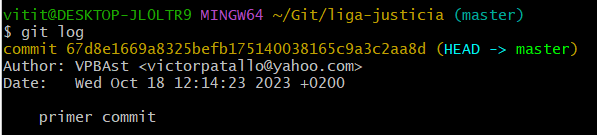
Si usamos git status después de hacer el commit:



Nos indica que todo lo que habíamos configurado para hacer el seguimiento está guardado y no hay cambios que necesiten hacer commit (“nothing to commit”).

Podemos consultar el histórico de commits realizados mediante el comando:

git log



# Desde VSCode utiliza la terminal de git

Abre el directorio de nuestro repositorio desde VSCode: *File > open Folder …*

Modifica uno de los archivos, por ejemplo añade una ciudad en ciudades.md y guárdalo.

Desde la terminal git bash de VSCode comprueba el estado del repositorio, deberá aparecer los ficheros modificados



Si queremos revertir o deshacer todos los cambios podemos utilizar el siguiente comando que reconstruye el proyecto al estado del último commit realizado.

git checkout -- .

