Ver 191016  
Maximiliano Sepúlveda

# Introducción

El objetivo de estos apuntes es más que nada para ayudarte a como dar los primeros pasos en varios ámbitos de la programación, no obstante, no es una guía completa. En otras palabras, estos apuntes te servirán para dar los primeros pasos y dejar todo preparado y funcional para desarrollar algún proyecto. Al fin y al cabo, el resto lo deberás aprender tu.

Estos apuntes los he ido construyendo a medida que voy avanzando en experiencia, puede que esté incompleto.

En esta especie de guía voy a intentar explicar lo mas “resumido” posible, pero intentando mantener cierto sentido de los contenidos que voy mencionando. La idea de esto es también que tu aprendas el “por qué” de las cosas en programación. Todo tiene un sentido, y voy a intentar ser lo más fiel para mantenerlo.

También intentare no mantener un lenguaje extremadamente formal, tampoco es la idea que te ahogues en aburrimiento, intentare escribir de forma natural.

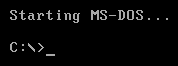
Sin mas que decir, comencemos desde el inicio…

# Terminal

El terminal (también llamado consola, símbolo del sistema, o shell) es el lugar final (de ahí su nombre) donde el usuario puede interactuar directamente con los servicios del sistema operativo en su completitud, en base a instrucciones definidas. Cada sistema operativo tiene (o debería tener, al menos como norma general) un terminal disponible.

## Un poco de cultura

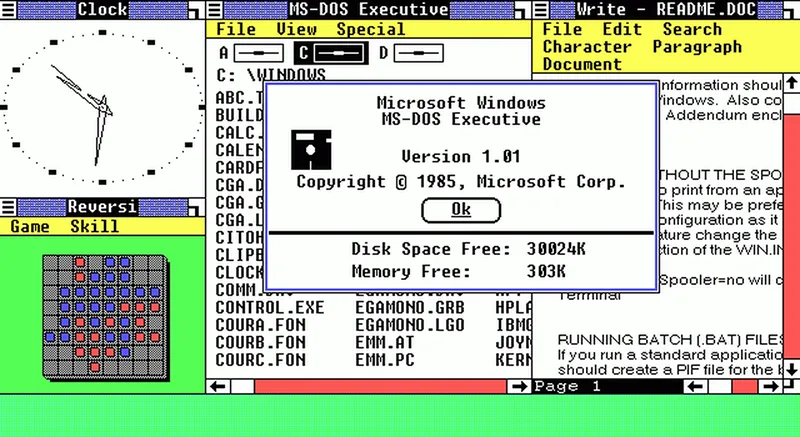
En los inicios, antes de Windows tal como lo conocemos, existía MS-DOS (Microsoft Disk Operating System), uno de los primeros sistemas operativos en ser comercialmente populares en su época (1980).



MS-DOS no tenía una interfaz gráfica: no existían cuadros de dialogo, no existían botones, no había puntero, etc. Todo lo que se hacia allí, ya sea ejecutar programas, crear carpetas, crear archivos, crear documentos, etc., era en base a texto, precisamente instrucciones o “comandos”.

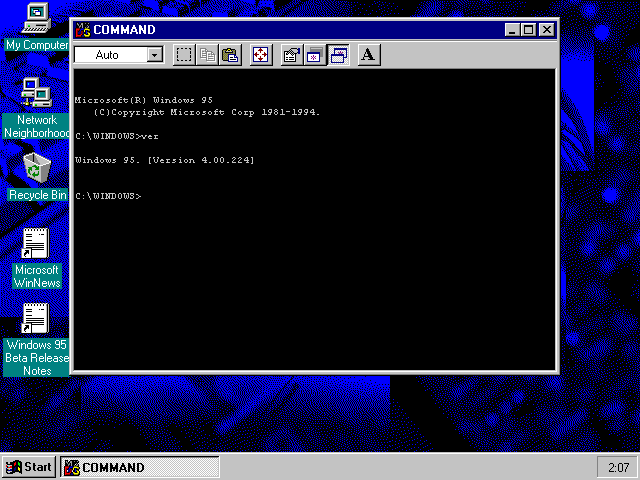
Por eso, podemos decir que MS-DOS es un “sistema operativo basado en una interfaz de línea de comandos”.

Luego en 1985, Microsoft finalmente le daría “vida” y color a MS-DOS con una interfaz gráfica, con el fin de agilizar la productividad, siendo más familiar con el usuario, haciendo mucho más sencillo realizar tareas diarias: Microsoft Windows.



Aunque parezca un sistema completamente nuevo, sigue siendo MS-DOS, pero esta vez, la capa de interfaz gráfica en la que se ejecuta encima es Windows.

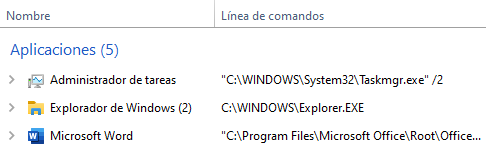
Luego, en 1995, llegaría algo que reemplazaría a MS-DOS como sistema operativo, y reemplazaría a Windows 3.1 como interfaz gráfica de ese momento: Microsoft Windows 95.



Obviamente, no podían deshacerse de MS-DOS, ya que básicamente es el pilar del sistema operativo. Por esa razón, a pesar de que tenías una interfaz gráfica llamativa y fácil de usar, aun podías seguir usando la línea de comandos por si la necesitabas.

Te preguntaras ¿Por qué estoy explicando esto? Simple: mi objetivo al explicar esto, es que entiendas un poco más a fondo el “concepto” de “terminal” en cualquier sistema operativo, y por qué es tan importante su uso.

Considerando que has leído todo hasta aquí, quisiera que pienses lo siguiente: la mayoría de las cosas que haces en un sistema operativo ya sea manipular archivos, crear carpetas, ejecutar algún programa, y todo el resto; en el fondo, se ejecuta en base a una línea de comando, una instrucción, una orden al sistema, y el sistema en respuesta, simplemente hace lo que el usuario le ordena.



Y no lo estoy diciendo por decir, es que así funciona: simples ordenes, y nada más.

## Sintaxis

En este apunte, utilizare la consola de Windows, también llamado “símbolo del sistema”, en inglés “command prompt”, o de forma abreviada “CMD” (abreviación de “command”).

**🔹Nota**: A partir de ahora me referiré a la consola de Windows como “CMD”.

Antes de comenzar con los comandos, voy a explicar de forma breve como se compone una línea de comando. Esta se divide en dos partes:



Ubicación (C:\Users\Max>) e instrucción (echo Hola mundo).

**Ubicación**: representa el lugar donde te encuentras actualmente en el terminal, de esto dependes para poder acceder a otros archivos que estén a “tu alcance”, en otras palabras, solo puedes acceder a los archivos que están en tu ubicación.

“C:\Users\Max”, en donde “C:” se refiere a la partición del disco, y cada backslash (\) o slash (/) refiere a un nivel más dentro de cada carpeta, siendo el primero la carpeta raíz (o root del disco).

Un ejemplo de esquema de las carpetas puede ser así:

**C**:   
├ Program Files  
├ **Users**  
│ ├ Default  
│ ├ **Max**  
│ │ ├ **Documents**  
│ │ │ ├ miDocumento.docx  
│ │ │ └ archivoImportante.txt  
│ │ ├ Downloads  
│ │ └ Desktop  
│ └ Public  
└ Windows

Puedes leer “C:\Users\Max\Documents” como: “La carpeta Documents, que está dentro de Max, que está dentro de Users, que está dentro de C:”

**Instrucción**: después del símbolo de “esperando instrucción” es donde corresponde tu comando, cuando lo veas, es porque el sistema esta listo para recibir una orden.

## Comandos

A continuación, intentare mencionar alguno de los comandos más comunes que se utilizan en la consola y brevemente explicar cómo funcionan.

Te recomiendo que mientras lees esto, también lo intentes tu. Ejecuta los comandos en una consola, y mira lo que pasa. ¡Experimenta!

dir

El comando “dir” hace alusión a “directorio”, y te muestra una lista con todos los archivos y carpetas que hay en tu ubicación.

md [nombre]

El comando “md” hace alusión a “make directory” (crear directorio). Crea una carpeta con el nombre que le ingreses en la ubicación actual.

cd [carpeta]

El comando “cd” hace alusión a “cambiar directorio”, y te permite moverte a una carpeta que esté en tu ubicación.

cd ..

Si en vez de poner el nombre de una carpeta, pones dos puntos consecutivos, vas a moverte a “una carpeta arriba”. Por ejemplo, si estoy en “C:\Users\Max”, al usar ese comando, voy a moverme a “C:\Users”.

echo [texto]

El comando “echo” alusión a “eco” te permite imprimir texto en pantalla.

echo > [archivo]

Con este comando, podemos crear un archivo vacío en la ubicación actual. Por ejemplo: “echo > nuevoArchivo.txt”.

**🔹Nota**: El símbolo “>” se llama “redirección”, y su función es redireccionar toda la salida de un comando a un archivo que se creara en la ubicación. Por esa razón, si usamos un “echo” vacío, se creará un archivo vacío.

del [archivo]

Con este comando, alusión a “delete”, puedes borrar un archivo. Si pones el nombre de una carpeta, solamente los archivos se eliminarán, no la carpeta en sí, para ello existe otro comando.

rd [carpeta]

Con este comando, alusión a “remove directory”, puedes eliminar una carpeta. No funciona con carpetas que tengan archivos dentro.

rd /s [carpeta]

Para forzar eliminar una carpeta, y todo su contenido, se usa el argumento “/s”.

code [archivo]

Si utilizas [Visual Studio Code](https://code.visualstudio.com/), con este comando puedes abrir o crear un archivo.

**🔸Importante**: Asegúrate que hayas seleccionado “Add VSCode to Path” en la instalación.

rm [archivo]

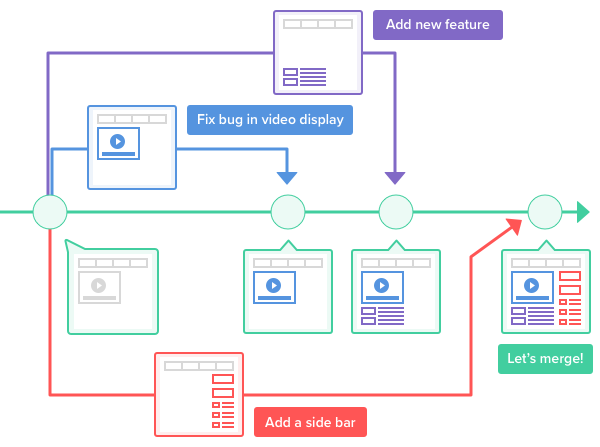
En otras consolas como Git Bash (a continuación, lo vamos a ver), el comando “remove” te permite borrar archivos o carpetas.

Si la carpeta que intentas eliminar posee contenidos, no te va a dejar eliminarlo, para eliminar una carpeta y todo su contenido, usa el siguiente comando:

rm -f [carpeta]

# Git

Git es un sistema de control de versiones muy útil para poder guardar el estado actual de algún proyecto que involucra muchos archivos, o colaboración entre varios programadores en un mismo proyecto, con cada uno trabajando en un área distinta, sin afectar a los demás.



Uno de los servicios de alojamiento de repositorios Git en la nube más famoso es GitHub, permitiéndote poder trabajar con programadores en todo el mundo.

**🔹Nota**: Por favor **no confundir Git con GitHub**. Git es el sistema de control de versiones, en cambio GitHub es el servicio que nos permite guardar nuestros repositorios en la nube. Ten eso en cuenta

## Filosofía

Desde el inicio de la programación es que existen distintas formas de manejar las versiones de los programas que se crean. Pero la mayoría de ellos eran lentos, necesitaban mas de un computador a la vez, se requería hacer copias de archivos (que cuando eran grandes no era muy eficiente que digamos), etc. Git es un sistema sencillo, rápido y que es capaz de manejar proyectos muy grandes sin mayores complicaciones.

La gracia de Git frente a otros sistemas es que usa copias instantáneas de todos los archivos que modificamos en nuestro proyecto, siempre de forma local, es decir, en nuestro PC, esto nos asegura que nada se pierda y sea un sistema integro.

Antes de comenzar a usar Git, es necesario ver como es que este sistema maneja los archivos y como se les denomina.

Cada archivo puede estar en uno de los siguientes estados: modificado (modified), preparado (staged) o confirmado (commited).

Un archivo “**modificado**” quiere decir que el archivo se ha modificado (valga la redundancia) desde la ultima vez que se confirmó.

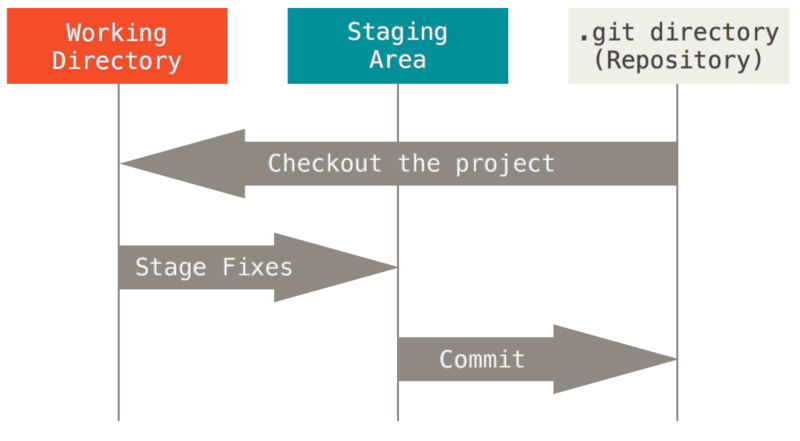
Un archivo “**preparado**” es aquel archivo que ha sido modificado, y que ha sido marcado para que, en la próxima confirmación, tales modificaciones sean efectivas en la nueva versión.

Por último, un archivo “**confirmado**” es aquel archivo que ya esta listo y esta guardado de forma segura dentro de la base de datos de Git. Si luego se desea cambiar algunos datos de tales archivos confirmados, volverán al estado “modificado”.

Por tal razón, Git se separa en 3 áreas: el área de trabajo, el área de preparación, y la base de datos o repositorio.

El **área de trabajo** es básicamente tu carpeta de trabajo, donde tienes tus archivos. El **área de preparación** es donde están todos los archivos preparados para la confirmación. Y finalmente en el **repositorio** es donde están todos los archivos confirmados, con sus respectivas versiones almacenadas de forma segura.

Resumiendo, la forma de trabajar con Git es la siguiente:



1. Modificamos los archivos que veamos necesarios para nuestro proyecto.
2. Preparamos los archivos que deseamos que sean efectivos en la próxima confirmación.
3. Confirmamos los cambios y se crea una nueva instantánea del proyecto.

## Instalando Git

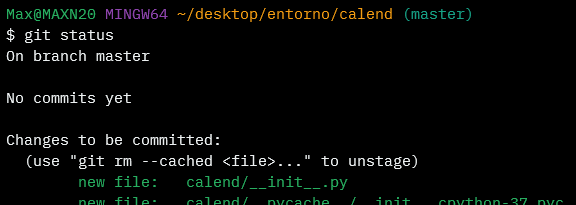
Para usar Git, es necesario instalarlo en nuestro PC, para eso, vamos a la pagina oficial de Git: <https://git-scm.com>.



Allí tocamos “Download”, y lo instalamos. En la pantalla de instalación, dejamos todo como esta predeterminado, no es necesario cambiar nada. Los que nos interesa es “Git Bash”.

## Git Bash

En pocas palabras, Git Bash es la consola que nos permite controlar los comandos de Git, por ejemplo, iniciar un repositorio, lanzar versiones, etc.



Si nos damos cuenta, tiene un aspecto muy parecido a CMD; es porque es un terminal, pero la diferencia es que este ultimo posee los comandos necesarios para que podamos controlar nuestro repositorio.

**🔹Nota**: Algunos comandos de CMD como “cd” o “dir” si funcionan en Git Bash. A estas alturas, puede que ya se te sea familiar como usarlos. Nótese [Comandos](#_Comandos) para más info.

## Configuración inicial

Ya instalado, necesitamos, primero que nada, configurar nuestro nombre de usuario y nuestro email.

Esto es necesario ya que cuando confirmamos cambios en las versiones, esta información se utilizará para dar conocimiento del autor, muy útil cuando se trata de colaboración con distintos programadores.

En Git Bash, para establecer nuestro nombre, hacemos:

git config --global user.name “Mi Nombre”

Para establecer nuestro email, hacemos:

git config --global user.email “example@dom.com”

**🔹Nota**: El argumento “--global” significa que esos ajustes solo aplican al usuario actual del PC.

## Comenzando un repo

Estando dentro de una carpeta en Git Bash debemos inicializar el sistema Git. Para ello, escribimos el siguiente comando:

git init

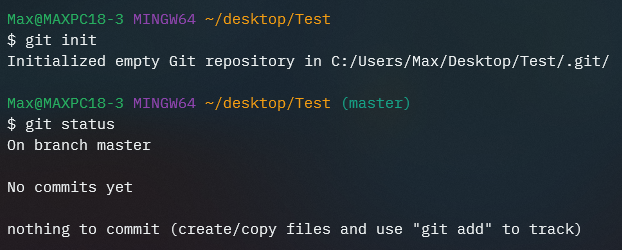
**🔷Nota**: en la ubicación de la consola, el signo “~” se refiere a tu carpeta personal (en Windows, ubicada en C:\Users”, recuerda que puedes usar “cd”.

De esta forma, se crea una carpeta oculta llamada “.git”, allí es donde se almacenan todas nuestras versiones y las instantáneas de nuestro proyecto.

El paréntesis que dice “master” hace alusión al “branch” o “rama” de nuestra versión, esto lo veremos más adelante.

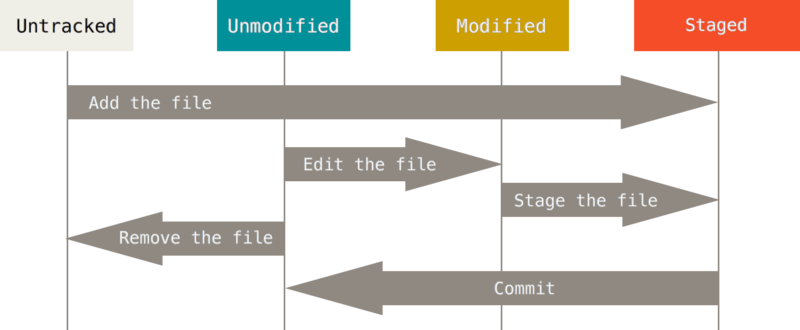
Para ver el estado de Git, utilizamos:

git status



Como vemos, no hay nada rastreado ni confirmado aún. Es porque recién comenzamos en una carpeta vacía.

El ciclo de vida de los archivos en Git sigue el siguiente diagrama:

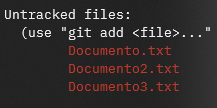


1. Los archivos que aun no son agregados al repositorio están “untracked” (sin rastrear).
2. Al añadirlos al repo, se vuelven al estado “staged”, listos para la confirmación.
3. Una vez que se confirma, si ese archivo se edita, pasa a estar en estado “modified” hasta que se vuelva a preparar para otra confirmación de versión.

Veamos ahora como funcionan los comandos para aquello.

## Preparar archivos

Si creamos archivos y volvemos a ejecutar el comando “git status”…



Vemos que los archivos que hemos creado están en estado “untracked”, esto significa que Git no está rastreando esos archivos ya que **aun no los hemos añadido al repositorio**.

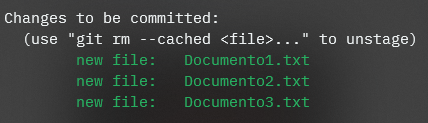
Para añadir archivos al repositorio, usamos:

git add [carpeta/archivo]

Si ahora añadimos “Documento2.txt” al repositorio ¿Qué pasa?



Ahora “Documento2.txt” está preparado y listo para la confirmación, pero el resto no. Preparemos los 2 archivos que quedan.

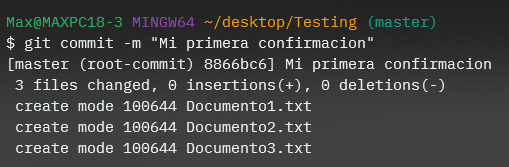


## Confirmar archivos

Teniendo nuestros archivos preparados, podemos confirmar los cambios, para ello usamos el comando:

git commit -m “Mensaje”

El parámetro “-m” se refiere al mensaje que va adjunto junto con la confirmación, y es obligatorio.



Ahora Git acaba de guardar el estado de los archivos **en el momento que los preparamos** (es decir, cuando usamos “git add” sobre ellos).

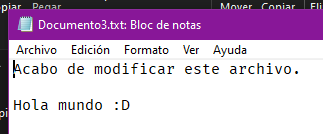
**🔷Nota**: en algún momento puedes que tengas demasiados archivos que hayas modificado, y preparalas uno por uno puede ser muy tedioso. En esos casos, puedes usar el parámetro “-a” para hacer que todos los archivos que no estén preparados pasen a estar preparados para luego ser confirmados sin necesidad de usar “git add”.

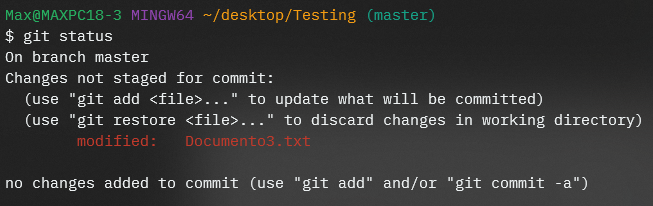
git commit -a -m “Mensaje”

## Archivos modificados

Ya que confirmamos los archivos que teníamos en ese momento, Git los considera como “no modificados”. Si ejecutamos “git status” ahora, no mostrará nada, porque ningún archivo se ha modificado desde que confirmamos los cambios.

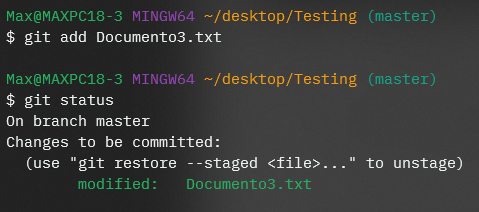
Si modificamos el archivo “Documento3.txt” y ejecutamos “git status”…





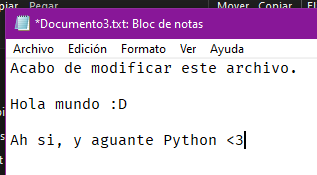
El archivo “Documento3.txt” aparece como “modificado” y “no preparado”.

Git se dio cuenta que hemos modificado el archivo, pero aun no hemos preparado nada, intentemos prepararlo ya que estamos.

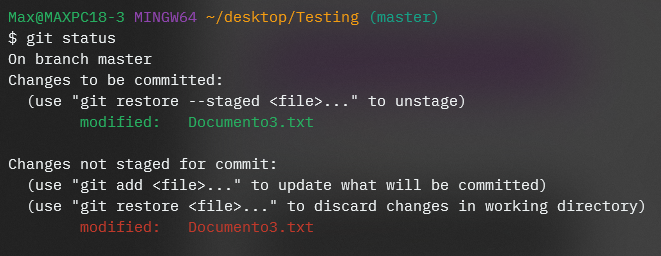


Ahora esta preparado. Si confirmamos los cambios, como he dicho antes: Git guardara el estado de ese archivo en el momento que usamos “git add” sobre este.

¡Oh no! Se me olvido agregar algo mas a mi documento, voy a volver a modificarlo.



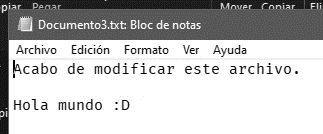
Si ejecutamos “git status” ahora, pasara algo curioso…



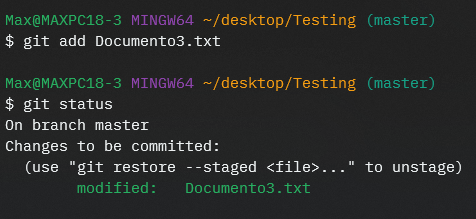
¡Nuestro archivo esta preparado y no preparado a la vez! ¿Por que?

La razón de esto es que Git toma el estado del archivo en el preciso momento cuando ejecutamos “git add”, es muy importante tener esto en cuenta, por eso lo repito varias veces.

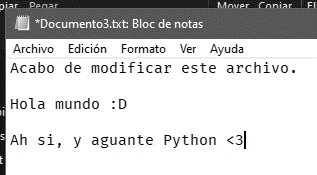
Es decir, Git ahora mismo posee la versión del archivo antes de que lo modificáramos por segunda vez. Si confirmáramos ahora mismo, Git no va a guardar la versión modificada que dice “aguante Python”, sino este:



Si queremos que Git tome en cuenta la nueva versión de nuestro archivo, hay que volver a “prepararlo”.



Ahora si Git esta tomando en cuenta la segunda modificación del archivo:

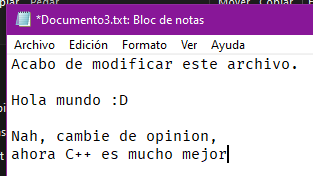


## Diferencias

Sabrás que usar “git status” es muy útil para saber que archivos están preparados y cuáles no.

Por ejemplo, cuando preparamos “Documento3.txt”, pero después lo modificamos, volvió a estar como no preparado, pero exactamente ¿Qué se modificó?

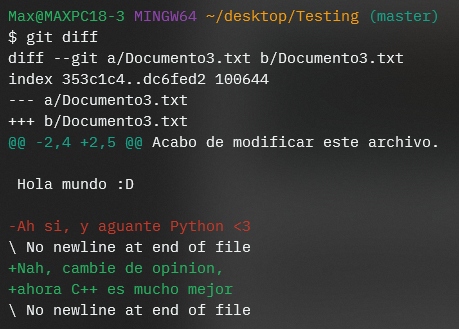
En este caso, vamos a volver a modificar ese archivo, y vamos a agregar y quitar otras líneas más:



Como es de esperarse, si ejecutamos “git status”, el archivo volverá a estar como “no preparado”, ya que lo modificamos después de haberlo preparado. Pero si ahora queremos saber exactamente “qué cambió”, entonces usamos el siguiente comando:

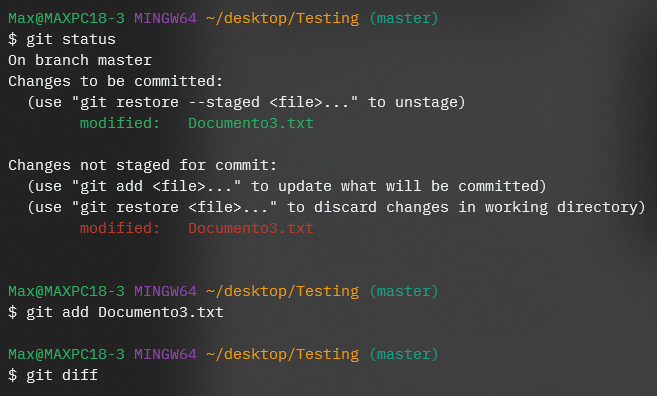
git diff

Viene de “difference”, y te mostrara una comparación de todos los cambios que hay **entre los archivos que están “preparados” y los “no preparados”.**



Ahí vemos que pasó en realidad: eliminé la línea de “aguante Python” y lo cambié por otras dos líneas mas diciendo que “C++ es mucho mejor”.

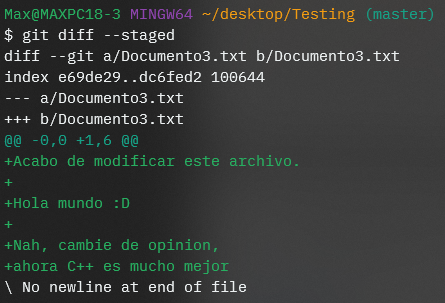
Vale, pero si ahora volvemos a preparar ese archivo, ya no habrá diferencias, ya que no hay archivos no preparados.



Pero… ¿recuerdas [la última vez que confirmamos los cambios](#_Confirmar_archivos)? Fue cuando recién creamos esos archivos, cuando eran archivos en blanco. ¿Qué ha cambiado desde la ultima confirmacion?

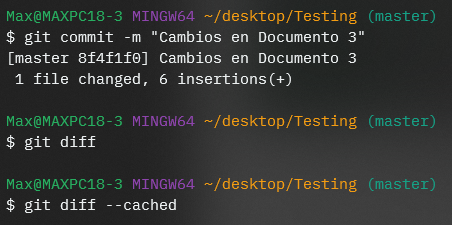
Para responder eso, podemos ver la diferencia que hay **entre los archivos preparados y los de la última confirmación** con:

git diff --staged



Aquí se nota que desde la última confirmación (donde los archivos que teníamos eran solo archivos en blanco) solo se han agregado líneas.

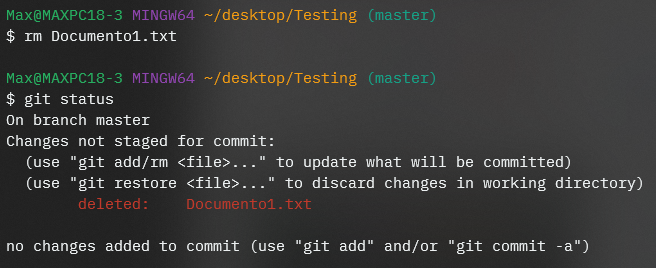
Si confirmamos ahora los cambios…



Ya no hay diferencias.

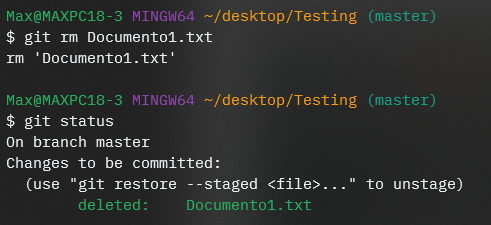
## Eliminar archivos

Git también tiene la habilidad de saber cuando un archivo se elimina, por ejemplo, si simplemente eliminamos “Documento1.txt” con “rm” (véase [comandos](#_Comandos)):



El archivo se borrará de nuestra carpeta, pero los cambios no se confirmarán. Para confirmar los cambios, usamos el siguiente comando:

git rm [carpeta/archivo]

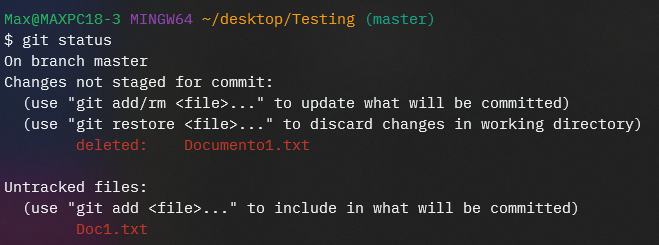


Los cambios estarán ya preparados para la próxima confirmación. Si confirmamos ahora, el archivo se eliminará y no será rastrado.

## Renombrar archivos

Si cambias de nombres algún archivo, Git no se dará cuenta, y pensará que el archivo antiguo se eliminó y que otro archivo nuevo se generó.

Si cambiamos el nombre de “Documento1.txt” a “Doc1.txt”:

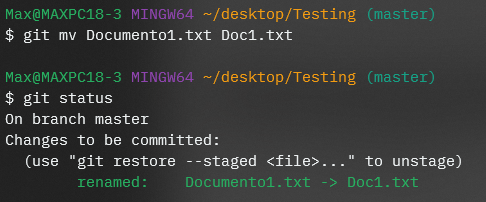


Lo que tendríamos que hacer seria eliminar “Documento1.txt” del repositorio con “git rm” y añadir “Doc1.txt” para que Git vuelva a rastrearlo.

Para evitar todo eso, y que Git lo haga automáticamente, se usa el siguiente comando:

git mv [archivo] [nuevo nombre]

Usando esto, Git sabrá automáticamente que el archivo simplemente se renombró.



## Revisar historial

Para ver el historial de cambios de tu repositorio Git,

# Python

Python es un lenguaje de programación, en donde la mayor característica que lo diferencia de otros lenguajes es su simpleza y su fácil “lectura”.

La [filosofía de Python](https://es.wikipedia.org/wiki/Python#Filosofía) es clara: un lenguaje que sea fácil de aprender, leer, que sea atractivo, y potente.

## Descarga

Puedes descargar Python desde su página oficial: <https://www.python.org/downloads/>.



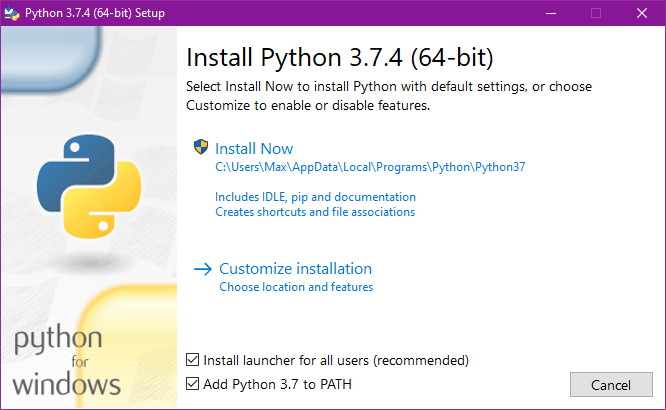
🔸**Importante**: En esta guía usaré la versión 3.7.4 de Python para Windows. Ten en consideración esto ya que algunas veces cambian comandos, o ciertos paquetes dejan de funcionar correctamente.

## Instalación

Aunque parezca obvio como instalar Python, siento que es necesario explicar este tema, ya que, recomiendo unas configuraciones necesarias para el buen funcionamiento de Python desde la consola y esas cosas, o como yo le llamo en mi opinión: una instalación “correcta” de Python.

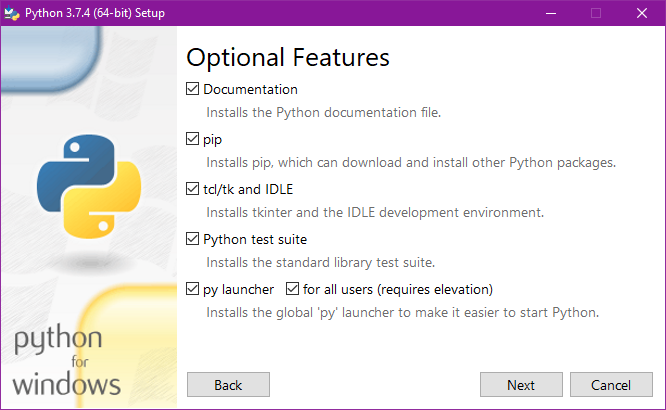
Voy a adjuntar imágenes de cómo debería verse tu instalación. Asegúrate que sea igual en cada paso, incluyendo las casillas de verificación.

Cuando ejecutes el instalador de Python, verás esta pantalla:

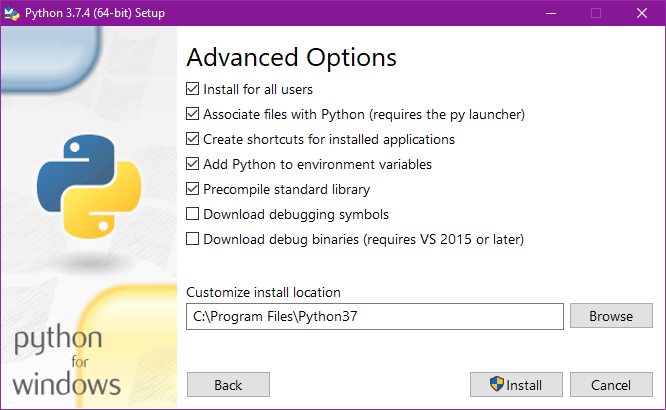


En la zona de abajo, veras una opción que dice “Add Python 3.7 to PATH”, asegúrate de marcar esa casilla.

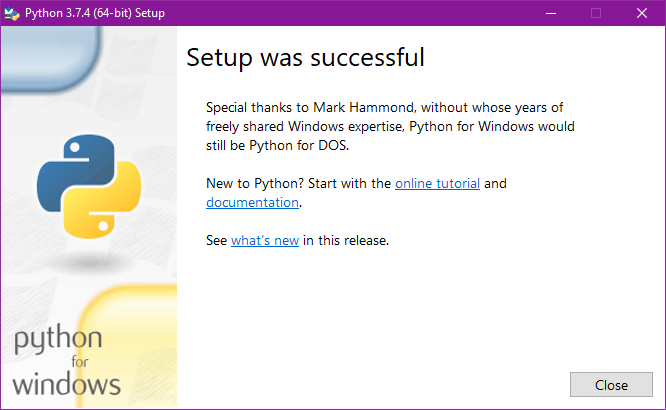
Luego, selecciona “Customize installation”, verás esta pantalla a continuación:



Selecciona “Next”, verás esto ahora:



Asegurate de que la opción “Install for all users”, “Add Python to environment variables” y “Precompile standard library” estén marcadas, luego toca “Install”. Espera unos minutos y Python ya estará instalado en tu computador.



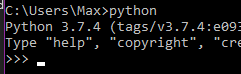
🔸**Importante**: Si ves una opción que dice “Disable path length limit”, selecciónala.

## Iniciar Python

Para asegúrate que funciona, abre un nuevo símbolo del sistema (CMD), y ejecuta el siguiente comando:

python

Si accedes a la consola de Python (puedes darte cuenta por el característico símbolo de “esperando instrucción” >>>) entonces todo está correcto.



Felicidades, ya tienes instalado Python.

## Notas adicionales

El instalador de Python instala dos programas que puedes ver en el administrador de programas en Windows (puedes acceder desde el panel de control).



Python 3.7.4 es el lenguaje como tal.

Python Launcher es un pequeño programa que da un “acceso directo” a Python desde la consola.

Para desinstalar Python por completo, debes desinstalar esos dos programas.

# Módulos y Paquetes

Un módulo (o librería) en Python, es un conjunto de código, funciones, clases y archivos que podemos importar en nuestro proyecto, y poder utilizar sus recursos. Un paquete es un conjunto de módulos.

Un ejemplo de modulo puede ser “random”:

>>> import random  
>>> random.randint(0,100)  
6

Los módulos se deben importar en nuestro proyecto para que podamos usarlos después. Existen módulos adicionales que no vienen incluidos en Python, y que podemos instalar con el administrador de paquetes de Python, o también llamado Pip.

>>> import pygame  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
ModuleNotFoundError: No module named 'pygame'

Como vemos, al intentar importar “pygame”, nos da un error, ya que el módulo no se encuentra, vamos a intentar instalarlo.

## Pip

Pip es un sistema de administración y distribución de paquetes con el que podemos instalar nuevas características para nuestros proyectos.

Para poder utilizar Pip, debemos manejar los comandos desde el terminal del sistema. Para poder ver una lista de los paquetes que tenemos instalados en Python actualmente, escribimos el siguiente comando:

pip list

Para poder instalar algún paquete, simplemente debemos utilizar el comando pip install junto al nombre del paquete que queremos instalar (generalmente las paginas oficiales de dichos paquetes explican como instalarlo).

pip install pygame

**NOTA**: Utilizar el comando de esta forma requiere permisos de administrador ya que se instala para todos los usuarios del PC. Para instalar un paquete únicamente para el usuario activo, utilizar el siguiente comando:

pip install --user pygame

De esta forma, “pygame” ya esta instalado en nuestro sistema, y ahora podemos importarlo:

>>> import pygame  
pygame 1.9.6  
Hello from the pygame community.

Para desinstalar un paquete, usamos el comando pip uninstall, de la misma forma:

pip uninstall pygame

Para actualizar Pip, utiliza el siguiente comando:

python -m pip install --upgrade pip

Si los comandos dan un error algo como “EnvironmentError: [WinError 5] Acceso denegado”, asegúrate de tener permisos de administrador y vuelve a intentarlo.

# Entornos virtuales

Un entorno virtual de Python se puede considerar como una “instalación aislada” a la del sistema. Nos permite poder tener un mayor control de los paquetes de Python que se utilizaran sin necesidad de alterar los paquetes globales, poder mover el entorno a cualquier computador sin temor a que algún paquete falle, etc.

La utilidad de un entorno virtual se ve al tener que transportar el proyecto a muchos usuarios, y para asegurarse que el programa no le falte ninguna dependencia y funcione, todos los paquetes se instalan dentro de un entorno virtual.

## Instalar Virtualenv

Ya sabemos cómo usar Pip, simplemente escribimos:

pip install virtualenv

**NOTA**: Es recomendable instalarlo de forma global, ya que tendremos que usar comandos directos desde la consola. Asegúrate de abrir CMD con permisos de administrador, y ahi recién ejecutar el comando anterior.

Con eso ya tendremos instalado el paquete que nos permitirá crear un entorno virtual. Para ver si funciona, escribe:

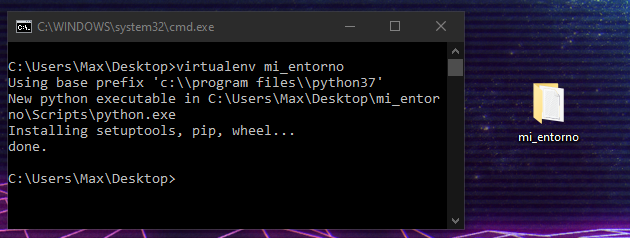
virtualenv

Esto debería abrir un texto de ayuda sobre como utilizar el comando. Si tienes un error, asegúrate de haberlo instalado de forma global, si es necesario, desinstálalo y vuelve a instalarlo.

## Crear un entorno

Para crear un entorno, usa el comando virtualenv junto con el nombre de la carpeta en donde se creará.

virtualenv mi\_entorno

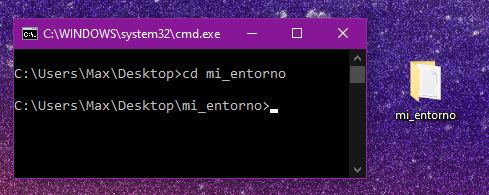


De esta forma, en el lugar donde estaba ubicado el terminal, se habrá creado una carpeta llamada “mi\_entorno”, dentro están todos los archivos necesarios.

## Activar entorno

Si entramos en la carpeta, podemos ver una carpeta llamada “Scripts”, dentro de esta hay un archivo que nos permitirá activar el entorno virtual.

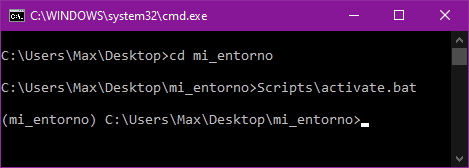
Antes, debemos mover la ubicación del terminal a la carpeta de nuestro entorno (véase “Ubicación de la consola” en la sección [Conceptos](#_Conceptos)):



y ahora debemos escribir lo siguiente:

Scripts\activate.bat

Si todo sale bien, deberás ver que ahora al lado de la ubicación de la consola aparece el nombre de tu entorno entre paréntesis. Significa que el entorno esta activado.



Prueba ejecutar Pip, veras que ya no tienes los mismos paquetes instalados. Recuerda que ahora estas en una instalación “aislada” a la del sistema.

Mientras el nombre del entorno este en paréntesis en el terminal, significa que esta activado y en uso. Para desactivar el entorno, usa el siguiente comando:

deactivate

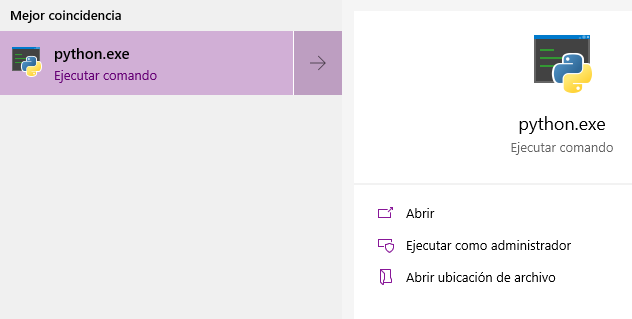
# Django

# Troubleshoot

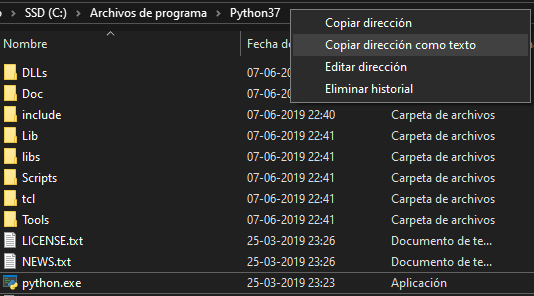
Para evitar cualquier problema durante la ejecución de algunos comandos, asegúrate que los siguientes puntos estén configurados de forma correcta.

## Ubicación de Python

Asegúrate de saber dónde tienes instalado Python. En Windows 10, abre el menú inicio y escribe “python.exe”, en la derecha aparecerá una opción que dice “Abrir ubicación de archivo”, selecciónala y se te va a abrir una carpeta.

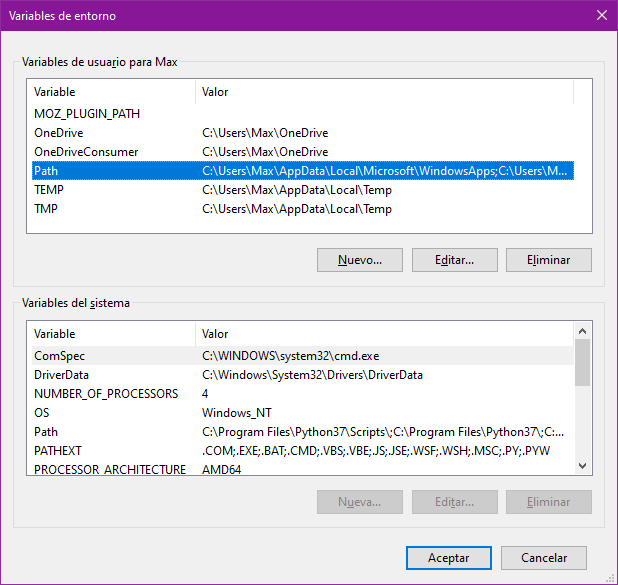


Toca la barra de direcciones para copar la dirección de la carpeta. En mi caso, la dirección es: “C:\Program Files\Python37”.

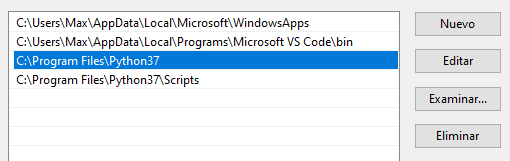


## Variables de entorno

Para ver las variables de entorno en Windows 10, abre el menú inicio y escribe “env”, la primera opción debería decir algo como “Editar las variables de entorno”, selecciónalo presionando [Enter].



En la lista, busca una variable que se llame “path” y toca “Editar”.



En esa nueva ventana, toca “Nuevo” y pega la dirección de tu carpeta de Python, luego agrega otro más y de la misma forma, pero al final agrégale “\Scripts” (sin comillas). Luego en “Aceptar” y “Aceptar”.

Para probar si funciona, abre CMD y ejecuta los siguientes comandos:

python

Con este debería abrirse el intérprete de Python.

pip

Con este debería aparecer un texto de ayuda de Pip.

Si los dos funcionan, Python ahora esta correctamente definido en el sistema.