

Introducción

Git

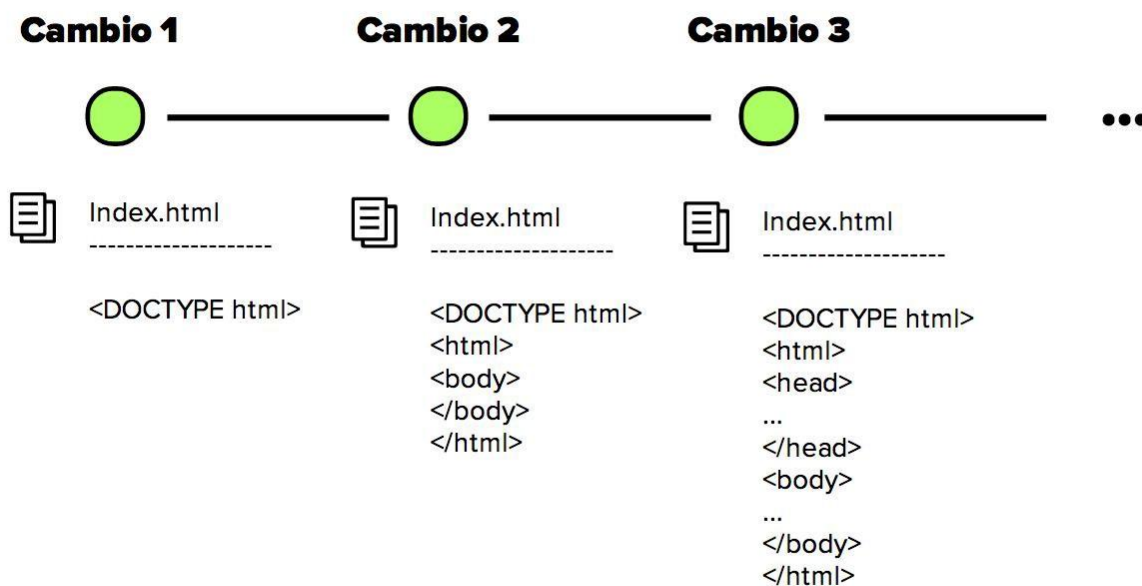
Definimos como control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto es decir a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración. Git, es un software de control de versiones diseñado por [Linus Torvalds](#) Y, entonces ¿a qué le llamamos sistema de control de versiones?

Un sistema de gestión de versiones es la herramienta por excelencia de los desarrolladoras y desarrolladores utilizan para administrar sus proyectos, ordenando así el código de cada una de las nuevas versiones que sacan de sus proyectos o aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

Así pues, Git es uno de estos sistemas de control, que permite comparar el código de un archivo para ver las diferencias entre las versiones, restaurar versiones antiguas si algo sale mal, y fusionar los cambios de distintas versiones. También permite trabajar con distintas ramas de un proyecto, como la de desarrollo para meter nuevas funciones al programa o la de producción para depurar los bugs.

Las principales características de la plataforma es que ofrece las mejores características de este tipo de servicios sin perder la simplicidad, y es una de las más utilizadas del mundo por los desarrolladores. Es multiplataforma, y tiene multitud de interfaces de usuario.

Así pues, Github es un portal para gestionar las aplicaciones que utilizan el sistema Git. Además de permitirte mirar el código y descargarte las diferentes versiones de una aplicación, la plataforma también hace las veces de red social conectando desarrolladores con usuarios para que estos puedan colaborar mejorando la aplicación.



Cambios por los que puede pasar un archivo durante su estadía en nuestro proyecto.

GitHub.

Github permite que las y los desarrolladores alojen sus proyectos en repositorios públicos o privados y participen de forma gratuita. Como te hemos mencionado más arriba, en Github también puedes entrar a los proyectos de los demás y colaborar para mejorarlos. Esto quiere decir que los usuarios pueden opinar, dejar sus comentarios sobre el código, colaborar y contribuir mejorando el código. También pueden reportar errores para que los desarrolladores lo mejoren.

Github también ofrece una serie de herramientas propias con las que complementar las ventajas que ya tiene el sistema Git de por sí solo. Por ejemplo, puedes crear una Wiki para cada proyecto, de forma que puedas ofrecer toda la información sobre él y anotar todos los cambios de las diferentes versiones.

Tiene un sistema de seguimiento de problemas, para que otras personas puedan hacer mejoras, sugerencias y optimizaciones en los proyectos. Ofrece también una herramienta de revisión de código, de forma que no sólo se pueda mirar el código fuente de una herramienta, sino que también se pueden dejar anotaciones para que su creador o tú mismo después si es tu proyecto las podáis revisar. Se pueden crear discusiones también alrededor de estas anotaciones para mejorar y optimizar el código.

A su vez, se pueden encontrar gráficos para ver cómo trabajan los desarrolladores en sus proyectos y bifurcaciones del proyecto, viendo las actualizaciones realizadas a partir de la primera versión o los cambios que se han realizado. Y por último también se incluyen características de redes sociales, como un sistema para seguir a tus creadores favoritos y no perderte sus actualizaciones.

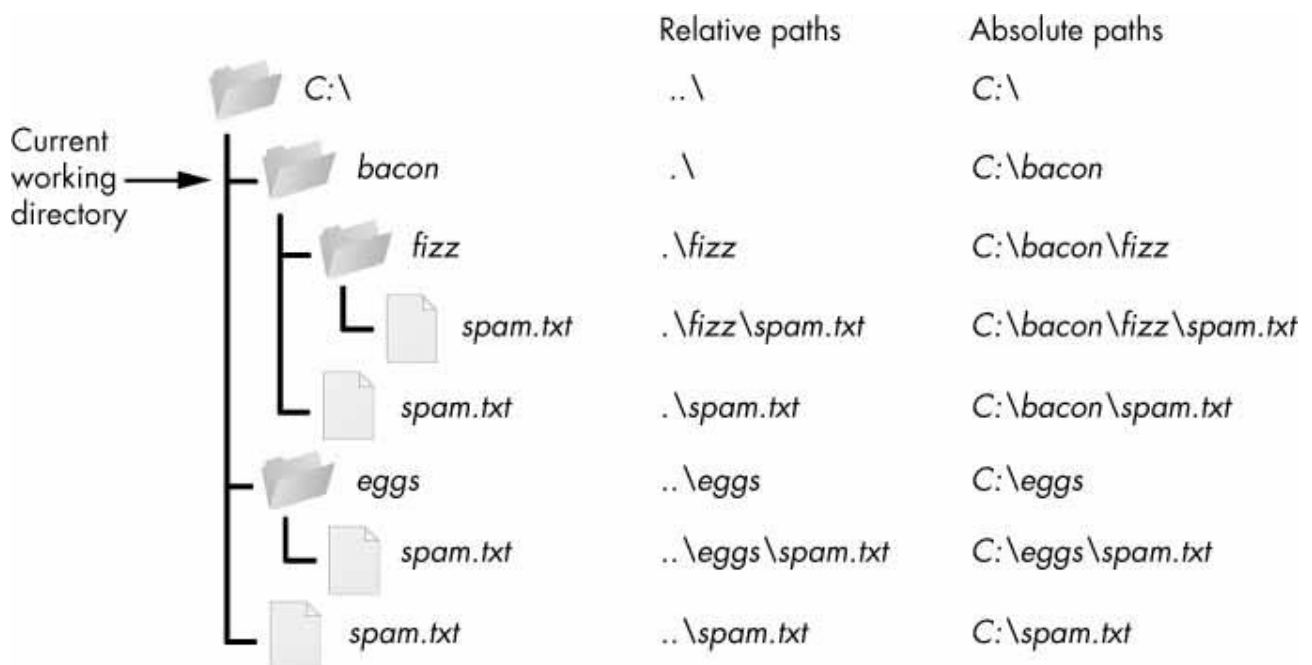
Cuando trabajamos en git nuestros archivos pasan por diferentes estados, a esto le llamamos ciclos de vida de un archivo. esto permite saber cómo es tu estado en un momento determinado.

- **untracked**= Sin rastrear, no está presente en Git. → **git init**
- **unstaged** = Viven dentro de Git pero no se tiene registro actualizado, solo en HDD. → **git add archivo**
- **staged** = Viven dentro de Git. aunque no sus últimos cambios → **git commit -m "mensaje"**
- **tracked**= Guardado con sus upgrades en el local repo a la espera de subirlos. → **git push origin branch**

Primeros Pasos

Es muy importante antes de comenzar a ejecutar comandos que puedas conocer sobre rutas relativas y rutas absolutas.

- Una ruta absoluta, que siempre comienza con la carpeta raíz
- Una ruta relativa, que es relativa al directorio de trabajo actual del programa

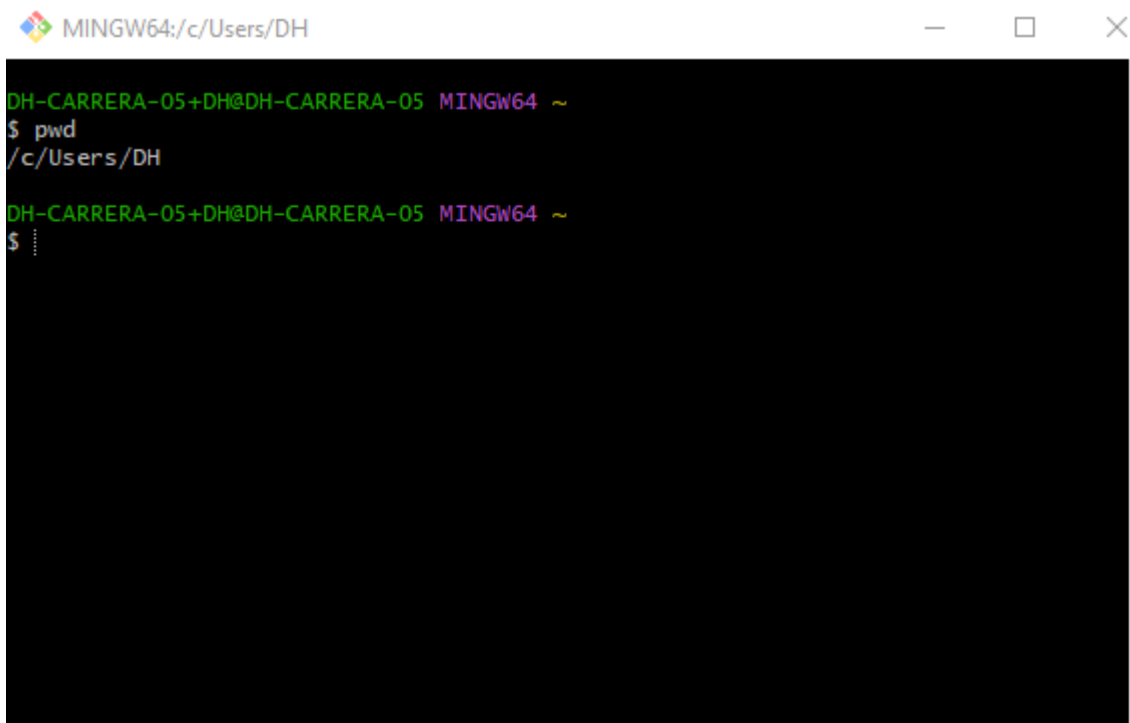


El `.\` al principio de una ruta relativa es opcional. Por ejemplo, `.\spam.txt` y `spam.txt` se refieren al mismo archivo.

Cuando nos referimos a una ubicación con `./` o `../` no son carpetas cómo tales, sino que nos permiten tomar atajos haciendo referencia a rutas relativas desde la ubicación que la terminal se encuentra en este momento y hacia donde necesitamos dirigirnos.

La imagen es un ejemplo de algunas carpetas y archivos. Cuando el directorio de trabajo actual se establece en `C:\bacon`, las rutas relativas para las otras carpetas y archivos se establecen como están en la figura.

Para conocer donde estamos situados podemos introducir el comando "pwd"

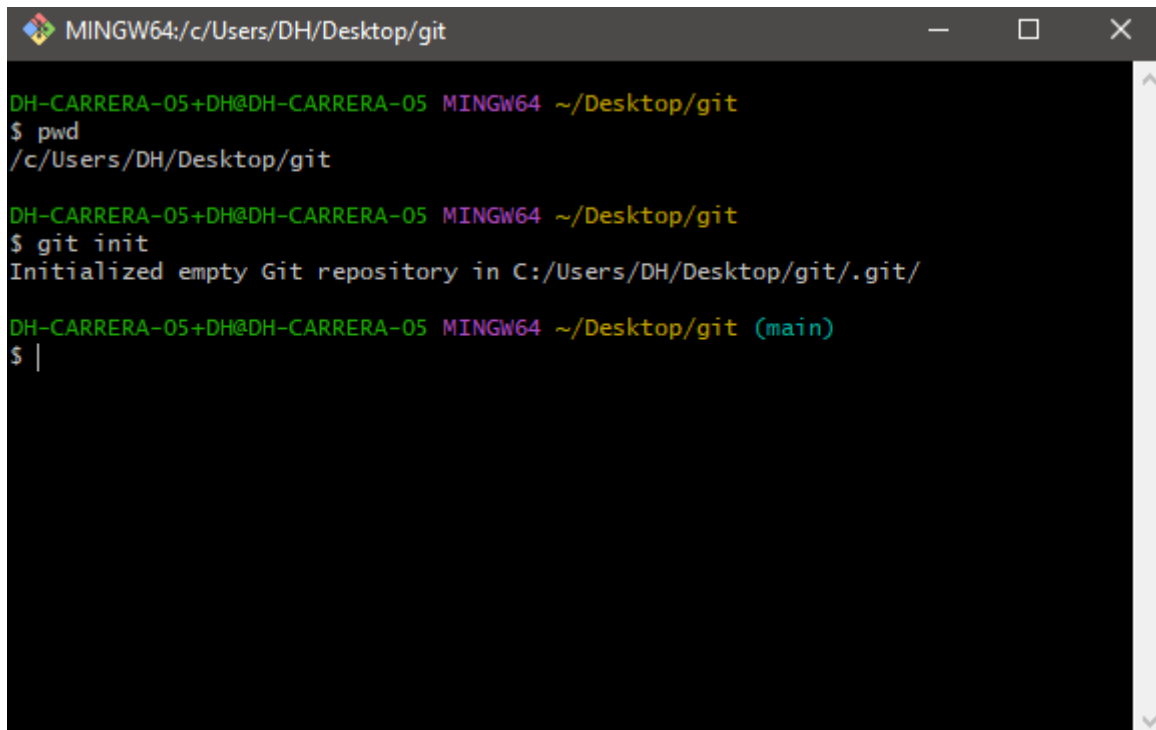
A screenshot of a MINGW64 terminal window. The title bar at the top reads 'MINGW64:/c/Users/DH'. The terminal has a black background with green text. The prompt 'DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~' is shown. The user enters '\$ pwd' and the output is '/c/Users/DH'. The prompt is shown again below the output.

```
MINGW64:/c/Users/DH
DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~
$ pwd
/c/Users/DH
DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~
$
```

Este comando nos devolverá la ruta absoluta de donde estamos situados o ejecutando la terminal bash.

Una vez elegida nuestra ruta de donde necesitamos crear el repositorio local procedemos a tipear el siguiente comando:

git init.

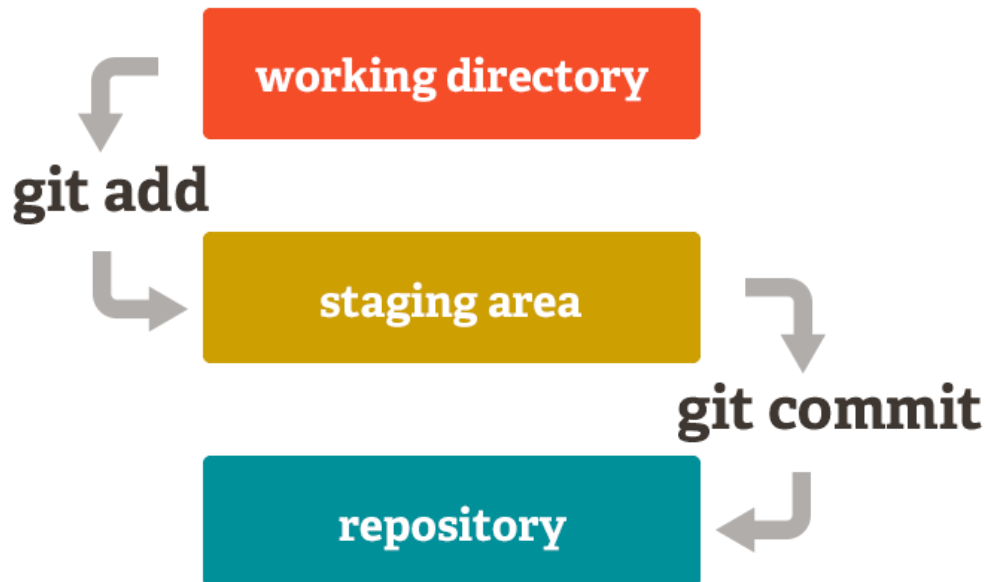


```
MINGW64:/c/Users/DH/Desktop/git
DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~/Desktop/git
$ pwd
/c/Users/DH/Desktop/git
DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~/Desktop/git
$ git init
Initialized empty Git repository in C:/Users/DH/Desktop/git/.git/
DH-CARRERA-05+DH@DH-CARRERA-05 MINGW64 ~/Desktop/git (main)
$ |
```

Inicializado esto procederemos a ver un mensaje diciendo que inicializamos un repositorio “vacío”. En este punto si inicializamos un repositorio con archivos adentro el mensaje cambiará por “inicializaste un repositorio no vacío” Como podemos observar también se ha creado una línea de flujos independientes conocida como rama principal o “main branch”. Si tuviéramos una rama denominada “master” procederemos a tipear el siguiente comando:

config --global init.defaultBranch main

Esto nos da lugar a tres estados posibles dentro de git



El working directory es el directorio de trabajo, es el espacio en disco duro que vamos a disponer para poder agregar, modificar y eliminar dentro de nuestro repositorio.

La Staging Area es el área de preparación, es un espacio en memoria ram destinado a trabajar con los cambios que van sufriendo los archivos y que aún no fueron confirmados.

El repository es la base de datos que nos va a ir recolectando todos los snapshot que vamos obteniendo una vez que confirmamos (hacemos commit) todos los cambios en ese momento en la línea de tiempo o flujo independiente de trabajo (branch) que estemos trabajando.

Comandos Básicos

`git status`: Nos permite ver el estado de nuestros archivos.

`git add` Nos permite añadir un archivo a la Staging o área de preparación.

- `git add nombre_archivo`
- `git add .` (añade todos los archivos)

`git commit`: envía al local repository todos los archivos que fueron agregados.

git commit -m “mensaje” [Udacity Git Commit Message Style Guide](#)

git reset head: quita los archivos de la zona staged y/o los devuelve a su estado anterior.

git rm: Nos permite eliminar el archivo del working directory pero no lo elimina del historial ya almacenado en git.

git rm –cached: Nos permite mover los archivos al estado anterior o untrucked.

git rm –force: Nos permite eliminar los archivos de git y del disco duro.

Ramas o Branches

Las ramas o branches en git son aquellos flujos de trabajo independientes entre sí, que nos permiten trabajar en diferentes situaciones de nuestro proyecto para lo cual no vamos a afectar a los cambios que ya están reflejados. Una rama o branch contiene todos los commits ordenados en el tiempo. Es acá donde cobra importancia ya que si no de otra manera no podríamos hacerle seguimiento para regresarnos a una versión anterior si lo quisiéramos.

La rama o branch por defecto es **Main**. **Esto quiere decir que es la rama principal y no se debe trabajar allí.**

Cuando necesitamos trabajar en una rama y no afectar el historial de commits lo que podemos hacer es clonar a partir de una ya existente. (Se clona a partir del último commit) Para eso debemos tipear:

git branch “nombre”

git checkout “nombre” para cambiarse

git checkout -b “nombre” crea y se cambia en el mismo paso.

git show muestra el último commit enlazado con dos ramas.

Fusión de dos ramas

Cuando necesitamos traernos el flujo de trabajo de una rama hacia otra. nos posicionamos en la rama que deseamos agregar los commits realizados en la rama en donde se trabajaron dichos commits. una vez allí tipeamos

git merge branch.

Por ejemplo si necesitamos traer a Main lo realizado en hotfix hacemos(Posicionados en Main):

git merge hotfix.

Otros Comandos interesantes

git show: muestra los cambios que han existido sobre un archivo.

git diff: muestra la diferencia entre una version y otra.

Por ejemplo:

`git diff commit1 commit2`

`git diff --staged` vemos los cambios por etapas entre dos versiones.

git log: obtenemos el ID de los commits y mostramos el historial de commits de forma local.

git log --graph muestra un grafico sobre los commits en la branch.

git branch -d branch : elimina la rama local.

git remote add origin URL: permite enlazar un repositorio local con uno remoto (se debe haber creado con anterioridad).