¿ Que es git?

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Permite gestionar y realizar un seguimiento de los cambios realizados en los archivos de un proyecto a lo largo del tiempo.

Con Git, puedes realizar un seguimiento de cada modificación, ramificar tu proyecto en diferentes líneas de desarrollo, fusionar cambios entre ramas y colaborar con otros desarrolladores de manera eficiente. Además, proporciona un historial completo de revisiones, lo que te permite regresar a versiones anteriores en caso de necesidad.

Git funciona mediante la creación de un repositorio, donde se almacenan todos los archivos y su historial. Cada vez que realizas cambios en tus archivos, puedes hacer un "commit" para guardar esos cambios en el repositorio. También puedes crear "ramas" para trabajar en nuevas características o solucionar problemas sin afectar la versión principal del proyecto.

Una vez que te sientas satisfecho con tus cambios, puedes fusionarlos con otras ramas o enviarlos a un repositorio remoto, como GitHub o GitLab, para colaborar con otros desarrolladores. Estas plataformas también facilitan la gestión de proyectos y el trabajo en equipo.

En resumen, Git es una herramienta poderosa para el control de versiones que te permite mantener un registro preciso de los cambios en tus proyectos, colaborar con otros desarrolladores y mantener un historial completo de revisiones.

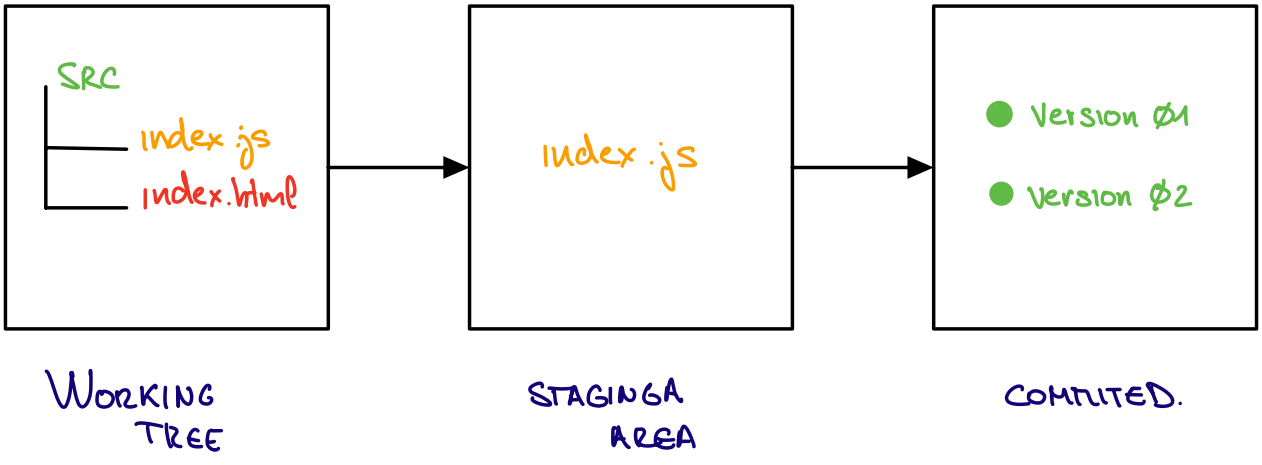
¿Como funciona git?

1. Working Tree (Árbol de trabajo): Es el directorio de tu proyecto donde estás realizando modificaciones en los archivos. Es la versión actual de tu proyecto en la que estás trabajando y donde puedes ver y editar los archivos.

2. Staging Area (Área de preparación): También conocido como "index", es una zona intermedia en Git. Después de hacer cambios en los archivos en tu árbol de trabajo, puedes seleccionar qué cambios deseas incluir en el próximo commit. Agregar cambios al área de preparación te permite controlar qué modificaciones se incluirán en el próximo commit y cuáles no.

3. Committed (Confirmado): Después de agregar los cambios deseados al área de preparación, puedes realizar un commit para confirmar esos cambios en el repositorio de Git. Un commit representa una instantánea de tu proyecto en un punto específico del tiempo y crea una nueva revisión en la historia de tu proyecto. Los commits son permanentes y se guardan en el repositorio, lo que te permite rastrear y volver a cualquier estado anterior en el futuro.

En resumen, el working tree es donde realizas las modificaciones en los archivos, el staging area es una zona intermedia donde seleccionas los cambios que se incluirán en el próximo commit, y el committed representa una confirmación permanente de los cambios en el repositorio de Git. Estos conceptos trabajan juntos para gestionar y controlar los cambios en tu proyecto a lo largo del tiempo.



¿ Como instalo git ?

Por supuesto, aquí tienes un resumen de cómo instalar Git en Windows y Linux:

Windows:
1. Ve al sitio web oficial de Git en https://git-scm.com/downloads.
2. Descarga el instalador de Git para Windows haciendo clic en el enlace correspondiente.
3. Una vez descargado, ejecuta el instalador.
4. Sigue las instrucciones del instalador, aceptando los valores predeterminados a menos que desees realizar configuraciones personalizadas.
5. Durante la instalación, asegúrate de seleccionar la opción "Git Bash Here" para habilitar el uso de la línea de comandos de Git.
6. Después de completar la instalación, Git estará disponible en tu sistema. Puedes abrir Git Bash desde el menú Inicio o mediante clic derecho en una carpeta y seleccionando "Git Bash Here".

Linux (Ubuntu):
1. Abre una terminal.
2. Ejecuta el siguiente comando para actualizar la lista de paquetes: `sudo apt update`.
3. Luego, ejecuta el siguiente comando para instalar Git: `sudo apt install git`.
4. Se te pedirá ingresar la contraseña de administrador. Proporciónala y presiona Enter.
5. El sistema descargará e instalará Git en tu máquina.
6. Una vez completada la instalación, puedes verificar que Git esté instalado correctamente ejecutando el comando `git --version` en la terminal.

Recuerda que estos son solo resúmenes básicos para instalar Git en Windows y Linux. Pueden haber variaciones dependiendo de la distribución de Linux que estés utilizando. También puedes encontrar instrucciones más detalladas en la documentación oficial de Git para cada plataforma.

¿ Que comandos puedo usar en git?

Comando	Uso	Descripción
`git init`	`git init [directorio]`	Inicializa un nuevo repositorio Git en el directorio actual o en el especificado.
`git clone`	`git clone [URL]`	Clona un repositorio Git existente desde una URL remota y crea una copia local en tu máquina.
`git add`	`git add [archivo]`	Añade un archivo específico o todos los archivos modificados al área de preparación.
`git commit`	`git commit -m "mensaje"`	Crea un nuevo commit con los cambios en el área de preparación y agrega un mensaje de descripción.
`git push`	`git push [repositorio] [rama]`	Envía los commits locales al repositorio remoto y actualiza la rama correspondiente.
`git pull`	`git pull [repositorio] [rama]`	Descarga los cambios del repositorio remoto y los fusiona con la rama local actual.
`git branch`	`git branch`	Lista todas las ramas del repositorio y muestra la rama actual destacada con un asterisco (*).
`git checkout`	`git checkout [rama]`	Cambia a la rama especificada y actualiza el directorio de trabajo para esa rama.
`git merge`	`git merge [rama]`	Fusiona los cambios de la rama especificada en la rama actual.
`git status`	`git status`	Muestra el estado actual del repositorio, incluyendo archivos modificados o sin seguimiento.
`git log`	`git log`	Muestra el historial de commits en el repositorio, incluyendo información de autor y fecha.
`git remote`	`git remote add [nombre] [URL]`	Configura un nuevo repositorio remoto con un nombre y una URL específica.
`git config`	`git config --global user.name "[nombre]"`	Configura el nombre del usuario global para los commits.
`git diff`	`git diff`	Muestra las diferencias entre los archivos modificados y los cambios en el área de preparación.
`git branch -d`	`git branch -d [rama]`	Elimina la rama especificada.
`git stash`	`git stash`	Guarda temporalmente los cambios sin confirmar en una pila de “stash”.
`git stash pop`	`git stash pop`	Restaura los cambios guardados en el último stash y los aplica al directorio de trabajo.
`git reset`	`git reset [commit]`	Descarta commits específicos deshaciendo los cambios realizados.
`git revert`	`git revert [commit]`	Crea un nuevo commit para deshacer los cambios introducidos por un commit anterior.
`git fetch`	`git fetch [repositorio]`	Descarga los últimos cambios del repositorio remoto, pero no los fusiona con la rama local.
`git tag`	`git tag [nombre]`	Crea una etiqueta para marcar versiones o hitos importantes en el historial del repositorio.
`git remote -v`	`git remote -v`	Muestra los repositorios remotos configurados y sus URL asociadas.
`git blame`	`git blame [archivo]`	Muestra quién modificó cada línea de un archivo y en qué commit se realizó el cambio.
`git cherry-pick`	`git cherry-pick [commit]`	Aplica los cambios introducidos por un commit específico a la rama actual.
`git log --graph`	`git log --graph`	Muestra el historial de commits en forma gráfica, con ramificaciones y fusiones.
`git clean`	`git clean -f`	Elimina archivos no rastreados en el directorio de trabajo de manera forzada.
`git remote rm`	`git remote rm [nombre]`	Elimina un repositorio remoto especificado de la configuración del repositorio local.

El primer repositorio

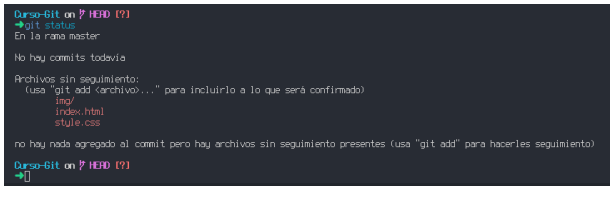
Vamos a crear el primer proyecto en el cual implementaremos git, para controlar el desarrollo.

1º Tenemos que crear un proyecto, o tenerlo previamente, aquí vamos a usar uno vacío completamente y meteremos código si importancia solo para probar

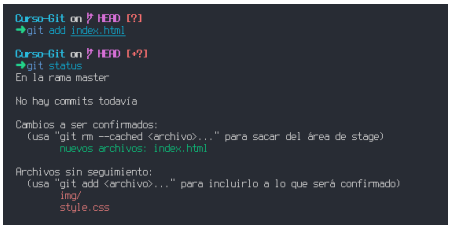
2º Con el código preparado, en la consola dentro de la carpeta donde esta el proyecto hacemos:

david@desktop ~/Curso-Git (master) ⟶ ESTO ES EL PROMT
$ git init ⟶ NUESTROS COMANDOS LOS ESCRIBIREMOS EN ROSA

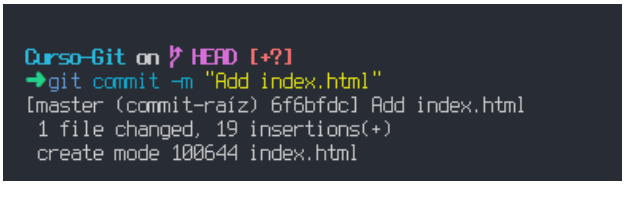
3º Una vez tenemos el repositorio iniciado con git status comprobamos cual es la situación nos de volverá algo asi:



4º Ahora pasamos al staging área con git add + nombre del archivo, nos devolverá algo así:

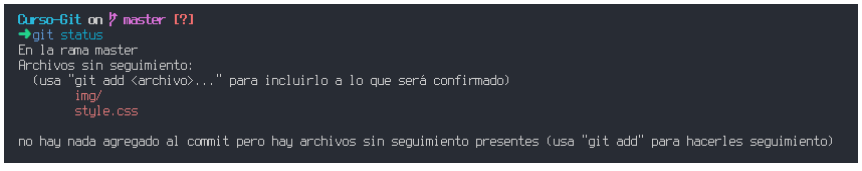


5º El siguiente paso es hacer un git commit: git commit -m “Mensage del commit”
En esta parte del git commit puede que no tengamos configurados nuestro usuario de git, seguimos las instrucciones de la consola y hacemos el commit

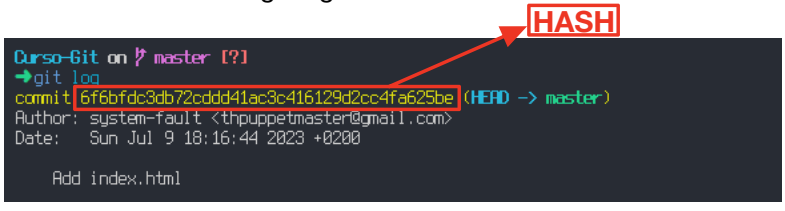


Y veremos como nos indica que se ha agregado el fichero, con datos que iremos usando según avancemos

7º Si volvemos ha lanzar un git status, veremos como el archivo index.html no nos aparece ya que esta “commiteado”, y los únicos que permanecen el el WorkTree son lo que aparecen en rojo.

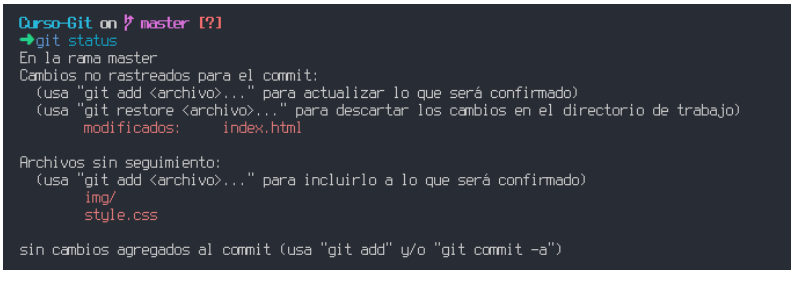


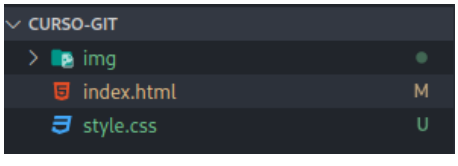
8º Se puede verificar los commits con git log



El **hash**: es un numero el cual identifica el commit, con el podemos acceder al estado del proyecto justo en este instante, es como una foto, a todo, pero una foto interactiva, donde tenemos todo como estaba en ese momento, pudiendo ramificar desde ahí nuevas posiblilidades, o simplemente retornar a ese punto y seguir desde el.

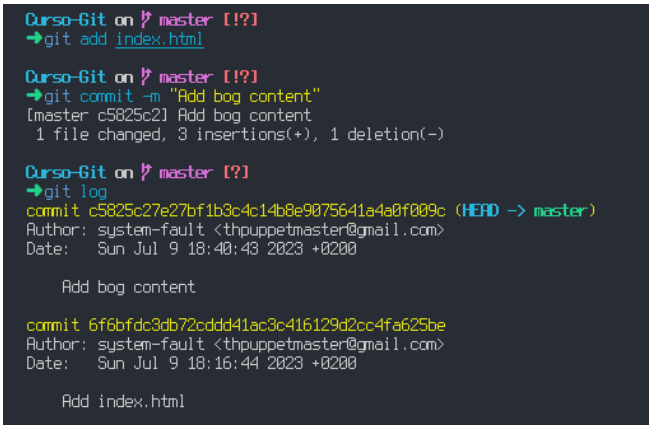
9º Cambiamos el código y comprobamos que cuando volvemos ha hacer un git status, no dice varias cosas, index.html se ha modificado respecto a la versión que tenemos “commiteada”



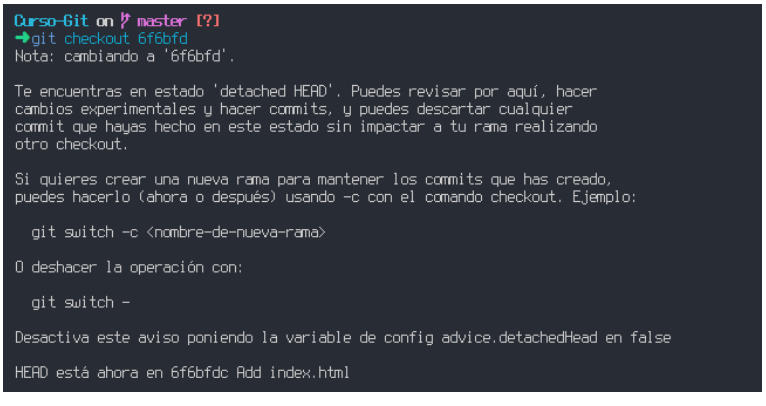


También vemos en el editor VS que tiene git integrado que hemos cambiado el archivo index.html y siguen untracked la carpeta img/ y el archivo Style.css. En la pestaña correspondiente a la parte de git del editor seleccionando distintos archivos modificados podemos ver con un solitario del archivo que hemos cambiado o añadido o quitado respecto a lo que tenemos “commiteado”

10º Ahora volvemos a pasar a staging área los cambios de index.html, siempre que hacemos un cambio hay que pasar por esto antes de hacer el commit, tras esto volvemos ha hacer un git log y veremos que tenemos dos commits, que serian como dos versiones del programa, la que se guardo sin contenido, y ahora la actual que hemos añadido contenido

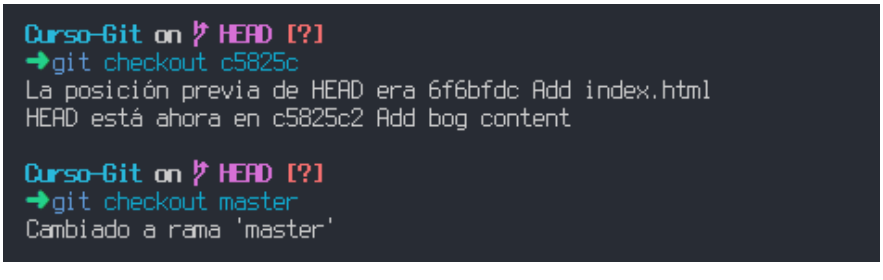


11º Podemos volver a la primera versión con el comando git checkout + iniciales del hash, veremos lo siguiente

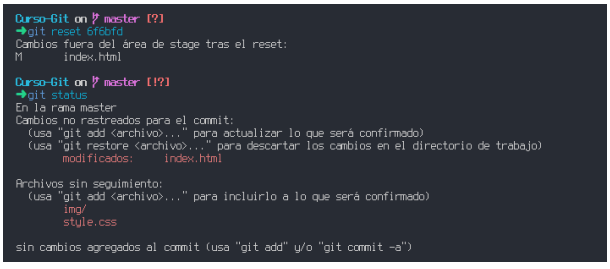


Aquí tenemos un aviso, que nos indica que estamos en “detached HEAD” esto es que en realidad tenemos un puntero apuntando al ultimo commit que realizamos, pero nuestro árbol WorkTree esta en el primer commit que hicimos donde no existe el contenido al blog que si commiteamos. Para volver a nuestra cabecera simplemente hacemos otro git checkout + hash del ultimo commit que hicimos

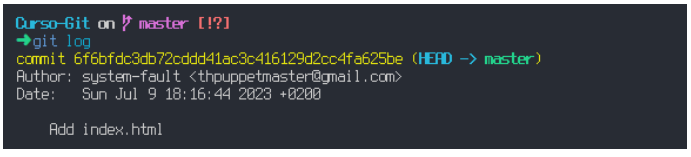
Volvemos a tener el contenido del blog en el archivo index.html
Pero todavía seguimos “detached HEAD”, por lo que tenemos que hacer un checkout máster, a la rama principal (mas adelante se verán las ramas)



12º Para deshacer cambios, podemos hacer git reset + hash, esto nos devuelve a ese commit pero en el archivo tendremos los cambios hechos hasta el reset, si hacemos un git status nos mostrara los ficheros con cambios



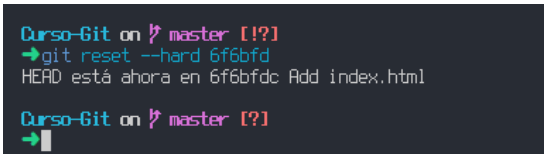
Si hacemos un git log podemos ver que solo tenemos el commit al cual hemos hecho el reset.



Ahora podríamos hace un git add con los cambios y seguiríamos trabajando haríamos commit de los cambios y tendríamos las versiones si “morralla entre medias”, eso si si quieres recuperar esa morralla git permite hacerlo a no se que uses la siguiente forma de usar git reset.

HARD-RESET

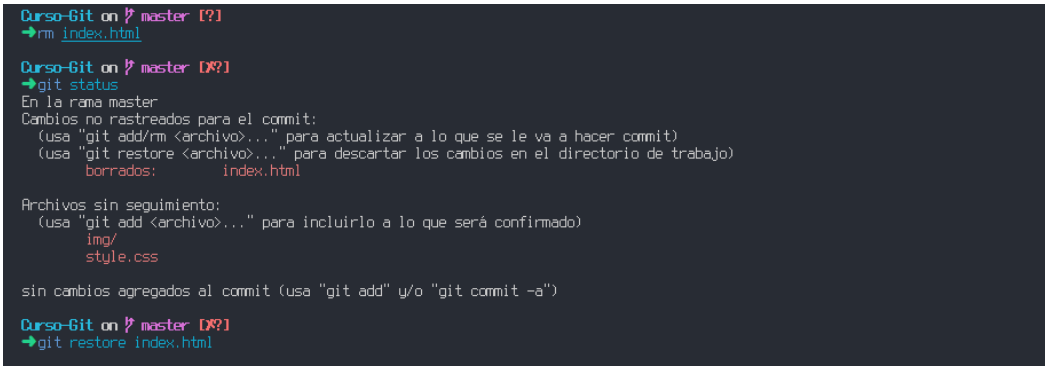
En git si crees que lo que has hecho hasta un punto es malo y no sirve para nada, puedes eliminarlo completamente con el comando git reset - - hard, con este comando borramos todo lo que hay por delante del commit al que nos vallamos con el reset, así que mejor pensar antes de hacer un hard-reset



Con este comando nos llevamos la cabecera y comenzamos de cero en eso punto sin los cambios realizados ni nada, es como si volvieses al punto justo después de hacer el commit.

¿ Como recupero un archivo borrado. Pero trackpads con git ?

Podemos borrar un archivo, pero si esta trackeado o se puede recuperar con git restore + nombre_arch



Cambiamos todo el contenido y hacemos un navegar, y modificaciones en el nombre de la imagen y los estilos del css.

```
Curso-Git on  master [?]
➜git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
        modificados:     index.html

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que será confirmado)
        img/
        style.css

sin cambios agregados al commit (usa "git add" y/o "git commit -a")

Curso-Git on  master [!?]
➜git add .

Curso-Git on  master [!+]
➜git status
En la rama master
Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
        nuevos archivos: img/git.png
        modificados:     index.html
        nuevos archivos: style.css
```

Hacemos el commit

```
Curso-Git on  master [!+]
➜git commit -m "Create navbar"
[master 33574cd] Create navbar
 3 files changed, 26 insertions(+), 2 deletions(-)
 create mode 100644 img/git.png
 create mode 100644 style.css

Curso-Git on  master
➜git log --oneline
33574cd (HEAD -> master) Create navbar
7ed176c Add bog content
6f6bfdc Add index.html
```

GITHUB

Introducción a GitHub y su integración con Git

GitHub es una plataforma en línea basada en la web que proporciona un entorno colaborativo para alojar, revisar y administrar proyectos de software utilizando el sistema de control de versiones Git. Es ampliamente utilizado por desarrolladores de todo el mundo para compartir y colaborar en proyectos de código abierto y privados.

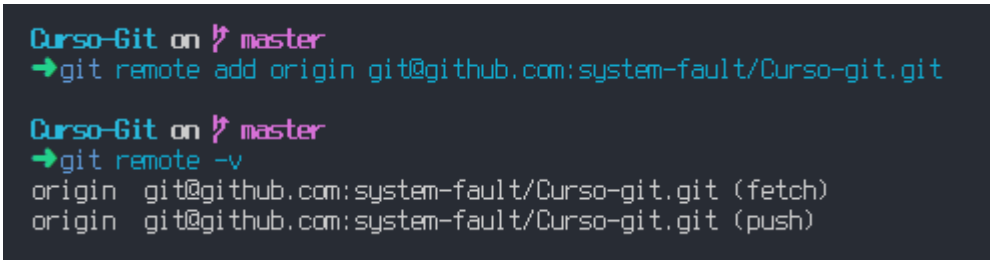
GitHub se integra estrechamente con Git, lo que permite a los desarrolladores utilizar Git para gestionar el control de versiones de sus proyectos locales y luego sincronizar esos cambios con repositorios remotos alojados en GitHub. Esta integración proporciona una forma eficiente y segura de colaborar en proyectos, ya que facilita la colaboración entre desarrolladores y la gestión del código fuente.

Principales conceptos de GitHub y Git:
A continuación, se muestra una tabla que explica algunos de los principales conceptos relacionados con GitHub y Git:

Estos son solo algunos de los conceptos clave relacionados con GitHub y Git. La plataforma y el sistema de control de versiones ofrecen una variedad de características y funcionalidades adicionales que facilitan la colaboración y el seguimiento de los proyectos de software.

Concepto	Descripción
Repositorio	Un repositorio es un lugar donde se almacena el código fuente de un proyecto. Puede ser local (en tu máquina) o remoto (en GitHub).
Commit	Un commit es una instantánea de los cambios realizados en el repositorio en un momento determinado. Representa un conjunto de modificaciones que se agregan al historial.
Pull Request	Un pull request es una solicitud para fusionar los cambios de una rama en un repositorio con otra rama. Permite revisar y aprobar los cambios antes de ser fusionados.
Branch (Rama)	Una rama es una línea independiente de desarrollo que permite trabajar en características o correcciones de errores sin afectar la versión principal del proyecto.
Fork	Un fork es una copia completa de un repositorio en tu cuenta de GitHub. Te permite trabajar en una versión independiente del proyecto y proponer cambios mediante pull requests.
Issue	Un issue es un registro que se utiliza para rastrear tareas, errores o solicitudes de mejora en un proyecto. Facilita la comunicación y el seguimiento de los problemas del proyecto.
Merge	El merge es el proceso de combinar los cambios de una rama en otra. Permite incorporar los commits de una rama en otra y mantener la historia del proyecto en un estado coherente.

¿Como subimos los repositorios locales a GITHUB?
Para subir un repositorio la forma más rápida es crear un repositorio vacio en GITHUB y desde nuestra consola dentro de nuestra rama máster hacer un git remote, al crear el repositorio en la web nos aparecerán las instrucciones para subir un repositorio ya existente u otras opciones,



Como podemos observar tenemos un fetch que es desde donde podemos bajar y puso donde podemos subir.

Ahora vamos a subir nuestros cambios al repositorio de GITHUB, antes de hacer todo esto tenemos que tener definidas las claves ssh en el equipo, hay un manual en la pagina de GITHUB para hacer todo el proceso según el OS
Una vez tenemos conexión con GITHUB hacemos git push -u original main

```
Curso-Git on  main
➜ git push -u origin main
Enumerando objetos: 12, listo.
Contando objetos: 100% (12/12), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (9/9), listo.
Escribiendo objetos: 100% (12/12), 3.13 KiB | 1.57 MiB/s, listo.
Total 12 (delta 2), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (2/2), done.
To github.com:system-fault/Curso-git.git
 * [new branch]      main -> main
rama 'main' configurada para rastrear 'origin/main'.
```

Ahora aunque borremos el repositorio de nuestra máquina estará accesible desde cualquier sitio, simplemente tendríamos que clonar el repositorio
Por ultimo una vez hemos subido por primera vez el repositorio con hacer simplemente git push bastaría para actualizar los datos en la version remote también.

```
Curso-Git on  main [!]
➜ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
        modificados:     index.html


Curso-Git on  main [!]
➜ git commit -m "Change title"
[main 695c059] Change title
 1 file changed, 1 insertion(+), 1 deletion(-)

Curso-Git on  main [↑]
➜ git status
En la rama main
Tu rama está adelantada a 'origin/main' por 1 commit.
  (usa "git push" para publicar tus commits locales)

nada para hacer commit, el árbol de trabajo está limpio

Curso-Git on  main [↑]
➜ git push
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 354 bytes | 354.00 KiB/s, listo.
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:system-fault/Curso-git.git
   33574cd..695c059  main -> main
```

RAMAS

En GitHub, las ramas juegan un papel fundamental en el desarrollo y la colaboración de proyectos. Una rama es una línea independiente de desarrollo que permite trabajar en características o correcciones de errores sin afectar la versión principal del proyecto. Proporcionan un entorno aislado para realizar cambios y experimentar sin comprometer la estabilidad del proyecto.

Cuando se crea un repositorio en GitHub, se crea automáticamente una rama principal llamada "main" o "master". Esta rama representa la versión estable y utilizada comúnmente del proyecto. Sin embargo, en lugar de trabajar directamente en la rama principal, se recomienda crear ramas adicionales para realizar modificaciones o agregar nuevas funcionalidades.

Las ramas permiten a los desarrolladores trabajar de forma paralela en diferentes aspectos del proyecto. Por ejemplo, puedes crear una rama llamada "feature-login" para implementar la funcionalidad de inicio de sesión, mientras que otro desarrollador puede crear una rama llamada "bug-fix" para solucionar errores específicos. Cada rama tiene su propio historial de commits y se puede fusionar posteriormente con la rama principal o con otras ramas.

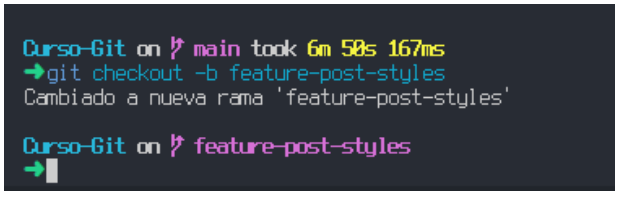
La ventaja de utilizar ramas es que puedes desarrollar características de forma aislada sin afectar la rama principal. También facilita la colaboración, ya que varios desarrolladores pueden trabajar en diferentes ramas al mismo tiempo y fusionar sus cambios cuando estén listos.

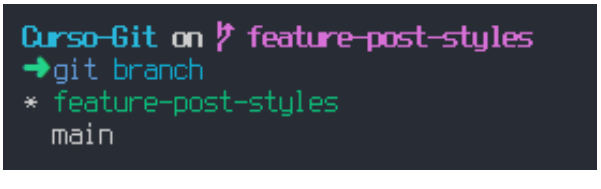
GitHub proporciona una interfaz intuitiva para crear, cambiar y fusionar ramas. Además, facilita la revisión y aprobación de cambios a través de pull requests, lo que permite a otros desarrolladores revisar y discutir los cambios antes de que se fusionen en la rama principal.

En resumen, las ramas en GitHub permiten un desarrollo más flexible, colaborativo y organizado de proyectos. Proporcionan un entorno seguro para trabajar en nuevas características y correcciones de errores sin afectar la versión principal del proyecto, y facilitan la integración de cambios mediante la fusión de ramas a través de pull requests.

Creando una rama

Hacemos un git checkout -b + nombre_rama con la opción -b creamos la rama a la vez que nos cambiamos a ella.





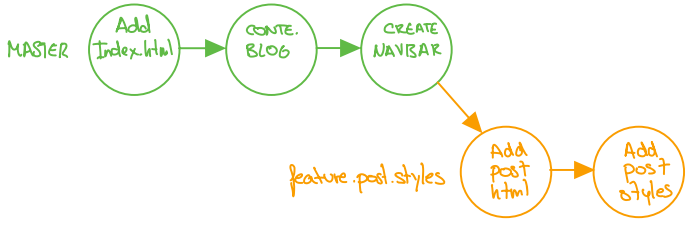
Ahora estamos en la rama nueva, también se puede ver en que ramas estamos en el editor, y el comando git branch

Ahora hemos vuelto a modificar los ficheros por lo tanto podemos hacer un commit en la nueva rama, un atajo para hacer un commit sin pasar por el staging área es usar la opción -a junto con la opción -m

```
Curso-Git on  feature-post-styles
➜git status
En la rama feature-post-styles
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
        modificados:     index.html
        modificados:     style.css

sin cambios agregados al commit (usa "git add" y/o "git commit -a")

Curso-Git on  feature-post-styles [!]
➜git add index.html

Curso-Git on  feature-post-styles [!]
➜git commit -m "Add post html"
[feature-post-styles f807ddf] Add post html
 1 file changed, 8 insertions(+), 2 deletions(-)

Curso-Git on  feature-post-styles [!]
➜git add style.css

Curso-Git on  feature-post-styles [!]
➜git commit -m "Add post styles"
[feature-post-styles 811ef67] Add post styles
 1 file changed, 22 insertions(+)
```

El siguiente paso es hacer unir nuestra rama nueva con la máster/main, podemos ver el historial de commits pero a una línea por commit con git log - - oneline

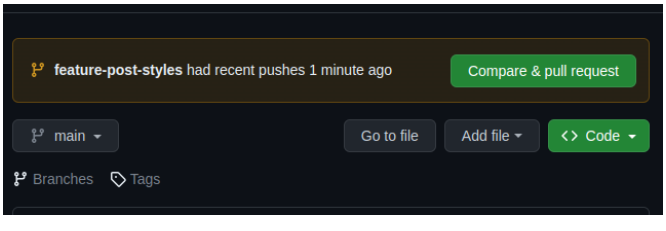
```
Curso-Git on  feature-post-styles
➜git log --oneline
811ef67 (HEAD -> feature-post-styles) Add post styles
f807ddf Add post html
695c059 (origin/main, main) Change title
33574cd Create navbar
7ed176c Add bog content
6f6bfdc Add index.html

Curso-Git on  feature-post-styles
➜
```



Este ultimo git log no informa también que nuestro repositorio remoto esta dos commits por detras respecto a nuestro repositorio, vemos a nuestra cabeza apuntar al ultimo commit, pero el origin/main apunta al ultimo commit que hemos subido
Antes de fusionar con máster, debemos crear la rama en GITHUB, ya que tener una rama local no significa que automáticamente se cree en GITHUB
Para ello usamos git push -u origin + nombre_de_la_rama

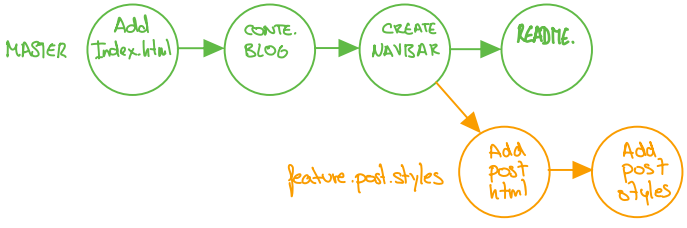
```
Curso-Git on  feature-post-styles
➜git push -u origin feature-post-styles
Enumerando objetos: 9, listo.
Contando objetos: 100% (9/9), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (6/6), listo.
Escribiendo objetos: 100% (6/6), 989 bytes | 989.00 KiB/s, listo.
Total 6 (delta 2), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'feature-post-styles' on GitHub by visiting:
remote:      https://github.com/system-fault/Curso-git/pull/new/feature-post-styles
remote:
To github.com:system-fault/Curso-git.git
 * [new branch]      feature-post-styles -> feature-post-styles
rama 'feature-post-styles' configurada para rastrear 'origin/feature-post-styles'.
```

Ahora la rama ya existe en el repositorio de GITHUB, y nos aparecerá una notificación, en la que nos aparece por primera vez el concepto pull request



Vamos a simular que tenemos otra persona trabajando haciéndonos un clon de nuestro repositorio. Ahora volvemos a la rama main en nuestro equipo y creamos un README.me.

```
Curso-Git on  feature-post-styles
↓2% ➜git checkout main
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.

Curso-Git on  main
↓2% ➜git add README.md

Curso-Git on  main [+]
↓2% ➜git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
        nuevos archivos: README.md


Curso-Git on  main [+]
↓2% ➜git commit -m "Add README.md"
[main 03a7dda] Add README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

Curso-Git on  main [↑]
↓2% ➜git push
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 352 bytes | 352.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To github.com:system-fault/Curso-git.git
   695c059..03a7dda  main -> main
```



Las ramas pueden avanzar en paralelo y luego mezclar sus cambios siempre que no halla conflictos Seguimos los pasos para realizar el merge(fusión de ramas) y por ultimo ya que no tenemos en nuestro main local los cambios realizados en la otra rama hacemos un git pull en nuestro main local para actualizar el contenido del repositorio.

```
Curso-Git on  main
↓1% ➜git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
Desempaquetando objetos: 100% (2/2), 779 bytes | 779.00 KiB/s, listo.
Desde github.com:system-fault/Curso-git
   03a7dda..141a5be  main       -> origin/main
Actualizando 03a7dda..141a5be
Fast-forward
 index.html | 10 +++++++++--
 style.css  | 22 ++++++++++++++++++++++
 2 files changed, 30 insertions(+), 2 deletions(-)
```