GIT es un software de control de versiones, su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos (También puedes trabajar solo no hay problema)

***Configuración inicial (desde VS Code)***

* git version (corrobora la versión instalada de git en nuestra pc)
* Debemos abrir la carpeta con los archivos que vamos a respaldar. El proyecto
* git init 🡪 con esto vamos a colocar el proyecto en “woking directory” de git.

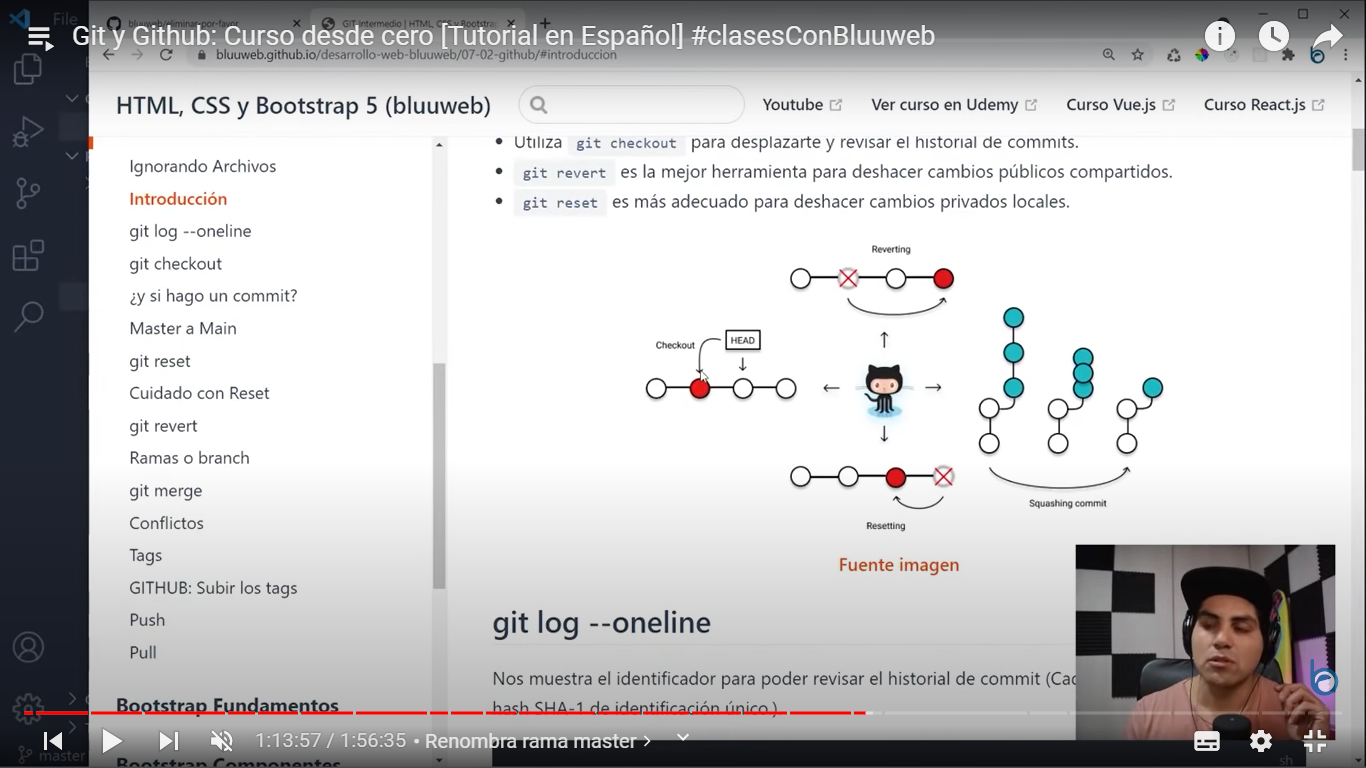


* git status –s 🡪 Nos muestra el estado de los archivos. Si se encuentran en el working directory, en staging área o local repo.
  + git status –sb 🡪 Muestra información de la rama maestra
* git add nombreArchivo.ext – o bien git add . (para añadir todo archivo con cambio) 🡪 Con estos comandos añadimos los archivos al staging área.
* git commit –m “el –m permite añadir un mensaje entre comillas” 🡪 Este comando agrega los archivos de staging área a local repo
* git log --oneline 🡪 Esto muestra todos los commit, es decir, los movimientos hacia local repo con sus mensajes. El (HEAD -> master) muestra nuestra posición actual.
  + git log --oneline --graph 🡪 Muestra de manera más gráfica
  + git log --oneline --decorate --all --graph 🡪 Muestra más elegante
* git reset --hard identificador 🡪 comando para volver a un punto commit, eliminando todos los sucesores de dicho commit. Esto se hace con el identificador asociado.
  + Sin embargo, si tenemos algún identificador eliminado, con el mismo comando podemos volver al punto sucesor.
  + Hay que tener en cuenta que de manera local no daría problemas, pero si lo subimos a github, al ser diferente el log de commits local y del server, daría error.
* git revert IDdelCommitAEliminar 🡪 elimina un commit específico, es diferente al reset. Esto se usa para que no haya conflictos al subirlo a github
* Para subir nuestro repositorio git a un server externo debemos asociarlo. Con Github por ejemplo debemos crear una cuenta en éste, creamos un repositorio y luego copiamos el código que inicia con “git remote add origin url….”
* git branch -m nombreRama 🡪 Elijo la rama en la cual voy a trabajar (Esto se hace solo la primera vez)
* git push –u origin NombreRama 🡪 para subir por primera vez los archivos del directorio. Puede pedirnos logearnos con los datos de Github en este caso (Esto se hace solo la primera vez).
* git push 🡪 Este comando usaremos luego, el mismo se encargará de subir solamente los archivos modificados al servidor externo.
* A veces sale el editor VIM, que bloquea la terminal. Para salir de éste debemos tipear :q! y saldrá del mismo
* git ignore 🡪 creamos un archivo dentro del proyecto llamado “.gitignore”, y dentro de éste pondremos los nombres de los archivos que deseemos que git ignore.
  + Archivo.js 🡪 archivo específico
  + \*.js 🡪 ignora todos los archivos con extensión x (js en este caso)
  + nombreCarpeta 🡪 Ignora todos los archivos de la carpeta
* git reflog 🡪 Muestra todos los cambios, el Historial, incluso si borramos los commit.
* git checkout identificador 🡪 nos mueve hacia un commit. Lo visita solamente, no está ubicado sobre ninguna rama.

***Ramas en Git***

Se usa para programar en equipo. Cada desarrollador puede moverse en ramas diferentes, por lo general fuera del master/main

* git branch 🡪 nos muestra en qué rama estamos
* git branch nombreRama 🡪 Crea una nueva rama
* git checkout nombreRama 🡪 Nos movemos a dicha rama
* git branch –d nombreRama 🡪 Elimina una rama
* git merge nombreRama 🡪 Unir la rama Master/main o la RamaX con otra. Para ellos debemos estar posicionados en Master/main o RamaX.
  + Si en la rama actual hay cambios, y en otra rama también hay cambios sobre el mismo objeto, al intentar unir las ramas se generará un conflicto. En ese caso deberíamos aceptar los cambios de una de las ramas y luego hacer el add y el commit.
* git branch –m nombreRamaActual NuevoNombreRama 🡪 Cambia el nombre de la rama
  + para que el cambio sea permanente, por ej en el caso de la rama Master, podemos usar: git config --global init.defaultBranch nuevoNombre



***Tags***

Se utilizan para poder hacer versiones del proyecto

* git tag nombreVersion –m “mensaje” 🡪 crea un tag con un mensaje asociado
* git tag 🡪 Mostrar la lista de Tags
* git tag –d nombreTag 🡪 Elimina un tag
* git tag –a nombreTag IDcommit –m “mensaje” 🡪 Hacer una versión en un commit anterior
* git show nombreTag 🡪 Muestra información del tag
* git push --tags 🡪 sube al server los cambios de Tag

***Github***

* git remote –v 🡪 Permite observar a que url está enlazado Git
* git pull 🡪 Cuando realizamos cambios directamente en github de forma remota utilizaremos este comando, el cual descargará los cambios realizados.
* git fetch 🡪 compara nuestros archivos locales con los del servidor, si existiera diferencia no pedirá realizar un git pull
* git clone url nombreCarpeta – o – ubicarnos en la carpeta desde cmd o pwershell y tipear “git clone url” 🡪 para descargar un repositorio completo. Clonarlo.