Cambiar nombre de usuario en git:

$ Git config –global user.name "Tiranoexe"

Cambiar mail de usuario en git:

$ Git config –global user.email .ªzulpablin@gmail.com"

Ver que el nombre y mail estan puestos

$ Git congif –list

Subir un archivo

1. Git status
2. Git add "file.html"
3. Git commit -m 'comentario del commit'

Editar un archivo al que ya hice un commit

1. abrir archivo en VScode para poder editarlo: code file.html
2. editarlo
3. guardarlo
4. Git status
5. se deberia obtener un *modified: file.html*
6. En caso de ser un solo archivo, hacemos un add: Git add file.html
7. si son mas de 1 archivo: Git add .
8. Git status muestra que ya fueron añadidos los archivos modificados
9. commit: Git commit -m "mensaje nuevo con los cambios"

Ver historia del archivo

Se usa: Git log y lo que aparezca mas arriba es la version mas reciente.

Usando Git log –stat se ven los cambios especificos que se hicieron en un archivo a partir del commit.

Con la letra 'Q' salgo en caso de que se muestren cambios mas largos de lo que se pueda ver en pantalla.

Ver los cambios del archivo

Se usa Git show file.html
Solo muestra los cambios entre el archivo viejo y el nuevo, no todo el historial

Comparar cambios de una version con otra a eleccion

Git diff (numero del commit) (numero del commit a comparar).
A estos numeros, que se llaman *Hash*, se los saca al hacer git log file.html La primera deberia ser la version original o la mas vieja, para que los cambios tengan sentido. La segunda es la version nueva o una mas reciente que la primera, asi se nota bien si algo fue agregado o eliminado.

Volver a un archivo anterior

Con Git reset (numero del commit) –hard **TODO** vuelve al estado del commit al que lo reseteamos, volviendo a ese commir la nueva **HEAD**
Git reset (numero del commit) –soft Hace que tambien se vuelva a la version anterior deseada, pero lo que esta en *Staging* sigue ahi, por ende, disponible para el proximo *Commit*

title

Hay directorio en donde estan los archivos del proyecto, por ejemplo, el directorio Html, del proyecto Responsive Coffee shop", en donde estan los archivos: index.html, style.css y script.js junto con la carpeta images.

Al entrar por consola en el archivo index.html y escribir el comando Git init pasan 2 cosas:

Se crea un area en memoria RAM llamada *Staging*, un area completamente desconectada que es donde se van agregando los cambios.

Se crea un *Repositorio Local*, la carpeta /.git/, en donde se encuentran todos los cambios al final del proyecto.

Una vez realizados los cambios al archivo, se lo agrega al mismo a la *Staging Area* usando el comando Git add index.html. En este momento el archivo queda a la espera de ser agregado en el *Repositorio Local*, mientras se pueden agregar otros archivos o se puede remover este archivo usando el comando Git rm.

Usando el comando Git commit -m "mensaje con cambios del commit" el archivo se mueve al *Repositorio*, llamado **Master**, en donde se encuentran todos los cambios realizados.

Para enviar los archivos y cambios realizados hacia un *Repositorio Remoto* en un servidor en la nube, como **Github**, se debe usar Git push.

Desde el *Repositorio Remoto*, en este caso **Github**, usando Git clone url podemos traer los datos hacia un Repositorio Local.

Desde el Repositorio Remoto, usando Git fetch puedo traer una copia de un archivo hacia mi Repositorio Local y para poder llevarlo hacia mi *Directorio de trabajo*, debo realizar un Merge. Para ahorrar estos 2 comandos (fetch y merge) puedo simplemente utilizar Git pull para copiar al repositorio local y al directorio de trabajo la base de datos de cambios, asi siempre tengo una compia actualizada de los ultimos cambios realizados en el repositorio.

Estados del archivo

Antes de utilizar el Git add, el archivo esta sin rastrear (*Untracked*) dentro del directorio de trabajo, luego del Add, el archivo entra al estado *Tracked* para luego irse al *Staging*. Si luego de un tiempo no se hace nada con los archivos en Staging, o si se reinicia la PC, los mismos desaparecen por el *Garbage Collector*.

Al usar Git Commit -m los cambios pasan de estar trackeados en *Staging* a estar trackeados en el *Repositorio*, y aqui es donde se le da a cada cambio los indicadores del commit (la serie de numeros y letras que identifica a cada cambio en el archivo visto en Git Log)

Desde la rama *Master*, puedo traer los cambios de la version que desee hacia mi carpeta, usando Git checkout (numero de commit) file.html. En caso de hacer un commit luego de esto, se borraran todos los cambios posteriores a la version de commit actual. Se puede hacer Git checkout master file.html para volver a la version **Master** del archivo, la ultima version de la que se habia hecho un commit.

Ramas

Por defecto, uno siempre se encuentra en la rama **Master** en donde estan todas las versiones de los cambios realizados por los commit. Si por cualquier motivo, en alguna version quiero hacer algo mas experimental, para eso se crea la rama **Development** en la que se copia una version de la rama **Master** y le pongo un nombre, en este caso, **Experimentos**. Toda la rama Development es completamente diferente en codigo y contenido que la rama Master.

En caso de encontrar un bug en la version actual de la **master**, debo crear una rama especial, **Bugfixing** o mas conocida como **Hotfix**. Luego de eliminar los bugs y realizar cambios en la rama Hotfix, para volver a la rama Master, se realiza un **Merge**, creando una nueva version dentro de la Master. Tambien se puede hacer un Merge entre una version de la rama Development y la rama Hotfix para crear una nueva version de la Master.

Se debe tener en cuenta en el momento de hacer un Merge, de que no haya un conflicto en el que algunos archivos se rompan, o que algun cambio realizando en **Bugfixing** rompa un cambio en **Development**.

Para crear una nueva rama se usa Git branch NuevaRama usando Git show vemos que el *HEAD* apunta al *Master* y tambien a la *NuevaRama*.

Para moverme hacia la nueva rama debo usar Git checkout NuevaRama.

Un Git status muestra que ya no estamos en la Master, sino en la nueva rama. Realizamos los cambios que querramos hacer en el archivo y luego usamos Git add y Git commit -m, o para ahorrarnos un paso, podemos usar Git commit -am . Es necesario commitear los cambios **antes** de cada cambio de rama, o se pueden perder los datos.

Si usamos Git checkout master volvemos a la rama principal y el archivo vuelve a mostrarse sin los anteriores cambios realizados en NuevaRama.

Fusion de ramas con Merge

Primero debo poner el **HEAD** en *Master* y uso Git merge NuevaRama. Cabe destacar que un Merge es un *Commit*, por lo que tambien debe llevar un mensaje, en caso de que se abra VIM, ponemos el mensaje y para salir de ahi: Escape + Shift + Z + Z. Luego ya quedan fusionadas las ramas, todo dentro de la Master. Realizamos el commit y guardamos todos los cambios.

Conflicto en el Merge

Puede ocurrir si 2 o mas personas cambian la misma parte del codigo por cosas diferentes y realizan un Merge entre estos archivos con cambios en el mismo lado. La terminal dice que arregles los conflictos y luego vuelvas a hacer commit. El mismo VS Code te pregunta si deseas mantener los cambios del Master o de la otra rama, para luego hacer el commit.