**Git**

Sistema de control de versiones

**Git Tiene 3 Zonas**

**Working Area :** es la zona donde se encuentras los archivos que tienen cambios y que no han sido agregados al steging area. (los archivos de las carpetas en broto en ese momento sin ser agregados por el git add)

**Staging Area :** es la zona donde se indica que archivos están preparados para hacer commit, es decir archivos agregados en ese momento con git add

Git directory : es la zona donde se guardan los datos de los commits, archivos guardados con git commit –m “nombre del commit”



**Configuraciones iniciales**:

git config - - global user.name “jesus dario” 🡪 configura el nombre en git.

git config - - global user.name 🡪 muestra el nombre configurado en git.

git config - - global user.name “Jesus Dario” 🡪 cambiamos el nombre con el mismo comando de creación.

git config - - global user.email “jesusmarencodev@gmail.com” 🡪 configurar el correo en git.

git config - - global color.ui true 🡪 configurar los colores de los resultados.

git config - - global - -list 🡪 muestra todas las configuraciones globales.

**Empezando el proyecto:**

git help 🡪 documentación de los comandos de git

git init 🡪 marca el inicio del proyecto.

git status 🡪 muestra el estado de los archivos del proyecto.

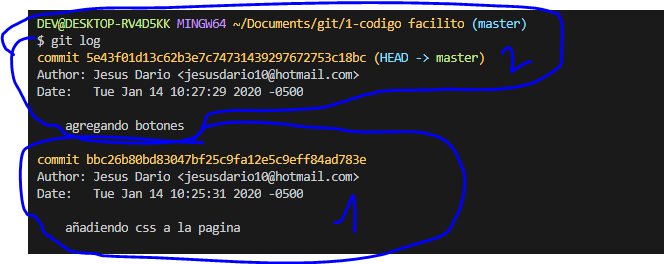
git add archivo completo (eje: git add index.html)🡪 agrega un archivo para ser añadido al commit.

git add . (al final espacio punto)🡪 agrega todos los archivos para ser añadidos al commit.

git commit –m “mensaje para guardar” 🡪 guarda un punto del proyecto a esto se le llama commit.

**git log**

Este comando lo tratare especialmente porque los muestra una lista de todos los commits registrados.

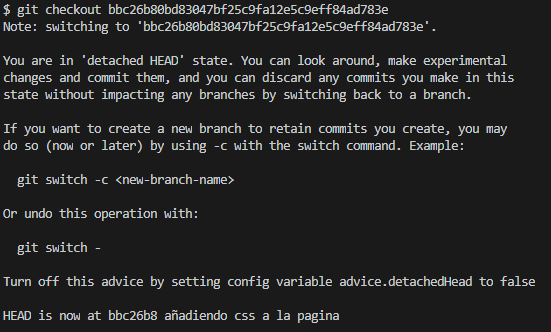


Cada commit genera un código SHA, este código es el que se ve en amarillo al lado de la palabra commit, este código generado es único para cada commit, mas adelante los usaremos.

**git checkout código SHA del commit al que queremos viajar**

**(ejemplo: git checkout bbc26b80bd83047bf25c9fa12e5c9eff84ad783e)**

Permite movernos entre commits sin perder los cambios, es como viajar en el tiempo en un proyecto



**git checkout master**

Nos mueve nuevamente al ultimo commit existente.

**git reset --soft**

borra el commit elegido pero no toca el código o “Working Area”

ejemplo: git reset--soft bbc26b80bd83047bf25c9fa12e5c9eff84ad783e

**git reset --mixed**

borra el “Staging Area”, sin tocar el “Working Area”. No se usa mucho

**git reset --hard**

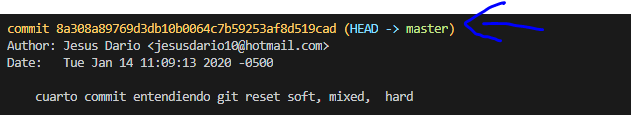
borra el commit commit elegido y el código también

ejemplo: git reset--hard bbc26b80bd83047bf25c9fa12e5c9eff84ad783e

**ramas y fuciones**

**terminología de la sección:**

**Head**: es el commit donde nos encontramos en ese momento.



**Ramas**: una rama es una línea de tiempo en nuestro proyecto, que nos sirve para arreglar errores, experimentar, hacer grandes cambios etc.

**Rama master:** cuando hacemos git init se crea por default y es la rama principal.

* **git branch** 🡪 este comando muestra el listado de ramas existentes actualmente.

 el asterisco muestra en que rama estamos ubicados

* **git branch nombre\_rama** 🡪 crea una rama con el nombre proporcionado.

 cuando se crea una rama se crea una copia de master o de la rama actual y se empieza desde allí.

* **git checkout –b nombre\_rama** 🡪 crea una rama y se me mueve a ella.
* **git chetckout nombre\_rama** 🡪comando para moverse entre ramas

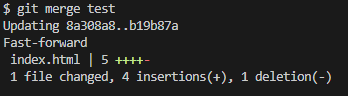
 todos los commits que se hagan desde el cambio se iran a la rama donde estemos, en este caso test.

* **git branch –d nombreRama** 🡪comando para eliminar una rama del repositorio



**Fusiones**: es la creación de un nuevo commit juntando una rama con otra. Juntemos la rama test con la rama master. Para hacer esto hay que seguir unos pasos.

1. Debemos pararnos en la rama que va a absorver a la otra con el comando. **git checkout master**.
2. Usamos el comando git merge rama\_absorvida 🡪 **git merge test**.



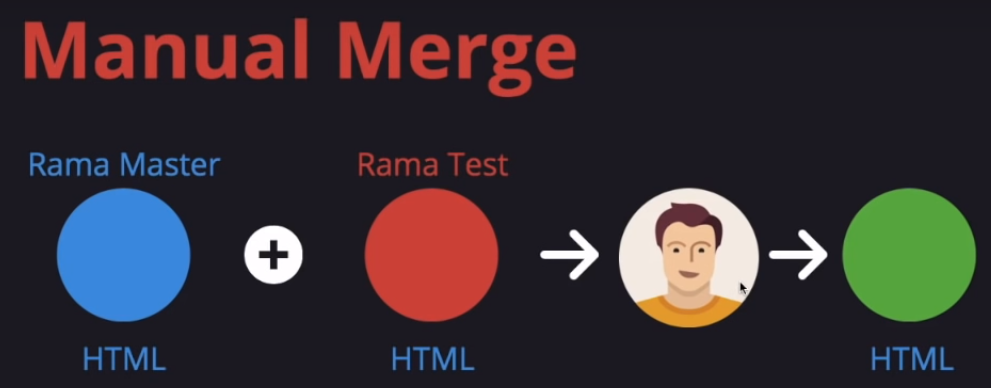
**Existen 2 casos que pueden ocurrir cuando hacemos una fusión**



**Fast-Forwart**: solo hace la fucion, pasa mucho cuando se trabajan con archivos diferentes o líneas de código diferentes.



**Manual Merge**: antes de hacer la fusión tiene que opasar por nosotros antes, normalmente ocurre cuando se trabaja en los mismos archivos o líneas de códigos.



**Github**

Repositorios o proyectos en la nube, Facebook de coders.

Repositorios remotos 🡪github

Repositorios locales🡪 en la pc

**git clone** 🡪 clona un proyecto de github

**git remote add origin url\_del\_repositorio** 🡪 vinculamos nuestro repositorio local con uno remoto.

Ejemplo: git remote add origin <https://github.com/jesusdario10/repasoGit.git>

**git remote –v** 🡪 muestra los orígenes de repositorios remotos configurados en el proyecto.

**git remote remove origin** 🡪 elimina los orígenes remotos de un repositorio local.

**git push origin master** 🡪 pasa lo que tenemos en la pc a github a la rama master.

**git push origin master -f** 🡪 fuerza un push al repositorio, esto ocurre cuando esta actualizado y queremos que igualk haga el push.

**Conceptos**

**Issues:** mejorar el código o solucionar un error en el repositorio o proyecto.

**Milestones:** grupos de issues que aplican para un proyecto, característica o periodo de tiempo.

**labels:** son una manera de organizar diferentes tipos de problemas.

**Tags:** puntos específicos en la historia del proyecto o repositorio, se usan para marcar una versión.

**git tag –a v1.0 –m “mensaje”**🡪 crea un tag de tipo anotado que permite agregar mas información. En mensaje puede ser el commit.

**git tag v1.0** crea un tag de tipo ligero no permite agregar información.

**git push origin --tags** hace push al repositorio remoto de los tags.

**git push origin v1.0** hace push de un tag en especifico.