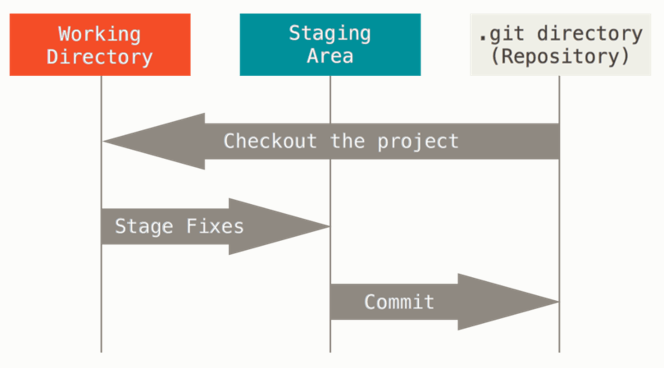
**Manual de Git y GitHub**

**¿Qué es git?**

Git es un software manejador de versiones, pues almacena las versiones de nuestros proyectos dentro de archivos, lo cual nos ayuda a regresar a versiones pasadas de nuestro proyecto si así lo necesitamos.

**Flujo de trabajo de git**



**Configuración básica de git.**

Lo primero que debemos hacer después de haber instalado git es configurar el nombre y el correo de nuestro usuario, para eso haremos uso de los siguientes comandos:

**git config --global user.name "UserName"**

**git config --global user.email "UserEmail"**

**git config --global color.ui true**

El comando anterior será el encargado de cambiar la configuración de colores del interfaz, por lo que no es obligatorio realizarlo si no lo deseamos.

**Comandos de Git.**

El man de git:

**git help comando**

Comando encargado de indicarle a git que debe iniciar un nuevo repositorio dentro de esa carpeta:

**git init**

Muestra el estado de nuestro proyecto (archivos no agregados, archivos modificados, etc):

**git status**

Agrega archivos nuevos o modificados que no se encuentren dentro de nuestro commmit ( staging area ):

**git add [nombres o expresiones regulares]**

Agrega un mensaje al nuevo commmit y realiza el commit ( git repository ):

**git commit -m "Mensaje"**

Muestra todos los commits realizados de ese proyecto, en proyectos grandes cuando ejecute el comando "git log" aparecerán muchos comentarios, para terminar la lista de esos comentarios únicamente presiono la letra "q":

**git log**

Agrega todos los archivos que se encuentren dentro del stage al último commit realizado y permite modificar el mensaje del ultimo commit.:

**git commit –amend -m “nuevoMensaje”**

Permite viajar a travez de nuestros commits y entre ramas realizadas:

**git checkout "shaCommit"**

**git checkout rama**

Comandos para borrar commits:

Respeta el workingArea (workingArea=codigo):

**git reset --soft**

Borra el Staging Area sin tocar el Working Area:

**git reset --mixed**

Borra todo sobre los commits (código, estados y commit):

**git reset –hard**

**Ramas de Git.**

Una rama es una línea de tiempo alternativa dentro de nuestro proyecto, generalmente se utiliza para el desarrollo de una nueva funcionalidad de nuestro proyecto principal, sin que esta afecte el proyecto final, hasta que este 100% terminada.

Rama Master 🡪 Rama principal de proyecto.

Lista las ramas de nuestro proyecto:

**git branch**

Crea una nueva rama dentro de la rama actual:

**git branch nombreRama**

Crear una nueva rama y moverte a la nueva rama:

**git checkout -b nombreRama**

Nos mueve de la rama actual a la rama que recibe como argumento:

**git checkout nombreRama**

Es importante decir que, al crear una nueva rama dentro de nuestro proyecto, esta rama hará una “copia” de todo nuestro proyecto hasta el momento en que realizamos el branch, pero a partir de ahí, todos los cambios y modificaciones que realicemos al proyecto serán vistos y almacenados únicamente en la rama que estemos situados, por eso antes de realizar cualquier cambio al código es importante saber en qué rama estamos situados.

Eliminar una rama:

Local:

**git branch -D nombreRama**

Remota:

**git push origin --delete <NombreRama>**

Una vez que eliminemos una rama todos los cambios realizados en ella desaparecerán a menos que antes de hacerlo la hayamos unido a otra rama, en dicho caso la rama si se eliminara, pero los cambios realizados quedaran guardados en la rama a la cual se unió.

**Fusiones de ramas.**

* Primero nos movemos a la rama a la cual queremos unirle otra.
* Fucionamos la rama con el comando:

**git merge nombreRamaAUnir**

Una vez realizado lo anterior, podremos ver que todos los cambios realizados dentro de la rama que unimos aparecerán en la rama a la cual la unimos.

Es importante resaltar que el comando anterior une las ramas de manera automática y une todos los archivos a la rama, sin embargo, existe una forma de unir ramas de forma manual especificando que archivos queremos unir y que archivos no queremos unir.

**GitHub**

¿Qué es GitHub y cuál es la diferencia entre git y github?

GitHub es un servicio en la nube, controlador de versiones.

Para fines prácticos diremos que la principal diferencia entre git y github es que el primero es un servicio local, que se encuentra solo dentro de nuestro ordenador, por ende, todos los repositorios que creemos solo existirán dentro de nuestra computadora.

Por otro lado, GitHub es un servicio en la nube, por lo que nuestros repositorios se almacenan dentro de un servidor remoto conectado a la red y podremos acceder a estos desde cualquier dispositivo conectado a la red.

Descarga el repo indicado en el directorio actual.

**git clone url**

Conectar un repo remoto con un repo local:

**git remote add origin url**

Origin es el nombre por defecto que le da git al servidor del cual clonaste o vinculaste el repo remoto.

Checar si el repo local actual está conectado con un repo remoto:

**git remote -v**

En caso de estar conectado con un repo remoto nos aparecerían los archivos del origin (fetch y oush).

Desvincular un repo local de un repo remoto:

**git remote remove origin**

Mover archivos de un repo local a un repo remoto:

**git push origin nombreRama**

Podemos utilizar la bandera -f para forzar a que se realice el commit.

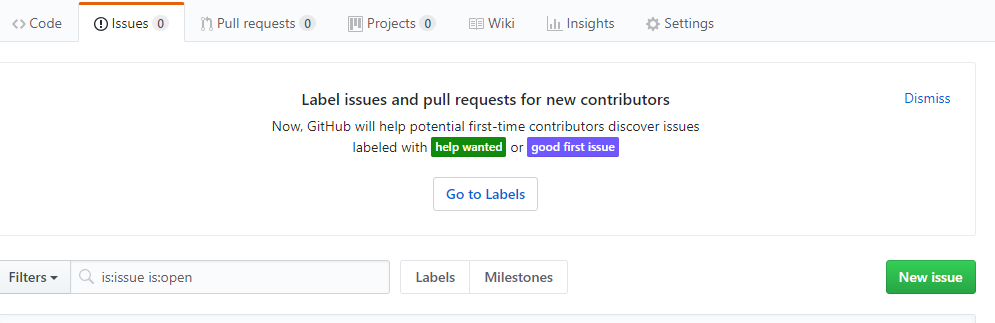
Subir una rama al repo remoto de git:

**git push origin nombreRama**

Como te darás cuenta el comando es el mismo, pues lo que hace realmente es subir una rama del repo sin importar cuál sea, evidentemente estos cambios y ramas también aparecerán separados en nuestro repo remoto.

**Issues**

Un issue es una tarea pendiente por realizar del proyecto, estas notas se crean y checan dentro de la página de github en la pestaña issues

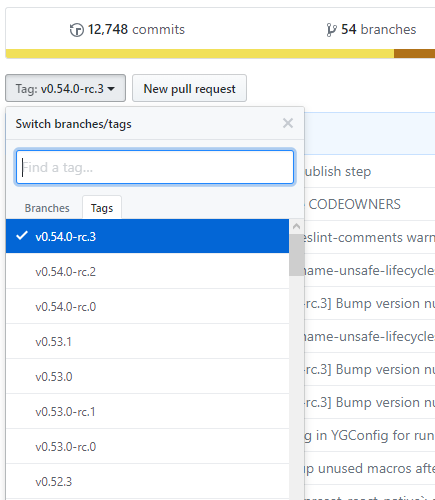


Puedes asignarle un título a la tarea por terminar, así como una descripción lo más explícita posible de lo que se tiene que hacer, para después consultar tu pestaña de issues y terminar las tareas pendientes.

**Tags**

Como su nombre lo dice los tags dentro de git son etiquetas que nos ayudan a describir un poco mejor cada commit de nuestro proyecto, son comúnmente usados para indicar la versión de nuestro proyecto.

Los tags dentro de GitHub se pueden consultar en:



Agregar tag a nuestro commit:

**git tag -a v[NumeroVersion (0.8, 1.0)] -m “Mensaje” SHAcommit**

Subir un tag a nuestro repo remoto (Los tags de nuestros proyectos no se suben con el push normal que hacemos al subir un nuevo commit):

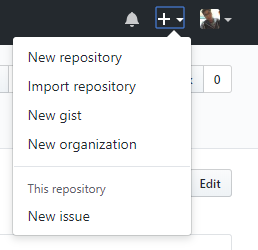
**git push origin v[versiónProyecto]**

Subir todos los tags a nuestro repo local al repo remoto:

**git push origin –tags**

**Workflows (Flujos de trabajo)**

Hasta todo lo que hemos realizado lo hemos realizado únicamente con nosotros mismos, sin embargo, uno de los objetivos de Git es el trabajo en equipo, para lo cual ofrece la opción de crear una organización con más personas para el trabajo en equipo.



Después de haber creado la organización, debemos invitar a las personas con las cuales queremos formar la organización, solo es cuestión de seguir los pasos que te aparecen dentro de la página, los usuarios los puedes buscar por nombre o userName.

Ya terminada nuestra organización el nuevo repositorio se creará con el nombre de la organización, por cual debemos actualizar nuestro repo local con el repo remoto de la organización.

Cuando trabajamos en equipo se crea la rama origin, dicha rama será la rama auxiliar en la cual se almacenan todos los cambios bajados con fetch hasta hacer el merge.

Pasos para actualizar nuestro repo local después de que alguien realizo cambios y los subio al repo remoto.

Hacemos un fetch (Bajamos los cambios realizados del repo remoto) a la rama “oculta” origin:

**git fetch origin**

Recordemos que la rama orgin/master es la rama “oculta” dentro de nuestro repositorio cuya función es precisamente la de unir los cambios de repositorios remotos con el repositorio original.

Actualizar nuestro repo local con los cambios realizados dentro del repo remoto:

**git merge origin/master**

**Resolución de conflictos en commits.**

El caso anterior es el más ideal, en el cual solo uno de los colaboradores realiza cambios al código, sin embargo, esto no pasa casi nunca, generalmente dos o más personas están realizando cambios al mismo tiempo y habrá uno que suba sus cambios primero que otro, por lo cual el colaborador secundario se encontrar con conflictos de código si es que ambos editaron el mismo código, para esto git te mandara una advertencia de que necesitas resolver los conflictos.

Afortunadamente para nosotros la consola de git nos indicara en que archivos del proyecto se encontraron estos conflictos y los marcara con una etiqueta para poder identificarlos, el código que se encuentre dentro de la etiqueta <<<<<<<<<<HEAD será aquel código que realizamos nostros, mientras que el código que se encuentre en la etiqueta ========= será el código descargado durante el fetch.

Para la resolución de estos conflictos tendríamos que ponernos de acuerdo con nuestros colaboradores y decidir qué cambio será el que dejaremos.

Ahora que ya resolvimos los conflictos ya podemos realizar nuestro **git push** de manera normal.

**Git fork**



Cuando realizamos un fork a algún repositorio remoto, lo agregamos a los repositorios remotos de nuestra cuenta en github, para después hacer el clone del proyecto y terlo dentro de nuestra computadora como repositorio local.

Para realizar la actualización de nuestro proyecto, en esta ocacion debemos no solo de actualizar el repo remoto de nuestra cuenta, sino que también tenemos que actualizar el repo remoto de la cuenta en la cual se encuentra el repo original:

**git fetch nombreRamaOculta**

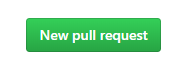
Después actualizamos el repo remoto nuestro:

**git fetch origin**

Y que tenemos todo actualizado podemos hacer el push a nuestro repo remoto, para después solicitar un merge con el repo remoto original.

Para solicitar un merge con el repo remoto original debemos ir a la sección en la cual dice **pull request**, lo cual enviara un mensaje al dueño del repo acerca de lo que hemos hecho y que solicitamos tome en cuenta nuestro commit para el proyecto original.





Como dueño del repo original, tú puedes decidir si aceptar el commit o rechazarlo para después (mediante los comandos normales) realizar el merge con el repo original.

**GitHub Pages.**

¿Qué es GitHub pages?

GitHub pages es un servicio más que te ofrece GitHub para subir tus páginas web (Solo frontEnd) a sus servidores y visualizarlas como si de un dominio particular se tratase.

Para poder hacer uso de este servicio únicamente debemos crear la rama gh-pages.

**git branch gh-pages**

Para poder visualizar la página desde el servidor de github debemos acceder a la direccion:

**userName.github.io/repoNama**

Para cualquier modificación dentro de nuestra página debemos tratar a esta rama como cualquier otra y los cambios se verán reflejados.

Recordar que al hacer el push debemos hacerlo de la siguiente manera:

**git push origin gh-pages**

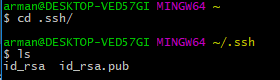
**Deployment con Git.**

**Conectarte a GitHub con SSH.**

Para poder conectarte a github con SSH, lo primero que debemos hacer es crear una llave primaria y una llave publica que será la que pasaremos a todos los perfiles de github que deseemos indicarles que se trata de nuestra computadora.

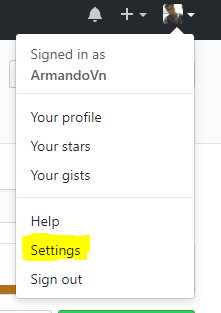
**ssh-keygen**

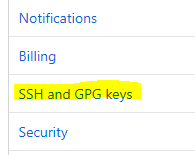
Seguimos las instrucciones del generador, una vez terminada la generación de nuestras llaves públicas y privadas lo que nos falta hacer es ir a la carpeta en la cual se generaron y nos aparecerán los siguientes dos archivos:

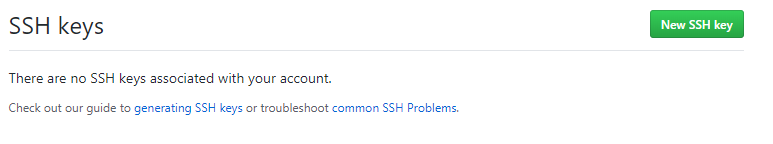


El archivo **id\_rsa** es el archivo que contiene nuestra llave privada y la que nunca debemos mostrarle a nadie, mientras que el archivo **id\_rsa.pub** es el archivo que contiene nuestra llave pública y la que podemos mostrarle a quienes queramos.

Para asociar nuestra computadora con una cuenta de github remota por medio de ssh, lo que debemos de hacer es pasarle la llave publica en la siguiente sección:







Agregamos la llave publica que nos generó el comando ssh y damos en agregar.



Ya que tenemos el ssh remoto listo, debemos cargar de manera local con el comando:

**git remote add origin urlSSH**

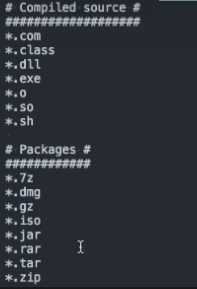
A partir de ahora ya no tendremos que pasarle la contraseña cada que deseemos hacer un commit, para hacer un commit lo realizamos de manera normal y si es la primera vez que hacemos un push usando el protocolo de ssh nos preguntara si deseamos seguir conectados a lo que contestaremos que sí.

**Git Ignore**

Dentro de este archivo se definirán las extensiones o archivos que no deseamos se carguen a nuestros repos cuando realicemos un commit.

Para esto lo único que debemos hacer es crear un archivo llamado .gitignore al mismo nivel que iniciamos el repositorio (.git) y dentro del mismo definir todos los archivos y/o extensiones que **NO** deseemos agregar a nuestros repos.

Ejemplo:



**Git Hook (Pendiente)**

Los git hook son scripts que se ejecutan en git después de realizar alguna acción con git.