Comandos Git

Ezequiel Remus — Email: ezequielremus@gmail.com

ÍNDICE

1. Iniciales 1

2. Repositorios 1

3. Git Colavorativo 5
- 3.1. Introducción 5
- 3.2. Pasos 5
- 3.3. Comandos 5
- 3.4. Ahora a trabajar de verdad 5

Referencias 6

1. INICIALES

Tener en cuenta que el doble guión debe ser consecutivo.

Todos los comandos git comienzan con la palabra **git**.

1. *git - -version*:
   Para conocer la version de git que tenemos.

2. *git config - -global user.name "nombreDeUsuario"*:
   Lo primero que tenes que hacer después de instalar **git** es establecer tu nombre de usuario (y también después de correo electronico). Este comando sirve para establecer tu nombre de usuario.

3. *git config - -global user.email "TuCorreo"*:
   Lo primero que tenes que hacer después de instalar **git** es establecer tu nombre de usuario y correo electronico. Este comando sirve para establecer tu correo.

   Para mas información sobre el comando *config* utilizar el comando: git - -help config

2. REPOSITORIOS

Hay varias formas para trabajar con repositorios, en particular la que explico primero sin haber creado el repositorio desde git y otra habiendo primero creado un repositorio (hablando de repos propios, ya voy a explicar como laburar con repos en grupo). Yo en particular les recomiendo la segunda que es mas rapida (para mi).

Empecemos con la primera:

Creamos una carpeta, abrimos git bash y corremos el comando *git init*. Este comando lo que hace es decirle a git que inicie el seguimiento de la carpeta sobre la cual corrimos el comando. El comando crea una carpeta oculta llamada *.git* donde git almacena su base de datos. Para ver esta carpeta, corremos el comando *ls -a* este muestra todos los directorios, incluso los ocultos.

Luego, dentro de nuestra carpeta repositorio creamos el archivo **README.md**. Este archivo es un archivo de texto markdown el cual se utiliza para realizar la descripción del repositorio e inclusive dar una idea de lo que esta documentado. Dentro de este pondremos el titulo del repositorio de la siguiente forma

$$\#NombredelRepositorio$$

Este va a ser nuestro titulo del *README.md*.

Luego de guardar el *README.md* corremos el comando *git status* que nos sirve para conocer en que estado esta el proyecto. Los estados pueden ser 2, todo rojo o todo verde. Si esta en rojo, significa que el archivo esta modificado y no esta subido (todabia esta en el working directory) a la base de datos git, en cambio si esta en verde significa que lo subis al *staging area*, que es el area de preparacion. Para mover el archivo de la zona de trabajo a el area de preparacion se utiliza el comando *git add .*, al pasarle el punto le decis que suba todos los modificados, si queres solo subir uno de esos archivos le pasas el nombre del archivo con su extencion.

Luego, para que el archivo forme parte de nuestro historial de proyecto es necesario hacer un **commit**, esto es subir el archivo al directorio de git. el comando completo es: *git commit -m "Comentario que represente al commit"*. (*No necesariamente se abre vim para agregar el mensaje, que es lo que dice en el tutorial, lo mas rapido es hacer el comando como se los pase yo sin tener que entrar a ningun editor de texto para escribir el mensaje*).

Si corremos *git status* despues de este paso, vamos a ver que nos dice que ya esta todo listo para subir. Para poder subirlo tenemos que asociar la carpeta a un **repositorido github** para luego realizar un **push**.

Para hacer un repositorio, entramos a nuestra cuenta github y nos logeamos.

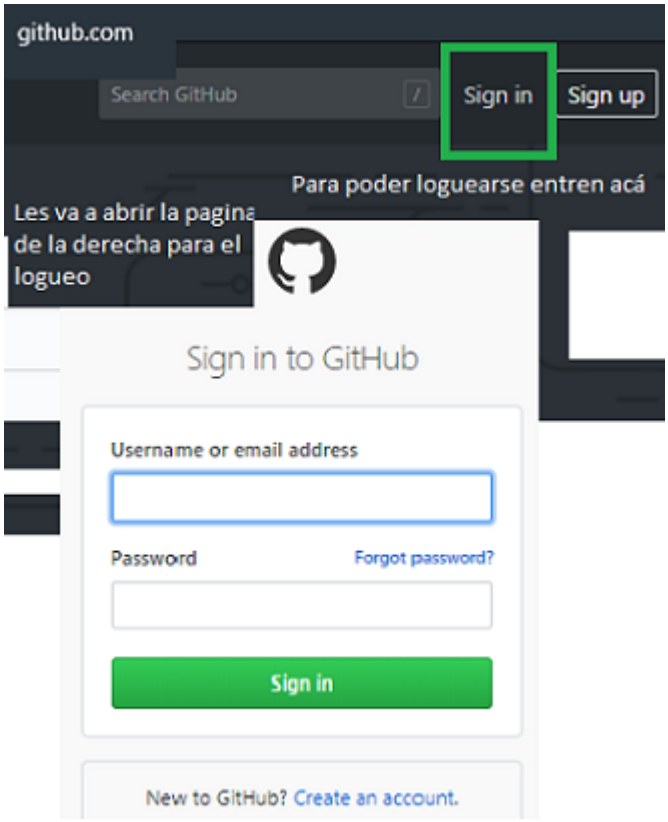

Figura 1. Pagina de github y logueo (aparecen en paginas diferentes)

Despues del logueo van a la solapita con un signo de suma y apretan en **New Repository**

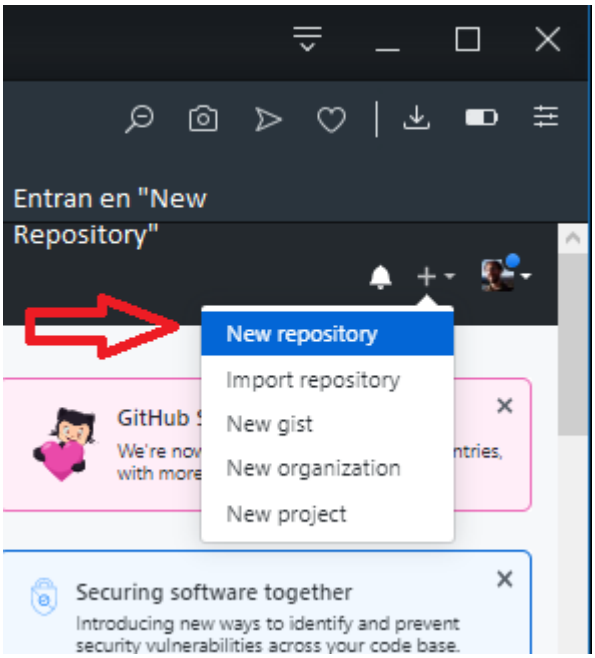

Figura 2. Una vez logueados podemos crear un repo nuevo entrando a ese Boton

Al entrar nos abre una ventana donde debemos tener en cuenta un par de cosas:

Primero que nada, tenemos que ponerle un nombre si o si al repo (esto es poque sino no sabe a que corno apuntan los datos).

Por otra parte, *yo prefiero en particular ahorrarme los pasos anteriores y crear primero el repositorio* con el *README.md* desde la pagina (estamos hablando ya de la otra forma de las que les hable), te ahorras un par de comandos y es basicamente lo mismo.

Los pasos anteriores sirven cuando ya tenes algo hecho y queres subirlo a un repo que ya existe o queres hacer un seguimiento de lo que fuiste haciendo a travez de los commits sin subirlo a un repositorio, en nuestro caso queremos trabajar con repos, por lo que les recomiendo primero crear el repositorio, clonar este en su pc (eso ya se los digo, es sencillo), poner los archivos que quieran subir y correr los comandos que venimos viendo de reconocimiento de estado, add y commit para poder subirlo y hacer el push directamente. Esto lo digo para que no se tengan que romper la cabeza para entrelazar la carpeta que creamos con las ramas del repositorio (origin y master) mediante comandos.

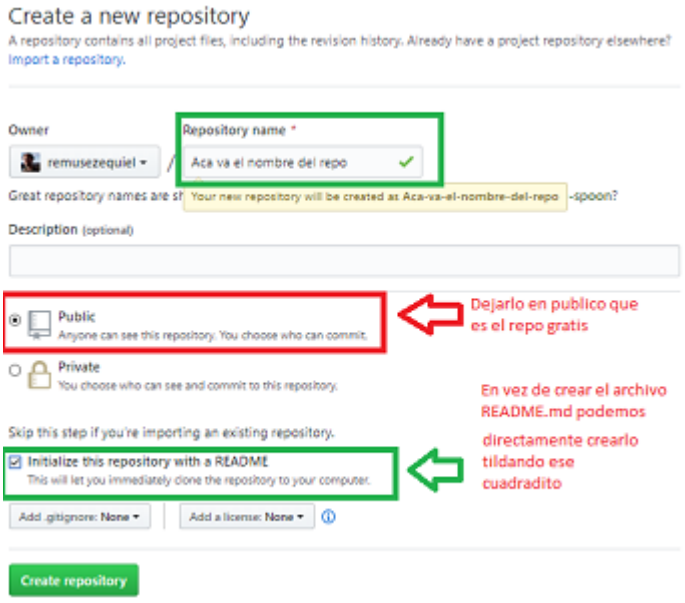

Figura 3. Creando el repo

Bueno, seguimos los pasos que aparecen en la *Figura 3* y listo, ponemos **create repository** y se crea el repositorio. Yo cree uno para poner este pdf y despues voy a usarlo para poner apuntes, les dejo el link a este repositorio en las referencias por si lo quieren clonar cuando suba mas cosas y sino para que sepan que tambien pueden clonar repositorios de otros sin modificarles nada a los demas y asi poder leer las cosas de un repoistorio x que les interese de una forma mas comodo.

Bueno, siguiendo con esto de los repositorios, una ves creado el repositorio nos manda a la pagina del repo, que van a ver algo como esto:

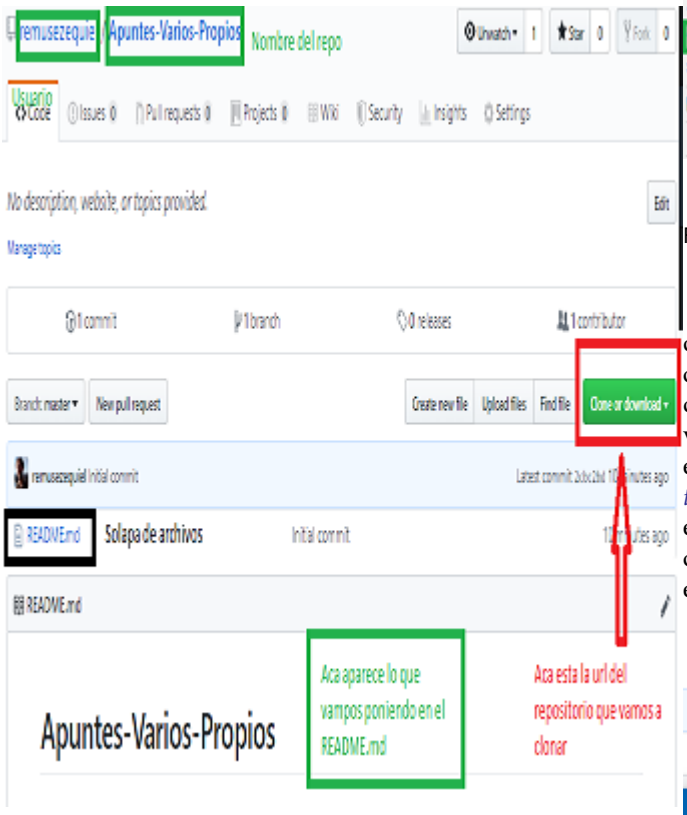

Figura 4. Repo nuevo

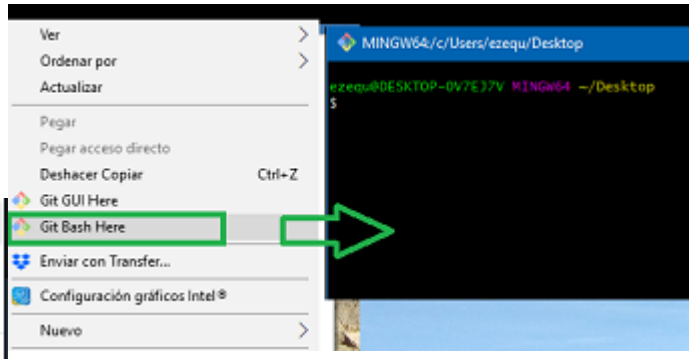

Figura 5. Terminal desde windows

Sino, si trabajan en linux y instalaron git con el comando *sudo apt-get install git* abren una terminal de linux y corren los mismos comandos que voy a poner ahora, solo que clonan el repositorio que quieran clonar. En este caso, voy a clonar el repo que hice para este apunte. Para clonar el repositorio se utiliza el comando *git clone urlDelRepositorio*. El url lo sacan de la solapa **clone Repository** que esta en el repositorio, lo copian y lo pegan despues del comando git clone. Verifiquen que esten parados sobre el escritorio, sino se los va a clonar en otra ruta.

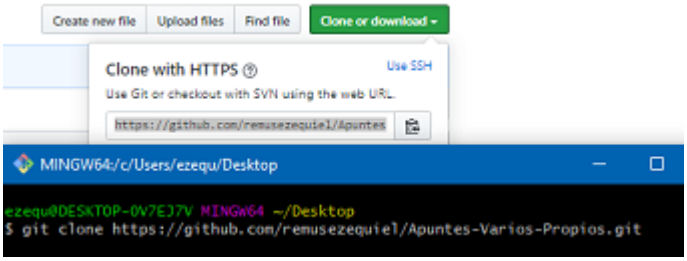

Figura 6. Clonando el repositorio en el escritorio

Ese es lo que se llama repositorio. Tenemos la parte de los archivos, donde ya de entrada vemos que aparece el archivo README.md, abajo de los archivos aparece lo que vamos escribiendo dentro del README (Si quieren no les ponen nada) este archivo es una guia sobre el repositorio, un resumen de lo que hay o muchas veces te explican como se usan o como se instalan los repositorios. Por ejemplo, estube viendo que el manejador de paquetes de Haskell *Cabal* tiene su repositorio de github (dejo el link en referencias).

Ahora, para trabajar con este repositorio, tenemos que clonarlo en nuesto escritorio. Para eso, abrimos la terminal desde nuestro escritorio. tenemos que tener en cuenta, que si instalamos git en linux abrimos una terminal comun y nos corremos al Escritorio y ya esta, en windows, si siguieron los pasos del tutorial que puse en referencias para instalarlo e instalaron esa terminal que dice el pibe seria asi:

Vamos a ver que nos clona la carpeta en el escritorio con el nombre del repo. Abrimos la carpeta y vemos que el contenido es el mismo mas la carpeta oculta *.git* de la que hablamos antes.

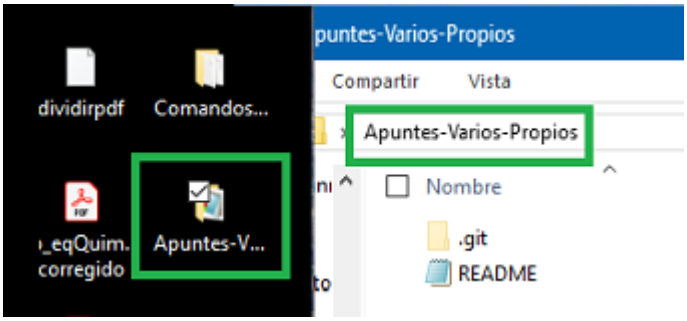

Figura 7. Clonando el repositorio en el escritorio

Ahora, lo que voy a hacer es poner la carpeta donde esta este pdf en el repositorio y voy a seguir los pasos de arriba:

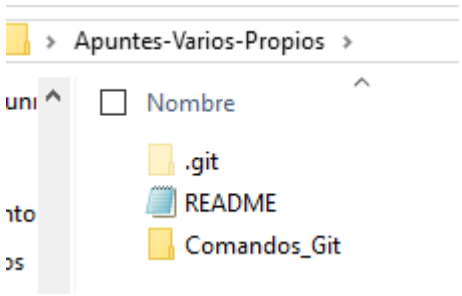

Figura 8. Pongo mi proyecto en el repo

Ahora, abro una terminal sobre el repositorio o me corro adentre del repositorio, pero asegurense de estar dentro de la carpeta del repositorio y corro *git status*.

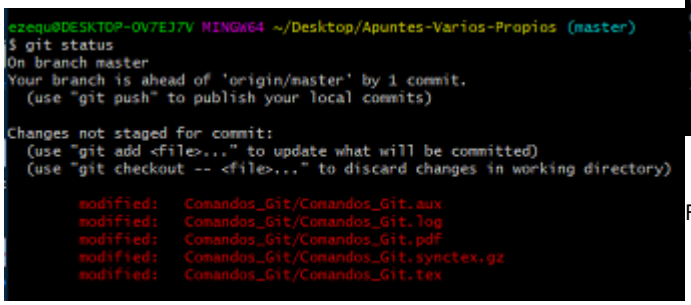

Figura 9. Realizo un status

Como vemos, esta en rojo y nos dice "a estas cosas les podes hacer el add . ". Hacemos el *add .* y corremos git status denuevo.

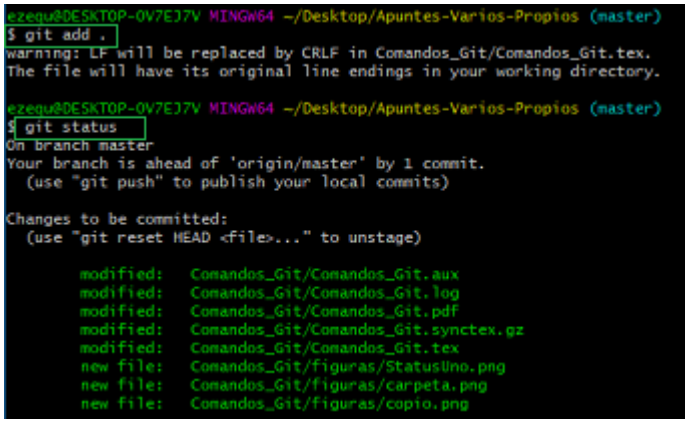

Figura 10. Realizo el add .

Ahora, hacemos el *commit*:

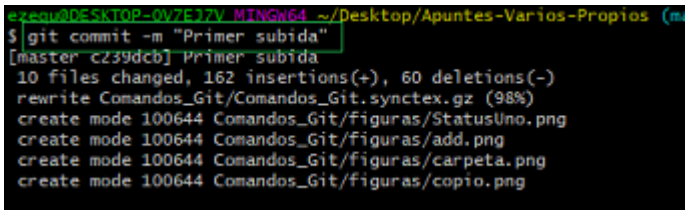

Figura 11. Realizo el commit pasandole el comentario *primer subida*

Y ahora es la hora de hacer el push lo cual se hace con el comando *git push origin master*. Este comando lo voy a explicar mejor un poco mas adelante.

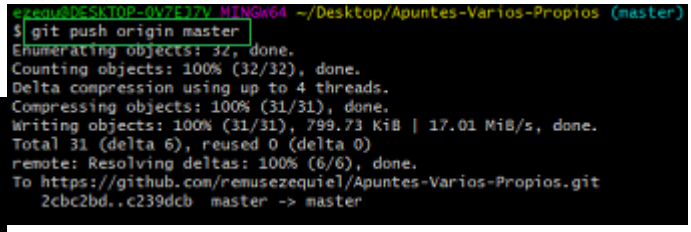

Figura 12. Realizo el push

Lo que es necesario entender hasta aca es que este comando lo sube a tu nube. Al ir a tu pagina en github, vas a ver que se renovo el repositorio de tu cuenta.

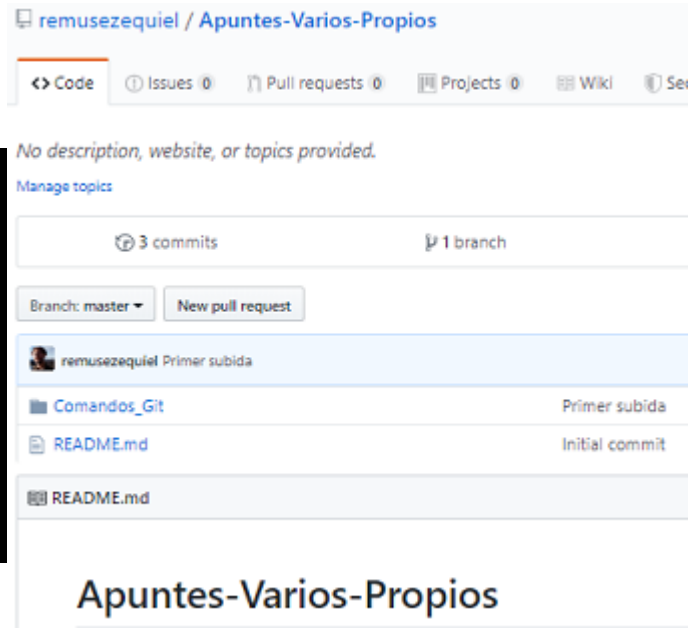

Figura 13. Realizo el push vamos a la cuenta y vemos que se subieron las cosas (refresquen la pagina)

3. Git Colavorativo

3.1. Introducción

Supongamos que estás trabajando en un proyecto y queres agrgar una nueva funcionalidad. La forma correcta es creando un *Branch* o *Rama* con el nombre de la nueva funcionalidad donde agregaremos nuestros cambios.

Por defecto, cuando creamos un repositorio tiene una rama llamada *master* (Como buena practica, nada debe desarrollarse en master, pero eso es otro tema). Para esto estan las ramas para experimentar y hacer cambios.

Un Branch (Rama), es una copia del proyecto, bajo el control de versiones, de forma que los cambios realizados en esta rama no afecten al resto del proyecto y viceversa. En español esto dice que el *Branching o Ramificación* es la manera de trabajar en diferentes versiones de un repositorio a la vez.

Un *Fork* es una copia exacta de un repositorio; creando uno nuevo en tu cuenta de Github con una url diferente que podemos utilizar como un repositorio git cualquiera. Tendremos dos repositorios independientes y cada uno evoluciona de forma autonoma.

Un pull request es una petición que el propietario de un fork de un repositorio hace al propietario del repositorio original para que este último incorpore los commits que están en el fork.

3.2. Pasos

Los pasos a seguir para trabajar de forma colaborativa son:

1. Hacer *fork* desde la interfaz de github del proyecto en el cual queremos colaborar.
2. Clonamos el repositorio al que le hicimos *fork*.
3. Creamos una nueva rama para desarrollar nuestro aporte
4. Realizamos los commits para describir las modificaciones u aportes.
5. Se hace un push de las modificaciones en nuestra copia del repositorio
6. Se pide un pull request desde la interfaz de github

3.3. Comandos

Vamos a tener que tener en claro que hace cada uno de estos comandos:

- *git branch nombreDeLaRama*: Crea la rama llamada nombreDeLaRama
- *git branch*: Enlista todas las ramas en el repositorio actual.
- *git branch -d nombreDeLaRama*: Elimina la rama especificada
- *git checkout nombreDeLaRama*: cambia a la rama especificada
- *git remote -v*
- *git remote add nombreQueVaATenerElRemoto url*
- *git remote rename old new*
- *git log –online*

3.4. Ahora a trabajar de verdad

Veamos como podemos aportar a un repositorio ya existente.

Imaginense que en mi cuenta tengo un repositorio en el cual vamos a trabajar los tres. Como el repositorio original, es decir el que tiene la rama master, es mio ustedes tienen que hacer la copia de mi repositorio en el suyo, esto se hace mediante el *fork*. Supongamos que el repositorio al que le tienen que hacer fork es el repositorio con url https://github.com/remusezequiel/Haskell-Algebra1, ustedes tienen que entrar al repo y darle al fork.

Figura 14. Boton del fork (disculpen la calidad horrenda de imagen)

Al apretar ahi se genera en su cuenta una copia de mi repositorio (Si no me equivoco es una especie de puntero, pero desconosco como se enlaza a nivel programación). Ahora pueden trabajar con una copia de mi repositorio como si fuera de ustedes, y si lo modifican no van a modificar mi repositorio directamente. Para hacer que las modificaciones que hagan aparescan en mi repositorio basicamente yo tengo que aceptar sus cambios, porque el repo original es mio. Esto se hace para que justamente, si vos queres sacarle info a un desconosido porque te sirve o queres aprender de su codigo y modificarlo a tu gusto no modifiques directamente sus cosas, solo se puede modificar si el administrador te lo permite. El hace la copia del repo, pero no lo actualiza todo el tiempo a menos que vos lo actualizes cuando te convenga.

Ya podemos clonar nuestro repositorio fork de la misma forma que lo hicimos en la sección 2.

Por defecto el al clonarlo el comando *clone* crea por defecto el remoto *origin* que apunta al fork que realizaste, es decir a tu repositorio. Fijense, corran el comando *git remote -v*, este les va a tirar algo parecido a esto:

```
ezequ@DESKTOP-0V7EJ7V MINGW64 ~/Desktop/Apuntes-Varios-Propios (master)
$ git remote -v
origin  https://github.com/remusezequiel/Apuntes-Varios-Propios.git (fetch)
origin  https://github.com/remusezequiel/Apuntes-Varios-Propios.git (push)
```

Nos interesa mantener un enlace entre el repositorio original y con el fork, por lo tanto tenemos que renombrar el remoto que apunta al repositorio propio, para hacer esto tenemos que usar el comando *git remote rename old new*, donde *old* representa el nombre actual del remoto (es decir origin) y *new* es el nombre que queres ponerle vos al remoto, ejemplo: *git remote rename origin papasConCheddar*, en este caso cambiamos el nombre del remoto pasando

de llamarse origin a papasConCheddar. Vean corran de nuevo el comando *git remote -v* y van a ver que en donde antes decia origin dice papasConCheddar.

Una vez hecho este paso tenemos que configurar el repositorio original, es decir el del tipo que esta administrando, para esto vamos al repositorio original en github y copiamos la url que usamos para clonar los repositorios, la del boton verde que dice clone or download. Una vez copiada la url, volvemos a la terminal y corremos el comando *git remote add nombreQueVaATenerElRemoto url*, ejemplo: *git remote add remusEzequiel https://github.com/remusezequiel/Haskell-Algebra1.git*. En particular puede ser el nombre que se les haga mas comodo a ustedes.

Luego de hacer esto ustedes van a tener su repositorio enlazado tanto con su repositorio como con el mio, pueden comprobarlo corriendo *git remote -v* (no les pongo fotos porque no llegue a hacer una cuenta random para poder hacer la ejemplificacion mas grafica)

Ahora, ya tenemos todo enlazado, asi que podemos empezar a trabajar modificando los archivos del repositorio. En particular, no se debe modificar las cosas estando parados en la rama *master*, por lo que **tenemos que crear una nueva rama y pararnos sobre esta antes de empezar a hacer una modificacion**. Hay un comando que al mismo tiempo de crearte una rama te deja parado sobre esta, el comando es *git checkout -b nombreDeLaNuevaRama*, ejemplo: *git checkout -b ramita*. Listo creamos la rama *ramita* y nos movimos a esta, es decir, pasamos de estar en master a estar en ramita. Ahora, podemos realizar los cambios en el proyecto. Una vez que terminamos de realizar los cambios seguimos los pasos para hacer un commit (status, add ., status, commit -m "comentario"). Antes de realizar otra cosa, corran el comando *git log –online*, este te enlista todos los commits, vas a ver que el primer commit que te aparece es el que le corresponde a tu ultimo commit y que aparece con *HEAD* apuntando a *ramita*.

Llego la hora de solicitarle al administrador que incluya los cambios en el repositorio original. Primero, tenemos que enviar la rama local al repositorio que le hiciste el fork, para esto corremos *git push remoto rama*, es decir, en nuestro caso la rama es ramita, porque estamos parados sobre la rama y el remoto es papasConCheddar. Ahora, los cambios que se van a enviar corresponden a la rama en la que estamos ubicados al momento de correr el push, en este caso la rama local se llama ramita. Como le dijimos que los cambios los realice en esa rama, el remoto se va a encargar de crear dicha rama si es que no esta creada todavia. Ahora, si van a su repositorio van a ver que existen dos ramas *master y ramita*. Al lado de la rama *ramita* hay un boton que dice **New Pull Request**, con ese boton le pedimos al administrador que acepte los cambios sobre el repositorio original (es decir, el repo del admin). Nos va a mandar a una especie de editor de mensajes donde podemos comunicarle al administrador que cambios hiciste y demas, por defecto se llena con el comentario del commit que hiciste. Luego apretamos en el boton verde de abajo que dice **Create pull request**.

Una vez hecho el *pull*, el administrador va a ser notificado. Ustedes lo unico que tienen que hacer es esperar a que el administrador acepte los cambios. El administrador los acepta haciendo una cosa que le dicen *mergear un pull request*, esto es crear un nuevo commit en el repo original. Una vez hecho el merge, github les va a mandar un mensaje diciendo que sus cambios fueron aceptados. Si van a su repositorio, van a la solapa de pull request y van a ver que en si todavia no esta aceptado va a aparecer como *open* y si esta aceptado aparece como *Merged*.

Luego, tengamos en cuenta que una vez que hagamos un fork y modifiquemos las cosas usando la rama ramita el repositorio nestro va a estar desactualizado respecto del repositorio original, por lo que tenemos que traer el nuevo historial del repositorio original a nuestro repositorio forkeado. Para esto nos movemos a la rama master con el comando *git checkout master*. Como el proyecto esta apuntando a dos remotos. Ahora tenemos que hacer un pull de origin con el comando *git pull origin master*, esto trae todo el historial nuevo de la rama master del repositorio original a la rama en la que estamos ubicados (master), luego tenemos que enviar ese historial a tu fork esto se hace con el comando *git push papasConCheddar master*. Y listo, ya tenemos todo actualizado. Luego, para mantener actualizado el repositorio respecto de los cambios que realize el administrador corremos el comando *git pull origin master* para traer los cambios del repositorio del administrador y para subirlo a sus repositorios solo deben corres *git push origin master*

REFERENCIAS

[1] *StackOverflow*, Link: https://stackoverflow.com

[2] *Link a un cursito de git basico*,
Link: https://www.youtube.com/watch?v=m4wh8GhzcYg&list=PLmUnyBCRHkvUPkrsseI1SmMtYgfc-f8Kn&index=3

[3] Cuenta propia de github .
Link: https://github.com/remusezequiel?tab=repositories

[4] Repositorio de github donde se encuentra el codigo de este proyecto .
Link: https://github.com/remusezequiel/Apuntes-Varios-Propios

[5] Repositorio de github donde se encuentra el proyecto Cabal .
Link: https://github.com/haskell/cabal

[6] Mas links .
Link: https://guides.github.com/activities/hello-world/?source=post_page-----62dea67aa2eb----------
Link: https://www.linuxito.com/programacion/890-como-mantener-tu-fork-sincronizado-con-upstream-en-git
Link: https://blog.muktek.com/git-colaborativo-fd8baafb8614