Git – Sistema de control de versiones

Es una herramienta rastreadora (terminal).

Todos los cambios y las modificaciones se guardan y las puede rastrear

No significa que clone el proyecto las veces que lo guardes, si no que por cada cambio encapsula el conjunto de líneas de código que modificaste en el registro commit

Puedes regresar a un estado anterior del proyecto o conocer la evolución que ha tenido

1. Registra y guarda los cambios
2. Te da acceso al registro
3. Moverte hacia atrás o hacia adelante

Se puede rastrear cualquier archivo, pero los PDF y Zip, los muestra con código binario

Tu avisas al programa en qué momento se guardan los cambios

ARQUITECTURA DE ARBOL

Existen 3 areas en local:

* Working área (se usa **git add**, para subir a staging area)
* Staging área: Es como el área de preparación (Se usa **git commit** para subir al repository). Donde se escoge qué archivos irán al commit. Gracias a esta área, no se tienen que subir todos los archivos que editas, si no únicamente los que se seleccionaron.
* Repository

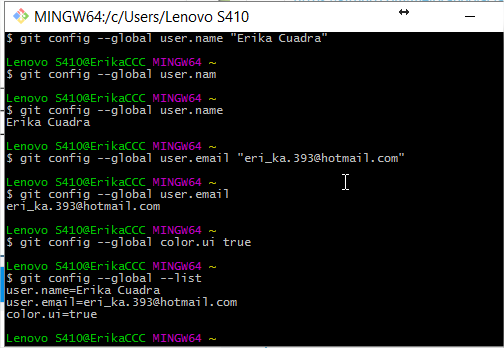
Los cambios son llamados commits (Son las líneas de código que modificaste y se encapsulan)

El conjunto de cambios es el repositorio

GitHub – Es la plataforma de hosting de proyectos

Git comandos:

Para hacer modificaciones al registro se utiliza:  como se configura en la siguiente imagen



En nuestro equipo hay 3 niveles donde se desarrolla registros de configuración de GIT:

System – git config –-system (Configuración para todos los usuarios) regularmente se encuentra en esta ubicación 

Para User – **git config --global**

Para Project – **git config**

Comandos:

**git status** – te dice los cambios que ha sufrido el proyecto y los archivos

**git add –A** guarda todos los archivos y los empieza a rastrear. Se suben del working área, al staging área. Significa agrega todos los archivos al staging

**git commit –m “comentario”** con este comando se suben al repositorio

**git log** nos dice cuantos commits se han metido en el repositorio, quien los hizo y cuando los hizo

**git config –global –list**: para ver lo que configuraste

**git help [comando]:** te ayuda a saber qué hace el comando que ingresaste

**git push**: se utiliza para subir a GitHub o a servidores

\*atajo\*: **git commit –am:** Sube los archivos y te puedes saltar el **add –A.**  Agrega los archivos por default. Ej.: **git commit –am “Cambio de titulo”**

Checkout

**git checkout**. Nos ayudara a regresar en el tiempo, es decir a saber como se encontraba el proyecto en determinado punto, esto basándose ó con la ayuda del numero de commit, ejemplo:

 por lo tanto al ingresar este comando, si refrescas el proyecto, te mostrará cómo se encontraba en ese momento la página y el código. Pero te puedes regresar a lo último que has hecho. También nos ayuda a movernos entre ramas.

Y para regresarte a ver la última versión del proyecto, es decir, como salirte del checkout, se usa

**git checkout master**

**git reset** lo mismo que git checkout, pero este si plancha o borra los cambios siguientes al cambio que te decidiste regresar. Te borro lo que tengas hasta el punto que tú me digas.

Tiene 3 tipos de ejecución:

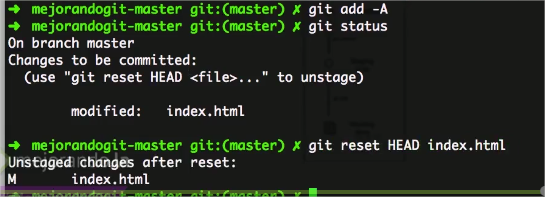
**git reset –soft**: un reset “tranquilo”. Protege que no borre del sublime el código, no se mete con tu local. Hace los cambios en el repositorio, pero te deja con el código en el sublime

**git reset mixed**: hace lo mismo que el anterior pero los cambios los hace en el Staging área, y de igual forma te deja el código en tu working área. Refleja lo que hay en el repositorio con el staging área. Es como una mezcla de soft y de hard.

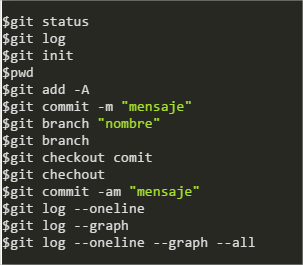
**git reset hard**: Borra todo, incluso lo que estás trabajando en tu sublime, en tu área local. Existe una forma de recuperar lo que se borró, solo si hubiste guardado los commits que hiciste.

\*\*\*\*Una forma rapida de acceder a la carpeta "mejorandogit-master" desde GitBash, es que ya teniendo la carpeta en el escritorio, le das click derecho y clickeas la opcion "Git Bash Here" e imediantamente te abre el bash con la ruta de la carpeta\*\*\*

\*\*Cuando le das **add –A** para subirlo al staging y te das cuenta de que te equivocaste, puedes revertirlo con el comando que te manda al hacer el **add:**



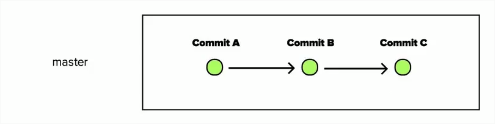
**git remote**: Te ayuda a moverte a diferentes servidores o plataformas



RAMAS (branches)

Es una línea alterna del tiempo de nuestro repositorio para hacer como propuestas

Podemos resolver bugs, experimentar, etc, sin afectar la línea o versión estable .

master: Es la línea estable del proyecto, es donde trabajamos la primera clase 

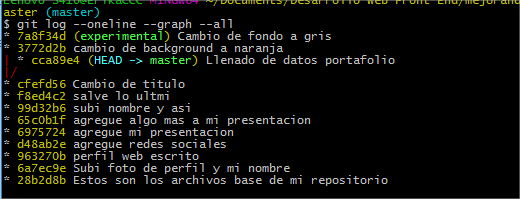
Cuando generas una rama alterna, y llega un punto en el que lo quieres fusionar, se toma la última versión de ambas ramas y se fusionan

**git branch nombreRama:** Se crea la rama

**git checkout nombreRama:** Nos ayudara a movernos a la rama

HEAD: Señala el punto de la historia de nuestro proyecto nos encontramos

**git log --oneline:** nos muestra únicamente los commits de las ramas

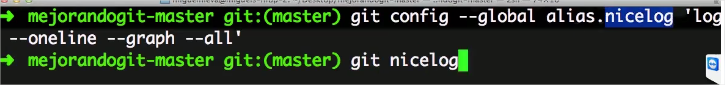
Con **git log --oneline --graph –all:** Nos muestra gráficamente las ramas de nuestro proyecto: 

Git maneja los “alias” nos ayuda a automatizar los comandos. Te ayuda a decir que comando

quieres que se ejecute con otro comando que tu creas. Ejemplo

**git config --global alias.nombreAlias ‘comando’**

Ejemplo: **git config --global alias.niceLog ‘log --oneline --graph –all**

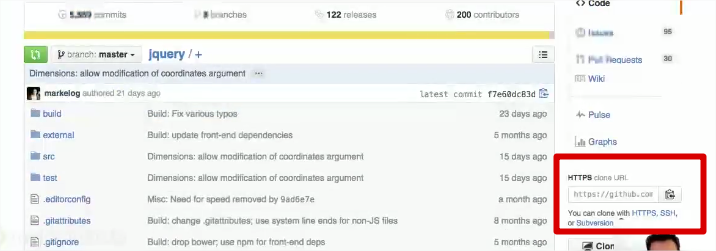


WORKFLOWS

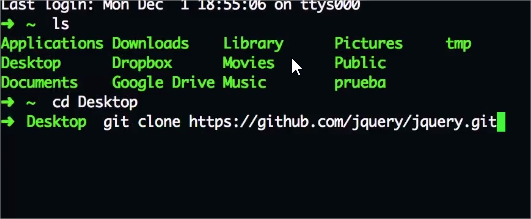
Son flujos de trabajo colaborativos. Es la forma en la que se integra y organiza el equipo para la creación de un proyecto web

Proyectos de terceros

Se pueden clonar proyectos para tenerlos en el equipo local jalándolos de gitHub: **git clone [https ssh]** y **git log**

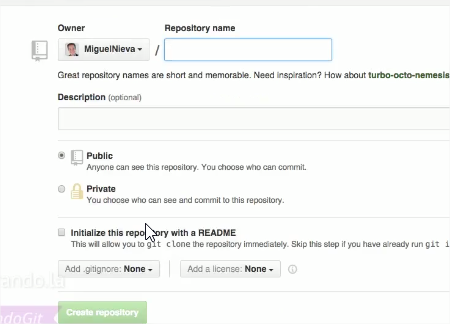
Se copia la url HTTPS del proyecto de gitHub 

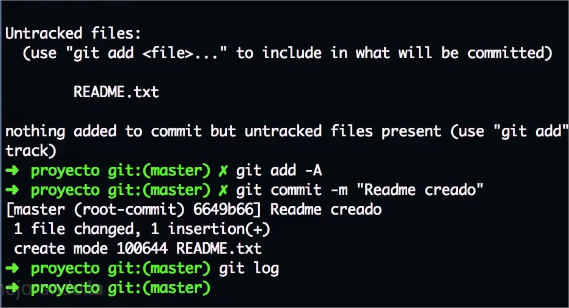
Posteriormente en la consola se pega el link con el comando para clonar el proyecto:

 y se baja el proyecto en la carpeta

Tambien se pueden bajar con .**zip**, pero no vendrían incluidos los commits como con **clone**

Repositorios propios para subirlos al repositorio

1. Primero se crea un repositorio en gitHub para subir allí el proyecto: 
2. Se realizan los commits de lo que quieres subir a tu repositorio:



1. Se enlazan servidores con el comando **git remote**:



* + **origin:** es el nombre del remoto. Siempre que te conectes al remoto es recomendable usar origin, EN PROYECTOS PROPIOS.
  + **https://github.com/ErikaCuadra/repo-mejorandola.git**: Es la liga del repositorio que creamos en GitHub y donde se subirán los archivos.

1. Para mostrarnos los enlaces generados vía repositorios usamos el comando: **git remote -v**
2. Posteriormente para empujar nuestros cambios, es decir, subirlos: **git push origin master**
   * **origin**: es el remoto, a donde se va a mandar
   * **master**: la rama que vamos a mandar

Repositorios propios (con equipo de trabajo)

Igual que el anterior pero primero se bajan cambios

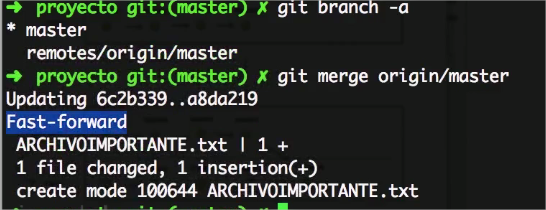
Existe una rama escondida: **origin master (se encuentra en nuestro local),** la cual **espejea** el repositorio remoto a tu repositorio local, esto lo hace con **git fetch**

En este proceso lo que pasa es que la rama **origin master** de nuestro repositorio local, espejea de la rama **master** del repositorio remoto, así que tendrán lo mismo.

Así la rama **master** de nuestro local se tendrá que fusionar con nuestra rama local **origin master**, que es la que tiene la copia de la rama **master** del repositorio remoto.

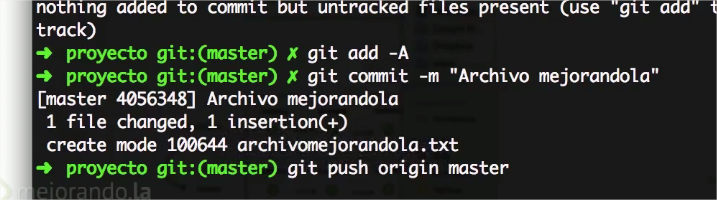
Una vez fusionados nuestros cambios con los del repositorio remoto, podremos subir nuestros cambios.

1. Se deben bajar cambios primero con el comando **git fetch origin.** En este momento ya descarga los cambios a mi local, pero a mi rama escondida **origin/master** no a mi rama **master,** por lo tanto, no veremos aun en nuestra carpeta los cambios que hemos bajado del repositorio remoto.
2. Como ahora los cambios están en la **origin master**, lo que hacemos es fusionar, hacer merge a la rama **origin master**: **git merge origin/master**
   * Este comando se aplica desde la rama donde se va a absorber (**master**).
   * **origin/master:** la rama que se fusiona, que es donde se bajaron los cambios.



\*Para que nos muestre todas las ramas usamos el comando **git Branch -a\***

En este momento ya tendremos los cambios abajo en nuestro local.

1. Ahora para subir lo nuestro, se sube con el commit como siempre luego se usa el comando **git push origin master:** 

En este momento ya subieron nuestros cambios.

Repositorios forked (de terceras personas)

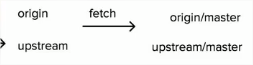
Cuando generamos la copia de un repositorio de alguien más en nuestro perfil de gitHub (con **fork**).

Se necesitan dos repositorios, uno donde tenemos que seguir actualizando lo que suban en el proyecto del tercero (principal), y otro donde podamos seguir subiendo nuestros cambios (personal) que es el resultado de haberle dado **fork**.

El personal lo vamos a conectar o enlazar con la rama **origin**.

El principal que es donde se reflejarán los cambios se conecta o enlaza con la rama **upstream** donde no podremos hacer cambios, ya que es donde se bajan los cambios del tercero.

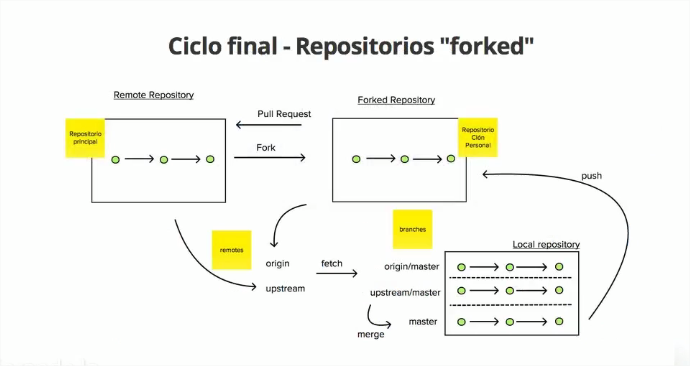
Se generan dos **fetch** para cada caso

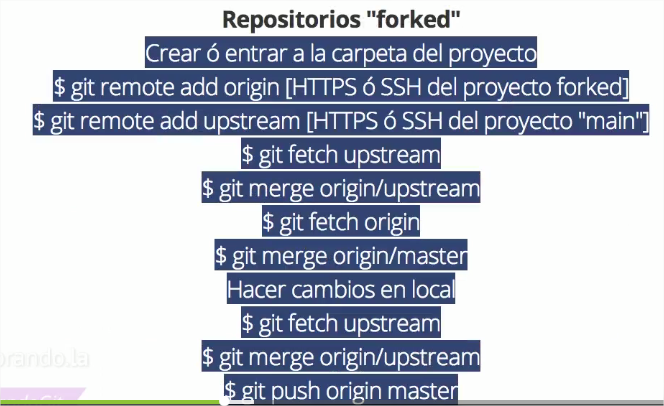


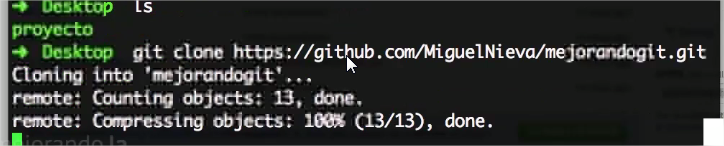
Tenemos estas dos ramas en nuestro local escondidas **origin/master** y **upstream/master**

Estas dos ramas escondidas tienen que hacer fusiones con nuestra **master** (con **merge**)

Una vez que lo tenemos en nuestro local, lo subimos con **push** a nuestro remoto para posteriormente mandar el **pull request** al remoto de otra persona, del que le hiciste fork que es como la propuesta de mi desarrollo.



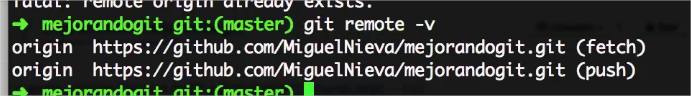


Despues de hacer el **fork** clonamos el proyecto: **git clone** 

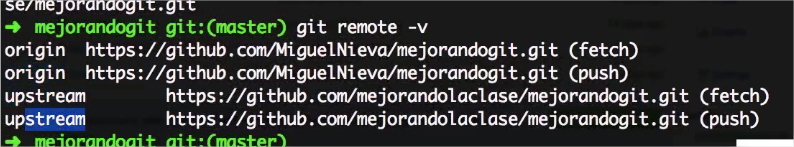
Ya lo tenemos en nuestro local, y realizamos los cambios por ejemplo en el **index.html** y se agregan los **commits** para subir los cambios del archivo.

Ahora se concectan los remotos, tanto el oficial, que es el de terceros de donde hicimos el fork o de donde lo copiamos

Primero es **git remote add origin https://liga-de-mi-proyecto.git** (para conectar el repositorio propio)

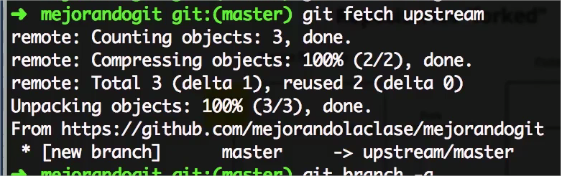
Luego **git remote -v** (conecta nuestro proyecto a gitHub)

Ahora nos falta el remote de upstream -> **git remote add upstream httpd://liga-de-proyecto-principal.git**

Luego **git remote -v** y vemos que ahora tenemos las dos conexiones remotas, hemos creado dos enlaces remotos: **origin** y **upstream**

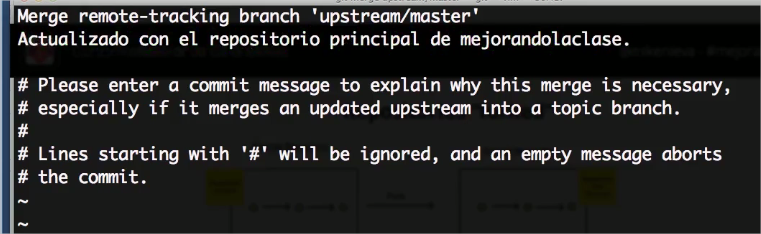
Posteriormente sigue **fetch** para espejear las dos ramas ocultas que tiene el proyecto: **git fetch origin**

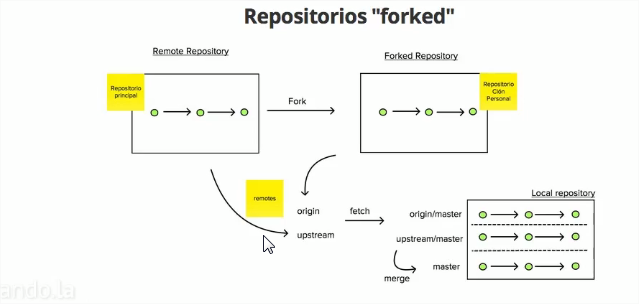
Si se le agregara un archivo al proyecto principal, nosotros aun no lo tenemos en nuestro local, por lo tanto para actualizar en nuestro local y tener lo mismo utilizamos el **fetch:**

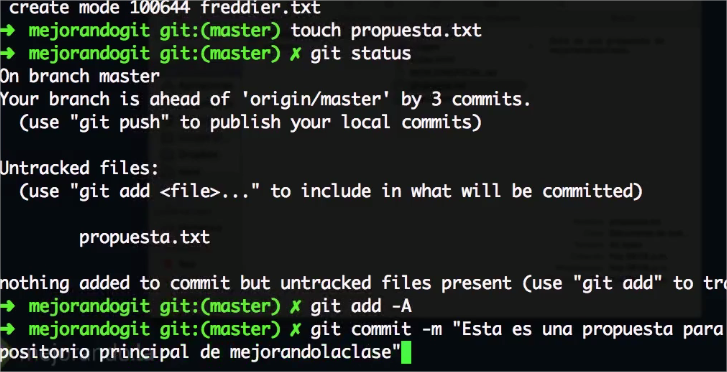
**git fetch upstream** 

Despues de esto, ahora ya tenemos los cambios que subieron al principal, en nuestro local **upstream**

Ahora lo que queremos es hacer una fusión de **upstream** con **master** para actualizar con nuestra rama principal: 

Ponemos ahora el comentario de nuestra fusión: 

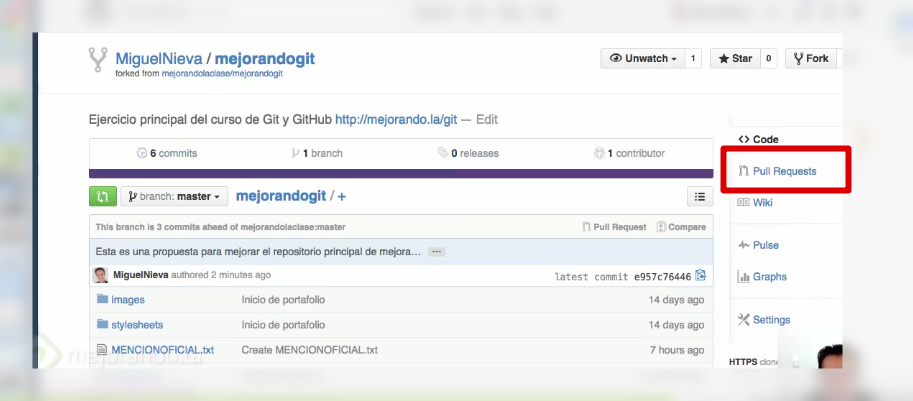
Esto es lo que se ha hecho: 

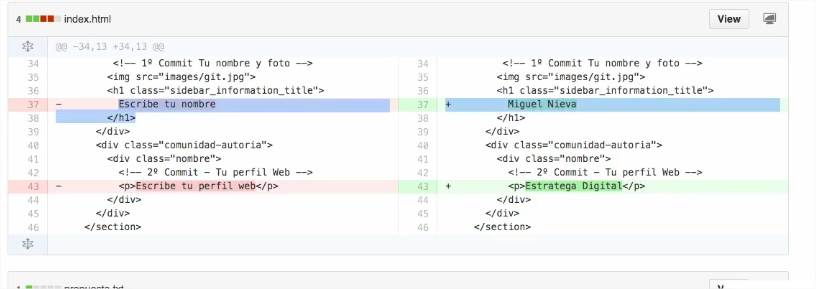
Ahora como todo está en nuestro local, lo tenemos que subir pero editamos antes el cambio que queremos mandar como propuesta: 

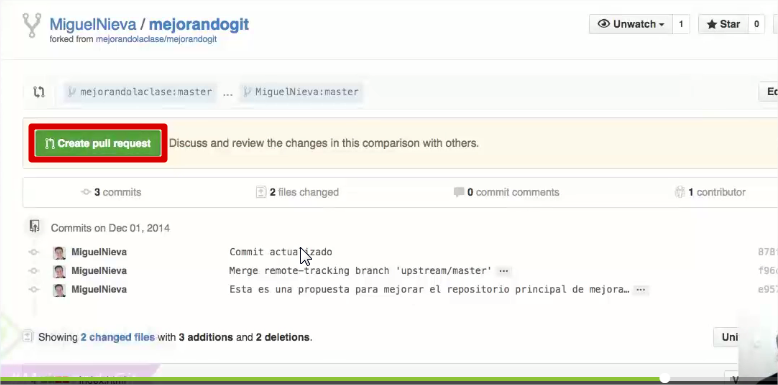
Ahora hemos bajado todo y creado nuestra propuesta, lo siguiente es subirlo al repositorio fork con el **push:**

**git push origin master** (origin es el que esta vinculado con el repository y master es con la principal, con la que se esta trabajando, por lo tanto los cambios principales siempre deben estar en master)

Ahora ya esta en el fork repository, lo siguiente es el **pull request** para mandar nuestra propuesta al proyecto principal:



Con esto, nos muestra los cambios que hemos realizado para la propuesta: 

Ahora creamos pull request: 

Escribimos comentario y descripción y lo mandamos, y esperamos a que el dueño del proyecto principal acepte la propuesta que dimos.

git remote -enlaza servidores-areas locales, conectamos nuestro proyecto a github  
git remote add origin -- origin es el nombre del remoto (origin-> convencion, Buena practica? :D)  
git remote -v   
git push  
git fetch -actualiza la rama origin/master local, mas no la master local  
git merge -despues de la actualizacion, fusionamos el commit que tenemos al proyecto actualizado de github, asi podemos hacer push sin problemas.  
git branch -a muestra todas las ramas que hay, por ejemplo nos aparece la origin master que esta escondida

AMBIENTES FTP

Se usa para hacer el proceso de liberación para producción.

La desventaja es que es un protocolo de transferencia de archivos, es decir, los pasa, pero no los guarda. Se puede perder todo, si el servidor de producción se va al carajo.

Es mejor usarlo solo para **binarios** (imágenes, blob y raw, mp3, zip, tar, rar), no para generar control de versión de sistemas.

Como no hay control de versiones, se pueden encimar los códigos, y en caso de que quisieras regresar en el tiempo, no se podría.

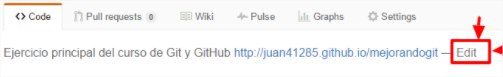
GITHUB PAGES

El despliegue con GitHub Pages es automático.

El repositorio remoto, que es el ambiente de GitHub tiene dos ambientes: Central (donde se guarda el código) y Producción (se puede vincular con un dominio www y al final se conecta con SSH)

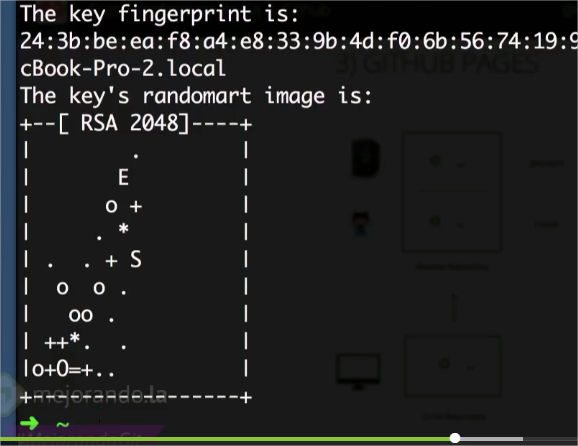
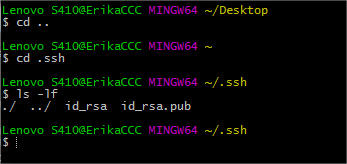
La única forma de subir avances al proyecto de gitHub es vía SSH secure shell.

SSH es un protocolo de seguridad y **para subir** de tu local al repositorio remoto, lo que hace es que por medio de un código se vinculan los dos ambientes como sincronizando el código de cada uno para, como un candado y una llave cuando los dos códigos se encuentren, se abren camino para poder subir las cosas.

Para hacer un despliegue rápido en GitHub, se hace desde la rama **gh-pages**, desde “Edit”: 

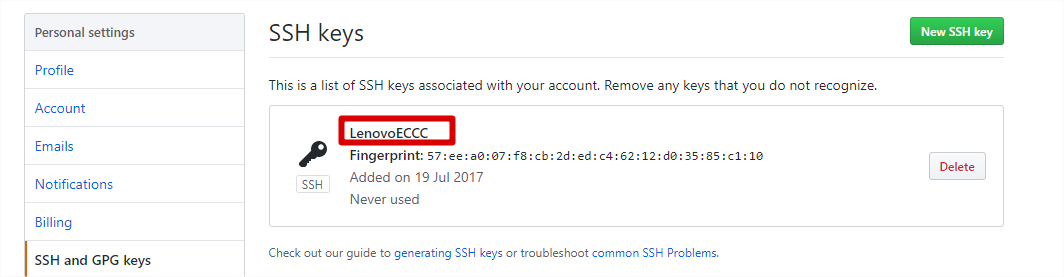
La sintaxis es: **nombreUsuario.github.io.nombreProyecto**

Para poder subir vía SSH se debe conectar el github con la computadora, para que github sepa que esa computadora es tuya y para que github lo sepa, es necesario que tu tengas la contraparte de la llave:

1. Se debe generar una llave: **ssh-keygen**, después te pregunta si quieres crear la llave y t pide una contraseña dos veces y luego se genera lo siguiente: 
2. Nos vamos a la carpeta **ssh** de una carpeta atrás a la de nuestro desktop y nos mostrarán dos archivos: “**id\_rsa**” y “**id\_rsa.pub**“ 

**rsa**: es un tipo de encriptación.

Se muestran dos porque una es publica (**pub**), la que compartes a los lugares donde quieres generar la conexión, y la primera no se comparte.

1. Imprimimos la llave **id\_rsa.pub** y la copiamos para poder generar una conexión en github: 
2. Nos vamos a “Settings” de github y creamos una llave nueva, le ponemos nombre y pegamos el contenido de la **llave publica**: 
3. Ahora nos vamos al repositorio y copiamos la liga SSH, para hacer la conexión con el repositorio como se ha estado haciendo en https:



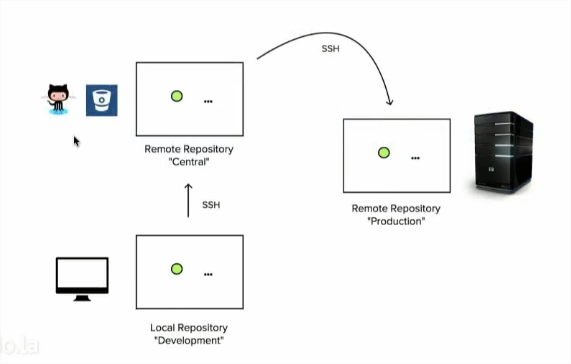
1. Con esto hemos hecho la conexión y ahora hacemos el pull con **git pull origin master** y nos pedirá la contraseña que pusimos al crear la llave. **El pull es como para descargar las ramas**.
2. Después vamos a descargar la rama de gh-pages que creamos en el repositorio: **git pull origin gh-pages**

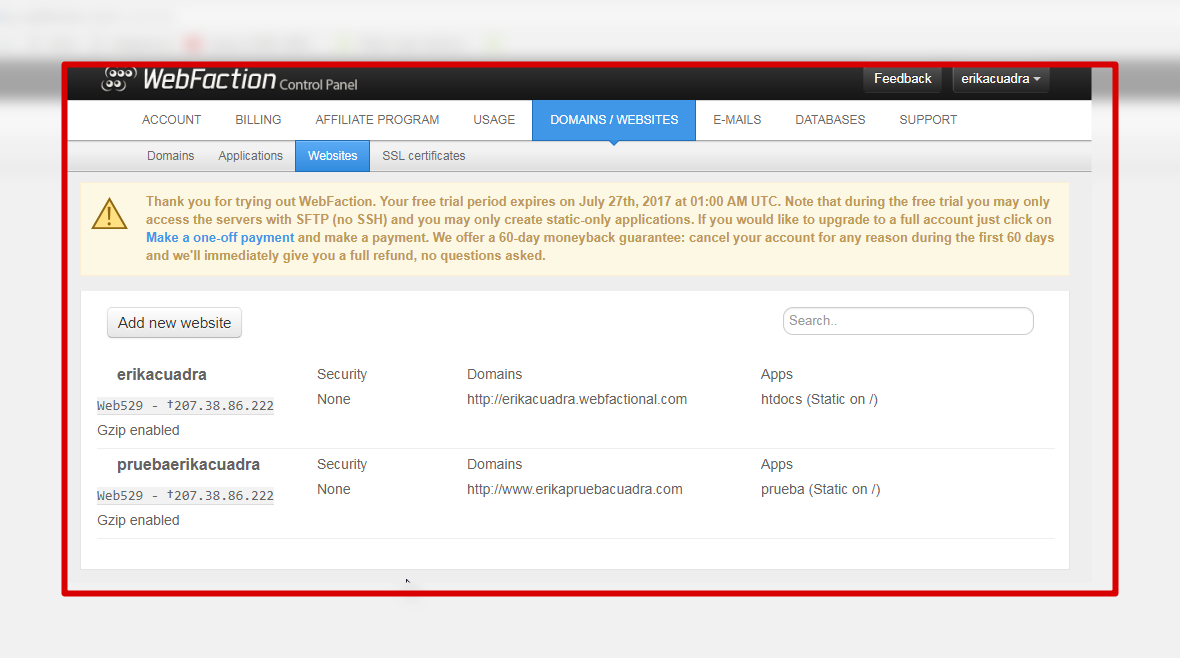
Debemos tener en nuestro local la rama **gh-pages**, **master** y con el **pull** ya debemos tener copiado cada rama del repositorio remoto. De manera que al buscar las ramas con **git Branch -a**  nos muestre lo siguiente

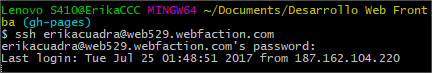
\*\*Los cambios se deben hacer en la rama de **gh-pages** para que se reflejen en la liga de producción\*\*

MANUAL DEPLOYMENT (Para un equipo)

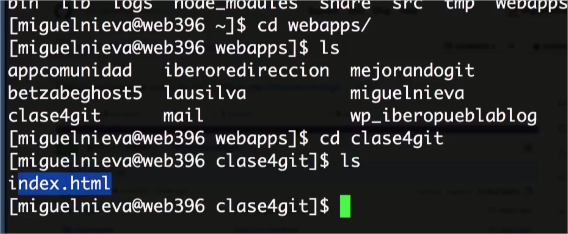
Se puede hacer un despliegue de nuestro local a nuestro repositorio central remoto (GitHub, etc), y posteriormente del nuestro repositorio remoto desde gitHub, podemos hacer un despliegue a producción, pero por seguridad, todo esto mediante SSH:

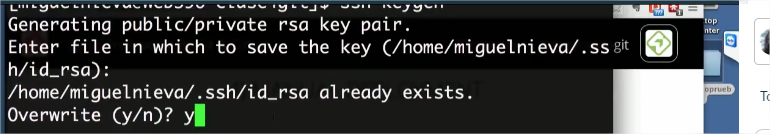
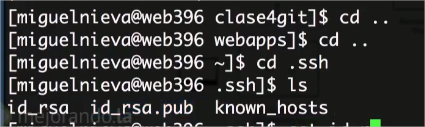
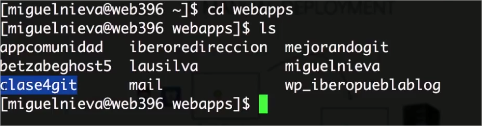
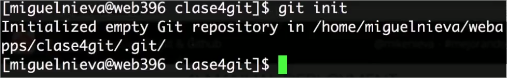


1. Primero se configura el servidor que se usara de producción, es decir, el hosting que vas a usar para alojar tu proyecto. Elegimos webfaction.com 
2. Nos conectamos vía **ssh** desde la terminal a la dirección que nos proporciona el hosting webfaction con el comando: **ssh nombreusuario@web529.webfaction.com**

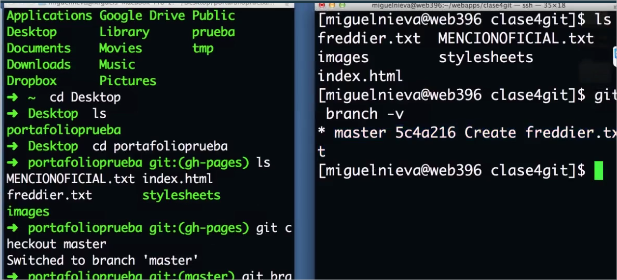
nombreusuario: se refiere al nombre de usuario ssh y posteriormente te pide la contraseña que le pusiste a esa “llave”: 

1. Una vez que estamos dentro del servidor, enlistamos los archivos con **ls** (los siguiente es dentro del servidor)
2. Vamos a la carpeta **webapps**, donde se guardan nuestras aplicaciones y nos ubicamos a nuestro proyecto:



1. Se genera automaticamente un **index.html** con un hello world
2. Ahora vamos a generar la llave ssh (**ssh-keygen**) sobreescribiendo la anterior que habíamos hecho 
3. Sobre nuestro servidor nos vamos a la carpeta de **.ssh** y nos enlista las llaves 
4. Ahora imprimimos la llave con **cat llaveLiga.pub** para copiarla, y me voy a SSH de Settings de git hub para crear una en la que le pegamos la llave que creamos en el servidor, y **no** se mostrará activa
5. Ahora para poner lo que se encuentra en nuestro git hub, se pasará a nuestro servidor, entramos a **webapps** donde creamos nuestro proyecto para enlistar si se hicieron y nos vamos al proyecto que hicimos: 
6. Si se generó un index.html lo borramos. Ahora activamos un repositorio Git en nuestro servidor: 
7. Hacemos la conexión remota con **git remote –v**. Nos vamos a git hub y copiamos la liga ssh de nuestro proyecto para enlazarlo: **git remote add origin ligaSsh.git**  y comprobamos que tenemos conectado el repositorio de git hub con el del servidor
8. Ahora se jala la info de github al servidor: con el comando **ls –lf** nos mostrará la carpeta oculta de git que creamos en el servidor: 
9. Como ya hicimos la conexión hacemos el **git pull origin master** y nos pide la contraseña de la llave que creamos: 

**En este momento se ha hecho el deployment**

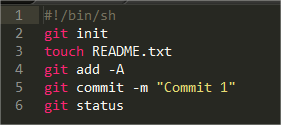
1. Ahora si queremos hacer cambios en nuestro proyecto, abriremos **dos terminales**, una para local y la otra para el servidor: 
2. En ambas terminales nos ubicamos en la rama master, y ahora nos **sincronizamos** con git pull origin master en la terminal del servidor, para que estén iguales ambos.
3. Hacemos el cambio que queremos en nuestro local, y subimos como siempre con el **commit** y eso, y podemos comprobar el cambio en github. Ahora sigue pasarlo al repositorio de nuestro servidor (produccion). Ahora subimos todo con **git pull origin master.** Con esto ya podemos ver nuestros cambios en producción

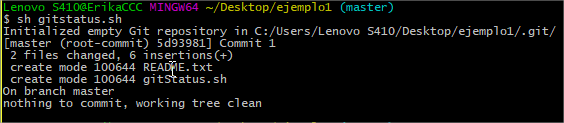
Servidores recomendados: webfaction, digital ocean, amazon web services, heroku (un poco dificil)

SHELL SCRIPTS

Comandos encapsulados dentro de un .sh

Si tienes problemas con sh en Windows, descarga Cygwin

Se crea un texto.sh y se meten los comandos que necesitas: 

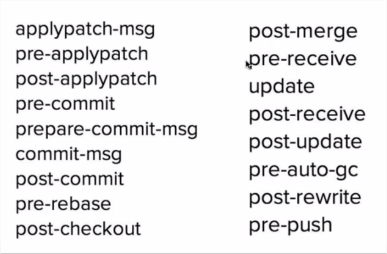
Para ejecutarlo, desde el bash se pone el comando: **sh nombreArchivo.sh** 

**GIT HOOKS**

Son Shell scripts (sh) pero se ejecutan antes, durante y después de un movimiento con git. Pueden trabajar con diferentes intérpretes.

Usaremos Shell (.sh).

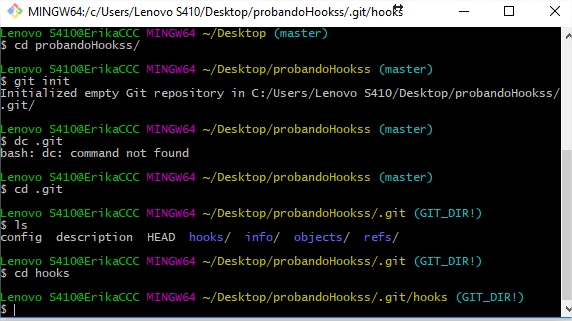
Existen 17 “hooks”:



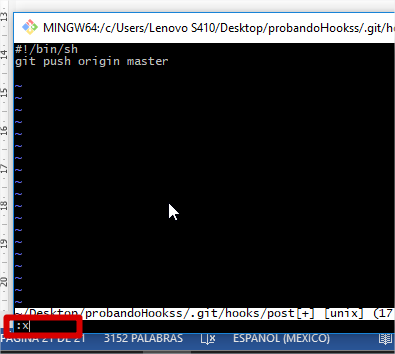
El post-commit: es una serie de comandos que se ejecutan después de generar un commit.

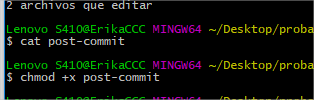
Después de que hagas un commit, se va a ejecutar tu .sh

Para un ejemplo, creamos una carpeta e inicializamos con **git init**

Nos vamos a los archivos ocultos de **.git**  con **cd .git**  y enlistamos (**ls**) y nos metemos a la carpeta de **hooks** 

Creamos el archivo **post-commit** y nos metemos al editor de texto de la terminal: **vim post-commit** y con **ctrl+o** podremos escribir y cerramos y guardamos





Modificas el archivo y lo subes

