Curso de git y github

**Git** es un sistema de control de versiones

**Github** es una plataforma donde podemos guardar nuestros proyectos y usar **git** para dministrarlos

**En git hay tres estados (por asi decirlos)**

**Working directory:** directorio donde yo estoy trabajando con todos mis archivos de mi proyecto

**Staging area**: donde voy a estar agregando todos los archivos que voy a preparar para el guardado

**Repository:** donde guardo mis archivos

**Comandos básicos**

**Git help** es una ayuda para ver los comandos de git

**Git help nombreComando** ayuda específica de un comando

**git init**: para crear un proyecto nuevo con git

**git add** <file>: para pasar los archivos del working directory al stagin area

**git add**  . Agrega todos los archivo que he incluido en mi proyecto

git add –A Agrega todos los archivo que he incluido en mi proyecto

**git status**: para ver el estatus de los archivos

**git commit**: para pasar los archivos del staging area al repository, para hacer nuestro primer snapshot

**git push**: para subir a un repositorio remoto

**git pull:** cuando trabajo con multiple desarrolladores con este comando me traigo los últimos cambios que han hecho el resto de los desarrolladores. El git pull es una combinación de fetch +merge

**git clone:** me crea una copia del servidor donde esta el código a mi repositorio local para comanzar a trabajar

**git config** --global user.email [henrycr1982@gmail.com](mailto:henrycr1982@gmail.com) para decirle a git quien soy, le digo que el usuario es mi correo electrónico.

**git config** --global user.name "henrycr" para decirle mi nombre de usuario

**git config** --global user.name para ver el nombre de usuario que esta logeado

**git config** **--global color.ui true** para configurar los colores de los mensajes que me arroja git

**git config --global --list** para ver la configuración que tenemos en git

**pwd** para saber en que directorio estoy

cuando hago un **git commit** se me abre un editor de texto, para poder escribir presiono la tecla “I”, luego que escribo un mensaje presiono la tecla “Esc”, luego presiono “:wq”

si escribo **git commit –m “mensaje que deseo guardar”** me evito abrir la consola para escribir el mensaje.

el comando **git log** este comando me da un hash que me permite identificar cada snapshot de mi proyecto

**git checkout -- welcome.blade.php** me lleva a mi copiar anterior que esta localmente guardada después de yo haber hecho un cambio local .

**git diff welcome.blade.php** para ver las diferencias entre los archivos que he modificado localmente

en la carpeta raíz de mi proyecto cuando creó un archivo .gitignore en el puedo almacenar los archivos que nos deseo que estén en mi repositorio en la nube.

**Git branch** me muestra las ramas de mi proyecto

**Git branch nombreDeLaRama** me crea una rama, esta rama se crea con la información del ultimo commit

**Git checkout** **nombreDeLaRama** me permite seleccionar la rama de mi proyecto con la que deseo trabajar

**Git checkout –b nombreRama** me crea una rama y se mueve de una vez a ella.

**Git checkout** me permite moverme entre commit y ramas de mi proyecto

$ git config --global http.proxy http://castrohs:castrohs45d@167.134.148.27:8080

PARA SETEAR EL PROXY

Formato markdown

**git checkout -- código SHA** me lleva a la versión que deseo (de los commit que he realizado)

**git checkout master** me mueve a mi ultimo commit

**git log > commits.txt** me genera un archivo que contiene todos los commits que he realizado

**git reset --soft** **código SHA** el git reset mas simple y que no toca nuestro “Working Area” (No se mete con nuestro código) .

**git reset --mixed** **código SHA** este git borra el “Staging Area”, sin tocar el “Working Area”.

**git reset --hard código SHA** borra absolutamente todo lo que hay en el commit

**HEAD** es el commit donde nos encontramos actualmente

Una rama es una line de tiempo en nuestro proyecto y nos sirve para repara errores, hacer cambios de interfaz, lógica, etc.

**Git branch -D**  borra una rama

Fusión es la unión de varios commit de distintas ramas. Para hacerlo tenemos que posicionarnos en la rama que va absorber, ejemplo tengo unos cambios en la rama **Test** y los quiero llevar a la rama **master,** como quiero llevarlos a la rama **master** me posiciono en ella **git checkout master**

Y absorbo la rama que tiene los cambios que necesito, en este caso seria la rama test, esto lo hago con el comando **git merge test**

Cuando hacemos una fusión nos pueden aparecer dos mensajes **Fast-Forward** (simple y automático) o **Manual Merge** (Largo y manual)

**Fast-Forward,** hace la fusión, no pregunta nada, esto pasa normalmente cuándo se trabaja con archivos diferentes o líneas de código distintas.

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

Tenemos dos tipos de repositorio, repositorios remotos los cuales están en github y repositorios locales los cuales están en nuestro equipo

**git remote add origin** me permite conectar un repositorio local con uno remoto, decirle a git que ambos repositorios son los mismos

**Git remote -v** me permite ver el repositorio al cual estoy conectado

**Git remote remove origin** remueve el origin que tengo asignado

**git push -u origin master** sube lo que tenemos en el repositorio local al remoto. Si cambio **master** por el nombre de otra rama me permite subirla.

**Issues** son una forma para continuar, mejorar o solucionar un error en nuestros repositorios (proyectos).

**Milestones** Son grupos de issues que aplican para un proyecto, características o periodo de tiempo

**Labels** son una manera de organizar diferentes tipos de problemas

**Git commit --amend -m “mensaje”** me edita el mensaje de mi ultimo commit

**git push -origin master -f** me forza a que se suban los cambios realizados en la versión local a la versión remota.

**Tags** (etiquetas) son simples puntos específicos en la historia de nuestro proyecto y se usan para marcar alguna versión del mismo.

**tags anotadas,**  son almacenadas como objetos completos dentro de la base de Git y contienen más información. **Git tag -a v1.0 -m “mensaje” .**  la versión que coloquemos “v1.0” debería corresponder con el numero de nuestro ultimo commit.

**git tag -a v0.7 -m "version 0.7 de nuestro proyecto"** crea un tag del ultimo commit

**git tag -a v0.7 -m "version 0.7 de nuestro proyecto" código SHA** crea un tag de un commit en especifico

**git push origin v0.7** sube el tab al servidor don v0.7 es la versión del tag que creamos

**git push origin --tags** sube todos los tags que he creado.

**Tags ligeras**, son otra forma de crear tags, más simples y con poca información. **Git tag v1.0**

**Workflows (Flujos de trabajo)**

proyectos propios

proyectos en equipo

proyectos con terceros

Cuabdo nosotros conectamos el repositorio local con el repositorio remoto, hay una rama oculta y prácticamente es un espejo entre dos (idéntico a master).

**Git branch -a** para ver las ramas ocultas

Repositorio remoto master

Origin/master -> rama oculta

Repositorio local master

**PARA PROYECTOS EN EQUIPO**

**Git fetch origin** con este comando obtengo el ultimo commit que se encuentra en el repositorio remoto a origin/master

**Git merge origin/master** con este comando fusionamos el commit del repositorio remoto con nuestro commit local para generar un nuevo commit

**Git push origin master** subo mi commit local al repositorio remoto

Cuando dos o mas usuarios, modifican el mismo fichero, va a tener prioridad el que suba primero el cambio al repositorio remoto. Si intento subir un cambio de un archivo que ya otro usuario modifico Git no me dejara, porque ve que hay una versión más reciente en el repositorio remoto y me exige que primero descargue la versión más actual del repositorio remoto para modificarla y así poder subir los cambios, si el cambio es en la m misma línea que yo modifique, haríamos un **Manual Merge,** cuando hago maual merge Git me identifica el ultimo commit donde estábamos Head y también me identifica el ultimo commit del repositorio remoto.

**PARA PROYECTOS CON TERCEROS**

Cuando no somos dueños ni colaboradores del proyecto remoto, pero estamso interesados en participar, el **fork** nos hace un clon del repositorio principal y nos crea un nuevo repositorio.

Git clone clona un repositorio y **fork** toma un repositorio de terceros y lo pasa a nuestros repositorios.

Hacemos esto después de hacer el fork

**git clone repositorio**

**git remote -v**

**git fetch upstream** baja los últimos cambios del repositorio original, si tenemos los últimos cambios no hace falta ejecutar este comando

**git fetch origin** baja los últimos cambios del repositorio clonado

si se edita el código y desea subirse se debe hacer lo siguiente

**git add -A**

**git commit -m “mensaje”**

**Git push origin master**

Dentro de github debe seleccionar el proyecto y darle clic donde dice **new pull request** luego **create new pull request,** agregamos un titulo, podemos también al igual que los issues agregar un contenido para iniciar una conversación. Para finalizar presinamos el botón **create new request.**

Dentro de la organización verán los cambios en la sección Pull Requests

Para aceptar y combinar los cambios debe hacer clic en **Merge pull request** luego en **commit merge** y luego confirmamos

**GITHUP PAGES**

Nos permitegenerar un sitio web a partir de nuestra organización o proyecto, muy útil para portafolios, blogs y todo tipo de páginas del lado front-end (totalmente gratis).

Para crear un page tenemos que crear un nuevo repositorio

nombreUsuario.github.io

En mi casi seria henrycr82.github.io

Después de esto podemos clonar el repositorio y crear nuestro sitio web.

**SITIOS WEB PARA NUESTROS PROYECTOS**

Creamos un nuevo repositorio

Lo clonamos con **git clone**

Creamos una n ueva rama con **git branch gh-pages** todo lo que este en la rama pages es lo que estará en nuestro sitio web

Luego subimos la rama con **git push origin gh-pages**

Luego chequeamos que la rama este en el servidor remoto en la sección **branch**

Para comprobar que subimos el sitio web solo tipiamos la siguiente dirección en el navegador

**Nombredelusuario.github.io/nombredelrepositorio en mi caso quedaría de la siguiente forma**

**henrycr82.github.io/proyecto** donde proyecto es el nombre del repositorio.

**DEPLOYMENT**

Ssh nos sirve para conectarnos al servidor sin pedir la contraseña al servidor a cada rato

**Ssh-keygen** para generar una llave privada, solo tengo que escribir ese comando y presionar enter unas cuantas veces, la llave primaria se creao en c/Users/Henry Castro, con ls –rf veo los archivos ocultos.

La llave privada es id\_rsa y la publica id\_rsa.pub

Para ver lo que contiene la llave publica escribimos el siguiente comando **cat id\_rsa.pub**

**GIT HOOKS**

Mecanismos para activar scripts, cuando suceden ciertas acciones

Buscar los hooks disponibles

Vamos a trabajar con

**post-commit**

**post-checkout**

**post-merger**

**post-rewrite** lo usaremos cuando editemos un mensaje de un commit por ejemplo

en el directorio donde tenemos nuestro proyecto listamos los archivos o carpetas ocultos con

**ls -rf**

entramos a la carpeta **.git**

luego a la carpeta **hooks** y aquí creamos el hooks post-commit con el comando **touch post-commit**

luego abro mi hooks y lo edito

**#!/bin/sh**

**git push -u origin master**

con este hooks cada vez que haga un commit automáticamente se realizara un push

luego vemos el contenido del hooks post-commit con el siguiente comando **cat post-commit**

luego le damos los permisos pertinentes al hooks con el siguiente comando **chmod +x post-commit**

luego volvemos a la carpeta raíz, en este caso es **/d/xampp/htdocs/gitFacilito**