MANUAL BASICO DE GIT (SVC)

Estando dentro de la teminal:

$git init

Para iniciar un nuevo repositorio sobre nuestro proyecto

Empezando dando los datos del usuario general:

GIT utilizara estos datos para cualquier confirmación (commits) y demás consulta que se hagan. Generalmente solo se hace una vez, al instalar por primera vez GIT

$git config --global user.name “Gustavo Gonzales”

$git config –global uer.email “gustavo.gonda@gmail.com”

$git config –global user.telephone “994919737”

Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando sin la opción --global cuando estés en ese proyecto.

Para listar todos los atributos asignados y los que están por defecto en GIT:

$ git config --list



$ git init : Inicio y creo un nuevo repositiorio:

$git log –pretty=”format:%ae -- %cr -- %h -- %s”

Imprime algo como lo siguiente



Los alias: son las abreviaciones que se hacen para optimizar el uso de los comandos

EJ.

Esta línea arroja la lista de commits con datos como el mail del autor del comit, tiempo desde que se ejecuto el commit , entre otros. Pero resulta muy liosos estar escribiendo tal piedra por su gran tamaño, asi que utilizo un alias.

$ git log --pretty=”format:%C(green)%ae %Creset-- %cr -- %C(yellow)%h %Creset -- %s”

A continuación se ve el uso del alias:

$ git config –global alias.commitlist ‘log --pretty=”format:%C(green)%ae %Creset-- %cr -- %C(yellow)%h %Creset -- %s’

Donde *commitlist* es el nuevo nombre que le doy al comando anterior, es importante el uso de ‘’ (comillas simples)



Donde :

%ae : obtiene el correo del que hizo el commit.

%cr : tiempo relativo, el que paso desde que se hizo el coomit.

%h : la abreviación(7 primeros caracteres de la izquierda) del hash del commit.

%s : el subject con el que se denomino al commit.

A continuación una lista mas deteallada:

%H: commit hash

%h: commit hash abreviado

%T: árbol del hash

%t: árbol del hash abreviado

%P: hashes padres

%p: hashes padres abreviados

%an: nombre del autor

%aN: name del utor (respecting .mailmap, see git-shortlog(1) or git-blame(1))

%ae: email del autor

%aE: author email (respecting .mailmap, see git-shortlog(1) or git-blame(1))

%ad: fecha de autor (formatos de respects --date= option)

%aD: fecha del autor, estilo RFC2822

%ar: fecha del autor, relativo

%at: fecha del autor, timestamp UNIX

%ai: fecha del autor, formato ISO 8601

%cn: nombre del commiter

%cN: nombre del committer (respecting .mailmap, see git-shortlog(1) or git-blame(1))

%ce: correo del committer

%cE: correo del committer (respecting .mailmap, see git-shortlog(1) or git-blame(1))

%cd: fecha del committer

%cD: fecha del committer, estilo RFC2822

%cr: fecha del committer, relativo

%ct: fecha del committer, timestamp UNIX

%ci: fecha del committer, formatoISO 8601

%d: ref names, como la --opción de decorado de git-log(1)

%e: encoding

%s: subject

%f: subject linesanitizada, apropiado para filename

%b: body

%B: raw body (unwrapped subject and body)

%N: notas del commit

%gD: reflog selector, e.g., refs/stash@{1}

%gd: reflog selector recortado, e.g., stash@{1}

%gs: reflog subject

%Cred: cambia color a rojo

%Cgreen: cambia color a verde

%Cblue: cambia color a azúl

%Creset: resetea color

%C(…): especificación del color, como se describió en color.branch.\* config option

%m: izquierda, derecha o boundary mark

%n: nueva línea

%%: un raw %

%x00: pinta un bit de un código hexadecimal

%w([[,[,]]]): cambia el line wrapping, como en la opción -w de git-shortlog(1).

**Ejemplo interesante: formateando los resultados**

$ git log --pretty=”format:%C(green)%ae %Creset-- %cr -- %C(yellow)%h %Creset -- %s”



Explicacion:

%C(green) : lo que este a continuación de este tag se imprimirá en verde

&Creset : lo que este a continuación se imprimirá con el color predeterminado(el blanco, generalmente)

$C(yellow) : lo que este a continuación se pintara de amarillo

**Formato rápido:** Útil para ver los subject y el id hash en la brevedad.

$ git log –format=oneline



Lo anterior también se hubiera obtenido asi:

$ git log --pretty=”format:%C(yellow)%H Creset%s”

$ git log –format=”%C(yellow)%H Creset%s”

Las tres formas arrojan el mismo resultado.

**Ver los archivos modificados por un commit.**

$ git config –global alias.showfiles ‘show --pretty=”format:” --name-only’

Ejecutando el alias

$ git showfiles id\_hash

Donde id hash es el id del hash del que se quiere conocer a los archivos afectados en dicho commit

$git showfiles 5e6d845



En la imagen anterior primero ubique el id del hash y luego ejecuto el alias showfiles

Para conocer a los archivos que fueron tratados en ese commit.

**Cambiando un commit en particular(el subject, mensajes y comentarios)**

Problema: por error puse un comentario de un commit con errores ortográficos que hacen que no se entienda el sentido de su mensaje. Entonces quiero cambiarle el subject de ese commit en particular.

Paso 0: Dejar todo comiteado. Nada puede quedarse en el stage o unstage.

Paso 1: Identifico al commit en particular, necesitamos su id hash(el resumido o el largo).



\*NOTA: commitlist es un alias que defini anteriormente, y se especifica en este manual.

Supongamos que es la confimarcion con id\_hash *2ecfadb* el commit del que quiero cambiar el subject(el mensaje con el que se guardo). Debemos entonces usar el id\_hash del commit inmediato anterior, osea el id : *f378579*

Paso 2: Usamos el sgte. Comando

$ git rebase -i f378579

Y debe aparecer un editor tipo vim como sigue

En ese editor aparecen los tres commits superiores a ese commit, mejor dicho los hermanos el orden esta en inverso (del más antiguo al más reciente), identificamos al que estamos buscando (*2ecfadb*). 

Notar también que más abajo aparece comentado varias líneas (# - símbolo para comentar-), que nos explican el uso de los comandos que se pueden usar en vez de *pick,* nos interesa *reword* que sirve para modificar el subject del commit que queremos trabajar.

El comando *pick* indica que no se haga algo, lo borramos y escribimos en su lugar *reword,* cuidando *de no* modificar lo demás.

\*NOTA: Para empezar a editar debemos teclear [INSERT] y notaremos que el cursor cambiara de apariencia(de cuadricular 􇼛a un subguion \_)



Seguidamente tecleamos [ESC] y aparecerá abajo del terminal el focus para escribir *:wq (dos puntos incluidos)* esto significa en el editor vim (de linux) guardar los cambios y cerrar el editor.

Quedando como sigue a continuación:



Y presionamos [ENTER] o [INTRO] para ejecutar el *:wq.*

A continuación aparecerá por breves segundos el terminal y luego nos mandara automáticamente al editor, pero solo para modificar el subject del commit con id hash: *2ecfadb*. Con una pantalla como esta:



Para modificar en el editor vim ya sabemos que tenemos que usar el [INSERT] para que cambie el cursor a modo insert con \_ (un subguion). Luego de editar el mensaje y/o los comentarios tecleamos [ESC] y en el terminal de abajo escribimos el comando: wq (guardar cambios y cerrar editor). Asi me queda el editor antes de ejecutar el *:wq*



Y así luego de ejecutar *:wq .*Notar *el mensaje Successfully rebased and updated …*



Asi finalmente si vemos la lsita de commits hechos hasta ahora…



Notar que fue cambiado el subject del commit que buscábamos pero notar que el id hash de susodicho commit y los que le siguieron en la historia de las confirmaciones fueron cambiadas por otros, es como si se hubieran sido hechas en momento actual nuevas confirmaciones, por tal notar que el tiempo relativo de cada uno también fue afectado.

\*NOTA: Notamos que sale un aviso de END y para volver a la normalidad debemos teclear [q]. Y ya podremos de nuevo escribir comandos…$.

\*NOTA: En el editor VIM de Linux, que estamos trabajando en este GIT en Windows los comandos básicos de archivos son:

:q! Salir de Vim sin guardar los cambios.

Z! Salir de Vim sin guardar los cambios.

:wq Guardar los cambios y terminar.

ZZ Guardar los cambios y terminar.

\*NOTA: Estos son otros comandos independientes

***mkdir*** nombre\_directorio : para crear un directorio

***ls***: para listar el contenido del diretorio actual

***ls*** > nombre\_archivo: hace que el resultado del ls sea escrito en el archivo especificado, sobrescribiendo su contenido antiguo(tened cuidado).

***touch*** nombre\_archivo: crea un nuevo archivo solamente.

***vim*** nombre\_archivo: crea un nuevo archivo solamente y abre el VIM para empezar a editarlo.

***cat*** nombre\_archivo: muestra el contenido de un archivo.

**exit**:para salir del terminal.

**echo** “mensaje”> nombre\_archivo:escribe la palabra mensaje en el archivo especificado.

**Sacar una copia de un archivo en un commit específico**

$ git show a30ab2ca64d81876c939e16e9dac57c8db6fb103:ruta/al/archivo > ruta/al/archivo.bak

Especificamos el id\_hash del commit y la ruta al archivo, agregamos > yy agregamos la ruta a donde queremos que guarde la copia de ese archivo agregando la extensión .bak y listo.

\*NOTA: el archivo.bak tendrá el contenido del archivo que se guardo en el con el commit indicado, y además tendrá anexado el mensaje del commit con que se guardo, (#el subject-por ejemplo-).

\*NOTA: Tal vez es más fácil solo indicar los 6primeros caracteres del id\_hash.

**Volver a un commit especifico**

**$ git checkout id\_hash :** lleva el directorio de trabajo a como estaba en ese commit, regresar en el tiempo.

**\*NOTA:** Si hacemos cambios en ese punto, debemos confirmarlos y luego pasarnos a la rama estable, desde ayi crear una nueva rama con el nombre cualquiera mas el id\_hash de ese commit.

**Como quitar del repositorio, pero no físicamente a una carpeta o archivo:**

Esto será útil cuando nos hayamos olvidado de agregar a este archivo o carpeta en el archivo .gitignore (donde se establecen q que/cuales archivos y/o carpetas no queremos que estén en seguimiento por GIT).

**$ git rm --cached nombre\_archivo/**carpeta: Para quitar del repositorio al archivo o carpeta especificado. Pero no borrara al archivo o carpeta físicamente de nuestro proyecto. Seguidamente agregar al arichivo .gitignore el archivo o carpeta que ya no será seguido por GIT. Si se trata de una carpeta hacerlo así: nombre\_carpeta/ (la barra es importante-indica a todo lo que esté dentro-)

RAMAS

$ git branch:Para ver(listar) las ramas creadas.Aparecera marcada con (\*) aquella rama donde nos encontremos actualmente.

$ git branch nombre\_rama: crea la rama.

$ git checkout nombre\_rama : Nos lleva a la rama especificada. Sirve para dirigirnos y saltar de rama en rama.

$ git branch –d nombre\_rama :borra una rama especificada

$ git merge nombre\_rama: Fusiona la rama especificada con la rama actual donde nos encontramos, usualmente la usaremos estando en la rama master

**Guardar varios archivos modificados al stage de forma rapida**

Esto solo sirve para archivos que ya estan siendo seguidos, estan siendo trackeados. No para aquellos que aun no estan en seguimiento.

$ git add -p : Preguntara por cada archivo que fue modificado donde se tendra que seleccionar entre una serie de opciones, principalmente (y/n) de yes/no respecticamente.

**Comandos de configuracion remota**

Para trabajar y colaborar en cualquier proyecto GIT es necesario saber gestionar repositorios remotos-una version del proyecto alojada en algun punto de la red-

Estos comandos permiten agregar, modificar y eliminar repositorios remotos facilmente.

**Clonar un repositorio remoto:** Es la forma de traer a una ubicacion local un repositorio almacenado en un punto de la red, asi se comenzaria a trabajar sobre un repositorio ya empezado.

**$ git clone git://github.com/schacon/ticgit.git**

**Agregar un acceso de repositorio:** Para poder hacer push y pull a un repositorio en local ya empezado.

$ git remote add nombre\_de\_referencia direccion\_de\_acceso **:** Agrega un nuevo acceso remoto a la lista de accesos remotos con un nombre o alias de referencia.

**$ git remote add paul git://github.com/paulboone/ticgit.git**

**Listar los accesos actuales a nuestro repositorio:**

$ git remote -v **:** Va a listar todos los accesos remotos que se tienen agregados.

**$ git remote -v**

**bakkdoor git://github.com/bakkdoor/grit.git**

**cho45 git://github.com/cho45/grit.git**

**defunkt git://github.com/defunkt/grit.git**

**koke git://github.com/koke/grit.git**

**origin git@github.com:mojombo/grit.git**

**paul git://github.com/paulboone/ticgit.git**

**Renombrar un acceso a un repositio remoto**

$ git remote rename paul paulOlson : Cambio la referencia al repositorio paul -> paulOlson.

**Borrar un acceso a nuestro repositorio:**

$ git remote rm nombre\_del\_remoto : Elimina una acceso remoto de la lista de accesos remotos.

**$ git push paul master :** Estamos subiendo el estado de nuestro proyecto de la rama master de nuestro repositorio a una ubicacion remota que tiene el nombre de referencia llamda: “paul”.