**Nombre: fecha: 1/jul/24**  
 GIT  
 &   
 GITHUB  
Unos asociados mas a nuestro camino.  
  
  
  
Quiero dedicar este informe a las futuras personas que lo lean creame no se rindan, den todo lo que puedan cuiden su mente tomen agua y sigan sus sueños sin mas que aportar sigamos.

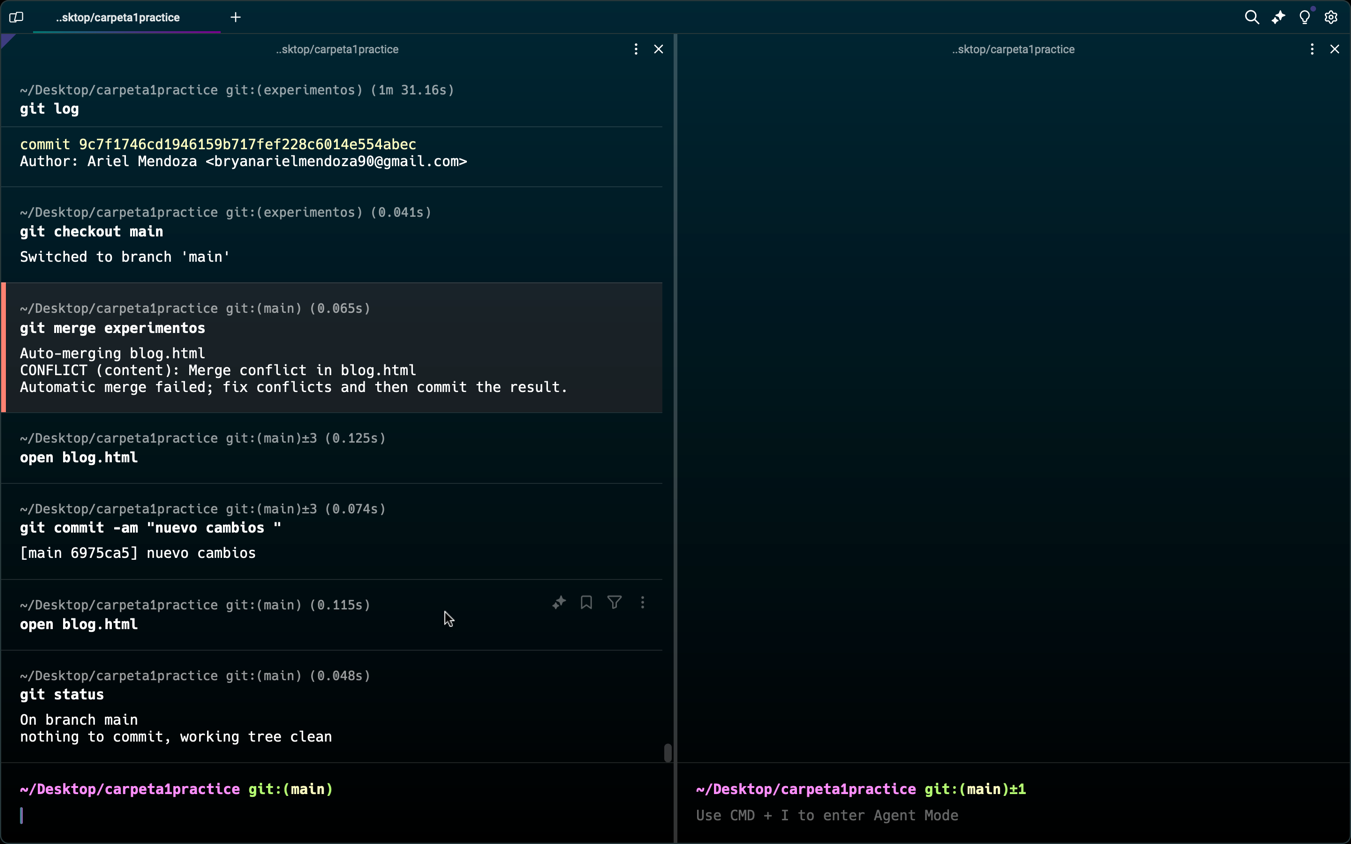
Que es Git?  
  
Este es un sofware elemental en nuestro camino como ingenieros este nos ayudara a subir nuestros repositorios y ver cambios o subirle , en caso de una expecion podremos regresar a la version anterior de forma rapida como ya nos dimos cuenta este es un sistema de gestion de version pero es mucho mas que esto ya que podemos agregar colabrorador por medio de servidor no esta demas a clarar que git no es una nube es un sistema que funciona localmente y podemos activar desde la terminal mas adelante veremos sus comandos principales pero este puede funcionar desde un servidor sin nigun incovenienete.  
Se que algunos estudiante al igual que yo hemos pensado que git y github son lo mismo pero no es asi cada que te adentres a este mundo te daras cuenta que el minino carácter o coma puede alterar todo .  
Github es un facebook o twiter de personas de la industria de la informatica ya que es aquí donde ellos desmuestran sus trabajos y sus logros , ten en cuenta este lugar eh aquí un sector de conocimentos donde se puede aprender mucho por tal no esta demas que vayas y revises su pagina , sube tus primeros proyectos , codigo o lo que hagas este sera el mejor lugar para demostrar quien eres y hacia donde vas.  
  
  
  
Quien invento Git?  
  
   
Su creardor es linus Torvalds el es concido por los kernel de linux para quienes no saben que son estos en pocas palabas son los nucleos de seguridad que tiene un sistema operativo y se cuentan desde la capa 0 hasta la 3 0 4 dependiendo del S.O bueno este surgio por una necesidad ya que ellos usaban un sistema de version anteriormente pero se le reboco la licencia por lo tanto se creo git y es lo que hoy conocemos no profundizo mucho por que creame hay mucho mas que saber , aprender, comprender y poner en practica.  
Como funciona git?  
  
the sistema funciona de forma sincronica paso a paso tanto para poder actualizar un archivo o subir cabe recalcar que este funciona con cualquier sistema de caracteres desde texto plano, binarios, imágenes, archivos comprimidos aunque su eficinecia y confiabilidad es mejor para texto plano pero como funciona git desde nuestro sistema fue la pregunta que me hice este tenemos que instalarlo desde la terminal con el siguiente comando sudo apt install git verificamos su instalacion con git –version este funciona desde nuestra terminal como ya se le a recalcardo antes este nos ayudara a llevar un historia de versiones muy seguro y confiable de nuestros proyectos.  
 Para que puedan entender mejor por que guarda un archivo 10 veces cada que tenga un minino cambio este es el problema que soluciona git ya que solo actualizamos el documento y guarda en su historial el cambio por si algo no sale como esperabas.  
  
  
Caracteristicas   
  
Este cuenta con caracteristicas eficazes para el flujo de trabajo de un desarrollador una que los caracterisan son sus orden y multiple colaboratividad :  
  
  
- **Control de versions**como ya lo hemos venido mencionado es un de sus fuerte el control de versiones   
  
.- **Colaboracion**esto es inpresindible ya que podemos trabajar con mas personas en conjunto cada uno tendra un copia del proyecto donde debera modificar su parte y subirla.  
  
.- **Seguridad** no existe cambio o alteracion en un achivo o proyecto que git no sepa cuando o quien lo hizo por lo que es muy confianble este cuenta con tres estados para localizar archivos:  
  
  
1.- Staged este se refiere al proceso de preparar archivos modificados de trabajo para poder agregarlos con commit (Que es) commit es el proceso por el codiga se agregan o se guardan los cambios realizados.  
  
2.- Modified este es el proceso que nos indica que se han hecho cambios pero estos aun no han sido cambiados   
  
3.- Commit este es el proceso en el cual nos indica que los cambios fueron guardados exitasosamente.  
  
  
  
.- **Flexibilidad**  
  
esto se debe a casi todo es local es raro cuando se necesite archivos o recursos externo pero esto se puede solucionar  
  
 .-**Comando**  
este tiene. Una sintaxis muy simple es asi que cualquiera en poco tiempo puede entenderla   
  
.-**Ramificaciones**  
 se pueden crear ramas para poder trabajar en distintas cosas en al mismo tiempo   
  
  
  
  
  
  
  
  
**Comandos** Los comandos esenciales son fundamentales para tener un conociemiento previos antes de profundiza una vez instalado el git en tu sistema podemos empeza primero verifica con git –version desde la terminal.  
  
  
  
.- GIT INIT  
  
este es el principal con este arrancamos git no esta demas aclarar que debemos estar en el directorio o carpeta que queramos agregar a git una vez hay escriba el comando  
  
  
.- GIT ADD ARCHIVO/  
este comando nos ayudara agregar archivos o directorios a git  
  
.- GIT COMMIT -M ARCHIVO1/  
este nos ayudara a add nuevas versiones de nuestro proyecto  
  
.- GIT ADD..  
  
este nos ayudara todo los archivos modificados donde nos encontremos   
  
.-GIT COMMIT “ARCHIVOS ARCHIVOS1”  
en este subieremos cambios a la ARCHIVOS1  
  
  
.- GIT STATUS  
 este nos dara informacion de nuestros archivos o repositorios.  
­­­­  
.- GIT SHOW  
este nos ayudara haber todas las actualizacions hechas y quien las hizo  
  
.- GIT LOG nombredelarchivoorepositorio  
  
con este veremos una historia en orden cronologico del archivo desde modificaciones y ramificaciones.   
  
.-GIT RM nombredelarchivo  
este comando nos ayudara cachear nuestro documento para poder eliminarlo desde git  
  
.- GIT RM –CACHED NOMBREDELARCHIVO  
  
este nos eliminara el archivo desde git  
  
  
 pero cual es la diferencia entre estos comandos a simple vista es una palabra pero es aquella que hace que tengan funciones distintas el cach se significa que todavia esta en memoria ram y que no han sido guardados sus cambios.  
  
.- Git ff este nos ayudara a comparar versiones de commit para poder saber cual fue el cambio su sintaxis es simple solo busca el numero de commit que te tira la ultima version con la que deseas comparar o con la que deseas.  
  
  
  
Que es Straging?  
  
esta es una capa intermedio ya que se encuentra entre el directorio de trabajo y el repositorio de git he esta zona es conocida porque aquí el archivo espera ordenes para saber que hacer tanto commit o rm algunos dicen que straging esta almacenado en la ram pero no es asi este se encuentra en el mismo directorio git pero en otra carpeta  
Para que pueda compender esto desde el momento que usted crea un init se crean tres capas el directorio git la parte donde nos encontramos ahora.  
  
  
  
  
  
  
  
  
  
  
Ramas  
 normalmente cuando iniciamos un directorio con git estamos en la linea master o main que son las principales al paso que hacemos commit se crean versiones de distintas de estas pero como seria si desea trabajar en conjunto con mas personas o tratar un bug. Estas son muy importante para un flujo de trabajo en equipo o para llevar un registro ya que en las ramas podemos regresar a cierta version tambien.  
  
.-Master   
Como ya lo mencione esta es la rama donde siempre nos encontraremos despues de crear nuestro directorio con git esta es la rama principla donde los cambios y las versiones son actualizadas.  
  
  
.-Developer experimentos  
esta es la rama segundaria donde se ramifica el codigo en pequeñas partes para poder hace experimentos con estos y en cuanto este correcto poder subirlo a la rama main.  
  
  
.-Bugfifing  
en esta rama se trabajan todos los bug que llegue a tener para cunado se resultos enviarlos a la rama main o master.  
.- **Como volver a versiones anteriores como crear estas ramas?**  
  
  
Para volver al pasado existen dos tipos de formas el primero es <git reset 97152cbf6e3be65b5e3fe5ee3cbadb9faf31dffb (numero del commit al que deseas volver) --hard> este comando nos regresara este commit pero tenemos que tener mucho cuidado ya que este borrara todo en absolu de lo que hemos hecho.  
  
 Para poder regresar a versiones anteriores es simple   
git checkout. Numero decommit y nombre del archivo  
tengo que aclararles que podemos regresar a versiones tanto actuales como las primeras sin tener algun inconveniente.  
  
 simplemente para regresar a la version mas actual usamos master su es sintaxis:  
  
git checkout master nombredelarchivo  
  
asi tendremos el commit de el archivco.  
  
algo que se olvidaba mi pequeño dev en caso que usted requiera regresar a x version y desea dejar en la rama master (main) como la mas recien simplemente escriba git add ‘.’ , en pocas palabras el comadno checkout es aquel que nos deja retroceder , regresar y pasear entre versiones.  
  
git switch main o master o old-state : este comando no sera fundamental para movernos entre las ramas de nuestro proyecto  
  
  
git checkout main – namearchivo: este traera el archivo a la rama principal o tambien lo podemos usar para regresar a la version mas actual de nuestro proyecto   
  
  
  
como regresamos. Al linea main si sin querer elimamos este cambio  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Git reset**  
este comando es muy poderoso de por si este nos permite deshacer o revertir cambios en tu repositorio este se puede ejecutar de tres maneas.Pero no sin olvidarnos de checkout ya que este tambien nos permite movernos pero este si nos deja regresar a la version de donde hicimos el salto sin eliminar nada.  
listo pero antes de profundizar en los comandos de git reset debemos repasar conceptos basicos ya que necesito que tengan una compresion solida para asi poder comprender y poner en practica a futuro lo aprendido.  
  
 **Que es un nodo? Y que es un nodo de arbol?**  
  
En general en la informatica es un almacen que puede albergar objetos o valores que pueden representados por sus hijos o estos pueden ser llamados por este y asi crear un puente o conexión entre nodo e hijo estos son fundamentales en estructuras de datos como arboles, listas enlazadas y grafos .  
Sus caracteristicas son:  
.- un nodo es un almacen que contiene informacion o datos  
 .- cada nodo puede tener hijo o estar conectados a otros   
.- en un arbol el principal se llama raiz que es el punto de todo, nodo padre este es el termino que se le da a los terminos que tengan hijos, nodo hijos, nodo hermano es para aquellos que tienen el mismo padre, nodo hoja todo aquello que no tiene hijo y nodo rama aquellos que son raiz y tienen al menos un hijo.  
recuerden que los arboles son jerarquicamente.  
como ya lo he mencionado los nodos estan inmersos en las estruturas de datos pero porque bueno. Los nodos nos ofrecen caracteristicas unicas como flexibilidad al agregar , eliminar o extraer datos ya no se fragmentara cuando se hagan algunas de estas cosas , tambien por ser jerarquico y no lineal el poco consumo de ram para datos estaticos este tiene soporte para algortimos muchos complejos y por ultimo el codigo reutilizable y moduralidad.  
 **Que es la moduralidad?**Primero tenemos que comprender que son las buenas practicas estan seran una mas de nuestras amiga en este camino ya que tener un codigo limpio y legible con buenas practicas nos dara puntaje no solo para terceros sino para nosotros mismo por tal procura tener un codigo culcro sin mas que decir.  
la moduralidad se refiere en tener un codigo ordenado en pequeñas parte cada una bien definida y estructura con el fin de poder tener un mantemiento y deruparcion facil , fomenta la colaboracion legibilidad y facilidad de escabilidad siempre ten buenas practicas ya asi consideradas a tu yo del futuro o a terceros.(recuerda que mucho de estos codigos pueden ser reutilizados para ahorra tiempo).  
  
querido developer chiquitos les recalco nuevamente ponga mucha atencion a la moduralidad esto es fundamental ya que esta es quien nos permitira hacer cambios a futuro a nuestro codigo o por escarlo a mucho mas por tal recuerden modular sus codigos y siempre llevar practicas muy buenas(reutiliza y dinamisa).

**Que son los punteros?**para que podemos comprender y entender este concepto lo dare en metaforas y ejemplos este es un valor que apunta a otro o contiene a otro en pocas palabras imaginemonos numeros del uno al diez y que dentro de estos se contiene valores pero no en todos solo en algunos digamos que el 5 contiene al numero 10 y si miramos al 10 este contiene el 100 es asi como se ultilizan los punteros y desde ahora cada movimiento o valor que obtenga el 5 se vera reflejado con el numero que contiene ya que este apunta directamente al el numero que almacena   
**Por que git utiliza nodos y punteros?**esta es una muy buena interrogante porque se utilizan nodos para ser mas especificos se utilizan nodos de arboles ya que estos ofrecen un sistema jerarquico y no lineal, menor consumo de ram, flexiblidad soporte para algorimos muy complejos, podemos navegar eliminar agregar y extraer datos sin que la lista o carpeta se fragmente por hacer esto tambien tenemos flexibilidad a soporte de codigo y modularidad en lo que respesta en punteros. Lo vemos cada que actualizamos un commit ya que el archivo anterior ahora apunta al nuevo.  
  
pronto profundizaremos mas ☺ por el momento ten claro estos conceptos. **Git reset y sus variables** Para este punto ya sabemos lo que es commit, stragnig, master estos son los ya conocidos tres arboles de git y sus repetivos punteros que son los encargados de ayudarnos con las versiones que se guardaran para que en algun momento podemas regresar a estas.  
discupelmen futuros lectores por ser muy recovecos con los conceptos pero en redundacia les recuerdo estos una y otra vez   
  
Como ya sabemos git reset es quien nos ayuda a navegar en los archivos pero sin retorno tenemos variantes de este son:  
  
  
.- Git reset –soft.   
borra el historial y los registros de git de commit anteriores pero guardara los cambios en straging para aplicar estas actualizaciones a un nuevo commit  
 sintaxis:  
git rese  
  
.-Git reset –hard head~n  
 este comando nos ayudara a deshacer absolumentamente todo del area de commit y el historial de straging.  
Pero no se elimina de por si todo los cambios que se hagan van a un arhcivo en tu repositorio old-state que traducida al español dice estado anterior desde aquí los podemos recuperar en caso que cometamos un error.  
  
.-Git reset. –mixed head~neste es poderoso nos ayudara a eliminar todo desde los commit y straging no quedara nada. Pero en el area de trabajo se conservara los cambios de los commit eliminados.  
  
.- Git reset head~n  
este comando nos ayudara a sacar el archivo del area de preparacion o straging para regresarlo al area de trabajo eso sin borrarlo o realizar otra acciones este nos permitira regresar a los antiguas head hasta donde tengas permitido sus sintaxis es git reset head~n la (n) es numero hay pondras al que deseas volver. Para poder retorna al head inicia ponga git reset head@{1}.  
  
algo que acotar futuros ingenieros recuerden practicar pero antes de que confundan comandos como git reset –soft este nos ayudara traer el head a la zona de straging para poder hacerles cambios por otro lado el git reset head~1 nos dejara regresa al anterior head sin tocar nada o mover archivos   
Un punto clave al tener en cuenta si usted tiene su git conectado con github al deshacer un commit en su repositorio creara una nueva historia de archivos en github esto puede traer errores tengalo en cuenta.

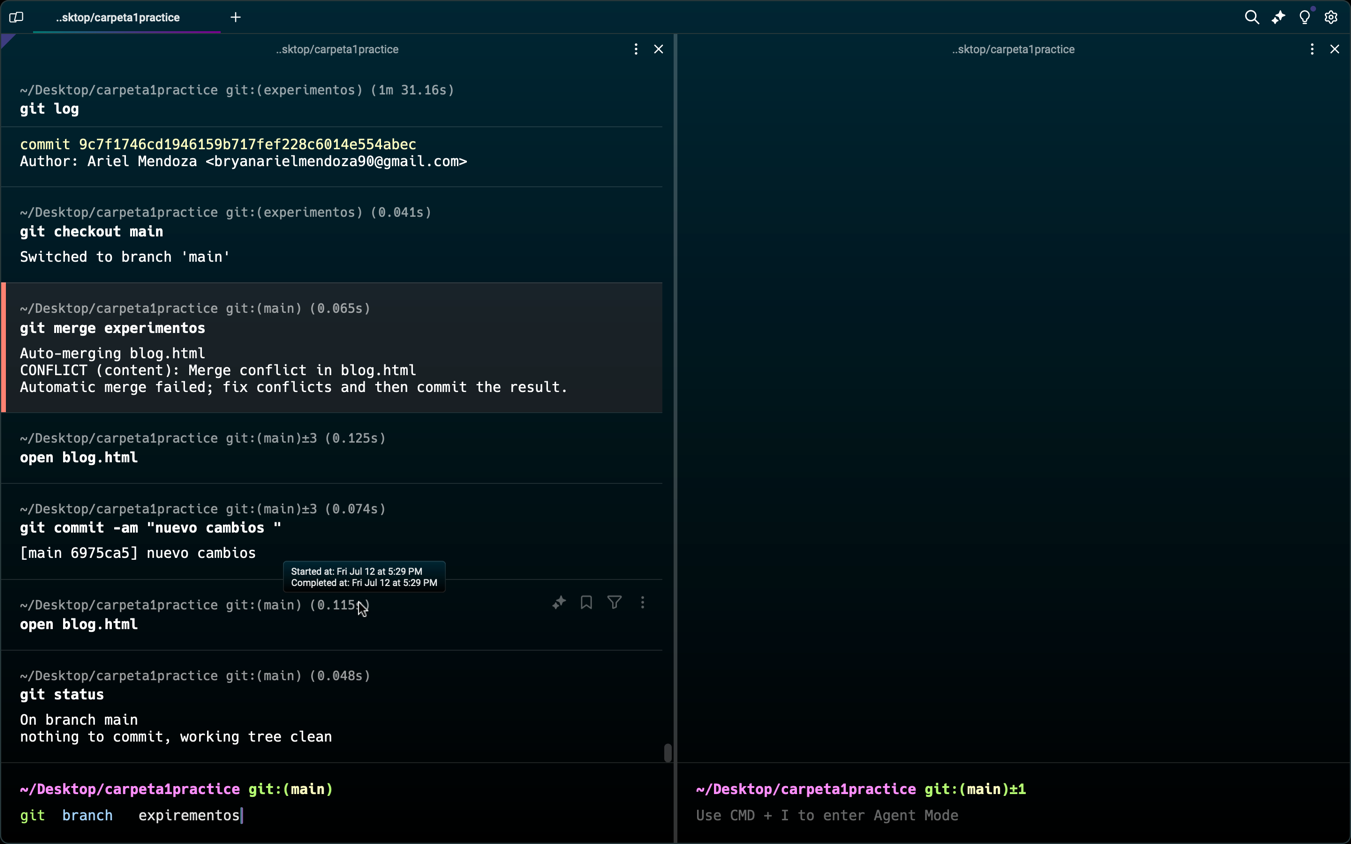
**Git rm**Para quienes tenemos previos conocimiento en la terminal rm es un viejo conocido pero para aquienes no les dejo un cahito de lo aprendido en lo que respesta a la terminal rm se usa para eliminar archivos existen variaciones de este por lo tanto es muy posible que lo veamos pero el flujo de trabajo aquí es muy distinto que en la terminal por lo tanto adentremonos mas en este mundo.  
Como les comentaba el rm es un cargo mas para poder eliminar archivo pero sin eliminar su historia de versiones ya que con este podemos recuperarlo si asi queremos.  
  
 Pero no esta demas recordarle que antes de usar algunas de la variantes que vamos a tener disponibles en unos momentos tenemos que evaluar que es lo que queremos hacer.  
nuestras variantes o flags son:  
  
.- Git rm –cached  
Elimina archivos del repositorio local y area del Staging(o preparacion) pero queda en el disco duro este se deja de trackear en el historial de cambios y queda en modo untracked.  
  
.-Git rm --force   
Se eliminaran los archivos de git. Y del disco duro git guardara todo por lo tanto se podra recuerda archivos eliminados si es necesario.  
  
   
 y al usar git rm se eliminara todo complemente de git.  
  
  
  
Algunos mini developer o de otras ramas confunden estos dos comandos git reset y git rm   
 el primero nos ayudara a extraer un archivo de la zona de staging y el otro eliminara por completo de git aunque recuerda revisar sus flags para poder saber cual es el que mas te conviene en ese momento recuerda nunca ejecutes un comando si antes evaluar lo que vas hacer.  
  
 **Git revert**  
  
este nos ayudara para deshacerse confirmaciones anteriores de forma segura sin modificar la historia del repositorio.Mi pequeño developer hago un punto aquí para que no confundas los comando mucho confunden con git reset y este no sirve para traer archivos de la zona de Straging este cambiara la historia del documento cambio revert no lo hara he aquí sus diferencias minimas pero tendran mucho sentido cuando las necesites usar. **PARA ESTE PUNTO ANTES DE SEGUIR TE RECOMIENDO PRACTICAR PIERDETE VUELVE A ENCONTRARTE. Y JUEGA CON TU TERMINAL Y REGRESA COMO ALGUIEN NUEVO☺ . YA QUE SINO TIENES LOS CONCEPTOS BIEN COMPRENDIDO TENDRAS INCONVENIENTES ESTE ES EL PUNTO DE NO RETORNO MI ESTIMADO VIAJERO.  
  
  
(recalco se frustaran mucho mas de lo que piensan no se rindan y sigan )**

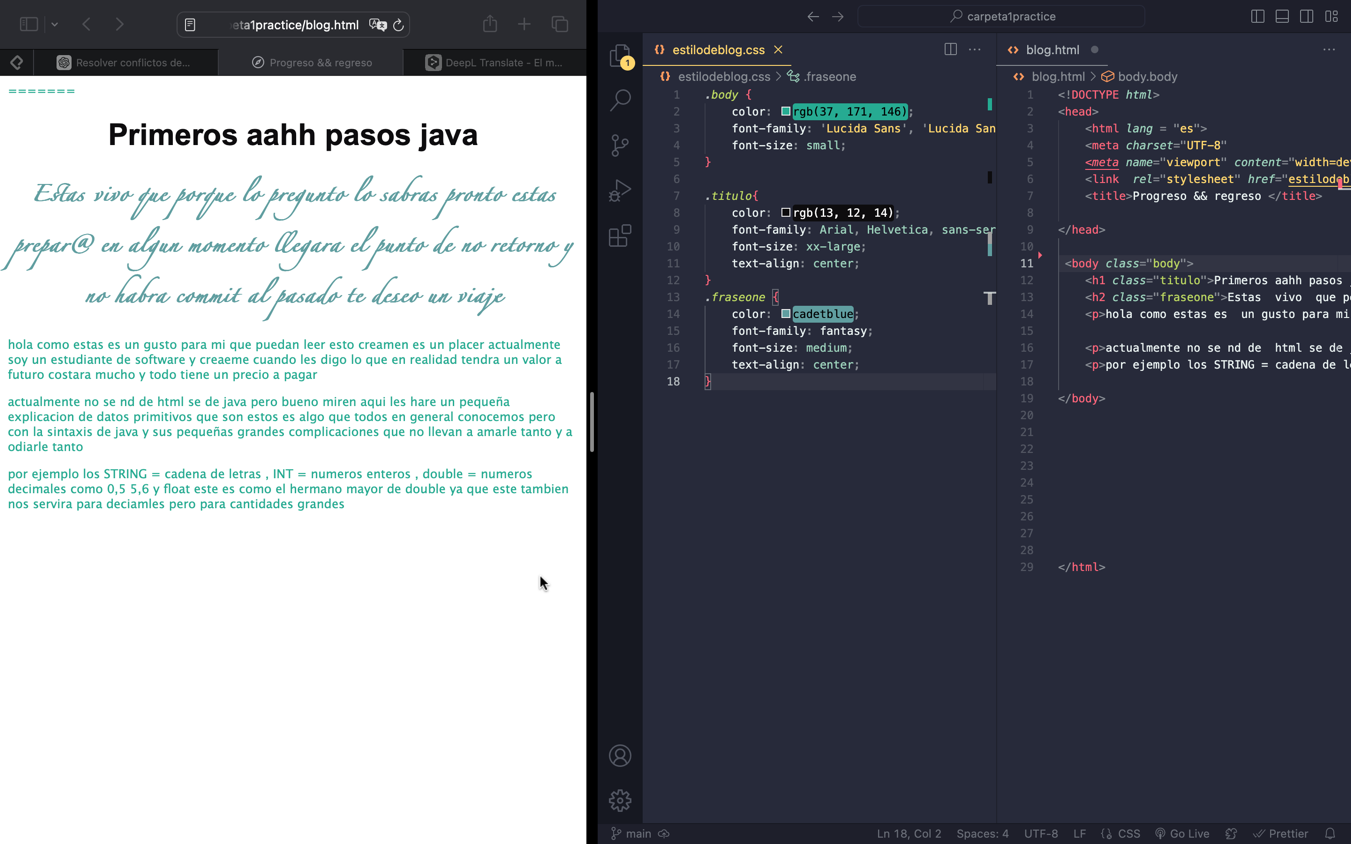
.

Repositorios remotos  
Para este momento como ya lo habiamos mencionado entramos a una nueva etapa como sabemos todo inicia con git init desde la carpeta o lugar donde queramos crear nuestro directorio de git instantaneamente surgen tres carpetas working(area de trabajo) straging , directorio local de git cada paca aquí es fundamental para llevar un flujo de trabajo ordenador como ya conocemos pero cual es el cambio que sufre este mismo proceso cuando trabajamos con un servidor remoto y con mas personas acompañemen averlo esto es impresionante y si gusta te puedes crear un servidor hacer con el lo que quieras.  
  
Al tener en medio un servidor primero cada integrante tendra en su maquina local un copia exacta del repositorio clonadole con git clone url desde tu ordenador podras trabajar y modificar haras todo el proceso hasta add al commit despues de esto haras git push para enviar las actualizaciones y como traigo yo actualizaciones con git fect esto lo trae al repositorio local, esto no me trae directamente con a nuestros archivos por lo tanto tendremos que hacer git merge tenemos un comando que fuciona estos dos se le conoce como git pull  
  
Pronto profundizaramos por el momento necesitamos bases.  
  
  
**introduccion a las ramas** primero comprendamos que las ramas trabajan cada por separados lo que pasa en la rama experimentos no vera visualizado en la rama main al menos que lo subamos aunque sean el mismo archivo y que podemos crear ramas con muchos fines .  
  
 **cual es la funcion de las ramas en git**?  
esta fueron creadas para trabajar en multiples versiones del mismo proyecto sin afectar a la rama principal sus principales ventajas son trabajos colaborativo, experimentos de forma alistada, solucion de errores, control de versiones y un flujo de trabajo de mejor calidad.   
  
  
  
.- git branch nombredeturama: este comando nos ayudara crear nueva ramas para poder hacer uso de sus beneficios

Recuerda que al crear ramas tomas un copia del ultimo comimit hecho por lo tanto ve y experimenta pero aquí surgue una nueva interrogante.  
el comando mergen es uno de los mas escuchado al igual que el push pero bueno por el momento nos centraremos en merge este nos ayudara a traer cambios de otras ramas pero debemos estar en la rama donde queremos traer los cambios no donde estan los cambios mucha precaucion con esto   
  
  
les explicare paso a paso los primero creamos nuestra nueva rama   
  


Primero vemos nuestra rama para ver donde nos encontramos despues de esto utilizamos el comando git branch <nombredenuestrarama>



Las creamos con el nombre que nosotros deseamos y le das enter despues usamos el concepto git swtich nombre de la rama o git checkout nombre de la rama para cambiar comenzamo a modificar nuestro proyecto o lo que usted desee una vez hecho esto regresamos a la rama principal y hacemos el merge es muy posible que te errores y tengas que resolverlos manualmente abre el archivo y elimina lo que no deseas y trackear el archivo y agrega.   
  
 haras hecho tu primer merge mi estimado dejo mi codigo y mi merge   
  
  
  
  
  
  
  
  
  
  
  
 Hacemos nuestro merge pero que es lo que sucede despues de esto. En la rama de donde extraemos estos cambios pues nada absolumante nada esta sigue como si no hubiera pasado nada pero podemos hacer un merge a la inversa para traer lo que esta en otras claro que si solo repetiriamos el mismo proceso que hicimos anteriormente si listo.  
  
el manejo de errores en los merge es simple cunado tengas un error simplmente abre el archivo del proyecto saque lo que necesite y deje lo que deseas aunque esta recomendación no aplica para trabajos en equipo si llegas a tener un problema con un codigo de un compañero y al hacer merge. No se soluciona o sabes que si al hacerlo lo afectaras. Habla con el recuerdo la comunicación es el cambio siempre recuerda comunicarte mi pequeño dev.   
  
  
  
  
  
  
  
  
  
  
  
  
Gracias por llegar hasta aqui esto sera un blog en la internet donde compartire todo lo aprendido durante estos años como un autodidactada ya que esta informacion vale oro tanto para quienes vienen del futuro como del pasado se que suena loco aquello pero sigamos espero que estudies como yo por mero gusto y no por obligacion ☺ sin mas que decir sigamos a un nuevo modulo.  
  
 CREAME EL APRENDIZAJE ES LA BASE DE TODO QUIEN ESTE ENCONTRA DE ESTE ESTA ENCONTRA DEL DESARROLLO TANTO PERSONAL COMO GRUPAL.  
  
  
  
  
  
  
  
GIT HUB  
“La red social del Software”  
  
  
  
  
  
  
Que es Github?  
Ya en el paso cuando comenzamos github cuando comenzamos pero no esta demas profundizas simplemente nuestra respuesto fue una red social donde. Los desarrollares pueden compartir su codigo o desmotrar lo que han hecho o colaborar en esto pero hay mas para hoy conocemos conceptos basicos , como un pequeño dato github usa el sistema de versiones de git   
  
  
  
  
  
Que son LLaves publicas, llaves privadas MATEMATICAS APLICADAS ASINCRONICA  
  
Primero no se asusten si tenemos matematicas son la base de la informatica pero creame una vez que las entienden todo cambia pero centremonos en las llaves primero imaginate que deseas enviar un mensaje cifrado que nadie mas le puedo saber aunque sea intercetado o lo publiques en toda la internet para ellos usamos las llave publica donde cualquier pude agregar inforamcion pero no leerla pero solo tu podras descifrar el mensaje con la llave privada al mismo tiempo con la llave publica les enseñare el proceso matematico y su formula para despues despejar la pregunta Para que nos sirven estas llaves y que tiene que ver con github?  
  
  
  
.- Su formula matematica es:  
  
  
 \* n= p x q   
\* φ(n)=(p−1)×(q−1)  
\*e = 1<e<φ(n) = gcd(e,φ(n))=1

se que lo estan pensando algunos que tiene que ver esas formulas con programacion y peor aun con github primero acompañemen a resolver y ver para que sirve cada una:  
  
  
  
  
\* n = p x q.   
estos dos primero valores tiene que ser numeros primos grandes que son los numeros primos estos son divisibles para uno y para si mismo nomas.  
  
  
n = 73 x 83 = 6059   
  
  
\* (n)=(p−1)×(q−1)

en este proceso de aquí restamos y multiplicamos   
  
  
(n) = (73-1) x (83-1) = 72 x 82 = 5904 (ojo con este resultado nos ayudara mas tarde )  
  
  
\* e = 1 < e. < φ(n)  
  
 e = 1 < 19 < φ(n)  
  
  
5904 / 19 = 310   
  
  
310 \* 19 = 5890.   
  
  
  
\*. 5904 – 5890 = 14   
  
  
gcd (19,14) =1 -> (19, 6059) = 1

Configuracion de las llaves en tu ordenador  
  
  
Como ya sabemos que son las llaves simplemente pasaremos a la su creacion en nuestro dispositivo local tengo que aclarar que en todos los sistemas es diferente por lo tanto en este me enfocare en el S.o de Unix especialmente en Mac os tenemso que primero crear esta abrimos nuestra terminal y escribimos git config -l user.email "tu correo de preferencia el que tenes vinculo con github" despues de tenerlo configurado el email procede con la creacion de nuestra llave con “ssh-keygen -t rsa -b 4096 -C "tu\_correo@example.com"” listo despues ponerla en la ruta que usted desea   
  
->tendremos que configurar y crear un archivo si usted se encuentra en las mac mas nuevas regresamos al apartado home y buscamos ls -al. Buscamos el archivo .ssh y verificamos que tenga una carpeta llamada config si usted cuenta con aquella procesa al siguiente paso sino esto lo dejo a criterio de cada quien hay mucho metodos podemos crear el archivo desde la terminal y desde afuera eso si este no debe de contar con ningun formato lo que contendra sera esto:  
  
  
Host \*

AddKeysTAgent yes

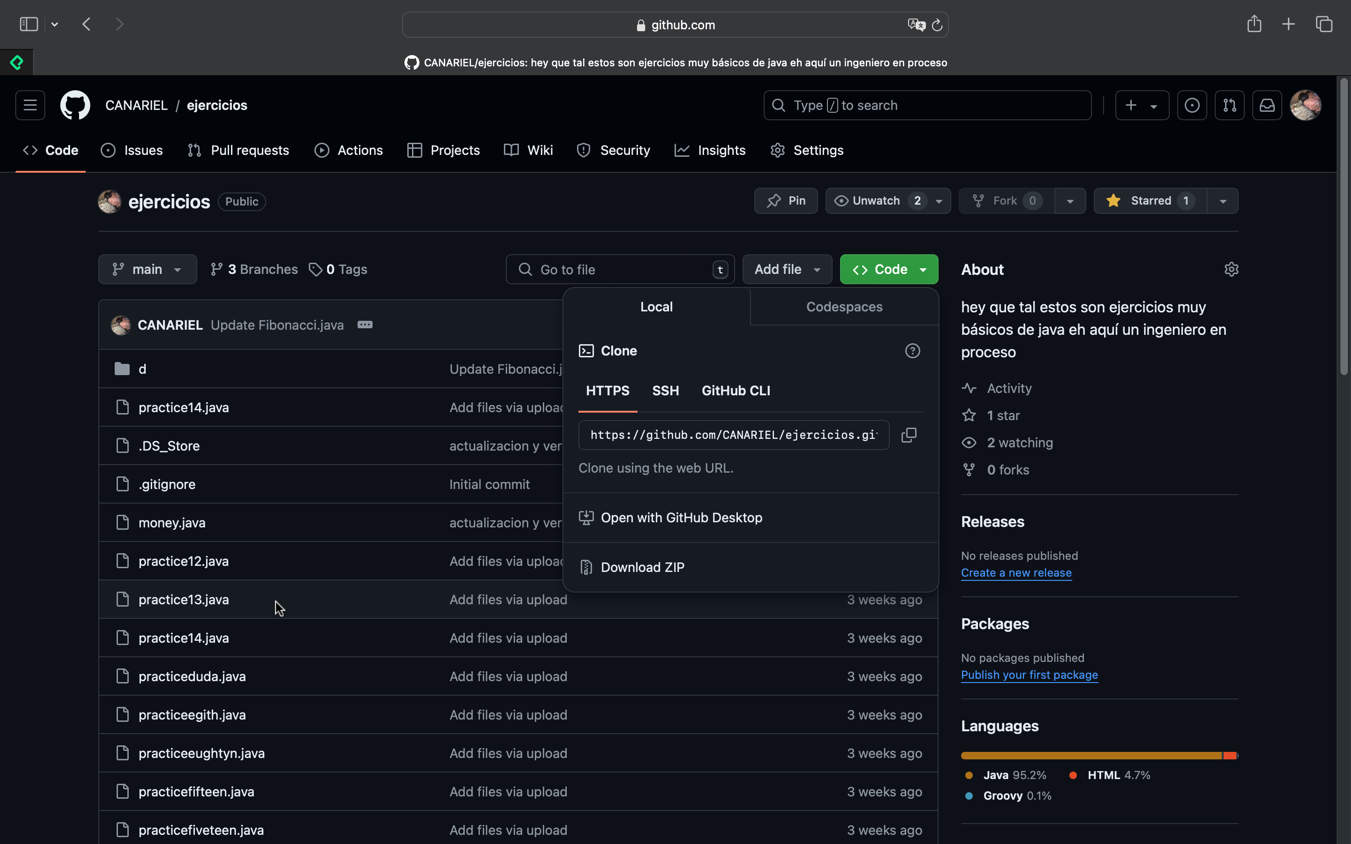
UseKeychain yes

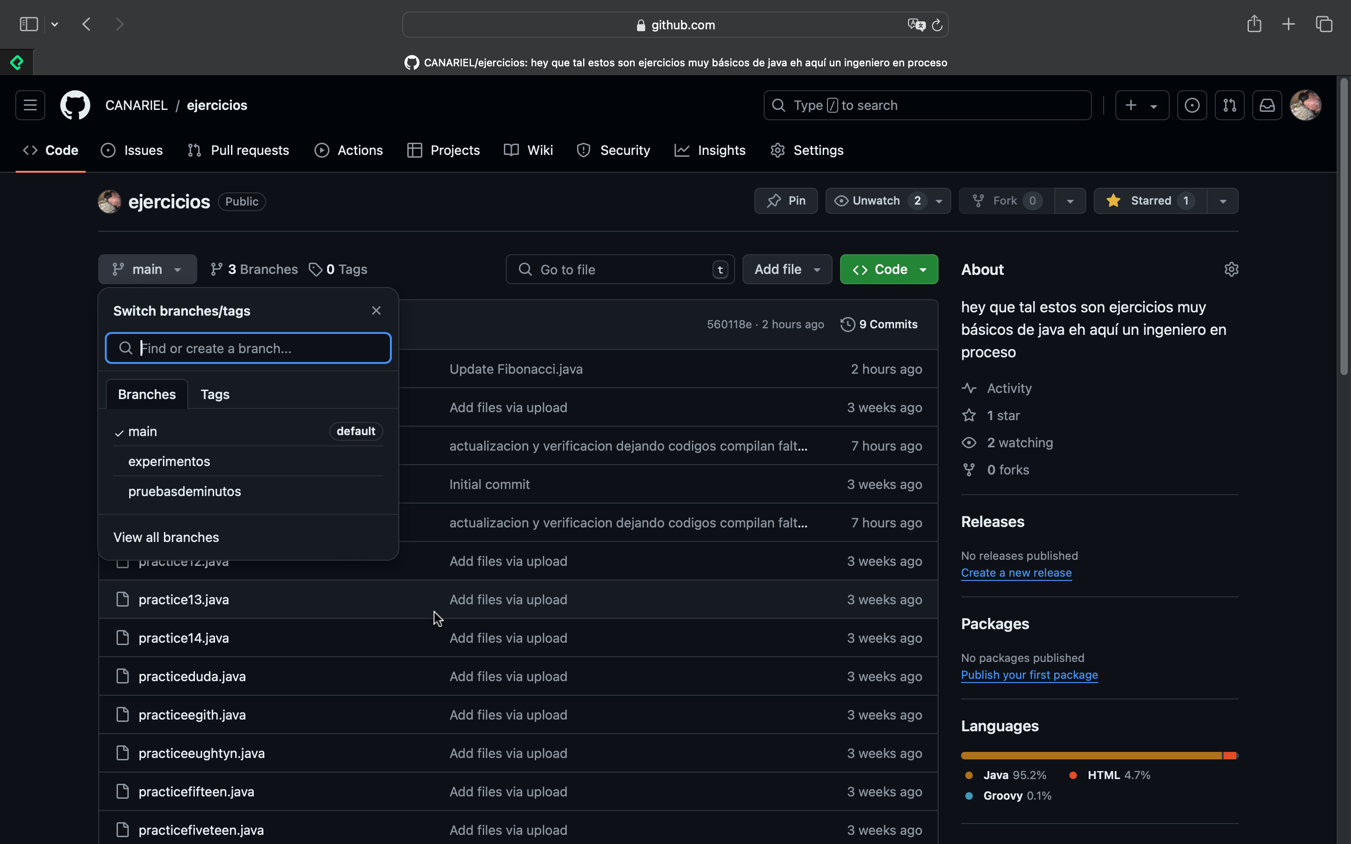
IdentityFile ~/.ssh/id\_rsa

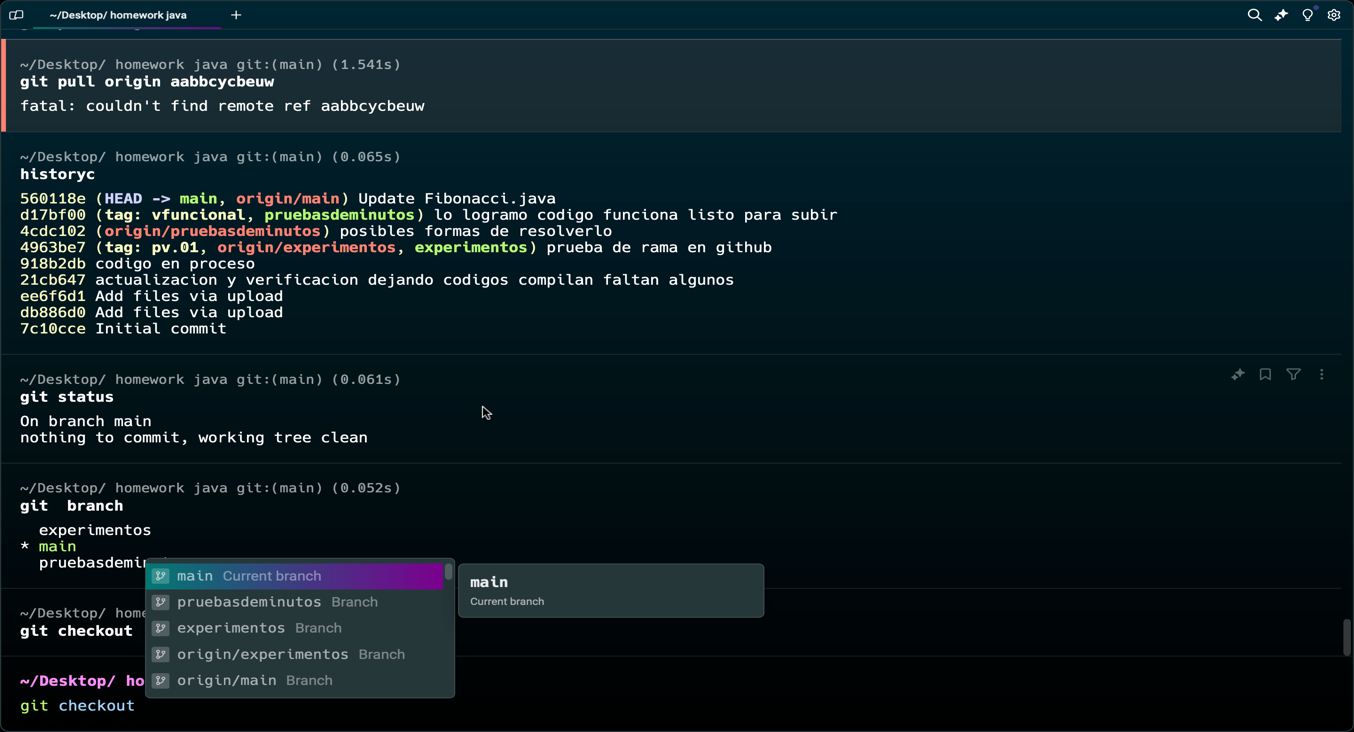
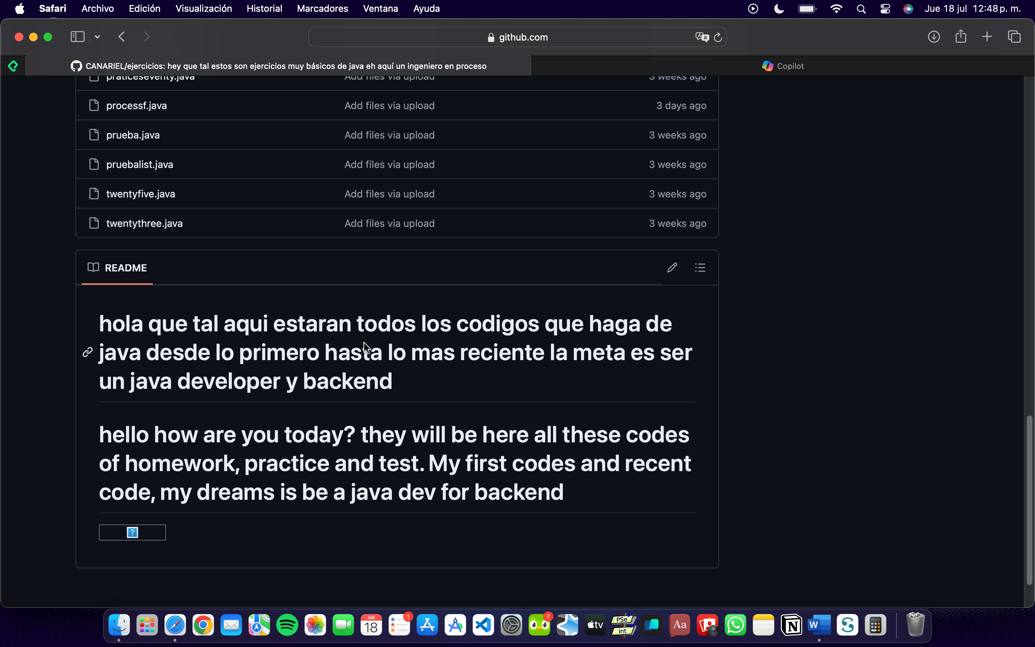
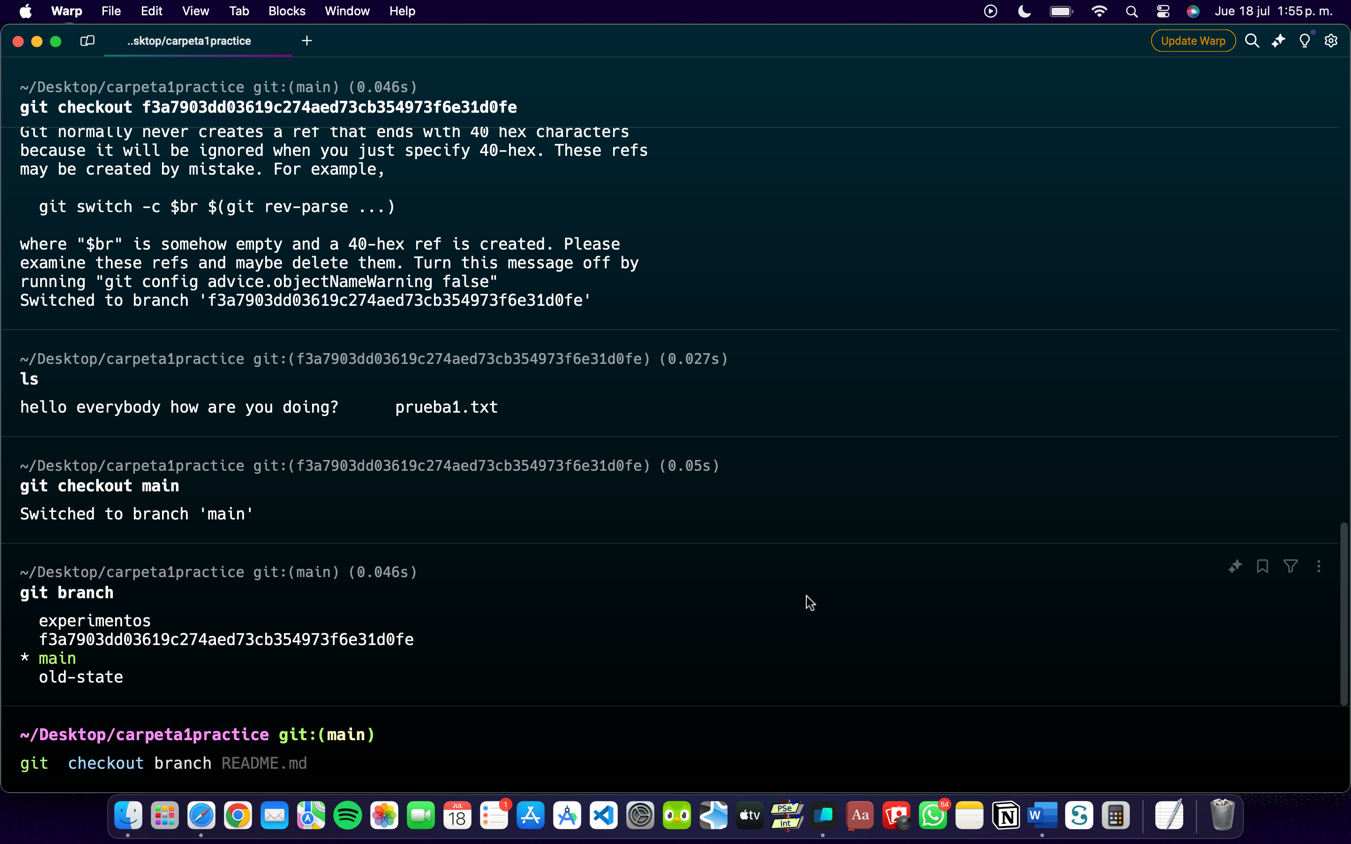
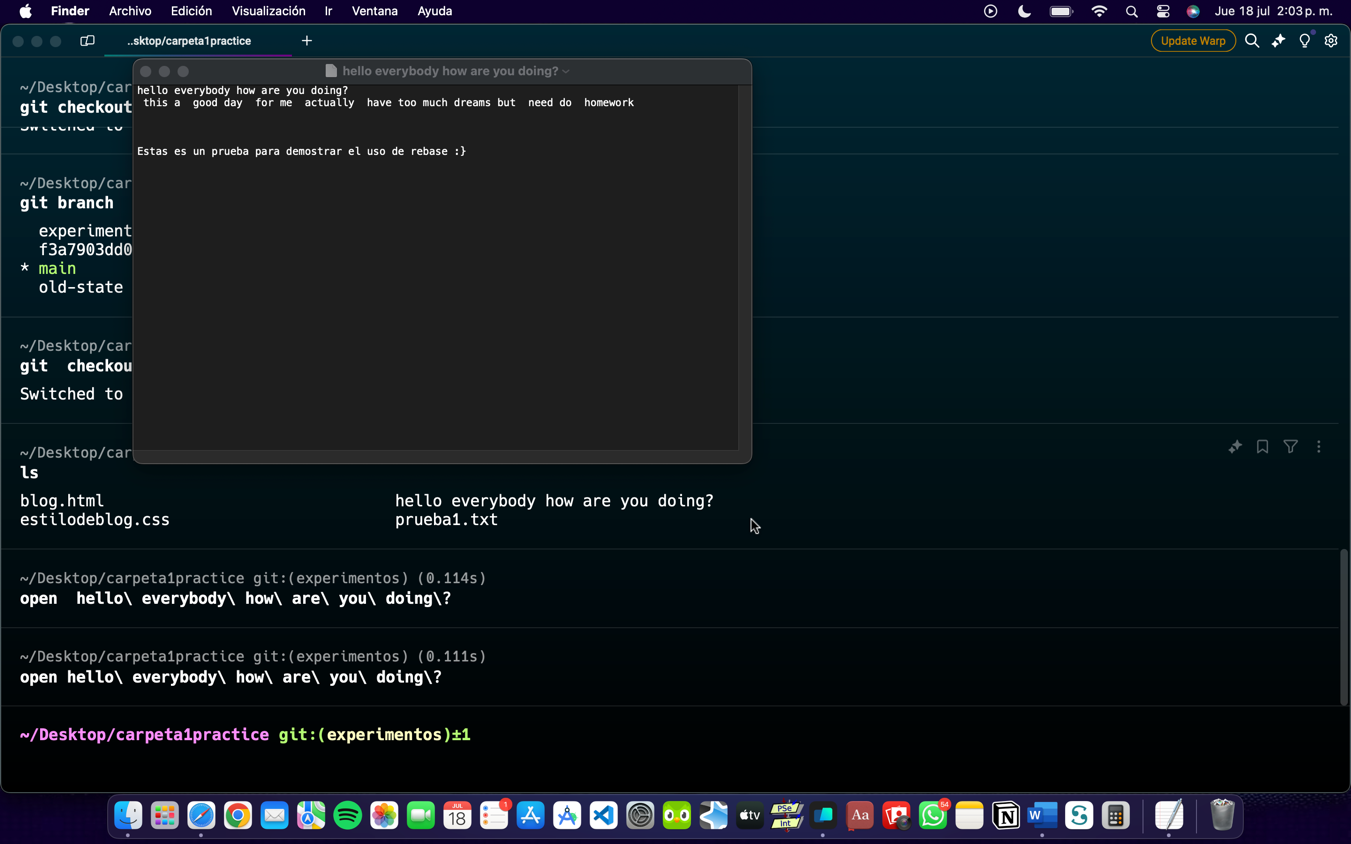
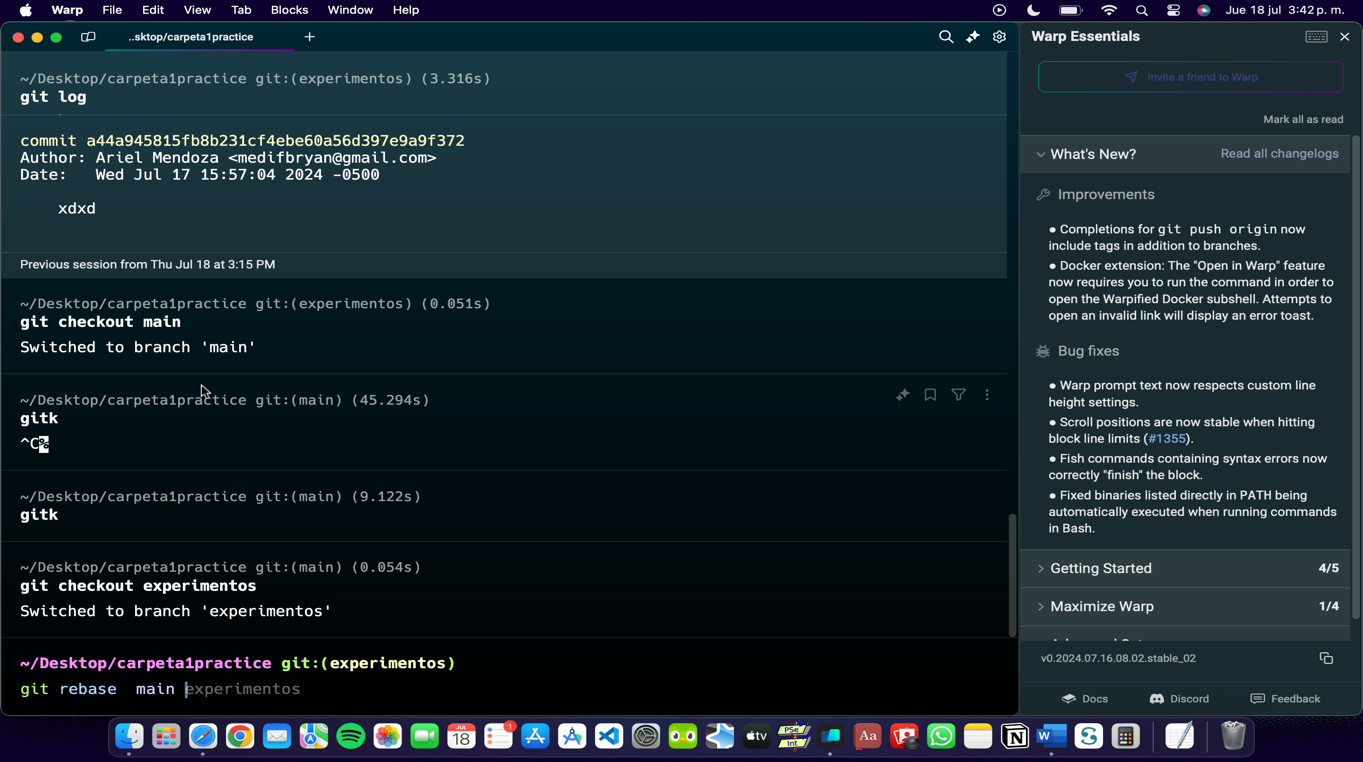
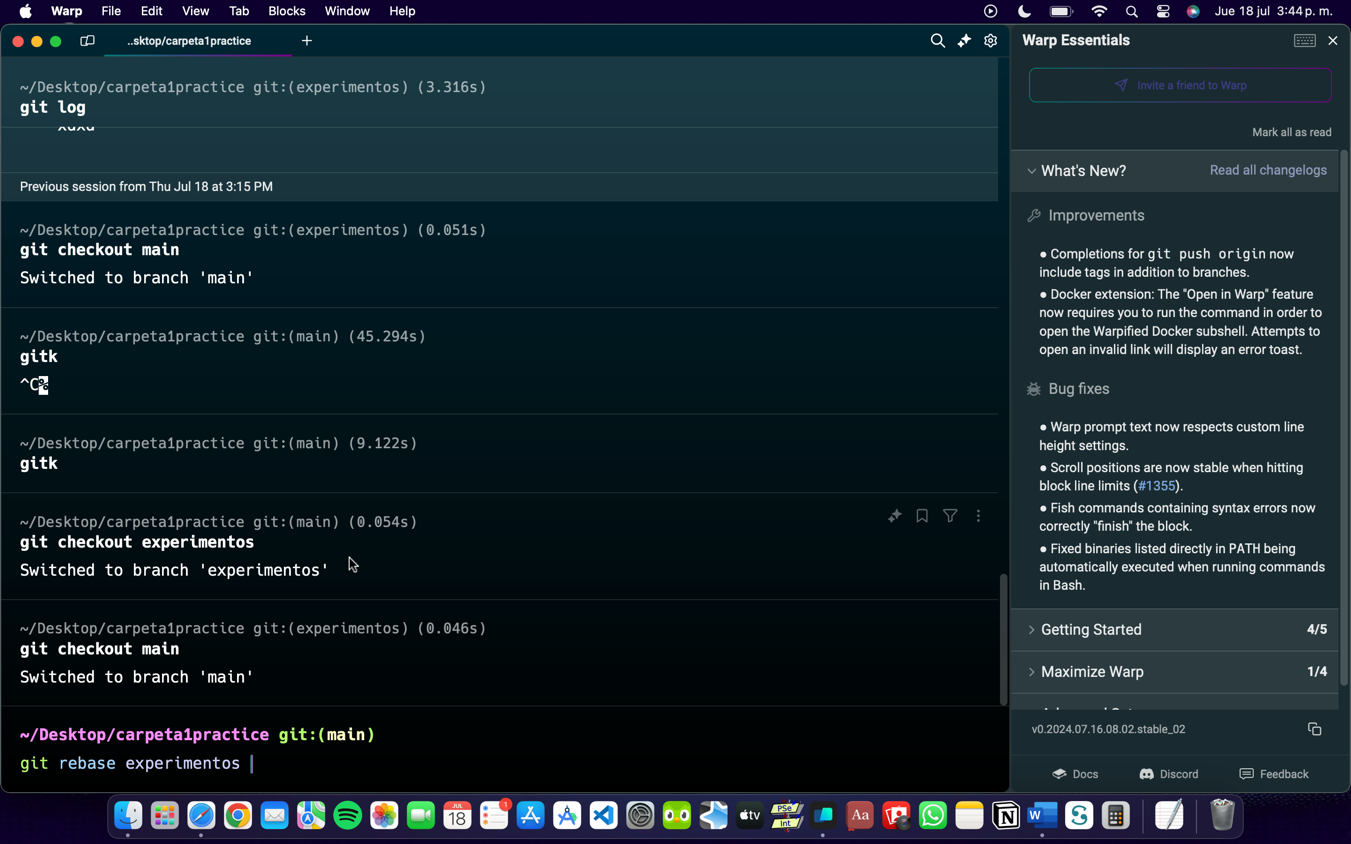
despues de esto tendremos que verficar que el proceso se este ejecutando con el siguiente comando \*ssh -t git@github.com \* despues vincularemos nuestra llave publica con nuestra cuenta de github para poder subir cambios desde nuestro escritorio y hacer nuestro primer push, si todo esta correcto simplementes no dirigimos a nuestro directorio de git en y escribimos git remote origin “linkdelrepsoitoriodegithub” despues de esto no pedira permiso concedacelos y listo, siempre antes de enviar algo debemos llamar a la ultima version que se encuentra en la nube con el siguiente comando git pull origin master y despues el git push origin master recuerda el primer es para traer el segundo para enviar con eso estaras listo.  
Este comando nos ayudara a enviar aboslutamente todo desde tu carpeta de origen git pull origin master --allow-unrelated-histories  
  
.- Git log –all   
es un comando que nos mostrara absolutamente todo los commit   
  
.-Git log --all –graph  
Esto nos mostrara la historia de nuestro proyecto y las ramas   
  
.- Git log –all --graph –oneline --decorate  
este nos mostrara de forma mucho mas detallada la historia de nuestro proyecto

puedes usar alias para poder optimizar tu tiempo de trabajo desde la terminal.

Git tag  
  
  
Estas son referencias que agregamos cuando necesitamos dejar un logro o hipto importante durante el desarrollo de tu proyecto ojo no se confundan los commit son diferente ya que son actualizaciones de nuestro proyecto los git tag son comentarios como “el codigo se encuentra estable ” con estas pueden apuntar directamente a cualquier commit sus sintaxis es simple – git tag -a v0.1 -m “esta es una prueba en mis anotaciones”. – este apuntara nuestro ultimo commit y si quisieramos que apunte a uno especifico seria-- git tag -a v0.1 -m “esta es una prueba en mis anotaciones” f00398ca29e8181c73a3aeb3f3b2973be6de561a—de esta forma el tag se quedara en la version donde le hemos indicado. Estos normalmente son utiles en github para programadas de codigo abierto en git simplemente lo podemos usar como guia o referencia pero como lo enviamos a internet que es donde realmente son de mucha ayuda el comando es – git push origin –tags – y en caso que agregres un tag mal la forma de eliminarlo es git tag -d “namedeltag” como ya esta decir que cada cambio que hagamos primero tendremos que hacer pull y despues push pero como el eliminamos el tag que se nos fue mal desde github con el siguiente comando git push origin: refs/tags/dormido  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Manejo de ramas  
Git Hub  
  
  
Lo fundamental que es aprender a manejar la ramas y usarlas para este puntos ya sabemos o tenemos una idea merge, pull y push no todas nuestras ramas tiene que estar github o todos tiene que estar a la vista el manejo de esta nos dara un plus a la hora de trabajar con este dos programas por tal acompañemen pero no sin antes darle un comando que nos dejara ver todo lo aprendido hasta ahora gitk.  
.-\*\* Si usted uso el comando anterior y le dio error estas lineas son para usted primero si usted esta en window use estas lineas de comandos **sudo apt-get update  
sudo apt-get install gitk.** Si usted se encuentra en mac os instale homebrew o verifique lo tenga instalado” gitk --version” o intalale con **brew install git-gui.**  Y despues llame al comando inicial no esta demas decirlo que lo debes hacer desde la terminal.  
  
  
  
  
Comandos fundamentales   
  
  
git branch creacionista de ramas o nos muestra cuantas ramas tenemos   
  
  
git show-branch historia de las ramas y cuantas tienes actualmente   
  
git show-branch --all   
es lo mismo que el anterior pero con mas datos en el ellos destaca   
  
  
pero que hay de diferente entre dos comandos que parecen ser iguales A SIMPLE VISTA PARECE QUE HACEN LO MISMO pero el git show-branch es aquel que solo nos muestra las ramas local y sus cambios.  
  
  
.-git show-branch --all no deja ver las ramas remotas que tenemos y las locales y sus respectivos cambios.

hago un alto aquí futuro ingeniero no te olvide de disfrutar el proceso creeme y si no esta en tus posibilidades disfrutar de este pues adelanta termina de leer y continua se que no todos estan en la posibilidad pero recuerdo el proceso importa y mucho sal toma sol corre y lucha por lo que deseas sin que decir pequeño dev sigamos.  
  
  
  
  
  
 **Configuracion de ramas para multiples colaboradores**   
  
Primero debemos comprender que cada programador trabaja en una rama diferente para poder ultimo hacer merge a la rama main casi siempre esto lo hace el seo o el proyect manager pero les mostraremos como dar permisos a nuevos usuarios a su repositorio para poder trabajar en conjunto o que simplemente sepas como hacer.   
  
  
  
  
.- Primero debemos crear nuestra cuenta en github buscamos el repositorio con el que trabajar   
  
  
  
  
  
cojes el link abres el terminal y pones – git clone <https://github.com/CANARIEL/ejercicios.git> pero debemos saber que tenemos que asignarle un lugar una carpeta una dirrecion a donde descargase como se dan cuenta no hicimos el famosisimo git init.  
.-Despues comenzamos ubicandonos en la rama designada por como nos unicamos en esta primero verificamos si esta existe tanto lo podemos hacer desde el codigo en la internet como lo veremos a continuacion



O directamente desde la terminal de esta forma y como cambiamos de rama con git checkout nombredeturama o git switch nombredeturama como lo vemos:  
  
  
listo una vez usted ubicado en la rama podra hacer sus cambios pero nos podremos hacer git push origin main pero tengan en cuenta que normalmente el dev no hace aquello para hacer esto tenemso que tener permisos asegurate que el jefe del proyecto te de estos ya que sino no podra hacer ni el git pull origin main para poder trabajar como nos aseguramos de estos debes tener tu correo publico en github para que te puedan encontrar te dejo te tarea eso.  
  
  
  
  
  
Que es un pullrequest? Como colaboramos con terceros en proyecto opensource?  
  
  
Primero no esta demas recordar que cuando se trabaja de forma profesional la rama main esta bloqueada pullrequest es un filtro antes de esta donde los devOps o el seo o quien este encargado revisa nuestro codigo a profundidad para poder aprobarlo o mandar cambios para despues hacer el merge con la rama principal que es aquella que esta concetada con el servidor que es conocido como produccion se que si estas aquí y llegaste hasta aquí tienes una idea de que es produccion.  
  
Como colaboramos con terceros primero tenemos que verificar si es un proyecto de codigo libre o open source despues buscaremos el fork en el repositorio en el que vamos a trabajar. Y este se hara una copia en nuestro perfil tomaremos el codigo y lo clonaremos en nuestro S.o local y comenzaremos hacer cambios despues de esto lo guardamos y agregamos a con push como siempre este proceso esta demas darlo en caso que el archivo original tenga cambios tenemos que crear una nueva donde almacenaremos estos git add nombredelanuevarama. Linkdelrepositoriooriginal   
despues de esto simplemente lo git pull nombredelanuevarama main en este ultimo traemos lo ultimo que tiene la nueva la rama a la rama de nosotros main por ultimo despues de estar al dia con la informacion y hacer nuestro cambios hacemos pullrequest y esperamos a que el dueño del proyecto acepte o deniegue nuestros aportes.  
  
  
  
  
  
  
  
  
  
  
  
  
  
 Archivos .gitignore  
  
Que son este tipo de archivos este son fundamentalmentales cuando no queres dejar informacion expuesta o estamos subiendo imágenes como ya sabemos en github no le va muy con las imágenes ya son binarios y le es dificil mostrar los cambios realizados con esta, para ponerlo mas en contexto estos archivos son de gran ayuda ya que esconde a nuestra informacion y archivos para que git y github no las tenga directamente a vista de todos ya que esto puede ser peligroso. O crear errores por lo tanto estos archivos van desde la raiz de nuestro proyecto.  
  
  
  
 Que es un Readme?   
  
  
este es un archivo que en resumen nos dara una carta de presentacion del proyecto que tenemos o estamos trabajando donde encontraremos el lenguaje usado, las licencias y versiones, ejemplos de uso, ejemplos de ejecucion y resumen de este en su totalidad no confundamos este por lo que conocemos como documentacion del proyecto ya que esta es quien responda a duda como porque, paraque entre otras.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 Que es un Git rebase?  
  
  
Este es un comando que puede llegar a ser un poco confunso pero por que digo aquello puesto que al usuarlo elimina registros y hace como si estos nunca exitirian se que hasta este momento no has comprendido al igual que yo pero acompañame comprenderemos.  
  
  
Te pido mucha atencion para que puedas entender y comprender primero recalco no debemos usar este comando nunca pero nunca en servidores remotos. Por que eliminara las historia y los commits de la rama de donde salieron y puede llegar a causar mucho problema sino se hace bien que solo se podran arreglar con git reset y sus variantes.  
  
  
  
Este comando lo que hace en pocas palabras es el eliminar la rama de donde vieron esos cambios nuevos y pegarlo a la rama main como si la otra nunca ubiera existido rescribiendo la historia y eliminado commit a continuacion ejemplos del uso de esta:  
  
Tenemos varia ramas no movemos a cualquiera de esta:  
  
  
despues creamos cambios y agregamos sus respitivos commit en esta rama:  
  
  
Despues hacemos el primer rebase a la rama main:  
  
  
 nos cambiamos de rama y hacemos el rebase a la rama donde hicimos las modificaciones:  
  
  
  
despues de este proceso verificamos podemos tener errores que tendremos que resolver manualmente despues de esto en caso que tengas sigue las intruciones que te da la consola no me enfoco en una sola solucion porque caso sera muy diferente por lo tanto recalco sigan a su consola.  
  
Algo que me olvida en decirle es que primero debemos realizar el rebase desde la rama donde tenemos nuestros cambios a la main despues en la main traemos a la rama donde tenemos estos cambios asi no garantizo que no vayan a tener incovenientes pero se ahorraran mucho incovenientes simas que decir practicen, de hecho me perdido ya varios archivos con estos comandos.  
  
  
  
  
  
  
  
  
  
  
  
  
  
Que es git stash?

Este es un comando que nos permite guardar temporalmente cambios, este también nos ayudara poder regresar al pasado con cambios que no se han guardado ejemplo

tenemos comandos como:

.- Git stash: este nos ayudara cuanto necesitamos guardar cambios temporalmente como para poder agregarlo a otra rama sin tener que eliminar lo que tenias antes en este mismo lugar que este se que era guardado en stash y para poder regresar a lo que antes teníamos usamos git stash pop

para ponerlo en una rama lo que hacemos es lo siguiente con el comando git stash branch nombredelarama

Que es lo que hace este comando es simple envia el stash a la rama donde le indicamos podemos crear tantos como necesitemos con git stash list vemos todos los que tenemos guardados es posible muy posible modificar un documento por completo antes de crear la rama para ello usaremos el mismo comando pero como crearemos mucho stash le pondremos comentarios nuestro comando quedaría como :

.- Git stash push -m "titulo":  
Este comando nos ayudara para poder llevar un idea de donde va cada stash ya que si hacemos muchas no tendremos ideas cual es cual podremos comentarlo

.- Git stash apply stash@{0}.:  
Este aplicara el stash que el indiquemos en el documento o proyecto donde los encontramos

Diria que git stash es un lujo y de mucha ayuda creme te ayudara mucho por lo tanto continuemos.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 Que es Git clean?  
  
Querdios estundiantes yo que se mas que uno en su directorio o archivos de estudios al momento de desarrollar tiene archivos o imágenes que se les pueden duplicar o simplemente no saben como llego ahí, yo lo se por que me paso el comando git clena es fundamental para estos tipos de ocasiones pero si ustedes van a su terminal y escriben ese asi por asi este no funcionara por lo que este primero debemos probarlo de esta forma – git clean --dry-run ya que en cuestion vamos a eliminar archivos y esta tarea puede ser un poco riesgosa si eliminamos cosas que no debemos por lo tanto se toma esta precaucion listo una vez que vemos que queremos borrar lo hacemos con el siguiente git clean -f pero tengamos estos en cuenta a git no le importa las carpetas sino los archivos si tenemos cierto archivo que fue trackeado duplicado este no sera eliminado por lo tanto tendras que proceder manualmente con el delete pero ustedes recuerdan que el archivo .gitignore donde ponemos terminaciones o comodines de archivos que este ignorar por lo tanto si tu archivo esta en esta lista no podra git hacer su trabajo por lo tanto sera manual.

Que es Git Cherry-pick?  
Este es un commando enfocado directamente a traer un commit especifico de otra rama no es mas es simple pero no solo lo podemos aplicar para ya que podemos hacerlo con varios pero normalemente se hace con solo uno , esto se debe a que con mas commits podria dar multiples errores que tendran que ser arreglados manualmente como todo lo visto hasta el momento.  
  
Comando: git cherry-pick <numerodecommit> <numerodecommit>  
  
  
  
  
  
  
  
  
  
  
Usos en caso de emergencias git  
  
  
  
Git reset and Git reflog  
  
Uno de estos ya es un Viejo conocido pero primero vayamos con el nuevo no tan nuevo el comando git reflog es aquel que nos muestra una historia con todos nuestros archivos como sabemos git nunca pero nunca elimina nada simplemente lo ignora pero lo conserva, imaginate en una situacion donde borres algo muy importante sin querer con este comando no podras encontrar y con git reset que para este punto ya sabemos cuales son sus variantes podemos recuerda los archivos si se lo que pensaste como es posible esto dejame te explica recuerdas que git nunca pero nunca elimina nada y que cada variable de reset especifica un uso no funciona de forma lineal este es mas te podria llegar a sorprende.  
  
  
Git amend  
  
este nos ayudara add cambios a nuestro ultimo primero tenes que add este pero si el commit simplemente git add . y hacemos git commit --amend

Git Grep y log  
  
  
bueno mi estimado viajero este es el final son nuestros dos ultimos comandos con estos comandos puedes buscar desde palabras o letras o hasta incluso si tienes otros tipos de caracteres como >) \*& los podes buscar pero recuerda tiene que estar con comillas para esto nos ayudara grip grep color para saber la linea exacta donde ese encientra es gip grep -n gatos podemos contar la cantidad de veces que se repite tambien con grep -c la y git log nos ayudara en las historias de las commit. Con git log -S primero  
  
comandos colaborativos  
  
  
.- git shortlog. -sn   
muestra cuantos commit ha hecho cada miembro del equipo y si le agregamos. –all este nos mostrara todos los commit que fueron borrados  
  
  
.- git shortlog --sn –all --no-merges  
  
no incluye los merge solo commits  
.- git config –global alias.nombre “comando”  
este no necesita explicacion por lo tanto lo dejare asi  
  
.- git blame   
quien hizo que  
  
  
.- git comandoquedeseassaber –help   
  
 esto te enviara a la documentacion de este  
  
.- git branch -r o git branch -a  
ver ramas remotas esta ultima nos dejara ver tanto las locales como remotas