**Comandos principales de git**

cd e: para entar en apetas

cd.. : para Sali de carpeta

ls : para mostrar contenido en carpeta

git init : inicializar

touch xxxx : crear un arhivo

mkdir xxx : crear carpeta

cat xxxx: para leer en git el contenido del archivo

rm xxxx: para borrar arhivo

ls –al : muestra archivos ocultos

git config –list : para mostrar la configuaracion

git config --global user.name "Eliezer"

git config --global user.email "elefredmason@gmail.com"

code xxxx : abre el archivo en Visual studio code

git add . : para enviar cambio en stagin

git commit -m "cambios xxxxxx"

git commit –a “esto hace cambios enteros como add y commit de una vez peo solo si se la ha hecho git add al archivo msi no no funciona” para pode inertar texto de referencia “esc+i”. y para guardar “esc+shift+zz”

git log xxx : para mostrar los cambios que se han hecho

Q : para Salir del git log

git show xxxx : para mostrar los cambios en archivos

ESC+shift+zz : para salir del mensaje de alerta por no haber agregado un mensaje de referencia

git diff e2021ddf0f0c40f281ba4f6970b2f500f1cff709 18c64729507d576ebc4862dedcfb056f006b0072^C (muestra todas las versiones Hechas)

git log –stat : para mostar los cambios detalladamente

git diff : muestra tanto contenido en memoria ram como en el disco duro

git checkout + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión

git status : para ver modificaciones estado del proyecto

git branch: vemos las remas existentes

git branch –r: vemos las ramas remotas en internet

git branch –a: vemos las ramas tanto local como interrnet

git branch xxxx : crea las ramas

git checkout xxxx: cambia de rama

git merge xxxx: para fusionar cabecera y master (FUSIONAR LAS RAMAS) desde master

**GIT HUB**

git remote add origin https://github.com/elifred21/proyecto1.git

para forzar fusinjar; git pull origin master –allow-unrelted-histories

git remote

Asus@DESKTOP-SE7T0GF MINGW64 /e/eliezer/proyecto1 (master)

$ git remote -v

origin https://github.com/elifred21/proyecto1.git (fetch) BAJAR COSAS

origin https://github.com/elifred21/proyecto1.git (push) ENVIAR COSAS

Asus@DESKTOP-SE7T0GF MINGW64 /e/eliezer/proyecto1 (master)

$ ls -al

total 8

drwxr-xr-x 1 Asus 197121 0 Apr 23 21:56 ./

drwxr-xr-x 1 Asus 197121 0 Apr 23 20:56 ../

drwxr-xr-x 1 Asus 197121 0 Apr 23 22:07 .git/

drwxr-xr-x 1 Asus 197121 0 Apr 23 21:56 CSS/

-rw-r--r-- 1 Asus 197121 790 Apr 23 22:02 inicio.html

-rw-r--r-- 1 Asus 197121 0 Apr 23 20:58 iniciogit.txt

git push origin master (para subir el proyecto a git hub)

git pull origin master (para bajar el proyecto de git hub a git)

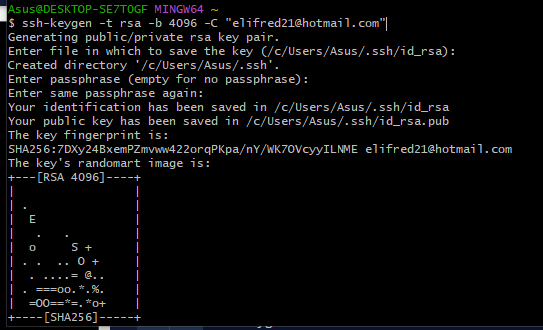
**OJO; ES RECOMANDABLE ANTES DE EMPEZAR A TRABAJAR HACER GIT PULL Y LUEGO EL GIT PUSH**

**MENSAJE SECRETO SIFRADO**

**Llaves pulicas y llaves privadas**

ssh-keygen -t rsa -b 4096 -C "elifred21@hotmail.com"

luego se cra un passprhase; roraima2003



Llave publica desde pc hecto

|  |
| --- |
| ssh-rsa  elifred21@hotmail.com |

Llave pivada desde pc hector

|  |
| --- |
| b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABBX3tC1lE  SAouJ9MzbRimKwAAAAEAAAAAEAAAIXAAAAB3NzaC1yc2EAAAADAQABAAACAQCgPQ5IjJWK  hmUp5LFBTp3ZvUuTb1PeawTQCH86JmRuCUBTLTt0Rr8S64kR8xCRVWAbv8JRgKMv6sLtFT  4MhZrSAjVaR6fy50Bx4ZcarFvXM3gF6jTOJ6NxmXVR+imVfVzSIPqUnn7YOsG0nMeJBKN8  XEy7lPi9P9OFp1++osagXC4sFNn+cr0L9wHrSWoz5gFDNQ2vDeOsVui22NEBbkUK8sD2IP  ggM+Ey7n/TiJ8R385uckW111QZ9/SDJWTkEihyo8WU7hWiRJehUWP4+LQNHsfKTV/4MbLn  XH2AFVDqWDUwbXPQGUMejqB8WCltmcGkfWGbnzCmrmEfACjUNFjWQdLQZnwqtHezDIbT/M  edCTmgK4wy/P9a4aeZ7hyHJwW29cen97yXzMxpfA9qmFxcSRSrMSSkjoBuKqhFnRbO/ABv  unGGtN+oX/TeOpKD+dxa9P9i0iQuFD0RAlhmrLcdI6TeM4MkJA/kxS610oz0AyUmNMcBIL  nQMhFvCtv0DBaZ9Nm+pMyC4nQRFhuv3yJZA8H/0cMuvleWZwD1XElMDPvbbnVqKtko1+L+  grNb7y4dFxxgdvwnZWiGZLmz0ksO965lW3kK0JtTZSE9/Xp5sK54cHOJDtoy67m/Z5Fdpt  M9ngmDNBUmsAckNFzsoWQBYYVWBmsthN4OgG/Y69tjpwAAB1A8+ov4RnL010DcaRpbwmXf  J5/Fau47lDo4knS4jUj9p7VdYaaB59Kut5J4cZ6vZsw46fsHINJO/pA/Ywz9zAoCZ3DaEL  xiiWEPVWhCzadLaYVboVdjCGYeAOEqVxxY0MhSJJyBz71xDIHUVJ4D8fd2il6bNi5XQjfx  utnAkJm0TuP/Iav052dwPLY8oDYATVtDvxy3fvCUMyGPaOmH/17SLDUyOX8w2YRtu1j43o  y6cvZEi/U4jxPnlJhhxfmA/srwJ5gQ93lgHU30NEvgyOpO0epnKcBbQdMIvVkNTEKqmHGY  WYBO5rGh4jnzd96zlXlg7j4K3yA5pGE7qb24JwAM6UAnkQQx8RvaUZ86DiQ0jc6xaYI2QY  DX0sBWzhretgaHul6grL+qf48IyhE1JalEqZ03uRkdJzCB6c5g+do2fS/6IOIChpDXUAls  KDmy8JwXBqBMtHzxuEYxlzzi7zbWQL1s1E1r0ovcWngp5pIfdw65UVx9S0EMngMRfPbb2V  VwfGIpjWI13QTUHB+VGjrW+EQ5ta3NcyHZmOUlU3WP6FmHiTgTi2gUcRNHTLsWH3vhzmQx  PZxLcr3ENDMCuyBr75vlXDoH/XoY3urP0E2ReBcLsEDCRbYe6Z+G10iqS5ROXDRnuLmj+X  xcc55x14E7lZhrSI20glF7UMRW1Ss834UmzGdSIFTqAmasqODabfF0HI0dMl5YjWp9dDcV  L3AljkEvLMH+wHwq8PK+5q/ewpMBXyzIL9lLDLBCe/zzpuRPMpIN7jJxmZVKVhbIumnflu  fJDUTfzzcb4B0sL5TS3sd8BQJwhFkx7YYdnlvMq9jrlBUat40Vy6wrz8s5hEgNJYZV2TQH  8NWlcmW78/3HW9pikdeIn0fv9tIQoyPdtreKk5VWU74HpeDDB3nDKG+jaZjkusVL4wsYW2  7e4ttipWesJMj+UC3om4aOPG5j4ya3nH8aM0cUTd27eB2nX1MWZZZJwcdZ33nPWN+JQyNA  AXwVFZb2a3vy4n3hp60WIcfUg+MaJTa+wBBpFyYxkW2X2AzHOMqVDGiJbmIArcPg/+bLY+  kGqnB7FDHU70ZV/kxo2GNXETt/nYeRxEFFCqXnmHUBCDbhrmfpNqDgenC+b5FOPYSxF0pe  L63B3z1rsjWrf8O042umKlayhfKcQksbtylzn2YUs9JGP14mnpuRyg7CjGiMHgQb10mZL3  3KnFn7D+WXBBjKT7ip/VkT0t9QsQEg7ZwcDTRiuiB7Qc98mQUdgx65v9w44cg5HdyR5rT+  ki62mNrp1Q1y3UhFT2l7RuMwKy36pjol2NhHDN1wQBoU3v455CNbX2DDdyOJ3hYdv3dMCc  Dz5j1JHiOZqT8a6RNPaP4Kq+006D1sEMjkhCHz1yk5lzbWKDfu6jA6mnsS3w52mz8CHBXh  UigdsrI7gETZdp9BDpOXnHbO8y0bKT3kTjAE9rkmFpm+dUJh5jQOlGF5Gcv0kUJmJBoV+h  UxvoiRcAiYojW0JgWnoGvGsZAaQH4yNt7F1/HG73rt0ssY3+LR8DROAxfnWZQHneedC48f  MQ7tVijVdtGdyCgLludXVuZRGMFE+k3W4ufdpblDTvHbtQK8SICbJBwCumIM+8C/i6xEUb  EdCbc5iM4lq6jM3dGB9ZTkMRLTg0TnuSK9A3HRw5uUCJggi5rhMPq07moiqod4Xwlcecgw  wJmdQ7WsdPGc5GT/YsYGLJwQ/ujNOC+oauHXqnlEnAyFfzL+LRiQwfsBSfWN5nQmWMUjRy  kL3XeQGHT46BUurPpCsn4kc9yP9N5OO35URQSlXsxzAf9jMz3GMbt85Ie8K1jL7h/4lyD6  FpGUYbcUIkd0gG6sfyWx9Kkaen+9ZszGjxjFx9CT+CxWuvITMGvTOPul0CCZCgpIx2NdUN  4QrRTvKHhLKcVK9y7ZdZ0tbjGljKgbBUM7SUrDg0XekKZ5FyHwssdpmZVfozgu06z2lnY1  04M5V4hNAe1WfzIo5Cy5Turn8QbzmsgWHseYDXR9StyT02WUGufJNLjEd/Lp90dY3QIEkO  Cz48dItXaSQINvF3tw6H5pQmn4VYTNReeABlj9zq/UhUdbL1c4rapfyNXQNERpzLUFsdDg  f+u2aU510kjmUekknZCm6WF+vWeHAJxSE5NKf3GkHuu6qEdwnLzd0bEQOLUQJ4107i2bum  pqZu9fO4CkpYin0QDQqgTZdjeyHVKkGcSCJwIwuv4+1ul2XCwnfAoJWhIYqwxWCbsqSQxi  im/oBoiEPXtg9RRQnYzCdysNVDHoJYGeZaQ3GyLvIJXjs1+1iGzRuHdRxLGFydBRynnIlE  NOYy0qe4gRkHH+F2v6PHcVJGg4IxOobcPNhauxTn9RrqUeNMzVzJjapkqzEUgS+aFlTAnx  0dguBu12ChlbnWmlrowc7czJMWcnHusoePo3zwov38huGf233pomZRO2aE/WdA9/GtxZKj  +PSQpwlHXiTrt/9KAXZ6hNmt8= |

eval $(ssh-agent -s) para evaluar que están activas las llaves

Asus@DESKTOP-SE7T0GF MINGW64 ~

$ eval $(ssh-agent -s)

Agent pid 1226 SIGNIFICA QUE TODO ESTA BIEN

**PARA SUBIR LA LLAVE AL GITHUB**

Símbolo de virgluilla ~ (alt+126)

**Conexión a GitHub con SSH**

22/43

La creación de las SSH es necesario solo una vez por cada computadora. Aquí conocerás **cómo conectar a GitHub usando SSH**.

Luego de crear nuestras llaves SSH podemos entregarle la llave pública a GitHub para comunicarnos de forma segura y sin necesidad de escribir nuestro usuario y contraseña todo el tiempo.

Para esto debes entrar a la [Configuración de Llaves SSH en GitHub](https://github.com/settings/keys), crear una nueva llave con el nombre que le quieras dar y el contenido de la llave pública de tu computadora.

Ahora podemos actualizar la URL que guardamos en nuestro repositorio remoto, solo que, en vez de guardar la URL con HTTPS, vamos a usar la URL con SSH:

ssh

git remote **set**-**url** origin **url**-ssh-del-repositorio-en-github

**Comandos para copiar la llave SSH:**

-**Mac**:

pbcopy < ~/.ssh/id\_rsa.pub

* **Windows (Git Bash)**:

clip < ~/.ssh/id\_rsa.pub

* **Linux (Ubuntu)**:

cat ~/.ssh/id\_rsa.pub

CAMBIAR EL URL PARA LA LLAVE PUB Y PRIVADA

git remote set-url origin git@github.com:elifred2022/proyecto.git^C

PARA VER ERSIONES HECHAS

git log --all –graph (para mostrar con rayitas)

git log --all --graph --decorate –oneline (lo muestra ma decorado)

ALIAS;

alias tree="git log --all --decorate --oneline"

TAG

$ git tag -a v0.1 -m "resultado de las primeras clases del curso" 3f231e0

git tag (muestra la versión)

git show-ref -–tag

history (muesta todo lo que sa ha hecho en el mproyecto)

git push origin -–tags (para enviarle los tags a git hub pero primero hay q bajarlo)

git tag v0.1 (para eliminar tags en local)

git push origin :refs/tag/v0.1 (para eliminar tag en git hub)

**CLONAR PROYECTO PARA TABAJAR**

git clone git@github.com:elifred2022/proyecto.git

**Cómo se realiza un pull request**

* Se trabaja en una rama paralela los cambios que se desean git checkout -b <rama>.
* Se hace un commit a la rama git commit -am '<Comentario>'.
* Se suben al remoto los cambios git push origin <rama>.
* En GitHub se hace el pull request comparando la rama master con la rama del fix.
* Uno, o varios colaboradores revisan que el código sea correcto y dan feedback (en el chat del pull request).
* El colaborador hace los cambios que desea en la rama y lo vuelve a subir al remoto (automáticamente jala la historia de los cambios que se hagan en la rama, en remoto).
* Se aceptan los cambios en GitHub.
* Se hace merge a master desde GitHub.

**Pasos para hacer deployment en un servidor web:**

* Entrar a la capeta de los archivos del servidor.
* Copiar link en *clone*, elegir entre HTTPS o SSH del repositorio a contribuir.  
  -En la carpeta deseada se clona el repositorio:

git **clone** **url**

Deploy:

* Realizar cambios y *commit* en GitHub.
* Traer al Repositorio local las actualizacion para el servidor en la capeta de los archivos del servidor.

git pull ramaRemota main

Nota: Siempre se debe proteger el archivo .git. Dependiendo del software para el servidor web, existen diferentes maneras. La conexión entre GitHub y el servidor se puede realizar mediante: Travis (pago) o Jenkis (Open source).

# Ignorar archivos en el repositorio con .gitignore

No todos los archivos que agregas a un proyecto deberían ir a un repositorio. Por ejemplo, cuando tienes un archivo donde están tus contraseñas que comúnmente tienen la extensión .env o cuando te estás conectando a una base de datos; **son archivos que nadie debe ver**.

Por diversas razones, no todos los archivos que agregas a un proyecto deberían guardarse en un repositorio. Esto es porque hay archivos que no todo el mundo debería de ver, y hay archivos que al estar en el repositorio ralentizan el proceso de desarrollo (por ejemplo: los binary large objects, blob, que tardan en descargarse).

Para que no se suban estos archivos no deseados se puede crear un archivo con el nombre .gitignore en la raíz del repositorio con las reglas para los archivos que no se deberían subir: Aquí puedes ver la [sintaxis de los .gitignore](https://git-scm.com/docs/gitignore).

Las razones principales para tomar la decisión de no agregar un archivo a un repositorio son:

* Es un archivo con contraseñas (normalmente con la extensión .env)
* Es un blob (binary large object, objeto binario grande), mismos que son difíciles de gestionar en git.
* Son archivos que se generan corriendo comandos, por ejemplo la carpeta node\_modules, que genera **npm** al correr el comando npm install

**SITIO PARA COLGAR IMÁGENES: IMGUR**

**README.MD**

**u sitio web público con GitHub Pages**

34/43

**RECURSOS**

**MARCADORES**

GitHub tiene un servicio de hosting gratis llamado **GitHub Pages**. Con él, puedes tener un repositorio alojado en GitHub y hacer que el contenido se muestre en la web en tiempo real.

Este es un sitio para nuestros proyectos donde lo único que tenemos que hacer es tener un repositorio alojado. En la página, podemos seguir las instrucciones para crear este repositorio

**Pasos para subir un repositorio a GitHub Pages**

* Debemos tomar la llave SSH y hacer un git clone #SSHexample en mi computador local (Home).
* Luego, accederemos a la carpeta nueva que aparece en nuestra máquina local.
* Creamos un nuevo archivo que se llame index.html
* Guardamos los cambios, hacemos un git pull y seguido de esto un git push a master.
* Vamos a las opciones de *settings* de este repositorio y, en la parte de abajo, en la columna Github Pages, configuramos el *source* o fuente para que traiga la rama *master*
* Guardamos los cambios.

Después de esto, podremos ver nuestro trabajo en la web como si tuviéramos nuestro propio servidor.

[**https://pages.github.com/**](https://pages.github.com/)

**vim index.html (crea un arhivo de html en el git)**

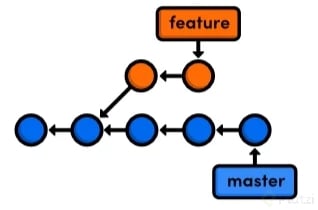
# Git Rebase: reorganizando el trabajo realizado

35/43

**RECURSOS**

**MARCADORES**

Rebase es el proceso de mover o combinar una secuencia de confirmaciones en una nueva confirmación base. La reorganización es muy útil y se visualiza fácilmente en el contexto de un flujo de trabajo de ramas de funciones. El proceso general se puede visualizar de la siguiente manera.

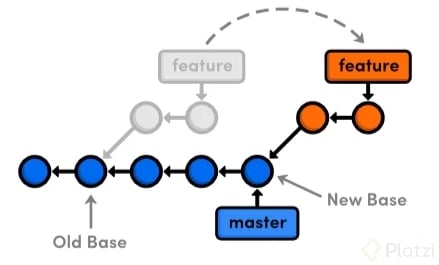


Para hacer un rebase en la rama feature de la rama master, correrías los siguientes comandos:

git checkout feature

git rebase **master**

Esto trasplanta la rama feature desde su locación actual hacia la punta de la rama master:



Ahora, falta fusionar la rama feature con la rama master

git checkout **master**

**git** rebase feature

# No reorganices el historial público

Nunca debes reorganizar las confirmaciones una vez que se hayan enviado a un repositorio público. La reorganización sustituiría las confirmaciones antiguas por las nuevas y parecería que esa parte del historial de tu proyecto se hubiera desvanecido de repente.

El comando **rebase** es \*\*\_una mala práctica, sobre todo en repositorios remotos. Se debe evitar su uso, pero para efectos de práctica te lo vamos a mostrar, para que hagas tus propios experimentos. Con rebase puedes recoger todos los cambios confirmados en una rama y ponerlos sobre otra.

# Cambiamos a la rama que queremos traer los cambios

git checkout experiment

# Aplicamos rebase para traer los cambios de la rama que queremos

git rebase master

# Git Stash: Guardar cambios en memoria y recuperarlos después

36/43

**RECURSOS**

**MARCADORES**

El stashed nos sirve para guardar cambios para después, Es una lista de estados que nos guarda algunos cambios que hicimos en Staging para poder cambiar de rama sin perder el trabajo que todavía no guardamos en un commit

Ésto es especialmente útil porque hay veces que no se permite cambiar de rama, ésto porque tenemos cambios sin guardar, no siempre es un cambio lo suficientemente bueno como para hacer un commit, pero no queremos perder ese código en el que estuvimos trabajando.

El stashed nos permite cambiar de ramas, hacer cambios, trabajar en otras cosas y, más adelante, retomar el trabajo con los archivos que teníamos en Staging, pero que podemos recuperar, ya que los guardamos en el Stash.

## git stash

El comando git stash guarda el trabajo actual del Staging en una lista diseñada para ser temporal llamada Stash, para que pueda ser recuperado en el futuro.

Para agregar los cambios al stash se utiliza el comando:

git stash

Podemos poner un mensaje en el stash, para asi diferenciarlos en git stash list por si tenemos varios elementos en el stash. Ésto con:

git stash save "mensaje identificador del elemento del stashed"

## Obtener elelmentos del stash

El stashed se comporta como una [Stack](https://es.wikipedia.org/wiki/Pila_(inform%C3%A1tica)" \t "_blank) de datos comportándose de manera tipo [LIFO](https://es.wikipedia.org/wiki/LIFO) (del inglés Last In, First Out, «último en entrar, primero en salir»), así podemos acceder al método pop.

El método **pop** recuperará y sacará de la lista el **último estado del stashed** y lo insertará en el **staging area**, por lo que es importante saber en qué branch te encuentras para poder recuperarlo, ya que el stash será **agnóstico a la rama o estado en el que te encuentres**. Siempre recuperará los cambios que hiciste en el lugar que lo llamas.

Para recuperar los últimos cambios desde el stash a tu staging area utiliza el comando:

git stash **pop**

Para aplicar los cambios de un stash específico y eliminarlo del stash:

git stash **pop** stash@{<num\_stash>}

Para retomar los cambios de una posición específica del Stash puedes utilizar el comando:

git stash apply stash@{<num\_stash>}

Donde el <num\_stash> lo obtienes desden el git stash list

## Listado de elementos en el stash

Para ver la lista de cambios guardados en Stash y así poder recuperarlos o hacer algo con ellos podemos utilizar el comando:

git stash list

Retomar los cambios de una posición específica del Stash || Aplica los cambios de un stash específico

## Crear una rama con el stash

Para crear una rama y aplicar el stash más reciente podemos utilizar el comando

git stash **branch** <nombre\_de\_la\_rama>

Si deseas crear una rama y aplicar un stash específico (obtenido desde git stash list) puedes utilizar el comando:

git stash **branch** nombre\_de\_rama stash@{<num\_stash>}

Al utilizar estos comandos **crearás una rama** con el nombre <nombre\_de\_la\_rama>, te pasarás a ella y tendrás el **stash especificado** en tu **staging area**.

## Eliminar elementos del stash

Para eliminar los cambios más recientes dentro del stash (el elemento 0), podemos utilizar el comando:

git stash drop

Pero si, en cambio, conoces el índice del stash que quieres borrar (mediante git stash list) puedes utilizar el comando:

git stash drop stash@{<num\_stash>}

Donde el <num\_stash> es el índice del cambio guardado.

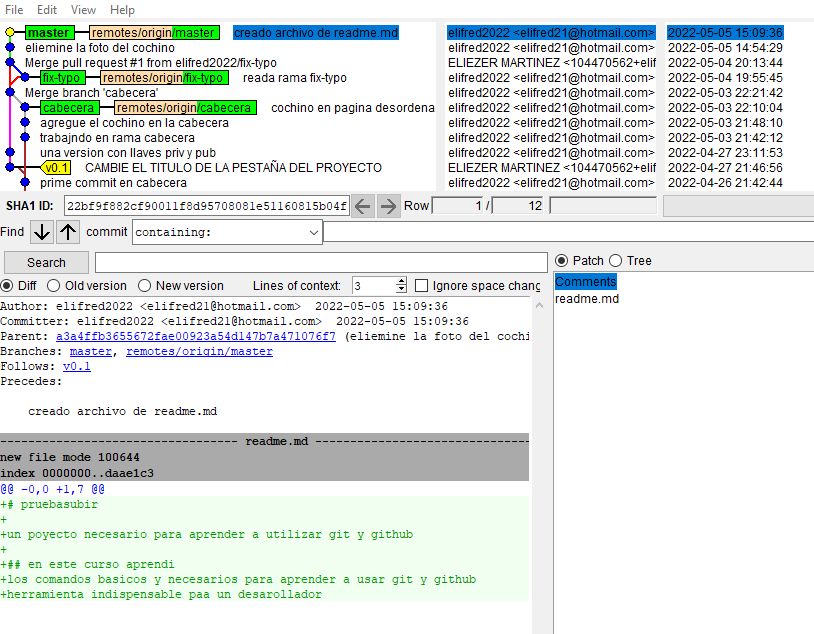
Si, en cambio, deseas eliminar todos los elementos del stash, puedes utilizar:

git stash clear

## Consideraciones:

* El cambio más reciente (al crear un stash) **SIEMPRE** recibe el valor 0 y los que estaban antes aumentan su valor.
* Al crear un stash tomará los archivos que han sido modificados y eliminados. Para que tome un archivo creado es necesario agregarlo al Staging Area con git add [nombre\_archivo] con la intención de que git tenga un seguimiento de ese archivo, o también utilizando el comando git stash -u (que guardará en el stash los archivos que no estén en el staging).
* Al aplicar un stash este no se elimina, es buena práctica eliminarlo.

**GITK**

****

# Git Clean: limpiar tu proyecto de archivos no deseados

37/43

**RECURSOS**

**MARCADORES**

Mientras estamos trabajando en un repositorio podemos añadir archivos a él, que realmente no forma parte de nuestro directorio de trabajo, archivos que no se deberían de agregar al repositorio remoto.

El comando clean actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando add.

$ git clean

La ejecución del comando predeterminado puede producir un error. La configuración global de Git obliga a usar la opción force con el comando para que sea efectivo. Se trata de un importante mecanismo de seguridad ya que este comando no se puede deshacer.

## Revisar que archivos no tienen seguimiento.

$ git clean --dry-run

## Eliminar los archivos listados de no seguimiento.

$ git clean -f

Git clean tiene muchísimas opciones adicionales, que puedes explorar al ver su [documentación oficial](https://git-scm.com/docs/git-clean).

# Git cherry-pick: traer commits viejos al head de un branch

38/43

**RECURSOS**

**MARCADORES**

**Git Cherry-pick** es un comando que permite tomar uno o varios commits de otra rama sin tener que hacer un merge completo. Así, gracias a cherry-pick, podríamos aplicar los commits relacionados con nuestra funcionalidad en la rama master sin necesidad de hacer un merge.

Para demostrar cómo utilizar git cherry-pick, supongamos que tenemos un repositorio con el siguiente estado de rama:

**a** -**b** - c - d Master

\

e - f - g Feature

El uso de git cherry-pick es sencillo y se puede ejecutar de la siguiente manera:

git checkoutmaster

En este ejemplo, commitSha es una referencia de confirmación. Puedes encontrar una referencia de confirmación utilizando el comando git log. En este caso, imaginemos que queremos utilizar la confirmación ‘f’ en la rama master. Para ello, primero debemos asegurarnos de que estamos trabajando con esa rama master.

git cherry-pick f

Una vez ejecutado, el historial de Git se verá así:

**a** -**b** - c - d - f Master

\

e - f - g Feature

La confirmación f se ha sido introducido con éxito en la rama de funcionalidad

## Atención

Cherry-pick **es una mala práctica** porque significa que estamos reconstruyendo la historia, **usa cherry-pick con sabiduría**. Si no sabes lo que estás haciendo, mejor evita emplear este comando.

**Git Reset y Reflog: úsese en caso de emergencia**

39/43

**RECURSOS**

**MARCADORES**

Git guarda todos los cambios aunque decidas borrarlos, al borrar un cambio lo que estás haciendo sólo es actualizar la punta del branch, para gestionar éstas puntas existe un mecanismo llamado registros de referencia o reflogs…La gestión de estos cambios es mediante los hash’es de referencia (o ref) que son apuntadores a los commits…Los recoges registran cuándo se actualizaron las referencias de Git en el repositorio local (sólo en el local), por lo que si deseas ver cómo has modificado la historia puedes utilizar el comando:

git reflog

Muchos comandos de Git aceptan un parámetro para especificar una referencia o “ref”, que es un puntero a una confirmación sobre todo los comandos:

* git checkout Puedes moverte sin realizar ningún cambio al commit exacto de la ref
* git checkout eff544f
* git reset: Hará que el último commit sea el pasado por la ref, usar este comando sólo si sabes exactamente qué estás haciendo
* git reset --hard eff544f # Perderá todo **lo** que **se** encuentra **en** staging **y** **en** **el** Working directory **y** **se** moverá **el** head **al** commit eff544f
* git reset --soft eff544f # Te recuperará todos los cambios que tengas diferentes **al** commit eff544f, los agregará **al** staging area **y** moverá **el** head **al** commit eff544f
* git merge: Puedes hacer merge de un commit en específico, funciona igual que con una branch, pero te hace el merge del estado específico del commit mandado
* git checkout master
* git **merge** eff544f # Fusionará **en** un nuevo commit **la** historia **de** master con el momento específico **en** el **que** vive

¿Qué pasa cuando todo se rompe y no sabemos qué está pasando? Con git reset HashDelHEAD nos devolveremos al estado en que el proyecto funcionaba.

* git reset --soft HashDelHEAD te mantiene lo que tengas en staging ahí.
* git reset --hard HashDelHEAD resetea absolutamente todo incluyendo lo que tengas en staging.

**Atención**

git reset es una mala práctica, **no deberías usarlo en ningún momento**. Debe ser nuestro último recurso.

# Reconstruir commits en Git con amend

40/43

**RECURSOS**

**MARCADORES**

Remendar un commit con amend puede modificar el commit más reciente (enmendar) en la misma rama. Lo ejecutamos así:

git **add** -A # Para hacer uso de amend los archivos deben de estar en staging

git commit --amend # Remendar último commit

Este comando sirve para agregar archivos nuevos o actualizar el commit anterior y no generar commits innecesarios. También es una forma sencilla de **editar o agregar comentarios al commit anterior** porque abrirá la consola para editar este commit anterior.

## ****Atención****

Usar amend es una **mala práctica**, sobre todo cuando ya se ha hecho **push o pull** al repositorio remoto. Al momento de hacer amend con algún commit que esté en remoto, va a generar un **conflicto** que se va a arreglar haciendo un commit adicional y se **perderá el beneficio** del amend.

Aporte creado por: David Behar.

#### **Archivos de la clase**

**Buscar en archivos y commits de Git con Grep y log**

41/43

**RECURSOS**

**MARCADORES**

A medida que nuestro proyecto en Git se hace más grande, vamos a querer buscar ciertas cosas.

Por ejemplo: ¿cuántas veces en nuestro proyecto utilizamos la palabra *color*?

Para buscar, empleamos el comando git grep color y nos buscará en todo el proyecto los archivos en donde está la palabra *color*.

* Con git grep -n color nos saldrá un output el cual nos dirá en qué línea está lo que estamos buscando.
* Con git grep -c color nos saldrá un output el cual nos dirá cuántas veces se repite esa palabra y en qué archivo.
* Si queremos buscar cuántas veces utilizamos un atributo de HTML lo hacemos con git grep -c "<p>".

# Comandos y recursos colaborativos en Git y GitHub

42/43

**RECURSOS**

**MARCADORES**

A continuación veremos una lista de comandos colaborativos para facilitar el trabajo remoto en GitHub:

* git shortlog -sn: muestra cuantos commit han hecho cada miembro del equipo.
* git shortlog -sn --all: muestra cuantos commit han hecho cada miembro del equipo, hasta los que han sido eliminados.
* git shortlog -sn --all --no-merge: muestra cuantos commit ha hecho cada miembro, quitando los eliminados sin los merges.
* git blame ARCHIVO: muestra quien hizo cada cosa línea por línea.
* git COMANDO --help:muestra como funciona el comando.
* git blame ARCHIVO -Llinea\_inicial,linea\_final: muestra quien hizo cada cosa línea por línea, indicándole desde qué línea ver. Ejemplo -L35,50.
* git branch -r: se muestran todas las ramas remotas.
* git branch -a: se muestran todas las ramas, tanto locales como remotas.

Aporte creado por: Juan Sebastián Rodriguez.

#### **Lecturas recomendadas**

[[https://www.google.com/s2/favicons?domain=https://git-scm.com/docs/git-blame/favicon.ico](https://git-scm.com/docs/git-blame)](https://git-scm.com/docs/git-blame" \t "_blank)

[Git - git-blame Documentation](https://git-scm.com/docs/git-blame" \t "_blank)

[https://git-scm.com/docs/git-blame](https://git-scm.com/docs/git-blame" \t "_blank)

# Tu futuro con Git y GitHub

43/43

**RECURSOS**

**MARCADORES**

¡Felicitaciones por terminar el Curso profesional de Git y GitHub!

Aprendimos **cómo usar Git y GitHub**, hacer merge request, investigar quién hizo qué a través de la línea de comandos, cómo utilizar GitHub Pages, cómo revertir cambios y mucho más. Ahora queda de tu parte experimentar, fallar, subir, borrar y por último hacer deploy de tu proyecto y compartirlo con la comunidad.

Recuerda resolver los ejercicios, completar el examen, darle 5 estrellas al profesor y compartir tu proyecto, notas, todas tus dudas y comentarios en la sección de discusiones.

#### **Lecturas recomendadas**

[Curso Básico de Jenkins](https://platzi.com/jenkins/" \t "_blank)

[https://platzi.com/jenkins/](https://platzi.com/jenkins/" \t "_blank)

[Curso Microsoft Azure IaaS | Platzi](https://platzi.com/cursos/azure/" \t "_blank)

[https://platzi.com/cursos/azure/](https://platzi.com/cursos/azure/" \t "_blank)

[Curso de GitLab](https://platzi.com/gitlab/" \t "_blank)

[https://platzi.com/gitlab/](https://platzi.com/gitlab/" \t "_blank)

[Curso Profesional de DevOps](https://platzi.com/devops/" \t "_blank)

[https://platzi.com/devops/](https://platzi.com/devops/" \t "_blank)

[Curso de Azure Iaas](https://platzi.com/cursos/azure-iaas/" \t "_blank)

[https://platzi.com/cursos/azure-iaas/](https://platzi.com/cursos/azure-iaas/" \t "_blank)

**recomienda el cusod e travis o jenkis**