# GIT <Control de versiones>

Kenth1977@gmail-----BNcr1234

**git config --global user.name "kenth1977"** 🡪 Usuario

**git config --global user.email** [**kenth1977@gmail.com**](mailto:kenth1977@gmail.com) 🡪 Email

**git config --global -l** 🡪 Muestra el usuario y el correo creado

**git init** 🡪 Con CMD llegar hasta la carpeta que se va a monitorear y al crearla aparece esta línea

Initialized empty Git repository in C:/Users/Minios/Documents/PYTHON/.git/

**git status** 🡪 Permite conoce el estado de las modificaciones en el proyecto

Lo que está en rojo es información pendiente

**git add .** 🡪 Sube la información a un área de stage antes de subirla al servidor pero antes hay que crear un comentario conocido como commit

**git add pepino.html** 🡪 Agrega al stage la información únicamente de este archivo

**git add python/\*.py** 🡪 Agrega al stage únicamente los archivos con extensión .py

**git reset python /pepino.py** 🡪 Elimina del stage el directorio pepino.py para no ser subido al servido final

**git add python/** 🡪 Agrega un directorio que se encuentra dentro de otro

**git commit -m** "**Descripción inicial perrrrfdsno**" 🡪 Crea un mensaje para poder subir la carpeta.

**git commit --amend -m** "**Descripción inicial pepino**" 🡪 Corrige un commit ya creado

**git diff** 🡪 Muestra la diferencia que hubo entre el documento modificado con el guardado

**git checkout .** 🡪 Muestra diferencia en un documento abierto y retorna al estado

El documento tiene que estar abierto.

**git log** 🡪 Muestra el histórico de cambios creados en la carpeta general

**git help** 🡪 Muestra todos los comandos disponibles de git ejemplo *git help* ***commit***

**git mv pepinillo.py pepino.py** 🡪 crea un control y renombra un archivo creado

**git rm pepino.py** 🡪 Borra un archivo

**git reset --soft** 🡪 recupera un archivo. OJO cuando un commit es creado, crea un registro alfanumérico largo llamado ID. Ese número se puede copiar antes de borrar y respaldar en algún lado por si es necesario retornar a la visualización del histórico. Pero este *no* recupera el cambio.

**git reset –hard** 🡪 Al contrario de soft, hard se devuelve al histórico como soft pero este si recupera el archivo borrado. El único detalle es que al recuperar el commit no aparecerá con git log. Para ver todo los históricos de los commits se utiliza lo siguiente.

**git reset-- mixed** seguido de ID 🡪???

**git reflog** 🡪 Reflog permite recuperar todos los commits inclusive los borrado por rm junto a sus cambios. Reflog muestra la lista y cada elemento del histórico muestra una número con este formato > e33d0c9 < antes del su detalle y tanto rest soft y hard pueden se utilizados con esa numeración alfanumérica como por ejemplo: git reset--soft e33d0c9

**.gitingnore**

Para ignorar un archivo y que este no se suba al proyecto degit primero se crea un documento con el nombre **.gitignore** y dentro de este digitar el nombre del archivo que no se desea. Por ejemplo: se escribe pepino.py, este no se subirá al proyecto remoto. Pero si hay una lista grande de archivos y se quiere subir desde uno en particular para adelante, se usa **pepinillo.py\*** donde el asterisco indica que se va a subir desde ahí para adelante. También se puede crear una carpeta y ahí colocar todo lo que no se desea subir al remoto escribiendo dentro de .gitingnore la ruta de la carpeta por ejemplo: /pepinoingnorado/. También se puede ejecutar para extenciones, por ejemplo .css, .js, .py, .html..etc

**Ramas y uniones(Merge)**

**git branch nueva\_rama 🡪** branch son ramas de prueba que se crean para no alterar la rama master ni los documentos. Con **git log** se visualiza en que rama se está presente y por default aun se encuentra en master viendo algo como esto: *HEAD ->* ***master****, nueva\_rama.*

**git checkout -b nueva\_rama 🡪** Con esta instrucción se pasa a la nueva\_rama. Aquí se siguen utilizando los mismos comandos. Para saber en la rama que se está ubicado se utiliza git log y aparece la información siguiente: *HEAD ->* ***nueva\_rama,*** master*.* NOTA: todo lo que haga en la rama nueva no se verá en la master o viceversa OJO: la -b es una forma de crear una rama y automáticamente me traslade a ella.

**Merge**

Para hacer la unión de **master** con la **nueva\_rama**, **hay que retornar** a la rama master colocando **git checkout master** y se escribe ***git merge nueva\_rama*** y a partir de aquí se puede borrar nueva\_rama solo si no se va a utilizar más

**git branch -d nueva\_rama** 🡪 Borra la rama esclava o la rama de trabajo dejando únicamente la master.

Pueden existir conflicto entre ramas. Los conflictos es que tanto en master como en la nueva\_rama están los mismos documentos, ahí todo bien, pero cuando en la nueva\_rama se cambia el código y luego se vuelve a master y se cambia nuevamente el código, dependiendo el orden de sintaxis o semántica del lenguaje, puede existir una un problema enviando una alerta de **CONFLICT (content)** y donde se encuentra el código. Así que antes de unir es necesario corregir la alerta y luego pasar a master para desde ahí unir master con branch

**git tag pepino** 🡪 Es una etiqueta que se creará en las ramas de prueba quedando de la siguiente manera (HEAD -> nueva\_rama, **tag: pepino**, master). La etiqueta debe ser muy descriptiva colocando una versión consecutiva o colocando el mensaje con su finalidad.

**git tag -d pepino** 🡪 Permite borrar la etiqueta.

**git tag -a v1.0.0 🡪 IMPORTANTE:** Este también es un tag, pero se usa de forma más profesional. Por ejemplo, la **versión 1.0.0** es la inicial de todo el proyecto. Este formato se lee de derecha a izquierda . El valor de la derecha es mínimo cambio el del centro es cambio moderado y el cambio de la izquierda es un cambio significativo.

**git tag -a v1.0.0 -m "Versión inicial" 🡪**Muestra el tag con un mensaje.

**v1 . 0 . 0**

Cambio

significativo

Cambio

moderado

Cambio

mínimo

**git show v1.0.0 🡪 muestra todos los cambios que hay dentro de esta versión.**

**NOTA:** En caso de que exista una versión beta (v0.0.1) y no se haya acordado de escribirla, se puede colocar **git reflog** para ver los commits, buscar el detalle con el formato alfanumérico conocido como id o indicador.

**git tag -a v0.0.1** e33d0c9 🡪 asigna a ese commit con ese identificado, la versión alfa. Recordar que el id o indicador es diferente para cada uno de los commits

**ENLAZAR A REMOTO**

git remote add origin <https://github.com/kerm1977/python.git>🡪 enlaza la carpeta donde se encuentra el init

git push -u origin master 🡪 Envía la orden para subir la carpeta y solicita el username y el password

git pull 🡪 es a la inversa de push. Este traé los cambios efectuados dentro de mi cuenta github.com y las carga a la carpeta de la computadora.

git clone <https://github.com/kerm1977/python.git> 🡪 Esta opción permite descargar un proyecto duplicado a otra carpeta.

**En caso de problemas con el push**:

git config --global pull.rebase true

git config --global rebase.autoStash true

**Abreviar los comando con alias**

$ git config --global alias.co checkout

$ git config --global alias.br branch

$ git config --global alias.ci commit

$ git config --global alias.st status

## Sublime text 3

Sidebarenhancement

SublimeREPL