Notas de Git y GitHub

Lucho Cervantes Jorge Luis

20 de septiembre de 2024

Índice

1. Introducción

Git: Software de control de versiones distribuido.

Una vez instalado Git, en la terminal se configura el nombre de usuario y el correo electrónico, considerando que existe la jerarquía:

Sistema	(toda la computadora)
Global	(todos los repositorios de un usuario)
Local	(un repositorio)

donde se da prioridad al espacio más pequeño (*Local*). Las configuraciones en el espacio *Global* se hacen con los siguientes comandos:

```
git config --global user.name "nombre_usuario"
git config --global user.email "correo_usuario"
```

Para ver las configuraciones globales se usa:

```
git config --global --list
```

y las configuraciones en general:

```
git config --list
```

Nota: La terminal se limpia con el comando:

```
clear
```

Para configurar VS Code como editor de código:

```
git config core.editor "code --wait"
```

Nota: - -wait le indica a la computadora que no guarde los cambios hasta que se cierre el editor de código.

Para configurar el color de la interfaz:

```
git config --global color.ui true
```

El salto de línea en Windows (\r \n) es distinto al de Linux, Mac y Unix (\n). Esto puede generar conflictos al subir y descargar un archivo. Para evitar esto se usa:

```
git config --global core.autocrlf true
```

Nota: si no se está en Windows se usa input en lugar de true.

2. Repositorios

Repositorio: Espacio para guardar administrar y organizar archivos, imagenes, códigos, proyectos etc. Se separa en tres áreas:

Área de trabajo: Donde se realiza la creación y modificaciones del proyecto.

Área de staging (de preparación): Donde se almacenan todos los cambios.

Área de commit: Donde está todo el trabajo ya confirmado.

Para cambiar a un directorio (carpetas):

```
cd nombre_directorio
```

Para regresar un directorio en la ruta:

```
cd ../
```

Para crear una nueva carpeta:

```
mkdir nombre_carpeta
```

Eliminar una carpeta:

```
rmdir nombre_carpeta
```

Inicializar git:

```
git init
```

Mostrar el contenido de la carpeta en la que se está trabajando:

```
ls
```

Mostrar la ruta en la que se está trabajando:

```
pwd
```

Mostrar archivos ocultos en la que se está trabajando:

```
ls -a
```

Nota: el nombre de los archivos ocultos comienza con .

Subir un archivo del área de trabajo (El directorio donde se inicializó git y que contiene la carpeta oculta .git) al área de preparación (Nota:El archivo aún no se sube al repositorio):

```
git add nombre_archivo
```

Nota: Si se quiere subir mas de un archivo, se ponen los nombres separados por espacio

Para mostrar información del área e trabajo y el área de preparación:

```
git status
```

remover un archivo del área de preparación:

```
git rm --cached nombre_archivo
```

Hacer un commit (pasar lo del área de preparación al repositorio):

```
git commit -m "comentario o nota cortos" -a
```

Nota: Al hacer esto el área de preparación ya no cuenta con estos archivos.

Otra forma de hacer un commit es poniendo el archivo en la carpeta que contiene las carpeta oculta .git y usar:

```
git commit
```

De este modo se abrirá el editor para agregar el comentario o la nota. Al guardar y cerrar el archivo se ejecutará la instrucción.

Para crear un archivo en un repositorio, se usa

```
touch nombre-archivo
```

Eliminar un archivo de un repositorio: Al remover el archivo con el comando "rm" y checar el estatus con "git status", se indica que hay un cambio (la eliminación del archivo) que no está listo para ser un commit. Para esto se tiene que subir la nota de que el archivo fué eliminado. Esto se hace nuevamente con el comando "git add nombre_archivo". Al no estar el archivo el programa interpreta la instrucción "add" como que el archivo se borró y cambia su estatus a delete. Finalmente se agraga el comentario con el comando "git commit -m "el archivo se borró etc."

Para restaurar en el área de trabajo, un archivo subido al área de preparación, se utiliza:

```
git restore nombre_archivo
```

Si se tiene un archivo en el área de trabajo y se modifica, esto se señalara al aplicar "git status".

Para observar la versión antes de los cambios se usa:

```
git checkout nombre_archivo
```

El checkout te regresa hasta el último commit. Sin embargo, si se sube el archivo al area de programación (con "add") sin hacer commit, el checkout te llevará a esa versión. Para evitar eso, se utiliza

```
git reset --hard
```

que te regresa al últmo commit descartando todos los cambios locales.

Cambiar el nombre de un archivo:

```
git mv nombre_actual nombre_nuevo
```

Al usar "git status" se indicará como rename.

Para obtener un "status" más sintético:

```
git status -s
```

2.1. Diferencia entre versiones.

Para eliminar un repositorio basta borrar todos los archivos del directorio incluyendo la carpeta oculta ".git".

Para mostrar la última versión commiteada de un archivo en el bash

```
git show nombre_archivo
```

Para comparar lo que se tienen en el área de preparación y el área de commit se usa:

```
git diff --staged
```

Al ejecutar, se muestra en rojo lo de commit y en verde lo del área de preparación.

Comparación entre commits. Primero se obtiene la identificación del commit con:

```
git log
```

Para obtener una versión abreviada (con solo los primeros 7 caracteres de la identificación):

```
git log -- oneline
```

Para configurar el número de caracteres mostrados en dicha versión abreviada se usa:

```
git config --global core.abbrev numero
```

Obteniendo las claves (o sus primeros caracteres) se realiza la comparacción de dos commits con:

```
git diff clave1 clave2
```

Para mostrar solo el nombre de los archivos que cambiaron:

```
git diff --name-only clave1 clave2
```

Para mostrar las líneas que cambiaron:

```
git diff --word-diff clave1 clave2
```

2.2. Modificar un commit

En primera instancia, es menos problemático modificar solo el último commit. Commits intermedios pueden modificarse pero es complejo y se cambian algunas claves lo cual puede no ser conveniente. Para modificar el último commit se usa:

```
git commit --amend
```

Si además de cambiar el comentario se desea modificar o agregar un archivo, se tienen que hacer el cambio y agregar al area de preparación con "add" y después usar el comando anterior.

2.3. Deshacer un commit

El último commit está indicado por el puntero:

(HEAD -> master)

Así, para eliminar los últimos k commits de los n que hay en total, se tiene que colocar el puntero en el commit $n - k$. Esto se hace con

```
git reset --soft clave(n-k)
```

En el caso de "soft", los commits en sí no se eliminan. Estos se pasan al área de preparación donde pueden modificarse y re subirse como nuevos commits. Si el archivo ya existe por un comit anterior, este se reescribe, de otro modo solo se agrega o se mantiene.

Una forma alternativa para el comando anterior es:

```
git reset --soft head ~ (n-k)
```

Nota: el signo ~ (virgulilla) se obtiene tecleando *alt gr* +.

Si en lugar de usar "soft" se usa "mixed" el área de preparación se límpia por completo sin afectarse los cambios realizados en el área de trabajo.

Si se usa "hard" se limpia tanto el área de trabajo como el área de preparación y los archivos del último commit se colocan en el área de trabajo (en particular todos los cambios se pierden por lo que hay que tener cuidado).

3. Ramas (branches)

Para mostrar todas la ramas se usa:

```
git branch
```

Nota: La rama en la que uno se enecuentra se indica con un *.

Para crear una nueva rama se usa:

```
git branch nombre-de-la-rama
```

Para movernos a una rama se usa:

```
git checkout nombre-de-la-rama
```

Una alternativa más moderna y específica para moverse entre ramas es:

```
git switch nombre-de-la-rama
```

En ambos casos hay una manera para crear la rama y trasladarse a ella con un solo comando:

```
git checkout -b nombre-de-la-rama
git switch -c nombre-de-la-rama
```

Nota: la rama se crea en la rama en la que uno se encuentra. Así, según en donde se esté, se pueden crear sub ramas.

Para borrar una rama se usa

```
git branch -d nombre-de-la-rama
```

Nota: es necesario no estar en la rama que se desea borrar (o en alguna de sus subramas).

Para cambiar el nombre de una rama en la que no estamos se usa:

```
git branch -m nombre-actual nuevo-nombre
```

Si se quiere cambiar el nombre de la rama en donde uno se encuentra se usa:

```
git branch -m nuevo-nombre
```

3.1. Fucionar ramas

Para fucionar una rama hay que situarse en la rama principal (o donde quieran agragarse los cambios de la sub rama) y usar:

```
git merge nombre-de-la-otra-rama
```

Nota: en rigor, lo que se fucionan son los commites. no tanto las ramas. De hecho, al usar "branch", aún se verán las dos ramas.

Para revertir una fución de ramas se usa:

```
git reset --hard clave
```

Donde la clave es del commit anterior al commit donde las ramas ya están fucionadas.

Nota: En realidad lo que pasa es que se elimina el commite con las ramas fucionadas sin afectar a las ramas secundaria y principal

Una ves satisfechos con los cambios y fucionados a la rama principal, se puede borrar la rama donde se hcieron las modificaciones con "git branch -d nombre-de-la-rama".

3.1.1. Merge conflicts

Con el comando "git log –oneline" se obtienen todos los commits de la rama en la que uno se encuentra y de las rammas anteriores. Para ver también los commits de las ramas posteiorez se usa:

```
git log --oneline --all
```

Puede ocurrir que al crearse otra rama para realizar algún cambio o mejora, la rama principal sigua sufriendo cambios o mejoras a la par, de manera que al fusionar las ramas surgirá un conflicto sobre cuáles cambios se mantienen, ya sea los hechos en la rama principal o los hechos en la rama secundaria. Esto será decisión de quienes trabajen en el proyecto. Para facilitar la toma de decisiones y la combinación de los cambios, el editor de código cuenta con la herramienta "Merge Editor".

3.2. Git ignore

Archivo de nombre ".gitignore" que contiene el nombre de todos los archivos del área de trabajo son ignorados al usar "add ." (subir todo al área de preparación) y al hacer commits. i.e. no se suben.

En el archivo se pueden hacer comentarios con # y en cada linea se colocan los nombres de los archivos a ignorar. Si se quieren eliminar todos los archivos con algún formato se coloca *. Por ejemplo, si se quieren ignorar todos los archivos de texto se usa:

```
*.txt
```

Si se quiere hacer una excepción se usa:

```
*.txt
!nombre_archivo.txt
```

Para ignorar todo un directorio se coloca:

```
Nombre-directorio/
```

Si se quiere hacer una excepción se usa:

```
Nombre-directorio/nombre-archivo
```

Para observar todos los archivos que contiene un commit, solo por su nombre, se usa:

```
git ls-tree -r --name-only clave-del-commit
```

Nota: Cuando se trate del último commit, puede sustituirse su clave por la palabra HEAD.

3.2.1. Git ignore global.

Si se tiene un archivo que sin importar el proyecto nunca va a subirse, para no estar incluyendolo en el archivo ".gitignore" de cada repositorio se puede crear un archivo que contenga su nombre y configurarlo a manera de que sea ignorado, de manera global, en cualquier repositorio. Para esto se usa:

```
git config --global core.excludesfile ruta
```

donde "ruta" es la ruta del archivo que contiene el nombre o el formato de los archivos que se ignorarán de manera global.

3.3. Alias

Se usa para renombrar se manera simple un comando largo que es usado con regularidad. Por ejemplo se tienen los siguientes comandos:

```
git log --oneline

git log --oneline --all

git log --oneline --all --graph

git log --oneline --all --graph --pretty=format
"%C(auto)%h%d %s %C(black)%C(bold)%cr"
```

El agregado "graph" sirve para que los commites se muestren en un diagrama donde se pueden observar las ramas del proyecto.

El agregado "pretty=format:%C..." sirve para que tambien se muestre hace cuanto se hizo cada commite con un cierto formato y color de fuente.

Naturalmente Desde el tercer ejemplo ya es algo largo para teclear constantemente, de manera que se usa alias para renombrarlos de manera más simple:

```
git config --global alias.nombre-corto "comando
```

Nota: en el caso del último comando, este incluye comillas dobles. Estas deben sustituirse por comillas simples al reescribirse en el comando anterior.

3.4. Git reflog

Lleva un registro de todos los movimientos del apuntador HEAD ->.

Al borrar un commit en realidad no se borran los archivos que estos contienen, lo que se borra es la refeencia a estos. Así en caso de eliminar accidentalmente un comit con "git reset –hard clave" este puede recuperarse nuevamente con "git reset –hard clave". Sin embargo puede que no recordemos dicha clave. Para esto se usa:

```
git reflog
```

Que muestra todos los commits hechos.

4. Github

plataformas para crear proyectos abiertos de herramientas y aplicaciones, y se caracteriza sobre todo por sus funciones colaborativas.

4.1. Git clone

Es la descarga de un repositorio cargado en el servidor de Github para poder trabajar con él desde nuestra computadora.

Para abrir VS Code desde el bash se usa:

```
code .
```

El editor se abre con la carpeta en la que nos encontremos.

Para clonar un repositorio de Github se selecciona la opción $< >$ Code, se copia la URL en HTTPS y se usa en el bash de Git:

```
git clone URL
```

4.2. Git push

Una vez realizados los cambios deseados en el repositorio descargado deseamos subir estos cambios de vuelta al servidor. En principio, estos cambios se encuentran en nuestra área de trabajo, por lo que primero deben subirse al área de preparación con " git add" y luego deben ser commiteados con cualquiera de las dos maneras de las que ya se ha hablado. Hecho esto, se tiene que configurar el correo electrónico con:

```
git config --global user.email "email"
```

donde el email es aquel con el con el que se abrió la cuenta de git. Hecho esto, los cambios se suben con el comando:

```
git push origin master
```

donde "origin" hace referencia al lugar de donde se descargó y master, que se quiere subir en la rama principal del repositorió original del servidor.

4.3. Git pull y git fetch

Cuando se baja un repositorio de un servidor para hacerle cambios, nada garantiza que el repositorio sufra otros cambios en el mismo servidor o que algún otro usuario lo descarge, lo modifique y lo vuelva a subir. En principio esto causará un conflicto cuando querramos subir nuestros cambios. Antes de aprender a solucionar esto es util saber cómo bajar solo los cambios realizados por otras personas en el servidor. para esto se usa:

```
git pull
```

Nota: al usar este comando no solo se descargan los nuevos arhivos. También se actualizan los cambios en los archivos ya existenetes.

Cuando se quiere descargar un cambio con "pull" pero nosotros ya hemos realizado también otros cambios, al momento de hacer la combinacion de los cambios se tendrá un problema similar al obtenido al fucionar dos ramas por lo que los se tendran que elegir que cambios hacer. Para tener un control de esto se usa:

```
git fetch
```

Para ver los cambios tenemos que crear a una rama ficticia en donde están dichos cambios. Para esto se usa:

```
git switch --detach origin/master
```

4.4. Git remote

Sirve para conectar un repositorio local (creado desde la computadora) a un servidor. En el caso de Github, debe crearse en el servidor un repositorio en blanco y aplicar el comando que no es sugerido después de crearlo:

```
git remote add nombre URL
```

donde "nombre" no es necesariamente el nombre del repositorio, en realidad como desea referenciarse al repositorio remoto desde nuestra computadora. "URL" es la dirección del repositorio remoto y es proporcionada por el servidor al crearlo.

Un repositorio local puede conectarse a distintos servidores, por ejemplo, se puede tener un repositorio local conectado a Overleaf y a Github a la vez. Para saber en qué servidores está conectado un repositorio local y las acciones que pueden hacerse en cada uno de ellos se usa:

```
git remote -v
```

Una vez conectado el repositorio local con el remoto, se crea la rama principal con

```
git branch -M master
```

Y se sube con un "push" lo deseado desde el repositorio local al remoto.

```
git push -u origin master
```

Nota: en particular, en el primer "push" se utiliza -u (abreviación de –set-upstream) para configurar origin master por defecto y no tener que escribirlos cada que se haga un push.