INTRODUCCION:

Muchas de las compañías y empresas hoy te piden tu portafolio de Git. Este portafolio es donde guardas las distintas versiones de los códigos que hayas usado anteriormente, así la empresa puede ver en que proyectos has trabajado antes y que tipo de código puedes escribir.

Tools:

Diff: Esta herramienta se usa para mostrar la diferencia entre dos scripts, o entre dos archivos. Esta misma función puede tener variaciones como por ejemplo el -u que se usa para poder dar un contexto mas amplio de que es lo que sucede, esto te dirá que líneas fueron agregadas y que líneas fueron eliminadas.

Wdiff: Solo muestra las palabras que han sido cambiadas en el archivo.

Patch: Se le llama Patch al hecho hacer cambiar un código con un Diff file y sobrescribirlo, es lo mismo que hago cada vez que patcheo un programa como after effects o 3 DS Max.

>> patch cpu\_usage.py < cpu\_usage.diff

>> patching cpu\_usage.py

>> |

Como podemos ver el patching usa el standard input, o sea, el signo menor que para indicar que debemos introducirle el Diff a el archivo .py. Hay que recordar que el primer parámetro del patch debe de ser el archivo a patchear y no viceversa.

Que es un Repositorio:

El repositorio de git, es una carpeta oculta, las carpetas ocultas tienen un punto delante de estas, un repositorio git se puede crear fácilmente mediante el uso de **git init** el cual es un comando que crea automáticamente una carpeta llamada .git en el directorio en el que estamos trabajando, y dentro de esta carpeta crea las demás carpetas que son necesarias para trabajar con el repositorio.

Cuando se esta creando el git, podemos salvar en la configuración de git, nuestro nombre y nuestro email, haciendo uso de:

git –global user.email “example”

git –global user.name “My name”

El directorio o área fuera de .git, es lo que se llama el working tree o sandbox, donde se realizan todas las modificaciones al archivo y donde se encuentra la última versión del archivo.

El hecho de que el archivo este en esta carpeta no significa que git lo esta siguiendo, o teniendo en cuenta, para esto hay que añadirlo, como aparece debajo.

El área o directorio dentro de /.git es lo que se llama **index**, o **staging area** , el index es un archivo git que contiene toda la información sobre que archivos y cambios van a ir en su siguiente commit.

Para añadir un elemento al git, el elemento debe de estar en la carpeta exterior, o sea en el mismo directorio padre que /.git, luego que el archivo a añadir al repositorio esta en esta carpeta, podemos ejecutar el comando **git add file.py** para agregar este al repositorio.

Luego podemos comprobar el estado del repositorio con **git** **status**, lo que nos devolverá los archivos que tenemos actualmente en nuestro repositorio.

Si devuelve **to be committed**, esto significa que el cambio aun esta en el archivo index, o staging area, o sea que los cambios no se han aplicado a los archivos que esta fuera, en el working tree.

Para poder modificar el archivo podemos ejecutar **git commit**, este comando abrirá por defecto el editor de texto que le hemos pasado como default a la hora de instalar git. Una vez ejecutamos el commit, se guarda, para cada caso un ID único para cada commit que se ha realizado.

Si necesitamos revertir un cambio, porque el script no está funcionando como debía después de modificarlo y este aun se encuentra en la etapa de modificado y no ha sido agregado al index, o sea, que cuando revisemos **git status** nos devuelva **Modified** en rojo podemos volver atrás haciendo uso del comando **git checkout**.

Si el script pasa para la Staged area, y lo queremos como estaba antes porque se nos olvido hacer **git checkout**, podemos usar **git reset** cuando el script está en esa etapa.

Si nos equivocamos cuando estamos haciendo el commit o queremos reescribirlo, podemos cambiar el ultimo commit con el comando **git commit --amend**

Para revisar los cambios que se han hecho podemos marcar **git log**.

Podemos revisar cuales fueron las líneas que se cambiaron en el código haciendo uso de **git log -p.** La p viene de patch, muestra cuales son las líneas que fueron patcheadas o cambiadas. Esto es equivalente al uso de diff **-u**.

Otra forma de mostrar información en la pantalla es usar **git log –stat**  este mostrara las diferencias en un formato diferente.

Podemos saltarnos el paso de pasarlo por el index si sabemos que eso es lo que queremos finalmente para nuestro código, si y solo si, este archivo ya a ha sido tomado en cuenta por git antes, o sea si el archivo está tracked. Esto lo hacemos usando el comando **git commit -a**, en lugar de usar **git add**, y luego **git commit**.

Si no queremos saber todos los datos de todos los commit que se han realizad, y solamente queremos saber las especificaciones de uno de ellos podemos primero usar **git log** para extraer el commit ID, y con este usar **git show s6gb48er4** (De cuatro a 8 caracteres debe de ser suficiente).En este caso ese numero tan largo, es el ID del commit que queremos expandir, es como marcar **git log -p** pero solamente a uno de los elementos.

Para remover los archivos del repositorio podemos usar **git rm.**

Para renombrar los archivos del repositorio podemos usar **git mv.** Como estamos haciendo un cambio en el archivo, que esta dentro del directorio git, y no lo estamos eliminando como con **git rm**, a este cambio hay que pasarle el comando **git commit**, para que sea efectivo.

Cuando ya le hemos pasado el comando commit a nuestro script, y este ya esta en la historia, y tiene un bug, podemos hacer uso de **git revert**  que no significa que va a volver al anterior script y va a eliminar este error del historial, sino que va a hacer un nuevo commit, dejando plasmado en la historia que este error ocurrió en el pasado, y que se hizo **git revert** que en realidad lo que hace es hacer un commit opuesto al que se hizo en el anterior, dándonos así, el anterior commit nuevamente y dejando nuestro script funcionando para así tener tiempo de encontrar donde es que se encuentra nuestro bug,.

Para devolvernos a una que esta mucho más atrás que el solamente el ultimo

Cada proyecto git, que no es mas que modificar un archivo para guardar sus cambios en el tiempo, consiste en tres secciones.

El directorio /.git: Es el fucking directorio git

El index, el cual es un archivo llamado index dentro del directorio /.git

Working tree: es el directorio padre del directorio git.

Todo esto se hace para que se entienda mejor el proceso, pero normalmente cuando estemos trabajando con estos archivos solamente los meteremos en el directorio padre o working tree, y la historia o git dir, y el index eran cosas que se trabajaran desde el prompt.

Cada vez que hacemos commit a un archivo, una copia de que fue lo que se modificó y cuando es guardado en la historia en el directorio /.git.

Hay dos tipos de archivos los que están siendo tracked y los que no, los untracked, podemos convertir un archivo de untracked a tracked añadiéndolo con el comando git add.

Los archivos tracked, son los archivos que git esta tomando en cuenta para modificar y guardar estas modificaciones en su historia. Estos archivos pueden estar en tres fases, Modified, Staged, Committed.

Siendo modificado la etapa en la que los cambios no se han guardado en el archivo index, Staged, cuando los cambios han sido debidamente guardados al index o Staged area, y committed, cuando finalmente la modificación surte efecto en el archivo.

El repositorio git debe ser puesto en nuestra carpeta donde almacenamos nuestros scripts, así cada vez que realicemos una modificación a este, esta se guardara en la historia.

Una vez que agregamos un nuevo script a nuestra carpeta de scripts, si le pasamos git status, este nos devolverá que esta untracked, para poder agregar este archivo al git directory, y, por tanto a la historia, debemos ponerlo en la Staged area, y luego pasarle el comando commit, esto se hace para crear una nueva instancia, una primera línea en la historia de este archivo que bien podría ser, se ha creado el archivo, si le pasamos el git commit como está, esto nos redirigirá a al editor de texto, para evitar esto solamente debemos pasarle el parámetro -m al comando y pasar el comentario que queremos mostrar en la historia: git commit -m ”Se ha creado el archivo,”

Head:

Head es el alias, es el valor que utiliza git para indicar que este en la última modificación en esta rama, que este es el que este es el más reciente. Nos muestra en que momento estamos en el código.

Branch:

Las Branches o Ramas son las diferentes ramas o versiones de las versiones de un script o programa, la rama principal se llama master Branch, normalmente cuando tenemos una idea nueva, lo que hacemos antes de introducirla en la master Branch, es probarla en una rama por separado y luego, si esta funciona como esperamos, la introducimos en la rama principal.

Para poder ver todas las ramas del script o proyecto que tenemos actualmente, podemos usar  **git branch**. Cuando aplicamos ente comando nos sale una lista de todas las ramas que existen, y la rama en la que estamos trabajando actualmente nos sale con un asterisco y un color diferente para identificarla del resto.

Para crear una nueva rama, podemos usar **git branch new-feature**.

Para cambiar de rama usamos **git checkout new-feature**.

También para usar estos dos comandos juntos podemos usar **git checkout -b even-better-feautre**.

Si necesitamos eliminar una rama podemos hacer uso de **git branch -d new-feature**.

Merging:

Se usa para poder combinar una rama con la master branch, o con otra. Esto combina tanto los datos dentro de las ramas con las historias de cada una de ellas. Esto se hace estando en la rama principal, o en la rama que vamos a tomar como tronco principal del código, y luego pasándole merge a la rama a la que vamos a agregar, así:  **git merge even-better-feature**.

Cuando hacemos esto, el HEAD va a apuntar a las dos ramas, separándolas por una coma.

Hay dos tipos de Merging, está el Merging donde no se hizo ningún commit, y el merge no necesita más que solo agregar el nombre de la rama al head, y esta el otro donde los commit divergen y merge si necesita trabajar y ver como los combina.

En este caso, se pueden dar lo que se llaman Merging conflicts.

Merging Conflicts:

Estos aparecen cuando editamos las mismas líneas de código en ambas ramas, y hacemos que devuelvan resultados diferentes, para resolver el conflicto solamente debemos editar eso y dejar solamente el resultado que queremos.

Podemos usar **git log --graph --oneline** para poder ver gráficamente la divergencia.

Para abortar el merge podemos usar  **git merge --abort**

El merge lo que hace no es combinas las dos ramas como yo creía, lo que hace es agregar, a la rama en la que estamos, el contenido de la rama que le pasemos, si queremos que estas dos ramas estén al día, tenemos que hacer el merge dos veces, la una contra la otra.

A las ramas remotas, no se les toca, con ellas se trabaja desde los repositorios locales y se les modifica con los git logs,

El fetch se usa cuando hubieron modificaciones en el repositorio remoto y le hacemos git log y estas no d se cargan en el git log del repositorio local, para esto, hacemos git fetch y este nos cargara toda le historia.

Notas:

La única manera de poder trabajar con los remotos es hacer pull o push, no puedo intentar ahcelo de otra forma