TAREA I

TEC | Tecnológico de Costa Rica

MICROPROCESADORES Y MICROCONTROLADORES

Prof. Rodolfo José Piedra Camacho

Herramientas GitHub, PyTest y Flake 8 para control de versiones y revisión de código Python.

Alfaro Cordero Randall
Castro Hidalgo Ignacio
Fallas Martinez Jasson
Gonzalez Rosabal Francisco

I SEMESTRE – FEB. 2020

¿Qué es Git?

Corresponde a un software de control de versiones. A su vez, un software de control de versiones es una herramienta que permite gestionar los cambios o actualizaciones que se hacen sobre los elementos de un producto.

Este producto puede ser, por ejemplo, un software o una página web. Por ende, Git permite subir o modificar a un grupo de programadores código a una nube de manera sencilla.

De forma más específica, permite tratar con distintas versiones de un mismo software mediante el manejo de una versión: guarda los archivos de cada versión distinta. Además, permite flexibilidad al programador: si este desea realizar un cambio sobre el que no está seguro, este puede ejecutar dicho cambio en una forma tal que el código original no se vea alterado, o bien, en caso de que el programador desee realizar "*rollback"* o borrar el cambio (por ejemplo, debido a un bug), dicha acción se puede hacer sin ningún problema. Este tipo de características, son las que le dan forma a un software de control de versiones como Git.

¿Qué es GitHub?

Es una plataforma que brinda el servicio de nube, para que desarrolladores puedan almacenar y administrar sus repositorios mediante Git. Lo que permite centralizar el contenido del proyecto para poder colaborar en equipo o bien compartir alguna fuente de código abierto, por ejemplo.

Un repositorio se define como un directorio donde se almacenan los archivos del proyecto. Se puede guardar localmente en el ordenador o en el almacenamiento de GitHub o plataforma similar (como GitLab). Permite guardar prácticamente una gran variedad de tipo de archivos.

¿Qué es un Branch?

Es una copia del repositorio. La rama se suele utilizar para realizar un desarrollo de forma aislada. Al trabajar en una rama no se afectará el repositorio central u otras asociadas. Si se completa el trabajo es posible combinar la rama con otras, o con incluso, el repositorio central. A su vez, las ramas permiten respaldar si los cambios de la rama no están del todo finalizados, y permiten regresar a la versión del proyecto original antes de hacer los cambios mencionados.

Por ejemplo, suponga por un momento que una versión está en su estado estable, pero el programador o diseñador desea agregar una nueva característica experimental. Es trivial ver que, si esta característica no funciona en el primer intento, entonces el código original alterado tendría que devolverse a su estado original: o se pierde el trabajo

realizado sin poder mejorarlo a posteriori, o los usuarios deben descargar un código original alterado que puede no funcionar bien. Aquí, es donde un Branch entra en acción: al crear uno, se pueden realizar modificaciones al Branch sin que el código original se vea alterado.

Con el siguiente comando se puede crear un nuevo *branch*, copiado del *branch* en el que se está a la hora de ejecutar el comando:

git branch <branch-name>

Asimismo, para ver los *branches* existentes, se utiliza:

git branch

¿Qué es un Commit?

Corresponde a la acción de guardar archivos a un repositorio remoto, en otras palabras, una actualización o añadido de un nuevo archivo. A su vez, se puede guardar localmente, dependiendo del lugar donde esté almacenado el repositorio.

Para ilustrarlo de una mejor manera: agregar un archivo o modificar un archivo físico en la carpeta del repositorio, no modificará lo que Git contiene del archivo (internamente continuará con el código viejo). Git sí detectará sin embargo nuevos archivos o archivos modificados y notificará al usuario que debe realizar una especie de *submit* a dichos cambios. Esta acción se le conoce como *encomendar* o *commit*, y corresponde a decirle a Git que tome en consideración las nuevas modificaciones para el repositorio. Mientras el *commit* no se realice, los archivos estarán en un estado transitorio mientras el programador termina de realizar los cambios (con este mecanismo, se puede controlar de una mejor manera la versión y al mismo tiempo los cambios de este).

El commit se realiza con:

git commit -m “Algún mensaje para el commit”

En donde el mensaje representa alguna pequeña descripción de la actualización.

¿Qué Se Entiende Cuando Se Dice Que Un Archivo Está Staged?

Forma parte del flujo de trabajo de un archivo en Git, junto con la etapa de modificado y consolidado. La etapa de modificado se da cuando en el archivo se han detectado las modificaciones y se han indexado, pero no se han consolidado (*commit*). La etapa “*staged*” o en escenario se da cuando al archivo se le han marcado las modificaciones y su siguiente paso corresponde a consolidarlo. Consolidar un archivo a

su vez, significa que el archivo y su información han sido guardados en la base de datos de Git.

En otras palabras, un archivo *staged* es aquel que está en la etapa transitoria mencionada en explicaciones anteriores (en la explicación del commit): Git conoce de la existencia de los cambios, pero no los toma en consideración de forma permanente hasta que no se realice *commit*.

¿Qué Hace El Comando Git Checkout?

Se utiliza para cambiar entre ramas (Branch) y alternativamente las puede crear. Actualiza los archivos en el árbol de trabajo para que coincidan con la versión en el índice o árbol especificado; es decir, los archivos actuales de la carpeta del repositorio serán actualizados de tal forma que coincida con el del *branch*.

Se debe tener en consideración que los archivos que hayan sido *staged* por *modificaciones*, no guardarán sus cambios si se realiza *checkout* del *branch* actual antes de realizar *commit*.

Para cambiarse a un *branch* ya existente, se utiliza:

git checkout <branch-name>

Y para crear un nuevo *branch* y cambiarse al de forma inmediata se puede utilizar:

git checkout -b <branch-name>

Estos comandos tienen como función prepararse para trabajar en la rama especificada, se cambia actualizando el índice y los archivos en el árbol de trabajo y señalando HEAD en la rama (HEAD es el *commit* más nuevo del *branch* actual). Las modificaciones locales se mantienen de los archivos de los árboles de trabajo para que puedan consolidarse en la rama. Al especificar “-b”, se crea una rama como si la rama en la que se viene trabajando se haya llamado y cerrado. En caso de poner “-B”, crea una nueva rama en caso de que no haya una, en caso de haber una, la reinicia.

¿Qué Hace El Comando Git Stash?

Permite congelar el estado en que se encuentra el proyecto en un momento determinado, con todos los cambios antes de realizar un commit (incluyendo los *staged files*), guardándolo en un espacio provisional permitiendo poder recuperarlo más tarde. Es usado cuando es necesario mantener el estado actual del proyecto, pero se quiere regresar a verlo sin las modificaciones realizadas, o bien continuarlo posteriormente sin necesidad de formalizarlo con un *commit*.

Los cambios guardados en el espacio provisional se pueden listar con el comando *git stash list,* se pueden inspeccionar con *git stash show,* y restauradas con *git stash apply.*

Es importante mencionar que llamar al comando *git stash* sin argumentos es igual a realizar el comando *git stash push.* El anterior comando graba el estado actual de los archivos y los acumula en pila. Luego, regresa los archivos al HEAD. Estos archivos se vuelven a recuperar al realizar *git stash pop*.

¿Qué Hace El Comando Git Add?

Git estará consciente siempre de los archivos modificados, sin embargo, al realizar un commit no los considera al menos que estén *indexados*. Es decir, se le haya especificado a Git que esas modificaciones se deben tomar en consideración en el siguiente *commit* y por ende los debe agregar al índice que él maneja.

Este comando permite mover al índice las modificaciones que se hayan realizado. Se puede decir que el índice a su vez es un *snapshot* del contenido del área de trabajo en un momento dado. Este *snapshot* se convertirá después en un commit al realizar *git commit -m "Algún mensaje para el commit"*.

En otras palabras, el comando actualiza el índice usando el contenido que se encuentre en el momento, en el árbol de trabajo. Generalmente agrega el contenido de las rutas existentes en su conjunto, pero con opciones específicas puede agregar el contenido con solo una parte de los cambios hechos en los archivos del árbol de trabajo. Es importante resaltar, que este comando no agrega archivos "ignorados" por default. Para agregar dicho tipo de archivos se debe agregar -f a la sentencia (archivos ignorados son aquellos listados en un archivo llamado ".gitignore", que pretende decirle a Git cuáles son los archivos que no se deben tomar en consideración en el repositorio aunque se encuentre en la carpeta, por diversas razones) .

git add nombre_del_archivo

El comando debe ejecutarse por cada nuevo archivo o archivo modificado antes de realizar commit. Si se desea ver el estado de los archivos indexados (archivos *staged*) o no, se puede utilizar *git status*.

¿Qué Hace El Comando Git Reset HEAD~?

El comando *reset* como tal, tiene como naturaleza regresar todos los archivos, incluyendo los que estaban en el índice, a un commit pasado. Cada *commit* posee un *ID* el cual se puede utilizar para especificar hacia cuál encomienda se desea regresar. Asimismo, el comando se puede utilizar en combinación con *HEAD*, que indicará que todos los archivos en el área de indexado se deben regresar al último commit indicado.

Ahora bien, si se agrega el símbolo ~ a HEAD, se está indicando entonces a Git que se desean deshacer los cambios del último commit. Si se agrega --soft al comando, los cambios no se pierden, sin embargo, si no que se colocan en un estado de "pendiente", para un próximo *commit*. Asimismo, si se agrega --hard, todos los archivos, se forzará el regreso hacia el *commit* reiniciado.

Más aún, con *git reset HEAD~2* por ejemplo, se pueden retroceder dos *commits* en el *branch* actual.

¿Qué es PyTest?

Herramienta que permite escribir códigos de prueba usando Python. Entre los alcances de dicha herramienta se incluye:

1) La posibilidad de probar desde bases de datos, interfaces de programación de aplicaciones e interfaces de usuario.
2) Brinda información detallada sobre las declaraciones de aserción fallidas, autodescubrimiento de módulos y funciones de prueba.
3) Una arquitectura rica de complementos con más de 315 complementos externos, a su vez, la compatibilidad con versiones de Python a partir de la 3.5.

En otras palabras, es una herramienta especial para pruebas unitarias, donde esta corresponde a la revisión de cada módulo por separado (de ahí unitarias) de tal forma que se inyectan parámetros de entrada de prueba, y se revisa que la respuesta de dicho módulo sea efectivamente la esperada ante dicho estímulo.

Bajo El Contexto de PyTest, ¿Qué Es Un Assert?

Verifican si las expresiones devuelven o son equivalentes a lo esperado o no. De devolver lo correcto, el programa muestra que el módulo responde correctamente al estímulo y continua a la siguiente prueba unitaria. En caso de fallo, la herramienta mostrará un mensaje de fallo o lo contabilizará como un *assertion failure*, si el programa detalla o no el error o para la ejecución de las pruebas ante un *assertion failure*, depende de la configuración del usuario que ejecuta las pruebas unitarias.

Por ejemplo:

assert "ropa" == "cuatro" es un assertion failure.
assert 4==4 es un successful assertion.

¿Qué Es Flake 8?

Corresponde a una librería que permite revisar la integridad de un código, según formato (PEP8, es decir, si los elementos están bien espaciados, entre otros detalles), errores de programación mediante PyFlakes (por ejemplo, librerías importadas pero inutilizadas o variables sin usar) y complejidad ciclomática mediante McCabe (que tan compleja una función puede llegar a ser, por ejemplo, ante el uso excesivo de *nested ifs* o *else if*). La herramienta permite evaluar la calidad del código generado y generar un reporte de cuál es el error y en qué línea se encuentra.

Una complejidad ciclomática atañe a un software que permite medir el número de rutas independientes a través del código fuente. A grandes rasgos, entre mayor número de estas rutas dentro de una función, mayor camino tendrá; por ende, mayor complejidad ciclomática.