PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada
2° semestre 2015

Actividad 05

Estructura de Datos - Nodos y Árboles

Instrucciones

Se desea desarrollar un **Sistema de Control de Versiones** similar a **Git** pero mucho más simple llamado **Jit**. En vez de considerar cambios entre lineas y bytes en archivos, **solo se considerarán la creación y el borrado de archivos que serán representados con sus nombres en formato *strings***.

Recordemos que *Git* es como un flujo de trabajo que se compone de ***branches*** o ***ramas*** que van guardando cambios a través de *commits*. Toda *branch* nace a partir de un *commit* de otra *branch*. Las *branches* sirven para trabajar en algún punto particular del proyecto sin interferir en el trabajo en las otras ramas.

Esto mismo queremos replicar en *Jit*. Para cada *commit* debe ser posible conseguir el estado del repositorio. Esto se hace partiendo del *commit inicial* y componiendo todos los cambios de los *commits* sucesivos hasta el *commit* deseado. Así uno puede *viajar en el tiempo* a una versión anterior.

Cada ***commit*** es representado con:

- Un *id* numérico (asignado automáticamente).
- Un *message* con una descripción del *commit*.
- Una **lista de tuplas de strings** llamada *files* que indica los cambios ocurridos. El primer ítem de la tupla indica la acción *CREATE* ó *DELETE* en formato *string*. El segundo ítem lleva el nombre del archivo como *string* sobre quien se aplica la acción. De modo que un *commit* válido sea del estilo:

```
commit = Commit(
    message="Borrar datos viejos y nuevas instrucciones",
    changes=[("DELETE", "data.txt"), ("CREATE", "README.md")]
)
```

Similar a *Git*, en *Jit* todo repositorio parte con un ***branch*** llamado *master*. Adicionalmente, se tiene que crear un *'commit inicial'* que crea un archivo llamado *.jit*.

Requerimientos

Debe trabajar en el archivo *main.py*, asegúrate de leer las instrucciones y revisar el final del archivo donde sale el uso esperado del programa para un repositorio de ejemplo. Al final de este documento se encuentra un diagrama de ese mismo ejemplo.

- Modificar y completar las clases *Commit* y *Branch* para que sea posible establecer la estructura de *Jit*. **Está prohibido usar como atributo en la clase *Branch* una lista (*list()*, []) de *commits*. Pero sí se puede usar listas en funciones para realizar ciertas operaciones.**
- Debe ser posible realizar **pull** a algún *branch* en específico del repositorio. Al hacer esto se debe retornar el último estado del repositorio (la lista de archivos final hasta el último *commit* de esa rama).
- Debe ser posible crear nuevas *branches* en el repositorio.
- Debe ser posible hacer *checkout* de un *commit* en específico en el repositorio. Este método recibe la *id* de un *commit* y debe retornar el estado del repositorio hasta ese *commit*. Este *commit* puede estar en cualquier *branch*.

Consideraciones

Asuma que todo uso del *jit* será correcto.

- No se creará un archivo si ya está creado (dentro de la misma *branch*).
- No se borrará un archivo si ya fue borrado o todavía no se crea (dentro de la misma *branch*).
- No habrán *commits* vacíos, es decir, siempre tendrán al menos un cambio.
- No se intentará acceder ni modificar ramas no creadas.

To - DO

- (1.50 pts) Clase *commit* y *branch* correctas. Se asigna 0 puntos si no respetas la restricción de las *listas*.
- (1.50 pts) Método `pull` correcto.
- (1.00 pts) Método `create_branch` y correcto.
- (2.00 pts) Método `checkout` correcto.

Anexo: representación del repositorio de ejemplo