Vamos a crear nuestro propio sistema de control de versiones, que permite administrar los sucesivos cambios que los usuarios hacen sobre sus archivos.

Archivos y carpetas

Los archivos sobre los que trabajan los usuarios constan de un nombre que los identifica y un contenido que se puede representar mediante una cadena de caracteres. Los archivos están agrupados en una misma carpeta, de la que también se conoce su nombre. No puede haber dos archivos llamados igual en una misma carpeta, pero no hay ningún conflicto con que se repitan los nombres de archivo en diferentes carpetas.

Se pide:

1. Averiguar si una carpeta contiene un archivo con un nombre dado.

Commits

Un commit es como una “receta” de tareas para hacer en una carpeta. Tiene una descripción y un conjunto de cambios a realizar. Cada cambio es sobre un sólo archivo, para lo cual se debe indicar su nombre. Los cambios existentes son los siguientes, aunque en el futuro podría haber otros:

- “Crear”: hace que aparezca el archivo con el nombre indicado y vacío de contenido. Si ya existe un archivo con ese nombre en la carpeta, no se puede realizar este cambio.
- “Eliminar”: hace que desaparezca el archivo de la carpeta.
- “Agregar”: implica agregar el texto indicado al final del contenido del archivo.
- “Sacar”: implica sacar el texto indicado del final del contenido del archivo.

Salvo “Crear”, los demás cambios no pueden realizarse en caso que no exista en la carpeta ningún archivo cuyo nombre sea el indicado. Cuando un cambio no pueda realizarse, debe interrumpirse la ejecución y advertir apropiadamente.

Se pide:

1. Dada una carpeta, **aplicar** todos los cambios de un commit lo que implica que la carpeta quede modificada. Por ejemplo:
    - En una carpeta vacía llamada“pdep” se aplican los 3 cambios de un commit con descripcion "commit inicial”: uno que crea el archivo “leeme.md”, otro que crea el archivo “parcial.wlk” y otro que agrega “Este es un parcial” al archivo “leeme.md”. Finalmente, la carpeta debe quedar con 2 archivos: “leeme.md” con el contenido “Este es un parcial” y el archivo “parcial.wlk” vacío.
    - El mismo commit anterior, cuando se aplica en una carpeta que ya contiene un archivo llamado "parcial.wlk", al realizar el "Crear" debe interrumpirse la ejecución.

Branches

Una “branch” consta de un conjunto de commits que representa la evolución de una carpeta a lo largo del tiempo.

Se pide:

2. Hacer el **checkout** de una branch en una carpeta: Consiste en aplicar sucesivamente los cambios de todos los commits de la branch en la carpeta.
3. Conocer el **log** de un archivo a partir de una branch: Consiste en obtener una lista de los commits de la branch que afectan a dicho archivo.

Revert
El revert es un commit que deshace el efecto de otro commit. Hacer el revert de un commit consiste en obtener un nuevo commit con el opuesto de cada uno de los cambios del commit original, en orden invertido, y agregarle "revert " a su descripción.

Se pide:

4. Obtener el **revert** de un commit. Por ejemplo:
    - Al hacer un revert del commit del ejemplo anterior se consigue un nuevo commit con la descripción “revert commit inicial” y 3 cambios: Primero sacar el contenido “Este es un parcial” del archivo denominado “leeme.md”, luego eliminar “parcial.hs” y por último eliminar “leeme.md”.
    - En una carpeta vacía, al hacer un checkout de una branch que contiene un commit y su correspondiente revert, la carpeta vuelve a quedar vacía.

Autores
El sistema incluye información sobre sus usuarios y gestiona las acciones que pueden o no realizar. Cuando un usuario crea una nueva branch puede agregar a otros usuarios como colaboradores, habilitándolos de esta manera a comitear, es decir, agregar un nuevo commit al branch. Obviamente, él mismo también tiene permiso para commitear.
Hay usuarios administradores que están habilitados para commitear sobre un branch sin necesidad que su creador les haya dado permisos. Los administradores tienen también la atribución de hacer que cualquier usuario se convierta en otro.
También existen usuarios comunes y usuarios bot. Los usuarios bot sólo pueden commitear en branches con más de 10 commits, y no pueden modificar permisos.
Se registra siempre qué usuario es el autor de cada commit.

Se pide:

5. Crear una branch agregando colaboradores.
6. Commitear: Hacer que un usuario commitee en un branch, en caso de tener los permisos necesarios.
7. Hacer un blame de una branch y un archivo: Consiste en obtener todos los autores diferentes que modificaron al menos una vez dicho archivo.
8. Que un administrador cambie el rol a un conjunto de usuarios dado (por ejemplo, haga que todos sean bot).

Por ejemplo:
El usuario Marcelo crea una branch y agrega como colaboradores a Guillermo y Gustavo, todos usuarios comunes. Guillermo commitea creando un archivo "Final" y agregándole como contenido "Boca 2". Guillermo sigue trabajando y hace un nuevo commit agregandole "River 0" al archivo, Marcelo commitea sacando "0" del archivo "Final" y agregando "3" al mismo archivo. Viene elAdmin, que es administrador y hace un nuevo commit revirtiendo el último commit. Gustavo hace un commit creando el archivo "recaudacion". elAdmin convierte en administradores a todos los usuarios del blame del archivo "Final". Marcelo hace que elAdmin sea un bot. Gustavo quiere volver a hacerlo administrador a elAdmin y no puede. ¿Qué se obtiene si se hace un checkout de la branch en una carpeta vacia y se le pide el contenido al archivo "Final"?

Importante
Se debe realizar:

- La codificación de la solución en Wollok
- Un diagrama de clases
- En base a los ejemplos, hacer 3 tests:
    - uno que verifique que se haya producido efecto correctamente,
    - uno que verifique que no se pueda hacer algo,
    - y uno que verifique que se devuelva bien algo
- Justificar la utilización de polimorfismo