Sistemas Operativos

Proyecto de primer corte

1 Sistema de Control de Versiones

Los Sistemas de Control de Versiones (VCS) permiten guardar el rastro de las modificaciones sobre determinados elementos. En el contexto de este proyecto, se gestionarán versiones de archivos y directorios.

Se deberá implementar un sistema de control de versiones simple, que permita:

- Adicionar un archivo al repositorio de versiones.
- Listar las versiones de un archivo en el repositorio de versiones.
- Listar todos los archivos almacenados en el repositorio
- Obtener la versión de un archivo del repositorio de versiones.

En esta implementación sólo se deberá realizar el control de versiones por directorio, en el cual sólo se pueden agregar archivos que se encuentren en el directorio actual (no recursivamente).

La aplicación implementada se deberá llamar **versions**. Su uso será el siguiente:

```
$ ./versions
Uso: versions add archivo "Comentario"
     versions list archivo
     versions list
     versions get numver archivo
```

1.1. Repositorio de versiones

Al invocar el comando **./versions**, se deberá crear un sub-directorio llamado **.versions**, si no existe, en el directorio actual de trabajo. Dentro de este directorio se almacenarán las diferentes versiones del archivo, y un archivo llamado `versions.db` que contendrá la descripción de cada versión.

1.2. Adición de archivos al repositorio

Al invocar el comando:

```
versions add test.c "Primera version"
```

Se deberá verificar si la versión actual del archivo es diferente a las anteriores versiones guardadas. Si se cumple esta condición, se creará la nueva versión del archivo en el sub-directorio y se añadirá una línea al archivo `versions.db`.

El siguiente pseudocódigo presenta una posible implementación de la adición de un archivo al repositorio de versiones:

```
estructura registro_archivo
  char nombre_archivo[PATH_MAX] //Nombre original del archivo
  char comentario[BUFSIZ] //Comentario
  char hash[256] //hash del contenido y nombre del backup
fin estructura


/Adiciona un archivo al repositorio con un comentario
//Retorna 1 si el archivo fue adicionado correctamente
//        0 en caso contrario
funcion add(char * archivo, char * comentario)

  //Verifica si el archivo existe
  si stat(archivo, &s) < 0
    retornar 0
  fin si

  //Verifica si es un archivo regular
  //! significa negación
  si ! S_ISREG(s.st_mode)
    retornar 0
  fin si

  //Obtener el hash del archivo
  hash = obtener_hash(archivo)

  //Nuevo nombre del archivo: .versions/ABCDEF....
  //ABCDEF... es el hash del archivo

  //Construir la cadena (strlen, malloc, strcpy, strcat)
  nuevo = ".versions/" + hash

  //Copia el archivo al directorio .versions, adiciona el
      registro
  //al archivo versions.db
```

```
  //r : estructura de tipo registro_archivo

  si copiar_archivo(archivo, nuevo)
    r.nombre_archivo = archivo
    r.comentario = comentario
    r.hash = hash
    //TODO adicionar la version al archivo "versions.db"
    adicionar_version(r)
    retornar 1
  fin si

  retornar 0

fin funcion
```

Tenga en cuenta que la función `adicionar_version` adiciona un nuevo registro al archivo `versions.db`. Esto se puede implementar de dos maneras:

1. Leyendo el contenido completo del archivo, almacenarlo en una estructura de datos temporal, adicionar el nuevo registro al final y escribir de nuevo todo el contenido del archivo.

2. Abrir el archivo `versions.db` en modo `adicionar` (append), y escribir únicamente el nuevo registro.

1.3. Listado de versiones

El comando:

```
versions list test.c
```

Deberá imprimir la información que se encuentra dentro del archivo versions.db del sub-directorio correspondiente para el archivo.

El siguiente pseudocódigo presenta el proceso para listar las versiones:

```
estructura registro_archivo
  char nombre_archivo[PATH_MAX] //Nombre original del archivo
  char comentario[BUFSIZ] //Comentario
  char hash[256] //hash del contenido y nombre del backup
fin estructura

funcion list(char * nombre_archivo)

  //Abrir el archivo ".versions/versions.db"
  fp = fopen(".versions/versions.db", "r")

  si fp = nulo
    retornar
  fin si
```

```
  cont = 1
  //Leer hasta fin de archivo
  mientras (!feof(fp)) {
    //Realizar una lectura y validar
    si fread(&r, sizeof(registro_archivo), 1, fp) != 1
      break
    fin si

    //Si el registro corresponde al archivo buscado, imprimir
    //Comparacion entre cadenas: strcmp
    si r.nombre_archivo = nombre_archivo
      imprimir cont, r.hash, r.comentario
      cont = cont  + 1
    fin si
  fin mientras

  fclose(fp)

fin funcion
```

1.4. Obtener una versión del archivo

El comando:

```
./versions get 1 test.c
```

Obtendrá la versión solicitada del archivo del repositorio, sobreescribiéndolo si éste ya existe.

El siguiente pseudocódigo presenta una posible implementación de la lógica para obtener la versión del un archivo.

```
estructura registro_archivo
  char nombre_archivo[PATH_MAX]
  char comentario[BUFSIZ]
  char hash[256]
fin estructura


//Obtiene una versión solicitada (numero secuencial) de un
    archivo
funcion get(char * nombre_archivo, int version)

  //Abrir el archivo ".versions/versions.db"
  fp = fopen(".versions/versions.db", "r")

  si fp = nulo
    retornar
  fin si
```

```
  cont = 1
  //Leer hasta fin de archivo
  mientras (!feof(fp)) {
    //Realizar una lectura y validar
    si fread(&r, sizeof(registro_archivo), 1, fp) != 1
      break
    fin si

    //Buscar el archivo y la version solicitada
    si r.nombre_archivo = nombre_archivo
      si cont = version
  //Copiar el archivo
  copiar_archivo(r.hash, r.nombre_archivo)
  break
      sino
  //Buscar la siguiente version
  cont = cont  + 1
      fin si
    fin si
  fin mientras

  fclose(fp)

fin funcion
```

1.5. Detección de cambios

El esquema para detección de cambios en un archivo puede ser muy complejo. Para la implementación de este programa, se usará el módulo auxiliar `sha256`, el cual implementa el algoritmo del mismo nombre y permite obtener el código SHA de 256 bits (64 bytes + NULL). (ver función `get_file_hash` en el código base proporcionado).

1.6. Copiar un archivo

Se deberá implementar una función para copiar el archivo al subdirectorio de versiones. Dado que no existe una llamada al sistema específica para este propósito, se deberá abrir el archivo original y copiar su contenido al nuevo archivo. Esto se puede lograr mediante dos familias de funciones diferentes: `open read write close`, que operan sobre descriptores de archivo, o `fopen fread fwrite fclose` que operan sobre flujos de datos (streams). En cualquier caso, la lógica es la que se presenta en el siguiente pseudocódigo, cambiando las funciones de alto nivel por las respectivas llamadas:

```
funcion copiar_archivo(char * fuente, char * destino)


  //Abrir el archivo fuente
  f = abrir(fuente, MODO)
  //TODO Verificar si se pudo abrir

  //Abrir el archivo destino
  df = abrir(destino, MODO)
  //TODO Verificar si se pudo abrir

  //leer determinada cantidad de bytes en buf
  //copiar a destino, mientras no sea fin de archivo
  //TODO dependiendo de la familia usada, se debe
  //verificar que se hayan leido datos
  mientras EXISTAN DATOS POR LEER
    leer(f, buf)
    //TODO Verificar cuatos bytes se leyo
    //Solo se debe escribir la cantidad de bytes leidos
    escribir(df, buf)
  fin mientras
  cerrar(f)
  cerrar(df)
fin funcion
```

1.7. Llamadas al sistema

Algunas de las llamadas al sistema y funciones de biblioteca que se pueden usar para la solución del problema presentado son: `fork` `exec` `popen` `exit` `open` `read` `write` `close` `fopen` `fclose`, `sha256sum` `stat`. Para gestionar cadenas de caracteres, se debe reservar memoria dinámica con `malloc`, copiar (`strcpy`), concatenar (`strcat`), longitud (`strlen`), comparación (`strcmp`).

1.8. Condiciones de entrega

- Se debe entregar el código fuente de la aplicación, debidamente identificados sus desarrolladores.
- El código debe estar debidamente documentado.
- Entrega individual o en grupos de máximo dos (2) personas.