14 de Junio de 2012

R212 Estructuras de Datos y Algoritmos I

Trabajo Práctico Final

Tema 1: Versionado de Archivos

Motivación

Este trabajo práctico es para permitir al alumno demostrar que puede almacenar de manera eficiente en espacio la representación de la historia de cambios de un conjunto de archivos, y que puede manipular esa representación de forma eficiente en tiempo.

Objetivos

Se deben diseñar dos programas:

- my-diff: que compara dos archivos no binarios y muestra sus diferencias línea a línea
- my-repo: que guarda persistentemente la historia de los cambios a una jerarquía de archivos y directorios para poder mostrar las diferencias entre la jerarquía en el momento actual y el estado que tuvo la jerarquía en un punto determinado del pasado

Detalles

El trabajo deberá ser hecho por una sola persona. Ambos programas deben estar escritos en C, y cada uno debe tener un Makefile que lo construye. Los programas pueden usar bibliotecas para todo lo que no sean los algoritmos involucrados en el cálculo de la LCS y de la construcción y manipulación del árbol de hashes de archivos. Pero de usarse una biblioteca ésta debe ser open source, siguiendo los lineamientos del DCC para la entrega de trabajos prácticos.

Ninguno de los dos programas debe fallar con errores no controlados como por ejemplo *segmentation fault* ante entradas extrañas o parámetros incorrectos. Además ambos programas no deben tener *memory leaks*, sino en cambio deben liberar todos los recursos que ya no utilicen.

El programa my-diff debe mostrar una salida idéntica a la mostrada por el programa estándar diff sin argumentos. Por ejemplo para la diferencia entre hello1.c y hello2.c:

```
$ cat hello1.c
#include <stdio.h>

int main (int argc, char *argv[]) {
  printf("Hello World!\n");
```

```
  return 0;
}

$ cat hello2.c
#include <stdio.h>

int main (int argv, char *argc[], char *env[]) {
  // yes, argv and argc are not mandatory names...
  puts("Hello World!");
  return 0;
}

$ my-diff hello1.c hello2.c
3,4c3,5
< int main (int argc, char *argv[]) {
<   printf("Hello World!\n");
---
> int main (int argv, char *argc[], char *env[]) {
>   // yes, argv and argc are not mandatory names...
>   puts("Hello World!");
```

Estudiar la salida del diff estándar para varios tipos de diferencias posibles, y diseñar el programa my-diff para que funcione de la misma forma. Utilizar para el cálculo de las diferencias de líneas el algoritmo de Longest Common Subsequence como se explicó en clase y con los detalles de [Cormen sección 16.3].

El programa my-repo debe implementar los comandos:

- init
- add
- rm
- status
- commit
- log
- compare

El comando init crea en el directorio actual un directorio `.index` en Unix o _index en Windows, conteniendo un archivo `contents` que guardará el catálogo de los archivos bajo control de versiones por my-repo en el directorio actual. Si el directorio .index o _index que se intenta crear ya existiera, el comando debe fallar. Inicialmente ningún archivo estará bajo control de versiones por my-repo, hasta que no se invoque `my-repo add`. El catálogo deberá contener la hora y fecha en que él mismo fue creado con `my-repo init`. Cuando un directorio contenga un directorio .index o _index con un catálogo de `my-repo`, ningún subdirectorio de éste podrá contener otro directorio .index o _index y por lo tanto ningún subdirectorio deberá contener un repositorio `my-repo`.

El comando `add` recibe como parámetro un archivo o un directorio y lo coloca bajo control de versiones, hasta tanto no se haga `rm`. Si el parámetro es un directorio, pone bajo control de versiones todos los archivos que contenga ese directorio y recursivamente todos los directorios que estén bajo él.

El comando `rm` recibe como parámetro un archivo y provoca que el mismo deje de estar bajo control de versiones. Si recibe como parámetro un directorio, usted debe decidir como manejar

ese caso.

El comando `status` en un directorio que esté bajo control de versiones, muestra el estado en el que encuentra cada archivo contenido en el directorio, sea que esté bajo control de versiones o que no lo esté. Los estados pueden ser:

- Modified (el archivo versionado fue modificado desde la última vez que se le hizo commit)
- UpToDate (el archivo versionado no fue modificado desde la última vez que se le hizo commit)
- Untracked (el archivo no se encuentra bajo control de versiones)

El comando `status` en un directorio que no esté bajo control de versiones muestra un mensaje de error diciendo que el directorio no está bajo control de versiones con `my-repo`.

El comando `commit` recibe como parámetro un flag `-m` seguido de una cadena entre comillas dobles que corresponderá a un mensaje de commit. El comando deberá guardar para todos los archivos en estado Modified la diferencia entre el contenido actual y el contenido que tuvo la última vez que se le hizo `commit` o `add`, lo que haya ocurrido más tarde. El comando commit solo tendrá éxito en el directorio raíz del repositorio (el que contiene el .index o _index). Un commit exitoso se guardará con un número entero identificador. Los identificadores de commit's deben crecer incrementalmente de a 1 unidad.

El comando `log` sin parámetros deberá mostrar en orden cronólogico descendente la lista de todos los commit's hechos en el repositorio, con su identificador, fecha, hora y mensaje de commit correspondiente. Si el comando `log` recibe un parámetro lo interpretará como un número de commit y mostrará archivo por archivo las diferencias entre el repositorio como estuvo en el commit dado por el parámetro y el repositorio actual.

El comando `compare` recibirá como parámetros dos directorios .index o _index correspondientes a repositorios y determinará si los repositorios son iguales o no. Si los repositorios difieren, además mostrará para cada archivo diferente la diferencia entre ellos como la muestra el comando `my-diff`.