PRACTICA 2: Programación de Shell Scripts e IPC

SHELL SCRIPTS

Ejercicio 1: Comando tac

a) Interpretar la página de manual del comando **cat**, realizar pruebas del comando

b) Crear un script que imprima el contenido de un archivo de texto de manera invertida, es decir primero la última línea, luego la penúltima y así sucesivamente hasta imprimir en último lugar la primer línea.

Nota: existen muchas formas de realizar esto, pueden ser de utilidad los comandos **head** y **tail**

Ejercicio 2: Día de cumpleaños

El usuario debera ingresar en formato dd-mm-aaaa la fecha en que nació, el script deberá retornar el día de la semana de dicha fecha.
Nota: La fecha introducida debe ser valida, el script verificará esto, en caso de fecha inválida el script aborta comentando dicha situación.

Ejercicio 3: Días de vida

El usuario debera ingresar en formato **dd-mm-aaaa** la fecha en que nació, el script deberá retornar la cantidad de días transcurridos hasta la fecha.
Nota: La fecha introducida debe ser valida, en caso contrario el programa aborta.

Ejercicio 4: Compresión tar.bz2

Escribir un programa que solicite al usuario la siguiente información:

- El path absoluto a un directorio que contenga archivos
- El nombre dedicado al archivo comprimido

Con esta información se debe crear un archivo comprimido .tar.bz2 con todo el contenido del directorio de entrada.
Al finalizar el script debe mostrar el tamaño total del directorio sin comprimir y el tamaño del archivo comprimido.
Nota: La existencia del directorio de entrada debe ser validada, si no existe el programa aborta.

Ejercicio 5: Renombre masivo de archivos

Crear un sript que permita, dado un directorio como entrada, renombrar un conjunto de archivos que posean una extensión determinada.

Por ejemplo:

```
$ ls /directorio/*
a.c  b.c  c.c  x.c

$ bash ejercicio3.sh
Ingrese un directorio: /directorio
Ingrese extension: .c
Ingrese patron de renombre: programa
Se renombra a.c a programa1.c
Se renombra b.c a programa2.c
Se renombra c.c a programa3.c
Se renombra x.c a programa4.c
Se muestra contenido:
programa1.c   programa2.c   programa3.c   programa4.c
```

Ejercicio 6: Verificar password

a) Crear un script que tome como entrada dos archivos de texto *usuarios.txt* y *claves.txt*, ambos deberán contener una única palabra por línea. El primero contendrá nombres de usuario, mientras que el segundo claves de acceso en formato plano. Ninguno puede ser vacío y deben contener la misma cantidad de líneas (ambas condiciones deben ser validadas por el script).
El script deberá generar un nuevo archivo *passwd.txt* que contendrá registros de la forma *usuario:clave*, donde usuario proviene del archivo usuarios.txt y clave del archivo claves.txt.

b) Crear un script que emule el proceso de login de un usuario, solicitando nombre de usuario y luego clave, la clave introducida no deberá mostrarse por pantalla mientas de tipea y debe cotejarse con la clave correspondiente al usuario en el archivo passwd.txt creado en el apartado a).

Ejercicio 7: Front-end para el comando find

Mediante las *man pages* u otro recurso investigar el funcionamiento del comando *find* y crear una interfaz de usuario que permita buscar archivos según los siguientes criterios:

> Tipo (directorio, archivo, pipe con nombre, symbolic link)
> Tamaño (bytes, kbytes, megabytes)
> Permisos (lectura, escritura, ejecución , combinaciones de los anteriores)
> Por inodo

Ejemplo:

```
$ bash my_find.sh
```

deberá mostrar un menú similar al siguiente

```
Interfaz para comando find
1 - Busqueda por tipo de archivo
2 – Busqueda por tamaño de archivo
3 – Busqueda según los permisos del archivo
4 – Busqueda por inodo
Introduzca criterio de búsqueda:
```

En este punto se le pedirá al usuario que especifique el *path* desde donde se comenzará la búsqueda y las entradas correspondientes a la selección realizada.

Comentarios:

- Para la interfaz de usuario utilizar la estructura de control *select*
- Utilizar *case* para determinar y tratar cada criterio
- Considere el uso de funciones para reducir código

IPC

Ejercicio 8: Pipes y popen

(a) Construir un programa en C que emplee popen (ver presentación dada en clases) donde el padre permita al usuario ingresar un texto de varias líneas por pantalla (nueva linea <Enter>, para terminar) y el proceso hijo muestre la cantidad de líneas, palabras y caracteres que contiene (usar comando `wc`).

(b) Repetir el ejercicio anterior sin usar popen, es decir, emplear pipe, fork, dup2 y exec.

Ejercicio 9: Shared Memory

Un proceso denominado *director* alloca y escribe en un segmento de *shared memory* un número entre 0 y 2 el cual hace las veces de "identificador" (**ID**), y un número (**NRO**) con valor inicial 1 (en este orden).

Además existen 3 procesos *jugadores*, identificados por los **ID** en [0, 2] (no es el PID es un número que se le asigna explícitamente en una variable global), que realizan continuamente lo siguiente:

- Leen de la memoria compartida ambos valores, el **ID** y el **NRO**.
- Si **ID** es el que le corresponde tomará el valor de **NRO** y lo reemplazará por el entero que le sigue que no contenga entre sus cifras a 3 ni sea múltiplo de 3.
- Escribirá por pantalla un mensaje similar a este:
  "Se ejecutó el proceso [ID], se realizó el reemplazo NRO_viejo -> NRO_nuevo"
- Cambiará el valor de **ID** por el del *siguiente proceso* (secuencia ... 0 1 2 0 1 2 ...)
- Cuando **NRO** alcance el valor de 50, el proceso correspondiente, deberá asignar a **ID** el entero -1. (**ID** del proceso *director*)

Finalmente el proceso *director* deberá liberar la memoria compartida utilizada por los procesos e imprimir por pantalla **"Ciclo finalizado"**

Notas:

- en cada intervención de los procesos jugadores estos "attachan" y "dettachan" la memoria compartida al accederla
- Cada uno de los procesos jugadores deberá ser lanzado en una terminal diferente y se le pasara el id del segmento de memoria compartida como argumento

Ejercicio 10: Mapped Memory

Un proceso, denominado *productor*, crea un archivo con 100 enteros aleatorios incluidos en el rango [1,1000]. Posteriormente mapeará el contenido del archivo a memoria.

Otro proceso denominado *sorter*, haciendo uso de `qsort`, ordenará estos enteros directamente en memoria. Finalmente *productor* debe mostrar por pantalla el contenido del archivo ordenado.

Nota: Las tareas de creación y eliminación y mapeo del archivo son responsabilidad de *productor*.

Ejercicio 11: Pipes

- Crear un proceso que solicite al usuario una operación aritmética sencilla, que involucre solo los operadores binarios **+**, **-**, ***** y /, y que de la forma: ***entero operando entero***.
- Este proceso entonces creará un proceso hijo, el cual recibirá dicha operación mediante un *pipe*, la resolverá y luego enviará al proceso padre el resultado por *otro pipe*.
- El padre, a continuación, mostrará el resultado de la operación por pantalla.
- Finalmente consultará al usuario si pretende realizar otra operación.
  En caso afirmativo deberán repetirse los pasos anteriores.

Utilidades:

- Convertir cadenas a enteros: **`int atoi(const char * cadena)`** de **<stdlib.h>**
- Convertir enteros a cadena: **`int sprintf(const char*, char *fmt, ...)`** de **<stdio.h>**

Ejercicio 12: FIFOs

Repetir el ejercicio anterior, pero ahora vinculando procesos no emparentados utilizando FIFOs.