Trabajo Práctico Nº 1: Scripting

1. Manipulación del contenido de un archivo:

   a) Convierta el contenido de un archivo de texto a mayúsculas y guarde el resultado sobre el mismo archivo.

   b) Reemplace los dígitos del contenido de un archivo por un carácter dado como parámetro.

   c) Implemente el comando "tac", que genera la inversa de un archivo de texto, es decir, la última línea primero y así sucesivamente.

2. Realice un listado recursivo de los archivos del directorio `HOME` de un usuario y guarde la información en un archivo.

3. Liste, uno a la vez, todos los archivos mayores de 100K en el directorio `HOME` de un usuario. De al usuario la opción de eliminar o comprimir el archivo (usando el comando `read`), luego proceda a mostrar el siguiente. Escriba en un archivo de log los nombres de todos los archivos eliminados y la fecha. Puede utilizar el comando `ls` o `find`.

4. Dada una lista de archivos, escriba un script que basado en el tipo (extensión) de cada uno de ellos (.gz, `.bz2`, .zip, `.tar`), invoque automáticamente el comando apropiado para descomprimirlo (`gunzip`, `bunzip2`, `unzip`, `tar`). Si un archivo no está comprimido, el script debe mostrar un mensaje y continuar con el siguiente archivo.

5. Un directorio contiene archivos cuyos nombres poseen mayúsculas, minúsculas y espacios. Escriba un script que convierta todos los nombres de archivos en minúsculas y los espacios en '_'. Informe cuántos archivos se renombraron. Nota: puede utilizar el comando `tr`.

6. Implemente un script reciba un nombre de directorio, correspondiente al directorio de código fuente de un proyecto Java, y construya los siguientes reportes:

   a) Cantidad de clases/interfaces importadas (diferentes) en cada archivo de código fuente.

   b) Igual al anterior, pero a nivel de un paquete (no recursivo). Si una misma clase es importada desde diferentes paquetes, deberá generar dos entradas diferentes en el reporte.

7. Codifique un script para mantener un log circular de aplicación. Un log en general es un archivo donde una aplicación guarda mensajes de error o advertencia, mientras que un *log circular* funciona como sigue: cada vez que el script es invocado, verifica en un directorio de aplicación (parámetro) dentro de "/var/log" por la existencia de un archivo determinado. Si el tamaño de dicho archivo no sobrepasa un valor T, no hace nada. Si sobrepasa, comprime dicho archivo mediante gzip, y lo borra. Al archivo comprimido se le asigna un prefijo que indica el número de archivos gzip creados en sucesivas invocaciones al script. Cuando la cantidad de archivos es L (tamaño del log circular), se debe también borrar el archivo gzip creado más viejo. Por ejemplo, si las entradas son:

   * /var/log/tomcat5.5/catalina.out (log de la aplicación)

   * /var/log/tomcat5.5/slot#1-catalina.out.gz

   * . . .

   * /var/log/tomcat5.5/slot#[L]-catalina.out.gz

   el próximo archivo gzip a borrar es "slot#1-catalina.out.gz". Implemente luego una variante del script que cree versiones del archivo de log principal siempre y cuando no ocupen en conjunto más de B bytes.

8. Muestre el árbol de directorios a partir de un directorio pasado como parámetro, de acuerdo a las siguientes variantes:

   a) Mostrando sólo subdirectorios

b) Mostrando subdirectorios y archivos.

c) Las dos anteriores, tomando como parámetro un valor entero que indica el nivel máximo para la navegación de un directorio/subdirectorio.

El comando debería mostrar por pantalla el *árbol de directorios* en modo texto, donde el desplazamiento a derecha del nombre de un directorio indica proporcionalmente su profundidad. Por ejemplo, la siguiente salida corresponde al listado del contenido del directorio "/" (los asteriscos indican cada entrada dentro de un subdirectorio)

* boot
  * grub
  * . . .
* dev
* etc
* . . .

Ejercicios de tipo parcial

1. Implemente un script de shell que dada una ruta a un archivo de texto como parámetro, imprima por pantalla todas las palabras y su valor TF asociado. El algoritmo TF (Term Frequency) determina la importancia de cada una de las palabras de un documento de texto. Para esto, la importancia de una palabra *p* se determina contando la cantidad de veces que *p* aparece en el documento y dividiendo por el número de ocurrencias de la palabra que más se repite en el mismo. Por ejemplo, si un documento contiene p_1 (2 ocurrencias), p_2 (1 ocurrencia) y p_3 (3 ocurrencias), entonces:

   - TF(p_1) = 2/3 = 0.66
   - TF(p_2) = 1/3 = 0.33
   - TF(p_3) = 3/3 = 1

2. Escriba un script de shell que calcule el número Fibonacci de un valor recibido como parámetro, de acuerdo a su definición recursiva:

   fib(0)=1

   fib(1)=1

   fib(n)=fib(n-1)+fib(n-2)

   a) El script debe guardar (sin repeticiones) los valores calculados en un archivo intermedio "computed" en el directorio home que asocie un valor individual con su Fibonacci. El script además debe utilizar los valores presentes en dicho archivo para evitar recalcular el Fibonacci de un número ya calculado. No considere tratamiento de errores. Se asume que los parámetros recibidos siempre tienen un valor asociado y del tipo correcto (no negativos).

   b) Modifique el script para que calcule el Fibonacci para una lista de valores recibidos como parámetro bajo las mismas condiciones listadas anteriormente.

3. En el archivo "/etc/cups/printers.conf" se almacena la cantidad de caracteres que puede imprimir cada impresora del sistema dependiendo del tamaño de página que se le asigne. Este archivo posee el siguiente formato:

   <printerid>:<pagfrmt>:<filas>:<cols>

   Por ejemplo, la línea 2:a4:24:80 indicaría que la impresora 2 puede imprimir en una hoja tipo A4 24 filas de 80 caracteres cada una. Escriba un script que dado como parámetro un archivo de texto, un número de impresora y un tipo de página retorne la cantidad de páginas necesarias para imprimir contenido del archivo. Tener en cuenta que si una línea de texto del archivo supera la cantidad de caracteres configurados para el tipo de página de la impresora, se deberá imprimir en la siguiente línea.

4. El comando “ps -A” retorna los procesos activos ejecutando en el sistema en forma de estructura tabular (puede suponer que las columnas se separan por un tab o espacio en blanco):

PID	TIME	CMD
...	...	...
3643	00:31:17	update-manager
3002	00:00:13	update-notifier
3933	00:35:10	bash
...	...	...

La columna PID indica el identificador único de proceso a nivel sistema operativo. TIME indica el tiempo acumulado de uso de CPU en formato hh:mm:ss. Finalmente, CMD indica el nombre del programa ejecutable. Implemente un script que dado un parámetro (string) retorne todas las filas de la tabla correspondientes a los procesos cuyo nombre contiene ese substring y además tienen un uso acumulado de CPU mayor a 1/2 hora. Si el parámetro es vacío, entonces el script debe retornar todos los procesos con uso de CPU acumulado mayor a 1/2 hora. Por ejemplo, para la tabla anterior:

La ejecución con parámetro = “update” debe retornar:

PID	TIME	CMD
3643	00:31:17	update-manager

La ejecución con parámetro = “” debe retornar:

PID	TIME	CMD
3643	00:31:17	update-manager
3933	00:35:10	bash