Práctica 3- Introducción a los sistemas operativos 2022

1) ¿Qué es shell Scripting? ¿A qué tipos de tareas están orientados los scripts? ¿Los scripts deben compilarse? ¿Por qué?
   Shell scripting es la técnica (habilidad / destreza) de diseñar y crear Script (archivo de automatización de tareas) mediante un Shell (preferiblemente) de un Sistema Operativo, o un Editor de Texto (Gráfico o Terminal). Este es un tipo de lenguaje de programación que generalmente es interpretado. Se orienta a la automatización de tareas y a aplicaciones interactivas o con interfaz gráficas. No se compilan ya que son programas interpretados
2) Investigue la funcionalidad de los comandos echo y read
   Echo es un comando para la impresión de un texto en pantalla
   Read lee su entrada estándar y asigna las palabras leídas en la(s) variable(s) cuyo nombre se pasa como argumento
   a) ¿Cómo se indican los comentarios dentro de un script?
      Se indican usando #
   b) ¿Cómo se declaran y se hace referencia a variables dentro de un script?
      Para declararlas haríamos `NOMBRE="pepe".Para` hacer referencia usamos el $, entonces seria `echo $NOMBRE`
3) Hecho en pc
4) Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $*, $? Y $HOME dentro de un script?
   Para acceder a los argumentos usamos $1,$2.. $n, dependiendo de la cantidad de argumentos que pasemos.
   La variable $# contiene la cantidad de argumentos recibidos, $* contiene la lista de todos los argumentos, $HOME es una variable que contiene la ruta /home la cual es la que contiene todos mis documentos
5) ¿Cuál es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cuál es su significado?
   El comando exit es usado para terminar un script. Puede devolver diferentes parámetros donde cada uno tendrá un significado
   - 0 indica que el script se ejecutó se form exitosa
   - Un valor distinto indica un código de error

   Puedo consultar el exit status imprimiendo la variable $?
6) El comando expr permite la evaluación de expresiones. Su sintaxis es: expr arg1 op arg2, donde arg1 y arg2 representan argumentos y op la operación de la expresión. Investigar que tipo de operaciones se pueden utilizar
   - Igualdad: arg1 -eq arg2
   - Desigualdad: arg1 -ne arg2
   - Mayor: arg1 -gt arg2
   - Mayor o igual: arg1 -ge arg2
   - Menor: arg1 -lt arg2
   - Menor o igual: arg1 -le arg2
7) El comando “test expresión” permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [expresión]. Investigar qué tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.
   El comando test permite efectuar una serie de pruebas sobre los archivos, las cadenas de caracteres, los valores aritméticos y el entorno de usuario. El comando test posee dos sintaxis: test expresión y [expresión], donde "expresión" representa el test que se debe efectuar.
   - -b fichero: cierto si el fichero existe y es un dispositivo de bloques

- -c fichero: cierto si el fichero existe y es un dispositivo de caracteres.
- -d fichero: cierto si el fichero existe y es un directorio.
- -e fichero: cierto si el fichero existe.
- -f fichero: cierto si el fichero existe y es un fichero normal.
- -g fichero: cierto si el fichero existe y tiene el bit de grupo activado.
- -k fichero: cierto si el fichero tiene el bit de sticky activado.
- -L fichero: cierto si el fichero existe y es un enlace simbólico.
- -p fichero: cierto si el fichero existe y es una tubería nombrada.
- -r fichero: cierto si el fichero existe y es legible.
- -s fichero: cierto si el fichero existe y su tamaño es mayor que cero.
- -S fichero: cierto si el fichero existe y es un socket.
- -t [df]: cierto si df está abierto en un terminal. Si fd es omitido, se toma 1 (salida
- estándar) por defecto.
- -u fichero: cierto si el fichero existe y tiene el bit de usuario activo.
- -w fichero: cierto si el fichero existe y es escribible.
- -x fichero: cierto si el fichero existe y es ejecutable.
- -O fichero: cierto si el fichero existe y es propiedad del identificador efectivo del usuario.
- -G fichero: cierto si el fichero existe y es propiedad del identificador efectivo del grupo.
- fichero1 -nt fichero2: cierto si fichero1 es más reciente (en base a la fecha de modificación) que fichero2.
- fichero1 -ot fichero2: cierto si fichero1 es más antiguo que fichero2.
- fichero1 -ef fichero2: cierto si fichero1 y fichero2 tienen el mismo número de dispositivo y de nodo-i.
- -z cadena: cierto si la longitud de cadena es cero.
- -n cadena: cierto si la longitud de cadena no es cero.
- cadena1 = cadena2: cierto si las cadenas son iguales.
- cadena1 != cadena2: cierto si las cadenas no son iguales.
- ! expr: cierto si expr es falsa.
- expr1 -a expr2: cierto si expr1 y expr2 son ciertas
- expr1 -o expr2: si expr1 o expr2 son ciertas.
- arg1 OP arg2: OP es uno de los siguientes valores: -eq, -ne, -lt, -le, -gt, o -ge. Estos operadores binarios aritméticos devuelve cierto si arg1 es igual, distinto, menor que, menor o igual que, mayor que, o mayor o igual que arg2, respectivamente. arg1 y arg2 pueden ser enteros positivos, enteros negativos, o la expresión especial -l cadena, la cual significa la longitud de cadena

8) Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:

- **if**

```
if [ condition ]
then
        block
fi
```

- **case**

```
case $variable in
"valor 1")
        block
;;
"valor 2")
        block
;;
```

```
*) #bloque usado en caso que no haya entrado en ninguno
      block
;;
esac
```

- **while**

```
while [ condition ] #mientras se cumpla la condición
do
      block
done
```

- **for**: puedo llegar a tener dos casos

```
#primer caso, estilo C
for ((i=0; i < 10; i++))
do
      block
done
#segundo caso con lista de valores (foreach)
for i in value1 value2 value3 valueN;
do
      block
done
```

- **select**

```
select variable in opcion1 opcion2 opcion3
do
      #en $variable está el valor elegido
      block
done
```

9) ¿Qué acciones realizan las sentencias break y continue dentro de un bucle? ¿Qué parámetros reciben?

   `break`: usado para terminar la ejecución del loop entero después de completar la ejecución de todas las líneas de código anteriores a la sentencia break, una vez llegada a la sentencia break, saldrá del loop y ejecutará el código siguiente fuera del loop. Si indico la sentencia `break n` donde n es un número, n es la cantidad de bucles de los que saldré, por default es 1

   `continue`: es similar a la sentencia break pero lo que hará será salir de la interacción actual en vez de salir del loop entero. Podría servir para cuando un error ocurre pero quiero intentar ejecutar la siguiente iteración del loop. Si indico la sentencia `continue n` donde n es un número, n es la cantidad de loops de los que saldré, por default es 1

10) ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?

    Existen las variables globales y locales. No es fuertemente tipado.

    Para definir arreglos tengo dos formas

```
arreglo_a=() #Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) #Inicializado
```

11) ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?

    Si, puedo definir funciones dentro de un script. Puedo declararlo de dos formas `function nombre {block}` o como `nombre() {block}`.

    Las variables de entorno son heredadas por los procesos hijos. Para exponer una variable global los procesos hijos se usa el comando export

```
        export VARIABLE_GLOBAL="Mi var global"
comando # comando verá entre sus variables de # entorno a VARIABLE_GLOBAL
```

12) Evaluación de expresiones
    a) Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cuál es el mayor de los números leídos
    b) Modificar el script creado en el inciso anterior para que los números sean recibidos como parámetros. El script debe controlar que los dos parámetros sean enviados
    c) Realizar una calculadora que ejecute las 4 operaciones básicas: +, - ,*, %. Esta calculadora debe funcionar recibiendo la operación y los números como parámetros
13) Uso de estructuras de control
    a) Realizar un script que visualice por la pantalla los números del 1 al 100
    b) Realizar un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:
        i) Listar: lista el contenido del directorio actual
        ii) DondeEstoy: muestra el directorio donde me encuentro ubicado
        iii) QuienEsta: muestra los usuarios conectados al sistema
    c) Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso de afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/ directorio cree un directorio con el nombre recibido como parámetro
14) Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parámetro agregándole una CADENA, contemplando las opciones:
    - "-a CADENA": renombra el fichero concatenando CADENA al final del nombre del archivo
    - "-b CADENA": renombra el fichero concatenado CADENA al principio del nombre del archivo
15) Comando cut. El comando cut nos permite procesas las líneas de la entrada que reciba archivo, entrada estándar, resultado de otro comando, etc.) y cortar columnas o campos, siendo posible indicar cuál es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso

PARÁMETRO	ACCIÓN
-b rango/s	Selecciona los bytes indicados en el o los rangos
-c rango/s	Selecciona los caracteres indicados en el o los rangos
-f rango/s	Selecciona los campos indicados en el o los rangos que se encuentran delimitados por el carácter tabulador
-d carater_delimitador	Especifica el carácter delimitador de los campos
-s	Indica que las líneas que no posean delimitador no sean mostradas

    Algunos ejemplos pueden ser
    - Suponiendo que tengo un fichero de texto con el nombre sofia.txt que tiene un contenido, si solo quiero mostrar el cuarto carácter de cada una de las líneas haría `cut -c 4 sofia.txt`
    - Si quiero mostrar los caracteres 1 y 2 lo puedo hacer mediante `cut -c 1,2 sofia.txt`
    - Para mostrar los 2 últimos caracteres de cada línea: `cat sofia.txt | rev | cut -c 1,2`
16) Realizar un script que reciba como parámetro una extensión y haga un reporte con 2 columnas, el nombre de usuario y la cantidad de archivos que posee esa extensión. Se debe guardar el resultado en un archivo llamado reporte.txt

17) Escribir un script que al ejecutarse imprima en pantalla los nombre de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula). Ejemplo, directorio actual:
    ```
    IsO
    pepE
    Maria
    Si ejecuto: ./ejercicio17
    Obtendré como resultado:
    iSo
    PEPe
    mRI
    ```
    Ayuda: Investigar el comando tr
    El comando tr (translate) se usa en Linux principalmente para traducir y eliminar caracteres. Puede usarse para convertir mayúsculas a minúsculas, apretar caracteres repetidos y borrar caracteres.
18) Crear un script que verifique cada 10 segundos si un usuario se ha loqueado en el sistema (el nombre del usuario será pasado por parámetro). Cuando el usuario finalmente se loguee, el programa deberá mostrar el mensaje ”Usuario XXX logueado en el sistema” y salir
19) Escribir un Programa de “Menú de Comandos Amigable con el Usuario” llamado menú, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de cómo proceder deben mostrarse junto con el menú.
    El menú deberá iniciarse y permanecer activo hasta que se seleccione Salir. Por ejemplo:
    MENU DE COMANDOS
    03. Ejercicio 3
    12. Evaluar Expresiones
    13. Probar estructuras de control
    ...
    Ingrese la opción a ejecutar: 03
20) Realice un script que simule el comportamiento de una estructura de PILA e implemente las siguientes funciones aplicables sobre una estructura global definida en el script:
    push: Recibe un parámetro y lo agrega en la pila
    pop: Saca un elemento de la pila
    length: Devuelve la longitud de la pila
    print: Imprime todos elementos de la pila
21) Dentro del mismo script y utilizando las funciones implementadas: agregue 10 elementos a la pila, saque 3 de ellos, imprima la longitud de la cola, luego imprima la totalidad de los elementos que en ella se encuentran.
22) Implemente un script que recorra un arreglo compuesto por números e imprima en pantalla sólo los números pares y que cuente sólo los números impares y los informe en pantalla al finalizar el recorrido.
23) Implemente un script que recorra un arreglo compuesto por números e imprima en pantalla sólo los números pares y que cuente sólo los números impares y los informe en pantalla al finalizar el recorrido
24) Dada la definición de 2 vectores del mismo tamaño y cuyas longitudes no se conocen.
    vector1=(1 .. N)
    vector2=(7 .. N)
    Por ejemplo:
    vector1=(1 80 65 35 2) y vector2=(5 98 3 41 8).
    Complete este script de manera tal de implementar la suma elemento a elemento entre ambos vectores y que la misma sea impresa en pantalla de la siguiente manera:
    La suma de los elementos de la posición 0 de los vectores es 6

La suma de los elementos de la posición 1 de los vectores es 178
...
La suma de los elementos de la posición 4 de los vectores es 10

25) Realice un script que agregue en un arreglo todos los nombres de los usuarios del sistema pertenecientes al grupo “users”. Adicionalmente el script puede recibir como parámetro:
    - “-b n”: retorna el elemento de la posición n del arreglo si el mismo existe. Caso contrario, un mensaje de error
    - “-l”: devuelve la longitud del arreglo
    - “-i:” imprime todos los elementos del arreglo en pantalla
26) Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se reciba uno). Cada parámetro representa la ruta absoluta de un archivo o directorio en el sistema. El script deberá iterar por todos los parámetros recibidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el quinto, etc.), verificar si el archivo o directorio existen en el sistema, imprimiendo en pantalla que tipo de objeto es (archivo o directorio). Además, deberá informar la cantidad de archivos o directorios inexistentes en el sistema.
27) Realice un script que implemente a través de la utilización de funciones las operaciones básicas sobre arreglos:
    inicializar: Crea un arreglo llamado array vacío
    agregar_elem <parametro1>: Agrega al final del arreglo el parámetro recibido
    eliminar_elem <parametro1>: Elimina del arreglo el elemento que se encuentra en la posición recibida como parámetro. Debe validar que se reciba una posición válida
    longitud: Imprime la longitud del arreglo en pantalla
    imprimir: Imprime todos los elementos del arreglo en pantalla
    inicializar_Con_Valores <parametro1><parametro2>: Crea un arreglo con longitud <parametro1>y en todas las posiciones asigna el valor <parametro2>
28) Realice un script que reciba como parámetro el nombre de un directorio. Deberá validar que el mismo exista y de no existir causar la terminación del script con código de error 4. Si el directorio existe deberá contar por separado la cantidad de archivos que en él se encuentran para los cuales el usuario que ejecuta el script tiene permiso de lectura y escritura, e informar dichos valores en pantalla. En caso de encontrar subdirectorios, no deberán procesarse, y tampoco deberán ser tenidos en cuenta para la suma a informar.
29) Implemente un script que agregue a un arreglo todos los archivos del directorio /home cuya terminación sea .doc. Adicionalmente, implemente las siguientes funciones que le permitan acceder a la estructura creada
    - verArchivo <nombre_de_archivo>: imprime el archivo en pantalla si el mismo se encuentra en el arreglo. Caso contrario imprime el mensaje de error “Archivo no encontrado” y devuelve como valor de retorno 5
    - cantidadArchivos: imprime la cantidad de archivos del /home con terminación .doc
    - borrarArchivo <nombre_de_archivo>: consulta al usuario si quiere eliminar el archivo lógicamente. Si el usuario responde SI, elimina el elemento solo del arreglo. Si el usuario responde NO, elimina el archivo del arreglo y también del FileSystem. Debe validar que el archivo exista en el arreglo. En caso de no existir, imprime el mensaje de error “Archivo no encontrado” y devuelve como valor de retorno 0
30) Realice un script que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio “bin” del directorio HOME del usuario actualmente logueado. El script debe imprimir en pantalla los nombres de los que mueve, e indicar cuántos ha movido, o que no ha movido ninguno. Si el directorio “bin” no existe,deberá ser creado.