Administración de Sistemas

Práctica 2

28/09/2021

Marcos Eladio Somoza Corral

21711787

ÍNDICE

1. Ejercicio 1

- ## Análisis del problema

El problema pide una estructura de carpetas determinada. Para ello hay que saber crear carpetas en sh (***mkdir***) y crear carpetas y saber cómo moverse entre los directorios para crear sus subcarpetas correspondientes. Para ello, se ha creado una función que crea una carpeta padre dada y crea tres sub-archivos con el nombre dado.

- ## Codificación del script

```
#!/bin/bash
create()
{
        mkdir -p DP/$1;
        for i in 1 2 3;do
        echo > DP/$1/f$2$i
                done
}

create "DA" "a"
create "DB" "b"
create "DC" "c"

```

- ## Comprobación de funcionamiento

Para asegurarse que funciona correctamente, se ha ido haciendo prueba y error a cada funcionalidad a la hora de programarlo, asegurando que cree la carpeta DP padre primero, que cree sus subcarpetas con los nombres correctos segundo y finalmente que los sub-ficheros estén correctamente nombrados y situados.

2. Ejercicio 2

- ## Análisis del problema

Se pide un script que, con un directorio y una extensión dados, cuente cuantos ficheros de dicha extensión existen en el directorio. En caso de que no se pase ningún directorio, se deberán buscar los ficheros en la carpeta actual. La complejidad reside en diferenciar cuándo se está dando un directorio válido y buscar ficheros por tipo.
Para el directorio válido, se comparará el directorio dado con el ***tag -d***, que retornará true si se trata de un fichero existente y es directorio. Para buscar en base al tipo de archivo, se usará ***find*** y especificará que se buscan archivos de tipo fichero cuyo nombre termine en la extensión dada.

- ## Codificación del script

```
#!/bin/bash
counter=0
incrementer=1
print_files() {
    for i in $(find $1 -type f -name "*.$2"); do
      ((counter+=incrementer))
    done
echo "hay $counter ficheros con la extension $2"
}

if [ -d "$1" ]; then
    echo "Directorio $1 encontrado :)"
    echo "************************* "
    print_files $1 $2

else
    echo "Directorio $1 no encontrado :("
    echo "*****************************"
    print_files . $2

fi

```

- ## Comprobación de funcionamiento

Para comprobar el correcto funcionamiento del script, se han hecho pruebas pasando un directorio inventado y uno vacío como parámetro (cuyo resultado es el script contando los ficheros del directorio actual), y pasando como segundo parámetro extensiones inventadas (donde el script muestra como resultado 0, dado que no se encuentran). Ni siquiera sin pasar ningún parámetro deja de funcionar, puesto que buscará en el directorio actual los archivos que acaben en “**.**”.

3. Ejercicio 3

- ## Análisis del problema

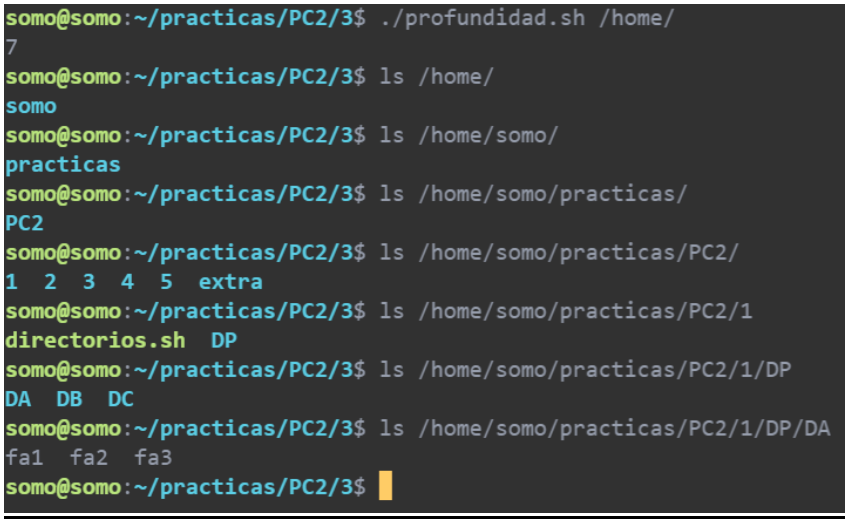
Se pide un script que devuelva el nivel de profundidad máximo de un directorio dado. Para ello, deberá recorrer sus subcarpetas (usando funciones recursivas) y devolver el tamaño de profundidad de la mayor de ellas. Además, igual que en el ejercicio anterior, se deberá comparar si el directorio dado existe y si no, usar el directorio actual.
Para llevar una cuenta de la profundidad se creará una variable ***counter*** y otra ***max***, el ***counter*** llevará el registro de subcarpetas de la llamada actual y el ***max*** el ***counter*** máximo. Para recorrer los directorios recursivamente, se usará una función ***dir_depth*** que se llamará a sí misma con el directorio como parámetro. Recorrerá el directorio dado y comprobará para cada fichero que sea una carpeta, si lo es, llamará de nuevo a la función ***dir_depth*** y se incrementará el ***counter*** (a cada llamada). Cuando detecte al recorrer los ficheros del directorio que ninguno de ellos es una carpeta, comprobará que el ***counter*** actual es mayor que el ***max*** guardado, y si lo es, el ***counter*** actual pasará a ser el ***max***. Después, se restará 1 al ***counter*** y si vale 0 (es decir, si estamos en el directorio inicial) mostrará por consola la profundidad. Si no, volverá al directorio anterior con ***cd ..***

- ## Codificación del script

```
#!/bin/bash
dir_depth(){
    local dir
    cd "$1"
    ((counter++))

    for dir in *
    do
      if [ -d "$dir" ]
      then
        dir_depth "$dir"
      fi
    done
    if ((counter > max))
    then
        max=$counter
    fi

    ((counter--))
    if (( counter == 0 ))
    then
        echo $max
        unset counter
    else
        cd ..
    fi
}
dir_depth $1

```

- ## Comprobación de funcionamiento

Para comprobar el funcionamiento del script, se ha pasado por parámetro ***/home/*** y ha mostrado 7 de profundidad. Después, se ha ido mirando el recorrido que ha hecho el script usando ***ls***, para contar que efectivamente, la profundidad máxima de ***/home/*** es 7.



4. Ejercicio 4

- ## Análisis del problema

Se pide un script que devuelva ***SI*** o ***NO*** si el parámetro pasado coincide con algún usuario conectado. Para ello, se debe primero saber qué usuarios están conectados en el sistema. Este listado línea a línea se almacena en ***/etc/passwd***, junto a mucha otra información para cada usuario. Para coger solamente la primera parte de la línea (correspondiente al nombre de usuario) se puede usar el comando ***cut***, diciéndole que use el tabulador como separador con ***-d*** y que mire en el rango de 1 (el primero elemento) con ***-f 1*** al directorio donde queremos leer, ***/etc/passwd***. Finalmente, se comparará el parámetro dado con cada uno de los nombres resultantes, y cuando se encuentre, cambiaremos la booleana que controla si debe mostrar ***SI*** o ***NO***.

- ## Codificación del script

```
#!/bin/bash
founded_user=false
for name in $(cut -d: -f 1 /etc/passwd); do
if [[ $name == $1 ]]; then
        founded_user=true
    fi
done

if [[ $founded_user == true ]]; then
    echo "SI"
else
    echo "NO"
fi

```

- ## Comprobación de funcionamiento

Para comprobar que el script funciona correctamente, se ha ido probando a cada funcionalidad que funcione correctamente. Para ello, se ha ido probando con los parámetros del comando cut y buscando dónde se almacenan los usuarios. Una vez funcionando con la booleana de control, se ha probado el script sin usar ningún parámetro y usando inventados, donde en ambos casos retorna ***NO***.

5. Ejercicio 5

- ## Análisis del problema

Se pide un menú con cinco casos posibles que se mantendrá ejecutando hasta que el usuario seleccione la opción de salir. Esto indica que se deberá usar un ***switch*** (case en sh) para cada caso, dentro de un ***while*** cuya condición será la condición de salida, llamada cuando el usuario seleccione la opción de salir (5).
Para cada bucle ***while*** se pide mostrar de nuevo las opciones y a fin de mantener un código limpio, se ha creado una función ***print_base_menu*** que se encarga de esto mismo. Por otro lado, dado que se pide para cada opción comprobar que el directorio dado exista (y si no, aplicar la función en el directorio actual) se ha creado una función ***check_dir*** para hacer las comprobaciones de esto y guardar el directorio a aplicar la opción en una variable ***dir***.
Finalmente, cada opción del ***switch*** (menos de salida) llama a su propia función con la lógica correspondiente:

- ***list_content_in_dir()***
  Recorre cada elemento resultante del resultado de hacer ***ls*** al directorio ***$(ls "$1")*** y lo muestra por consola.
- ***delete_files_zero_size()***
  Usando ***find*** busca en el directorio dado ***$1*** los elementos de tipo fichero ***-type f*** cuyo tamaño sea de cero bytes ***-size 0b*** y lo elimina ***-delete***.
- ***delete_files_type_o()***
  Usando ***find*** busca en el directorio dado ***$1*** los elementos de tipo fichero ***-type f*** cuyo nombre acabe con la extensión de objeto ***-name "*.o"*** y lo elimina ***-delete***.
- ***delete_specific_extension()***
  Pide al usuario la extensión que quiera eliminar ***read -r extension*** y usando ***find*** busca en el directorio dado ***$1*** los elementos de tipo fichero ***-type f*** cuyo nombre acabe con la extensión guardada anteriormente ***-name "*.$extension"*** y lo elimina ***-delete***.

- ## Codificación del script

```
#!/bin/bash
delete_specific_extension() {
    echo "Set extension: "
    read -r extension
    find "$1" -type f -name "*.$extension" -delete
}
delete_files_type_o() {
    find "$1" -type f -name "*.o" -delete
}
delete_files_zero_size() {
    find "$1" -type f -size 0b -delete
}
list_content_in_dir() {
    echo "parent: $PWD"
    for item in $(ls "$1"); do
        echo "$item"
    done
}
```

```
print_base_menu() {
    echo "Select an option: "
    echo "1>> Listar contenido en dir dado"
    echo "2>> Eliminar los archivos de tamaño 0"
    echo "3>> Eliminar los archivos de tipo objeto (*.o)"
    echo "4>> Eliminar con una extensión concreta"
    echo "5>> Salir"
}
check_dir() {
    echo "Set directory:"
    read -r readed
    if [ -d "$readed" ]; then
        dir="$readed"
    else
        dir="."
    fi
}

EXIT=false
dir=.
while [ $EXIT = false ]; do
    print_base_menu
    read -r selected
    case $selected in
    "1")
        check_dir
        list_content_in_dir "$dir"
        ;;
    "2")
        check_dir
        delete_files_zero_size "$dir"
        ;;
    "3")
        check_dir
        delete_files_type_o "$dir"
        ;;
    "4")
        check_dir
        delete_specific_extension "$dir"
        ;;
    "5")
        EXIT=true
        ;;
    "*") ;;

    esac
    sleep 1
    clear
done

```

- **Comprobación de funcionamiento**

Para comprobar que este ***script*** funciona correctamente, se ha probado cada una de las opciones de igual forma que se ha hecho con los anteriores scripts. Para las opciones de eliminar, se han creado directorios de prueba con ficheros de diferentes extensiones ***.txt, .o, .sh...*** y tamaños a fin de comprobar que todo funciona correctamente. Cabe destacar que durante el proceso de comprobación, se decidió añadir ***clear*** y ***sleep 1*** en puntos clave a fin de hacer que el script se asemeje más a un menú y menos a comandos escritos en consola.

6. Ejercicio extra

- **Análisis del problema**

Se pide un menú con tres submenús que muestren diferentes datos sobre la máquina en la que corre. Para el menú se ha realizado la misma aproximación que en el anterior ejercicio, así como para los submenús (simplemente replicando la estructura del menú en cada caso del ***switch*** principal). Cabe destacar que se han creado funciones para mostrar los elementos de cada menú, a fin de mantener un código limpio.
Finalmente, cada opción de los submenús (menos de salida) llama a su propia función con la lógica correspondiente:

- ***show_cpu()***
  Mediante el comando ***top*** (usado para mostrar los procesos de Linux), seleccionando el proceso con ***PID*** deseado ***-bn1***. Con ***grep*** buscamos el ***load*** del resultado, ***grep load***, y con ***awk*** lo mostramos, descartando la información no deseada ***awk '{printf "CPU Load: %.2f\n", $(NF-2), usage }'***.
- ***show_memory()***
  Mediante el comando ***free*** podemos obtener la memoria en megabytes ***-m***, con ***awk*** lo mostramos, descartando la información no deseada ***awk 'NR==2{printf "Memory Usage: %s/%sMB (%.2f%%)\n", $3,$2,$3*100/$2 }'***.
- ***list_interfaces()***
  Gracias a ***ifconfig*** podemos ver la lista de interfaces disponibles, para que solo muestre el nombre de las interfaces se seleccionarán sólo los nombres mediante ***cut -d ' ' -f1***. Se guarda el comando entero en una variable y se muestra por pantalla con ***echo***.
- ***assigned_ips()***
  Usando ***ip addr*** se pueden ver todas las conexiones ip actuales, y separar las líneas que contengan ***/inet/*** y con el uso de ***awk '/inet/ {print $2}'*** limpiar toda información no deseada. Con ***grep -v*** se buscarán todas líneas que no contengan ***^::1***, es decir, no mostrará dichas líneas, ***grep -v ^::1***.
- ***show_current_user()***
  Simplemente haciendo ***echo "$USER"*** se puede ver cuál es el usuario actual, guardado en esa variable del sistema, aunque también se podría usar el comando ***whoami***.
- ***show_current_so()***
  Gracias al comando ***uname -o*** es sencillo saber cuál es el sistema operativo donde se esta corriendo el script. Se puede visualizar en consola mostrando el resultado con echo ***$(uname -o).***

- **Codificación del script**

```
#!/bin/bash

print_base_menu() {
    echo "Select an option: "
    echo "A>> Uso de recursos"
    echo "B>> Conectividad"
    echo "C>> Información general"
    echo "D>> Salir"
}
print_A_menu() {
    echo "Select an option: "
    echo "1>> Uso de CPU"
    echo "2>> Uso de Memoria"
    echo "3>> Uso de Disco"
    echo "4>> Volver"
}
show_cpu() {
    top -bn1 | grep load | awk '{printf "CPU Load: %.2f\n", $(NF-
2), usage }'
}
show_memory() {
    free -
m | awk 'NR==2{printf "Memory Usage: %s/%sMB (%.2f%%)\n", $3,$2,$3*100/$2 }
'
}
show_disk() {
    df -h | awk '$NF=="/"{printf "Disk Usage: %d/%dGB (%s)\n", $3,$2,$5}'
}
print_B_menu() {
    echo "Select an option: "
    echo "1>> Listado de interfaces"
    echo "2>> IPs asignadas"
    echo "3>> Volver"
}
list_interfaces() {
    interfaces=$(ifconfig | cut -d ' ' -f1)
    echo $interfaces
}
assigned_ips() {
    ips=$(ip address | awk '/inet/ {print $2}' | grep -v ^::1 | grep -
v ^127)
    echo $ips
}
print_C_menu() {
    echo "Select an option: "
    echo "1>> Usuario actual"
    echo "2>> Sistema operativo actual"
    echo "3>> Volver"
}
```

```
show_curren_user() {
    echo "$USER"
}
show_current_so() {
    echo $(uname -o)
}
EXIT=false
while [ $EXIT = false ]; do
    print_base_menu
    read -r selected
    case $selected in
    "A")
        clear
        RETURN=false
        while [ $RETURN = false ]; do
            print_A_menu
            read -r selected
            case $selected in
            "1") show_cpu ;;
            "2") show_memory ;;
            "3") show_disk ;;
            "4") RETURN=true ;;
            esac
            sleep 1
            clear
        done
        ;;
    "B")
        clear
        RETURN=false
        while [ $RETURN = false ]; do
            print_B_menu
            read -r selected
            case $selected in
            "1") list_interfaces ;;
            "2") assigned_ips ;;
            "3") RETURN=true ;;
            esac
            sleep 1
            clear
        done
        ;;
    "C")
        clear
        RETURN=false
        while [ $RETURN = false ]; do
            print_C_menu
            read -r selected
            case $selected in
            "1") show_curren_user ;;
            "2") show_current_so ;;
```

```
            "3") RETURN=true ;;
            esac
            sleep 1
            clear
        done
        ;;
    "D")
        EXIT=true
        ;;
    "*") ;;

    esac
    sleep 1
    clear
done

```

- ## Comprobación de funcionamiento

A la hora de probar el buen funcionamiento de este script no se ha tenido que hacer demasiado hincapié en la dificultad de los menús, dado que se han realizado de la misma forma que la resuelta en el ejercicio 5. Sin embargo, para extraer la información del sistema se ha debido de investigar mucho para saber de dónde se puede ver y cómo convertir dicha vista a un output legible para el script.
Por supuesto, el movimiento entre menús y el buen funcionamiento de cada opción se ha probado de todas las maneras posibles, moviéndose desde un menú a otro e incluso intentando seleccionar una opción de un submenú en la selección del menú principal, sin éxito (ergo se cumple el buen funcionamiento).