Prácticas

1. Gestión de los archivos

Objetivo: efectuar operaciones básicas en el sistema de archivos.

1. A partir de su directorio personal, cree la estructura siguiente utilizando un único comando:

```
|-----carpeta1
|       |-----carpeta3
|-----carpeta2
|       |-----carpeta4
```

Utilice el comando **mkdir** con el parámetro `-p`:

```
$ mkdir -p carpeta1/carpeta3 carpeta2/carpeta4
```

2. Vaya al directorio carpeta1 con una ruta absoluta y cree el archivo archivo1 en este directorio.

La ruta absoluta sale de la raíz sin ninguna ruta relativa. Teclee:

```
$ cd /home/user/carpeta1
```

Cree el archivo con touch:

```
$ touch archivo1
```

3. Copie archivo1 en el directorio carpeta3 con una ruta relativa.

   La ruta es relativa en función de la ubicación actual: .. sube un nivel, y . define la ubicación corriente.

   ```
   $ pwd
   /home/user/carpeta1
   ```

   Copie simplemente el archivo en la carpeta3 donde está usted ahora:

   ```
   $ cp archivo1 carpeta3
   ```

   o también:

   ```
   $ cp archivo1./carpeta3
   ```

4. Vaya a la carpeta2 utilizando una ruta relativa, y copie el archivo archivo1 de carpeta3 con el nombre archivo2 donde se encuentra usted.

   Para moverse:

   ```
   $ cd ../carpeta2
   ```

   Para copiar el archivo:

   ```
   $ cp ../carpeta1/carpeta3 ./archivo2
   ```

5. Renombre archivo2 como archivo3 y muévalo al directorio carpeta3.

   El comando **mv** mueve y renombra.

```
$ mv archivo2 ../carpeta1/carpeta3/archivo3
```

6. Suprima archivo1 del directorio carpeta3.

   El comando **rm** suprime el archivo.

   ```
   $ rm ../carpeta1/carpeta3/archivo1
   ```

7. Con rmdir suprime carpeta2, luego carpeta1 y todo su contenido. ¿Es posible? ¿Por qué? ¿Cómo conseguirlo?

   No puede suprimir carpeta2 directamente con rmdir, ya que contiene carpeta4 y por lo tanto no está vacío. Tiene que pasar por el comando **rm** con el parámetro `-r`.

   ```
   $ cd
   $ rm -rf carpeta2
   ```

8. ¿Cuál es el objetivo del comando `ls -l [a-z]*. ??[ !0-9]`?

   El archivo empieza con una letra de a a z y termina por cuatro caracteres: el punto, dos caracteres cualesquiera y el último, que es cualquier cosa excepto una cifra. Se muestran los detalles de los archivos que corresponden a este criterio.

9. Cree un archivo llamado «-i» con una redirección: echo > -i. Intente suprimirlo.

   Es un clásico. Si usted intenta suprimir el archivo, rm devuelve un error que indica que falta un parámetro (el nombre del archivo). Interpreta -i como una opción del comando. Así, la pregunta es: ¿cómo conseguir que -i no sea reconocido como una opción?

   Como se considera todo lo que empieza por un guión como un parámetro, debe indicar una ruta que permita aislar el guión:

   ```
   $ rm ./-i
   ```

2. Buscar archivos

Objetivo: buscar archivos con find, whereis y locate.

1. Visualice todos los archivos que tienen un tamaño inferior a 400 bytes y los derechos 644.

   Utilice los parámetros `-size` y `-perm` del comando **find**:

   ```
   $ find / -size -400c -perm 644 -print
   ```

2. Visualice todos los archivos en su directorio personal que tienen un tamaño inferior a 400 bloques.

   ```
   $ find ~ -size -400 -print
   ```

3. Liste en formato largo todos los archivos de su propio sistema que fueron modificados hace más de 7 días.

   Utilice los parámetros `-user` y `-mtime`:

   ```
   $ find / -user seb -mtime +7 -ls
   ```

4. Liste y visualice en formato largo los archivos en su directorio personal que tienen como propietario guest o que tienen un tamaño entre 512 y 1024 bytes, ambas cantidades incluidas.

   La pequeña trampa reside aquí en el «incluidas». Si usted indica +512c, se excluyen los archivos de 512 bytes. Debe modificar los límites en consecuencia.

   ```
   $ find ~ -user guest -size +511c -size -1025c -ls
   ```

5. Busque todos los archivos vacíos del sistema que no pertenecen a root e intente suprimirlos.

   Utilice los parámetros `-empty` y `-exec` para ejecutar un rm en cada archivo encontrado.

   ```
   $ find / -type f -empty -exec rm -f {} \;
   ```

6. Indique dónde se encuentra el comando binario ls.

   Utilice el comando **whereis** para ello:

   ```
   $ whereis -b ls
   ```

7. Ha perdido el archivo letra_importante.odf. ¿Dónde está, sin utilizar find?

   Para contestar hace falta que la base locatedb esté construida con updatedb. Luego, utilice el comando **locate**:

   ```
   $ locate letra_importante.odf
   ```

3. Las redirecciones

Objetivo: trabajar con las redirecciones de canales.

1. El comando **find /** devuelve muchos errores si un simple usuario lo utiliza debido a un problema de permisos. Evite los mensajes de error redireccionándolos hacia un «agujero negro»:

```
$ find / 2>/dev/null
```

2. En el caso anterior y a pesar de los errores, sigue teniendo acceso a muchas ubicaciones y se visualiza una lista muy larga que, por lo tanto, no se puede aprovechar. Coloque esta lista en un archivo llamado resultado.

```
$ find / 1>resultado 2>/dev/null
```

3. Ahora, no se visualiza nada. Finalmente, para saber por qué no puede acceder a determinados directorios, va a obtener también los mensajes de error en el archivo resultado, con la lista de los archivos. Haga una redirección del canal de error estándar en el canal de salida estándar:

```
$ find / >resultado 2>&1
```

4. No se visualiza nada más. Quiere ambas cosas: un archivo y la visualización de los resultados en pantalla. Se utiliza el comando **tee** con una tubería. Permite recuperar un flujo saliente, colocarlo en un archivo y volver a sacar este flujo como si no hubiera pasado nada:

```
$ find / 2>&1 | tee resultado.
```

Su archivo resultado contiene la lista de todos los archivos accesibles, los errores y el conjunto se visualiza también por pantalla.

4. Los filtros y herramientas

Objetivo: trabajar con filtros y herramientas en un archivo clásico.

1. El archivo /etc/passwd es un gran clásico en Unix. Se compone de siete campos separados por «:»: `login:passwd:UID:GID:Comentario:homedir:shell` . Recupere la línea del usuario root en /etc/passwd:

```
$ grep ^root: /etc/passwd
```

2. En esta línea, recupere el UID de root:

```
$ grep ^root: /etc/passwd | cut -d: -f3
```

3. Cuente el número de usuarios que contiene este archivo usando una redirección en entrada:

```
$ wc -l < /etc/passwd
```

4. Un poco más complicado: recupere la lista de los GID, ordénelos por orden creciente y suprima los duplicados:

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq
```

5. De ahí, extrapole el número de grupos diferentes utilizados:

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq | wc -l
```

6. Convierta todos los logins a mayúsculas:

```
$ cut -d: -f1 /etc/passwd | tr "[a-z]" "[A-Z]"
```

7. Aísle ahora la octava línea de /etc/passwd. Hay varias soluciones, veamos las dos siguientes:

```
$ head -8 /etc/passwd | tail -1
```

Y:

```
$ grep -n "" /etc/passwd | grep ^8: | cut -d: -f2-
```

5. Los procesos

Objetivo: gestionar los procesos.

1. Inicie el proceso sleep 1000 en segundo plano. Recupere su PID.

```
$ sleep 1000&
[1] 9168
```

(el PID varía, por supuesto).

2. Vuelva a colocar este proceso en primer plano, luego párelo (no lo mate) y de nuevo mándelo a segundo plano.

```
$ fg
sleep 1000
(CTRL-Z)
[1]+  Stopped                 sleep 1000
$ bg
[1]+ sleep 1000 &
```

3. Indique los detalles de este proceso:

```
$ ps p 9168 -f
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
seb       9168  8096  0 10:46 pts/1    S      0:00 sleep 1000
```

4. Modifique la prioridad de este proceso y paselo a un factor 10:

```
$ renice 10 9168
9168: prioridad anterior 0, nueva prioridad 10
```

5. Liste de nuevo el detalle de este proceso, pero con formato largo. Mire el valor de la columna NI:

```
$  ps p 9168 -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME CMD
0 S  1000  9168  8096  0  90  10 -  2324 restar pts/1    0:00
sleep 1000
```

6. Mande la señal 15 a este proceso. Esto va a terminarlo.

```
$ kill -15 9168
[1]+  Completado          sleep 1000
```

6. Programación de shell Nivel 1

Objetivo: en un archivo de texto tenemos las líneas siguientes:

1 3

5 7

12 19

...

Escriba un script que acepte este archivo como parámetro, que lo lea y que para cada una de sus líneas calcule la suma de dos números y la muestre con la forma siguiente:

1 + 3 = 4

5 + 7 = 12

12 + 19 = 31

...

1. Verifique al principio del script que el número de parámetros pasado al script es igual a 1 y que este parámetro corresponde efectivamente a un archivo.

```
[ $# -ne 1 -o ! -f $1 ] && exit
```

2. Declare el tipo como entero e inicialice una variable a 0, que contendrá el total de cada una de las líneas.

```
typeset -i result
Result=0
```

3. Se debe leer el archivo línea por línea. Escriba un bucle que lea una línea hasta que se haya alcanzado el final del archivo:

```
while read linea
do
...
done < $1
```

4. En el bucle, recupere los dos valores de las líneas, el separador es el espacio. Coloque dos valores en las variables c1 y c2:

```
while read linea
do
   c1=$(echo $linea | cut -d" " -f1)
   c2=$(echo $linea | cut -d" " -f2)
   ...
done < $1
```

5. Sume estos dos valores y coloque el resultado en result. Visualice result.

```
[ $# -ne 1 -o ! -f $1 ] && exit
typeset -i result
result=0
while read linea
do
   c1=$(echo $linea | cut -d" " -f1)
   c2=$(echo $linea | cut -d" " -f2)
   result=$((c1+c2))
   echo -e "$c1 + $c2 = $result"
done < $1
```

6. Guarde el script y hágalo ejecutable. Después, ejecútelo.

```
$ chmod u+x ./script.sh
./script.sh archivo
1 + 2 = 3
3 + 4 = 7
5 + 6 = 11
```

7. Función Shell

Objetivo: un número es primo cuando sus únicos divisores son 1 y él mismo.

Dicho de otro modo, si se puede dividir un número por otra cosa distinta de uno o este mismo número, y el resultado de esta división es un entero (o el resto de esta división es 0, lo que viene a ser lo mismo), entonces no es primo.

La cifra 1 no es primo ya que no tiene dos divisores diferentes.

La cifra 2 es primo (2 x 1: por lo tanto, dos divisores diferentes).

Ningún número par (excepto la cifra 2) es primo (ya que se pueden dividir todos por 2).

Lista de los diez primeros números primos: 2 3 5 7 11 13 17 19 23 29.

Escriba una función llamada **es_primo**, que coja como parámetro un número y que devuelva el código de retorno 0 si el número es primo, o 1 en el caso contrario.

```
es_primo()
{
    [ $1 -lt 2 ] && return 1
    [ $1 -eq 2 ] && return 0
    [ $(expr $1 % 2) -eq 0 ] && return 1
    i=3
    while [ $((i*i)) -le $1 ]
    do
        [ $(expr $1 % $i) -eq 0 ] && return 1
        i=$((i+2))
    done
    return 0
}
```