UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Práctica 1

Repaso general

*El objetivo de esta primera parte de la práctica es repasar los conceptos de **shell scripting** aprendidos en la materia **Introducción a los Sistemas Operativos**. Se realizará un repaso general sobre los comandos más comunes y su uso en **scripts**.*

Preguntas de repaso

1. ¿Qué es la shell? ¿Para qué sirve?
2. ¿En qué espacio (de usuario o de kernel) se ejecuta?
3. Si pensamos en el funcionamiento de una shell básica podríamos detallarlo secuencialmente de la siguiente manera:
   - Esperar a que el usuario **ingrese** un comando
   - **Procesar la entrada** del usuario y obtener el **comando** con sus parámetros
   - Crear un **nuevo proceso** para ejecutar el comando, iniciarlo y esperar que retorne
   - Presentar la **salida** (de haberla) al usuario
   - Volver a empezar.

   Este tipo de comportamiento, típico de las shell interactivas, se conoce como **REPL** (***R**ead-**E**val-**P**rint **L**oop*, Ciclo de leer, interpretar e imprimir).

   Analice cómo implementaría este ciclo básico de interpretación de *scripts*.
4. Investigue la system call `fork`:
   - ¿Qué es lo que realiza?
   - ¿Qué retorna?
   - ¿Para qué podrian servir los valores que retorna?
   - ¿Por qué invocaria a la misma al implementar una shell?
5. Investigue la system call `exec`:
   - ¿Para qué sirve?
   - ¿Comó se comporta?
   - ¿Cuáles son sus diferentes declaraciones POSIX?
6. Investigue la system call `wait`:
   - ¿Para qué sirve?
   - Sin ella, ¿qué sucedería, pensando en la implementación de la shell?

7. En una shell, ¿en qué se diferencian los comandos internos (o built-in) de los externos? ¿Qué función cumple la variable de entorno PATH? Hint: https://www.geeksforgeeks.org/internal-and-external-commands-in-linux/

Scripts

1. Realice un *script* que guarde en el archivo `/tmp/usuarios` los nombres de los usuarios del sistema cuyo *UID* sea mayor a 1000.

2. Implemente un *script* que reciba como parámetro el nombre de un proceso e informe cada 15 segundos cuántas instancias de ese proceso están en ejecución.

3. Desarrolle un *script* que guarde en un arreglo todos los archivos del directorio actual (incluyendo sus subdirectorios) para los cuales el usuario que ejecuta el *script* tiene permisos de **ejecución**. Luego, implemente las siguientes funciones:
   (a) `cantidad`: Imprime la cantidad de archivos que se encontraron
   (b) `archivos`: Imprime los nombres de los archivos encontrados en orden alfabético

4. Se le ha encomendado organizar las fotos (en formato `jpg`) de todos los eventos de los que su empresa ha participado en el último año, los cuales se encuentran organizados en directorios con el nombre del evento. Para facilitar su búsqueda posterior, los archivos deben tener nombres que sigan el siguiente patrón: `EVENTO-N.jpg`, donde:

   - `EVENTO` es el nombre del evento (el del directorio que se está procesando)
   - `N` es un índice de foto, comenzando en `1`

   Realice un *script* que renombre los archivos de cada subdirectorio del directorio actual siguiendo lo especificado en el párrafo anterior.

   Ejemplo: dada la siguiente estructura de archivos y directorios:

```
bashconf15/
  DSC01050.jpg
  DSC01051.jpg
  DSC01052.jpg
  DSC01053.jpg
  DSC01054.jpg
jsconf-14/
  DSC01230.jpg
  DSC01231.jpg
  DSC01232.jpg
  DSC01235.jpg
  DSC01236.jpg
oktoberfest-14/
  DSC02229.jpg
  DSC02230.jpg
  DSC02231.jpg
  DSC02232.jpg
```

   Se desea terminar con la siguiente estructura luego de ejecutar su *script*:

```
bashconf15/
  bashconf15-1.jpg
  bashconf15-2.jpg
  bashconf15-3.jpg
  bashconf15-4.jpg
```

```
  bashconf15-5.jpg
jsconf-14/
  jsconf-14-1.jpg
  jsconf-14-2.jpg
  jsconf-14-3.jpg
  jsconf-14-4.jpg
  jsconf-14-5.jpg
oktoberfest-14/
  oktoberfest-14-1.jpg
  oktoberfest-14-2.jpg
  oktoberfest-14-3.jpg
  oktoberfest-14-4.jpg
```

5. Escriba un *script* que liste en orden alfabético inverso el contenido del directorio actual. Es decir, si el contenido son los archivos:

   ```
   archivo_1.txt articulo.doc directorio directorio_2 script.sh
   ```

   se espera que el *script* los liste de la siguiente manera:

   ```
   script.sh directorio_2 directorio articulo.doc archivo_1.txt
   ```

6. Realice un *script* que **copie** todos los archivos del directorio home del usuario que lo ejecuta, a un subdirectorio del mismo llamado `backup` cambiándoles el nombre para que esté en **mayúsculas**. **No se deben procesar los subdirectorios del home del usuario**, únicamente los archivos ubicados directamente en este. Si el directorio `backup` existe al iniciar el script, el contenido del mismo debe borrarse antes de copiar los archivos.

   Ejemplo: si el home del usuario actual contiene:

   ```
   /
     home/
       mi_usuario/
         so/
           practica1.pdf
           ejercicios/
             ejercicio-1.sh
             ejercicio-2.sh
         archivo1.txt
         mi-script.sh
         otro_archivo.txt
   ```

   se espera tener lo siguiente luego de la ejecución del *script*:

   ```
   /
     home/
       mi_usuario/
         backup/
           ARCHIVO1.TXT
           MI-SCRIPT.SH
           OTRO_ARCHIVO.TXT
         so/
           practica1.pdf
           ejercicios/
             ejercicio-1.sh
             ejercicio-2.sh
         archivo1.txt
         mi-script.sh
         otro_archivo.txt
   ```

7. Un escritor tiene organizados los capítulos de su próximo libro en distintos archivos de texto plano en un mismo directorio, y le ha solicitado ayuda para **concatenar** el contenido de cada uno de ellos en un único archivo final llamado `libro.txt`, de modo tal que éste último contenga el texto de todos los otros archivos, uno luego del otro. Puede asumir que los archivos de los capítulos tienen nombres alfabéticamente ordenados: `capitulo-01.txt`, `capitulo-02.txt`, ..., `capitulo-48.txt`, por ejemplo.

   Tip: `man cat`.

:

Redes y Sistemas Operativos

*En esta sección se aprenderá cómo integrar conceptos del sistema operativo, como redirecciones y **pipes**, con una red TCP/IP. Para ello se utilizará principalmente la herramienta `netcat`, también conocida como `nc`. Investigue su funcionamiento y resuelva. **Tip:** `man nc`*

1. La herramienta `netcat` provee una forma sencilla de establecer una conexión TCP/IP. En una terminal levante una sesión de `netcat` en modo servidor, que escuche en la IP 127.0.0.1 (localhost) en un puerto a elección. En otra terminal conéctese, también vía `netcat`, al servidor recién levantado. Interactúe y experimente con ambas terminales.

2. `netcat` también es bueno al momento de transmitir archivos sobre una red TCP/IP. Utilizando dos terminales como se hizo en el ejercicio anterior, transmita el archivo `/etc/passwd` desde una sesión de `netcat` hacia la otra.

   Tip: recordar *pipes* y redirecciones.

3. Desarrolle un script que reciba en su entrada estándar una lista de *hosts* e imprima en su salida estándar únicamente aquellos que tienen el puerto 80 abierto. Cuando un *host* no tiene el puerto 80 abierto, `netcat` tardará varios segundos en determinar que la conexión no se puede establecer.

   Tip: utilizar la opción `-w` de `netcat` para disminuir el tiempo de *timeout*.

4. Desarrolle un *script* que reciba en su entrada estándar una lista de *hosts* con el puerto 80 abierto y, para cada uno, realice un requerimiento HTTP GET a la URI raíz y devuelva el valor del campo `Content-Length` de la respuesta. Deberá ser posible utilizar como entrada estándar la salida estándar del *script* anterior.

```
echo www.google.com www.debian.org www.linti.unlp.edu.ar | ./cl.sh
www.google.com: 262
www.debian.org: 470
www.linti.unlp.edu.ar: 223
```

   Tip: `printf "GET / HTTP/1.0\r\n\r\n"| ...`

5. Interprete y describa qué es lo que hace el siguiente fragmento de código extraído de la *man page* de `netcat`.

```
$ rm -f /tmp/f; mkfifo /tmp/f
$ cat /tmp/f | /bin/sh -i 2>&1 | nc -l 127.0.0.1 1234 > /tmp/f
```

   Tip: `man mkfifo`

6. Desarrolle un *script* que permita al usuario chatear con otra instancia del **mismo** *script*. Para ello, el *script* deberá recibir como parámetro si va a funcionar como **(c)**liente o como **(s)**ervidor. También deberá recibir como parámetro un *nickname* para el usuario. Por ejemplo, para invocar el *script* en modo servidor con el *nick* **jvg**, debería ejecutar:

```
./nc-chat.sh s jvg
```

Además, los mensajes transmitidos deben comenzar con la fecha, hora y *nick* del emisor. Por ejemplo:

```
Thu Mar 12 13:03:14 ART 2015, jvg says:
Knock knock Neo.
```

Subshells

1. ¿Qué es una subshell?
2. Indique en cuáles situaciones se puede generar una subshell
3. Los comandos que forman parte de un pipe, ¿ejecutan en secuencialmente o en parelelo?
4. Las variables de una subshell, ¿pueden ser accedidas por la shell *padre*?
5. Compare el resultado de los siguientes agrupamientos de comandos (ejecutar los comando desde el home directory del usuario)

```
CATEDRA=so
( cd /tmp; echo $CATEDRA > subshell; CATEDRA=so1 )
echo CATEDRA
{ cd /tmp; echo $CATEDRA > subshell1; CATEDRA=so2; }
echo CATEDRA
```

   - En el primer agrupamiento de comandos, ¿qué valor queda en el archivo subshell? Al terminar el comando, ¿se modificó el valor de la varible CATEDRA? Jusiticar
   - En el segundo agrupamiento de comandos, ¿qué valor queda en el archivo subshell1? Al terminar el comando, ¿se modificó el valor de la varible CATEDRA? Justificar
   - Además de lo visto en los puntos anterior, ¿notó alguna otra diferencia?

   Hint: https://www.tldp.org/LDP/abs/html/subshells.html