TRABAJO PRACTICO Nº 2:
BAASH

ALUMNOS:
Sosa Ludueña, Diego Matias.
Choquevilca Zotar, Gustavo Braian.

MATERIA: Sistema Operativo I

OPERACIONES BASICAS: Respuestas

1. ***¿Cómo ejecutamos un archivo que se llama exactamente como un builtin?***
   Sabiendo que built-in son comandos internos que interpreta la shell, supongamos que se ejecuta un archivo llamado *exit* igual que el comando interno *exit*, por lo tanto, para ejecutar dicho archivo primero debemos especificar la ruta (camino absoluto) y luego ejecutarlo. Por ejemplo:

   diego@diego-Satellite-L515:~/Documentos/SO1/TP2/EJEMPLO$ ./exit
   o también puede ser:
   diego@diego-Satellite-L515:~$ Documentos/SO1/TP2/EJEMPLO/./exit

2. ***¿Por qué las recomendaciones de seguridad indican que es peligroso tener ./ en PATH al más puro estilo DOS?***
   Si un usuario crea en un directorio un fichero shell script llamado *ls*, donde se tiene permisos de escritura. Luego, si otro usuario entra en ese directorio y teclea *ls* al tener en la variable PATH los directorios *(.:/home/diego/bin:/home/diego/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin)* buscaría primero en ese directorio el programa *ls* y ejecutaría el shell script creado, por lo tanto, este shell script podría realizar una ejecución inesperada para el usuario y luego llamar al verdadero *ls (/bin/ls)*, el usuario habría ejecutado un programa que no es el *(/bin/ls)* y no se enteraria.

3. ***Supongamos que existen 2 comandos posibles dentro de la secuencia que contiene PATH, donde el primero en la secuencia no está marcado como ejecutable y el segundo si. ¿Qué hace el intérprete bash, ejecuta el segundo o informa que el primero no tiene permiso de ejecución?***
   El intérprete bash, informa que no tiene permiso de ejecución y no realiza el siguiente comando.

4. ***Indique qué sucede cuando se tiene un directorio en el pwd actual con el mismo nombre que un archivo ejecutable en el PATH. ¿Dice que no puede ejecutar un directorio o ejecuta el comando que encontró?***
   lo que realiza el intérprete bash es ejecutar el comando que encontró.

5. ***Obtengan la lógica más sencilla que unifique los tres tipos de búsqueda.***
   Si el comando no es un built-in, el shell deberá buscar un archivo dentro de su sistema de archivos, cargarlo y ejecutarlo pasándole los argumentos. El problema principal es dónde encontrar este archivo. Existen tres formas:
   Camino absoluto
   Camino relativo
   Búsqueda en secuencia PATH
   Cuando el comando comienza con /, éste se toma como un camino absoluto dentro del árbol del filesystem, el shell lo cargará en memoria y ejecutará el comando. En

cambio si el comando comienza con el nombre de un directorio, o ..., se debe seguir las reglas usuales de camino relativo de UNIX, cargar y ejecutar el archivo comando, relativo al camino actual.
Otro mecanismo entra en juego cuando el comando no comienza con un delimitador de camino absoluto o relativo. La variable de entorno PATH, que puede ser leída con el comando env o con echo $PATH, sirve de secuencia de caminos absolutos o relativos, separados por ':' que serán prefijados al comando hasta encontrar un archivo que pueda ser leído y ejecutado.

6. ***¿Podemos poner directorios relativos en PATH?***
   Si se puede y ya fue especificado en uno de los puntos anteriores.

PROCESOS EN SEGUNDO PLANO: Respuestas

1. ***Investiguen cuáles son los comandos internos para manejo de procesos en background de bash.***
   Los scripts son algo indispensable en Linux y en algunas ocasiones puede ser necesario ejecutarlo en segundo plano (o background), ya sea porque tarde mucho en finalizar o porque el programa tiene que ejecutarse de forma indefinida y al mismo tiempo se quieren analizar sus entradas/salidas en tiempo real, o cuando, en el caso de conexiones remotas, por el motivo que sea, se pueda producir una desconexión.

   & y bg: Si añadimos un ampersand (&) al final del comando o del script que queremos ejecutar, éste se ejecutará en segundo plano.
   ./script.sh &

   Si ejecutamos un comando de la forma habitual (en primer plano o foreground) y, después de pasado un cierto tiempo, nos damos cuenta que hubiera sido mejor ejecutarlo en segundo plano o background, lo podemos hacer sin necesidad de tener que matar el proceso y volver a ejecutarlo de nuevo con el ampersand. Esto se hace presioando “CTRL+Z” para suspender la ejecución del comando actual, y después invocando el comando bg:
   ./script.sh

   pulsamos “CTRL+Z” y se mostrará que el comando ha sido pasado a segundo plano:
   bg

   Este proceso que se acaba de pasar al segundo plano, también se puede volver al primer plano con el comando fg y si hubiese varios procesos, añadiremos el número de proceso.
   fg 2

   nohup y &: Al finalizar una sesión en un terminal se envía un señal (SIGHUP) a todos los procesos que esté ejecutando nuestro usuario. Como resultado, dichos procesos se mueren (aunque les hayamos puesto & al final)

Para evitar esto utilizamos el comando nohup. Este comando hace que un proceso ignore la señal SIGHUP, y redirige la salida de nuestro script a un archivo nohup.out que es creado en el directorio actual.
nohup ./script.sh &

Una buena práctica sería redireccionar stdin, stdout y stderr. Básicamente, por dos razones: rastrear la salida de nuestro script en caso de producirse algún error, y evitar problemas al terminar nuestra sesión ssh, si es que la ejecutamos en un servidor remoto.
nohup ./script.sh > foo.out 2> foo.err < /dev/null &

Kill: detiene la ejecución de procesos.

Jobs: muestra los procesos que están corriendo.

2. ***En el ejemplo de arriba el operador '&' funciona como operador de composición paralela. Hay un operador de composición secuencial en Bourne shell. ¿Cuál es?***
   La composición secuencial de comandos es con el símbolo &&. cmd1 && cmd2 .. etc. Hasta falle alguno. Otra posibilidad para ejecutar comandos en secuencia es separándolos con el símbolo || . cmd1 || cmd2 || ..|| ejecuta los comandos especificados hasta que uno de ellos no falle.

3. ***Investiguen cómo bash forma el árbol de procesos cuando ejecutamos cmd1 & cmd2 & cmd3 & ... & cmdN. Piensen la respuesta y luego utilicen pstree para corroborarla.***
   Cuando ejecutamos cmd1 & cmd2 & cmd3 & ... & cmdN todos estos procesos se ejecutarán en paralelo.

4. ***Indique cuántas letras 'a' debería imprimir el siguiente programa:***
   El programa imprime 8 letras “a”.

REDIRECCIÓN DE LA ENTRADA/SALIDA: Respuestas

1. **Las corrientes estándar stdout y stderr están dirigidas ambas a la consola. ¿Cómo podemos utilizar los operadores de redirección para separarlas?**
   Handle Name Description
   0 stdin Standard input
   1 stdout Standard output
   2 stderr Standard error
   Ejemplo:
   command 2> archivo1
   Ejecuta el comando1, dirigiendo el flujo de error estándar que archivo1.
   Básicamente funcionaria de la siguiente manera:
   STDOUT representa el flujo de salida de comandos, ficheros... etc.

STDERR representa el flujo de salida de errores. Básicamente, STDERR es lo mismo que STDOUT, la diferencia es que contiene posibles errores que el programa reporta por pantalla al usuario. En el siguiente ejemplo vamos a utilizar un error producido por "cat" (lee ficheros) y lo vamos a guardar en un fichero utilizando ">" (STDOUT)

```
1 $cat mi_fichero.txt > mi_log.log
```

En este caso, mi_fichero.txt no existe. Dado que ">" es utilizado sólo para STDOUT, el fichero mi_log.log será creado, pero está vacío.
En cambio, podremos ver en la consola el error devuelto por STDERR:1 cat: mi_fichero.txt: No such file or directory
Si queremos capturar este mensaje de error en un fichero mi_log.err tendremos que añadir el número "2″ a ">" (STDERR):
1 cat mi_fichero.txt 2> mi_log.err
Aquí le estamos indicando a Bash que capture el descriptor número 2 (STDERR) y lo envié al fichero. Recordemos que STDIN es 0 y STDOUT es 1. De igual manera se puede mezclar STDOUT y STDERR. La utilidad de esto la encontramos cuando queremos ejecutar algún programa de forma automática (utilizando cron, por ejemplo) o desde un script y deseamos capturar toda la información posible en algún fichero al que podamos acceder posteriormente.Entonces, el tema es bastante sencillo. Vamos a hacer un "tail" (lee el final de un fichero) de dos ficheros "/var/log/dmesg" y "mi_fichero.txt" y todas las salidas vamos a enviarlas a dos ficheros mi_log.out y mi_log.err.
1 $tail /var/log/dmesg mi_fichero.txt >mi_log.out 2>&1mi_log.err
dmesg sabemos que existe, por lo tanto las últimas líneas devueltas por "tail" serán capturadas por el STDOUT al fichero mi_log.out. En cambio el error devuelto al no encontrar el fichero mi_log.txt será enviado por STDERR a mi_log.err:
1 tail: cannot open mi_fichero.txt for reading: No such file or directory
También podemos enviar las dos salidas a un único fichero e incluso encauzarlas como STDIN para otro programa.
Para capturar las dos salidas vamos a utilizar "2>&1″ lo cual vamos a ver en muchos scripts.
1 $tail /var/log/dmesg mi_fichero.txt > mi_log.log 2>&1
La mejor manera de silenciar a un programa para que no perturbe la ejecución de nuestro script es enviar todas las salidas a /dev/null.
1 $tail /var/log/dmesg mi_fichero.txt > /dev/null 2>&1
Si nuestro script necesita ejecutar otro programa y necesita tan sólo de su "exit status", el usuario que ejecuta el script no verá que programas estamos utilizando y no recibirá los posibles errores que de otra manera aparecerían en pantalla.

2. **De más ejemplos de cómo usar la redirección de entrada. Tenga cuidado que muchos programas deshabilitan la posibilidad de redirigir la entrada estándar a otra cosa que no sea una terminal.**
   a) *$ cat < /dev/stdin.*
   b) *# sort < items.*

TUBERÍAS (PIPE): Respuestas

1. **Los pipes existen en el filesystem mientras dura el comando. ¿Dónde se encuentran y qué atributos tiene?**
   Para implementar el operador de tubería 'sc0 | sc1' del shell, se debería utilizar la syscall pipe() y conectar de manera apropiada la salida estándar del sc0 al pipe_fd[1] y la entrada estándar del sc1 al pipe_fd[0].