

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

LABORATORIO: Sistemas Operativos

Práctica 6

Comandos del Shell, variables, redirectores y pipes

LosDos

Integrantes:

Amanda Velasco Gallardo - 154415
Carlos Octavio Ordaz Bernal - 158525

Fecha(s) de elaboración de la práctica:

28 de febrero de 2019

Introducción

El **shell** es un intérprete de comandos. Es el encargado de validar las instrucciones introducidas por el usuario y cuenta con variables de ambiente que permiten personalizar el entorno de trabajo. Una característica que distingue a los **shell** del resto de los intérpretes es que éste es un lenguaje de programación que cuenta con instrucciones para la ejecución selectiva y repetitiva de grupos de comandos. Cada programa escrito en un **shell** se conoce como guión (*script*).

Las variables de **shell** son aquellas que permiten personalizar el ambiente de trabajo. Existe un conjunto de variables predefinidas y el usuario es capaz de definir sus propias variables. Existen dos tipos de variables de **shell**: las booleanas y las que pueden recibir un valor. Además, existe otro tipo de variables conocidas como variables de entorno o globales que son heredadas a cualquier **subshell** creado. Si una variable de entorno y una variable de **shell** tienen el mismo nombre, la de mayor precedencia es la segunda.

Hay situaciones en las que, al trabajar dentro de la terminal, es preferible o necesario guardar la salida de un comando ejecutado en un fichero. Dentro de los ambientes **UNIX**, existen los redirectores, los cuales nos permiten mandar la salida en terminal a un archivo (por lo general de texto) dentro de cualquier directorio.

Para redirigir las salidas se utiliza el símbolo `>`. Si se utiliza una vez, guarda el comando que lo precede en el fichero especificado. Si éste no existe lo crea; si ya existe el fichero lo sobre escribe. Si el símbolo es utilizado de la siguiente manera `>>`, añade la salida del comando al fichero. Si no existe lo crea, y si existe, lo añade al final.

Existen ocasiones en las que deseamos realizar varias tareas al mismo tiempo, sin embargo puede ser necesario tener más de un programa para lograr el resultado esperado. Una herramienta para ejecutar de manera secuencial programas dentro de la terminal es el **pipe** (`|`), el cual nos permite encadenar la ejecución de programas pasando la salida de uno como entrada de otro. El orden en que se ejecutan los programas es de izquierda a derecha y de esa manera es como se pasan las salidas de uno a otro.

Es posible encadenar desde dos hasta una cantidad enésima de programas siempre y cuando se hayan tomando en consideración los flujos de información entre los programas. Además, es posible juntar con un **pipe** programas sin importar en qué lenguaje están escritos; es decir, su uso no se limita a los desarrollados en ambientes **UNIX**.

Desarrollo

1. En una primera terminal, que se titulará UNO, aplicar el comando `ps -el`. En una segunda terminal, que se titulará DOS, ejecutar el comando `ps -l` u `sisops`. Explicar cuál es la diferencia entre ambos comandos `ps`. Además, mostrar cuántos procesos despliega cada comando. También mostrar los comandos con los que obtuvo la cuenta de los procesos.

Con `ps -u usuario` se muestran los procesos cuyo nombre de usuario o ID de usuario corresponde a *usuario*. Entonces, en la terminal UNO se muestran todos los procesos (debido al `-e`) mientras que en la DOS aparecen solamente los procesos del usuario `sisops`. En ambas terminales, el listado se muestra en formato largo (por el `-l`).

En la terminal UNO hay 145 procesos y en la DOS hay solamente 62. La cuenta se obtuvo escribiendo los comandos correspondientes en cada terminal seguidos de `| wc`.

2. Si el punto (`.`), en lugar de estar al principio del `path` estuviera al final, ¿podría afectar el que se ejecute alguno de sus programas ejecutables que se encuentran en el directorio de trabajo? Pensar en un posible escenario en que pudiera no ejecutarse. Explique.

Cuando el shell busca dentro del `path`, lo hace de izquierda a derecha, ejecutando la primera coincidencia que encuentre. Entonces, si el punto se encuentra al final, el directorio de trabajo tendrá la menor prioridad, por lo que se buscará dentro de él en última instancia. Esto podría evitar que se ejecute un programa del directorio actual si existe otro con el mismo nombre en otra ubicación con mayor prioridad dentro del `path`.

3. ¿Cuál es la utilidad del comando `kill`?

Este comando envía una señal a un proceso. Por default, la señal que se manda es `TERM` (para terminar un proceso).

4. Desplegar la lista de las señales con el comando `kill -l`. ¿Cómo se llama la señal 9 y cuál efecto causará sobre el proceso que la recibe?

La siguiente imagen muestra la lista de señales del comando `kill -l`.

```
ubuntu:~> kill -l
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT
CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR SYS
RTMIN RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX
```

Fig. 1: Despliegue del comando `kill -l`.

La señal 9 es KILL y asegura que el proceso que la reciba en verdad sea terminado aunque se encuentre congelado o no responsivo.

5. Dentro de la ayuda de `kill`, ¿cuál es la acción por default de las señales 2, 9 y 15?

La señal 2 corresponde a INT e interrumpe un proceso. La señal 9 corresponde a KILL y fuerza la terminación de un proceso. Por último, la señal 15 corresponde a TERM y termina un proceso.

6. Desplegar el contenido del archivo *CiInfi.cc*. ¿Cuál es el objetivo de las funciones `signal` y `Fin`? En una terminal compilar y ejecutar el programa. Ahora, donde está corriendo el programa, teclear la combinación de teclas Ctrl+D. ¿Qué pasa? ¿Qué imprimió el programa? ¿Qué señal generó Ctrl+C hacia el programa?

El contenido del archivo se muestra en la figura siguiente.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void Fin(int nse)
{
    printf("\nADIÓS, No. de la señal recibida: %d\n\n", nse);
    exit(0);
}

int main()
{
    signal(SIGTERM, Fin);
    signal(SIGINT, Fin);

    while(1==1)
        printf("Aqui sigo\n");

    return 0;
}
```

Fig. 2: Contenido del archivo *CiInfi.cc*

La función **Fin** recibe el identificador (int) de una señal, imprime un mensaje para avisar la salida junto con el identificador de la señal que la causó, y termina el programa (exit). La función **signal** recibe el identificador de una señal y una función manejadora que se ejecutará cuando se reciba dicha señal; en este caso se asocia la recepción de las señales SIGTERM y SIGFIN a la función **Fin**. Mientras se corre el programa, al teclear **Ctrl+C** observamos que termina la ejecución del programa, imprimiendo antes un texto de salida que indica el número de la señal que lo causó. **Ctrl+C** genera una señal INT.

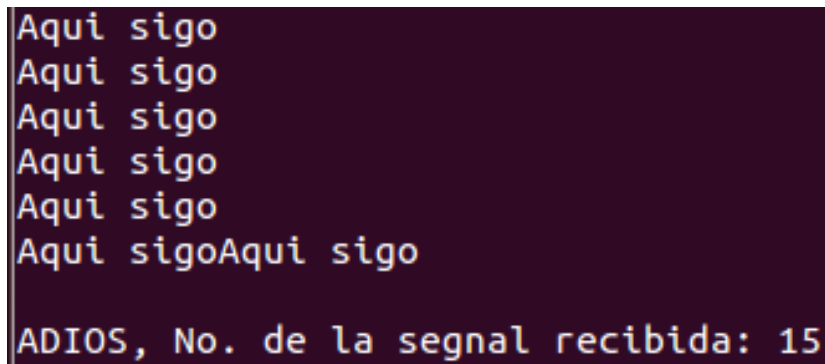
7. Mantener las dos terminales. En una de ellas dejar ejecutando *CiInfi.exe*. Desde la otra terminal obtener tanto el PID como el PPID de *CiInfi.exe*. Indicar cuál es su proceso padre. Para detener a *CiInfi.exe*, desde la otra terminal ejecutar el comando `kill -2 PID` o `kill -15 PID`.

La siguiente imagen muestra los procesos observados desde la segunda terminal.

```
0 S 1000 2928 2898 0 80 0 - 1439 rt_sig pts/2 00:00:00 tcsh
1 S 0 2940 2 0 80 0 - 0 worker ? 00:00:00 kworker/0:2
0 R 1000 2942 2906 3 80 0 - 502 - pts/1 00:00:00 CiInfi.exe
0 R 1000 2943 2928 0 80 0 - 1177 - pts/2 00:00:00 ps
ubuntu:~/uci> kill -15 2942
```

Fig. 3: Listado de procesos.

Se puede observar que el PID de *CiInfi.exe* es 2942 y el PID de su su padre, tcsh, es 2928. Al ejecutar el comando `kill -15 2942`, ocurre lo que se muestra en la siguiente figura.



```
Aqui sigo
Aqui sigo
Aqui sigo
Aqui sigo
Aqui sigo
Aqui sigoAqui sigo
ADIOS, No. de la se al recibida: 15
```

Fig. 4: Ejecuci3n y terminaci3n de *CiInfi.exe*.

La ejecuci3n del programa termina tras recibir la se al TERM.

9. Usando el archivo *Desordenado.cc*, combinar los comandos que se conocen mediante separadores, redirectores y pipe, para que en una sola l nea de comando se realice:

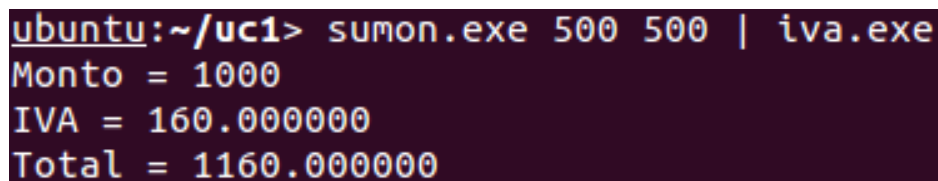
- Generar un archivo con los n meros ordenados
`Desordenado.exe | sort > ordenado.txt`
- Desplegar cu ntos n meros se generaron
`Desordenado.exe | wc`
- Generar un archivo con los primeros 10 n meros ordenados
`Desordenado.exe | sort | head > ordenDiez.txt`
- Generar un archivo con los  ltimos 10 n meros
`Desordenado.exe | sort | tail > invOrdenDiez.txt`

10. Combinar en una sola l nea los comandos necesarios para crear un archivo que contenga las l neas 7 a 15 a partir de lo generado por *Desordenado.exe* al ejecutarse.

`Desordenado.exe | sort | head -15 | tail -9 > lineas.txt`

11. Utilizando los programas *sumon.cc* y *iva.cc*, realizar lo siguiente:

- Compilar y ejecutar *sumon.cc*
`g++ sumon.cc -o sumon.exe;sumon.exe`
- Compilar y ejecutar *iva.cc*
`g++ iva.cc -o iva.exe;iva.exe`
- En una sola línea de comando hacer que el resultado de *sumon.cc* sea pasado a *iva.cc*
`sumon.exe 500 500 | iva.exe`



```
ubuntu:~/uc1> sumon.exe 500 500 | iva.exe
Monto = 1000
IVA = 160.000000
Total = 1160.000000
```

Fig. 5: Resultado del comando anterior.

Conclusiones

Esta práctica nos sirvió para conocer las diferentes formas con las que el **shell** cuenta para para enviar señales a los procesos así como redirigir y encadenar sus entradas o salidas. Las señales nos sirvieron, por ejemplo, para mandar mensajes a un programa que se encuentra en un ciclo infinito. Por el otro lado, aprender a utilizar los redirectores y el pipe fue de especial importancia para combinar distintos comandos en una sola línea y dirigiendo la salida hacia la localidad deseada, permitiéndonos saltar pasos innecesarios o tener que crear archivos temporales. Finalmente, comprendimos el uso de **pipe** para encadenar la ejecución de programas al pasar la salida de un programa que suma dos números, como entrada de otro programa que calcula el impuesto al valor agregado de un número.

Referencias

- Ríos, J. (2019). Notas del curso de Sistemas Operativos. Recuperado el 19 de febrero de 2019, del sitio web: Comunidad ITAM.
- Redirecciones y Pipes. *ADSL Ayuda*. Recuperado el 02 de marzo de 2019, del sitio web: <https://www.adslayuda.com/linux-redirecciones.html>