Práctica 1: Programación en SHELL

Moisés García Villanueva

20 de Agosto de 2012

1 Línea de comandos

Interfaz de Línea de Comandos (CLI), por su acrónimo en inglés de Command Line Interface (CLI), es un método que permite a las personas dar instrucciones a algún programa informático por medio de una línea de texto simple. Se señala que los conceptos de CLI, Shell y Emulador de Terminal no son lo mismo, aunque suelen utilizarse como sinónimos.

1.1 shell

Shell que significa en Castellano "concha" es el interprete de comandos del sistema. Es una interfaz de texto de altas prestaciones, que sirve fundamentalmente para tres cosas: administrar el sistema operativo, lanzar aplicaciones (e interactuar con ellas) y como entorno de programación.

1.2 Emulador de Terminal

Un emulador de terminal es un programa informático que simula el funcionamiento de un terminal de computadora en cualquier dispositivo de visualización.

Incorporan características tales como control de procesos, redirección de entrada/salida, listado y lectura de ficheros, protección, comunicaciones y un lenguaje de órdenes para escribir programas por lotes o (scripts o guiones). Uno de los lenguajes o intérpretes más conocidos, es el **Bourne Shell**, el cual fue el intérprete usado en las primeras versiones de Unix y se convirtió en un estándar.

2 El lenguaje de programación shell

2.1 Tipos de shell

Las diferentes distribuciones de Linux incorporan gran variedad de terminales. Los tipos principales son los siguientes:

- Shell Bourne (sh). Creado por S. Bourne, es el más utilizado en la actualidad. Su símbolo del sistema es $. Es el shell estándar y el que se monta en casi todos los sistemas UNIX/Linux.
- C-Shell (csh). Procedente del sistema BSD, proporciona funciones tales como control de trabajos, historial de órdenes, etc. Ofrece importantes características para los programadores que trabajan en lenguaje C. Su símbolo del sistema es
- Shell job (jsh). Incorpora algunas características de control al shell estándar del sistema.
- Shell Korn (ksh). Escrito por David Korn, amplía el shell del sistema añadiendo historial de órdenes, edición en línea de órdenes y características ampliadas de programación.
- Bourne Again Shell (bash). Fue creado para usarlo en el proyecto GNU. BASH, por lo tanto, es un shell o intérprete de comandos GNU que incorpora la mayoría de distribuciones de Linux. Es compatible con el shell sh. Además, incorpora algunas características útiles de ksh y csh, y otras propias como la edición de línea de comandos, tamaño ilimitado del historial de comandos, control de los trabajos y procesos, funciones y alias, cálculos aritméticos con números enteros, etc. Su símbolo del sistema es nombre_usuario@nombre_equipo.

2.2 Procesos

Un proceso se define como un programa que se transforma en proceso en el momento en que este se ejecuta y está en memoria. Además del nombre que el proceso recibe, que es el nombre del programa que esta corriendo, recibe también un número identificativo llamado PID (process ID, o ID de proceso). Si ejecutamos el comando `ps` veremos los procesos que están ejecutando en este momento con nuestro UID, es decir que estamos corriendo nosotros mismos.

2.3 Como ejecutar un script

Podemos ejecutar un script de las dos formas siguiente:

- Utilizando el interprete de comandos directamente:

```
lc96$ sh archivo_script.sh
```

- Asignando permisos de ejecución al script

```
lc96$ chmod ugo+x archivo_script.sh
lc96$ ./archivo_script.sh
```

2.4 El shell como interprete de ordenes

Primero recordemos algunas de las ordenes que podemos ejecutar en el shell.

1. tail
2. expr
3. echo
4. man
5. ls
6. cd
7. mkdir
8. rm
9. cp
10. mv
11. pwd
12. cat
13. more
14. less
15. head
16. cal
17. clear
18. date
19. tar
20. gzip

El alumno debe escribir que hace cada una de las ordenes o comandos anteriores.

2.4.1 Entrada y salida estándar

El sistema al iniciarse, carga 3 archivos en memoria, estos son el stdin, el stdout y stderr. El intérprete de comandos configurará estos archivos para que apunten al teclado, en el caso del stdin, y al monitor, en el caso del stdout y stderr. Existen numerosos comandos de Unix que utilizan la entrada estándar para tomar sus datos y la salida estándar para volcarlos a la pantalla. La configuración esta dada de este modo ya que el shell espera que se ingresen los datos por el teclado y que su resultado o algún error se reflejen en la pantalla. Un ejemplo será el caso del comando `cat`. Si usáramos el comando `cat`, sin ningún argumento, todo lo que ingresemos se vería repetido en la pantalla.

2.4.2 Tuberías

Existe la posibilidad de desviar la salida o la entrada a un comando para poder realizar funciones complejas u obtener los datos que saldrían por la pantalla directamente a otro archivo. Lo primero se denomina canalización, ver figura 1, y lo segundo redirección. El símbolo que se utiliza para efectuar la canalización es el denominado pipe o símbolo de canalización (|). Este símbolo permite que se pase la salida de unacomando o programa a la entrada de otro. Un ejemplo clásico de lautilización de este símbolo es cuando se requiere listar un directorioque ocupa más de una pantalla. Se podrá entonces utilizar el comandoclásico para listar ls y enviar su salida a otroprograma que lo muestre de a una página de pantalla por vez, por ejemplo el more.

```
[lc69@alumnofie:~]$ ls -l | more
```

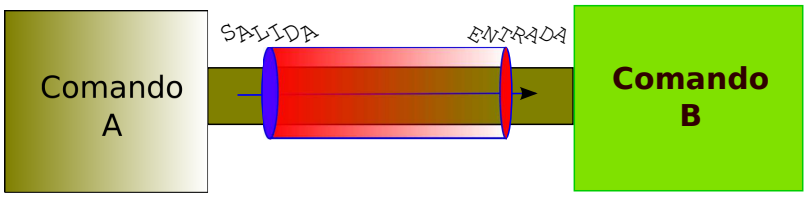

Figure 1: Proceso de canalización o entubamiento. La salida producida por el comando A llega como datos de entraqda al comando B.

Esto servirá para realizar lo antedicho ya que la salida del comando `ls` será canalizada para que sea la entrada del comando more y este se encargara de mostrar los datos por pantalla. Lo que se esta utilizando es la canalización de la entrada estándar y la salida estándar. La salida estándar del comando `ls` es redireccionada hacia la entrada estándar del comando more que es cambiada para que también apunte a la canalización. Otros dos símbolos utilizados son el <y el >. Lo que hacen es redirigir tanto la salida como la entrada estándar de o hacia un archivo. Por ejemplo, supongamos que necesitamos guardar en un archivo llamado listado la salida del comando ls.

```
[lc69@alumnofie:~]$ ls > listado
```

Con esto le indicamos al comando ls que redireccione la salida estándar hacia un archivo de nombre listado. En caso de que el archivo exista, será reemplazado con la nueva información. Para agregar contenido a un archivo, inmediatamente después del contenido que posea, se tendrá que poner el símbolo de redirección dos veces

```
[lc69@alumnofie:~]$ ls >> listado
```

3 Ejercicios

1. Hacer un script que imprima en la pantalla los datos personales.
2. Hacer un script que reciba dos números y nos muestre en la pantalla el resultado de las operaciones aritméticas: suma, resta, multiplicación y división.
3. Hacer un script que construya el directorio de la figura 2.

```
Dirtemp
├── apuntes
│   ├── Lab_Paradigmas_Programacion
│   │   └── practica1_shell.pdf
│   ├── Lab_Programacion_Computadoras
│   │   └── manual_LPC.pdf
│   └── LHC
│       ├── EERLHC.pdf
│       ├── PPLHC.pdf
│       ├── SPLHC.pdf
│       └── scfie.fie.umich.mx
│           └── %7Ea9700127a
└── comunicacion2.jpg
```

Figure 2: Árbol de archivos y directorios.

4. Realizar las siguientes actividades en la línea de comandos de Linux y señalar que hace cada una de las instrucciones indicadas:
   (a) Presionar las teclas `Alt + F2` y escribir en el campo de texto: `gnome-terminal`
   (b) Escribir en la terminal: `ls -l /sbin > listado_sbin.txt`
   (c) ¿Qué hace la siguiente secuencia de ordenes? `sort -r -k 5 -n listado_sbin.txt |head -n 20`
   (d) ¿Qué hace la siguiente secuencia de ordenes? `sort -r -k 9 listado_sbin.txt |head -n 15`
   (e) ¿Qué hace la siguiente secuencia de ordenes? `grep ^lrw listado_sbin.txt |sort -r -k 5 |head`