Programando con el intérprete de órdenes (shell) de UNIX

Fundamentos de Informática II.
Práctica 1

17 de marzo de 2006

Resumen

En esta primera práctica nos familiarizaremos con la orden **echo** y las variables, especialmente las variables locales.

Con el fin de facilitar la lectura y comprensión de los guiones de prácticas seguiremos el siguiente convenio que se presenta a continuación.

- Las palabras reservadas de UNIX se presentarán en **negrita** y deben escribirse tal y como se presentan.
- La escritura en letra *cursiva* está reservada para variables y constantes. Es decir, las palabras escritas de esta manera deben sustituirse por las variables o constantes que se consideren convenientes.
- Las palabras escritas en letras MAYÚSCULAS están reservadas para las variable de entorno que el intérprete predefine por defecto u omisión ($default$ en inglés). Estas variables se analizarán brevemente en este documento.
- Siempre que aparezca el símbolo **$** al principio de la línea se trata del **indicador** (*prompt* en inglés) de la línea de órdenes del intérprete y no debe escribirse como parte de las órdenes. De aquí en delante se tomarán las siguientes convenciones para la descripción de las órdenes del intérprete de UNIX.

1. Introducción

El intérprete de órdenes (*shell* en inglés) es un programa que se carga y ejecuta automáticamente cuando se inicia una sesión UNIX. Su función principal es servir de interlocutor entre el núcleo del sistema operativo y el usuario: se encarga de leer cada orden que se introduce por el teclado (o por algún fichero), analiza lo que se ha solicitado y procede a su ejecución si no existe ningún error de sintaxis.

La forma en la que el intérprete (shell) interactúa con el sistema operativo UNIX se muestra en la siguiente figura. El núcleo (kernel) es el corazón del sistema operativo y reside en la memoria desde que se enciende el sistema hasta que se apaga. Esta es la parte del sistema operativo que se comunica directamente con la circuitería (hardware) del ordenador.

<table>
<tr><td colspan="2">Usuario</td></tr>
<tr><td colspan="2">Intérprete de órdenes (shell)</td></tr>
<tr><td>Módulos residentes del sistema operativo</td><td>Programas (utilidades) residentes en disco</td></tr>
<tr><td colspan="2">Núcleo (kernel) del sistema operativo</td></tr>
<tr><td colspan="2">Circuitería (hardware) del sistema</td></tr>
</table>

Además del núcleo (kernel), existen otros módulos que también residen en memoria y que sirven para realizar operaciones como el control de entrada/salida, gestión de memoria, gestión de ficheros, gestión del tiempo de procesamiento, etc. El resto del sistema operativo está compuesto de programas (utilidades) que residen en disco y solo se cargan en memoria cuando se les utiliza.

El propio intérprete (shell), como se dijo antes, es un programa que se carga y ejecuta automáticamente cuando se conecta al sistema; cuando el intérprete está listo para recibir órdenes despliega un indicador (por ejemplo, $) en la línea de órdenes. Solo unas pocas de las órdenes que se introducen son ejecutadas

por el intérprete; la gran mayoría de las veces el intérprete (shell) examina cada orden y llama al programa (utilidad) de UNIX que realiza la tarea solicitada. Esto ocurre, por ejemplo, con la orden **ls**.

Sin embargo, el intérprete (shell) también tiene algunas ordenes propias; estas órdenes pertenecen al intérprete (shell) y se reconocen y ejecutan internamente. Algunas de estas órdenes son **cd** o **exit**. El sistema estándar de UNIX está compuesto por una gran cantidad de programas (utilidades) y uno de ellos es el propio intérprete.

"Intérprete de órdenes (shell).[es] un término genérico que se usa para designar al programa (utilidad) que hace de interfaz entre el usuario y el sistema operativo, sin embargo, existen, tanto para UNIX como LINUX, varios programas que pueden realizar esta función. Los intérpretes más conocidos para UNIX son 'Bourne shell (sh)', 'C shell (csh)' y 'Korn shell (ksh)', mientras que para LINUX se tienen los intérpretes 'Bourne Again shell (bash)', 'T shell (tsh)' y 'Z shell (zsh)'.

Existen diferencias de sintaxis y de enfoque entre los diferentes interpretes (shell) de órdenes, sin embargo, su propósito es el mismo: servir de enlace entre el usuario y el sistema operativo. El intérprete (shell) que se utilizará en este documento (y en las prácticas) es el "Bourne Again shell (bash)"de Linux, el cual está basado en el "Bourne shell (sh)"de UNIX.

2. Guiones o procesos del intérprete (shell scripts)

El intérprete de órdenes, además de interpretar y gestionar las órdenes escritas en la línea de órdenes, funciona como un lenguaje de programación y permite crear ficheros de órdenes. Estos ficheros son conocidos como **guiones** o **procesos** (*shell scripts*, en inglés) y pueden contener elementos tales como

- Órdenes UNIX
- Variables
- Órdenes o estructuras de programación propias del intérprete.

Existen tres formas de ejecutar un guión (script) o proceso con el intérprete:

1. Escribiendo en la línea de órdenes **sh** *nombre_del_guión*
2. Activando el permiso de ejecución del fichero que contiene el guión (script) para luego ejecutarlo desde la línea de órdenes.
3. Escribiendo en la línea de órdenes . *nombre_del_guión* (punto, espacio y el nombre del proceso)

Las dos primeras formas son equivalentes y se caracterizan porque al realizar la ejecución se abre automáticamente un nuevo proceso del intérprete de órdenes (shell) que nace y muere con el proceso ejecutado. Esto no ocurre si se ejecuta de la tercera forma, ya que al hacerlo así el proceso (script) se ejecuta desde el mismo intérprete que se carga automáticamente al iniciar la sesión de trabajo, es decir, desde el que se invocó la ejecución del proceso.

PRACTICA: Usando el editor de texto **vi** genere un fichero llamado prueba.sh que contenga el siguiente guión (script).

```
echo Dentro del script
exit
echo Fuera del script
```

Salga del editor y ejecute el guión usando las tres formas mencionadas anteriormente; observe los efectos de cada método de ejecución.

---

3. Desplegando información: la orden echo

Con mucha frecuencia es necesario desplegar información textual en la pantalla de la terminal; para este fin se puede utilizar la orden **echo**. Si no se le da ningún argumento, la orden **echo** produce una línea en blanco, mientras que si se le proporcionan argumentos imprime el valor de éstos seguido de una nueva línea. La forma en la que **echo** despliega los mensajes en la terminal se puede ajustar usando los caracteres de escape siguientes precedidos por la opción **-e**:

Carácter de escape	**Significado**
\n	Nueva línea
\t	Tabulador
\b	Retroceso
\r	Retroceder a inicio línea
\c	Inhibir nueva línea

PRÁCTICA: Usando el editor de texto **vi** genere un fichero llamado ecos.sh que contenga el siguiente guión (script). Salga del editor, active el permiso de ejecución del fichero y ejecútelo.

```
echo Hola, esto es una prueba.
echo Hola,      esto      es      una      prueba.
echo "Hola,     esto      es      una      prueba."
echo -e Hola, "\n.esto es una prueba.
echo -e Hola, "\t.esto es una prueba.
echo -e Hola, esto es una prueba. "\c"
```

3.1. Anulando el significado de los metacaracteres (comodines) en echo

En algunas ocasiones se requiere inhibir el significado de caracteres tales como <, >, *, y ? que tienen un significado especial para UNIX y el intérprete de órdenes. A este proceso de inhibición se le conoce como *escape* y se puede realizar usando los siguientes caracteres de escape:

Barra inclinada a la izquierda \: La barra inclinada a la izquierda [\] se utiliza para indicar que el carácter que le sigue debe interpretarse como un carácter alfanumérico ordinario. Por ejemplo, ? es un carácter de sustitución de archivo y tiene un significado especial para el intérprete, sin embargo, \? es un signo de interrogación y no un comodín. Así, para borrar un fichero de nombre temp? se puede utilizar la orden **rm** temp\?

Comillas dobles ": Los caracteres especiales que se encuentran inmersos en una cadena delimitada por un par de comillas dobles pierden su significado especial, a excepción del signo de dólar [$], las comillas dobles y las comillas simples. Para anular el significado de caracteres se les puede anteponer la barra inclinada a la izquierda. Las comillas dobles también conservan los espacios indicados por el espacio en blanco, el tabulador y la nueva línea, tal como se vio en el último ejemplo.

Comillas simples ': Las comillas simples funcionan de manera similar a las comillas dobles en el sentido de que cualquier carácter especial entre comillas simples pierde su significado especial, a excepción de él mismo. Para anular su significado especial se utiliza \'. No se debe confundir a las comillas simples ['] con el acento grave [`], que es un carácter especial que se verá más adelante.

PRÁCTICA: Usando el editor de texto **vi** genere un fichero llamado especiales.sh que contenga el siguiente guión (script). Salga del editor, active el permiso de ejecución del fichero y ejecútelo.

```
echo \¡\¿\"\' \$ \? \& \— \\
echo *
echo "*"
echo "\.El Sistema UNIX\""
echo '< > "$ ? & |'
echo >
echo ">"
```

4. Variables

Existen dos tipos de variables:

- Las variables de entorno
- Las variables locales (argumentos para los procedimientos shell)

4.1. Variables de entorno

Estas variables también se conocen como *variables estándar* y tienen nombres que son conocidos por el sistema. Son definidas por el administrador del sistema y controlan funciones esenciales del mismo. Por ejemplo, la variable estándar PS1 asigna la cadena de caracteres utilizada como signo del indicador. El signo indicador principal del bash, cargado por defecto, es el signo de dólar ($).

4.2. Variables locales

Asignación de valor: `nombre de variable=valor`

El valor asignado puede recuperarse precediendo el nombre de la variable con el signo $. En caso de que la variable esté inmersa dentro de una cadena de caracteres existen dos posibilidades para su identificación:

1. Encerrar el nombre de la variable entre llaves: $ {nombre de la variable}
2. Encerrar la variable entre comillas: '$nombre de la variable'.

Estas variables pueden crearse y usarse, tanto dentro de un proceso shell, como en el modo interactivo desde el prompt del sistema.

```
fruta=pera
comida=sopa
echo $fruta $comida
pera sopa
```

Una variable sólo tendrá valor dentro del proceso en el cual fue creada.
Para observar el valor de todas las variables definidas en un determinado instante, hay que ejecutar la orden **set**.

Asignaciones especiales

- El valor asignado a una variable puede intervenir en otra variable.

```
preposición=para
objeto=${preposición}caidas
echo $objeto
paracaidas
```

- Cuando en el valor asignado existan varias palabras separadas por espacios, hay que usar comillas para preservar estos espacios en la definición de la variable:

s="**ls** -l"
$s /practicas
Se mostrará en la pantalla el listado largo del contenido del directorio /practicas

- Se puede asignar como valor de una variable la salida de una orden UNIX. En este caso, después del signo igual se encierra la orden UNIX entre los caracteres "tilde francesa"(‘).

```
hoy=‘date‘
echo $hoy
obtendremos la fecha actual
```

4.3. Argumentos para los guiones (shell scripts)

Los argumentos que se añaden a la derecha del guión, cuando éste es ejecutado, se asignan a unas variables que se pueden referenciar dentro del mismo como:

$0 Nombre del guión ejecutado.

$* Conjunto de todos los argumentos (en una sola variable).

$# Número de argumentos pasados al guión.

$n enésimo argumento.

PRÁCTICA: Generar un guión (shell script) que muestre en la salida estándar lo siguiente

El nombre de este proceso es: *nombre* y tiene *número* argumentos que son: *argumentos*

donde *nombre* es el nombre del guión, *número* es el número de argumentos pasados al guión y *argumentos* son los argumentos colocados después del nombre del guión al ejecutarlo.
