Configuración del Entorno y Ejecución de Órdenes

Cómo configuar el entorno de ejecución de un shell y los modos de ejecución de un shell.

Tabla de Contenidos

1 Introducción
2 Variables de Entorno
3 Órdenes Internas
4 Ejecución de Órdenes
5 Modificación de la variable de entorno PATH
6 Iniciación
7 Ejercicios propuestos

Autor: Lina García Cabrera & Francisco Martínez del Río

1 Introducción

En este tema se van a tratar dos aspectos de forma breve. El primero es la configuración del entorno mediante el empleo de **variables de entorno** y **ficheros de iniciación**. El segundo es el estudio del método que sigue UNIX para localizar los ficheros ejecutables en el sistema de ficheros.

2 Variables de Entorno

Todo proceso está asociado a un **conjunto de variables de entorno**. Los procesos pueden consultar o modificar estas variables siendo heredadas por sus subprocesos. Un ejemplo de variable de entorno es HOME, que guarda el directorio base del usuario que ejecutó el proceso. Se pueden ver los valores vigentes de las variables de entorno ejecutando la orden env sin parámetros. También puede consultar el valor individual de una variable de entorno utilizando una sustitución:

```
$ echo $HOME
/home/dofer
```

Al iniciar un proceso como hijo de otro, UNIX asigna a las variables de entorno del proceso hijo los valores de las del padre. A partir de ese momento las variables de entorno del padre y del hijo son independientes, ya que un proceso no puede examinar o modificar las variables de entorno de otro. Cuando un proceso termina desaparecen las variables de entorno, y se pierden los cambios efectuados. El siguiente programa muestra sus variables de entorno, las cuales se pueden acceder en un programa C a

través de la variable global environ, que es un array de punteros a carácter:

```
1 #include <stdio.h>
2 extern char **environ;
3 int main (int argc, char *argv[])
4 {
5 int i = 0;
6 while (environ[i] != NULL)
7 puts(environ[i++]);
8 }
```

La orden **export** del **bash** permite cambiar el valor de una variable de entorno. Su sintaxis es:

$ export variable=nuevo_valor

> Es importante no dejar espacios alrededor del operador de asignacion “=”.

También es posible exportar variables locales:

$ variable=nuevo_valor
$ export variable

Para eliminar una variable, local o de entorno se debe utilizar el comando **unset**:

$ unset variable

3 Órdenes Internas

Normalmente una orden tiene asociado un programa en código máquina almacenado en un fichero. Cuando se teclea una orden a un shell éste ejecuta su programa asociado como un hijo (recuerde el shell simplificado del tema 1). La figura 1 esquematiza estas acciones.

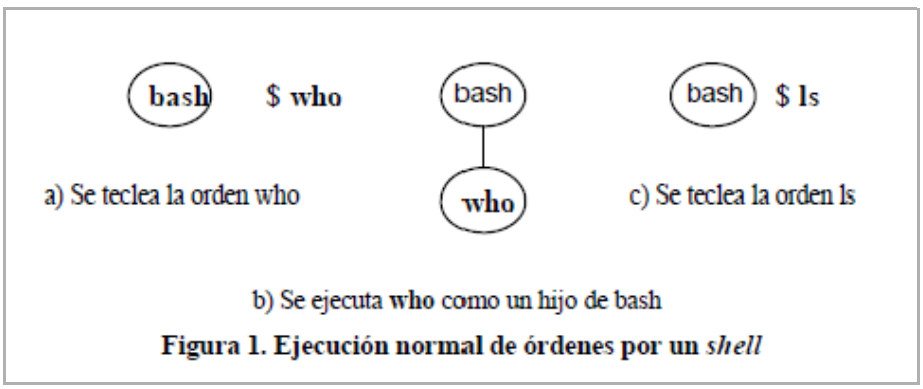


Figura 1. Ejecución normal de órdenes por un *shell*

Sin embargo, existen ciertas órdenes que no tienen asociado un fichero ejecutable en disco. Estas **órdenes se llaman internas**, o, a veces, intrínsecas. Son implementadas y ejecutadas por el propio shell.

Cada shell, por tanto, tiene su propio repertorio de órdenes internas que

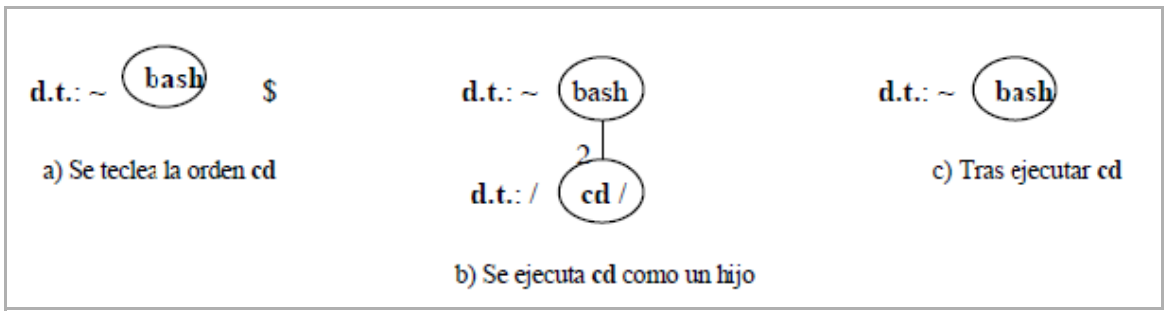
implementa y es capaz de ejecutar. Por cada orden interna que implementa un shell el tamaño de su ejecutable aumenta. Puesto que casi todos los usuarios ejecutan un shell, es deseable que su tamaño sea pequeño para que no ocupen mucho espacio en memoria principal. Por ello en UNIX se prefiere que las órdenes se implanten en ficheros ejecutables en lugar de que sean internas. No obstante, existen una serie de órdenes que sólo pueden ser implementadas por los shells.

Como ejemplo de una orden de este tipo analizaremos en el siguiente párrafo a la orden cd, que, como sabemos, sirve para cambiar el directorio de trabajo de nuestro shell.

En UNIX todo proceso tiene asociado un **directorio de trabajo**. De hecho, el directorio de trabajo forma parte de la información del bloque de control de un proceso. El directorio de trabajo inicial de un proceso es el que tuviera su padre en el momento en que creó al hijo. A partir de entonces padre e hijo siguen caminos independientes.

Cada uno puede modificar su directorio de trabajo mediante una llamada al sistema sin que afecte al directorio de trabajo del otro. Como se indicó arriba la orden **cd** sirve para cambiar el directorio de trabajo de nuestro shell. Esto significa que sirve para cambiar el directorio de trabajo inicial de cualquier orden que ejecutemos en nuestro shell. Esto es así porque **cuando ejecutamos una orden en un shell o bien se ejecuta como un hijo del shell, en cuyo caso hereda el directorio de trabajo del shell, o bien es ejecutada por el propio shell**. En este punto la pregunta es: ¿tiene que ser **cd** una orden interna? Y la respuesta es sí.

Para ilustrarlo analicemos la figura 2 en la que se representa la ejecución de **cd** como una orden asociada a un fichero ejecutable. En la figura el directorio de trabajo del shell es el directorio base del usuario que ejecutó el shell (figura 2.a). En 2.b) el shell ejecuta la orden cd como un hijo, el programa **cd** cambia el directorio de trabajo a /, sin embargo este cambio no afecta a bash pues padre e hijo modifican de forma independiente su directorio de trabajo. En 2.c) cd ya ha terminado y **bash** no ha cambiado de directorio de trabajo.



La orden **umask** presenta unas características similares a **cd**. La máscara de usuario también es una propiedad asociada a los procesos, su valor inicial se hereda del proceso padre y se cambia de forma independiente a éste mediante una llamada al sistema. Por lo tanto, umask debe implementarse como una orden interna. El siguiente ejemplo ilustra todo esto:

```
$ cd / ; pwd          Ejecuto cd en mi shell de trabajo
/
$ (cd; pwd)           Ejecuto cd en un subshell de mi shell de trabajo
```

/home/x1111111

$ pwd *Ejecuto pwd desde mi shell de trabajo*

/ *En mi shell de trabajo el directorio actual sigue siendo /*

$ cd *Ejecuto cd en mi shell de trabajo*

$ pwd *Ejecuto pwd desde mi shell de trabajo*

/home/x1111111 *En mi shell de trabajo el directorio actual ahora es mi directorio base*

La primera línea de órdenes ejecuta `cd /` en nuestro shell de trabajo, es decir, en el shell en que estamos ejecutando normalmente las órdenes. Esto hace que se cambie el directorio de trabajo de nuestro shell a `/`. Después se ejecuta la orden no interna `pwd` que muestra su directorio de trabajo (el que hereda del shell que la ejecuta).

La segunda línea de órdenes incluye dos órdenes encerradas entre paréntesis. Cuando **bash** observa una secuencia encerrada entre **paréntesis ejecuta la secuencia en un subshell**. Esto hace que la orden `cd` se ejecute en un subshell, por lo tanto, se cambia el directorio de trabajo en el subshell, y no en "nuestro" shell. A continuación `pwd` se ejecuta como un hijo del subshell, por lo que hereda como directorio de trabajo el directorio base del usuario. La siguiente orden muestra que la orden `cd` ejecutada en el subshell no ha afectado a nuestro shell.

Para terminar con esta sección indicamos un pequeño truco para saber si una orden no es interna. Para ello lo único que hay que hacer es buscar si existe un fichero con su nombre. Por ejemplo, ¿es la orden `pwd` interna?

$ (find / -name pwd –print >salidaEstandar) >& errorEstandar

$ cat salidaEstandar

/bin/pwd

/usr/share/zsh/help/pwd

$ /bin/pwd

/home/x1111111

En la orden `find` se redirecciona la salida estándar y el error estándar a distintos ficheros. Esto se hace así porque `find` en su búsqueda va a topar con muchos directorios en los que tiene negado el permiso de acceso. Por cada uno de estos directorios manda un mensaje al error estándar lo que hace que sea difícil observar la salida estándar del programa. Se observa que existen dos enlaces con nombre `pwd`. Puesto que uno cuelga de **/bin** es muy posible que sea el ejecutable que buscamos, para comprobarlo lo ejecutamos y descubrimos que efectivamente lo es.

4 Ejecución de Órdenes

bash, al igual que **sh**, tiene tres métodos para ejecutar órdenes: **ejecución directa**, **ejecución directa en un subshell**, y **ejecución indirecta en un**

subshell:

- En la **ejecución directa**, **bash** ejecuta cada orden conforme la detecta. Toda modificación que ocasione la orden en el estado de **bash**, como un cambio en el valor de una variable de shell, permanece hasta la terminación de **bash**. Las órdenes internas de **bash** siempre se ejecutan de forma directa, lo mismo que las órdenes contenidas en un archivo utilizado por la orden `source`.
- En la **ejecución directa en un subshell**, **bash** crea un subshell en el que se ejecutan las órdenes. El subshell hereda una copia de todas las variables del shell padre. Todo cambio de estado en el subshell, en particular las asignaciones a las variables de shell, no afectan al shell padre. El shell **bash** ejecuta en un subshell las órdenes entre paréntesis, y los procesos en segundo plano.
- En la **ejecución indirecta en un subshell**, **bash** crea un subshell igual que lo haría para la ejecución directa, pero llama a la rutina de sistema `exec` para que ejecute la orden en dicho subshell. El subshell hereda las variables de entorno, pero no las de shell (locales). Si la orden cambia alguna de las variables de entorno, estos cambios sólo afectan a las copias y no a los originales. Las órdenes no internas de csh, incluidos los guiones de shell y las órdenes implantadas como programas compilados se gestionan por ejecución indirecta en un subshell y se buscan de la manera que se describe a continuación.

Búsqueda de órdenes. Cuando **bash** detecta una orden que **no es interna**, ya sea un programa compilado o un guión, lo busca en la secuencia de directorios especificada por la variable de shell **PATH**. El shell **bash** obtiene el valor inicial de path de la variable de entorno **PATH**, que obtiene del entorno en el que se hace la llamada a bash. La variable path tiene una sintaxis ligeramente distinta a la de PATH, pero, en esencia, el significado es el mismo.

El **bash** acelera la búsqueda de órdenes mediante una tabla de dispersión (tabla hash). Esta tabla registra los nombres de los archivos ejecutables que existen en los directorios indicados en path. Esta tabla hash es mantenida automáticamente por el shell. Sólo en caso de que cambiemos la ubicación de un ejecutable que ya estaba previamente indexado (o sea o movemos de un directorio a otro del **PATH**) es necesario rehacer la tabla con el comando `hash –r`.

5 Modificación de la variable de entorno PATH

Vamos a seguir un ejemplo de modificación de la variable de entorno **PATH**. Tras seguir los distintos ejemplos de las prácticas disponemos de un buen número de ficheros ejecutables.

Sería interesante crear un directorio en el que almacenar esos ficheros e indicarle a UNIX que cuando tecleemos un nombre de orden también la busque en dicho directorio. Hagámoslo:

$ mkdir ~/bin *Creamos un directorio que cuelga de nuestro directorio base*

$ echo echo hola > ~/bin/eje *Creamos un fichero que almacena 'echo hola'*

$ chmod +x ~/bin/eje *Le damos permiso de ejecución*

$ eje *Lo intentamos ejecutar escribiendo su nombre*

g: Command not found.

$ ~/bin/eje *Lo intentamos ejecutar con una trayectoria absoluta*

hola

$ export PATH= ${PATH}:${HOME}/bin *Añadimos a PATH nuestro directorio bin*

$ eje *Lo intentamos ejecutar escribiendo su nombre*

hola

Lo más interesante de esta secuencia es la utilización de la orden `export` para cambiar el valor de la variable de entorno **PATH**. Este cambio sólo sirve para la sesión de trabajo actual o bien ejecuciones, directas o indirectas, realizadas desde esa sesión. Si queremos incluir al directorio **~/bin** en nuestro **PATH** para cualquier sesión que iniciemos deberemos incluir la orden de modificación de **PATH** en un fichero de iniciación como **.bashrc**.

6 Iniciación

Cuando se ejecuta **bash**, una de las primeas cosas que hace es buscar un fichero llamado **.bashrc** en el directorio base, y ejecuta las órdenes que contiene. Este fichero nos sirve para configurar nuestro entorno, pues siempre que iniciemos una sesión de un modo determinado. En este fichero podemos utilizar órdenes como `export` o simplemente la asignación para añadir nuevos directorios en la búsqueda de ejecutables, o como `umask` para establecer una mascarilla de usuario inicial.

7 Ejercicios propuestos

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

```
$ export X=1; set x=2
$ echo $X $x
$ (echo $X $x; export X=3; export x=4; echo $X $x)
$ echo $X $x
$ cat >prueba
```

```
#!/bin/bash
echo $X
echo $x
export X=5
x=6;
echo $X $x
ctrl-d
$chmod +x prueba
$prueba
$echo $X $x
$ source prueba
$ echo $X $x
```

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

```
$ cd
$ mkdir tmp
$ cp /bin/pwd tmp/who
$ PATH=~/tmp:$PATH
$ who
/home/x1111111                          #Muestra su directorio base
```

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

```
Bienvenido al sistema “usuario”. Hoy es Tue Nov 26
11:58:20 CET 2002
```

Para ello puede utilizar la variable de entorno **USER** que almacena su nombre de usuario y la orden **date** que manda a la salida estándar la fecha y hora actual.

Realice la siguientes acciones mediante órdenes en **bash**

- Cree un fichero de texto cuyo nombre incluya algún espacio en blanco, visualice su contenido con la orden **cat**..
- Utilice la orden **find** para buscar todos los ficheros de nombre **.bashrc** a los que tenga acceso en el sistema de ficheros.
- Como quiera que existen muchos directorios a los que no tiene acceso la salida de la orden **find** anterior habrá sido difícil de leer. Corrija la situación redireccionando la salida y error estándar de **find** a distintos ficheros.
- Emplee interconexiones y las órdenes **history**, **grep** y **tail** para

obtener en la salida estándar la última orden que escribió que contenía la palabra **ls**.

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

El siguiente ejercicio propone realizar un guión de shell, que llamaremos **nombreNue**: a veces es preciso utilizar ficheros temporales. Para designar el enlace a un fichero temporal normalmente se utiliza un nombre que no coincida con ningún nombre de enlace existente en el sistema de ficheros para evitar sobreescrituras. Cree un guión que muestre en la salida estándar un nombre de enlace que no existe en el directorio temporal **/tmp**.

Sugerencia: Pruebe con el nombre **f0**, si existe con **f1**, si también existe con **f2** y así sucesivamente.

Suponga que un guión utiliza a este guión para obtener un nombre de enlace que no existe y posteriormente crear un fichero con dicho nombre. ¿Es seguro que cuando se intente crear el enlace no existirá un enlace con el mismo nombre? Antes de responder piense que UNIX es un sistema multiusuario y multiproceso.

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

Solución Ejercicio

nombresenlaces.txt (921 B)

En este ejercicio debe diseñar un guión que dado el nombre de un enlace muestre en la salida estándar la trayectoria absoluta de los directorios del sistema de ficheros a los que tiene acceso el usuario que ejecutó el guión y que contienen un enlace con dicho nombre. Por ejemplo, si el guión se llama localiza y de los directorios del sistema de ficheros a los que tiene acceso el usuario únicamente existe un enlace de nombre **pwd** en los directorios **/bin** y **/usr/share/zsh/help**, entonces la salida sería la siguiente:

$ localiza pwd

/bin

/usr/share/zsh/help

En ningún caso el guión debe mostrar información sobre los directorios a los que no tiene acceso el usuario.

(nota: **bash** cuenta con una utilidad para eso llamada **which**)

Sugerencia: Utilice la orden **find** para localizar los directorios. Redireccione la salida y error estándar de **find** en ficheros temporales aplicando el guión

escrito en el ejercicio anterior. Cuando ya no necesite los ficheros temporales en el guión elimínelos.

La edición de la pregunta no ha terminado. Por favor, haga click y edite o elimine la pregunte.

Solución Ejercicio
localiza.txt (714 B)