Listado de Órdenes en Linux

por Pablo Meca

Recopilación de los comandos estudiados en la asignatura “Introducción a Sistemas Operativos” de segundo curso del grado en Ingeniería en Informática y otras órdenes de interés.

Contenido

TEMA 1: Listado de ficheros y directorios

~: directorio de usuario. Tanto en Windows como en Linux, pulsar ALT GR (el de la derecha) + 4, soltar y luego pulsar espacio.

pwd: muestra el directorio actual.

whoami: muestra quién eres.

ls [OPCIÓN]… [FICHERO]…

-l: listado en formato largo.

-R: listado de forma recursiva.

-a: lista también los elementos ocultos (su nombre empieza por ".").

-S: lista en orden decreciente de **tamaño**.

-t: lista en orden decreciente de **fecha**.

-r: revierte el orden de listado.

-d: trata a los directorios como ficheros.

Comodines

***:** cero o más caracteres cualesquiera.

?: un carácter cualquiera.

[...]: un único carácter que puede ser cualquiera de los indicados en los []. Se pueden señalar rangos [1-5], en el caso de letras del abecedario, señalar que el orden es aAbBcC...

[!...]: un único carácter que no sea de los indicados. Se pueden señalar rangos.

{nom1, nom2,...}: cualquier elemento de la lista.

Tabla de clases de caracteres

Clases de Caracteres	Descripción
[:alnum:]	Los caracteres alfanuméricos. En ASCII, equivalente a: [A-Za-z0-9]
[:word:]	Los mismo que [:alnum:], con el añadido del carácter subrayado (_).
[:alpha:]	Los caracteres alfabéticos. En ASCII, equivalente a: [A-Za-z]
[:blank:]	Incluye los caracteres del espacio y tabulador.
[:cntrl:]	Los caracteres de control ASCII. Incluyen los caracteres ASCII del 0 al 31 y el 127.
[:digit:]	Los números del cero al nueve.
[:graph:]	Los caracteres visibles. En ASCII, incluye los caracteres del 33 al 126.
[:lower:]	Las letras minúsculas.
[:punct:]	Los símbolos de puntuación. En ASCII, equivalente a: [-!"#$%&'()*+,./:;<=>?@[\\\]_`{|}~]
[:print:]	Los caracteres imprimibles. Los caracteres de [:graph:] más el carácter espacio.
[:space:]	Los caracteres de espacio en blanco, incluyendo el espacio, el tabulador, el retorno de carro, nueva linea, tabulador vertical, y salto de página. En ASCII equivalente : [\t\r\n\v\f]
[:upper:]	Los caracteres en mayúsculas.
[:xdigit:]	Los caracteres usados para expresar números hexadecimales. En ASCII, equivalente a: [0-9A-Fa-f]

Ficheros y directorios

touch: permite cambiar datos como el tiempo de un fichero, si se hace sobre un nombre de fichero no existente, lo crea.

cp [OPCIÓN]... ORIGEN DESTINO: copia ficheros.

-r o -R: copia recursivamente.

-i: pide confirmación antes de sobrescribir un fichero existente.

-a: copia recursivamente conservando todos los atributos posibles.

mv: mueve los ficheros. Tiene los mismos parámetros que cp.

rm: elimina ficheros y directorios.

-i: pide confirmación.

-f: anula la -i, no pide confirmación.

-r: borra de forma recursiva.

mkdir: crea un directorio.

rmdir: elimina un directorio. Si no está vacío no lo podrá borrar, elimina primero todo su contenido con rm -r.

cat *fichero*: muestra el contenido de un fichero en el terminal.

less *fichero*: muestra el contenido del fichero página a página. Se cierra con q.

nano *fichero*: muestra el fichero en un sencillo editor de texto.

echo *parámetros*: escribe a su salida estándar los parámetros que recibe.

vim fichero: abre un editor para ver/modificar el fichero. Escribe i para poder escribir. Pulsa ESC (si estás en modo insertar) y luego :w para escribir, :q para salir y :q! para salir sin guardar.

Permisos

chmod [OPCIÓN]... MODO-EN-OCTAL FICHERO...: para modificar los permisos de un fichero.

Búsqueda de ficheros

find [punto-de-inicio...] [expresión]: realiza una búsqueda de ficheros en los subárboles de los directorios indicados como puntos de inicio. La expresión puede estar formada por diferentes elementos entre los que destacan:

- *tests*: devuelven verdadero o falso. Por ejemplo, -empt devuelve verdadero si el fichero está vacío.
- *acciones:* llevan a cabo una acción y devuelven verdadero o falso si la han podido realizar.
- *operadores:* sirven para juntar al resto de elementos de una expresión. Los tres principales son -a (Y-lógico), -o (O-lógico) y ! (NO-lógico). De forma predeterminada está -a. Para escribir paréntesis usaremos \(y \).

Tests

-name patrón: para buscar ficheros cuyo nombre cumpla con el patrón dado.

-iname patrón: igual que -name pero no distingue entre mayúsculas y minúsculas.

-size [+|-]n[cwbkMG]: para buscar ficheros cuyo tamaño sea mayor que n bloques (+n), exactamente n bloques (n) o menor que n bloques (-n). Cada bloque suele ser considerado de **512** bytes y equivale a poner b como unidad. Otras unidades posibles son c para bytes, w para palabras de dos bytes, k para kilobytes (1024 bytes), M para megabytes (1 048 576 bytes) y G para gigabytes (1 073 741 824 bytes).

-mtime: [+|-]n: para buscar ficheros que fueron modificados por última vez hace más de n días (+n), exactamente hace n días (n) o hace menos de n días (-n).

-atime [+|-]n: igual que -mtime, pero teniendo en cuenta el tiempo del último acceso.

-ctime [+|-]n: igual que -mtime, pero teniendo en cuenta el tiempo del último cambio de estado.

-newer fic: para buscar ficheros que hayan sido modificados más recientemente que el fichero dado.

-type c: para buscar ficheros de tipo c, donde c puede ser "f" para ficheros regulares, "d" para directorios, "l" para enlaces simbólicos, etc.

-perm [-/]permisos: para buscar ficheros cuyos permisos en este momento sean bien exactamente los especificados, bien al menos todos cuando va precedido por "-", bien alguno de los permisos cuando va precedido de "/".

Acciones

-print: devuelve siempre verdadero y muestra el nombre del fichero actual. Es la acción que se realiza por defecto si no se indica ninguna otra acción o si no se indica ninguna expresión.

-exec orden: ejecuta la orden dada y devuelve verdadero o falso dependiendo de si la orden se ejecutó con éxito o no. Es posible indicar el fichero que se está procesando actualmente usando la cadena "{}". El final de la orden se indica con la cadena "\;".

-ok orden: igual que -exec pero pregunta al usuario primero si desea ejecutar la orden o no sobre cada fichero encontrado.

-printf formato: siempre devuelve verdadero e imprime, por cada fichero encontrado, una cadena de texto con formato que puede incluir cierta información útil sobre dicho fichero. Si en la cadena de texto incluimos %s, la orden nos mostrará el tamaño del fichero, con %u nos mostrará el usuario propietario del fichero, con %p nos mostrará la ruta completa del fichero, etc (ver página siguiente).

Opciones

-depth: procesa el contenido de cada directorio antes del directorio en sí.

-maxdepth N: desciende un máximo de N niveles de subdirectorios a partir de los directorios indicados como argumentos.

Principales directivas para usar en printf

\n **Salto de línea.**

\r **Retorno de carro.**

\t **Tabulador.**

\v Tabulador vertical.

\0 ASCII NUL.

\\ Barra como literal ('\').

\NNN **El carácter que tiene el código ASCII NNN (octal).**

%% Signo de porcentaje como literal.

%a **Último acceso de un archivo** (función de C: 'ctime').

%Ak Último acceso de un archivo en el formato especificado por k.

%b **Cantidad de espacio usada por el archivo** (512-byteblocks).

%c **Último cambio de estado de un archivo** ('ctime').

%Ck Último cambio de estado en el formato especificado por k.

%d **Profundidad del archivo dentro del árbol de directorios,** 0 significa que está en el punto de inicio.

%f **Nombre base del archivo.**

%g **Nombre del grupo al que pertenece un archivo.**

%h **Directorios en los que se encuentra el archivo.**

%H Punto de inicio desde el que fue localizado.

%k **Cantidad de espacio usado por el archivo (1KB).**

%m **Permisos del archivo en octal.**

%M Permisos del archivo de forma simbólica.

%p **Ruta completa del archivo.**

%P Nombre del archivo sin el punto de inicio.

%s **Tamaño del archivo en bytes.**

%t **Última modificación del archivo** (ctime).

%Tk Formato especificado por k.

%u **Nombre de usuario del archivo.**

%U ID numérico de usuario del archivo.

%y Tipo de archivo (f, d).

TEMA 2: Operadores para redireccionar la entrada y la salida

>: redirecciona la salida estándar a un fichero (lo sobrescribe).

>>: Igual que >, pero añade los datos al final.

2>: redirecciona la salida estándar de error a un fichero (lo sobrescribe).

2>>: Igual que 2>, pero añade los datos al final.

n>&m: redirecciona el descriptor de fichero n (1=estándar, 2=error) al descriptor m, si se omite n se sobrentiende un 1.

<: redirige la entrada estándar a un fichero.

| (tubería): redirecciona la salida estándar de una orden a la entrada estándar de la orden que le sigue.

|&: tubería en la que se redireccionan tanto la salida estándar como la salida estándar de error de una orden a la entrada estándar de la orden que le sigue.

Si queremos redireccionar solamente la salida estándar de error: find /etc 2>&1 > /dev/null | wc -l. **/dev/null** es un dispositivo nulo, una papelera donde mandar la información que no interesa.

GREP

grep [opción]... patrón [fichero]...: realiza una búsqueda del patrón de texto especificado en uno o más ficheros, mostrando todas las líneas de texto que lo contienen. Si no se especifica fichero, toma la entrada estándar del sistema.

-i: ignora las diferencias entre mayúsculas y minúsculas.

-n: muestra las líneas y el número de cada línea.

-c: muestra la cuenta de líneas coincidentes.

-l: muestra los nombres de ficheros con l. coincidentes.

-h: muestra las líneas coincidentes, pero nos los nombres de los ficheros.

-v: muestra las líneas que no coinciden.

-w: muestra las líneas que contienen el patrón como una palabra completa.

-x: muestra las líneas que coinciden completamente con el patrón buscado.

-o: muestra solo las partes coincidentes de una línea.

-q: suprime la salida normal. La búsqueda termina en la primera concordancia.

Se pueden usar caracteres especiales como:

.: un carácter cualquiera.

c*: cero o más ocurrencias del carácter c.

c\+: una o más ocurrencias del carácter c.

^: comienzo de línea.

$: final de línea.

[...]: uno de entre un conjunto de caracteres. En el interior de esta expresión, el carácter ^ tiene el significado de negación cuando aparece al principio, justo detrás de [.

c\{n,m\}: entre n y m repeticiones del carácter c.

\c: para deshabilitar el significado especial de c.

SORT

sort [opción]... [fichero]...: ordena alfabéticamente las líneas de los ficheros. Si no se especifica fichero, las líneas que recibe por su entrada estándar.

-n: ordena numéricamente.

-r: orden inverso.

-kn,m: ordena en función de columnas (campos) de las líneas. Ordena desde la n hasta la m, si se omite, ordena hasta el final. Si son iguales, ordena en base a una columna. Se puede concatenar varias veces esta instrucción para seleccionar campos separados.

-t: para especificar el carácter que marca la separación, si no se usa, el separador será la *cadena vacía*.

TR

tr [opción]... conjunto1 [conjunto2]: copia el texto desde su entrada estándar, reemplazando los caracteres indicados en conjunto1 por los caracteres correspondientes de conjunto2.

-s: elimina repeticiones contiguas.

-d: elimina los caracteres especificados.

-c: utiliza el conjunto complementario de caracteres, es decir, todos menos los indicados en conjunto1.

Especificaciones especiales para conjuntos:

x-y: un rango de caracteres desde x hasta y.

\c: secuencia de escape (\n salto línea, \t tabulador)...

[:class:]: una clase de caracteres (alnum para alfanuméricos, digit para dígitos, etc).

[c*n]: n repeticiones del carácter c.

CUT

cut opción... [fichero]...: seleccionar columnas de un fichero (si no se especifica, entrada estándar) de texto y mostrarlas en la salida estándar.

-c: columnas especificadas por posiciones de caracteres.

-f: especificadas por campos. Por defecto, tabulador.

-d: se usa junto a -f para indicar un delimitador distinto del tabulador.

-s: usada junto a -f para indicar que, si una línea no contiene tabulador, no debe mostrarse en la salida.

Si ponemos -n eliminará lo que haya después de la posición n.

Si ponemos n- eliminará lo que haya antes de la posición n.

Es importante no olvidar el uso de la orden tr antes de una orden cut cuando se usa la opción -f (ya que los campos pueden estar repetidos p.ej: ‘ ’ y no ‘ ’).

UNIQ

uniq [opción]... [fichero_entrada [fichero_salida]]: elimina líneas duplicadas de un fichero, siempre que aparezcan juntas. Si no se indica un fichero de entrada, no se podrá especificar uno de salida.

-c: escribe a la salida el número de veces que aparece cada línea en el fichero de entrada justo antes de escribir la propia línea.

-d: cada línea duplicada es escrita a la salida solamente una vez. No escribe ninguna línea que no estuviera duplicada en la entrada.

-D: igual que -d, pero sin sustituir las líneas duplicadas por una sola.

-u: escribe a la salida únicamente las líneas no duplicadas de la entrada.

-i: no distingue entre mayúsculas y minúsculas.

-s N: no tiene en cuenta los N primeros caracteres de cada línea a la hora de comparar.

-w N: solamente tiene en cuenta los N primeros caracteres de cada línea a la hora de comparar.

Otras órdenes

head [opción]... [fichero]...: muestra las primeras líneas de un fichero. Sin opciones muestra las primeras 10.

-n, --lines=[-]NUM: muestra las primeras NUM líneas. Si el número va precedido por un signo menos, muestra todas las líneas excepto las últimas NUM.

tail [opción]... [fichero]...: muestra las últimas líneas de un fichero. Por defecto las 10 últimas.

-n, --lines=[+]NUM: muestra las últimas NUM líneas. Si el número va precedido por un signo más, muestra todas las líneas a partir de la línea NUM.

column [opción]... [fichero]...: muestra una lista de elementos en columnas. Sin parámetros va rellenando las columnas antes que las filas, el número de columnas dependerá del ancho de la terminal.

-t, --table: determina el número de columnas que contiene la entrada y crea una tabla.

-s, --separator separadores: especifica los posibles separadores de columnas en las líneas de entrada.

wc [opción]... [fichero]...: muestra el total de líneas, palabras y bytes (en ese orden) para cada fichero dado.

-c, --bytes: muestra el total de bytes.

-m, --chars: muestra el total de caracteres.

-l, --lines: muestra el total de líneas.

-L, --max-line-legth: muestra la longitud de la línea más larga.

-w, --words: muestra el total de palabras.

tee [opción]... [fichero]...: lo que lee de la entrada estándar lo escribe en la salida estándar y en los ficheros dados.

-a, --append: los ficheros dados como argumentos que ya existen no se sobrescriben, se añade la información al final.

Tema 3: GUIONES SHELL

Creamos un fichero de texto que termine en .sh. En la primera línea escribimos: **#!/bin/bash** Para ejecutarlo usar un intérprete como bash: **$ bash limpiafecha.sh**, otra opción es darle permisos de ejecución (chmod) y ejecutarlo como si fuese un programa normal.

Variables

Las variables se asignan con =, no existen tipos, simplemente variable=valor. Se pueden asignar directamente desde la terminal.

Para ver el valor de una variable se usa $, por ejemplo, utilizando el comando echo: **echo $variable**. Es importante mencionar que echo muestra por pantalla la entrada usando como separador 1 espacio en blanco, por lo que si se introduce: Hola Pepe, imprimirá: Hola Pepe.

Parámetros

```
$1 $2 $3 ... ${10} ${11} ${12} ...
```

- La variable $0 contiene el nombre con el que se ha invocado el script, $1 contiene el primer parámetro, $2 contiene el segundo parámetro,....
- La variable $@ contiene todos los parámetros recibidos.
- La variable $# contiene el número de parámetros recibidos.

La orden **shift** desplaza los parámetros hacia la izquierda una vez, es decir $1=$2 y $@ $# reducirán en uno sus valores.

Se puede acceder a un parámetro en una posición determinada mediante la variable i (previamente definida con un valor numérico) de la siguiente forma: **${!i}**.

CÓDIGO DE SALIDA

Todos los procesos devuelven un código o estado de salida que será 0 cuando la ejecución haya sido correcta, y un valor entre 1 y 255 cuando se haya producido algún error. Se puede comprobar este valor en la variable: **$?**. Se puede obtener el valor opuesto anteponiendo **!** en la instrucción.

Aclaración sobre su uso: primero ejecutamos una instrucción cualquiera (poniendo un ! primero si buscamos el valor contrario) e, inmediatamente después de haberla ejecutado, usamos $? para conocer el valor que ha devuelto.

Si usamos **exit n** se devolverá el valor de n, si no usamos ningún exit, se devuelve el último valor de la última instrucción ejecutada en el script.

CARACTERES ESPECIALES Y ENTRECOMILLADO

Para preservar el valor literal de un carácter (y no su función especial) se puede colocar delante la **** , que es el carácter de escape. La única excepción a esto es cuando se coloca \ seguida de una nueva línea, en este caso anula el INTRO y sigue esperando el final de la orden en la línea siguiente:

```
$ ls \
> -l
```

sería equivalente a ejecutar:

```
$ ls -l
```

Encerrar uno o más caracteres en comillas simples (‘ ’) hace que todos ellos preserven su valor literal. Por otro lado, encerrarlos en comillas dobles (“ ”) preserva el valor de todos **menos $**, ‘ ’ y ****. Una comilla doble puede aparecer entre otras comillas dobles si está precedida de \.

SUSTITUCIÓN DE ÓRDENES – MUY IMPORTANTE

Si hacemos **$(**cadena**)** forzamos al shell a ejecutar la orden en **cadena** y sustituir $(cadena) por su resultado. Esto permite cosas como **usuario=$(whoami)**.

$(#variable**)** devuelve el número de caracteres que hay en la variable. Sería equivalente a hacer $(echo -n $variable | wc -c).

A la hora de usar el entrecomillado doble, hay que tener en cuenta que en ciertas órdenes (echo “ls -l *”) bash es el encargado de reconocer ese carácter especial y expandirlo, por lo que para echo * es *.

EJECUCIÓN DE UNA ÓRDEN DENTRO DEL SCRIPT

Si en algún momento necesitamos ejecutar una orden dentro de nuestro script, debemos llamar a bash: **bash** orden.

ÓRDENES RELACIONADAS CON EL TIEMPO

La orden **sleep** permite el retardo de un tiempo especificado, por defecto segundos (s). Uso: **sleep -[smhd]** número

La orden **date** nos proporciona la fecha actual. Si hacemos **date + “%s”** nos mostrará el segundo actual, %M los minutos, %k la hora en formato 24 y %l la hora en formato 12.

EVALUACIÓN ARITMÉTICA

-, +	Menos y más unarios
**	Exponenciación
*, /, %	Multiplicación, división, resto
+, -	Adición, sustracción
=, +=, -=, *=, /=, %=,	Asignación: simple, después de la suma, de la resta,...

Las constantes con un 0 inicial se interpretan como números octales, mientras que un 0x o 0X inicial denota un número en hexadecimal.

Podemos hacer cálculos directamente así: **$((**operando-operador-operando**))**, o con **let** variable=operación.

Otra forma de evaluar expresiones es con la orden **bc**. Si se coloca sola en la terminal, se podrán realizar tantas operaciones como se quiera hasta pulsar Ctr+C. También se puede usar con tuberías, si hacemos **echo "1+2" | bc** nos devolverá 3.

Este comando se vuelve muy potente cuando lo combinamos con **paste**, que combina (pega) líneas de ficheros (mirar anexo):

f1 f2: pega los ficheros línea a línea.

-s f1: coloca en una línea por fichero todas las líneas que contiene.

-s f1 -d 'delimitador': usa el delimitador indicado.

La orden **test** evalúa si la expresión que recibe como parámetro es verdadera o falsa, devolviendo como código de salida un 0 o un 1 respectivamente.

- Para números: **test arg1 OP arg2**, donde OP puede ser uno de los siguientes:

-eq	Igual a
-ne	Distinto de
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que
-ge	Mayor o igual que

- Para caracteres alfabéticos o cadenas teniendo en cuenta el orden lexicográfico:

-z cadena	Verdad si la longitud de cadena es cero.
-n cadena	Verdad si la longitud de cadena no es cero.
cadena1 = cadena2	Verdad si las cadenas son iguales. También se puede emplear == en vez de =.
cadena1 != cadena2	Verdad si las cadenas no son iguales.
cadena1 \<cadena2	Verdad si cadena1 se ordena lexicográficamente antes de cadena2.
cadena1 \>cadena2	Verdad si cadena1 se ordena lexicográficamente después de cadena2.

- En la expresión se pueden incluir comprobaciones sobre ficheros, entre otras:

`-e fichero`	El fichero existe.
`-r fichero`	El fichero existe y tengo permiso de lectura.
`-w fichero`	El fichero existe y tengo permiso de escritura.
`-x fichero`	El fichero existe y tengo permiso de ejecución.
`-f fichero`	El fichero existe y es regular.
`-s fichero`	El fichero existe y es de tamaño mayor a cero.
`-d fichero`	El fichero existe y es un directorio.

- Además, se pueden incluir operadores lógicos y paréntesis:

`-o`	OR
`-a`	AND
!	NOT
\(	Paréntesis izquierdo
\)	Paréntesis derecho

ESTRUCTURAS DE CONTROL

Orden if:

```
if  <orden_a_evaluar_if>
then
      <lista_ordenes_if>

elif  <orden_a_evaluar_elif_1>
then
      <lista_ordenes_elif_1>
      # (el bloque elif y sus órdenes son opcionales)

elif  <orden_a_evaluar_elif_2>
then
      <lista_ordenes_elif_2>
      # (el bloque elif y sus órdenes son opcionales)
...
else
      <lista_ordenes_else>
      # (el bloque else y sus órdenes son opcionales)
fi
```

Es conveniente utilizar " " con las variables, porque pueden contener espacios o provocar errores (cadena vacía). Cuidado con los paréntesis, usar \ y dejar un espacio: **\(argumentos \)**.

Si estamos dando un mensaje de error, es **MUY IMPORTANTE** redirigir el mensaje a la salida estándar de error con 1>&2 y devolver un exit con un número que no sea 0.

Orden case:

```
case $variable in
     patrón_1)
          <lista_ordenes_1>
          ;;
     patrón_2|patrón_3)
          <lista_ordenes_2_3>
          ;;
     patrón_n)
          <lista_ordenes_n>
          ;;
     *)
          <lista_ordenes_por_defecto>
          ;;
esac
```

Bucle condicional while:

```
while <orden_a_evaluar> # Mientras el código de retorno sea 0
do
     <conjunto_instrucciones>
done
```

Bucle incondicional for:

```
for variable in <lista_de_valores>
do
     <conjunto_instrucciones>
done
```

Aunque la lista de valores del for puede ser arbitraria (incluyendo no sólo números, sino cualquier otro tipo de cadena o expresión), a menudo lo que queremos es generar secuencias de valores numéricos. En este caso, la orden **seq**, combinada con el mecanismo de sustitución de órdenes puede resultarnos de utilidad: **seq desde de_cuanto_en_cuanto hasta**. Todos los números del 1 al 100: seq 1 1 100. Por ejemplo: **for i in $(seq 0 10 100).**

Ruptura de bucles: break y continue:

La orden **break** transfiere el control a la orden que hay tras la instrucción done, haciendo que el bucle termine antes de tiempo.

La orden **continue** transfiere el control a la instrucción done, con lo que la ejecución del bucle continúa en la siguiente iteración.

Tema 4: Guiones Shell avanzados

Hay dos tipos de variables en bash: locales y de entorno. La diferencia es que las segundas mantienen sus valores incluso después de que los procesos hijos ejecuten una llamada a **exec**.

Llamando a **export** podemos declarar nuevas variables de entorno, o convertir una local en variable de entorno. La orden **set** muestra todas las variables, mientras que la orden **env** muestra solo las variables de entorno.

- **HOME**: directorio de trabajo del usuario, tomado por defecto en la orden cd.
- **LANG**: idioma utilizado por los programas, para cambiar (por ejemplo) al francés, haremos: LANG=fr_FR.UTF-8
- **LOGNAME**: nombre de usuario.
- **PWD**: contiene el conjunto de rutas de directorios separadas por :, cuando se invoca una orden o programa indicando solo el nombre, el Shell buscará en estas rutas por orden.

ÓRDENES INTERNAS DE BASH

Son órdenes que están integradas directamente en /bin/bash, aunque también pueden tener sus propios ejecutables en /bin/nombre de la orden.

- **echo**: envía cadenas de caracteres que recibe a la salida estándar:

  echo [opción]... [cadena]...

  -n: no se realiza el salto de línea tras mostrar el texto.

  -e: habilita la interpretación de secuencias especiales:

  \b: retrocede una posición sin borrar.

  \f: alimentación de página.

  \n: salto de línea.

  \t: tabulador.

  Importante encerrar las cadenas entre “” para que reconozca los argumentos correctamente.

- **printf**: permite utilizar el formateo de datos de C.
- **read**: lee una línea de la cadena de entrada y le asigna una variable (como cin en C++).
- **cd, pwd, let, test**
- **export**: permite que una variable esté disponible para todos los procesos hijos.
- **exit**: finaliza la ejecución del guión (devuelve 0 de forma predeterminada, si se quiere detallar un código de error, hay que colocarlo al lado: exit 2).
- **true** y **false**: devuelven como código de retorno 0 y 1 respectivamente. En Linux el 0 es true y cualquier valor distinto es false.

GRAMÁTICA DE BASH: órdenes simples

Una orden simple está formada por cuatro elementos: el primer elemento (opcional) es una secuencia de asignaciones a variables. El segundo elemento es una lista de palabras separadas por espacios en blanco, donde la primera palabra es la orden a ejecutar y el resto son los argumentos de la misma. El tercer elemento, también opcional, lo conforman posibles redirecciones. Finalmente, el cuarto elemento es un operador de control.

LISTAS DE ÓRDENES

Una lista de órdenes es una secuencia de una o más tuberías separadas por uno de los operadores ;, &, &&, || y terminada opcionalmente por ;,. &, o <nueva-línea>.

La ordenes separadas por el operador ; se ejecutan secuencialmente, Shell espera a que cada orden termine a antes de ejecutar la siguiente. El estado devuelto es el estado de salida de la última orden ejecutada.

Los operadores de control && y || denotan AND y OR, donde la segunda orden depende su ejecución del estado de salida devuelto por la primera. El estado devuelto es el de la última orden ejecutada.

El último operador es el &, el cual ejecuta las ordenes en segundo plano, de forma paralela, sin que una orden espere a la otra, el código devuelto es siempre 0.

GRAMÁTICA DE BASH: órdenes compuestas

Es una lista de n órdenes que se ejecuta en un nuevo Shell en un proceso hijo (fork). Por ejemplo, para procesar la salida de varios procesos como una sola haremos: $ (echo bb; echo ca; echo aa; echo a) | sort

En bash, while, for, if o case son órdenes compuestas. La diferencia entre estas y las anteriores es que estas son ejecutadas por el mismo Shell, sin crear nuevos procesos hijos.

DEPURACIÓN

Bash pone a nuestra disposición unos pocos comandos (combinables) para la depuración, puedes usarse al llamar al shell desde bash: bash -depuración shell. O desde dentro del Shell en la primera línea si le vamos a dar permisos de ejecución (#!/bin/bash -dep).

-u: detecta variables o parámetros que son utilizados sin estar inicializados previamente (p.ej: error ortográfico).

-x: depuración línea a línea, sustituyendo las variables.

-v: similar a la anterior, pero muestra tal cual las líneas.

ANEXO: PATRONES DE USO DEL SHELL

- **Comprobación de la cadena vacía**

```
if test "$a" = "" ; then echo "cadena vacia" ; fi
```

- **Recorrer la salida de una orden**

```
#!/bin/bash
numerocliente=0
cat $1 | (while read cliente
do
      let numerocliente=$numerocliente+1
      echo "#$numerocliente : $cliente"
done;
echo "Numero total de clientes: $numerocliente")
```

Se hace una orden compuesta para que el valor de numerocliente no se pierda al salir del bucle.

- **Leer un fichero línea a línea**

```
#!/bin/bash
numerocliente=0
while read cliente
do
      let numerocliente=$numerocliente+1
      echo "#$numerocliente : $cliente"
done < $1
echo "Numero total de clientes: $numerocliente"
```

En este caso no hace falta hacer una orden compuesta porque tanto while como echo se ejecutan en el mismo proceso.

- **Utilización de ficheros temporales**

Utilizar la instrucción **mktemp** que crea un fichero en la carpeta /tmp y devuelve por la salida estándar el nombre del fichero creado. Se usa así: **variable=$(mktemp)**.
Cuando terminemos de usarlo, si queremos devolverlo, lo renombramos: mv $temp "convert-$1" y ya deja de ser un fichero temporal.

- **Comprobar si una determinada variable posee un valor numérico válido**

```
if echo $1 | grep -x -q "[0-9]\+"
then
     echo "El argumento $1 es un numero natural."
else
     echo "El argumento $1 no es un numero natural correcto."
fi
```

Se puede ampliar esto para que acepte también enteros con signo:

```
if echo $1 | grep -x -q "[+-]\?[0-9]\+"
then
     echo "El argumento $1 es un numero entero."
else
     echo "El argumento $1 no es un numero entero correcto."
fi
```

- **Convertir a mayúsculas una palabra ($2) de un fichero ($1) y guardar el resultado en uno nuevo no existente($3)**

```
if test $# -ne 3 -o ! -f "$1" -o -e "$3"
     then
     echo "Numero de argumentos incorrecto o alguno de los argumentos no es valido." >&2
     exit 1
fi
temp=$(mktemp)
mayuscula=$(echo $2 | tr "[:lower:]" "[:upper:]")
while read linea
do
     comienzo=""
     for palabra in $linea
     do
     if test "$palabra" == "$2"
     then
     echo -n "$comienzo$mayuscula" >> $temp
     else
     echo -n "$comienzo$palabra" >> $temp
     fi
     if test -z "$comienzo"
     then
     comienzo=" "
     fi
done
echo >> $temp
done < "$1"
mv $temp "$3"
exit 0
```

- **Combinar paste y bc para operar de forma masiva**

  Si tenemos que operar el total de algo, podemos combinar paste (que cambiará el separador por un operador) y bc.

  En el ejemplo sumaremos la cantidad de memoria virtual (VSZ) usada por alumno:

  ```
  ps aux | tail -n +2 | grep -w "^alumno" | tr -s " " | cut -f5 -d " " | paste -s -d'+' | bc
  ```

- **Tratamiento de errores**

```
#!/bin/bash

if test $# -ne 2
then
     echo "ERROR: El guion requiere 2 parametros." >&2
     echo "USO: $0 <numero_natural> <nombre_fichero>" >&2
     exit 1
fi

exit 0

####################################
# resto del codigo del guion shell #
####################################
```

Siempre poner exit 0 al final para asegurarnos de que se devuelve correctamente el código de error.
Es importante redirigir el mensaje a la salida estándar de error y devolver un código de error.