Bash.pdf

esteralvarez1

Programación Para Sistemas

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

BASH

Introducción

* Interfaz de usuario en el que introducir comandos, ejecutarlos y mostrar su salida y resultados.

* Información sobre el sistema: uname -a

* Para lanzar un comando en un terminal: micomando<ENTER>

Puede ser necesario especificar una ruta de directorios si se trata de un comando externo (necesario si no está en ciertos directorios en el denominado PATH).

* Ante un comando a ejecutar con nombre 'micomando':

1. El shell primero busca un comando interno del propio shell con nombre ‘micomando'.

2. Si no existe como comando interno, el shell busca un comando externo (al shell) en los directorios incluidos en la variable de entorno 'PATH'.

Por ello, si 'micomando' no está en un directorio incluido en 'PATH', sería necesario especificar

ruta_directorios/micomando<ENTER>

* El comando 'type' da información acerca de si un comando es interno del shell o bien un comando externo:

type type

type es una orden interna del shell

* En casos de comandos que son a la vez comandos internos de bash y también comandos externos, el shell en principio utiliza el comando interno.

El lugar (ruta o 'path') del comando de usuario se puede encontrar con which.

Comandos de Ayuda

* Para comandos externos de usuario:

Para documentación: man
Para ayuda corta o resumida: comando_de_usuario —help

* Para buscar palabras clave en las descripciones de las páginas de la documentación de man:

apropos palabra_clave

Nota: apropos admite también patrones (expresiones regulares) de búsqueda.

Para localización: which o whereis

Nota: whereis proporciona también información sobre localización de la documentación asociada.

Para sacar en pantalla pequeñas descripciones de las páginas man de un comando: whatis

* Para ayuda e información de los comandos internos del shell (bash), existe 'help' (que es a su vez un comando interno):

help comando_interno

* Existe asimismo el comando 'info', que proporciona una ayuda extendida con enlaces hipertexto.

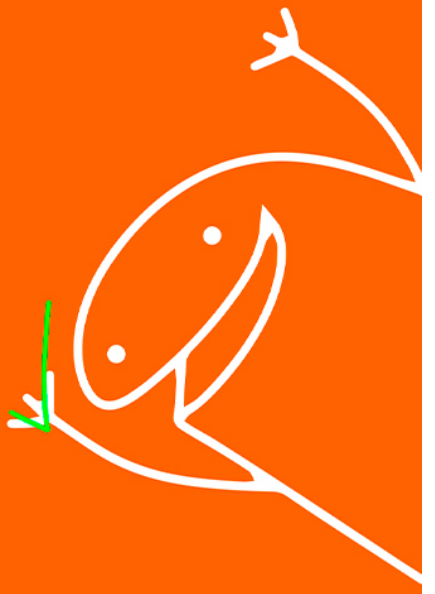
Dentro de info:

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

1/6
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

EXAMEN SORPRESA

¿Qué te pide la Cuenta NoCuenta?
a) Nada
b) Nada de nada



¿Qué comisiones* tiene?
a) 0 %
b) 0,00000000 %

¿Cuál es su permanencia?
a) Ninguna
b) Si ya sabes la respuesta pa qué preguntas

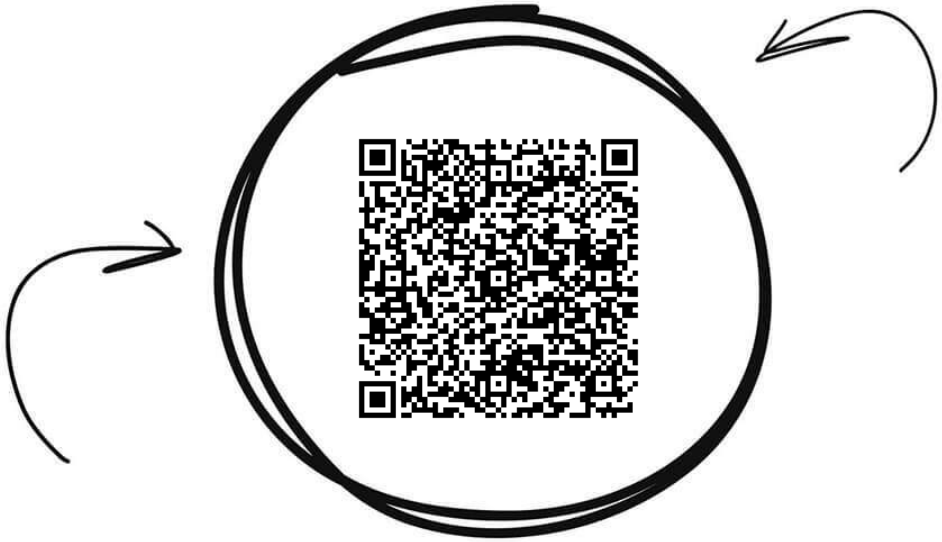
* Si has acertado todas las respuestas, esta cuenta es para ti

¡Me apunto!

* TIN 0 % y TAE 0 %.

do your thing

Programación Para Sistemas



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

-Para obtener ayuda: h

-Para salir: q

-Para avanzar página: barra de espacio

-Para retroceder página: tecla retroceso ('backspace')

-Para avanzar a siguientes enlaces hipertexto: TAB

-Para seguir un enlace hipertexto: ENTER

Si 'info' no dispone de información, se muestra la página de ‘man’.

Sistema de Ficheros y Comandos Básicos Asociados

* Unix proporciona un sistema de ficheros jerárquico

El nivel más alto (raíz) se denota con: '/'
Cada nivel puede contener ficheros y/o subdirectorios.
Las rutas que comienzan por '/' son rutas absolutas o completas
Una ruta relativa (que no comienza por '/') se refiere al directorio actual
El carácter '/' al final de un nombre de ruta de directorios es opcional.

* Nombres e identificadores de ficheros

Se diferencia entre mayúsculas y minúsculas.

* **Directorio inicial** de la cuenta o del usuario (directorio 'HOME').

Asociado a la variable de entorno 'HOME'.
Como nombre de directorio, el carácter '~' se expande al directorio 'HOME' de la cuenta. Junto a un nombre de usuario, se expande al directorio 'HOME' de dicho usuario:

~amigo/mis_notas.txt denota el fichero mis_notas.txt en el

directorio 'HOME' del usuario 'amigo'.

* Directorio actual de trabajo 'current working directory': 'cwd'

* Directorios especiales

'.' : directorio actual

'..' : directorio padre del directorio actual

Nota: los ficheros cuyo nombre empiezan por '.' se denominan 'escondidos' ('hidden').

*** Algunos comandos importantes:**

-Mostrar el directorio de trabajo actual: pwd

-Listar contenidos de directorios. ls /tmp

Para listar el directorio actual: ls

Para listar con información más detallada: ls -l /tmp

Para incluir ficheros 'escondidos' que empiezan por '.': ls -la /tmp

Para listar un subdirectorio: ls Dir1

Para mostrar directorios en vez de sus contenidos: ls -d Dir1

Para aplicar 'ls' recursivamente: ls -R

-Cambiar de directorio: cd /usr

Para para cambiar a un subdirectorio del actual: cd Dir1

Sin argumentos cambia al (HOME): cd

Para cambir al directorio 'padre': cd ..

-Crear directorio: mkdir /tmp/Prueba1

Si se quieren crear automáticamente los directorios padre se puede emplear la opción '-p': mkdir -p /tmp/Prueba2/Dir/Subdir

Borrar un directorio: rmdir Directorio

Nota: el directorio debe estar vacío. (Si no, y se quiere borrar todo su contenido, ver: rm -r

Copiar ficheros (y también directorios)

cp fichero_origen fichero_destino
cp fichero_origen1 fichero_origen2 ... directorio_destino

Para copiar un fichero al directorio padre: cp fichero_origen ..

Para pedir confirmación interactivamente antes de sobreescribir: cp -i

Para copiar directorios, recursivamente: cp -r dir_origen dir_destino

cp -r dir_origen1 dir_origen2 ... dir_destino

Para copiar directorios, recursivamente y preservando atributos de los ficheros: cp -a dir_origen dir_destino

cp -a dir_origen1 ... dir_origen2 dir_destino

Mover ficheros (y también directorios) mv dir_origen dir_destino

También utilizable para renombrar ficheros/directorios. Para pedir confirmación interactivamente antes de sobreescribir:

mv -i dir_origen dir_destino

Eliminar ficheros (y también directorios): rm conlleva un borrado irreversible.

Para pedir confirmación antes de eliminar un fichero: rm -i

Para borrar recursivamente un directorio y todo su contenido: rm -r

Para crear un fichero vacío o para modificar el tiempo de modificación de un fichero: touch

Para obtener la parte de directorios de un nombre de fichero:

dirname

- Para eliminar la parte de directorios de un nombre de fichero

basename

- Para mostrar el tamaño de los directorios (por defecto, en KB):

du

-Para mostrar en forma arborescente la estructura de directorios y ficheros:

tree

- Algunas opciones útiles:

Para mostrar sólo directorios: tree -d

Para mostrar sólo hasta un nivel de profundidad: tree -L 2

Para mostrar los ficheros con su ruta y sin indentar: tree -i -f

- Para información sobre un fichero, estimando el tipo: file

*** Metacaracteres de nombres de ficheros**

Para ajustar una cadena de 0 ó más caracteres: *

Para ajustar un carácter cualquiera: ?

Para ajustar un carácter del conjunto indicado: [abcd...]

Para ajustar un carácter no incluido en el cjto.: [!abcd...]

Directorio HOME del usuario de la cuenta: ~

Directorio HOME del usuario 'usuario': ~usuario

Comando 'alias'

* Permite definir 'alias', que se pueden considerar como abreviaturas de comandos más largos.

alias NOMBRE = VALOR

* Asimismo, pueden servir para añadir ciertas opciones útiles a comandos existentes.

Para mostrar un alias existente: alias ll

Para mostrar todos los alias definidos: alias

Para eliminar el alias ll: unalias ll

* Los 'alias' pueden ser funciones pero no pueden definir argumentos.

alias ll ='ls -l'

Cuentas y Usuarios - Superusuario ('root')

* Cada cuenta tiene asociados un único **nombre de usuario** y un número asociado **UID** ('user identifier').

* **Grupo**: facilitan la administración de usuarios. Cada usuario pertenece a un grupo al menos. Cada grupo tiene un nombre y un número GID ('group identifier') únicos asociados.

Nota: información en fichero del sistema /etc/group

* Las operaciones que una cuenta puede hacer dependen:

De su usuario (UID) y grupo (GID).
De los permisos que tengan los ficheros o recursos sobre los que se quiere operar.

* Comando 'id' para información de usuario y de grupo.

Del usuario actual: id
De otro usuario: id usuario

* Comando para cambiar la clave: passwd

* Una cuenta con privilegios para administrar el sistema es una cuenta 'superusuario' (administrador).Cuenta 'root' (usuario 'root' y grupo 'root').

Para hacer un login como superusuario (‘root'): su -l

* Para lanzar un comando aislado como superusuario ('root'), si el sistema se ha configurado para ello (y el usuario está habilitado):

sudo comando_a_ejecutar_como_superusuario

* Nota: además de las cuentas de usuarios normales y del superusuario, existen cuentas del sistema asociadas a componentes específicos (p. ej., asociadas al correo, a conexiones ssh, etc.).

Permisos de Ficheros

* Necesarios en sistemas multi-usuarios como Unix.

* Aspectos básicos de los permisos:

Se distingue, en cuanto a quién lo puede ejercer:

-Propietario 'u' ('owner' / 'user')

-Grupo 'g' ('group'): se pueden establecer grupos de usuarios (por parte del administrador del sistema).

-Otros 'o' ('other'): resto de usuarios.

Nota: la letra 'a' ('all') designa a todos. (Opción por defecto en modo simbólico (aunque en este caso los bits con valor 1 en la umask no se cambian).)

Se distingue, en cuanto al tipo de permiso:

-De lectura: 'r'
-De escritura: 'w'
-De ejecución: 'x'

Para mostrar los detalles de los permisos con 'ls':

ls -l fichero

ls -ld directorio

Ej.: ls -l /bin/bash
-rwxr-xr-x 1 root root 1021112 oct 7 2014 /bin/bash

Nota: al hacer un ls con formato largo con -l, aparecen 10 caracteres por cada elemento listado:

La posición 1: indica el tipo ('-': fichero; 'd': directorio; 'l': enlace simbólico, etc.).

Las posiciones 2,3,4: permisos rwx del propietario.

Las posiciones 5,6,7: permisos rwx del grupo.

Las posiciones 8,9,10: permisos rwx de otros.

* Para establecer los permisos: chmod

Existe también una notación simbólica:

El operador '+' añade los bits indicados.

El operador '-' elimina los bits indicados.

El operador '=' añade los bits indicados y elimina los demás

Como se puede observar, el separador en la designación de varios modos es la coma ','.

* Directorios y permisos:

En el caso de los directorios, así hay caso de lectura (y/o, de escritura), en principio se debe establecer también el permiso de ejecución , el cual está especialmente ligado al acceso al directorio.

* Hay aspectos más avanzados al respecto del tema de permisos:

Bit setuid, bit setgid, 'sticky bit'

* Para cambiar de propietario:

chown

Para ello se necesita ser el propietario del fichero, o bien ser root (superusuario) o lanzar el comando como root (con 'sudo').

* Para cambiar de grupo: chgrp

* Para mostrar o establecer la máscara de creación de ficheros o directorios:
umask

La máscara se da como número en octal.

Si se quiere formato simbólico: umask -S

La máscara hace que el sistema inhabilite los permisos cuyas posiciones q tengan un '1' en la máscara.

Las que tengan un '0', no estarán habilitadas necesariamente (dependerá de quien cree el fichero).

Nota: un valor por defecto típico es: 0022

En principio, es en general recomendable establecer el fichero de configuración de bash de la cuenta: ~/.bashrc

---

Enlaces ('Links')

* Un enlace es una referencia a otro fichero o directorio.

ln

* Enlaces simbólicos ('soft links')

ln -s fichero_o_directorio_original nombre_enlace_simbolico

Se refiere al fichero o directorio original por su ruta.

Si se borra el original, el enlace sigue existiendo aunque no funciona (no apunta a nada).

Un enlace es un fichero y se borra con rm (también si apunta a un directorio).

Similares a los 'shortcuts' de Windows.

* Enlaces 'duros' o físicos ('hard links')

ln fichero_original nombre_enlace_simbolico

Establecen un segundo nombre para el fichero original.

No se permiten enlaces duros para directorios (excepto al superusuario, con restricciones).

El fichero enlazado debe estar en la misma partición.

Comandos de Salida en Pantalla

* Salida simple (no formateada)

echo Hola

Para suprimir el carácter nueva línea al final:

echo -n Hola

* Para salida formateada:

printf

* Para concatenar el contenido de ficheros y sacarlos a pantalla (salida estándar):

cat fichero1 fichero2

Con un solo fichero, cat envía dicho fichero a stdout

cat fichero

* Para sacar contenidos de ficheros pantalla a pantalla:

more

less

* Para sacar las primeras líneas de ficheros (por defecto, 10 líneas; se puede indicar el número con la opción -n número_de_líneas):

head

head -n 15

* Para sacar las últimas líneas de ficheros (por defecto, 10 líneas; se puede indicar el número con la opción -n número_de_líneas):

tail

tail -n 15

* Para sacar en pantalla una cadena de caracteres ('string') indefinidamente (por defecto: 'y'):

yes

* Para ordenar líneas de ficheros:

sort

* Para seleccionar campos de la entrada:

cut

*Para especificar el segundo campo, con delimitador ',':

cut -f 2 -d ‘,'

* Para obtener el número de líneas, palabras y bytes de los ficheros de entrada:

wc

* Para comparar dos ficheros línea a línea:

diff

* Para convertir un conjunto de caracteres en otro de entrada estándar:

tr

Variables de Entorno

* Las **variables de entorno** son aquellas normalmente 'heredadadas' cuando un terminal se lanza.

* Se pueden sacar todas en pantalla con el comando:

```
printenv
```

Y se puede sacar el contenido de una variable con el comando 'echo', precediendo el nombre de la variable con '$' para acceder a su contenido:

```
echo $HOME
```

Puede ser necesario delimitar el nombre de la variable entre llaves (para resolver ambigüedades y delimitar el identificador):

```
$HOME
${HOME}
```

* Algunas variables de entorno importantes son las siguientes:

HOME: directorio inicial al hacer login.
PATH: directorios donde el sistema busca programas ejecutables. Nota: separados por ':'.

```
PATH=$HOME/bin:$PATH
```

El orden de búsqueda en el PATH es de izquierda a derecha, por lo que, si se añade por la izquierda, el sistema operativo buscará primero en el nuevo directorio añadido.

LOGNAME: nombre de usuario (establecido en login).
USER: nombre de usuario (establecido en login).
SHELL: el ejecutable del shell de la cuenta (por ejemplo, /bin/bash).
PWD: el directorio actual (establecido por el comando cd).

OLDPWD: directorio previo (previo al último comando cd).
HOSTNAME: nombre del equipo.
MAIL: el fichero de correo entrante, con su ruta completa.

EDITOR: el editor empleado por algunas utilidades (por ejemplo, vim).
TERM: tipo del terminal (p.ej.: xterm o vt100).

Otras Variables del 'Shell'

* Variables **establecidas por el shell** y normalmente utilizadas dentro de 'scripts'.

* Existen algunas variables internas importantes que son de sólo lectura ('read only').

Valor de salida del último comando ejecutado: ?
Variable por defecto utilizada por 'read': REPLY
Número de proceso: $
Número de proceso del último comando en segundo plano ('background')

Nota: el carácter '$' previo al nombre de la variable la expande a su valor:

$?

$$

$!

* Se verán más adelante las variables posicionales de los argumentos de 'scripts' y funciones.

Variables

* Además de las variables ya existentes en el shell, se pueden definir nuevas variables y operar con ellas.

* Se pueden definir variables a la vez que que se asignan. El contenido

se accede precediendo el nombre de la variable con '$':

```
MIVAR=libro

echo $MIVAR
Salida: libro


echo "Se evalúa \$MIVAR: [$MIVAR]"
Salida: Se evalúa $MIVAR: [libro]


MIFRASE="Me gusta este $MIVAR"

echo $MIFRASE
Salida: Me gusta este libro
```

Las dobles comillas pueden evaluar contenidos de variables y ciertas expresiones.

* En ocasiones es necesario delimitar el identificador con llaves:

```
echo ${MIVAR}s
```

Sin llaves, bash entendería como nombre de variable 'MIVARs'.

* Si se utiliza una variable que no ha sido definida, no se produce error.

```
echo $VARIABLE_NO_EXISTE
```

Es en general recomendable utilizar dobles comillas cuando una variable se pase como argumento y pueda ser vacía.

Nota: se puede forzar un error si se utiliza una variable no definida, y se puede indicar el mensaje de error asociado, si se desea:

Lo anterior da error, y el texto 'ERROR: no definida.' aparece en el mensaje de error producido.

* Para declarar en el terminal variables y que sean visibles o exportadas a nuevos procesos 'hijo' (variables de entorno que modifican el entorno):

Con ‘export':

Otra forma:

Para 'desactivar' que una variable de entorno sea de entorno:

declare +x VAR_A_EXPORTAR

Nota: para comprobar los atributos de una variable:

declare -p VARIABLE

*Se puede modificar o crear una variable en la misma llamada a un programa, y sólo afectará la ejecución de dicho programa:

HOME='/tmp' cd

Después de ejecutar programa, el contenido de HOME será el que había previamente.

* Pasar una variable vacía (una cadena de longitud 0) a un comando puede dar resultados problemáticos,según el caso.

```
# Variable vacía  MIVAR:  comando $MIVAR
# Mejor es poner la variable entre dobles comillas "$MIVAR"
```

* Se puede eliminar el valor de una variable con:

unset nombre de la variable

Strings, Comas y Entrecomillados ('Quoting')

* Las comillas permiten agrupar palabras e incluir espacios en nombres.

Nota: El espacio es el separador común de expresiones para el shell. Sin embargo, se agrupan ambas palabras con comillas:

Si se quiere continuar un comando en la siguiente línea, se hace generalmente escribiendo el carácter escape antes del salto de línea.

```
ls -ld /tmp \
/usr \
/usr/bin
```

* Las comillas dobles permiten cierta evaluación de construcciones del shell como la evaluación de variables.

Comillas dobles “…”:

Nota: caracteres especiales (como '$', '"', etc.) se deben preceder con '\' (carácter 'escape').

*Comillas simples '...': se toma el entrecomillado como algo literal, sin evaluaciones del contenido.

Nota: con comillas simples, para introducir dentro del texto a su vez comillas simples.

* Concatenación de strings

Ej: CAD3=“${CAD1}${CAD2}"

Nota: Otra forma, utilizando el operador '+=' que concatena añadiendo por el final:

* La salida estándar de un comando se puede capturar con la construcción: $(…)

* Otra forma es con comillas inversas `...` ('backquoting'):

* Para expresiones numéricas se tiene la siguiente sustitución echo
$((3 + 4))

*Para expresiones aritméticas sencillas, es emplear el comando 'expr'

* Por defecto, las variables se consideran ‘strings’.

Expansión en el 'Shell'

* El 'shell' realiza cierto procesamiento de la línea de comandos.

En una llamada a un comando, el 'shell' procesa la llamada y los argumentos antes de ejecutar el comando.

Nota: sería distinto con comillas (no habría expansión):

```
ls "*.txt"
ls '*.txt'
```

```
echo a*
```

Hay expansión en el primer caso. Es decir, si hay ficheros que comienzan por 'a', echo a* los sacará en pantalla. En el resto de los casos, se saca en pantalla: a*

* La expansión de corchetes (‘brace expansion’) no se ordena.

Valores Devueltos por Comandos.
'Éxito' / ‘Fracaso'

* la ejecución exitosa de un programa tiene normalmente asociado un valor de retorno de 0.

* El resultado de un comando se guarda en la variable ?. Para ver su contenido:

echo $?

Los números distintos de 0 se consideran como códigos de error.

Nota: en Unix, se suelen considerar en lo posible códigos de error normalizados .

/usr/include/sysexits.h

* El comando 'true' simplemente devuelve un resultado de éxito (0).

Se 'niega' con '!' :

* El comando 'false' simplemente devuelve un resultado de fracaso (=1).

Se 'niega' con '!' :

* Comando nulo ':' no hace nada. El status que devuelve es de éxito:

Nota: el resultado de un comando es diferente a su salida.

true

Ejecución de Comandos

* Variantes de secuenciación de comandos:

Secuencia normal:

comando1


```
comando2
comando3
```

También, en una línea:

```
comando1 ; comando2 ; comando3
```

Secuencia 'cortocircuitada' con '&&' ('y' lógico):

```
comando1 && comando2 && comando3
```

Secuencia 'cortocircuitada' con '||' ('o' lógico):

```
comando1 || comando2 || comando3
```

Los operadores lógicos && y || se dice que 'cortocircuitan'.

No se evalúan necesariamente todos los argumentos, y si el resultado de la operación lógica ya se ha establecido no se evalúan el resto de los argumentos.

Evaluación 'perezosa' (en contraposición a evaluación impaciente).

Nota: este tipo de evaluación de los operadores lógicos es lo normal en los lenguajes de programación.

* Las llaves '{ ... ; }' permiten agrupar una lista de comandos en un bloque.

Se puede redireccionar la salida del bloque en su conjunto.

Nota: Se necesita el carácter ';' al finalizar la lista de comandos del bloque.

Debe haber espacios alrededor de los caracteres '{', '}'.

```
# Nuevo valor de variable en el bloque, y después del bloque.
    C='c'; echo $C; { echo "(en bloque) $C"; C='C'; echo "(en bloque) $C" ; } ; echo $C;
```

Salida:
c
(en bloque) c
(en bloque) C
C

* Los paréntesis '(...)' agrupan un bloque de comandos que se ejecutan en un subshell (nuevo proceso).

Se puede redireccionar la salida del bloque en su conjunto.

Nota: un subshell hereda variables del shell externo (además de las de entorno).

# Nuevo valor de variable en el subshell, aunque no cambia de valor en el shell externo.
C='c'; echo $C; (echo "(en bloque) $C"; C='C'; echo "(en bloque) $C") ; echo $C;

Salida:
c
(en bloque) c
(en bloque) C
c

* Ejemplo: si se quiere combinar el cambiar de directorio y lanzar un comando en dicho directorio destino, es aconsejable combinar con '&&':

```
cd Dir1/Dir2 &&
```

* Ejemplo: para cambiar de directorio a /tmp y lanzar 'ls -l' dos posibilidades son:

```
cd /usr && ls -l
```

Nota: con construcción condicional if-then:

```
if cd /usr
then
  ls -l
fi
```

La versión con composición con '&&' es más concisa.

Entrada

* Un comando que lee una línea: 'read'

Para finalizar la cadena introducida, se teclea ENTER

```
echo -n 'Teclee algo: ';
read MIENTRADA;
echo "Se ha tecleado: $MIENTRADA"
```

Lee una línea y la separa (con el espacio como separador):

```
echo -n 'Teclee algo: ';
read ;
echo "Se ha tecleado: $REPLY"
```

Nota: con la opción '-p' se puede indicar una cadena de caracteres a escribir antes de la lectura:

```
read -p 'Teclee algo: '; echo "Se ha tecleado: $REPLY”
```

Entrada/Salida y Redireccionamiento

* Existen tres descriptores de fichero predefinidos

Entrada estándar 'stdin': por defecto, por teclado.
Número de descriptor asociado: 0.

Salida estándar 'stdout': por defecto, a pantalla.
Número de descriptor asociado: 1.

Salida error 'stderr': por defecto, a pantalla.
Número de descriptor asociado: 2.

* Redireccionar salida estándar 'stdout' a ‘fichero.txt’:

comando > fichero.txt

Nota: si el fichero 'fichero.txt' existía, se crea de nuevo (no se conserva su contenido anterior).

Ídem: comando 1> fichero.txt

Redireccionar y añadir ('append') salida estándar 'stdout' a 'fichero.txt':

comando >> fichero.txt

Nota: el contenido previo de 'fichero.txt' que pudiera existir se conserva, y el nuevo texto se añade al final.

Redireccionar salida error 'stderr' a 'fichero.txt':

comando 2> fichero.txt

Redireccionar y añadir salida error 'stderr' a 'fichero.txt':

comando 2>> fichero.txt

Redireccionar la salida estándar 'stdout' a 'fichero_salida.txt' y salida error 'stderr' a 'fichero_error.txt':

comando > fichero_salida.txt 2> fichero_error.txt

Redireccionar ambas salida estándar 'stdout' y salida error 'stderr' a 'fichero.txt':

comando &> fichero.txt

Nota: las dos partes de salida 'stdout' y 'stderr' se mezclan, y en principio el orden de las líneas de ambas salidas no está determinado.

Redireccionar y añadir ambas salida estándar 'stdout' y salida error 'stderr' a 'fichero.txt':

comando &>> fichero.txt

Redireccionar la salida estándar 'stdout' a la salida error 'stderr':

```
comando 1>&2
```

Redireccionar la salida error 'stderr' a la salida estándar 'stdout':

```
comando 2>&1
```

Redireccionar la entrada estándar para leer de un fichero:

```
comando < fichero
```

Ídem: `comando 0< fichero`

* Existen los siguientes ficheros dispositivo asociados a stdout, stderr y stdin

```
/dev/stdout
/dev/stderr
/dev/stdin
```

* Ejemplo:

```
El siguiente 'echo' saca en pantalla 'Hola':
  echo Hola
Es equivalente a:
  echo Hola > /dev/stdout
```

El dispositivo asociado a salida estándar 'stdout' se considera como un fichero.

Para redireccionar la salida a un fichero 'salida.txt':

```
echo Hola > salida.txt
echo Hola 2> error.txt
```

entonces:

- La salida estándar 'stdin' sigue yendo a pantalla.
- El error estándar 'stderr' se direcciona al fichero 'error.txt'

Nota: como en este caso no se genera en principio salida de error, el fichero 'error.txt' se crea aunque es vacío.

El comando 'echo' escribe en salida estándar 'stdin', y para redireccionarla a la salida error 'stderr':

```
echo Hola 1>&2
```

La cadena 'Hola' sale a la salida error 'stderr'. Esto puede ser útil, por ejemplo, en 'scripts' que quieran sacar mensajes de error a la salida error 'stderr' (en vez de 'stdout')

También se puede hacer, para redireccionar salida con echo a 'stderr':

```
echo Hola > /dev/stderr
```

* Nota: es posible 'eliminar' fácilmente la salida de un comando redirigiendo a '/dev/null':

```
ls /dir_que_no_existe 1> /dev/null  # para 'eliminar' la salida estándar
ls /dir_que_no_existe 2> /dev/null  # para 'eliminar' la salida error
ls /dir_que_no_existe &> /dev/null  # para 'eliminar' ambas salidas
```

* Un comando puede generar salida tanto a 'stdin' como a 'stderr', diferenciando la salida normal de posibles mensajes de error.

Ejemplo con 'ls' (primero se crea un fichero con 'touch'):

```
touch /tmp/prueba.txt
ls /tmp/prueba.txt fichero_que_no_existe
```

En pantalla aparece:

```
ls: no se puede acceder a fichero_que_no_existe: No existe el archivo o el directorio  /tmp/prueba.txt
```

Se pueden distinguir dos partes en lo que aparece en pantalla:

La información proporcionada por 'ls' sobre '/tmp/prueba.txt' va a stdout.
La información del error sobre el fichero inexistente va a salida error.

Dichas salidas se pueden redirigir como se ha comentado antes:

```
ls /tmp/prueba.txt fichero_que_no_existe > /tmp/salida.txt
ls /tmp/prueba.txt fichero_que_no_existe 2> /tmp/error.txt
```

ls /tmp/prueba.txt fichero_que_no_existe > /tmp/salida.txt 2> /tmp/error.txt
ls /tmp/prueba.txt fichero_que_no_existe &> /tmp/salida_y_error.txt

Redirigiendo la entrada estándar con '<':

ls /tmp > /tmp/ls_salida.txt
head < /tmp/ls_salida.txt
Salida: las primeras 10 líneas de /tmp/ls_salida.txt

* 'Pipes' (Tuberías): permite conectar la salida estándar de un comando con la entrada estándar de otro.

comando1 | comando2

La salida estándar del comando1 se envía como entrada estándar del comando2.

Ej.: para obtener la salida de ls en orden inverso (en orden inverso de diccionario). En orden aleatorio:
ls | sort -R
Ej.: para obtener las primeras líneas de un fichero en orden aleatorio:
head nombre_de_fichero | sort -R
También se pueden combinar varios pipes:
cat nombre_de_fichero | head | sort -R

Ej.: Para mostrar los usuarios en un sistema (primer campo en fichero /etc/passwd):
cat /etc/passwd | cut -f 1 -d ':'
Ej.: número de ficheros y subdirectorios en el directorio actual:
ls | wc -l

* Convertir las vocales minúsculas en mayúsculas con 'tr' :

Ej.: de un fichero fich.txt :
cat fich.txt | tr 'aeiou' 'AEIOU'
Ej.: de la salida de un comando como 'ls' :
ls /tmp | tr 'aeiou' 'AEIOU'

* Cuando finaliza el último comando del pipe, finalizan los previos:

Ej.: en el siguiente pipe, el comando 'yes' finaliza cuando acaba el comando 'head' de sacar en pantalla 10 líneas:

yes | head

* Para utilizar 'more' o 'less' cuando la salida de un comando es larga:

* Los pipes son generalmente eficientes.

En principio más eficientes que redirigir salidas a ficheros temporales y combinarlos.

* La mayor parte de las utilidades Unix utilizan la entrada estándar, la salida estándar y la salida error.

Simplifican frecuentemente su utilización y la combinación de diversos comandos.

* Un comando útil para leer de la entrada estándar y escribir (a la vez) a la salida estándar y a ficheros:

tee

Ej.: en la siguiente secuencia, 'tee' envía el contenido de fich1.txt a fich2.txt y, además, a la salida entándar:

```
cat fich1.txt | tee fich2.txt
```

Algunos Comandos de Fechas

* Comando básico: date

La salida se puede formatear.

* Para información de fecha hasta de días:

```
date '+ %Y %m %d'
```

También concatenando tres comandos date:

```
$(date +%Y)$(date +%m)$(date +%d)
```

Con formato separando con el carácter '/':

```
date '+ %Y / %m / %d'
```

* Fecha según el formato local (en principio, dd/mm/yy):

```
date '+%x'
```

* Fecha según el formato %Y-%m-%d :

```
date '+%Y-%m-%d'
```

También, de forma más concisa:

```
date '+%F'
```

* Para información de fecha hasta minutos:

```
date '+%Y%m%d%H%M'
```

* Para información de fecha hasta segundos:

```
date '+%Y%m%d%H%M%S'
```

* Se puede cambiar el formato de una fecha concreta:

```
date -d '2015-03-09' '+%Y/%m/%d'
```
Salida: 2015/03/09

* Nota: el comando date permite realizar comprobaciones generales sobre fechas en formatos estándares.

Ejemplos:

```
# Formato fecha válido
date -d '2014-12-05'; echo $?
```

Salida:
0

Formato fecha no válido
date -d 'esto_no_es_una_fecha'; echo $?
Salida:
1

Comando 'test'

* El **comando 'test'** permite evaluar expresiones y devolver status de 0 (verdadero o éxito) ó de 1 (falso o fracaso).

test ...

Sinónimo: [...]

Nota: importante dejar espacios entre los argumentos, y alrededor de '[' y ']'.

Para consultar la documentación:

help test

Inmediatamente después del comando test, se hace:

echo $?
la salida es: 0

* Tests de ficheros:

Si nombre es un directorio:	-d nombre
Si nombre es un fichero:	-f nombre
Si nombre es un enlace simbólico:	-L nombre
Si existe y es legible:	-r nombre

Si existe y es modificable:	-w nombre
Si existe y es ejecutable:	-x nombre
Si existe y su tamaño es no nulo:	-s nombre
Si n1 es más reciente que n2:	n1 -nt n2
Si n1 es más antiguo que n2:	n1 -ot n2

* Tests de cadenas de caracteres ('strings'):

Si el string s1 es igual al string s2:	s1 = s2
Si el string s1 no es igual al string s2:	s1 != s2
Comparaciones lexicográficas de strings:	s1 \< s2
(en versión test comando interno.)	s1 \> s2

Nota: se debe poner escape antes de '<', '>', ya que son caracteres de redireccionamiento.

Si el string s1 tiene longitud 0:	-z s1
Si el string s1 tiene longitud no nula:	-n s1

* Tests numéricos:

Si los enteros n y m son iguales:	n -eq m
Si los enteros n y m no son iguales:	n -ne m
Si el enteros n es mayor que el entero m:	n -gt m
Si el enteros n es mayor o igual que el entero m:	n -ge m
Si el enteros n es menor que el entero m:	n -lt m
Si el enteros n es menor o igual que el entero m:	n -le m

* Composición de tests:

Si test t1 y test t2 son ciertos ('y' lógico):	t1 -a t2
Si test t1 o test t2 son ciertos ('o' lógico):	t1 -o t2
Negación de test t1:	! t1
Los paréntesis permiten agrupar tests:	\(... \)

* Nota: cuidado que los tests dan verdadero con una expresión vacía (lo que conlleva a que se llame a test sin argumento. Lo anterior ocurre si una variable es vacía o está indefinida y no aparece con **dobles comillas.**

Se puede comprobar previamente si una variable es o no nula (tests -z y -n). Es también recomendable poner la variable en dobles comillas cuando se aplican los tests -z y -n si la variable puede contener una lista de elementos.

* Comando '[[...]]' :

Comando diferente.
No existente inicialmente en bash
Menor expansión de palabras y expresiones regulares
Menor necesidad de entrecomillado).
No se hace 'escape' en los paréntesis para agrupar condiciones.
Para componer condiciones con 'and' lógico:&&
Para componer condiciones con 'or' lógico: ||

* Nota: en principio se empleará el comando clásico test o [...] visto anteriormente.

'Scripts'

* Es un programa en texto que, generalmente, es interpretado, y que se enfoca, sobre todo en el caso de bash, a interactuar con el sistema operativo y con el usuario.

Comienzan por el carácter '#'.

* La primera línea es especial: indica el ejecutable (el shell bash) asociado al script, con su ruta completa:

#!/bin/bash

Nota: la ruta concreta de bash depende del sistema, y se debe poner la ruta indicada por el el comando: which bash

* También se puede encontrar:

#!/bin/sh

El conjunto de caracteres '#!' qué precede el ejecutable se suele denominar en inglés 'shebang'. Y la línea con '#!/bin/bash' se suele indicar en inglés como la 'shebang line'.

* En scripts no interactivos, se puede emplear /bin/sh en vez de /bin/bash por eficiencia (aunque se pierden funcionalidades). Actualmente, en Linux el fichero '/bin/sh' apunta a /bin/dash .

* Se puede considerar que dash es un subconjunto de bash. El intérprete de comandos 'dash' es más ligero que ‘bash'.

* Formas de crear y ejecutar 'scripts' más típicas:

1. Crear un fichero 'script' cuya primera línea es:

#!/bin/bash. O sh: #!/bin/sh

Hacer el fichero script ejecutable:

chmod +x nombre_fichero_script

El fichero script tenga la extensión .sh para facilitar su identificación como un script.

Ejecutar el script:

./nombre_fichero_script

Nota: si el script está en un directorio contenido en el PATH no haría falta indicar la ruta.

Nota: para depurar o desarrollar, se puede añadir la opción '-x' que activa la opción 'xtrace' para que se muestren los comandos y argumentos según se van ejecutando:

#!/bin/bash -x

El script se ejecuta en un proceso nuevo (en un shell nuevo).

2. Ejecutar el script pasándolo a bash:

```
bash nombre_fichero_script
```

El script se ejecuta en un proceso nuevo (en un shell nuevo).

3. Ejecutar en el propio shell del terminal las instrucciones del fichero script:

```
source nombre_fichero_script    O    . nombre_fichero_script
```

Con esta forma no se lanza el script, sino que las instrucciones del script se ejecutan en el shell existente (no se lanza uno nuevo).

Para consultar la documentación:

```
help source
```

Cuidado si el script tiene 'exit' ya que al hacer source del script provocaría la salida del shell.

* Nota: asimismo, se puede mencionar que es posible lanzar un bash pasándole comandos en un string con la opción '-c' :

* El comando 'exit' termina la ejecución del proceso

Permite devolver un valor status.

Nota: un script puede finalizar sin lanzar 'exit', en cuyo caso el valor devuelto por el script es el del último comando.

* Para interrumpir la ejecución de un comando en un terminal:

```
^C  (Control-C)
```

'Scripts' y Argumentos

* Las siguientes variables se emplean, sobre todo, en 'scripts'.

$0 : el nombre del comando, es decir, la primera palabra en una llamada en línea de comandos.

$1, $2, ... : argumentos posicionales en la línea de comandos.

Nota: para el argumento en posición 10 (y siguientes) se delimita el número entre llaves: ${10}

$# : Número de argumentos de la llamada en línea de comandos.

$*, $@ : todos los argumentos en la línea de comandos ($1 $2 ...).

"$@" : todos los argumentos en la línea de comandos, cada uno de ellos entrecomillado por separado ("$1" "$2" …).

"$*" : todos los argumentos en la línea de comandos como una sola cadena de caracteres ("$1 $2 …").

Funciones y Argumentos

* Sintaxis de función:

```
mifuncion() {
  ...
} [redirecciones]
```

Nota: otra variante en sintaxis:

```
function mifuncion {
  ...
} [redirecciones]
```

* Argumentos posicionales

El caso de los argumentos de funciones, con argumentos posicionales análogos. Las posiciones se refieren a los argumentos de la llamada a la función

Ej.: función que hace echo de dos argumentos posicionales:

```
miecho_2_args() {
  echo $1
  echo $2
  echo $#
}


miecho_2_args uno dos

Salida:
uno
dos
2
```

* El comando 'return' acaba la ejecución dentro de la función.

return número : acaba la función devolviendo como resultado o status de la función el valor número. si se omite número, se devuelve el resultado o status de la última orden ejecutada.

El comando 'return' es opcional.

* Se pueden definir variables locales a la función (no visibles fuera)con 'local'.

* Las funciones pueden ser recursivas.

Programación con 'Scripts'

* Condicionales 'if-then', 'if-then-else', 'if-then-elif-else' :

```
if comando
then
  # Parte 'then' si el resultado del comando es 0
  instrucciones y comandos
fi
```

Para consultar la documentación:

```
help if
```

* Condicional 'case'

```
#!/bin/bash
echo 'Elija destino (Segovia, Soria, Palma de Mallorca, Valencia):'
read DESTINO
case "$DESTINO" in
  Segovia)
    echo "$DESTINO no tiene puerto."
    ;;
  Soria)
    echo "$DESTINO no tiene puerto."
    ;;
  'Palma de Mallorca')
    echo "$DESTINO tiene puerto."
    ;;
  Valencia)
    echo "$DESTINO tiene puerto."
    ;;
  *)
    echo "$DESTINO no es un destino reconocido."
    ;;
esac
exit
```

* Bucle for:

Permite iterar fácilmente sobre los valores de una lista:

```
for variable in lista
do
  instrucciones y comandos
done
```

* Bucle 'while'

```
while command
do
  instrucciones y comandos
done
```

* Bucle until:

```
until command
do
  instrucciones y comandos
done
```

* Bucles ‘infinitos': para acabar, teclear ^C (Control-C).
* El comando 'break' sale del bucle más cercano.
* El comando 'continue' continúa en la siguiente iteración.

Procesos y Variables

* Variable $

Número de proceso ('process ID') del shell.

En un subshell, es el número de proceso del shell padre que lo invoca.

* Variable BASHPID

Número de proceso ('process ID') del proceso bash actual.

Al lanzar un subshell, la variable BASHPID proporciona el número del nuevo proceso subshell hijo:

```
echo "\$\$: $$; \$BASHPID: $BASHPID" ; ( echo "En subshell: \$\$: $$; \
$BASHPID: $BASHPID" )
```

Nota: sería diferente en comandos agrupados en un bloque delimitado por llaves '{ }' (donde no se lanza nuevo proceso):

```
echo "\$\$: $$; \$BASHPID: $BASHPID" ; { echo "En bloque: \$\$: $$; \
$BASHPID: $BASHPID"; }
```

Control de Tareas y Procesos

* Para mostrar las tareas: jobs
* Para ejecutar un proceso en segundo plano ('background'):

Añadir al final: &

* Suspender la tarea actual en primer plano: ^Z (Control Z)
* Para pasar un proceso suspendido o con ejecución en segundo plano ('background') al primer plano ('foreground'):

```
fg

fg %número_de_proceso
```

Nota: el número de proceso según indica el comando jobs.
Si no se indica el número de proceso, el comando actúa sobre el proceso suspendido o en segundo plano más reciente.

* Para pasar procesos suspendidos a ejecución en segundo plano ('background'), es decir, como si se hubieran lanzado con '&' :

bg

bg % número_de_proceso

Nota: el número de proceso según indica el comando jobs.
Si no se indica el número de proceso, el comando actúa sobre el proceso suspendido o en segundo plano más reciente.

* Para interrumpir la ejecución de un comando:

- Si es un proceso que se está ejecutando en el terminal, se puede interrumpir con:

^C (Control-C)

Nota: existe el comando 'reset' para reiniciar el terminal.

Nota: a veces puede que la tecla <ENTER> no funcione adecuadamente. Se puede probar ^J (Control-J).

- Para todo tipo de proceso, se puede obtener el número de proceso con 'ps' y luego se interrumpe con 'kill':

ps aux | grep nombre_de_proceso

kill -HUP número_de_proceso_obtenido_con_el_comando_anterior

Existe otra opción más 'brusca' de señal de kill, si acaso no funciona la señal -HUP:

kill -KILL número_de_proceso O kill -9 número_de_proceso

* Otros comandos de procesos:

ayudar a averiguar el estado de tu servidor, proporcionándote información

top

facilita información sobre la finalización de una serie de procesos relacionados entre sí:

pstree

* Para acabar un shell: exit

También se puede utilizar logout si el shell es de entrada (de login).

* Para apagar el sistema:

shutdown

* Para controlar el flujo a un terminal:

^S (Control-S) para el envío de caracteres a la pantalla.

Nota: se puede configurar y desactivar ('stty').

^Q (Control-Q) para desbloquear el flujo.

Procesos Lanzados

* Se puede lanzar una secuencia de comandos en un subshell (proceso 'hijo' nuevo).

* En script, puede haber problemas si se lanza un proceso en segundo plano ('background') y el script finaliza:

Nota: también se puede especificar a 'wait' un proceso determinado, para esperar sólo dicho proceso.

* Existen otros comandos

El comando exec reemplaza el shell con un comando (externo):

exec comando

Cuando acaba la ejecución del comando, no se vuelve al shell que lo lanzó (que no existiría).

Histórico de comandos - Comando 'history'

* En línea de comandos, se pueden recuperar los comandos previos con las flechas del teclado hacia arriba y hacia abajo.

* Así mismo, se pueden reutilizar comandos previos con:

Para ejecutar de nuevo el último comando: !!
Para ejecutar de nuevo el comando número N en el histórico: !N
Para ejecutar de nuevo el comando ejecutado N veces antes: !-N

* Para reutilizar argumentos del último comando:

El primer argumento se denota por: !^
El último argumento se denota por: !$
Todos los argumentos se denotan por: !*

Ficheros de Configuración del 'Shell'

* Permiten configurar opciones del shell, así como introducir nuevas funciones(documentación en 'man bash').

* En Bash, en shell de entrada (de 'login'):

1. Se ejecuta primero el siguiente fichero del sistema:

/etc/profile

2. Se ejecuta el primero que exista de entre los siguientes ficheros de usuario:

~/.bash_profile
~/.bash_login
~/.profile

3. Cuando el shell de login finaliza, bash ejecuta el fichero (si existe).

~/.bash_logout

Nota: las inicializaciones 1 y 2 se pueden evitar si el shell bash se lanza con la opción :

'--noprofile'

* En Bash, en shell no de entrada. Se ejecuta (si existen):

/etc/bash.bashrc
~/.bashrc

Nota: se puede evitar si el shell bash se lanza con opción '--norc' .

* Si se modifica ~/.bashrc, se puede ejecutar su contenido con:

source ~/.bashrc O . ~/.bashrc

* Nota: si se invoca sh, se ejecuta el fichero contenido en la variable ENV o bien el ejecutable sh .

Sobre Entrada/Salida, Redireccionamiento y EOT mediante el comando 'cat' :

* Ejemplo:

```
cat > fich.txt
<escribir en el terminal algunas líneas>
^D
```

El programa cat ha recogido las líneas introducidas por la entrada estándar, y las ha enviado por la salida estándar, la cual ha sido redirigida al fichero fich.txt .

Teclear ^D (Control-D), que significa el carácter EOT ('End of Transmission'), hace que:

-el terminal envíe el buffer donde va 'recolectando' los bytes de los caracteres tecleados.
-el programa cat reciba una lectura de 0 bytes, la cual se considera como fin de fichero EOF ('End of File'), que hace que cat finalice).

Y si ese buffer está vacío (porque se ha vaciado al teclear previamente ^D, o bien se ha tecleado nueva línea y estamos al inicio de una línea nueva),

Nota: si se quiere introducir en un fichero un carácter de control y se quiere incluir el carácter en sí sin que se interprete como carácter de control (para ser tratados como un literal),Teclear ^V (Control-V) previamente al mandato.

^V^D

Permisos Especiales

* Extensión GNU.

* SUID ('Set User Identification')

  En ficheros ejecutables, permite ejecutar un fichero como si el usuario efectivo que lo está ejecutando fuera el usuario propietario del fichero.

La 's' hace referencia a sí tiene activado el SUID. si en vez de 's' aparece 'S', eso significa que el SUID está activado, aunque el permiso de ejecución del fichero no está habilitado.

Para activar el SUID: chmod 4555 fichero

* SGID ('Set Group Identification')

Aplicado al grupo en vez de al usuario.

Si habilitado, el grupo efectivo con el que se ejecuta un fichero ejecutable es el del fichero (y no el del usuario que lanza el ejecutable).

Si se aplica a un directorio, entonces los ficheros creados en dicho directorio tienen asociado el grupo del directorio (en vez del grupo de quien crea el fichero).

Para activarlo: chmod 2555 directorio

También, para activarlo en un directorio: chmod g+s directorio

* **'Sticky bit':** Si un directorio tiene activado el 'sticky bit', entonces los ficheros creados en dicho directorio sólo pueden ser borrados por el propietario del fichero o por root.

Para activarlo (a la vez que se dan permisos rwx a todos con 777):

chmod 1777 directorio

Asimismo, para activarlo en notación simbólica:

chmod +t directorio

Otros Casos de Uso de Sentencia 'case'

caso con expresión regular:

Además de los terminadores ';;' para cada caso que se ejecuta por separado existen los separadores:

';&' : para que se ejecuten los comandos del siguiente caso.

';;&' : para que se compruebe la condición del siguiente caso.

Opciones del Shell - Opción '-x' para Depuración

* El Comando 'set' Para imprimir en pantalla las opciones:

set -o

* Para depuración, se pueden mostrar los comandos y argumentos según se van ejecutando.

Para activar la opción **'xtrace'**: set -x o set -o xtrace

Para desactivar una opción: set +x

También, se puede activar 'xtrace' en la llamada al shell:

bash -x script.bash

Asimismo, en la línea de un script que indica el ejecutable:

#! /bin/bash -x

Sobre Comando 'expr'

* Se recuerda que, por defecto, las variables se consideran como cadenas de caracteres ('strings').El comando 'expr' permite evaluar expresiones sencillas, escribiendo el resultado en la salida estándar.

Nota: cuidado con los espacios.

* Otra forma de operar numéricamente, con la sustitución numérica $((...)):

En general, se puede considerar más recomendable que ‘expr'.

Sustitución de Variables y Procesado de Cadenas de Caracteres ('Strings')

* En lo que sigue se incluyen algunas de las más empleadas.

* Subcadenas de variables

Parte de la cadena de una variable que empieza en posición 'pos' :

${NOMBRE_VAR:pos}

Nota: la primera posición es la de índice 0.

Parte de la cadena de una variable que empieza en posición 'pos' y de longitud 'long' :

${NOMBRE_VAR:pos:long}

Ej.:

```
CAD='abcdefgh'
echo "\$CAD: $CAD"
echo "\${CAD:2}: ${CAD:2}"
echo "\${CAD:2:3}: ${CAD:2:3}"
```

Salida:
$CAD: abcdefgh
${CAD:2}: cdefgh
${CAD:2:3}: cde

* Devuelve la longitud del contenido de la variable 'NOMBRE_VAR' :

${#NOMBRE_VAR}

Ej.:

```
NOMBRE='Alberto'
echo ${#NOMBRE}
```

Salida:
7

Nota: las llaves son necesarias para indicar el nombre de la variable. (Si no, se interpretaría como '$#' .)

* Si la variable 'NOMBRE_VAR' tiene un valor, devuelve dicho valor; en otro caso, devuelve 'cadena':

${NOMBRE_VAR:-cadena}

Ej.:

```
miecho() {
  local CADENA=${1:-Hola}
  echo $CADENA
}
```

miecho abc
Salida:
abc

miecho
Salida:
Hola

* Si la variable 'NOMBRE_VAR' tiene un valor, devuelve dicho valor; en otro caso, devuelve 'cadena' y asigna 'cadena' a la variable 'NOMBRE_VAR':

${NOMBRE_VAR:=cadena}

* Si la variable 'NOMBRE_VAR' tiene un valor, devuelve dicho valor; en otro caso, se imprime mensaje de error (que incluye 'cadena') y se hace 'exit'.

${NOMBRE_VAR:?cadena}

* Si la variable 'NOMBRE_VAR' tiene un valor, devuelve dicho valor; en otro caso, no devuelve nada:

${NOMBRE_VAR:+cadena}

* Devuelve el valor del contenido de la variable 'NOMBRE_VAR' después de eliminar por la izquierda la parte que ajuste a 'patron' (se elimina la parte más pequeña que ajusta):

${NOMBRE_VAR#patron}

Ej.:

```
MIFICH='MiDir/Subdir/prueba.txt'; echo ${MIFICH#*/}

Salida:
Subdir/prueba.txt
```

Ej.:

```
MIFICH='/tmp/Dir/prueba.txt'; echo ${MIFICH#*/}

Salida:
tmp/Dir/prueba.txt
```

* Devuelve el valor del contenido de la variable 'NOMBRE_VAR'

después de eliminar por la izquierda la parte que ajuste a 'patron' (se elimina la parte más larga que ajusta):

${NOMBRE_VAR##patron}

Ej.: similar a la salida de 'basename' (aunque con posibles problemas de portabilidad):
MIFICH='/tmp/Dir/prueba.txt'; echo ${MIFICH##*/}

Salida:
prueba.txt

* Devuelve el valor del contenido de la variable 'NOMBRE_VAR' después de eliminar por la derecha la parte que ajuste a 'patron' (se elimina la parte más pequeña que ajusta):

${NOMBRE_VAR%patron}

Ej.: similar a la salida de 'dirname' (aunque con posibles problemas de portabilidad):
MIFICH='/tmp/Dir/prueba.txt'; echo ${MIFICH%/*}

Salida:
/tmp/Dir

* Devuelve el valor del contenido de la variable 'NOMBRE_VAR' después de eliminar por la derecha la parte que ajuste a 'patron' (se elimina la parte más larga que ajusta):

${NOMBRE_VAR%%patron}

Ej.:
MIFICH='MiDir/Subdir/prueba.txt'; echo ${MIFICH%%/*}

Salida:
MiDir
