Introducción a las Funciones UNIX

Índice

1. METACARACTERES

Nombre	Símbolo	Descripción
Espacio en blanco		Separador
Punto y coma	;	Separador de línea
Asterisco	*	Equivale a cualquier cadena alfanumérica
Interrogación	?	Equivale a un carácter cualquiera
Y inglesa	&	Ejecuta el comando que le antecede como tarea de fondo. Se recomienda que no haya mucha salida por pantalla.
Corchetes	[]	Permite especificar un conjunto de caracteres. Son legales los intervalos como [0-9][a-z]
BackSlash	\	Se utiliza para eliminar el significado especial de algunos caracteres, o sea, toma literalmente el carácter que viene a continuación.
Mayor que	>	Dirige la salida de una fuente a un destino. Normalmente la salida de cualquier comando por pantalla se dirige a un fichero (el fichero se crea totalmente nuevo)
Mucho mayor que	>>	Agrega la salida de un fuente a un destino. Funciona igual que el anterior si el fichero no existe, en caso contrario se concatena los ficheros.
Menor que	<	Extrae de un fichero la entrada estándar.
Pipe	|	Conecta la salida estándar del programa que le antecede con la entrada estándar del programa que le sigue.

Ej. `cat pepito | wc –w`

El resultado de `cat` es enviado al comando `wc`, saliendo pro pantalla la salida de `wc –w`. Por lo que al final en la pantalla se verá el número de palabras que contiene el fichero pepito.

2. PERMISOS Y SEGURIDAD

Cada usuario es dueño de sus directorios y ficheros, y puede restringir el acceso a otros usuarios.

Debido a esto se crea la siguiente jerarquía: propietario (que es el único que puede dar permisos de accesos), grupo y usuarios en general. El propietario dispone de estos comandos:

- **`chown`**: cambia usuario
- **`chgrp`**: cambia grupo
- **`chmod`**: cambia los permisos de ficheros o directorios.

3. RESUMEN DE COMANDOS

3.1. AYUDA

3.1.1. man

Función: da información sobre un comando.

Sintaxis: `man nombrecomando`

Ejemplo: `man mail`

Visualiza la información del comando `mail`.

3.2. VISUALIZACIÓN DE FICHEROS

3.2.1. cat

Función: muestra un(os) fichero(s) en la salida estándar

Sintaxis: `cat fich1[fich2…][-n]`

Parámetros:

`-n` muestra cada línea con su número

Ejemplo: `cat prueba`

Visualiza el contenido del fichero prueba

3.2.2. head

Función: Mostrará las n primeras líneas de un fichero

Sintaxis: `head [-n] fichero`

Parámetros:

`-n` número que indica las líneas a visualizar

Ejemplo: `head -20 pepito`

Mostrará las primeras 20 líneas del fichero pepito.

3.2.3. more

Función: concatena y visualiza página a página fichero(s) en la salida estándar

Sintaxis: `more fich1 [fich2..]`

Ejemplo: `more pepe pepito.wri`

Visualiza pantalla por pantalla el fichero `pepe` y a continuación el fichero `pepito.wri`.

3.2.4. tail

Función: Muestra las n últimas líneas de un fichero

Sintaxis: `tail [-n] fichero`

Parámetros:

`-n` número que indica las líneas a visualizar

Ejemplo: `tail -12 pepito`

Mostrará las últimas 12 líneas del fichero `pepito`

3.3. MANEJO DE FICHEROS Y DIRECTORIOS

3.3.1. cd

Función: Cambia de directorio

Sintaxis: `cd [camino]`

Ejemplo:

`cd ..` cambia al directorio padre

`cd` cambia al directorio definido por defecto (en el que estábamos al inicio de la sesión)

`cd /tan/paso` cambia al directorio `tan/paso`

3.3.2. cp

Función: copia ficheros

Sintaxis: `cp fichero1 [fichero2..] directorio o cp fichero1 fichero2`

Ejemplo:

`cp pepito gri.yo feliz /usr/almacen` Copia las archivos `pepito gri.yo feliz` en el directorio `/usr/almacen`

`cp pepito hormiga` Copia el contenido del fichero `pepito` en el archivo `hormiga`. El resultado es dos archivos idénticos con distinto nombre.

3.3.3. ln

Función: crea enlaces simbólicos. En esencia permite operar con un fichero o un directorio mediante distintos nombres y distintos privilegios.

Sintaxis: `ln fichero1 [fichero2 ..] directorio o ln fichero1 fichero2`

Ejemplo:

`ln jaimito pepito jorguito /usr/amigos` Permite utilizar los tres ficheros estando en su directorio origen o en el directorio `/usr/amigos`. Esto no quiere decir que se haya hecho una copia de estos tres archivos.

`ln jaimito pepito` Permite referenciar al archivo `jaimito` como `jaimito` o como `pepito`.

3.3.4. ls

Función: lista el contenido de un directorio

Sintaxis: `ls [-a -c -l -t -p -r -R -s -d -n][directorio | fichero]`

Parámetros :

`-a`	Se listan todos los ficheros, incluidos los que empiezan por '.'
`-c`	Se muestran los ficheros por orden de creación.
`-t`	Se muestran los ficheros por orden de modificación
`-l`	Se muestran todos los datos del fichero o directorio.
`-p`	Se marcan los directorios con el carácter '/'
`-r`	Se muestran los ficheros en orden inverso al seleccionado.
`-R`	Listado del contenido de todos los subdirectorios que cuelgan del directorio seleccionado además del contenido de éste (listado recursivo)
`-s`	Se muestran el tamaño del fichero en bloques de disco.
`-d`	Se limita el listado al directorio seleccionado
`-n`	Se muestran primero los últimos en usarse.

Ejemplo:

`ls`	Muestra los nombres de ficheros y directorios incluidos en el directorio actual, sin incluir los que empiezan por `.`
`ls pepe`	Muestra los nombres de ficheros y directorios contenidos en pepe, listando todos los datos de cada uno de ellos. No incluye los ficheros que empiezan por `.`

3.3.5. mkdir

Función: Crea directorios

Sintaxis: `mkdir dir1 [dir2..]`

Ejemplo:

`mkdir dirpepito cuasi`	Crea los directorios `dirpepito` y `cuasi`

3.3.6. mv

Función: Mueve un fichero de un origen a un destino

Sintaxis: `mv fichero1 [fichero2..] directorio o mv fichero1 fichero2`

Ejemplo:

`mv tantomonta montatanto`	Cambia el nombre de `tantomonta` por montatanto
`mv sara maria juana /usr/mujeres`	Copia los ficheros `sara`, `maria` y `juana` en el directorio /`usr/mujeres` y borra los tres ficheros del directorio origen

3.3.7. pwd

Función: Indica directorio actual (en el que estamos trabajando)

Sintaxis: `pwd`

Ejemplo:

`pwd`	Después de ejecutar el ejemplo anterior daría `/tan/paso`

3.3.8. rm

Función: Borra ficheros

Sintaxis: `rm [-i -r] fichero1 [fichero2..]`

Parámetros:

`-i`	Pide confirmación de borrado para cada fichero
`-r`	Borra todos los archivos y subdirectorios del directorio actual.

Ejemplo:

`rm pepito` Borra, sin posibilidad de recuperación, el archivo `pepito`.

`rm -r` Desde el directorio raíz esto borraría todo el contenido del servidor que no estuviese protegido. (**¡OJO!**)

3.3.9. rmdir

Función: Borra directorios

Sintaxis: `rmdir dir1 [dir2…]`

Ejemplo:

`rmdir dir.pepito cuasi_2` Borra los directorios `dir.pepito` y `cuasi_2`.

3.4. COMANDOS DE PERMISOS Y SEGURIDAD

3.4.1. chgrp

Función: Cambia de grupo a ficheros y directorios

Sintaxis: `chgrp nombre_grupo (fichero-s | directorio-s)`

Ejemplo:

`chgrp grpkc *` Transfiere la propiedad de todos los ficheros del directorio en curso al grupo `grpkc`

3.4.2. chmod

Función: Cambia los atributos de ficheros y directorios

Sintaxis:

a) `chmod n_octal (fichero-s | directorio-s)`

`n_octal`: se compone de tres dígitos octales que representan a los tipos de usuarios:

Primer dígito → creador del fichero

Segundo dígito → grupo del creador del fichero

Tercer dígito → otros usuraos del sistema

Cada dígito octal se obtiene de la suma de los pesos de cada tipo de permiso:

`r`: lectura (peso 4)

`w`: escritura (peso 2)

x: ejecución (peso 1) [en el caso de los directorios es acceso]

b) `chmod [ugo][+-][rwx] (fichero-s | directorio-s)`

Actúa sobre:

u	propietario del fichero
g	grupo del propietario del fichero
o	otros usuarios del sistema

Operaciones:

+	añadir permiso
-	retirar permiso
=	asignar permiso

Permisos:

r	lectura
w	escritura
x	ejecución o acceso

Ejemplo:

`chmod 751 prueba`	el permiso `751` del fichero prueba, se ha obtenido de la siguiente forma:
Permiso del creador:	4(lectura) + 2(escritura) + 1(ejecución)
Permiso del grupo:	4(lectura) + 0(escritura) + 1(ejecución)
Permiso de otros:	0(lectura) + 0(escritura) + 1(ejecución)

Utilizando la segunda sintaxis sería:

```
chmod u = rwx
chmod g = rx
chmod o = x
```

3.4.3. chown

Función: Cambia de propietario a ficheros y directorios.

Sintaxis: `chown nombre_usuario (fichero-s | directorio-s)`

Ejemplo:

`chown usukc *`	Transfiere la propiedad de todos los ficheros del directorio en curso al usuario `usukc`.

3.5. UTILIDADES DEL SISTEMA

3.5.1. at

Función: Ofrece la posibilidad de entrar una serie de órdenes por teclado que se almacenen en un fichero de órdenes, el cual se deberá ejecutar en la hora y fecha indicados. Para finalizar la entrada de órdenes pulsamos CTRL+D.

Sintaxis: `at hora [fecha][fich_ordenes] ordenes o`

`at [-l -r][trabajos]`

Parámetros:

`-l`	Permite visualizar el número de identificación asignado a cada uno de los diferentes trabajos creados con este comando.
`-r`	Elimina los números de trabajos de la cola creada por dicho comando.

Ejemplo:

`at 12 00 luego ls` A las 12:00 en punto por el reloj del sistema ejecutará el comando `ls` que obtendrá del fichero luego.

3.5.2. cal

Función: sin argumentos visualiza el calendario del mes actual. Indicando el mes y el año muestra dicho mes, y si sólo se especifica el año, aparece el calendario de todo el año.

Sintaxis: `cal [[mes]año]`

3.5.3. cmp

Función: Compara dos ficheros. Dando como salida el resultado de la comparación o los bytes (octal) en que se diferencian.

Sintaxis: `cmp [-ls] fich1 fich2`

Parámetros:

`-l`	Imprimir bytes que difieran (en octal).
`-s`	No produce salida por pantalla solo devuelve el resultado de la comparación.

Ejemplo:

`sort +0 bin/fich1 | cmp -l -fich2` Comprueba que el fichero ordenado `fich1` es igual al fichero `fich2`. Se emplean dos opciones curiosas, el guión (-) representa la entrada estándar y mediante una tubería la asociamos a la salida del comando `sort`.

3.5.4. comm

Función: Escribir líneas comunes de dos archivos ordenados. La salida se distribuye en tres columnas:

La 1ª representa las líneas que solo aparecen en el primer fichero.

La 2ª representa las líneas que solo aparecen en el segundo fichero.

La 3ª representa las líneas que aparecen en los dos ficheros.

Sintaxis: `comm [-123] fich1 fich2`

Parámetros:

`-1`	No sacar columna 1
`-2`	No sacar columna 2
`-3`	No sacar columna 3.

Ejemplo:

`sort +0 bin/fich1 | comm -12 –fich2` Saca sólo la tercera columna, que representa las líneas comunes de `fich1` y `fich2`. se emplean dos opciones curiosas, el guión (−) representa la entrada estándar y mediante una tubería la asociamos a la salida del comando `sort`.

3.5.5. date

Función: Visualiza en pantalla la fecha y hora actuales.

Sintaxis: `date`

3.5.6. df

Función: Muestra el número de bloques disponibles en el sistema de ficheros. Si no se especifica el sistema de ficheros, muestra el espacio disponible en el sistema de ficheros actual. (Los bloques de los sistemas de ficheros son de 1024 bytes)

Sintaxis: `df [-t] [sistema_ficheros]`

Parámetros:

`-t`	Ofrece información adicional sobre el espacio total

3.5.7. diff

Función: Crea un fichero con las diferencias que haya entre unos ficheros o entre directorios.

Sintaxis: `diff (fich1 fich2 | direct1 direct2)`

Ejemplo:

`diff tal cual` Muestra las diferencias que existen entre tal y cual.

3.5.8. ed

Función: Es el editor de texto estándar. Si indicamos un argumento. Lo toma como fichero de entrada, siendo introducido en el buffer de ed, para poder ser editado.

Sintaxis: `ed [-] [nombre_fichero]`

Parámetros:

-	Elimina la salida de aclaraciones y debería ser usada cuando la entrada estándar es un editor de script

3.5.9. find

Función: Busca a través del sistema de archivos los ficheros que coinciden con el nombre indicado.

Sintaxis: `find direct_inicial –name nombre [-print]`

Parámetros:

`-name`	Busca el fichero nombre
`-print`	Muestra el camino de los ficheros encontrados

Ejemplo:

`find . –name joselito –print` Busca a partir del directorio actual todos los ficheros cuyo nombre sea `joselito` y devuelve el camino donde los encuentra.

3.5.10. grep

Función: Busca cadenas o expresiones dentro de un fichero

Sintaxis: `grep [cadena][expresion] fichero`

Parámetros:

`-v`	Borra las líneas que contiene cadena
`-n`	Muestra el número de línea donde aparece la cadena
`-c`	Devuelve el número de apariciones

Ejemplo:

`grep pepe lopez fichero –n` Daría todas las líneas que encuentra con `pepe lopez` y el número de línea donde lo encuentra.

`grep [0-9] fichero` Daría todas las líneas que contengan un dígito del 0 al 9.

3.5.11. login

Función: Es utilizado para comenzar la sesión de cada terminal y a continuación el usuario debe identificarse ante el sistema. Puede ser llamado como un comando o por el sistema cuando se establece la primera conexión.

Este comando pregunta el nombre de usuario, y, si procede, su clave, sin mostrarla al introducirla. Si no cumple los requisitos exigidos por `login` en el tiempo estipulado, será desconectada.

`login` inicializa el entorno con información especificando el directorio por defecto, el interprete de comandos, `shell`, localización principal y especificación de la zona de tiempo.

Sintaxis: `login [nombre]`

3.5.12. mail

Función: Sin parámetros muestra el correo del usuario, mensaje por mensaje, siguiendo una estructura de pila. Por cada mensaje, el prompt de usuario es ¿, para determinar un mensaje, y una línea es leída desde la entrada estándar.

Cuando una persona es llamada, mail toma la entrada estándar hasta el final del fichero y lo suma al fichero de correo de cada persona. El mensaje es precedido del nombre del emisor y una postmarca.

Sintaxis: `mail [-epqr] [-f fichero]`

Parámetros :

`-f fichero`	Utiliza fichero en lugar del fichero de correo estándar.

3.5.13. mesg

Función: Permite o deniega el acceso de mensajes. Sin parámetros informa del estado actual sin cambiarlo

Sintaxis: `mesg [n] [ y]`

Parámetros:

`n`	Prohíbe la entrada de mensajes por medio del comando `write` o cualquier otro, no permitiendo la escritura en el terminal.
`y`	Reinstaura el permiso de escritura para el comando `write`

3.5.14. nohups

Función: Ejecuta un comando y elimina la posibilidad de ser interrumpido durante la ejecución. En caso de errores estos se mandan a un fichero llamado `nohup.out`.

Sintaxis: `nohups comando-s`

Ejemplo:

`nohups man mail` Ejecuta el comando `man mail` e impide que se pare la ejecución con CTRL-C

3.5.15. passwd

Función: Cambia la clave de usuario. Primero pedirá la clave antigua (si existía) y después pedirá dos veces la clave nueva.

Sobre la clave:

- debe tener al menos seis caracteres
- debe contener al menos dos caracteres alfabéticos
- debe ser distinta al identificador del usuario

- debe diferenciarse de la clave antigua al menos en tres caracteres

Sintaxis: `passwd`

3.5.16. pr

Función: Imprime ficheros en la salida estándar. Si no se pone ningún nombre de fichero se asume la entrada estándar.

Sintaxis: `pr [parametros][nombre_fichero]`

Parámetros:

`+n`	Empieza a imprimir a partir de la página n
`-p`	Hace una pausa tras cada página.

Ejemplo:

`pr -p tal` Imprime el fichero tal en pantalla, haciendo una pausa en cada página

3.5.17. sort

Función: Ordena ficheros

Sintaxis: `sort [opciones] [nombre_fichero] [fichero]`

Parámetros :

`+num`	Ordena `nombre_fichero` por campos (empieza por 0)
`-c`	Verifica si un fichero está ordenado
`-r`	Invierte orden
`-o`	Escribe salidas en `nombre_fichero`, funciona igual que hacer una redirección
`-n`	Realiza un orden numérico
`-u`	Elimina duplicados

Ejemplo:

`sort +1 lio` Ordena las líneas del fichero lio por orden alfabético

3.5.18. uniq

Función: Suprime líneas consecutivas e idénticas de un archivo ordenado

Sintaxis: `uniq [-cdu] fichero1 fichero2`

Parámetros :

`-c`	Saca por pantalla el número de líneas idénticas y consecutivas
`-d`	Saca por pantalla sólo las líneas que están duplicadas
`-u`	Saca por pantalla sólo las líneas que no están duplicadas

Ejemplo:

`sort datos | uniq -c` Ordena las líneas del fichero datos y saca por pantalla el número de veces que aparece cada línea y la línea en cuestión. Una posible salida sería:

2 abax

1 gthdf

3.5.19. wc

Función: Contar los caracteres, palabras o líneas de un fichero

Sintaxis: `wc [-cwl] nombre_fichero`

Parámetros :

-c	Cuenta los caracteres del fichero
-w	Cuenta las palabras del fichero
-l	Cuenta las líneas del fichero

3.5.20. who

Función: Da una lista de los usuarios que están conectados al sistema e información sobre ellos..

Sintaxis: `who [opciones]`

Parámetros:

`am i`	Da información sobre sí mismo
`-d`	Muestra sólo los nombres de los que están conectados

Ejemplo:

who Daría una lista sobre los usuarios conectados que tendrá el identificador de cada uno, terminal y tiempo de conexión

3.5.21. write

Función: Copia líneas desde el terminal al de otro usuario

Sintaxis: `write otrousuario`

Parámetros:

!	El signo de admiración indica que lo que viene a continuación se tomará como un comando

4. CONTROL DE TRABAJOS Y PROCESOS (C-SHELL)

Se llama así a la parte que permite controlar varias tareas desde un mismo terminal.

4.1.1. bg

Función: Manda un trabajo a Background, previa detención del trabajo (con CTRL-Z). Seguirá su ejecución en el momento en que entre en el Background.

Sintaxis: `bg num_proceso`

4.1.2. fg

Función: Trae un trabajo de Background a FrontGround. Es lo contrario que la instrucción anterior. Igualmente, hay que detener el proceso antes de realizar la operación.

Sintaxis: `fg num_proceso`

4.1.3. jobs

Función: Muestra los trabajos actuales del usuario.

Parámetros:

+	Muestra el último trabajo detenido o el último puesto en Background
-	Indica el penúltimo
-l	Añade al listado el número de identificación de los trabajos

Salidas:

Exit1	Indica que un trabajo termina por no poder sobrescribir el fichero de trabajo
[n°]Done	Indica que el trabajo se ha realizado correctamente

Ejemplo:

La secuencia de instrucciones:

```
sort pp > pp.0 &
ls –lR
jobs –l
```

Daría como resultado:

```
[1] -328 Running sort –r /usr/pp
[2] +329 Running ls -lr
```

4.1.4. kill

Función: Termina con la ejecución de un comando que se esté ejecutando en BackGround

Sintaxis: `kill [n] [+] [-]`

Parámetros:

n	Número del trabajo a eliminar
+	Termina el último trabajo
-	Termina con el penúltimo proceso
-9	Acaba con completa seguridad un trabajo

Ejemplo:

```
kill 1          Termina el trabajo 1
kill +          Termina el último trabajo
```

4.1.5. ps

Función: Indica el estado de los procesos. Nos da información sobre:

`PID`: número de identificación de proceso

`TT`: terminal desde la que se lanzo el proceso

STAT: estado del proceso. Puede ser:

R: Ejecutando

T: Parado

P: Espera de página de memoria

D: Espera de uso de disco

TIME: tiempo de ejecución que lleva

COMMAND: comando que lanzó el proceso

Sintaxis: ps

Parámetros:

-a	Informa de todos los procesos
-u	Salida en pantalla orientada al usuario
-x	Informa de los proceso que no son controlados por el terminal (lanzados por el sistema)

Ejemplo:

```
ps          Devuelve: PID TT    STAT     TIME    COMMAND
                      110 CO1   R,T,P,D  0:48          Sun View
```

5. SHELL-SCRIPTS

5.1. DESCRIPCIÓN

Un Shell-Script es un fichero de texto que en su forma más simple es una sucesión de comandos UNIX, separados por retornos de carros o por el carácter ;. Tiene cierta semejanza con un fichero BATCH del los sistemas xx-DOS.

Para invocar a un Shell-Script (fichero de comandos) se utilizan dos métodos:

A) `% sh nombreshellscript [arg1][arg2]…`

B) `% chmod ugo +x nombreshellscript`

`% nombreshellscript`

al llamarlo se ejecutará en un nuevo shell, en su defecto un shell Bourne. Para que se lance un c-shell el shell script debe comenzar por '##'

5.2. VARIABLES

Las variables del SHELL deben comenzar con una letra aunque pueden contener números y subrayados.

- La asignación se realiza así: `nombreVar = valor`
- Para utilizar el contenido de una variable en un shell se usa la notación: $NombreVar
- Para añadir algo a una variable se utiliza /: `var2 = $var1/a`

5.2.1. Variables especiales

`$?`	Condición de salida devuelta por el último comando
`$0`	Nombre del ShellScript que se está ejecutando
`$#`	Número de parámetros del Shell Script
`$1, $2 ..`	Parámetros que se introducen al llamar a un ShellScript
`$$`	`PID` del proceso que se esta ejecutando
`$!`	`PID` del último proceso ejecutado en background
`$-`	Contiene la bandera actual del shell que estamos ejecutando
`$HOME`	Directorio al que se accede al entrar en el sistema
`$path`	Contiene los caminos de búsqueda
`$PS1`	Contiene el `prompt` primario, por defecto `$`
`$PS2`	Contiene el `prompt` secundario, por defecto `>`

5.3. ESTRUCTURAS DE CONTROL DE FLUJOS

5.3.1. Control de alternativas

`if`

Sintaxis:

```
if lista_comandos1
then lista_comandos2
else lista_comandos3
fi
```

Función: Tiene el mismo funcionamiento que un if normal, cierto se consigue cuando la ejecución de todos los comandos de lista_comandos1 es correcta

Ejemplo:

```
if( test –f fich2; cp fich2 fich3)
then echo "El fichero fich2 existe y ha sido copiado correctamente a fich3."
else echo "El fichero fich2 no existe y no ha sido copiado correctamente a fich3."
fi
```

Si `test` y `cp` se ejecutan correctamente entonces se ejecuta el comando detrás del `then`, sino se ejecuta el comando detrás del `else`.

`case`

Sintaxis:

```
case (palabra | variable) in
patron1) lista_comandos;;
patron2) lista_comandos;;
```

```
.
patronn) lista_comandos[;;]

esac
```

Función: Cuando (palabra | variable) coincide con un patrón, se ejecutará la lista de comandos asociada. Los caracteres `;;` indican final de lista de comandos.

Ejemplo:

```
case $1 in
-l) ln $2 $3;;
-*) echo "El parámetro es desconocido.";;
*) echo "Sintaxis incorrecta.";;
echo " util `-cl-l] fich1 fich2"
esac
```

En el case se investiga el valor del parámetro número 1 y lo compara con varios patrones, ejecutándose al final los comandos asociados al patrón válido.

5.3.2. Control de bucles

for

Sintaxis:

```
for VARIABLE [in LIST_Valores]
do LISTA_COMANDOS
done
```

Función: la iteración se repetirá hasta que se acabe el número de elementos de LIST_Valores; estos pueden ser palabras o variables. En cada iteración VARIABLE recibe un valor de LIST_Valores y se ejecuta LISTA_COMANDOS.

Si omitimos LIST_Valores se toman por defecto todos los parámetros introducidos al ejecutar el shell-script.

Ejemplo:

a)
```
for i in hola "hola de nuevo" adios
do
   echo $i
done
```

La salida por terminal será:

```
hola
hola de nuevo
adios
```

b)
```
var1 = " buenos dias "
for i in $var1 $*
do
```

```
   echo $i
done
```

En terminal se escribirá primero buenos días (`$var1`) seguido por todos los parámetros (`$*`) introducidos en la línea de comandos al llamar al `shell script`.

c)
```
for i in *ito
do
   echo $i
done
```

En terminal se escribirán los nombres de todos los ficheros del directorio actual que terminen en `ito` (`*ito`).

d)
```
for i
do
   echo $i
done
```

En terminal se escribirán todos los parámetros introducidos en la línea de comandos al ejecutar el `shell script`.

while

Sintaxis:

```
while lista_comandos1
do lista-comandos2
done
```

Función: Los comandos de lista_comandos2 se ejecutarán siempre que la ejecución de todos los comandos de lista_comandos1 sea correcta.

Ejemplo:

```
while test $1
do
      echo $1
      shift
done
```

En cada iteración el parámetro `$1` es escrito en la terminal y luego todos los parámetros son desplazados hacia la izquierda [$2 pasa a ser $1, $3 ->$2, $4->$3, etc], por lo que el bucle no termina hasta que $1 sea una cadena nula.

`test $1` devuelve ejecución correcta cuando `$1` es una cadena no nula.

until

Sintaxis:

```
until lista_comandos1
do lista_comandos2
done
```

Función: Se ejecutarán los comandos de lista_comandos2 hasta que la ejecución e todos los comandos de `lista_comandos1` sea correcta.

Ejemplo:

```
until test –f file
do
      sleep 300
done
```

En este bucle se comprueba si existe `file` cada 300 segundos, el bucle continua hasta que test devuelva ejecución correcta. [`test –f file` devuelve correcto cuando `file` existe].

El fichero será creado por otro proceso.

En las estructuras `if`, `while` y `until` los comandos se pueden combinar utilizando

`!`	Not lógico
`-a`	And lógico
`-o`	Or lógico
`(lista_comandos;)`	
`{lista_comandos;}`	Agrupo las distintas pruebas o grupos de pruebas.

5.4. COMANDOS EMBEBIDOS DEL SHELL-SCRIPT

`break`

Función: Salida de bucles sin que se terminen expresamente.

Sintaxis: `break [n]`

`n`	Es el número de bucles de los que saldremos si hubiera varios anidados

Ejemplo:

```
for i in parámetros "parametos de nuevo" adios "adios no valido"
do
      echo $i
      for a in $*
```

```
        do
              if( test $i = "adios") then
                    break 2
              fi
              echo $a
        done
done
```

Este bucle anidado se ejecutará hasta que `$i` valga `adios`, ya que entonces el comando *`break 2`* producirá la salida de los dos bucles directamente.

`continue`

Función: Recomenzar el bucle. Produce que la ejecución vuelva al principio del bucle.

Sintaxis: `continue [n]`

`n`	Es el número de iteraciones que haremos avanzar el bucle.

Ejemplo:

```
for i
do
      if( test $i = "-p") then
            continue 2
      fi
      echo $i
done
```

En este bucle sacará por el terminal todos los parámetros introducidos al llamar al `shell-script`; menos cuando el parámetro sea igual `a -p`, entonces no se escribe ni `-p` ni el parámetro que le sigue.

5.5. COMANDOS DEL SHELL-SCRIPT

echo

Función: Imprime una cadena y un salto de línea en la salida estándar.

Sintaxis:

```
echo [-n] ["cadena" | cadena]
```

`n`	Evita que se produzca el salto de línea.
`""`	Si la cadena se escribe entre comillas dobles, también se cogerán los espacios de principio y final de cadena

Ejemplo:

```
a) var1 = si utilizo
   echo –n “En este ejemplo “
   echo –n $var1
   echo “ comillas ”
```

Salida por terminal:

En este ejemplo si utilizo comillas

```
b) var1 = no utilizo
   echo En este ejemplo
   echo $var1
   echo comillas
```

Salida por terminal:

En este ejemplo
no utilizo
comillas

```
c) var1 = no utilizo
   echo –n En este ejemplo
   echo –n $var1
   echo comillas
```

Salida por terminal:

En este ejemplono utilizocomillas

`expr`

Función: Sirve para evaluar expresiones. El resultado lo saca por pantalla sino se redirecciona la salida.

Se pueden utilizar las operaciones `+, –, *, /` y `%` (módulo)

Sintaxis:

```
Exp. (variable | entero) operación (variable | entero)
```

Ejemplo:

```
expr 2 + 2
```

La salida por pantalla será: 4

read

Función: Realiza una entrada desde consola para asignársela a la`-s` variable `-s` de lista_variables. Si se redirecciona la entrada solo se tomará una línea para cada variable.

Sintaxis:

```
read lista_variable
```

Ejemplo:

```
a) read a b c
```

Este comando espera que se introduzcan tres valores cada uno seguido por ENTER.

```
b) read a b c < zootecnico
```

Este comando lee tres líneas del fichero `zootecnico` y las inserta en las variables `a b c`, una línea en cada variable.

shift

Función: Produce un desplazamiento hacia la izquierda de los parámetros del `Shell-script`, este desplazamiento no es circular. [$2 pasa a ser `$1, $3`→`$2, $4`→`$3`, etc]

Ejemplo:

```
while test $1 do
      echo $1
      shift
done
```

En cada iteración el parámetro $1 es escrito en la terminal y luego todos los parámetros son desplazados hacia la izquierda, por lo que el bucle no termina hasta que $1 sea una cadena nula. Test $1 devuelve ejecución correcta cuando $1 es una cadena no nula.

sleep

Función: Produce una pausa en segundos

Sintaxis:

```
sleep n
```

`n`	Es el número de segundos.

Ejemplo:

```
until test –f file do
      sleep 300
```

```
done
```

En este bucle se comprueba si existe file cada 300 segundos, el bucle continua hasta que test devuelva ejecución correcta. [`test –f file` devuelve correcto cuando `file` existe].

El fichero será creado por otro proceso.

test

Función: Realiza pruebas sobre ficheros, directorios, cadenas y enteros.

Sintaxis:

Para ficheros o directorios:

```
test [-f –r –w –d -s] (variable | cadena)
```

`-f`	Devuelve ejecución correcta si el fichero existe
`-r`	Devuelve ejecución correcta si el fichero existe y tenemos derecho de lectura sobre él
`-w`	Devuelve ejecución correcta si el fichero existe y tenemos derecho de escritura sobre él.
`-x`	Devuelve ejecución correcta si el fichero existe y tenemos derecho de ejecución sobre él.
`-s`	Devuelve ejecución correcta si el fichero existe y no es vacío
`-d`	Devuelve ejecución correcta si el directorio y existe

Para cadenas:

```
test [-n –z] (variable | cadena)
```

`-n`	Devuelve ejecución correcta si la longitud de la cadena es distinta de cero
`-z`	Devuelve ejecución correcta si la longitud de la cadena es cero
	Sin parámetros devuelve ejecución correcta si no es la cadena nula

```
test (variable | cadena) ( = | !=) (variable | cadena)
```

`=`	Devuelve ejecución correcta si las dos cadenas son iguales
`!=`	Devuelve ejecución correcta si las dos cadenas son distintas

Para enteros:

```
test (variable | entero) operación (variable | entero)
```

Las operaciones son:

-eq	Devuelve ejecución correcta si los dos enteros son iguales
-ne	Devuelve ejecución correcta si los dos enteros son distintas
-gt	Devuelve ejecución correcta si el dato1 es mayor que el dato2
-ge	Devuelve ejecución correcta si el dato1 es mayor o igual que el dato2
-lt	Devuelve ejecución correcta si el dato1 es menor que el dato2
-le	Devuelve ejecución correcta si el dato1 es menor o igual que el dato2

5.6. SUSTITUCIÓN DE COMANDOS

Si encerramos un comando entre comillas graves (` `), toda salida a pantalla que tuviera se transforma en una cadena que se le asigna a la variable que tuviera a la izquierda.

Ejemplo:

```
N = `expr 1 + 1` → n = 2
```

5.7. DEPURAR UN SHELL-SCRIPT

5.7.1. Depuración interna

set

Función: Produce una depuración del `shell script`. Se debe incluir en el `Shell Script` como un comando.

Sintaxis:

```
set -[x | v]
```

x	Produce que se imprime el comando y los valores que le corresponde a los parámetros, después se ejecutará el comando.
v	Produce que se imprima la línea a ejecutar y después sea ejecutada.

5.7.2. Depuración externa

Sintaxis:

```
sh -[x | v] nombre_script
```

x	Produce que se imprime el comando y los valores que le corresponde a los parámetros, después se ejecutará el comando.
v	Produce que se imprima la línea a ejecutar y después sea ejecutada.