Comandos LINUX

Introducción.

cat: muestra el contenido de un fichero de texto.

Si fichero existente, visualización por pantalla del contenido del fichero.
Para crear un nuevo fichero e introducir en él algunos datos es necesario utilizar un operador de redirección (>).

Sintaxis es la siguiente:
cat > nombre_fichero
....
....
....
<Ctrl>d

more: visualiza un fichero.

Concatenar y visualizar ficheros contenidos en lista_ficheros, visualiza página a página su contenido. La visualización se detiene después de cada página, pulsar **<Return>** para avanzar una línea del fichero, o **<Space>** para visualizar la siguiente página.

Sintaxis: more [lista_ficheros]

ls: lista los nombres de los ficheros.

Listado en pantalla de los nombres de los ficheros y directorios que contiene un directorio.

Sintaxis: ls [-laR] [nombre_directorio]|[nombre_fichero]

Opciones:

1. **-l** (listado largo) permite obtener información sobre las características de un fichero. Visualiza en pantalla el propietario y el grupo de cada fichero del directorio así como los derechos de acceso. Primer grupo de información: indica el tipo de fichero del que se trata y los derechos asociados a él. El primer carácter indica el tipo de fichero:
   · si es **-** entonces es un fichero ordinario.
   · si es **d** se trata de un directorio.
   · si es **c** significa que es un fichero especial en modo carácter.
   · si es **b** entonces es un fichero especial en modo bloque.
   · si es **l** se trata de un vínculo (o enlace) simbólico
   Los nueve caracteres siguientes indican: derechos de acceso al fichero, número de enlaces que contiene el fichero, el grupo al que pertenece, el tamaño del mismo en bytes, la fecha y hora de creación, y por último el nombre del fichero.

2. –a: permite visualizar todos los ficheros incluidos en el directorio, incluyendo los que podemos considerar "ocultos", que son los ficheros que comienzan con un punto. (ej: el fichero nuevo no se visualizaría con el comando ls, para verlo habría que poner ls -a).

3. –R: listado recursivo del directorio especificado (el actual si no se especifica ninguno) y de todos los subdirectorios que contiene.

cp: copia un fichero.

Sintaxis: cp [-ir] (fichero fichero | directorio directorio | lista_ficheros directorio)

Opciones:

1. **-i** pide confirmación de copia para cada fichero (**y** o **n**), si el fichero existe en destino.
2. **-r** copia de forma recursiva en todos los subdirectorios.

rm: elimina un fichero.

Este comando suprime uno o varios ficheros.

Sintaxis: rm [-ir] nombre_fichero1 nombre_fichero2 ...

Opciones:

1. **-i** pide confirmación de borrado para cada fichero (**y** o **n**).
2. **-r** borra de forma recursiva en todos los subdirectorios.

mv: cambia el nombre de un fichero o mueve ficheros.

Sintaxis: mv nombre_fichero nuevo_nombre_fichero
mv lista_ficheros directorio

head: muestra el comienzo de un fichero.

Opción por defecto = 10.

Sintaxis: head [-numero_líneas] nombre_fichero1 nombre_fichero2 …

tail: muestra el final de un fichero.

Opción por defecto = 10.

Sintaxis: tail [-numero_líneas] nombre_fichero1 nombre_fichero2 …

wc: visualiza el número de líneas, palabras y bytes de un fichero.

Sintaxis: wc nombre_fichero1 nombre_fichero2 …

pwd: muestra la ruta del directorio actual.

Sintaxis: pwd

mkdir: crea un directorio.

Sintaxis: mkdir nombre(s)_directorio(s)

Opciones:

1. **–p** crea los directorios padres que no existan en las rutas especificadas

cd: cambia de un directorio a otro.

Sirve para cambiar de directorio.

Sintaxis: cd [camino_directorio]

Si no ponemos ningún argumento nos devuelve al directorio de recepción. Existen dentro de cada directorio dos directorios especiales que representan al padre (**..**) y al propio directorio (**.**)

rmdir: elimina un directorio.

Sintaxis: rmdir nombre(s)_directorio(s)

Sistema de ficheros.

Caracteres "comodines" (* ,? , [ ])

1. El caracter (*****) sustituye a cualquier cadena (incluida la vacía).
   Por ejemplo, para listar todos los ficheros que empiezan con la letra p, ejecutaríamos el mandato:

   ls p*

2. El carácter (**?**) sustituye sólo a un carácter, pero éste debe existir (no admite una cadena vacía).
   Por ejemplo, para visualizar por pantalla el contenido de todos los ficheros cuyo nombre tiene tres caracteres y empieza y termina por la letra a, ejecutaríamos:

   cat a?a

3. Los caracteres (**[ ]**) sirven para especificar una lista de caracteres que puede corresponder con un carácter en el nombre del fichero. Es igual al carácter (?) pero limita la comparación al grupo de caracteres que se encuentran entre los corchetes.
   Por ejemplo, si ejecutásemos el siguiente mandato:

   ls PA[VTLC]O Nos listaría (si existen) los ficheros: PAVO, PATO, PALO, PACO

Caracteres de "redirección" (< , > , >> , |)

1. < : redirige la entrada de un comando hacia un fichero determinado.

2. > : redirige la salida de un comando hacia un fichero determinado, en lugar de a la pantalla. Actúa como si fuera una copia, es decir, si el fichero especificado no existe, lo crea, y si ya existe, machaca su contenido con la nueva información (salida del comando).
Ejemplo: **$ ls -l > listado**

3. >> : redirige la salida estándar de un comando hacia un fichero, pero en este caso, la información se añade al final del fichero y no se machaca el contenido anterior del mismo.

4. | : redirige la salida de un comando como entrada de otro. Por ejemplo, si quisiéramos visualizar cómodamente el listado largo de un directorio extenso podríamos hacer las siguientes operaciones:
$ ls -l /etc > listado
$ more listado
$ rm listado
Sin embargo, podemos conseguir ese mismo resultado sin necesidad de crear ningún fichero intermedio de la siguiente manera: **$ ls -l /etc | more**

Otros caracteres (; , &)

1. ; : separa más de un mandato dentro de la misma línea de órdenes. Por ejemplo: **$ cp * dir1 ; clear ; ls dir1**

2. & : ejecuta las órdenes como "tarea de fondo", o en "modo subordinado" (*background*).

Derechos de acceso a ficheros y directorios.

Ver derechos de acceso a un fichero o a un directorio: **ls -l**. Dichos permisos de acceso corresponden a los nueve caracteres siguientes al primer carácter, que es el que indica el tipo de fichero (fichero normal, directorio, o fichero especial). Suponiendo permitidos todos los accesos, esos nueve caracteres mostrarían lo siguiente: **rwxrwxrwx**
Usualmente no están todos los permisos habilitados, en cuyo caso aparece un guión en lugar de la letra correspondiente (ej. **rw-r--r--**).

1er carácter: **r** Permiso de lectura (y copia).
2º carácter: **w** Permiso de escritura (modificación y borrado).
3er carácter: **x** Permiso de ejecución.

Modificación de los derechos de acceso.

chmod: dos modos de funcionamiento: modo simbólico y modo absoluto.

1. **Modo simbólico.**

Cambia uno o varios de los nueve caracteres de acceso a los ficheros.

Sintaxis: chmod *categoría* operador *permiso* nombre_fichero

Dentro de *categoría* tenemos cuatro opciones:

u (= user): derechos del propietario
g (= group) : derechos del grupo
o (= other): derechos del resto
a (= all) : todos

Los operadores son los siguientes:

+ añade un derecho
- retira un derecho

Los permisos son los tres ya nombrados. (**r** , **w** , **x**)
Se pueden combinar varias modificaciones de estas separadas por comas y sin dejar espacios en blanco. Por ejemplo:

$ chmod a+r,u+w nombre_fichero

También se pueden juntar varias categorías o varios permisos. Por ejemplo:

$ chmod ug+x,o-rw nombre_fichero

2. Modo absoluto.

De esta forma se cambian de una vez todos los permisos de un fichero o directorio.

Sintaxis: chmod NNN nombre_fichero

Donde NNN representa tres cifras que pueden ir del cero al siete. La primera cifra corresponde al propietario, la segunda al grupo y la tercera al resto de usuarios. Cada una de estas cifras se transforma a binario, originando tres cifras binarias con las que se construyen los bits de acceso al fichero (1-permiso activado, 0-permiso desactivado).
Por ejemplo, queremos que los derechos de acceso a un fichero sean:

r w x	r - x	r - -
1 1 1	1 0 1	1 0 0
7	5	4

Por tanto el comando para ponerle esos permisos sería: $ **chmod** 754 nombre_fichero

Modificación de los derechos de acceso asignados por defecto.

Al crear un fichero nuevo, el SO asigna derechos de acceso por defecto. Para calcularlos se utiliza lo que se denomina la *máscara*, un número de tres cifras que van del cero al siete.

El proceso para calcular permisos:

- Directorios: el sistema realiza la operación XOR (OR exclusivo) bit a bit entre la *máscara* y el 777. El resultado de esa operación (que será una serie de 9 bits) son los derechos de acceso que se aplican al directorio que acabamos de crear (1- permiso activado, 0-permiso desactivado).
  Ejemplo: Si la máscara es 023 (000010011), cuáles serían los permisos por defecto asignados a un directorio de nueva creación. Realizamos la operación XOR con 777 (111111111):

  111111111
  000010011
  _____
  111101100

  Por tanto serían: **rwxr-xr—**

- Ficheros: proceso =, salvo que ignora el resultado para los permisos de ejecución en las tres categorías. *Nunca se puede dar por defecto permiso de ejecución para los ficheros*. Con la máscara anterior, los permisos que se asignarían a un fichero de nueva creación serían: **rw-r--r--**

Para consultar la máscara que está usando el sistema y para poder cambiarla se utiliza el comando **umask**.

Sintaxis: umask [NNN]

Donde NNN corresponde a la nueva máscara. Si se utiliza este comando sin argumento, nos devuelve la máscara que está utilizando el sistema.

ln: para hacer un enlace de un fichero.

Sintaxis: ln nombre_fichero nombre_enlace

Expresiones regulares.

. (punto) :representa cualquier carácter simple. Por tanto, la cadena **a.c** se referiría a cualquier cadena que comience con *a*, acabe con *c* y tenga cualquier carácter entre ellos.

[ ] (corchetes): representan a un conjunto de caracteres. Por ejemplo, la expresión **[tyu]** se corresponderá con cualquiera de los caracteres simples *t*, *y*, o *u*. La expresión **[aA]** se corresponderá con la letra *a* mayúscula o minúscula. Este tipo de expresión se corresponde sólo con un carácter en el fichero que se está buscando. Se pueden utilizar secuencias de estas expresiones para buscar más de un carácter. Por ejemplo, la expresión **[aA][bB]** se corresponderá con cualquiera de las cadenas **ab**, **Ab**, **aB** o **AB**.
Si queremos buscar en un fichero alguno de los caracteres que utilizamos como operadores dentro de las expresiones regulares (**- . [ ]**), debemos indicarlo para que no lo interprete como un operador sino como un carácter. La forma de *escapar* del significado especial del carácter es precederlo del carácter **** (barra invertida). Si quisiéramos buscar la cadena *[A* tendríamos que utilizar la siguiente expresión regular: **\[A**.

Es posible restringir las expresiones regulares al principio o final de una línea, de modo que se puede especificar la cadena que debe aparecer en la primera o última posición de una línea. Para designar el comienzo de una línea se utiliza el carácter **^** . La expresión regular **^Hoy** se corresponde con la cadena *Hoy*, únicamente si está al principio de una línea. Análogamente, se puede utilizar el carácter **$** para designar el final de una línea.
Por ejemplo, para buscar la cadena *fin* al final de una línea utilizaríamos la siguiente expresión regular: **fin$**.
Una expresión del tipo **hola^adios** no tiene sentido ya que las expresiones regulares no se extienden a múltiples líneas. Los corchetes también se pueden usar para correspondencias de un sólo carácter de un rango de caracteres en la secuencia de ordenación alfabética. Una expresión que encuentre cualquier carácter simple sería: **[A-Zaz]**.

Se pueden combinar expresiones regulares de un sólo carácter para buscar cadenas más largas. Por ejemplo, la expresión **[0-9][0-9][0-9]** se corresponderá con una cadena con tres dígitos seguidos. Se puede utilizar el operador ***** (asterisco) para designar cero o más ocurrencias de la expresión regular de un sólo carácter precedente. Para encontrar una secuencia de dígitos de cualquier longitud utilizaríamos la siguiente expresión regular: **[0-9][0-9]***. No podemos utilizar únicamente la expresión **[0-9]***, ya que también indica la cadena vacía.
Una regla importante es que las expresiones regulares se corresponden con la cadena de correspondencia más larga posible. Por ejemplo, si tenemos la cadena *abc12345fgh*, si utilizamos la expresión regular anterior **[0-9][0-9]***, dicha expresión se corresponderá con todos los dígitos, es decir:
12345. Hay que tener cuidado con la complejidad de las expresiones regulares puesto que pueden efectuar más correspondencias de las esperadas, o más amplias que las esperadas.

grep: busca una cadena dentro de ficheros.

La salida que produce este comando son las líneas del fichero que contienen la cadena coincidente.

Sintaxis: grep [-vcni] expresión_regular fichero (o lista de ficheros)

Por ejemplo, si quisiéramos encontrar las líneas que comienzan por una letra minúscula seguida de un dígito pondríamos:

```
grep "^[a-z][0-9]" nombre_fichero
```

La expresión regular se encuentra *acotada* (entre comillas), para impedir que el shell del Linux interprete los caracteres corchete (y los sustituya por una lista de ficheros) antes de que el comando **grep** los vea.
Si se incluye más de un fichero en la línea de órdenes, el comando **grep** informará del nombre del fichero al comienzo de cada línea que imprima.
Opciones:

1. La opción -v selecciona todas las líneas que no contengan el patrón especificado.
2. La opción -c devuelve solamente el número de líneas coincidentes.
3. La opción -n añade a la salida de cada línea, el número de línea que ocupa en el fichero fuente.
4. La opción -i ignora la distinción entre mayúsculas y minúsculas en la comparación.

cmp: compara dos ficheros.

Si los ficheros son iguales, no dice nada, si son distintos, informa del primer carácter en que se diferencian los dos ficheros.

Sintaxis: cmp [-l] fichero1|- fichero2

En lugar del primer nombre del fichero, este comando puede llevar el argumento **-** (menos). En este caso compararía la entrada estándar con el segundo fichero. Por ejemplo, podríamos poner:

```
cat prueba | cmp - dir/prueba
```

La opción **-l** hace que el comando liste el número de carácter y los valores diferentes para cada una de las diferencias entre los dos ficheros.

diff: compara dos ficheros.

Compara dos ficheros. Si son iguales, no dice nada. Si son distintos, da como salida un listado completo de todas las líneas que difieren entre los dos ficheros junto con sus números de líneas.

Sintaxis: diff [-wi] fichero1|- fichero2

El funcionamiento del **-** (menos) como sustituto del primer nombre de fichero es igual al del comando **cmp**.

Opciones:

1. La opción -w hace que el comando ignore las líneas que sólo se diferencian en sus espacios en blanco (también tabuladores).
2. La opción -i ignora la diferencia entre mayúsculas y minúsculas en la comparación.

sort: muestra el contenido de un archivo en orden

Muestra el contenido de un fichero en orden alfabético por líneas, pero no cambia el contenido del fichero original. Si se incluye más de un fichero en la línea de órdenes, este comando mostrará de forma ordenada por líneas el contenido de todos los ficheros unidos.

Sintaxis: sort [-un] fichero (o lista de ficheros)

Opciones:

1. La opción -u muestra el contenido del fichero en orden alfabético por líneas eliminando las líneas duplicadas.
2. La opción –n muestra el contenido del fichero (se supone que contiene números) ordenado por líneas siguiendo el orden numérico.

cut: corta campos de una tabla escrita en un fichero.

Una tabla se puede considerar como una colección de líneas, cada una de las cuales contiene un registro; y donde cada registro tiene un número fijo de campos. Normalmente, los campos están separados mediante tabuladores o espacios, aunque se puede utilizar cualquier separador de campos. El comando cut permite cortar uno o más campos de una tabla almacenada en uno o más ficheros; es decir permite trocear verticalmente un fichero.

Sintaxis: cut -c*lista* fichero (o lista de ficheros)
cut -f*lista* [-d*car*] [-s] fichero (o lista de ficheros)

lista es una lista de números que especifica el rango de caracteres o campos a extraer. Los caracteres o campos de la tabla se comienzan a enumerar desde 1. Si los números de esta lista están separados por coma, dichos números especifican los campos del archivo a extraer. En el caso de que los números de la lista estén separados mediante un guión (–), los números de la lista especifican el rango de caracteres o de campos a extraer.

car indica el carácter utilizado como separador de campos. El tabulador es el separador de campos por omisión.

Opciones:

1. La opción -c trata cada carácter como una columna y extrae los caracteres especificados en la *lista*.
2. La opción –f extrae los campos especificados en la *lista*.
3. La opción –d utiliza el carácter *car* en lugar del tabulador como separador de campos.
4. La opción –s no muestra las líneas del fichero que no contengan el carácter delimitador.

Por ejemplo, suponiendo que el contenido del fichero original *maquinas* es el siguiente:

eixe LaboratorioSO8 192.132.111.81 pcpardo
pindo LaboratorioSO7 192.132.222.33 nrufino
faro Laboratorio37 192.132.333.37 nanny

donde los campos han sido separados usando tabulador, los resultados que se obtendrían con los siguientes comandos son:

a. cut –f1 maquinas // Extrae el primer campo de cada línea
eixe
pindo
faro

b. cut –f1,3 maquinas // Extrae el primero y el tercer campo de cada línea
eixe 192.132.111.81
pindo 192.132.222.33
faro 192.132.333.37

c. cut –f1-3 maquinas // Extrae los tres primeros campos de cada línea

eixe LaboratorioSO8 192.132.111.81
pindo LaboratorioSO7 192.132.222.33
faro Laboratorio37 192.132.333.37

d. cut –c1,3 maquinas // Extrae el primero y el tercer carácter de cada línea
ex
pn
fr

e. cut –c1-4 maquinas // Extrae los cuatro primeros caracteres de cada línea
eixe
pind

faro

paste: concatena ficheros horizontalmente.

Este comando muestra el resultado de concatenar el contenido de los ficheros que se especifican como argumentos horizontalmente (la orden cat concatena ficheros verticalmente). Este comando, por lo tanto, complementa al comando anterior (cut), pues permite pegar tablas por columnas. No obstante, se debe tener en cuenta que la salida de esta orden se visualiza en pantalla y no se almacena en un fichero.

Sintaxis: paste lista de ficheros

Miscelánea.

ps ofrece información sobre los procesos en curso asociados con el usuario

Sintaxis: ps [-l]

Si se emplea la opción **–l** proporciona más información como, el número de identificación del proceso (PID), el terminal desde el cual se inició el proceso (TTY), el estado del proceso (S) que puede ser activo o ejecutando (**O**), bloqueado (**S**) y listo (**R**) ; la cantidad de tiempo que el proceso lleva ejecutándose (TIME), la línea de orden que se escribió para iniciar el proceso (COMMAND), el identificador del usuario que ha inicializado el proceso (UID, *UserIdentification*), el identificador del proceso superior (PPID) y la prioridad de ejecución del proceso (PRI) (Esta prioridad está asignada por el sistema operativo.).

fg o un signo de porcentaje (%) para conectar el teclado a una tarea que se está ejecutando en segundo plano.

Sintaxis: fg [[%]número de la tarea | comando usado para lanzar el trabajo]

jobs permite conocer el estado de los procesos suspendidos y en segundo plano.

Sintaxis: jobs [-l] [[%]lista de números de tarea]

Si no se especifica la lista de números de tarea, el comando jobs muestra el estado de todas las tareas en curso. Con la opción l, también se muestra el PID de los procesos asociados a esas tareas.

kill permite anular un proceso en curso. Para ello es preciso saber el identificador que le corresponde. Este mandato envía la señal de finalización al proceso.

Sintaxis: kill PID

nice se encarga de decirle al sistema cuáles son los trabajos no prioritarios. Éstos tardarán en ejecutarse, pero no molestarán a los restantes trabajos en curso, especialmente a aquellos que se realizan en modo interactivo y que deben siempre favorecerse. El mandato **nice** se antepone al comando al que queremos "quitarle urgencia".

Sintaxis: nice *comando*

nohup, que permite la ejecución de tareas después de desconectarse del sistema. Se debe utilizar con trabajos subordinados, ya que si no, no podríamos finalizar la sesión.

Sintaxis: nohup *comando*&

which. Muestra la ruta absoluta de una utilidad o comando.

Esta utilidad permite localizar la ubicación completa de una utilidad o comando dentro de los directorios de la ruta de búsqueda. Si en la ruta de búsqueda se encuentra más de un programa con el nombre del comando especificado, which sólo muestra la ruta completa del primero, ya que será el que se ejecuta.

Sintaxis: which nombre_comando

whereis. Muestra los archivos relacionados con una utilidad o comando

Esta utilidad busca en la lista de directorios estándar los archivos relacionados con una utilidad.

Sintaxis: whereis nombre_comando

find. Localiza un archivo dentro de una estructura de directorios

Este comando examina toda la estructura de directorios que se especifique, buscando los archivos que cumplan los criterios señalados en la línea de órdenes.

Sintaxis: find directorio_busqueda -[name|mtime] expresión

El argumento *expresión* se usa para indicar los criterios de selección de los archivos a localizar, pudiendo usarse caracteres comodín (*, ?).
Opciones:
· name: únicamente se buscan los archivos cuyo nombre se especifica con *expresión*;
· mtime n: se buscan los archivos que han sido modificados hace *n* días;
· mtime –n: se buscan los archivos modificados en los últimos *n* días;
· mtime +n: se buscan los archivos modificados hace más de *n* días.

bzip2. Comprime un archivo.

Esta utilidad permite comprimir un archivo (que no sea un enlace o directorio) analizándolo y almacenándolo de una forma más eficaz. La nueva versión del archivo tendrá la extensión *bz2* recordando que el archivo está comprimido y que no se puede visualizar hasta que no se descomprima.

Sintaxis: bzip2 [-vkd] nombre_fichero (o lista de nombres de ficheros)

Con la opción v, el comando bzip2 informa del porcentaje que puede reducir el tamaño del archivo. Con la opción k, se crea un nuevo fichero comprimido sin sobrescribir el fichero original, dándole a este nuevo fichero el mismo nombre que el fichero original con la extensión .bz2. Por último, la opción d descomprime el fichero, realizando la misma función que el comando bunzip2.

bunzip2. Descomprime un archivo.

Esta utilidad restaura el archivo que se ha comprimido con bzip2.

Sintaxis: bunzip2 fichero_comprimido.bz2 (o lista de ficheros comprimidos)

tar tiene dos funciones:

1. Almacenar (empaquetar) en un único archivo (llamado archivo tar) todos los ficheros y directorios (junto con su jerarquía de ficheros y directorios) especificados.

Sintaxis: tar -[vcr]f nombre_fichero.tar nombre_fichero_directorio (o lista de nombres de ficheros y directorios)

Opciones:

- f (write): este modificador debe situarse siempre al final de la lista de modificadores e indica que se utilice el nombre_fichero.tar como archivo para empaquetar. Si no se emplea esta opción, el valor predeterminado para archivar es /dev/mto (cinta magnética), dispositivo que puede no estar configurado en la máquina o al que no siempre se tiene permiso para su uso.
- v (verboso): sirve para que se visualicen los archivos y directorios que se van a archivar;
- c (create): crea un nuevo fichero tar donde se empaqueta la estructura especificada;
- r (append): añade los archivos al final del archivo tar actual.

2. Extraer los ficheros y directorios (con su jerarquía de ficheros y directorios) de un archivo tar.

Sintaxis: tar -[v]xf nombre_fichero.tar

Opciones:

- f (read): este modificador debe situarse siempre al final de la lista de modificadores e indica que se utilice el nombre_fichero.tar como archivo para desempaquetar. Si no se emplea esta opción, el valor predeterminado para recuperar es /dev/mto (cinta magnética), dispositivo que puede no estar configurado en la máquina o al que no siempre se tiene permiso para su uso.
- x (extract): extrae los archivos/directorios incluidos en el fichero tar.
- v (verboso): sirve para que se visualicen los archivos y directorios que se van a recuperar.

write. Envía un mensaje.

Esta utilidad permite a un usuario enviar un mensaje a otro usuario que está conectado al sistema, mostrándole dicho mensaje como un aviso en el terminal del otro usuario.

Sintaxis: write nombre_usuario [Terminal]

El nombre de usuario es el login del usuario con el que se desea comunicar, y el terminal (que es opcional) es útil ponerlo si el usuario ha iniciado más de una sesión.
Para finalizar la conversación se debe pulsar Control+D.

mesg. Deniega o acepta mensajes

Este comando sirve para denegar o aceptar la recepción de mensajes procedentes de otros usuarios.

Sintaxis: mesg [n / y]

Con la opción n no se aceptan mensajes procedentes de otros usuarios y con la opción y si se aceptan.
Si se ejecuta el comando sin opciones se puede comprobar cómo tenemos la recepción de mensajes. Si nos muestra el mensaje *is y* si aceptamos los mensajes y si es *is n* entonces no se aceptan.

gcc. Para compilar un programa escrito en C se usa el compilador de C.

Sintaxis: gcc nombre_fichero_programa (con la extensión .c)

Programación en Shell.

Parámetros (argumentos).

Los argumentos se pasan al *script* al ejecutarlo en la misma línea de órdenes:

$ nombre_script argumento1 argumento2 ...
Se pueden hacer referencia a estos argumentos desde el propio script, de la siguiente forma:

$n Representa el valor del argumento que aparece en la posición **n**, también llamados parámetros de posición. Estos parámetros de posición se pueden usar dentro de un script como cualquier otra variable de shell. Es necesario tener en cuenta que el argumento cero, representado por "$0" devuelve el nombre del script., y que a partir del parámetro nueve, la posición debe ir encerrada entre llaves.

$* Representa todos los argumentos excepto el argumento 0.

$@ Representa la cadena de argumentos entera (excluyendo el argumento 0), pero como una lista de cadenas, a diferencia de $* que obtiene todos los argumentos en una única cadena.

$# Representa el número de argumentos sin contar el argumento 0 (el nombre del script).

$$ Devuelve el identificador del proceso (PID) que se está ejecutando.

$? Devuelve un código según el resultado de la operación anterior (estado de salida). Esto puede ser utilizado para controlar si se ha producido algún error.

$! Devuelve el identificador de proceso (PID) del último proceso lanzado como tarea de fondo.

shift nos permite desplazar hacia la izquierda todos los argumentos, perdiéndose cada vez el argumento 1 (esto si el desplazamiento n es igual a 1, valor por defecto). El argumento 0 no se ve afectado. Es muy utilizado dentro de los bucles.

Sintaxis: *shift* n

Instrucciones.

echo

Este comando sirve para visualizar información por pantalla (salida estándar).

Sintaxis: *echo* [opciones] texto

Con esta orden se puede usar los siguientes parámetros para mejorar la visualización:

- –n para que no añada un salto de línea al final del texto que visualiza.
    - echo –n hola // Visualiza en la misma línea que visualiza hola el prompt del sistema
    - echo hola // El prompt del sistema lo visualiza en la siguiente línea
- –e para que interprete los caracteres de barra invertida que contiene el texto a visualizar.

Código	Significado
\b	Retroceso de espacio
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulador horizontal
\v	Tabulador vertical
\a	Alerta (pitido)
\”	Comillas dobles

read

Esta orden sirve para leer una información desde el teclado (entrada estándar) y se la asigna a una variable. Cada instrucción read lee una línea del teclado. Su sintaxis es:

Sintaxis: *read* var1 [var2 var3 .....]

La instrucción asignará a *var1* el primer argumento de la línea introducida, a *var2* el segundo y así sucesivamente. Si se introducen más argumentos que variables haya, todos los datos que sobran por la derecha se asignarán a la última variable de la lista. Los argumentos se separan por blancos o tabuladores.

exit

Se puede utilizar esta orden dentro de los scripts para finalizar inmediatamente su ejecución. Esta orden puede llevar un argumento, que se convierte en el valor que el script devuelve al shell que lo invocó.

if

Sintaxis:

if orden
then
    orden (o varias órdenes)
else
    orden (o varias órdenes)
fi

En primer lugar se ejecuta la orden que va detrás de *if*, si esta ha sido realizada satisfactoriamente (estado de salida cero) se ejecutan las órdenes que aparecen entre *then* y *else*, en caso contrario se ejecutan las que van entre *else* y *fi*.
La parte *else* es opcional:

if orden
then
    orden
        .
fi

Si no se realiza satisfactoriamente la orden que va detrás de *if*, se salta al final y no se ejecuta ninguna orden. Se pueden encadenar esta orden siguiendo la sintaxis siguiente:

if orden1
then
    orden
        .
elif orden2
then
    orden
        .
else
    orden
        .
fi

test

Esta orden permite evaluar expresiones lógicas.

Sintaxis: *test* expresión_lógica o [expresión_lógica]

Si la expresión es verdadera devuelve un estado de salida 0 y si es falsa un estado diferente. Las expresiones lógicas hacen referencia a cadenas, enteros y ficheros. Se suele utilizar a menudo conjuntamente con la orden *if* permitiendo bifurcar nuestros programas en base a una condición, que resulta de evaluar expresiones lógicas. Las expresiones se construyen utilizando *primitivas*, que se pueden referir a cadenas, enteros y ficheros.

Primitivas referidas a cadenas:

-n cadena Devuelve verdadero si la longitud de la cadena no es cero.
-z cadena Devuelve verdadero si la longitud de la cadena es cero
S1 = S2 Compara las dos cadenas S1 y S2. Devuelve verdadero si son iguales.
S1 != S2 Compara las cadenas S1 y S2. Devuelve verdadero si son diferentes.
Primitivas referidas a enteros:
n1 -eq n2 Devuelve verdadero si ambos enteros son iguales numéricamente.
n1 -ne n2 Devuelve verdadero si ambos enteros son diferentes numéricamente.
n1 -gt n2 Devuelve verdadero si numéricamente n1 es mayor que n2.
n1 -ge n2 Devuelve verdadero si numéricamente n1 es mayor o igual que n2.
n1 -lt n2 Devuelve verdadero si numéricamente n1 es menor que n2.
n1 -le n2 Devuelve verdadero si numéricamente n1 es menor o igual que n2.

Primitivas referidas a ficheros:

-d fichero Devuelve verdadero si el fichero existe y es un directorio.
-f fichero Devuelve verdadero si el fichero existe y no es un directorio.
-s fichero Devuelve verdadero si el fichero existe y tiene un tamaño mayor que cero (no está vacío).
-r fichero Devuelve verdadero si el fichero existe y tenemos permiso de lectura.
-w fichero Devuelve verdadero si el fichero existe y tenemos permiso de escritura.
-x fichero Devuelve verdadero si el fichero existe y tenemos permiso de ejecución.

Las primitivas anteriores se pueden combinar con los siguientes operadores booleanos:
! Operador unario *negación*.
-a Operador binario *and*.
-o Operador binario *or*.
(expresión) Se usan los paréntesis para realizar agrupamientos.

OPERADORES: && y ||

El shell posee también estos operadores para la ejecución condicionada de las instrucciones:

- Operador **&&**, cuya sintaxis es: orden1 **&&** orden2
  Se ejecuta la primera orden (*orden1*), si devuelve un estado de salida 0 se ejecuta la segunda orden (*orden2*).

- Operador **||**, cuya sintaxis es: orden1 **||** orden2
  Se ejecuta la primera orden (*orden1*), si devuelve un estado de salida distinto de cero, se ejecuta la segunda orden.

Se puede observar que en ocasiones estos operadores son equivalentes a la orden **if**. Por ejemplo:

```
if cat nombre_fichero
then
        echo correcto
else
        echo incorrecto
fi
```

Es equivalente a:

cat nombre_fichero && *echo* correcto || *echo* incorrecto

case

Esta orden permite comparar una variable frente a una serie de valores posibles.

Sintaxis: *case* valor *in*

```
            patron1) serie de mandatos 1;;
            patron2) serie de mandatos 2;;
            ......
            *) serie de mandatos n;;
    esac
```

Donde **esac** es la palabra-clave para el final de la instrucción **case**, y ***)** representa el caso por defecto. Esta instrucción **case** orienta las operaciones dependiendo de si *valor* es igual a *patron1*, *patron2*,... o distinto. Si el *valor = patron1*, entonces se ejecuta la serie de mandatos 1. Si el *valor = patron2*, se ejecuta la serie de mandatos 2, y así sucesivamente. Un doble punto y coma (;;) sirve para finalizar cada elección de **case**.
Los patrones pueden ser cualquier expresión regular. Una lista de patrones se establece al separarlos por el símbolo "**|**". Por ejemplo, *[aA]*e|jj?|u** es una lista de tres patrones.

El uso de **case** puede ser equivalente a usar operadores lógicos o varios **if**. Sin embargo, la utilización de **case** puede hacer más legible el script, además permite hacer comparaciones con patrones y no solamente con valores determinados.

expr

Esta orden nos permite realizar algunas operaciones aritméticas con enteros. Las operaciones aritméticas que podemos realizar son: suma (**+**), resta (**-**), multiplicación (*****), división (**/**) y módulo (%). Es necesario separar los operandos con un espacio.

Sintaxis: *expr* arg1 op arg2 [op arg3 …]

Por ejemplo:

expr 4 + 5 (devolvería 9 por la salida estándar)

Si queremos asignar el resultado de una expresión a una variable, debemos encerrar la expresión entre comillas invertidas simples (`) o entre paréntesis anteponiéndole el símbolo $. Por ejemplo:

NETO=`expr 100 - 80` o NETO=$(expr 100 - 80)
echo $NETO (mostraría 20)
echo `expr 100 – 80` o echo $(expr 100 – 80) (también mostraría 20)

Si se utilizaran las comillas dobles, lo consideraría texto y no ejecutaría la orden **expr**, por tanto, la instrucción *echo "expr 100 – 80"* mostraría por pantalla la cadena: *expr 100 - 80*

Nota: *En algunos casos, puede ser necesario indicar al shell que no interprete los caracteres* ***/*** *y* *******. Para eso se utiliza la barra invertida justo antes del carácter. Por ejemplo:* ***expr 2 * 6***

For

Sintaxis: *for* variable in lista de valores (valor1 valor2 ...)

```
do
        orden1
        orden2
        .
        .
done
```

Las palabras-clave **for**, **do** y **done** deben colocarse al principio de la línea. La variable de iteración puede tener cualquier nombre. La serie de instrucciones entre **do** y **done** se ejecutará para cada elemento de la lista de valores.
Veamos un ejemplo:

```
for i in 1 2 3 4 5
do
        echo el valor de la variable i en este paso es:
        echo $i
        echo
done
```

Cuando no se establecen de antemano los valores que va a tomar la variable del bucle, se asume que van a ser los argumentos del script, es decir, el parámetro de posición $@ que representa la cadena de argumentos entera excluyendo el nombre del programa.

while

Sintaxis: while lista_órdenes

do
	orden1
	orden2
	.
done

Se ejecuta la lista de órdenes (separadas por **;**) que aparece detrás de **while**. Si la última devuelve un estado de salida 0 (sin errores en la ejecución de la orden o devuelve "true" una orden *test*), se ejecutarán las órdenes entre **do** y **done**. A continuación se ejecuta de nuevo la lista de órdenes y si la última devuelve un estado de salida 0 se ejecutan las órdenes entre **do** y **done** otra vez. Esto se repite indefinidamente hasta que la última orden de la lista devuelva un estado de salida distinto de 0.
Veamos un ejemplo:

```
cont=0
while [ $cont –lt 10 ]
do
	echo El contador es $cont
	cont=`expr $cont + 1`
done
```

until

Sintaxis: *until* lista_órdenes

do
	orden1
	orden2
	.
done

Se ejecuta la lista de órdenes que aparece detrás de **until**. Si la última devuelve un estado de salida distinto de 0 se ejecutan las órdenes entre **do** y **done**. A continuación se ejecuta de nuevo la lista de órdenes y si la última devuelve un estado de salida distinto de 0 se ejecutan las órdenes entre **do** y **done** otra vez. Esto se repite indefinidamente hasta que la última orden de la lista devuelva un estado de salida igual a 0.
Veamos un ejemplo:

```
cont=20
until [ $cont –eq 10 ]
do
	echo contador $cont
	cont=`expr $cont - 1`
done
```

Se puede decir, por tanto, que las órdenes **while** y **until** son complementarias.

Existen dos órdenes **true** y **false**, que al ejecutarse lo único que hacen es devolver un estado de salida 0 y un estado de salida no 0, respectivamente. Si se utilizan estas órdenes junto a las órdenes **while** y **until** se pueden realizar bucles infinitos.

break y continue

Se utilizan junto a las órdenes **for**, **while** y **until**. La orden **break** finaliza el bucle en el que aparece. En el siguiente ejemplo la orden **break** aborta la ejecución de un bucle infinito.

```
while true
do
        break
done
```

La orden **continue** vuelve al comienzo del bucle en el que aparece. El siguiente ejemplo imprime indefinidamente "si", no llegando nunca a ejecutar "echo no":

```
while true
do
        echo si
        continue
        echo no
done
```

Tanto **break** como **continue** pueden llevar un argumento numérico. Ese número indica el número de bucles sobre el que tiene efecto su acción. Supóngase el siguiente ejemplo:

```
for a in 1 2 3
do
        echo $a
        for b in a b c
        do
                for c in d e f
                do
                        break 2
                        echo $c
                done
                echo $b
        done
done
```

Este ejemplo devuelve: 1, 2, 3. Esto se debe a que la orden *break 2* finaliza los dos bucles más próximos en que está incluida (los bucles referidos a las variables "b" y "c").