Tema 4.
Interprete de comandos. Bash

Interprete de comandos shell.

El programa que se encarga de pedirnos las órdenes de sistema se llama Shell, ó intérprete de comandos (mandatos). Este no solo se dedica a traducir nuestras órdenes sino que además nos ofrece un lenguaje de programación que comúnmente se denomina lenguaje de scripts.

Los shell en Unix se clasifican en dos grandes grupos: los tipo **Bourne** y los tipo **C**. Los shell tipo Bourne toman su nombre a partir de su inventor, Steven Bourne. Steven Bourne escribió el shell original de Unix, denominado sh; a partir de entonces, la mayoría de los shells tienen un nombre con el sufijo sh para indicar que son extensiones de la idea original. Existen varias implementaciones de este shell, que colectivamente llevan el nombre de shells Bourne. También son comunes los shells tipo C cuyo original fue implementado por Bill Joy. Tradicionalmente, los shell Bourne se han utilizado para los scripts de shell y por razones de compatibilidad con el sh original, mientras que los shells C han sido más comunes en su aplicación interactiva. (Los C tienen ventajas en cuanto a sus mejores características interactivas, aunque son más difíciles de programar.)

Linux viene con un shell Bourne denominado bash, escrito por la organización "*Free Software Foundation*". El nombre bash proviene de Bourne Again SHell, uno de los tantos juegos de palabras en Unix. Se trata de un shell Bourne "avanzado": tiene las capacidades estándar de programación que se encuentran en todos los shells Bourne y además varias de las características interactivas que se encuentran en los shells C. bash es el shell predeterminado cuando uno usa Linux.

En SUSE 9.0 disponemos de los siguientes shell.

- ash
- tcsh o csh
- sh
-

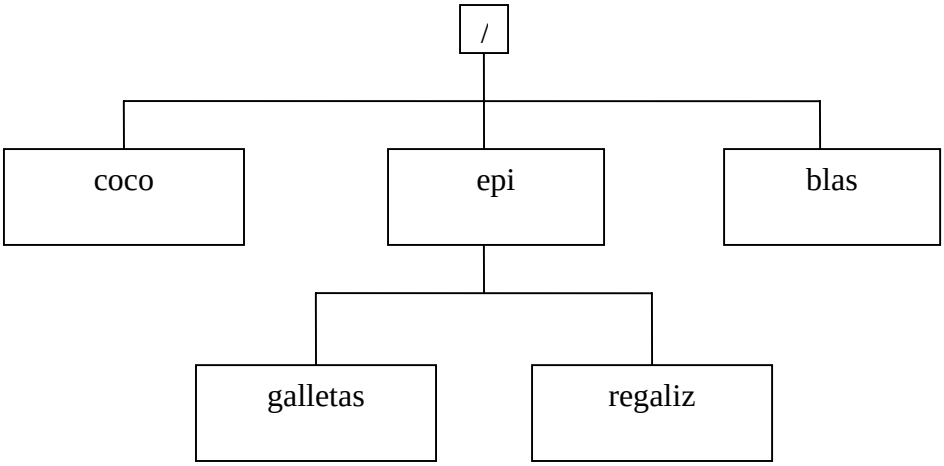
Para analizar ciertos comandos Unix, teclee Ctrl-d. que es el carácter EOF, para abreviar. Puede que en ciertos libros de texto aparezca como end-of-text. Nos referiremos a este carácter como EOF. Es un carácter de control que informa a los programas Unix que ha cesado el ingreso de datos.

Las rutas para localizar archivos y directorios pueden ser absolutas o relativas, se ha de usar en cada caso la más apropiada. Notar que las absolutas son aquellas que parten del directorio raíz y que las relativas son las que parten del directorio donde nos encontremos.

./ Directorio actual

../ Directorio inmediatamente superior.

Veamos unos ejemplos sobre esto:



Partiendo del directorio raíz, para indicar el directorio regaliz tenemos dos opciones:

/epi/regaliz → Ruta absoluta al comenzar por /
epi/regaliz → Ruta relativa al no comenzar por / sino por el directorio actual
./epi/regaliz → Ruta relativa al no comenzar por / sino por el directorio actual .

Supongamos ahora que estamos en galletas y queremos referirnos a blas:

/blas → Ruta absoluta al comenzar por /
../../blas → Ruta relativa indicando que desde donde este suba 2 directorios (hasta raiz) y luego descienda a blas

Y ahora que estando en regaliz queremos ir a galletas:

/epi/galletas → Ruta absoluta al comenzar por /
../galletas → Ruta relativa indicando que desde donde este suba 1 directorio (hasta epi) y luego descienda a galletas

**NOTAS: Aquí comentar que en el shell, si tenemos el comienzo de un archivo o directorio escrito, al pulsar el tabulador, se auto completa el nombre del archivo o directorio, y en caso de haber mas de una posibilidad, pita, y pulsando la segunda vez, nos aparecen las posibilidades, pudiendo continuar en el mismo punto en que lo dejamos.*

```
suse9:~ # cd .
.           .fvwm       .gstreamer  .mcop       .skel
..          .gconfd     .kbd        .mozilla
.fonts      .gnupg      .kde        .qt
suse9:~ # cd .g
.gconfd     .gnupg      .gstreamer
suse9:~ # cd .g_
```

Otra de las ayudas o atajos que nos ofrece linux, es la posibilidad de recuperar comandos anteriormente ejecutados, para ello simplemente hemos de pulsar las flechas de arriba y abajo y veremos como podemos navegar por los anteriores comandos.

Sintaxis de comandos

La sintaxis para ejecutar un comando en UNIX es:

Comando [opciones ...]

Existen una serie de operadores que nos van a permitir unir la ejecución de dos o mas comandos en una sola línea, de diversas formas. Veamos cuales:

El símbolo **;** permite que ejecutemos de forma **secuencial** 2 o mas comandos. Es decir, se ejecuta el primero, y después el segundo y así sucesivamente, independientemente de si alguno de ellos devuelve algún error.

Comando 1 ; Comando 2

El símbolo **&&** Permite ejecutar el segundo comando solo si el primero ha tenido éxito. El éxito o fracaso de un comando se decide en función de un valor numérico devuelto al SO por el propio comando[1].

Comando 1 && Comando 2
mount /floppy && echo "Hay un disco en la disquetera" && umount /floppy

El símbolo || Permite ejecutar el segundo comando solo si el primero ha fallado.

Comando 1 || Comando 2
mount /floppy || echo "Error, no hay disquete metido"

Si lo que nos interesa es conocer el valor exacto devuelto por la ejecución de un comando podemos usar echo $?. La interpretación que se hace sobre este valor en cuanto a verdadero o falso es:

=0 es verdadero o ejecución con éxito
<>0 es ejecución fallida. Puede ser un 1 o cualquier otro valor. Ese valor tiene su significado, y suele indicar el porque de su no éxito.

Un ejemplo de esto:

```
suse9:~ # grep "/dev" /etc/fstab
/dev/sda2            /                    reiserfs   defaults              1 1
/dev/sda1            swap                 swap       pri=42                0 0
devpts               /dev/pts             devpts     mode=0620,gid=5       0 0
/dev/cdrom           /media/cdrom         auto       ro,noauto,user,exec   0 0
/dev/dvd             /media/dvd           auto       ro,noauto,user,exec   0 0
/dev/fd0             /media/floppy        auto       noauto,user,sync      0 0
suse9:~ # echo $?
0
suse9:~ # grep "/d                " /etc/fstab
suse9:~ # echo $?
1
suse9:~ # grep /etc/fstab

suse9:~ # echo $?
130
suse9:~ # _
```

En el ultimo caso hemos pulsado CTRL+C.

[1] Para los que estén familiarizados con la programación en C, es el valor que ponemos en el **return** de la función main o en la llamada a **exit**().

Dentro de los parámetros que podemos pasar a los comandos, existe lo que se denominan comodines, los cuales nos permiten elegir un grupo de valores para tratar por igual a varios ficheros ó directorios. Estos son:

* Vale cualquier texto y de cualquier longitud
? Vale cualquier símbolo, pero solo de un carácter
[] Sustituye un carácter por cualquiera de los dados entre corchetes. Permite dar rangos o valores concretos. Para ello se basa en la tabla ASCII.

Ejemplos.

*ls op** → *Lista todos los archivos del directorio que comienzan por op*
operacion1
opcion.txt
oposicion.doc
...

*ls *txt* → *Lista todos los archivos del directorio que finalizan en txt*
datos.txt
gestiontxt
....

ls backup??092002 → *Lista todos los archivos que coincidan con el texto dado y en los interrogantes puede ir cualquier valor*
backup01092002
backup08092002
backupal092002
...

*ls [a-z]** → *Lista todos los archivos que comienzen por una letra minúscula.*
algo
datos.txt
...

En UNIX/LINUX cualquier archivo que comience por punto se considerará oculto y no aparecerá en el listado de directorio salvo que se le indique lo contrario (ls -a).

Ejecución en 2º plano

Si nos fijamos un poco, al ejecutar un comando, el sistema toma el control y hasta que no termina no nos le devuelve, salvo que cambiemos de consola. Para evitar esto, sobre todo cuando tenemos que ejecutar un comando que puede tardar un rato existe la ejecución de comandos en 2º plano o Backgroung. Para ello existen dos formas:

La primera es añadir al final del comando el símbolo &.

```
tar czvf backup.var.gz.tar /var &
```

Este ejemplo crea una copia de seguridad del directorio /var pero lo hace devolviendo el control de la consola, y cuando termina de ejecutarlo, nos avisa.

Cada usuario tiene una lista de procesos pendientes o en ejecución, y cada proceso de esa lista recibe un nº que le identifica en ella (1, 2, 3,...).

La otra opción es ejecutar el comando tal cual, a continuación pulsamos la combinación de teclas Ctrl. + Z. Esto realiza una pausa en la ejecución del proceso y lo mete en la lista de procesos pendientes asignándole un nº que nos aparece en pantalla. A continuación ejecutamos el comando bg, indicando como parámetro el nº del proceso que queremos lanzar.

Ejemplo

```
> tar czvf backup.var.gz.tar /var
ctrl + Z
[1] stopped          tar czvf backup.var.gz.tar /var
> bg 1
>
```

Al cabo de un rato, cuando termine, nos indicará que el proceso ha concluido.
Probar con el comando yes. Lanzarle a 2º plano y recuperarle.

Bueno, se acabó. Ahora, ejecute unos cuantos trabajos simultáneamente, como estos:

```
/home/larry$ yes > /dev/null &
[1] 1024
/home/larry$ yes | sort > /dev/null &
[2] 1026
/home/larry$ yes | uniq > /dev/null
[y aqui, pulse Ctrl-z para suspenderlo, por favor]
[3]+ Stopped yes | uniq >/dev/null
/home/larry$
```

Cada uno indica su número de trabajo. Los dos primeros también muestran sus números de identificación de proceso, o PID, después del número de trabajo. Los PIDs normalmente no son algo que se necesite conocer, pero a veces viene bien. Matemos el segundo, ya que creo que está ralentizando su máquina. Se puede teclear ***kill %2***, pero eso será demasiado fácil. Por el contrario, haga esto:

```
/home/larry$ fg %2
yes | sort >/dev/null
[pulse Ctrl-c para matarlo]
/home/larry$
```

Como esto demuestra, fg toma parámetros empezando con **%**. De hecho, se podrá teclear solo esto:

```
/home/larry$ %2
yes | sort >/dev/null
[pulse Ctrl-c para matarlo]
/home/larry$
```

Esto funciona por que el shell automáticamente interpreta un número de trabajo como una petición para poner ese trabajo en primer plano. Se puede indicar los números de trabajo con otros números precedidos por un **%**. Ahora teclee ***jobs*** para ver cuáles trabajos quedan en ejecución:

```
/home/larry$ jobs
[1]- Running yes >/dev/null &
[3]+ Stopped yes | uniq >/dev/null
/home/larry$
```

El **-** indica que ese trabajo número 1 es segundo en la lista para ser puesto en el primer plano, si solo se teclea ***fg*** sin dar parámetros. El **+** indica que el trabajo especificado es el primero en la lista un ***fg*** sin parámetros pondrá al trabajo número 3 en el primer plano. Sin embargo, se puede acceder a él llamándolo, si se desea, mediante:

```
/home/larry$ fg %1
yes >/dev/null
[ahora pulse Ctrl-z para suspenderlo]
[1]+ Stopped yes >/dev/null
/home/larry$
```

Al cambiar al trabajo número 1 y luego suspenderlo han cambiado las prioridades de todos los trabajos de usuario. Esto se puede ver con el comando jobs:

```
/home/larry$ jobs
[1]+ Stopped yes >/dev/null
[3]- Stopped yes | uniq >/dev/null
/home/larry$
```

Ahora los dos están parados (porque los dos se han suspendido con Ctrl-z), y el número 1 es el siguiente en la lista a entrar en el primer plano por defecto. Esto es así porque se le puso en el primer plano manualmente, y luego fue suspendido. El + siempre se refiere al trabajo más reciente que ha sido suspendido del primer plano. Se puede continuar con su ejecución otra vez:

```
/home/larry$ bg
[1]+ yes >/dev/null &
/home/larry$ jobs
[1]- Running yes >/dev/null
[3]+ Stopped yes | uniq >/dev/null
/home/larry$
```

Fíjese que ahora está en ejecución, y el otro trabajo se ha movido en la lista y tiene el **+**. Ahora matémoslos para que el sistema no esté permanentemente ralentizado por procesos que no hacen nada.

```
/home/larry$ kill %1 %3
[3] Terminated yes | uniq >/dev/null
/home/larry$ jobs
[1]+ Terminated yes >/dev/null
/home/larry$
```

Aparecerán varios mensajes sobre la terminación de los trabajos, nada muere tranquilamente, al parecer.

Redireccionamiento

Los shell, poseen la capacidad de redireccionar la entrada y salida estándar. Esto nos va a permitir poder guardar, por ejemplo, el resultado en pantalla de un comando en un fichero de texto, o bien que un comando en vez de recibir la información desde teclado la reciba desde un fichero de texto. Los símbolos utilizados son:

> Desvía la salida estándar a un fichero. Si existe el fichero le sobre escribe.
>> Como el anterior pero si el fichero existe, añade la información al final.
< Entrada estándar.

```
ls -a > listado.txt
```

Además de estos existen unos especiales que nos van a permitir redireccionar la salida pero de un dispositivo estándar concreto. Esto se hace por ejemplo para evitar que aparezcan en pantalla múltiples mensajes de error,...

```
find / -name "*.h" 2> /dev/null
```

Desvía todas las salidas destinadas al dispositivo estándar nº 2 (errores), al pozo negro.

Para probar la entrada estándar hemos de echarle algo de imaginación.

```
cat < a.txt > b.txt
```

¿Sabéis que haría esto?[2] Un ejemplo algo mas útil lo podemos ver en estos comandos:

```
at 4pm < a.txt
mail cesarsan < texto.txt
```

El primero ejecuta las tareas indicadas en el archivo a.txt a las 16:00. El segundo envía un email al usuario cesarsan con el contenido de texto.txt

Tuberías o pipes.

El símbolo | nos permite enlazar comandos sin necesidad de usar ficheros intermedios, es decir, puedo enlazar la salida de un comando con la entrada de otro.

```
ls -a | sort +2 | tail –1
```

Comandos típicos en linux

Comandos de manejo de directorios.

ls *[opciones] directorio*

[2] Copiar a.txt en b.txt

Lista el contenido de un directorio pasado por parámetro. Según que opciones le demos, nos mostrará la información en distintos formatos. Entre sus opciones podemos destacar: -i para ver el nº de inodo; -a para ver archivos ocultos; -l para ver detalles; -F para ver el tipo de archivos; ...

```
$ ls
arch1          arch2          ....
arch12  arch13  ....
....
$ ls -F
arch1          arch2          dir1/
....
$
```

***cd** [ruta]*

Comando que nos permite movernos por el árbol de directorios de LINUX. Admite tanto rutas relativas como absolutas.

```
/home/kaesar$ cd /
/$cd $HOME
/home/kaesar$ cd ../peter
/home/peter$
```

***mkdir** nombre1 nombre 2 nombre3 ...*

Comando que crea un directorio con el nombre dado. Al igual que el comando cd, también admite rutas relativas. Admite varios nombres de directorio en una sola línea.

***rmdir** nombre*

Comando que borra el directorio con el nombre dado. Al igual que el comando cd, también admite rutas relativas. No permite borrar directorios que no estén vacíos.

```
        /home/larry/report-1993$ rmdir .
rmdir: .: Operation not permitted
/home/larry/report-1993$
```

Comandos de visualización

***at** fichero1 [fichero2]...*

Comando para visualizar el contenido de un fichero de texto en pantalla. Se creó en un principio para concatenar archivos en uno solo.

```
$ cat archivo1
$ cat > miArchivo.txt
.....
Ctrl. + D
$ cat < miArchivo.txt
```

***more** [-l] [+número línea] [fichero1 fichero2 . . . ficheroN]*

more es más útil, y además es el comando recomendado para ver ficheros de texto ASCII. La única opción interesante es **-l**, que indica al comando que no se desea interpretar el carácter Ctrl-L como carácter de nueva página. El comando comenzará en la línea número línea especificada.

Al ser more un comando interactivo, hemos resumido a continuación las órdenes más comunes:

Barra espaciadora Pasar a la siguiente pantalla de texto.

d Pasar 11 líneas de pantalla, o aproximadamente la mitad de pantalla

/ Busca una expresión regular. La construcción de una expresión regular puede ser muy complicada por lo que es recomendable teclear simplemente el texto a buscar. Por ejemplo, /sapo. Intro Buscara la primera ocurrencia de \sapo" en el fichero a partir de la posición actual.

q Terminar more.

less *fichero*

Comando que visualiza el contenido de un fichero en pantalla pero lo pagina. Mas completo que more.

```
$ less miArchivo.txt
....
$
```

echo *texto*

Muestra en pantalla el texto que se le indica. Se suele utilizar en scripts para visualizar mensajes y también en consola para ver el contenido de alguna variable.

```
$ echo hola
hola
$ echo $HOME
/home/kaesar
$
```

head *[-líneas] [fichero1 fichero2 . . . ficheroN]*

head mostrará las primeras diez líneas de los ficheros especificados, o las primeras diez líneas de la *stdin* si no se especifica ningún fichero en la línea de comandos. Cualquier opción numérica se tomará como el número de líneas a mostrar, por ejemplo *head* -15 rana mostrará las primeras quince líneas del fichero rana.

tail *[-líneas] [fichero1 fichero2 . . . ficheroN]*

Como head, tail mostrará solo una parte del fichero, en este caso el final del fichero, las últimas diez líneas del fichero, o que provengan de la stdin. tail también acepta la opción de especificar el número de líneas, como en el caso anterior.

***file** [fichero1 fichero2 . . . ficheroN]*

El comando *file* intenta identificar que tipo de formato tiene un fichero en particular. Debido a que no todos los ficheros tienen extensión o otras formas de identificarlos fácilmente, este comando realiza algunas comprobaciones rudimentarias para intentar comprender exactamente que contiene el fichero.

Hay que tener cuidado ya que es bastante posible que *file* realice una identificación incorrecta.

***wc** [-clw] [fichero1 fichero2 . . . ficheroN]*

El comando wc simplemente cuenta el número de palabras, líneas y caracteres en los ficheros pasados como parámetros. Si no se especifica ningún fichero, operará sobre la stdin. Los tres parámetros, **-clw**, indican el elemento a contar: **-c** bytes, **-m** caracteres, **-l** líneas y **-w** para palabras. Por ejemplo, **wc -cw** contará el número de caracteres y palabras, pero no el número de líneas. Si no se indica ningún parámetro wc cuenta todo: palabras, líneas y caracteres. Se puede utilizar wc para saber el número de ficheros de un directorio: **ls | wc -w**. Si se desea saber el número de ficheros que acaban en c, entonces ejecute **ls *.c | wc -w**.

***cmp** fichero1 [fichero2]*

cmp compara dos ficheros. El primero debe ser obligatoriamente pasado como parámetro, mientras que el segundo puede ser pasado como un segundo parámetro o leído desde la entrada estándar. *cmp* es muy sencillo, y simplemente dice donde se diferencian los dos ficheros.

***diff** fichero1 fichero2*

Uno de los comandos estándar más complejos de UNIX/LINUX es el diff. La versión GNU de diff tiene hasta veinte opciones en la línea de comandos. Es una versión mucho más potente que cmp y muestra cuales son las diferencias en lugar de decir simplemente donde está la primera. Se usa para generar los parches del kernel y otras fuentes.

***sort** +n -u*

Comando que nos ordena un fichero de texto por la columna que le indiquemos. En caso de no indicar columna ordena por la primera. Supone nueva columna al encontrarse con separadores de espacio o de tabulación. Con la opción –u elimina los duplicados.

grep <opciones> patrón ruta

Comando que me permite analizar el contenido de un fichero de texto y quedarme solo con las líneas que cumplan un determinado patrón. Por defecto muestra todas las líneas coincidentes y si no las hay, no muestra nada. Entre las opciones interesantes, tenemos:

-v: invierte la búsqueda, de forma que muestra las líneas que no coinciden con el patrón.

-c: Nos muestra solo el nº de líneas coincidentes.

-l: Solo muestra el nombre del archivo donde lo encuentra. Útil cuando le decimos que busque en varios archivos.

split

Comando que parte un fichero de texto en partes iguales.

csplit

Como el anterior pero corta al encontrarse un patrón indicado.

find <ruta_donde_buscar> opciones

Para buscar archivos que cumplan un determinado patrón, permite además búsquedas por fecha de última modificación.... Algunas de las opciones son:

-name <patron> → Para buscar por el nombre del archivo
-maxdepth <nivel> → Indicamos cuantos sub directorios debe descender.
-user <valor> →pertenece a ese usuario (nombre o UID)
-group <valor> → pertenece a ese grupo (nombre o GID)
-amin <valor> →Accedido en ese periodo (minutos) [3]
-atime <valor> →Accedido en ese periodo (dias)
-mmin <valor> →Modificado en ese periodo (minutos)
-mtime <valor> →Modificado en ese periodo (horas)
-size[1] <valor> [bkcw]→ Con el tamaño indicado: b, en bloques de 512 bytes; k, en kilobytes; c, en bytes; w, en palabras de 2 bytes.

Manejo de ficheros.

***cp** arch1 arch2 ...*

Comando que copia archivos desde un origen a un destino. Para n parámetros pasados, interpreta los n-1 como origen y en enésimo como destino. Admite símbolos comodín. Con el parámetro **-i** pide confirmación.

```
/home/larry$ ls -F
frog passwd
/home/larry$ mkdir passwd_version
/home/larry$ cp frog passwd passwd_version
/home/larry$ ls -F
frog passwd passwd_version/
/home/larry$ ls -F passwd_version
frog passwd
/home/larry$
```

***rm** [-frvi] arch1 arch2 ...*

Comando que borra los archivos indicados. Hay que tener cuidado porque no pide confirmación para borrar. En caso de que alguno de los archivos pasados no exista o no se pueda borrar, seguirá intentando borrar el resto de archivos. Admite símbolos comodín. En cuanto a sus parámetros :con el parámetro **-i** pide confirmación; Con -r borra recursivamente; con -f no pide confirmación; con -v muestra información del proceso.

***mv** origen destino*

Comando que nos permite mover uno o varios archivos a través de nuestro árbol de directorios. También nos va a permitir cambiar de nombre. Para mover un archivo, como destino damos un directorio distinto al original y para renombrar simplemente indicamos

[3] Para los periodos de tiempo se puede poner +,- o nada delante del número. Esto indica
- -amin -5 accedido en los últimos 5 minutos
- -amin +5 accedido hace mas de 5 minutos
- -amin 5 accedido hace 5 minutos exactamente.

el nuevo nombre, acompañado si queremos de un directorio. Si existe un archivo destino, le machaca. Con el parámetro **-i** pide confirmación.

touch *fichero1 fichero2 . . . ficheroN*

Actualiza los registros de fecha y hora con la fecha y hora actual de los ficheros indicados en la línea de comandos. Si el fichero no existe, touch lo creará. También es posible especificar la fecha y hora a registrar en la información de los ficheros.

chmod *[-Rfv] modo fichero1 fichero2 . . . ficheroN*

Permite cambiar los permisos de acceso a un fichero o grupo de ficheros. Admite comodines. Los parámetros nos permiten: -R, recursivo para los archivos contenidos en el directorio; -f Para que intente modificar aunque no seamos propietarios; -v para mostrar información. Existen dos formas de cambiar permisos, una con números, donde se usa una codificación octal para cada grupo de permisos.

---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

Con esto tendríamos, por ejemplo

/home/larry$ chmod 640 peter.c → (rw- Propietario) (r-- Grupo) (--- Resto)

La otra forma de trabajar me permite cambiar solo uno o varios de los parámetros, para eso basa en tres símbolos: + - =, así como de las iniciales de cada grupo de permisos u g o.

/home/larry$ chmod u+x peter.c → (rwx Propietario) (r-- Grupo) (--- Resto)
/home/larry$ chmod g+x peter.c → (rwx Propietario) (r-x Grupo) (--- Resto)
/home/larry$ chmod o+rx peter.c → (rwx Propietario) (r-x Grupo) (r-x Resto)

Además de esto, el comando chmod me permite establecer el valor de 2 bits especiales. Sticky bit y el SUIG/SGID.

El Sticky bit para un directorio permite que solo el root y el propietario puedan borrar o renombrar los archivos que se contienen en el. Antiguamente se usaba para que un archivo una vez usado se quedara residente, para que en posteriores usos su carga fuera mas rápida.

El bit SGID y SUID permiten, para un archivo ejecutable, que este, al ser ejecutado asuma la identidad del propietario o grupo-propietario, de forma que el sistema trabaje con los privilegos de este, en lugar de quien lo ejecuta.

umask [-p][-S] <modo>

Este comando nos permite establecer los permisos iniciales que tendrán los nuevos elementos del sistema (directorios, ficheros,…). Si le ejecutamos sin parámetros, nos

muestra la mascara actual (por defecto es 0022, que equivale en directorios a rwxr-xr-x y en ficheros a rw-r--r--). La forma de trabajar de umask es distinta según sea el tipo de elemento a crear. Por ejemplo:

Mascara	Fichero	Directorio
0022	rw-r--r--	rwxr--r--

La razón de esto es que cuando creamos un nuevo elemento se realiza una operación XOR entre la máscara y el valor 666 para ficheros y 777 para directorios.

chattr [+-=]aA

Para asignar atributos especiales a los ficheros. Solo funciona en sistemas de ficheros ***ext***. Entre los modos disponibles:

a: Para que un fichero solo pueda abrirse en modo append. Esto solo puede hacerlo el root.
i: Archivo inamovible. No puede ser renombrado ni borrado.
s: Si un archivo con este atributo es borrado, antes de liberar sus bloques de disco, estos son sobrescritos con ceros.

lsattr

Muestra los atributos del fichero.

cut *opciones fichero*

Comando que me permite eliminar campos (columnas) de un fichero. Trabaja tanto a nivelde bits como de texto, por lo que es bastante versátil. Los parámetros típicos de uso son:
-f para indicar los campos que queremos seleccionar. Se indican separados por coma o mediante guión para indicar rangos. Si se indica mas de un campo aparecen los separadores.
-d para indicar que símbolo es el delimitador. Por defecto es TAB.
-s para ignorar líneas que no tienen separador

```
cut –f 3 –d “:” /etc/passwd
```

Muestra solo los UID del archivo.

```
grep “manuel:” /etc/passwd | cut –f 3 –d " :"
```

Muestra el UID del usuario manuel. Es equivalente al comando *id manuel -u*

Miscelánea.

expr *expr1 operador expr2*

Comando que nos permite realizar una operación matemática ó lógica. Los operadores disponibles son:

+ - / * \> < >= <= = !=

```
$> expr 5 + 6
11
```

```
$> expr 5 > 6
0
$> expr 5 \* 6
30
$> expr 6 < 12
1
$>
```

alias *pseudonimo=orden*

Esta orden permite crear un alias, seudónimo, para alguna orden. Para que esta orden quede permanente hay que incluirla en el archivo "*.bashr*" o en el local.

unalias *nombre*

Esta orden elimina un alias (si fue incluido en .bashr no lo elimina), para eso vas a tener que borrarlo de ahí.

env

Esta orden nos muestra las variables de entorno, por ejemplo sirve para ver que directorios están en el PATH.

logout

Esta orden te permite salir del sistema como usuario actual y entrar de nuevo como el usuario que quieras o puedas.

gzip

Comprime o descomprime archivos. Posee varios grados de compresión, siendo –9 el máximo. Con –d descomprime.

gzip –9 fichero.txt -> Comprime al máximo.
gzip –d fichero.txt.gz -> Descomprime

tar opciones archivo.tar <origen>

Es el empaquetador estándar en sistemas unix. Con el podemos unir varios archivos en uno solo con la posibilidad de comprimirlos o no. Entre sus usos están los backups o la distribución de software con fuentes (descargas de internet). Por convenio, se suele poner .tar para archivos solo empaquetados y tar.gz para los empaquetados y comprimidos.

Entre las opciones tenemos:

- x → Para extraer los archivos de un tar
- c → Para crear un tar
- r → Añade archivos al final del paquete
- t → muestra el contenido de un archivo
- u → añade solo archivos con fecha posterior a la del paquete. Y solo sirve si no está comprimido

- z → Indica que se debe tener en cuenta la compresión zip
- v → Mostrar información de las tareas realizadas según se ejecuta. c
- f → Indica que el siguiente parámetro es el archivo tar

Ejemplos:
Esto crea una copia comprimida de todo el directorio home, en el directorio en que se esta ejecutando.

tar czvf copia_home.tar.gz /home

Si lo queremos descomprimir en el directorio actual.

tar xzvf copia_home.tar.gz

cal [-3my [[mes] año]

Muestra un calendario con el mes o año, indicado.

cal –3 2 2004

```
    January 2004          February 2004          March 2004
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7      1  2  3  4  5  6
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   7  8  9 10 11 12 13
11 12 13 14 15 16 17  15 16 17 18 19 20 21  14 15 16 17 18 19 20
18 19 20 21 22 23 24  22 23 24 25 26 27 28  21 22 23 24 25 26 27
25 26 27 28 29 30 31  29                    28 29 30 31
```

Ejercicios

Los siguientes ejercicios deberan realizarse en una sola linea de comando. Salvo que se indique lo contrario realiza las tareas con cuenta de un usuario genérico, no como root.

a) Crea un archivo *lista.txt* con la lista nombres de archivos terminados en *conf* junto a su número de inodo, del directorio /etc, y ordenados por la columna de inodo.

b) Busca los archivos del directorio /usr/sbin que posean solo 2 enlaces duros.

c) Crea dentro de tu directorio la estructura a/b y mueve el archivo lista.txt al subdirectorio b

d) Borra a y todo lo que contiene con una sola instrucción.

e) Muestra solo los nombres de los archivos del directorio /usr/include que no contengan la palabra *include*.

f) Entra en otra consola como root y reinicia la máquina en 2 minutos.

g) Lanza el comando yes > /dev/null, y a continuación páralo.

h) Tráelo a primer plano y cancélalo.

i) Desde el directorio de trabajo de *usuario* crea la siguiente estructura de directorios: BarrioSesamo/Supercoco/ArribaYAbajo

j) Localiza el archivo *gpm.h* evitando mensajes de error por pantalla.

k) Cópialo en el directorio Supercoco

l) Cambia los permisos de la copia para que solo tu puedas modificarlo.

m) Borra el original.

n) Muestra los archivos que contienen la palabra vfat en el directorio */etc*. Solo se quiere que aparezca el nombre del archivo donde aparece y no la línea que lo contiene. Evitar la salida de mensajes de error del estilo a *permiso denegado*.

o) Entra como root y muestra los archivos vacíos de su directorio de trabajo.

p) Muestra la línea del archivo /etc/passwd en la que aparece tu usuario. Controla que solo aparece esa línea.