Órdenes básicas

Órdenes Linux			
man/info/help		rm	more
touch	file	mkdir/rmdir	head/tail
ls	cp	cd	sort
pwd	mv	cat	clear

Órdenes	Descripción
ls [directorio]	Lista los contenidos de un directorio
cd [directorio]	Cambia de directorio de trabajo. Las abreviaciones . y .. se pueden utilizar como referencia de los directorios actual y padre, respectivamente. El símbolo ~ (pulsando a la vez las teclas AltGr y 4) es el directorio HOME del usuario y el símbolo / al inicio de un camino es el directorio raíz del sistema.
pwd	Imprime el camino absoluto del directorio actual
mkdir directorio	Crea un directorio a partir del nombre dado como argumento
rmdir directorio	Borra un directorio existente (si está vacío)
cat [archivo(s)]	Orden multipropósito: muestra el contenido de un archivo o varios, concatena archivos, copia un archivo, crea un archivo de texto o muestra los caracteres invisibles de control.
cp archivo1 archivo2	Copia el archivo1 en el archivo2. Si archivo2 no existe, se crea.
mv fuente destino	Renombra archivos (el archivo fuente puede ser archivo o directorio, al igual que el destino) y puede mover de lugar un archivo o directorio
file archivo(s)	Muestra el tipo de archivo dado como argumento
more archivo(s)	Visualiza un archivo fraccionándolo una pantalla cada vez (existen otros paginadores como page, pg, etc.). Antes de usar esta orden es conveniente usar la orden file para comprobar que se trata de un archivo ascii.
rm directorio_archivos	Borra archivos y directorios con contenido
touch archivo(s)	Si existen los archivos dados como argumentos se modifican su fecha y hora. En caso contrario, se crean con la fecha actual del sistema.
clear	Borra el contenido del terminal actual
tail [archivo(s)]	Muestra la parte final del contenido de un archivo dado como argumento. Por defecto muestra 10 líneas.
head [archivo(s)]	Muestra la parte inicial del contenido de un archivo dado como argumento. Por defecto muestra 10 líneas.
sort [archivo(s)]	Ordena, según un criterio elegido, el contenido de los archivos dados como argumentos

La orden `ls` (list sources) muestra los archivos contenidos en el directorio que se especifica (si no se especifica nada, tomará como directorio el directorio actual).

```
ls [-option] … [FILE]...
```

Algunas de las opciones más corrientes de esta orden son:

`-a` lista los archivos del directorio actual, incluidos aquellos cuyo nombre comienza con un punto, ".".
`-C` lista en formato multicolumna.
`-l` formato largo (ver descripción a continuación).
`-r` lista en orden inverso.
`-R` lista subdirectorios recursivamente, además del directorio actual.
`-t` lista de acuerdo con la fecha de modificación de los archivos.

Permiso	Archivos	Directorios
r	Lectura (read)	Se puede listar su contenido
w	Escritura (write)	Podemos modificarlo
x	Ejecución (execute)	Podemos acceder a él
-	No hay permiso	

rwx	rwx	rwx
Propietario del archivo (user)	Grupo de usuarios (group)	Resto de usuarios (others)

Metacaracteres

Metacarácter	Descripción de la función
?	Representa cualquier carácter simple en la posición en la que se indique
*	Representa cualquier secuencia de cero o más caracteres
[ ]	Designan un carácter o rango de caracteres que representan un carácter simple a través de una lista de caracteres o mediante un rango, en cuyo caso, mostramos el primer y último carácter del rango separados por un guión "-".
{ }	Sustituyen conjuntos de palabras separadas por comas que comparten partes comunes.
~	Se usa para abreviar el camino absoluto (path) del directorio HOME.

Pract. 2

Órdenes Linux			
chmod	wc	echo	date

Modificación de permisos

En el modo simbólico, se debe indicar primero a qué grupo de usuarios se va a aplicar el cambio con una letra minúscula:

- `u` : propietario
- `g` : grupo
- `o` : resto de usuarios
- `a` : todos los grupos de usuarios

Después, se debe indicar si va a permitir el acceso (+) o se va a denegar el acceso (-). Por último, se indica qué tipo de permiso es el que estamos modificando (r , w , x) y el archivo al que se le van a modificar los permisos. A continuación, se muestran algunos ejemplos de utilización de la orden **`chmod`**:

```
$ chmod ug+rw ej1
-rw-rw-r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
$ chmod u+x,g-w ej2
-rw-rw-r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rwxr--r--  1 quasimodo alumnos    3410 May 18 2010 ej2
```

Metacaracteres de redirección

Metacarácter	Descripción
< nombre	Redirecciona la entrada de una orden para que la obtenga del archivo *nombre*.
> nombre	Redirige la salida de una orden para que la escriba en el archivo *nombre*. Si dicho archivo ya existe, lo sobreescribe.
&> nombre	La salida estándar se combina con la salida de error estándar y ambas se escriben en el archivo *nombre*.
>> nombre	Funciona igual que el metacarácter "**>**" pero añade la salida estándar al final del contenido del archivo *nombre*.
&>> nombre	Igual que el metacarácter "**&>**", pero añadiendo las dos salidas combinadas al final del archivo *nombre*.
2> nombre	Redirige la salida de error estándar a un archivo (sólo funciona en shells de "bash").
|	Crea un cauce entre dos órdenes. La salida de una de ellas se utiliza como entrada de la otra.
| &	Crea un cauce entre dos órdenes utilizando las dos salidas (estándar y error) de una de ellas como entrada de la otra.

Metacaracteres sintácticos

Metacarácter	Descripción
;	Separador entre órdenes que se ejecutan secuencialmente.
( )	Se usan para aislar órdenes separadas por ";" o por "|". Las órdenes dentro de los paréntesis son tratadas como una única orden.
&&	Separador entre órdenes, en la que la orden que sigue al metacarácter "&&" se ejecuta sólo si la orden precedente ha tenido éxito (no ha habido errores).
||	Separador entre órdenes, en la que la orden que sigue al metacarácter "||" se ejecuta sólo si la orden precedente falla.

Pract. 3

Órdenes Linux			
set, unset	env, printenv	declare	expr
export	alias, unalias	find	grep, fgrep, egrep
printf			

El bash contempla dos tipos de variables. Las **variables de entorno** o variables globales son aquellas que son comunes a todos los shells. Para visualizarlas puede probar a usar las órdenes `env` o `printenv`. Por convención, se usan las letras en mayúscula para los nombres de dichas variables.

Ejercicio 3.1: Escriba, al menos, cinco variables de entorno junto con el valor que tienen.

Otro tipo de variables son las llamadas **variables locales**, éstas son sólo visibles en el shell donde se definen y se les da valor. Para ver las variables locales se puede usar la orden `set`.

Crear y visualizar variables

```
$ numero=1
$ echo $numero
1
```

Para crear variables de tipo vector utilizamos la misma forma de definición pero los elementos del vector se ponen entre paréntesis y separados por espacios. Un ejemplo de creación de un vector es:

```
$ colores=(rojo azul verde)
```

Para acceder a uno de sus elementos:

```
$ echo ${colores[0]}
rojo
$ echo ${colores[1]}
azul
```

Variables especiales

Nombre de variable	Descripción
$BASH	Contiene la ruta de acceso completa usada para ejecutar la instancia actual de bash.
$HOME	Almacena el directorio raíz del usuario; se puede emplear junto con la orden **cd** sin argumentos para ir al directorio raíz del usuario.
$PATH	Guarda el camino de búsqueda de las órdenes, este camino está formado por una lista de todos los directorios en los que queremos buscar una orden.
$?	Contiene el código de retorno de la última orden ejecutada, bien sea una instrucción o un guion.

Si deseamos crear una variable con ciertos atributos, utilizaremos la orden **declare**, cuya sintaxis completa se puede ver con **help declare**. Podemos indicar que una variable es numérica con la opción **-i** y ver los atributos con la opción **-p**:

```
$ declare -i IVA=18
$ declare -p IVA
declare -i IVA="18"
$ declare -i IVA=hola
$ declare -p IVA
declare -i IVA="0"
```

Exportar variables

export variable

```
$ NOMBRE=FS
$ echo $NOMBRE


$ bash
$ echo $NOMBRE
```

Comillas en las órdenes

En el shell bash se puede hacer uso de lo que se denomina *sustitución de órdenes*, que permite la ejecución de una orden, con o sin argumentos, de forma que su salida se trata como si fuese el valor de una variable. La sustitución de órdenes se puede hacer poniendo **$(***orden argumentos***)**, o usando los apóstrofos inversos, es decir `` `orden argumentos` ``, y puede utilizarse en cualquier tipo de expresión, en particular en las expresiones aritméticas que se verán en la práctica siguiente.

Con el ejemplo siguiente se plantean dos expresiones que son equivalentes:

```
$ echo "Los archivos que hay en el directorio son: $(ls -l)"
$ echo "Los archivos que hay en el directorio son: `ls -l`"
```

Ejercicio 3.3: Compruebe qué ocurre en las expresiones del ejemplo anterior si se quitan las comillas dobles del final y se ponen después de los dos puntos. ¿Qué sucede si se sustituyen las comillas dobles por comillas simples?

En los casos anteriores, las comillas dobles se utilizan como mecanismo de *acotación débil* (*weak quotation*), para proteger cadenas desactivando el significado de los caracteres especiales que haya entre ellas, salvo los caracteres `!`, `$`, `\` y `` ` ``, que no quedan protegidos. También se pueden proteger cadenas usando comillas simples como meca-

nismo de *acotación fuerte* (*strong quotation*), aunque en este caso se protegen los caracteres especiales salvo `!`; en consecuencia, las comillas simples serán útiles cuando se quiera proteger una variable o una orden.

Cuando se usan comillas simples dentro de una expresión, por ejemplo, si se deseara imprimir un mensaje como el siguiente: `'En el libro de inglés aparece Peter's cat'`, si se emplea la orden **`echo`**, nos aparecerá en la línea siguiente el carácter **>** que representa una línea de continuación de la orden dada, es decir, es como si no se hubiera completado el mensaje de texto. Para hacerlo correctamente, habría que escapar la comilla de la palabra Peter's con el metacarácter **** y partir la frase en dos partes acotadas con comillas simples como se muestra a continuación:

```
$ echo 'En el libro de inglés aparece Peter's cat'
>
$ echo 'En el libro de inglés aparece Peter'\''s cat'
En el libro de inglés aparece Peter's cat
```

Asignar órdenes a variables

```
variable=`orden`
```

Como vemos en el ejemplo anterior, todo se ha convertido en carácter, y no se ha realizado la operación matemática que deseábamos. La solución a este problema viene de la mano de la orden del sistema **`expr`**, con la que podemos evaluar la expresión que le sigue.

```
$ numero=1
$ echo $numero
1
$ numero=`expr $numero + 1`
```

Printf

```
printf formato [argumentos]
```

Secuencia de escape	Acción
`\b`	Espacio atrás
`\n`	Nueva línea
`\t`	Tabulador
`\'`	Carácter comilla simple
`\\`	Barra invertida
`\0n`	n = número en octal que representa un carácter ASCII de 8 bits

Código de formato	Representa
`%d`	Un número con signo
`%f`	Un número en coma flotante (decimal) sin notación exponencial
`%q`	Entrecomilla una cadena
`%s`	Muestra una cadena sin entrecomillar
`%x`	Muestra un número en hexadecimal
`%o`	Muestra un número en octal

```
$ printf "%10d\n" 25
        25
```

Podemos justificar a la izquierda, si utilizamos un número negativo:

```
$ printf "%-10d %-10d\n" 11 12
11        12
```

Si el número que especificamos en el formato es un decimal, la parte entera se interpreta como la anchura de la columna y el decimal como el número mínimo de dígitos:

```
$ printf "%10.3f\n" 15,4
    15,400
```

Podemos convertir un número de octal o hexadecimal a decimal:

```
$ printf "%d %d\n" 010 0xF
8 15
```

Y a la inversa, de decimal a octal/hexadecimal:

```
$ printf "0%o 0x%x\n" 8 15
00 0xf
```

También podemos usar variables como argumentos de la orden **printf**. Por ejemplo, si queremos mostrar la variable IVA, antes declarada, junto con un mensaje explicativo, escribiríamos:

```
$ printf "El valor actual del IVA es del %d\n" $IVA
El valor actual del IVA es del 18
```

Alias

Los alias se crean con la orden empotrada **alias** y se borran o eliminan con la orden **unalias**. Puedes usar la orden **help** para conocer cuál es la sintaxis de estas dos órdenes.

```
$ alias dir='ls -l'
$ dir
```

Orden find

Se utiliza para buscar por la estructura de directorios los archivos que satisfagan los criterios especificados. Su formato es el siguiente:

find lista-de-directorios [expresiones]

1. Por el nombre del archivo: se utiliza la opción **-name** seguida por el nombre deseado. Este nombre puede incluir la expansión de metacaracteres de archivo debidamente acotados. Por ejemplo:

```
$ find / -name "*.c"
```

2. Por el último acceso: se utiliza la opción **-atime** seguida por un número de días o por el número y un signo + o - delante de él. Por ejemplo:

-atime 7 busca los archivos a los que se accedió hace 7 días.

-atime -2 busca los archivos a los que se accedió hace menos de 2 días.

-atime +5 busca los archivos a los que se accedió hace más de 5 días.

3. Por ser de un determinado tipo: se utiliza la opción **-type** seguida de un carácter que indique el tipo de archivo. Se usa la opción **f** para referirse a archivos regulares y la opción **d** para directorios. En el ejemplo siguiente se muestra la búsqueda de los archivos del directorio actual que sean archivos regulares:

```
$ find . -type f
```

4. Por su tamaño en bloques: se utiliza la opción **-size** seguida de un número con o sin signo (+ o -). Si el número va seguido de la letra **c** el tamaño dado es en bytes. Por ejemplo: **-size 100** busca los archivos cuyo tamaño es de 100 bloques.

Además, se puede negar cualquier operador de selección o de acción utilizando el operador **!** que debe ir entre espacios en blanco y antes del operador a negar. Por ejemplo, para buscar los archivos del directorio raíz que no pertenezcan al usuario llamado pat:

```
$ find / ! -user pat
```

También se puede especificar un operador u otro utilizando el operador **-o**. Este operador conecta dos expresiones y se seleccionarán aquellos archivos que cumplan una de las dos expresiones (y no las dos, como hasta ahora). En

```
$ find . -size 10 -o -atime +2
```

-print: visualiza los nombres de camino de cada archivo que se adapta al criterio de búsqueda. Es la opción por defecto. Por ejemplo, para visualizar los nombres de todos los archivos y directorios del directorio actual:

```
$ find . -print
```

-exec: permite añadir una orden que se aplicará a los archivos localizados. La orden se situará a continuación de la opción y debe terminarse con un espacio, un carácter **** y a continuación un **;**. Se utiliza **{}** para representar el nombre de archivos localizados. Por ejemplo:

```
$ find . -atime +100 -exec rm {} \;
```

Eliminará todos los archivos del directorio actual (y sus descendientes) que no han sido utilizados en los últimos 100 días.

-ok: es similar a **-exec**, con la excepción de que solicita confirmación en cada archivo localizado antes de ejecutar la orden.

Orden grep

La orden **grep** permite buscar cadenas en archivos utilizando patrones para especificar dicha cadena. Esta orden lee de la entrada estándar o de una lista de archivos especificados como argumentos y escribe en la salida estándar aquellas líneas que contengan la cadena. Su formato es:

grep opciones patrón archivos

Algunas opciones que se pueden utilizar con la orden **grep** son:

- **-x** localiza líneas que coincidan totalmente, desde el principio hasta el final de línea, con el patrón especificado.
- **-v** selecciona todas las líneas que no contengan el patrón especificado.
- **-c** produce solamente un recuento de las líneas coincidentes.
- **-i** ignora las distinciones entre mayúsculas y minúsculas.
- **-n** añade el número de línea en el archivo fuente a la salida de las coincidencias.
- **-l** selecciona sólo los nombres de aquellos archivos que coincidan con el patrón de búsqueda.
- **-e** especial para el uso de múltiples patrones e incluso si el patrón comienza por el carácter (-).

Existen dos variantes de **grep** que optimizan su funcionamiento en casos especiales. La orden **fgrep** acepta sólo una cadena simple de búsqueda en vez de una expresión regular. La orden **egrep** permite un conjunto más complejo de operadores en expresiones regulares. Usando **man** comprueba las diferencias entre estas tres órdenes.

Guiones

Un *guion* del shell (script o programa shell) es un archivo de texto que contiene órdenes del shell y del sistema operativo. Este archivo es utilizado por el shell como guía para saber qué órdenes ejecutar.

Siguiendo la similitud de ejemplos de lenguajes de programación de alto nivel, comencemos con el típico ejemplo `"Hola Mundo"` para mostrar dicho mensaje mediante un guion o script bash. Para ello, abriremos un editor de textos ascii (vi, kedit, gedit, emacs, xemacs, ...) y escribimos lo siguiente:

```
echo "Hola Mundo"
```

Mediante el editor guardamos este documento con el nombre `holamundo`; para mostrar el resultado de su ejecución nos vamos al terminal y escribimos lo siguiente:

```
$ bash holamundo
```

Para ello, debemos poner siempre en la primera línea del archivo los símbolos **`#!`** seguidos del nombre del programa del tipo de shell que deseamos ejecutar, para nuestro caso, `/bin/bash`. Con esto, nuestro ejemplo anterior quedaría:

```
#!/bin/bash
printf "El directorio $HOME contiene los siguientes archivos:\n"
ls $HOME
```

Ahora, podemos hacer nuestro archivo ejecutable (`chmod +x prueba`) con lo que la ejecución sería:

```
$ ./prueba
```

Nombre	Descripción
`$0`	Nombre del guion o script que se ha llamado. Sólo se emplea dentro del guion.
`$1 .. $9` `${n}, n>9`	Son los distintos argumentos que se pueden facilitar al llamar a un guion. Los nueve primeros se referencian con **`$1,$2,...,$9`**, y a partir de ahí es necesario encerrar el número entre llaves, es decir, **`${n}`**, para **`n>9`**.
`$*`	Contiene todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a `"$1 $2 $3 … $n"`.
`$@`	Contiene todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a `"$1" "$2" … "$n"`.
`$#`	Contiene el número de argumentos que se han pasado al llamar al guion.
`${arg:-val}`	Si el argumento tiene valor y es no nulo, continua con su valor, en caso contrario se le asigna el valor indicado por *val*.
`${arg:?val}`	Si el argumento tiene valor y es no nulo, sustituye a su valor; en caso contrario, imprime el valor de *val* y sale del guion. Si *val* es omitida, imprime un mensaje indicando que el argumento es nulo o no está asignado.

Pract. 4

Órdenes Shell Bash					
`$(( … ))`	`$[ … ]`	`bc`	`let`	`test`	`if/else`

Expresiones con variables

El shell bash ofrece dos posibles sintaxis para manejar expresiones aritméticas haciendo uso de lo que se denomina *expansión aritmética*, o *sustitución aritmética*, que evalúa una expresión aritmética y sustituye el resultado de la expresión en el lugar donde se utiliza. Ambas posibilidades son:

```
$(( … ))
$[ … ]
```

Por ejemplo, la orden **`date`**, que permite consultar o establecer la fecha y la hora del sistema, y que admite como argumento **`+%j`** para conocer el número del día actual del año en curso, puede utilizarse para saber cuántas semanas faltan para el fin de año:

Operadores aritméticos

Operador	Descripción
`+` `-`	Suma y resta, o más unario y menos unario.
`*` `/` `%`	Multiplicación, división (truncando decimales), y resto de la división.
`**`	Potencia.
`++`	Incremento en una unidad. Puede ir como prefijo o como sufijo de una variable: si se usa como prefijo (`++variable`), primero se incrementa la variable y luego se hace lo que se desee con ella; si se utiliza como sufijo (`variable++`), primero se hace lo que se desee con la variable y después se incrementa su valor.
`--`	Decremento en una unidad. Actúa de forma análoga al caso anterior, pudiendo usarse como prefijo o como sufijo de una variable (`--variable` o `variable--`).
`( )`	Agrupación para evaluar conjuntamente; permite indicar el orden en el que se evaluarán las subexpresiones o partes de una expresión.
`,`	Separador entre expresiones con evaluación secuencial.
`=`	`x=expresión`, asigna a `x` el resultado de evaluar la `expresión` (no puede haber huecos en blanco a los lados del símbolo "=");
`+=` `-=`	`x+=y` equivale a `x=x+y;` `x-=y` equivale a `x=x-y;`
`*=` `/=`	`x*=y` equivale a `x=x*y;` `x/=y` equivale a `x=x/y;`
`%=`	`x%=y` equivale a `x=x%y.`

Asignación y variables aritméticas

Otra forma de asignar valor a una variable entera es utilizar la orden **`let`** de la shell bash. Aunque esta orden se usa para evaluar expresiones aritméticas, en su forma más habitual su sintaxis es:

```
let variableEntera=expresión
```

Como en otras asignaciones, a ambos lados del signo igual (=) no debe haber espacios en blanco y `expresión` debe ser una expresión aritmética.

```
$ let w=3+2
$ let w='3 + 2'
$ let w='(4+5)*6'
$ let "w=4+5*6"
$ let w=4+5*6
$ y=7
$ let w=y%5                    (esta orden es equivalente a: let w=$y%5)
```

Como habrá observado en el ejemplo anterior, las dos primeras asignaciones producen el mismo resultado, a pesar de que en la segunda hay espacios en blanco. Por el contrario, las asignaciones tercera y cuarta no dan el mismo resultado debido al uso o no de paréntesis. Las asignaciones cuarta y quinta son equivalentes, y las dos últimas ponen de manifiesto que en la expresión pueden intervenir otras variables.

Hemos de indicar que `(( <expresión> ))` equivale a la orden `let` y presenta ventajas como por ejemplo a la hora de hacer comparaciones numéricas para usarlas en ejecuciones condicionales:

```
$ a=10
$ ((a<10))
$ echo $?
1
$ ((a==10))
$ echo $?
0
$ if let 'a<10'; then echo "es menor"; else echo "es mayor o igual"; fi
es mayor o igual
```

Operadores relacionales

Operador	Descripción: el resultado se evalúa como "verdadero" -*true*- si ... (en otro caso sería "falso" -*false*-)
`A = B` | `A == B` | `A -eq B`	`A` es igual a `B`.
`A != B` | `A -ne B`	`A` es distinta de `B`.
`A < B` | `A -lt B`	`A` es menor que `B`.
`A > B` | `A -gt B`	`A` es mayor que `B`.
`A <= B` | `A -le B`	`A` es menor o igual que `B`.
`A >= B` | `A -ge B`	`A` es mayor o igual que `B`.
`! A`	`A` es falsa; representa al operador NOT (negación lógica).
`A && B`	`A` es verdadera y `B` es verdadera; es el operador AND (conjunción lógica).
`A || B`	`A` es verdadera o `B` es verdadera; es el operador OR (disyunción lógica).

A veces es necesario poder relacionar dos expresiones aritméticas, `A` y `B`, o negar una expresión aritmética, de forma que se pueda evaluar si se da o no cierta relación. La evaluación de una relación entre expresiones tomará finalmente un valor numérico, de manera que el 1 representa una evaluación "verdadera" (*true*), mientras que el 0 indica que la evaluación ha sido "falsa" (*false*). Observe que esta forma de evaluar resulta un poco discordante respecto a lo que sucede cuando se evalúa la variable `$?` que se mencionaba en la práctica anterior.

Operadores de consulta de archivos

Para aplicar los operadores de consulta de archivos haremos uso de dos órdenes nuevas, **`test`** e **`if`**, aunque la segunda de estas órdenes la trataremos en un apartado posterior.

La sintaxis de la orden **`test`** es:

`test` *expresión*

Operador	Descripción: el resultado se evalúa como "verdadero" -*true*- si ... (en otro caso sería "falso" -*false*-)
-b *archivo*	*archivo* existe y es un dispositivo de bloques.
-c *archivo*	*archivo* existe y es un dispositivo de caracteres.
-d *archivo*	*archivo* existe y es un directorio.
-e *archivo*	*archivo* existe.
-f *archivo*	*archivo* existe y es un archivo plano o regular.
-G *archivo*	*archivo* existe y es propiedad del mismo grupo del usuario.
-h *archivo*	*archivo* existe y es un enlace simbólico.
-L *archivo*	*archivo* existe y es un enlace simbólico. Es igual que **-h**.
-O *archivo*	*archivo* existe y es propiedad del usuario.
-r *archivo*	*archivo* existe y el usuario tiene permiso de lectura sobre él.
-s *archivo*	*archivo* existe y es no vacío.
-w *archivo*	*archivo* existe y el usuario tiene permiso de escritura sobre él.
-x *archivo*	*archivo* existe y el usuario tiene permiso de ejecución sobre él, o es un directorio y el usuario tiene permiso de búsqueda en él.
archivo1 **-nt** *archivo2*	*archivo1* es más reciente que *archivo2*, según la fecha de modificación, o si *archivo1* existe y *archivo2* no.
archivo1 **-ot** *archivo2*	*archivo1* es más antiguo que *archivo2*, según la fecha de modificación, o si *archivo2* existe y *archivo1* no.
archivo1 **-ef** *archivo2*	*archivo1* es un enlace duro al *archivo2*, es decir, si ambos se refieren a los mismos números de dispositivo e *inode*.

La orden **test** *expresión* es equivalente a la orden **[** *expresión* **]**, donde los huecos en blanco entre los corchetes y *expresión* son necesarios.

If else

```
if condición;
then
      declaraciones
[elif condición;
      then declaraciones ]...
[else
      declaraciones ]
fi
```

Cuando se emplea el doble paréntesis, no es posible utilizar los operadores **-eq**, **-ne**, **-le**, **-lt**, **-ge** y **-gt**. En su lugar se utilizarán los operadores equivalentes mostrados en la tabla 4.3. Además, la última comparación obliga a usar el operador de igualdad ==. Si se emplea el = responderá con un mensaje de error.

(doble paréntesis ==, corchetes =)

A partir del directorio personal y utilizando una única orden, crea un directorio llamado **examen**, dentro de él otros dos directorios llamados **directorio1** y **directorio2**, y dentro de **directorio1** otro directorio llamado **archivos**.

Situados en el directorio **archivos**, con una única orden, crea tres archivos llamados **arch1.doc**, **arch2.doc** y **arch3.exe**. A continuación, copia aquellos cuya extensión tenga una d al directorio **directorio2**.

---

mkdir examen; cd examen; mkdir directorio1 directorio2; cd directorio1; mkdir archivos

touch arch1.doc arch2.doc arch3.exe

cp *.d* ../../directorio2

Supon que en el directorio actual se encuentran los siguientes ficheros:

guion_salario
guion_fs_p1
archivos.doc
guion_fs_p2
guion_fs_p3
modify.sh

Con una única orden, indica cómo se cambiarían los permisos de los archivos guion_fs_p1, guion_fs_p2, guion_fs_p3, de forma que se elimine el permiso de lectura al grupo y se les conceda permiso de ejecución a los tres tipos de usuarios.

---

chmod g-r,a+x guion_fs_p1 guion_fs_p2 guion_fs_p3

Redirecciona a un archivo llamado prueba.txt en tu directorio personal el resultado de buscar la palabra «matrices» en la ayuda (help) de la orden declare.

---

help declare>aux.txt

grep matrices aux.txt>prueba.txt

Escribe en una sola línea sin utilizar «;» una orden que permita listar los ficheros (así como sus permisos) que haya dentro de un directorio llamado lista y visualizar el contenido de un archivo llamado texto dentro del directorio lista; si cualquiera de esas órdenes fallara, se debería indicar con un mensaje «No se ha podido completar la orden».

(ls -l ../lista && head ../lista/texto) || echo "No se ha podido completar la oden"

Indica qué es necesario hacer para crear una variable RUTA que contenga la ruta absoluta del directorio actual. Indica el comando correspondiente para comprobar las variables locales de tu sistema.

RUTA=`pwd`

set

Tema 1

Ejercicios Sesion 2. Fundamentos del Software

Ejercicio 2.1. Cree el siguiente arbol de directorios a partir de un directorio de su cuenta de usuario. Indique tambien como seriá posible crear toda esa estructura de directorios mediante una unica orden (mire las opciones de la orden de creacion de directorios mediante man mkdir). Posteriormente realice las siguientes acciones:

a) En Ejer1 cree los archivos arch100.txt, filetags.txt, practFS.ext y robet201.me.

- $ touch arch100.txt
- $ touch filetags.txt
- $ touch practFS.ext
- $ touch robet201.me

b) En Ejer21 cree los archivos robet202.me, ejer11sol.txt y blue.me.

- $ touch robet202.me
- $ touch ejer11sol.txt
- $ touch blue.me

c) En Ejer2 cree los archivos ejer2arch.txt, ejer2filetags.txt y readme2.pdf.

- $ touch ejer2arch.txt
- $ touch ejer2filetags.txt
- $ touch readme2.pdf

d) En Ejer3 cree los archivos ejer3arch.txt, ejer3filetags.txt y readme3.pdf.

- $ touch ejer3arch.txt
- $ touch ejer3filetags.txt
- $ touch readme3.pdf

e) ¿Podrián realizarse las acciones anteriores empleando una unica orden? Indique como.

Se haria con la orden touch seguido de los nombre de los archivos a crear junto con su extension separado por espacios. Por ejemplo:

- $ touch robet22.me ejer11sol.txt blue.me

Ejercicio 2.2. Situados en el directorio ejercicio1 creado anteriormente, realice las siguientes acciones:

Para hacer este ejercicio nos situamos siempre en el directorio ejercicio1

a) Mueva el directorio Ejer21 al directorio Ejer2.

- $ mv Ejer1/Ejer21 Ejer2

b) Copie los archivos de Ejer1 cuya extension tenga una x al directorio Ejer3.

- $ cp Ejer1/*.*x* Ejer3

c) Si estamos situado en el directorio Ejer2 y ejecutamos la orden ls -la ../Ejer3/*arch*, ¿que´ archivo/s, en su caso, deberiá mostrar?

- arch100.txt
- ejer3arch.txt

Ejercicio 2.3. Si estamos situados en el directorio Ejer2, indique la orden necesaria para listar soío los nombres de todos los archivos del directorio padre.

$ ls ../*.*

Ejercicio 2.4. Liste los archivos que esteń en su directorio actual y fiǰese en alguno que no disponga de la fecha y hora actualizada, es decir, la hora actual y el diá de hoy. Ejecute la orden touch sobre dicho archivo y observe que´ sucede sobre la fecha del citado archivo cuando se vuelva a listar.

- Al hacer touch sobre un archivo ya existente se modifica su fecha y hora a la fecha y hora actual

Ejercicio 2.5. La organizacioń del espacio en directorios es fundamental para poder localizar faćilmente aquello que estemos buscando. En ese sentido, realice las siguientes acciones dentro de su directorio home (es el directorio por defecto sobre el que trabajamos al entrar en el sistema):

a) Obtenga en nombre de camino absoluto (pathname absoluto) de su directorio home. ¿Es el mismo que el de su compañ̃ero/a?

- Nos es el mismo que el de mi compañero, en mi caso es: /home/javia

b) Cree un directorio para cada asignatura en la que se van a realizar praćticas de laboratorio y, dentro de cada directorio, nuevos directorios para cada una de las praćticas realizadas hasta el momento.

- Para crear esto directorios lo haremos con la orden mkdir seguido del nombre de los directorios.

c) Dentro del directorio de la asignatura fundamentos del software (llamado FS) y dentro del directorio creado para esta praćtica, copie los archivos hosts y passwd que se encuentran dentro del directorio /etc.

- $ cp -a /etc/hosts -a /etc/passwd ~/Escuela/FS/Practicas

d) Muestre el contenido de cada uno de los archivos.

- Para mostrar el contenido de cada uno de los archivos utilizamos la orden cat, por ejemplo: $ cat hosts o $ cat passwd

Ejercicio 2.6. Situados en alguń lugar de su directorio principal de usuario (directorio HOME), cree los directorios siguientes: Sesion.1, Sesion.10, Sesion.2, Sesion.3, Sesion.4, Sesion.27, Prueba.1 y Sintaxis.2 y realice las siguientes tareas:

a) Borre el directorio Sesion.4

- $ rm -d Sesion.4 , ya que el directorio esta vacio, utilizamos la funcion -d

b) Liste todos aquellos directorios que empiecen por Sesion. y a continuación tenga un único carácter

- $ ls Sesion.?

c) Liste aquellos directorios cuyos nombres terminen en .1

- $ ls *.1

d) Liste aquellos directorios cuyos nombres terminen en .1 o .2

- $ ls -d *.[12]

e) Liste aquellos directorios cuyos nombres contengan los caracteres si

- Tras probar que el comando $ ls -d *{si}* no funciona, necesitamos 2 o mas partones dentro de {} separado por comas para que funcione, utilizo el comando:
- $ ls *si*

f) Liste aquellos directorios cuyos nombres contengan los caracteres si y terminen en .2

- $ ls -d *si*.2

Ejercicio 2.7. Desplacémonos hasta el directorio /bin, genere los siguientes listados de archivos (siempre de la forma más compacta y utilizando los metacaracteres apropiados):

a) Todos los archivos que contengan sólo cuatro caracteres en su nombre.

- ls -a ????

b) Todos los archivos que comiencen por los caracteres d, f.

- $ ls -a d* f*

c) Todos los archivos que comiencen por las parejas de caracteres sa, se, ad.

- $ ls -a sa* se* ad*

d) Todos los archivos que comiencen por t y acaben en r.

- $ ls -a t*r

Ejercicio 2.8. Liste todos los archivos que comiencen por tem y terminen por .gz o .zip :

a) De tu directorio HOME.

- $ ls tem*.gz tem*.zip

b) Del directorio actual.

c) ¿Hay alguna diferencia en el resultado de su ejecución? Razone la respuesta.

Ejercicio 2.9. Muestre del contenido de un archivo regular que contenga texto:

a) Las 10 primeras lińeas.

- $ head archivo.txt

b) Las 5 uĺtimas lińeas.

- tail -5 archivo.txt

Ejercicio 2.10. Cree un archivo empleando para ello cualquier editor de textos y escriba en el mismo varias palabras en diferentes lińeas. A continuacioń trate de mostrar su contenido de manera ordenada empleando diversos criterios de ordenacioń.

- $ sort hola.txt Ordena las lineas del documento situando en primer lugar las que empiezan por numeros y despues ordenandolas por orden alfabetico segun empiece cada linea

- $ sort -r hola.txt Ordena las lineas del documento totalmente al contraria que el caso anterior

Ejercicio 2.11. ¿Comó podriá ver el contenido de todos los archivos del directorio actual que terminen en .txt o .c?

- $ cat *.txt *.c

Tema 2

Ejercicios Sesion 3. Fundamentos del Software

Ejercicio 3.1. Se debe utilizar solamente una vez la orden chmod en cada apartado. Los cambios se harań en un archivo concreto del directorio de trabajo (salvo que se indique otra cosa). Cambiaremos uno o varios permisos en cada apartado (independientemente de que el archivo ya tenga o no dichos permisos) y comprobaremos que funciona correctamente:

□ □ □ □ □

- Dar permiso de ejecucioń al “resto de usuarios”.

$ chmod o+x archivo

- Dar permiso de escritura y ejecucioń al “grupo”.

$ chmod g+wx archivo

- Quitar el permiso de lectura al “grupo” y al “resto de usuarios”.

$ chmod go-r archivo

- Dar permiso de ejecucioń al “propietario” y permiso de escritura el “resto de usuarios”.

$ chmod u+x, o+w archivo

- Dar permiso de ejecucioń al “grupo” de todos los archivos cuyo nombre comience con la letra “e”. Nota: Si no hay mań de dos

archivos que cumplan esa condición, se deberán crear archivos que empiecen con “e” y/o modificar el nombre de archivos ya existentes para que cumplan esa condición.

```
$ chmod g+x e*
```

Ejercicio 3.2. Utilizando solamente las órdenes de la práctica anterior y los metacaracteres de redirección de salida:

- Cree un archivo llamado ej31 , que contendrá el nombre de los archivos del directorio padre del directorio de trabajo.

```
$ ls .. > ej31
```

- Cree un archivo llamado ej32 , que contendrá las dos últimas líneas del archivo creado en el ejercicio anterior.

```
$ tail -2 ej31 > ej32
```

- Añada al final del archivo ej32 , el contenido del archivo ej31 .

```
$ cat ej32 >> ej31
```

Ejercicio 3.3. Utilizando el metacarácter de creación de cauces y sin utilizar la orden cd:

- Muestre por pantalla el listado (en formato largo) de los últimos 6 archivos del directorio /etc.

```
$ ls -l /etc | tail -6
```

- La orden wc muestra por pantalla el número de líneas, palabras y bytes de un archivo (consulta la orden man para conocer más sobre ella). Utilizando dicha orden, muestre por pantalla el número de caracteres (sólo ese número) de los archivos del directorio de trabajo que comiencen por los caracteres “e” o “f”.

```
$ cat e* f* | wc -m
```

Ejercicio 3.4. Resuelva cada uno de los siguientes apartados.

a) Cree un archivo llamado ejercicio1, que contenga las 17 últimas líneas del texto que proporciona la orden man para la orden chmod (se debe hacer en una única línea de órdenes y sin utilizar el metacarácter “;”).

```
$ man chmod | tail -17 > ejercicio1
```

b) Al final del archivo ejercicio1, añada la ruta completa del directorio de trabajo actual.

```
$ pwd >> ejercicio1
```

c)Usando la combinación de órdenes mediante paréntesis, cree un archivo llamado ejercicio3 que contendrá el listado de usuarios conectados al sistema (orden who) y la lista de archivos del directorio actual.

```
$ (who ; pwd) > ejercicio3
```

d)Añada, al final del archivo ejercicio3, el número de líneas, palabras y caracteres del archivo ejercicio1. Asegúrese de que, por ejemplo, si no existiera ejercicio1, los mensajes de error también se añadieran al final de ejercicio3.

& wc ejercicio1 >> ejercicio3 2>> ejercicio3

e)Con una sola orden chmod, cambie los permisos de los archivos ejercicio1 y ejercicio3, de forma que se quite el permiso de lectura al "grupo" y se dé permiso de ejecución a las tres categorías de usuarios.

$ chmod g-r,a+x ejercicio1 ejercicio3

Tema 3

###Fundamentos del Software

##Relacion de ejercicios 6. Programacion del Shell

#####Ejercicio 6.1.
#####Escriba un guion que acepte dos argumentos. El primero será el nombre de un directorio y el segundo será un valor entero. El funcionamiento del guion será el siguiente: deberán anotarse en un archivo denominado archivosSizN.txt aquellos archivos del directorio dado como argumento y que cumplan la condición de tener un tamaño menor al valor aportado en el segundo argumento. Se deben tener en cuenta las comprobaciones sobre los argumentos, es decir, debe haber dos argumentos, el primero deberá ser un directorio existente y el segundo un valor entero.

```
1  #! /bin/bash
2  if[ $# == 2 ]; then
3        if[ -d $1 ] && [ $2 -eq $2 ];then
4              find $1 -size -$2 > archivosSizN.txt
5        else
6              echo "Los argumentos son invalidos"
7        fi
8  else
9        echo "El numero de argumentos no es valido"
10 fi
```

#####Ejercicio 6.2.
#####Escriba un guion que acepte el nombre de un directorio como argumento y muestre como resultado el nombre de todos y cada uno de los archivos del mismo y una leyenda que diga "Directorio", "Enlace" o "Archivo regular", según corresponda. Incluya la comprobación necesaria sobre el argumento, es decir, determine si el nombre aportado se trata de un directorio existente.

```
#! /bin/bash
if [ -d $1 ]; then
      for archivo in `ls $1`
      do
            if [ -d $1$archivo ];then
                  printf "Directorio: %\n" $archivo
            elif [ -L $1$archivo ];then
                  printf "Enlace: %/n" $archivo
            else
                  printf "Archivo: %\n" $archivo
            fi
      done
else
      printf "Argumento no valido "
fi
```

#####Ejercicio 6.3.
#####Escriba un guion en el que, a partir de la pulsación de una tecla, detecte la zona del teclado donde se encuentre. Las zonas vendrán determinadas por las filas. La fila de los números 1, 2, 3, 4, ... será la fila 1, las teclas donde se encuentra la Q, W, E, R, T, Y,... serán de la fila 2, las teclas de la A, S, D, F,...serán de la fila 3 y las teclas de laZ,X,C,V,...serán de la fila 4. La captura de la tecla se realizará mediante la orden read.

#####Ejercicio 6.4.
#####Escriba un guion que acepte como argumento un parámetro en el que el usuario indica el mes que quiere ver, ya sea en formato numérico o usando las tres primeras letras del nombre del mes, y muestre el nombre completo del mes introducido. Si el número no está comprendido entre 1 y 12 o las letras no son significativas del nombre de un mes, el guion deberá mostrar el correspondiente mensaje de error.

#####Ejercicio 6.5.
#####Escriba un guion que solicite un número hasta que su valor esté comprendido entre 1 y 10. Deberá usar la orden while y, para la captura del número, la orden read.

```
#! /bin/bash
while true;
do
      printf "Introduzca un numero: \n"
      read numero
      if [ $numero -gt 0 ] && [ $numero -lt 10 ]; then
            printf "Numero correcto \n"
            break
      fi
done
```

#####Ejercicio 6.6.
#####Copie este ejercicio y pruébelo en su sistema para ver su funcionamiento. ¿Qué podemos modificar para que el giro se vea más rápido o más lento? ¿Qué hace la opción -e de las órdenes echo del guion?
#####Ejercicio 6.7.
#####Escriba un guion que admita como argumento el nombre de un tipo de shell (por ejemplo, csh, sh, bash, tcsh, etc.) y nos dé un listado ordenado alfabéticamente de los usuarios que tienen dicho tipo de shell por defecto cuando abren un terminal. Dicha información del tipo de shell asignado a un usuario se puede encontrar en el archivo /etc/passwd, cuyo contenido está delimitado por ':'. Cada información situada entre esos delimitadores representa un campo y precisamente el campo que nos interesa se encuentra situado en primer lugar. En

definitiva, para quedarnos con lo que aparece justo antes del primer delimitador sera´ u´til la orden siguiente:

cut -d':' -f1 /etc/passwd

#####Donde la opcioń –d indica cuaí es el delimitador utilizado y la opcioń –f1 representa a la secuencia de caracteres del primer campo.
Realice, utilizando el mecanismo de cauces, el ejercicio pero usando la orden cat para mostrar el contenido de un archivo y encauzado con la orden cut para filtrar la informacioń que aparece justo antes del delimitador ':'4.
#####Realice tambień la comprobacioń de la validez del tipo de Shell que se introduce como argumento. Use para ello la informacioń que encontrara´ en el archivo /etc/shells donde encontrara´ los tipos de Shell que se pueden utilizar en el sistema.

#####Ejercicio 6.8.
#####Dos oŕdenes frecuentes de Unix son tar y gzip. La orden tar permite almacenar/extraer varios archivos de otro archivo. Por ejemplo, podemos almacenar el contenido de un directorio en un archivo con

tar -cvf archivo.tar directorio (la opcioń -x extrae los archivos de un archivo .tar).

#####La orden gzip permite comprimir el contenido de un archivo para que ocupe menos espacio. Por ejemplo, gzip archivo comprime archivo y lo sustituye por otro con el mismo nombre y con la extensioń .gz. La orden para descomprimir un archivo .gz o .zip es gunzip.
#####Dadas estas oŕdenes construya un guion, denominado cpback, que dado un directorio o lista de archivos como argumento(s) los archive y comprima en un archivo con nombre copiaYYMMDD, donde YY corresponde al anõ, la MM al mes y la DD al diá, dentro de un directorio denominado CopiasSeguridad. El guion debe realizar las comprobaciones oportunas: los argumentos existen, el directorio de destino existe y si no, lo crea.

#####Ejercicio 6.9.
#####Hacer un script en Bash denominado newdirfiles con los siguientes tres argumentos:

<dirname> Nombre del directorio que, en caso de no existir, se debe crear para alojar en eí los archivos que se han de crear.

<num_files> Numéro de archivos que se han de crear.

<basefilename> Sera´ una cadena de caracteres que represente el nombre base de los archivos.

#####Ese guion debe realizar lo siguiente:

Comprobar que el numéro de argumentos es el correcto y que el segundo argumento tenga un valor comprendido entre 1 y 99.

Crear, en caso de no existir, el directorio dado en el primer argumento a partir del directorio donde se este´ situado y que posea permisos de lectura y escritura para el usuario $USER.

Dentro del directorio dado en el primer argumento, crear archivos cuyos contenidos estarań vacioś y cuyos nombres lo formarań el nombre dado como tercer argumento y un numéro que ira´ desde 01 hasta el numéro dado en el segundo argumento.

Tema 4

Práctica 5: expresiones con variable y expresiones regulares

Ejercicio 1

**Utilizando una variable que contenga el valor entero 365 y otra que guarde el número del día actual
del año en curso, realice la misma operación del ejemplo anterior usando cada una de las diversas formas de
cálculo comentadas hasta el momento, es decir, utilizando `expr`, `$(( ... ))` y `$[ ... ]`.**

Primero declaramos las dos variables mediante el comando
declare.

```console
$ declare -i dias_anio=365
$ declare -i dia_actual=$(date +%j)
```

Ahora realizamos la operacion de dos formas:

```console
$ echo "Faltan $(( (dias_anio-dia_actual) / 7)) semanas hasta el fin de año"
$ echo "Faltan $[(dias_anio-dia_actual)/7] semanas hasta el fin de año"
```

Nótese que son necesarios los dobles paréntesis en la expresión $((()), y que ninguno de los casos es necesario preceder el nombre de las variables con $.

Ejercicio 2

**Realice las siguientes operaciones para conocer el funcionamiento del operador de incremento
como sufijo y como prefijo. Razone el resultado obtenido en cada una de ellas:**

```console
$ v=1
$ echo $v
$ echo $((v++))
$ echo $v
$ echo $((++v))
$ echo $v
```

```console
$ v=1
```

La variable es declarada e inicializada a 1.

```console
$ echo $v
1
```

Se muestra el valor de la variable, que es
claramente 1.

```console
$ echo $((v++))
1
```

Como v++ es incremento en una unidad sufijo, primero
se muestra el valor de la variable y luego se incrementa
ésta en 1.

```console
$ echo $v
2
```

Como ante se ha incrementado en 1, ahora el valor de la
variable es 2.

```console
$ echo $((++v))
3
```

Como `v--` es incremento en una unidad prefijo, primero
se incrementa v en una unidad y luego se muestra el valor
de la variable. Como el valor de la variable anteriormente
era 2, aumenta a 3 que es lo que se muestra

```console
$ echo $v
3
```

El valor mostrado es 3, pues no se ha realizado ningún
cambio desde el último incremento.

Ejercicio 3

**Utilizando el operador de división, ponga un caso concreto donde se aprecie que la asignación
abreviada es equivalente a la asignación completa, es decir, que x/=y equivale a x=x/y.
En el resultado del cálculo de expresiones aritméticas, bash solamente trabaja con números enteros, por lo que si
se necesitase calcular un resultado con decimales, habría que utilizar una forma alternativa, como puede ser la
ofrecida por la orden `bc` , cuya opción `- l` , letra “ele”, permite hacer algunos cálculos matemáticos (admite otras
posibilidades que pueden verse mediante `man`).**

Utilizando la asignación completa

```console
$ x=10
$ y=2
$ echo $[x=x/y]
  5
$ echo $x
```

```
  5
```

Utilizando la asignación abreviada

```console
$ x=10
$ y=2
$ echo $[x/=y]
5
$ echo $x
5
```

Lo que muestra que son equivalentes.

Ejercicio 4

**Compruebe qué ocurre si en el ejemplo anterior utiliza comillas dobles o simples para acotar todo lo
que sigue a la orden `echo`. ¿Qué sucede si se acota entre comillas dobles solamente la expresión aritmética que se
quiere calcular?, ¿y si se usan comillas simples?**

```console
$ echo "6/5 | bc -l"
6/5 | bc -l

$ echo "6/5" | bc -l
1.20000000000000000000

$ echo '6/5 | bc -l'
6/5 | bc -l

$ echo '6/5' | bc -l
1.20000000000000000000
```

No hay diferencia entre usar comillas simples y dobles
en este caso. Cuando se acota todo lo que sigue a echo,
se muestra literalmente eso. Sin embargo, si sólo se acota
la expresión aritmetica, el resultado 6/5 es pasado mediante
el cauce a la orden `bc -l`.

Ejercicio 5

**Calcule con decimales el resultado de la expresión aritmética (3-2)/5. Escriba todas las
expresiones que haya probado hasta dar con una respuesta válida. Utilizando una solución válida, compruebe qué
sucede cuando la expresión aritmética se acota entre comillas dobles; ¿qué ocurre si se usan comillas simples?, ¿y
si se ponen apóstrofos inversos?**

Probando como en el ejercicio anterior:

```
```

```
$ echo (3-2)/5 | bc -l
bash: error sintáctico cerca del elemento inesperado `3-2'
```

Como vemos la expresión (3-2)/5 no se ha evaluado ya que no
iba acotada por $(()) o por $[].

```
$ echo $[(3-2)/5] | bc -l
0
```

En este caso la expresión sí se ha ejecutado, pero la divisón
entre 1 y 5 se realiza de forma entera, es decir resultado 0.
Por tanto, la orden bc -l no devuelve otra expresión decimal
que el propio 0.

```
$ echo "(3-2)/5" | bc -l
.20000000000000000000
```

En este caso, la solución sí es la correcta.

Probemos ahora qué sucede con comillas simples:

```
$ echo '(3-2)/5' | bc -l
.20000000000000000000
```

Es decir, las expresiones son equivalentes.

En el caso de apóstrofos inversos.

```
$ echo `(3-2)/5` \ bc -l
bash: command substitution: línea 1: error sintáctico cerca del
elemento inesperado `/5'
bash: command substitution: línea 1: `(3-2)/5'
```

Resulta error, ya que los apóstrofos inversos intentan
ejecutar la orden `(3-2)/5`, la cual hemos visto que era posible
evaluar debido a la falta de `$(())` o `$[]`.

Ejercicio 6

**Consulte la sintaxis completa de la orden `let` utilizando la orden
de ayuda para las órdenes
empotradas (`help let`) y tome nota de su sintaxis general.**

`let arg [arg ...]`

Evalúa cada ARG como una expresión aritmética. La evaluación se hace
con enteros de longitud fija, sin revisar desbordamientos, aunque la
la división por 0 se captura y se marca como un error. La siguiente
lista de operadores está agrupada en niveles de operadores de la misma
prioridad. Se muestran los niveles en orden de prioridad decreciente.

Ejercicio 7

**Al realizar el ejercicio anterior habrá observado que la orden `let` admite asignaciones múltiples y
operadores que nosotros no hemos mencionado anteriormente. Ponga un ejemplo de asignación múltiple y, por
otra parte, copie en un archivo el orden en el que se evalúan los operadores que admite.**

```console
$ let a=1,b=2
$ echo "a es igual a $a y b es igual a $b".
a es igual a 1 y b es igual a 2
```

El orden en el se que se evalúan los operadores:

```console
    id++, id-- post-incremento, post-decremento de variable
    ++id, --id pre-incremento, pre-decremento de variable
    -, +              menos, más unario
    !, ~              negación lógica y basada en bits
    **          exponenciación
    *, /, %           multiplicación, división, residuo
    +, -              adición, sustracción
    <<, >>            desplazamientos de bits izquierdo y derecho
    <=, >=, <, >      comparación
    ==, !=            equivalencia, inequivalencia
    &           AND de bits
    ^           XOR de bits
    |           OR de bits
    &&          AND lógico
    ||          OR lógico
    expr ? expr : expr
                operador condicional
    =, *=, /=, %=,
    +=, -=, <<=, >>=,
    &=, ^=, |= asignación
```

Ejercicio 8

**Haciendo uso de las órdenes conocidas hasta el momento, construya un guion que admita dos
parámetros, que compare por separado si el primer parámetro que se le pasa es igual al segundo, o es menor, o es
mayor, y que informe tanto del valor de cada uno de los parámetros como del resultado de cada una de las
evaluaciones mostrando un 0 o un 1 según corresponda.**

Script compara

```bash
#!/bin/bash
# Titulo:       compara
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
```

```
# Version:      1.0
# Descripción:  Compara los dos argumentos dados
# Opciones: ninguna
# Uso: compara <num1> <num2>

echo "El valor del primer parámetro es $1 y del segundo $2"

let mayor="$1 > $2"
let menor="$1 < $2"
let iguales="$1 == $2"

echo "¿$1 es mayor que $2?: $mayor"
echo "¿$1 es menor que $2?: $menor"
echo "¿$1 es igual que $2?: $iguales"
```

Procedamos a ejecutarlo

```console
$ chmod +x compara
$ ./compara 1 2

El valor del primer parámetro es 2 y del segundo 1
¿2 es mayor que 1?: 1
¿2 es menor que 1?: 0
¿2 es igual que 1?: 0
```

```console
$ ./compara 2 1

El valor del primer parámetro es 1 y del segundo 2
¿1 es mayor que 2?: 0
¿1 es menor que 2?: 1
¿1 es igual que 2?: 0

$ ./compara 1 1

El valor del primer parámetro es 1 y del segundo 1
¿1 es mayor que 1?: 0
¿1 es menor que 1?: 0
¿1 es igual que 1?: 1
```

Ejercicio 9

**Usando `test` , construya un guion que admita como parámetro un nombre de archivo y realice las
siguientes acciones: asignar a una variable el resultado de comprobar si el archivo dado como parámetro es plano
y tiene permiso de ejecución sobre él; asignar a otra variable el resultado de comprobar si el archivo es un enlace
simbólico; mostrar el valor de las dos variables anteriores con un mensaje que aclare su significado. Pruebe el
guion ejecutándolo con /bin/cat y también con /bin/rnano.**

```bash
#!/bin/bash
# Titulo:       comprueba
```

```
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el archivo es plano y tienes
#               permiso de ejecución sobre él o si es un enlace
#               simbólico
# Opciones: ninguna
# Uso: comprueba <archivo>

planoejecucion=`([ -x $1 ] && [ -f $1 ]) && echo "es" || echo "no es"`
enlacesimbolico=`[ -h $1 ] && echo "es" || echo "no es"`

echo "El archivo $1 $planoejecucion un archivo plano ejecutable"
echo "El archivo $1 $enlacesimbolico un enlace simbólico"
```

Nota: hemos usado la sintaxis [ ] en lugar del comando test.

```console
$ chmod +x comprueba
```

Funcionamiento:

```console
$ ./comprueba /bin/cat
El archivo /bin/cat es un archivo plano ejecutable
El archivo /bin/cat no es un enlace simbólico

$ ./comprueba /bin/rnano
El archivo /bin/rnano es un archivo plano ejecutable
El archivo /bin/rnano es un enlace simbólico
```

Ejercicio 5.10)

**Ejecute `help test` y anote qué otros operadores se pueden utilizar con la orden `test` y para
qué sirven. Ponga un ejemplo de uso de la orden `test` comparando dos expresiones aritméticas y otro
comparando dos cadenas de caracteres.**

`test` puede operar con strings:

```console
    String operators:

      -z STRING      True if string is empty.

      -n STRING
         STRING      True if string is not empty.

      STRING1 = STRING2
                     True if the strings are equal.
      STRING1 != STRING2
                     True if the strings are not equal.
      STRING1 < STRING2
                     True if STRING1 sorts before STRING2
lexicographically.
      STRING1 > STRING2
```

```
                    True if STRING1 sorts after STRING2
lexicographically.

    arg1 OP arg2   Arithmetic tests.  OP is one of -eq, -ne,
                         -lt, -le, -gt, or -ge.
```

Compración de cadenas:

```console
$ echo `[ "hola" = "hola" ] && echo "hola y hola son cadenas
idénticas"`
 hola y hola son cadenas idénticas
```

Comparaciones aritméticas

```console
$ echo `[ 3 -ge 2 ] && echo "3 es mayor que 2"`
3 es mayor que 2
```

Ejercicio 11

Responda a los siguientes apartados:

a) Razone qué hace la siguiente orden:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe"; fi
```

Esta orden imprime por pantalla que el archivo sesion5.pdf del
directorio
donde es ejecutada la orden existe, si este, además de existir, es un
archivo
plano `(-f)`

**b) Añada los cambios necesarios en la orden anterior para que
también muestre un mensaje de aviso en caso
de no existir el archivo. (Recuerde que, para escribir de forma
legible una orden que ocupe más de una
línea, puede utilizar el carácter " \ " como final de cada línea que
no sea la última.)**

Para mostrar un mensaje en caso de que el archivo no exista:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe"; fi
```

**c) Sobre la solución anterior, añada un bloque elif para que, cuando
no exista el archivo ./sesion5.pdf,

compruebe si el archivo /bin es un directorio. Ponga los mensajes adecuados para conocer el resultado
en cada caso posible.**

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf existe\n"; \
  elif [ -d /bin ]; then printf "/bin es un directorio\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe y /bin no es un \
directorio"; fi

/bin es un directorio
```

**d) Usando como base la solución del apartado anterior, construya un guion que sea capaz de hacer lo mismo
pero admitiendo como parámetros la ruta relativa del primer archivo a buscar y la ruta absoluta del
segundo. Pruébelo con los dos archivos del apartado anterior.**

```bash
#!/bin/bash
# Titulo:       existe
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el primer argumento en un archivo plano
#               y existe o si el segundo es un directorio
# Opciones: ninguna
# Uso: existe <ruta_relativa_posible_archivo>
<ruta_absoluta_posible_directorio>

if [ -f $1 ]; then
    printf "El archivo $1 existe\n";
elif [ -d $2 ]; then
    printf "El archivo $1 no existe pero $2 es un directorio\n";
else
    printf "El archivo $1 no existe ni $2 es un directorio\n";
fi
```

Probemos teniendo en cuenta que el archivo `./ca` existe

```console
$ chmod +x existe

$ ./existe ./ca /bin
El archivo ./ca existe

$ ./existe ./co /bin
El archivo ./co no existe pero /bin es un directorio

$ ./existe ./co /bon
El archivo ./co no existe ni /bon es un directorio
```

Ejercicio 12

**Construya un guion que admita como argumento el nombre de un archivo o directorio y que
permita saber si somos el propietario del archivo y si tenemos permiso de lectura sobre él.**

La opción `-O` de `test`nos permite conocer si el archivo pertenece al usuario.
Y la opción `-r` de `test` permite conocer si podemos leer el archivo.

```bash
#!/bin/bash
# Titulo:       lecturaypropio
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si un archivo pertenece al usuario y
#               si este tiene permisos de lectura sobre él.
# Opciones: ninguna
# Uso: lecturaypropio <archivo>

if [ -O $1 ]; then
    echo "Eres el propietario del archivo $1";
else
    echo "No eres el propietario del archivo $1";
fi

if [ -r $1 ]; then
    echo "Tienes permisos de lectura sobre el archivo $1";
else
    echo "No tienes permisos de lectura sobre el archivo $1";
fi
```

```console
$ chmod +x lecturaypropio

$ ./lecturaypropio ca
```

Eres el propietario del archivo ca
Tienes permisos de lectura sobre el archivo ca

Ejercicio 13

**Escriba un guion que calcule si el número de días que faltan hasta fin de año es múltiplo de cinco
o no, y que comunique el resultado de la evaluación. Modifique el guion anterior para que admita la opción -h de
manera que, al ejecutarlo con esa opción, muestre información de para qué sirve el guion y cómo debe ejecutarse.**

```bash
#!/bin/bash
# Titulo:       diasmultiplo
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
```

```
# Descripción:  Comprueba si el número de días restantes para fin de
año
#               el múltiplo de 5
# Opciones: ninguna
# Uso: diasmultiplo [-h]

if [ "$1" == "-h" ]; then
    echo "Este programa comprueba si el número de días restantes para
\
fin de año es múltiplo de 5."
    echo "Para ejecutarlo simplemente ejecute ./diasmultiplo";
else
    dias_restantes=$[365 -$(date +%j)]
    echo "Quedan $dias_restantes días para el fin de año."

    if [ $[dias_restantes % 5] == 0 ]; then
        echo "Y $dias_restantes es múltiplo de 5!";
    else
        echo "Pero $dias_restantes no es múltiplo de 5";
    fi

fi
```

Funcionamiento

```console
$ ./diasmultiplo -h
Este programa comprueba si el número de días restantes para fin de año
es múltiplo de 5.
Para ejecutarlo simplemente ejecute ./diasmultiplo

$ ./diasmultiplo
Quedan 73 días para el fin de año.
Pero 73 no es múltiplo de 5
```

Ejercicio 14

¿Qué pasa en el ejemplo anterior si eliminamos la redirección de la orden `if`?

Lo que ocurre al eliminar la redirección del if es que, en caso de error,
se mostrará tanto nuestro mensaje propio como el error de la propia orden rm.

Ejercicio 15

Haciendo uso del mecanismo de cauces y de la orden **echo , construya un guion que admita un
argumento y que informe si el argumento dado es una única letra, en mayúsculas o en minúsculas, o es algo
distinto de una única letra.**

```bash
#!/bin/bash
# Titulo:       unicaletra
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Informa si el argumento dado es una única letra
#               en mayúsculas o en minúsculas o es algo distinto
#               de una única letra
# Opciones: ninguna
# Uso: unicaletra <algo>

if  echo $1 | grep '^[a-Z]\{1\}$' 2> /dev/null ; then
    echo "Es una única letra";
else
    echo "Es algo distinto de una única letra";
fi 2> /dev/null
```

Ejercicio 16

**Haciendo uso de expresiones regulares, escriba una orden que permita buscar en el árbol de
directorios los nombres de los archivos que contengan al menos un dígito y la letra e. ¿Cómo sería la orden si
quisiéramos obtener los nombres de los archivos que tengan la letra e y no contengan ni el 0 ni el 1?**

Ejercicio 17

Utilizando la orden `grep` , exprese una forma alternativa de realizar lo mismo que con `wc -l`.

Tema 5

###Fundamentos del Software

##Relacion de ejercicios 5. Expresiones con variables y expresiones regulares

#####Ejercicio 5.1: Utilizando una variable que contenga el valor entero 365 y otra que guarde el numéro del diá actual del añ̃o en curso, realice la misma operacioń del ejemplo anterior usando cada una de las diversas formas de caÍculo comentadas hasta el momento, es decir, utilizando expr, $((...)) y $[...].

```
        dia=365
    fechaActual=$(date +%j)
        echo "Faltan $(( (dia - fechaActual) / 7 )) semanas hasta fin de año"
    echo "Faltan $[ (dia - fechaActual) / 7 ] semanas hasta fin de año"
```

#####Ejercicio 5.2: Realice las siguientes operaciones para conocer el funcionamiento del operador de incremento como sufijo y como prefijo. Razone el resultado obtenido en cada una de ellas:

```
        $ v=1
        $ echo $v
    1
        $ echo $((v++))
    1
```

```
$ echo $v
2
    $ echo $((++v))
2
    $ echo $v
    3
```

#####Ejercicio 5.3: Utilizando el operador de división, ponga un caso concreto donde se aprecie que la asignación abreviada es equivalente a la asignación completa, es decir, que x/=y equivale a x=x/y.

```
    x=10
y=5

echo $((x/=y))
2

echo $((x=x/y))
2
```

#####Ejercicio 5.4: Compruebe que´ocurre si en el ejemplo anterior utiliza comillas dobles o simples para acotar todo lo que sigue a la orden echo. ¿Que´sucede si se acota entre comillas dobles solamente la expresión aritmética que se quiere calcular?, ¿y si se usan comillas simples?

```
    echo 6/5|bc -l
1.20000000000000000000

echo "6/5|bc -l"
6/5|bc -l

echo '6/5|bc -l'
6/5|bc -l

echo "6/5"|bc -l
1.20000000000000000000

echo '6/5'|bc -l
1.20000000000000000000
```

#####Ejercicio 5.5: Calcule con decimales el resultado de la expresión aritmética (3-2)/5. Escriba todas las expresiones que haya probado hasta dar con una respuesta válida. Utilizando una solución válida, compruebe que´sucede cuando la expresión aritmética se acota entre comillas dobles; ¿que´ocurre si se usan comillas simples?, ¿y si se ponen apóstrofos inversos?

```
    echo `expr $[3-2]/5`|bc -l
.20000000000000000000
```

#####Ejercicio 5.6: Consulte la sintaxis completa de la orden let utilizando la orden de ayuda para las órdenes empotradas (help let) y tome nota de su sintaxis general.

```
    help let
```

#####Ejercicio 5.7: Con la orden let es posible realizar asignaciones muítiples y utilizar operadores que nosotros no hemos mencionado anteriormente. Ponga un ejemplo de asignacioń muítiple y, por otra parte, copie en un archivo el orden en el que se evaluán los operadores que admite. Apoýese a traveś de la ayuda que ofrece help let.

let a=4+5 b=5+6

#####Ejercicio 5.8: Haciendo uso de las oŕdenes conocidas hasta el momento, construya un guion que admita dos paraḿetros, que compare por separado si el primer paraḿetro que se le pasa es igual al segundo, o es menor, o es mayor, y que informe tanto del valor de cada uno de los paraḿetros como del resultado de cada una de las evaluaciones mostrando un 0 o un 1 seguń corresponda.

```
1  #! /bin/bash
2  a=$1
3  b=$2
4  echo "a es igual a: "$a
5  echo "b es igual a: "$b
6  echo "a y b son igules: " $[$a==$b]
7  echo "a es menor que b : " $[$a<$b]
8  echo "b es menor que a : "$[$a>$b]
```

#####Ejercicio 5.9: Usando test, construya un guion que admita como paraḿetro un nombre de archivo y realice las siguientes acciones: asignar a una variable el resultado de comprobar si el archivo dado como paraḿetro es plano y tiene permiso de ejecucioń sobre eí; asignar a otra variable el resultado de comprobar si el archivo es un enlace simboĺico; mostrar el valor de las dos variables anteriores con un mensaje que aclare su significado. Pruebe el guion ejecutańdolo con /bin/cat y tambień con /bin/rnano.

#####Ejercicio 5.10: Ejecute help test y anote que´otros operadores se pueden utilizar con la orden test y para que´sirven. Ponga un ejemplo de uso de la orden test comparando dos expresiones aritmeticas y otro comparando dos cadenas de caracteres.

#####Ejercicio 5.11: Responda a los siguientes apartados:

a) Razone que´hace la siguiente orden:

if test -f ./sesion5.pdf ; then printf “El archivo ./sesion5.pdf existe\n”; fi

b) Anãda los cambios necesarios en la orden anterior para que tambień muestre un mensaje de aviso en caso de no existir el archivo. (Recuerde que, para escribir de forma legible una orden que ocupe maś de una liñea, puede utilizar el caracter “\” como final de cada liñea que no sea la uítima.)

c) Sobre la solucioń anterior, anãda un bloque elif para que, cuando no exista el archivo ./sesion5.pdf, compruebe si el archivo /bin es un directorio. Ponga los mensajes adecuados para conocer el resultado en cada caso posible.

d) Usando como base la solucioń del apartado anterior, construya un guion que sea capaz de hacer lo mismo pero admitiendo como paraḿetros la ruta relativa del primer archivo a buscar y la ruta absoluta del segundo. Pruebelo con los dos archivos del apartado anterior.

#####Ejercicio 5.12: Construya un guion que admita como argumento el nombre de un archivo o directorio y que permita saber si somos el propietario del archivo y si tenemos permiso de lectura sobre eĺ.

#####Ejercicio 5.13: Escriba un guion que calcule si el numéro de diás que faltan hasta fin de anõ es muĺtiplo de cinco o no, y que comunique el resultado de la evaluacioń. Modifique el guion anterior para que admita la opcioń -h de manera que, al ejecutarlo con esa opcioń, muestre informacioń de para que´ sirve el guion y comó debe ejecutarse.
#####El siguiente guion de ejemplo se puede utilizar para borrar el archivo temporal que se le de´como argumento. Si rm devuelve 0, se muestra el mensaje de confirmacioń del borrado; en caso contrario, se muestra el coḍigo de error. Como se puede apreciar, hemos utilizado la variable $LINENO que indica la liń ea actualmente en ejecucioń dentro del guion.

```bash
#!/bin/bash
declare -rx SCRIPT=${0##*/}
# donde SCRIPT contiene soĺo el nombre del guioń # ${var##Patron}
actuá eliminando de $var aquella parte # que cumpla de $Patron desde
el principio de $var
# En este caso: elimina todo lo que precede al
# uĺtimo slash “/".
if rm ${1} ; then
        printf "%s\n" "$SCRIPT: archivo temporal borrado"
else
    STATUS=177
        printf "%s – coḍigo de finalizacion %d\n" \
        "$SCRIPT:$LINENO no es posible borrar archivo" $STATUS
fi 2> /dev/null

```

#####Ejercicio 5.14: ¿Que´pasa en el ejemplo anterior si eliminamos la redireccioń de la orden if?

#####Ejercicio 5.15: Haciendo uso del mecanismo de cauces y de la orden echo, construya un guion que admita un argumento y que informe si el argumento dado es una uńica letra, en mayuśculas o en minuśculas, o es algo distinto de una uńica letra.

#####Ejercicio 5.16: Haciendo uso de expresiones regulares, escriba # Práctica 5: expresiones con variable y expresiones regulares

Ejercicio 1

**Utilizando una variable que contenga el valor entero 365 y otra que guarde el número del día actual
del año en curso, realice la misma operación del ejemplo anterior usando cada una de las diversas formas de
cálculo comentadas hasta el momento, es decir, utilizando `expr`, `$(( ... ))` y `$[ ... ]`.**

Primero declaramos las dos variables mediante el comando
declare.

```console
```

```
$ declare -i dias_anio=365
$ declare -i dia_actual=$(date +%j)
```

Ahora realizamos la operacion de dos formas:

```console
$ echo "Faltan $(( (dias_anio-dia_actual) / 7)) semanas hasta el fin
de año"
$ echo "Faltan $[(dias_anio-dia_actual)/7] semanas hasta el fin de
año"
```

Nótese que son necesarios los dobles paréntesis en la expresión $((), y que ninguno de los casos es necesario preceder el nombre de las variables con $.

Ejercicio 2

**Realice las siguientes operaciones para conocer el funcionamiento del operador de incremento
como sufijo y como prefijo. Razone el resultado obtenido en cada una de ellas:**

```console
$ v=1
$ echo $v
$ echo $((v++))
$ echo $v
$ echo $((++v))
$ echo $v
```

```console
$ v=1
```

La variable es declarada e inicializada a 1.

```console
$ echo $v
1
```

Se muestra el valor de la variable, que es
claramente 1.

```console
$ echo $((v++))
1
```

Como v++ es incremento en una unidad sufijo, primero
se muestra el valor de la variable y luego se incrementa
ésta en 1.

```console
$ echo $v
2
```

```

Como ante se ha incrementado en 1, ahora el valor de la
variable es 2.

```console
$ echo $((++v))
3
```

Como `v--` es incremento en una unidad prefijo, primero
se incrementa v en una unidad y luego se muestra el valor
de la variable. Como el valor de la variable anteriormente
era 2, aumenta a 3 que es lo que se muestra


```console
$ echo $v
3
```

El valor mostrado es 3, pues no se ha realizado ningún
cambio desde el último incremento.

Ejercicio 3

**Utilizando el operador de división, ponga un caso concreto donde se
aprecie que la asignación
abreviada es equivalente a la asignación completa, es decir, que x/=y
equivale a x=x/y.
En el resultado del cálculo de expresiones aritméticas, bash solamente
trabaja con números enteros, por lo que si
se necesitase calcular un resultado con decimales, habría que utilizar
una forma alternativa, como puede ser la
ofrecida por la orden `bc` , cuya opción `- l` , letra "ele", permite
hacer algunos cálculos matemáticos (admite otras
posibilidades que pueden verse mediante `man`).**

Utilizando la asignación completa

```console
$ x=10
$ y=2
$ echo $[x=x/y]
  5
$ echo $x
  5
```

Utilizando la asignación abreviada

```console
$ x=10
$ y=2
$ echo $[x/=y]
5
$ echo $x
5
```
```

Lo que muestra que son equivalentes.

Ejercicio 4

**Compruebe qué ocurre si en el ejemplo anterior utiliza comillas dobles o simples para acotar todo lo
que sigue a la orden `echo`. ¿Qué sucede si se acota entre comillas dobles solamente la expresión aritmética que se
quiere calcular?, ¿y si se usan comillas simples?**

```console
$ echo "6/5 | bc -l"
6/5 | bc -l

$ echo "6/5" | bc -l
1.20000000000000000000

$ echo '6/5 | bc -l'
6/5 | bc -l

$ echo '6/5' | bc -l
1.20000000000000000000
```

No hay diferencia entre usar comillas simples y dobles
en este caso. Cuando se acota todo lo que sigue a echo,
se muestra literalmente eso. Sin embargo, si sólo se acota
la expresión aritmetica, el resultado 6/5 es pasado mediante
el cauce a la orden `bc -l`.

Ejercicio 5

**Calcule con decimales el resultado de la expresión aritmética (3-2)/5. Escriba todas las
expresiones que haya probado hasta dar con una respuesta válida. Utilizando una solución válida, compruebe qué
sucede cuando la expresión aritmética se acota entre comillas dobles; ¿qué ocurre si se usan comillas simples?, ¿y
si se ponen apóstrofos inversos?**

Probando como en el ejercicio anterior:

```
$ echo (3-2)/5 | bc -l
bash: error sintáctico cerca del elemento inesperado `3-2'
```

Como vemos la expresión (3-2)/5 no se ha evaluado ya que no
iba acotada por $(()) o por $[].

```
$ echo $[(3-2)/5] | bc -l
0
```

En este caso la expresión sí se ha ejecutado, pero la divisón

entre 1 y 5 se realiza de forma entera, es decir resultado 0.
Por tanto, la orden bc -l no devuelve otra expresión decimal
que el propio 0.

```
$ echo "(3-2)/5" | bc -l
.20000000000000000000
```

En este caso, la solución sí es la correcta.

Probemos ahora qué sucede con comillas simples:

```
$ echo '(3-2)/5' | bc -l
.20000000000000000000
```

Es decir, las expresiones son equivalentes.

En el caso de apóstrofos inversos.

```
$ echo `(3-2)/5` \ bc -l
bash: command substitution: línea 1: error sintáctico cerca del
elemento inesperado `/5'
bash: command substitution: línea 1: `(3-2)/5'
```

Resulta error, ya que los apóstrofos inversos intentan
ejecutar la orden `(3-2)/5`, la cual hemos visto que era posible
evaluar debido a la falta de `$(())` o `$[]`.

Ejercicio 6

**Consulte la sintaxis completa de la orden `let` utilizando la orden
de ayuda para las órdenes
empotradas (`help let`) y tome nota de su sintaxis general.**

`let arg [arg ...]`

Evalúa cada ARG como una expresión aritmética. La evaluación se hace
con enteros de longitud fija, sin revisar desbordamientos, aunque la
la división por 0 se captura y se marca como un error. La siguiente
lista de operadores está agrupada en niveles de operadores de la misma
prioridad. Se muestran los niveles en orden de prioridad decreciente.

Ejercicio 7

**Al realizar el ejercicio anterior habrá observado que la orden `let`
admite asignaciones múltiples y
operadores que nosotros no hemos mencionado anteriormente. Ponga un
ejemplo de asignación múltiple y, por
otra parte, copie en un archivo el orden en el que se evalúan los
operadores que admite.**

```console
$ let a=1,b=2
```

```
$ echo "a es igual a $a y b es igual a $b".
a es igual a 1 y b es igual a 2
```

El orden en el se que se evalúan los operadores:

```console
    id++, id-- post-incremento, post-decremento de variable
    ++id, --id pre-incremento, pre-decremento de variable
    -, +              menos, más unario
    !, ~              negación lógica y basada en bits
    **          exponenciación
    *, /, %           multiplicación, división, residuo
    +, -              adición, sustracción
    <<, >>            desplazamientos de bits izquierdo y derecho
    <=, >=, <, >      comparación
    ==, !=            equivalencia, inequivalencia
    &           AND de bits
    ^           XOR de bits
    |           OR de bits
    &&          AND lógico
    ||          OR lógico
    expr ? expr : expr
                operador condicional
    =, *=, /=, %=,
    +=, -=, <<=, >>=,
    &=, ^=, |= asignación
```

Ejercicio 8

**Haciendo uso de las órdenes conocidas hasta el momento, construya un guion que admita dos
parámetros, que compare por separado si el primer parámetro que se le pasa es igual al segundo, o es menor, o es
mayor, y que informe tanto del valor de cada uno de los parámetros como del resultado de cada una de las
evaluaciones mostrando un 0 o un 1 según corresponda.**

Script compara

```bash
#!/bin/bash
# Titulo:       compara
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Compara los dos argumentos dados
# Opciones: ninguna
# Uso: compara <num1> <num2>

echo "El valor del primer parámetro es $1 y del segundo $2"

let mayor="$1 > $2"
let menor="$1 < $2"
let iguales="$1 == $2"

echo "¿$1 es mayor que $2?: $mayor"
echo "¿$1 es menor que $2?: $menor"
```

```
echo "¿$1 es igual que $2?: $iguales"
```

Procedamos a ejecutarlo

```console
$ chmod +x compara
$ ./compara 1 2

El valor del primer parámetro es 2 y del segundo 1
¿2 es mayor que 1?: 1
¿2 es menor que 1?: 0
¿2 es igual que 1?: 0
```

```console
$ ./compara 2 1

El valor del primer parámetro es 1 y del segundo 2
¿1 es mayor que 2?: 0
¿1 es menor que 2?: 1
¿1 es igual que 2?: 0

$ ./compara 1 1

El valor del primer parámetro es 1 y del segundo 1
¿1 es mayor que 1?: 0
¿1 es menor que 1?: 0
¿1 es igual que 1?: 1
```

Ejercicio 9

**Usando `test` , construya un guion que admita como parámetro un nombre de archivo y realice las
siguientes acciones: asignar a una variable el resultado de comprobar si el archivo dado como parámetro es plano
y tiene permiso de ejecución sobre él; asignar a otra variable el resultado de comprobar si el archivo es un enlace
simbólico; mostrar el valor de las dos variables anteriores con un mensaje que aclare su significado. Pruebe el
guion ejecutándolo con /bin/cat y también con /bin/rnano.**

```bash
#!/bin/bash
# Titulo:       comprueba
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el archivo es plano y tienes
#               permiso de ejecución sobre él o si es un enlace
#               simbólico
# Opciones: ninguna
# Uso: comprueba <archivo>

planoejecucion=`([ -x $1 ] && [ -f $1 ]) && echo "es" || echo "no es"`
enlacesimbolico=`[ -h $1 ] && echo "es" || echo "no es"`

echo "El archivo $1 $planoejecucion un archivo plano ejecutable"
```

```
echo "El archivo $1 $enlacesimbolico un enlace simbólico"
```

Nota: hemos usado la sintaxis [ ] en lugar del comando test.

```console
$ chmod +x comprueba
```

Funcionamiento:

```console
$ ./comprueba /bin/cat
El archivo /bin/cat es un archivo plano ejecutable
El archivo /bin/cat no es un enlace simbólico

$ ./comprueba /bin/rnano
El archivo /bin/rnano es un archivo plano ejecutable
El archivo /bin/rnano es un enlace simbólico
```

Ejercicio 5.10)

**Ejecute `help test` y anote qué otros operadores se pueden utilizar con la orden `test` y para
qué sirven. Ponga un ejemplo de uso de la orden `test` comparando dos expresiones aritméticas y otro
comparando dos cadenas de caracteres.**

`test` puede operar con strings:

```console
    String operators:

      -z STRING      True if string is empty.

      -n STRING
         STRING      True if string is not empty.

      STRING1 = STRING2
                     True if the strings are equal.
      STRING1 != STRING2
                     True if the strings are not equal.
      STRING1 < STRING2
                     True if STRING1 sorts before STRING2
lexicographically.
      STRING1 > STRING2
                     True if STRING1 sorts after STRING2
lexicographically.

    arg1 OP arg2   Arithmetic tests.  OP is one of -eq, -ne,
                         -lt, -le, -gt, or -ge.
```

Compración de cadenas:

```console
$ echo `[ "hola" = "hola" ] && echo "hola y hola son cadenas
idénticas"`
 hola y hola son cadenas idénticas
```

```

Comparaciones aritméticas

```console
$ echo `[ 3 -ge 2 ] && echo "3 es mayor que 2"`
3 es mayor que 2
```

Ejercicio 11

Responda a los siguientes apartados:

a) Razone qué hace la siguiente orden:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe"; fi
```

Esta orden imprime por pantalla que el archivo sesion5.pdf del
directorio
donde es ejecutada la orden existe, si este, además de existir, es un
archivo
plano `(-f)`

**b) Añada los cambios necesarios en la orden anterior para que
también muestre un mensaje de aviso en caso
de no existir el archivo. (Recuerde que, para escribir de forma
legible una orden que ocupe más de una
línea, puede utilizar el carácter " \ " como final de cada línea que
no sea la última.)**

Para mostrar un mensaje en caso de que el archivo no exista:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe"; fi
```

**c) Sobre la solución anterior, añada un bloque elif para que, cuando
no exista el archivo ./sesion5.pdf,
compruebe si el archivo /bin es un directorio. Ponga los mensajes
adecuados para conocer el resultado
en cada caso posible.**

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe\n"; \
  elif [ -d /bin ]; then printf "/bin es un directorio\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe y /bin no es un
\
directorio"; fi
```
```

/bin es un directorio
```

**d) Usando como base la solución del apartado anterior, construya un guion que sea capaz de hacer lo mismo
pero admitiendo como parámetros la ruta relativa del primer archivo a buscar y la ruta absoluta del
segundo. Pruébelo con los dos archivos del apartado anterior.**

```bash
#!/bin/bash
# Titulo:       existe
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el primer argumento en un archivo plano
#               y existe o si el segundo es un directorio
# Opciones: ninguna
# Uso: existe <ruta_relativa_posible_archivo>
<ruta_absoluta_posible_directorio>

if [ -f $1 ]; then
    printf "El archivo $1 existe\n";
elif [ -d $2 ]; then
    printf "El archivo $1 no existe pero $2 es un directorio\n";
else
    printf "El archivo $1 no existe ni $2 es un directorio\n";
fi
```

Probemos teniendo en cuenta que el archivo `./ca` existe

```console
$ chmod +x existe

$ ./existe ./ca /bin
El archivo ./ca existe

$ ./existe ./co /bin
El archivo ./co no existe pero /bin es un directorio

$ ./existe ./co /bon
El archivo ./co no existe ni /bon es un directorio
```

Ejercicio 12

**Construya un guion que admita como argumento el nombre de un archivo o directorio y que
permita saber si somos el propietario del archivo y si tenemos permiso de lectura sobre él.**

La opción `-O` de `test`nos permite conocer si el archivo pertenece al usuario.
Y la opción `-r` de `test` permite conocer si podemos leer el archivo.

```bash
#!/bin/bash
```
```

```
# Titulo:       lecturaypropio
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si un archivo pertenece al usuario y
#               si este tiene permisos de lectura sobre él.
# Opciones: ninguna
# Uso: lecturaypropio <archivo>

if [ -O $1 ]; then
    echo "Eres el propietario del archivo $1";
else
    echo "No eres el propietario del archivo $1";
fi

if [ -r $1 ]; then
    echo "Tienes permisos de lectura sobre el archivo $1";
else
    echo "No tienes permisos de lectura sobre el archivo $1";
fi
```

```console
$ chmod +x lecturaypropio

$ ./lecturaypropio ca
```

Eres el propietario del archivo ca
Tienes permisos de lectura sobre el archivo ca

Ejercicio 13

**Escriba un guion que calcule si el número de días que faltan hasta fin de año es múltiplo de cinco
o no, y que comunique el resultado de la evaluación. Modifique el guion anterior para que admita la opción -h de
manera que, al ejecutarlo con esa opción, muestre información de para qué sirve el guion y cómo debe ejecutarse.**

```bash
#!/bin/bash
# Titulo:       diasmultiplo
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el número de días restantes para fin de
año
#               el múltiplo de 5
# Opciones: ninguna
# Uso: diasmultiplo [-h]

if [ "$1" == "-h" ]; then
    echo "Este programa comprueba si el número de días restantes para
\
fin de año es múltiplo de 5."
    echo "Para ejecutarlo simplemente ejecute ./diasmultiplo";
else
    dias_restantes=$[365 -$(date +%j)]
    echo "Quedan $dias_restantes días para el fin de año."
```

```
    if [ $[dias_restantes % 5] == 0 ]; then
        echo "Y $dias_restantes es múltiplo de 5!";
    else
        echo "Pero $dias_restantes no es múltiplo de 5";
    fi

fi
```

Funcionamiento

```console
$ ./diasmultiplo -h
Este programa comprueba si el número de días restantes para fin de año
es múltiplo de 5.
Para ejecutarlo simplemente ejecute ./diasmultiplo

$ ./diasmultiplo
Quedan 73 días para el fin de año.
Pero 73 no es múltiplo de 5
```

Ejercicio 14

¿Qué pasa en el ejemplo anterior si eliminamos la redirección de la orden `if`?

Lo que ocurre al eliminar la redirección del if es que, en caso de error,
se mostrará tanto nuestro mensaje propio como el error de la propia orden rm.

Ejercicio 15

Haciendo uso del mecanismo de cauces y de la orden **echo , construya un guion que admita un
argumento y que informe si el argumento dado es una única letra, en mayúsculas o en minúsculas, o es algo
distinto de una única letra.**

```bash
#!/bin/bash
# Titulo:       unicaletra
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Informa si el argumento dado es una única letra
#               en mayúsculas o en minúsculas o es algo distinto
#               de una única letra
# Opciones: ninguna
# Uso: unicaletra <algo>

if  echo $1 | grep '^[a-Z]\{1\}$' 2> /dev/null ; then
```

```
    echo "Es una única letra";
else
    echo "Es algo distinto de una única letra";
fi 2> /dev/null
```

Ejercicio 16

**Haciendo uso de expresiones regulares, escriba una orden que permita buscar en el árbol de
directorios los nombres de los archivos que contengan al menos un dígito y la letra e. ¿Cómo sería la orden si
quisiéramos obtener los nombres de los archivos que tengan la letra e y no contengan ni el 0 ni el 1?**

Ejercicio 17

Utilizando la orden `grep` , exprese una forma alternativa de realizar lo mismo que con `wc -l`.
los archivos que contengan al menos un dígito y la letra e. ¿Cómo sería la orden si quisiéramos obtener los nombres de los archivos que tengan la letra e y no contengan ni el 0 ni el 1?

#####Ejercicio 5.17: Utilizando la orden grep, exprese una forma alternativa de realizar lo mismo que con wc -l.

Escriba, al menos, cinco variables de entorno junto con el valor que tienen.

Práctica 4: Variables, alias, órdenes de búsqueda y guiones

Ejercicio 1

Escriba, al menos, cinco variables de entorno junto con el valor que tienen.

```console
SESSION=xfce
SHELL=/usr/bin/fish
USER=ricardo
```

a) Imprima los valores de las tres variables en tres columnas con 15 caracteres de ancho.
b) ¿Son variables locales o globales?
c) Borre la variable VAR2.
d) Abra otra ventana de tipo terminal, ¿puede visualizar las dos variables restantes?
e) Cree una variable de tipo vector con los valores iniciales de las tres variables.
f) Muestre el segundo elemento del vector creado en el apartado e.

Ejercicio 2

Ejecute las órdenes del cuadro e indique qué ocurre y cómo puede resolver la situación para que la variable NOMBRE se reconozca en el shell hijo.

```console
$ NOMBRE=FS $ echo $NOMBRE
```

```

Es mostrado el contenido de la variable, es decir, FS

```console
$ bash $ echo $NOMBRE
```

No es mostrado nada debido a que la variable era local. Cuando se ejecuta otra
shell, desaparece el contenido de la variable.

Esto puede solucionarse mediante la orden export.

```console
$ export NOMBRE=FS
$ bash
$ echo $NOMBRE
```

En este caso sí se muestra el contenido, FS.

Ejercicio 3

**Compruebe qué ocurre en las expresiones del ejemplo anterior si se quitan las comillas dobles del
final y se ponen después de los dos puntos. ¿Qué sucede si se sustituyen las comillas dobles por comillas simples?**

a)

Al ejecutar

```console
$ echo "Los archivos que hay en el directorio son $(ls -l)"

    Los archivos que hay en el directorio son: total 4
    -rw-r--r-- 1 ricardo ricardo 489 oct  4 13:14 practica4-
ricardoruiz.txt
```

Mientras que al ejecutar,

```console
$ echo "Los archivos que hay en el directorio son:" $(ls -l)

    Los archivos que hay en el directorio son: total 4 -rw-r--r-- 1
ricardo ricardo 489 oct 4 13:14 practica4-ricardoruiz.txt
```

Es decir, en el segundo caso, no se produce salto de línea.

b)

Si sustituimos las comillas dobles por las simples:

a)

```console
$ echo 'Los archivos que hay en el directorio son: $(ls -l)'
```
```

```
    Los archivos que hay en el directorio son: $(ls -l)
```

Es decir, la orden ls -l no es ejecutada, sino que se muestra literalmente.

b)

```console
 $ echo 'Los archivos que hay en el direcotrio son:' $(ls -l)

    Los archivos que hay en el directorio son: total 4 -rw-r--r-- 1 ricardo ricardo 489 oct 4 13:14 practica4-ricardoruiz.txt
```

La orden es ejecutada, y todo se muestra en una única línea.

Ejercicio 4

Pruebe la siguiente asignación:

```console
$ numero=$numero+1
$ echo $numero
    +1
```

Esto ha ocurrido porque no se ha utilizado la orden expr. Es decir, todo se ha convertido a un carácter (+1)

```
$ numero=1
$ echo $numero
1
$ numero=`expr $numero + 1`
```

Ejercicio 5

Construya un guion que acepte como argumento una cadena de texto (por ejemplo, su nombre) y que visualice en pantalla la frase Hola y el nombre dado como argumento.

Escribimos el guión `helloworld.sh`

```bash
    #!/bin/bash

    printf "Hola $1\n"
```

Posteriormente, le otorgamos permiso de ejecución con console

```console
```

```
    $ chmod +x helloworld.sh
```

Y lo ejecutamos con un argumento

```console
$ ./helloworld.sh Ricardo
        Hola Ricardo
```

Ejercicio 6

Varíe el guion anterior para que admita una lista de nombres.

Simplemente sustituimos $1 por $*

```bash
#!/bin/bash

printf "Hola $*\n"
```

Ejercicio 7

Cree tres variables llamadas VAR1, VAR2 y VAR3 con los siguientes valores respectivamente `hola`,`adios` y `14`.

a)
```console
  $ printf "%15q %15q %15d\n" $VAR1 $VAR2 $VAR3
                 hola           adios              14
```

b) Son variables locales, las hemos creado mediante `variable=valor`

c) `unset VAR1`

d) No, porque son variables locales

e) `vector=($VAR1 $VAR2 $VAR3)`

f) `echo ${vector[1]}`

Ejercicio 8

**Cree un alias que se llame `ne` (nombrado así para indicar el número de elementos) y que devuelva
el número de archivos que existen en el directorio actual. ¿Qué cambiaría si queremos que haga lo mismo pero en
el directorio home correspondiente al usuario que lo ejecuta?**

```console
$ alias ne='ls -l | wc -l'
```

Para realizar lo mismo pero en el directorio home del usuario que lo ejecuta:

```console
```

```
$ alias ne='ls ~ -l | wc -l'
```

Ejercicio 9

**Indique la línea de orden necesaria para buscar todos los archivos a partir del directorio home que
tengan un tamaño menor de un bloque. ¿Cómo la modificaría para que además imprima el resultado en un archivo
que se cree dentro del directorio donde nos encontremos y que se llame archivos?**

```console
    $ find ~ -size -1
```

Para imprimir el resultado en un archivo dentro del directorio actual

```console
    $ find ~ -size -1 > archivosP
```

Ejercicio 10

**Indique cómo buscaría todos aquellos archivos del directorio actual que contengan la palabra
"ejemplo".**

```console
$ grep ejemplo ./*
```

Ejercicio 11

Complete la información de find y grep utilizando para ello la orden man.

```console
$ man find
```

Podemos destacar los siguientes argumentos para el parámetro type

-type c
File is of type c:

b block (buffered) special

c character (unbuffered) special

d directory

p named pipe (FIFO)

f regular file

```
              l      symbolic link; this is never true if the -L
option or the -follow option is in effect, unless the symbolic  link
is  broken.
                     If you want to search for symbolic links when -L
is in effect, use -xtype.

              s      socket

    $ man grep


        Podemos añadir la opción -h

           -h, --no-filename
                      Suppress the prefixing of file names on output.
This is the default when there is only one file (or only standard
input) to search.
```

Ejercicio 12

Indique cómo buscaría si un usuario dispone de una cuenta en el sistema.

Usar `who` no es una opción, ya que solo devuelve los usuarios identificados
en el sistema. Para comprobar si un usuario dispone cuenta, es mejor buscar su
nombre de usuario en el archivo `/etc/passwd`

```console
    $ cat /etc/passwd | grep usuario-a-buscar
```

Ejercicio 13

**Indique cómo contabilizar el número de ficheros de la propia cuenta de usuario que no tengan
permiso de lectura para el resto de usuarios.**

```console
    $ find ~ ! -perm -o+r | wc -l
      34431
```

Esto indica, a find que devuelva los archivos que no tengan permiso de
lectura para el resto de usuarios del directorio `~`.
Y luego, esta salida es redireccionada a wc mediante un cauce (|). Este,
cuenta el número de líneas (correspondiente al número de ficheros) mediante la
opción `-l`.

Ejercicio 14

**Modifique el ejercicio 8 de forma que, en vez de un alias, sea un guion llamado `numE` el que

devuelva el número de archivos que existen en el directorio que se le pase como argumento.**

```bash
Escribimos el guión numE


#!/bin/bash
# Titulo:       numE
# Fecha:        05/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Cuenta el número de archivos del directorio
#               dado como primer argumento
# Opciones: ninguna
# Uso: numE directorio

ls -1 $1 | wc -l
```


Le otorgamos permisos de ejecución con:

```console
$ chmod +x numE
```

Lo ejecutamos y comprobamos su funcionamiento

Para el directorio actual `.` (contiene 4 archivos)

```console
$ ./numE .
   4
```

Para el directorio home

```console
$ ./numE ~
    35
```

Notar que esta orden muestra tanto archivos como directorios.

Tema 6

Práctica 5: expresiones con variable y expresiones regulares

Ejercicio 1

**Utilizando una variable que contenga el valor entero 365 y otra que guarde el número del día actual
del año en curso, realice la misma operación del ejemplo anterior usando cada una de las diversas formas de
cálculo comentadas hasta el momento, es decir, utilizando `expr`, `$(( ... ))` y `$[ ... ]`.**

Primero declaramos las dos variables mediante el comando
declare.

```console
$ declare -i dias_anio=365
$ declare -i dia_actual=$(date +%j)
```

Ahora realizamos la operacion de dos formas:

```console
$ echo "Faltan $(( (dias_anio-dia_actual) / 7)) semanas hasta el fin
de año"
$ echo "Faltan $[(dias_anio-dia_actual)/7] semanas hasta el fin de
año"
```

Nótese que son necesarios los dobles paréntesis en la expresión $(()),
y que ninguno de los casos es necesario preceder el nombre de las
variables con $.

Ejercicio 2

**Realice las siguientes operaciones para conocer el funcionamiento
del operador de incremento
como sufijo y como prefijo. Razone el resultado obtenido en cada una
de ellas:**

```console
$ v=1
$ echo $v
$ echo $((v++))
$ echo $v
$ echo $((++v))
$ echo $v
```

```console
$ v=1
```

La variable es declarada e inicializada a 1.

```console
$ echo $v
1
```

Se muestra el valor de la variable, que es
claramente 1.

```console
$ echo $((v++))
1
```

Como v++ es incremento en una unidad sufijo, primero
se muestra el valor de la variable y luego se incrementa

ésta en 1.

```console
$ echo $v
2
```

Como ante se ha incrementado en 1, ahora el valor de la
variable es 2.

```console
$ echo $((++v))
3
```

Como `v--` es incremento en una unidad prefijo, primero
se incrementa v en una unidad y luego se muestra el valor
de la variable. Como el valor de la variable anteriormente
era 2, aumenta a 3 que es lo que se muestra

```console
$ echo $v
3
```

El valor mostrado es 3, pues no se ha realizado ningún
cambio desde el último incremento.

Ejercicio 3

**Utilizando el operador de división, ponga un caso concreto donde se aprecie que la asignación
abreviada es equivalente a la asignación completa, es decir, que x/=y equivale a x=x/y.
En el resultado del cálculo de expresiones aritméticas, bash solamente trabaja con números enteros, por lo que si
se necesitase calcular un resultado con decimales, habría que utilizar una forma alternativa, como puede ser la
ofrecida por la orden `bc` , cuya opción `- l` , letra "ele", permite hacer algunos cálculos matemáticos (admite otras
posibilidades que pueden verse mediante `man`).**

Utilizando la asignación completa

```console
$ x=10
$ y=2
$ echo $[x=x/y]
  5
$ echo $x
  5
```

Utilizando la asignación abreviada

```console
$ x=10
$ y=2
```

```
$ echo $[x/=y]
5
$ echo $x
5
```

Lo que muestra que son equivalentes.

Ejercicio 4

**Compruebe qué ocurre si en el ejemplo anterior utiliza comillas dobles o simples para acotar todo lo
que sigue a la orden `echo`. ¿Qué sucede si se acota entre comillas dobles solamente la expresión aritmética que se
quiere calcular?, ¿y si se usan comillas simples?**

```console
$ echo "6/5 | bc -l"
6/5 | bc -l

$ echo "6/5" | bc -l
1.20000000000000000000

$ echo '6/5 | bc -l'
6/5 | bc -l

$ echo '6/5' | bc -l
1.20000000000000000000
```

No hay diferencia entre usar comillas simples y dobles
en este caso. Cuando se acota todo lo que sigue a echo,
se muestra literalmente eso. Sin embargo, si sólo se acota
la expresión aritmetica, el resultado 6/5 es pasado mediante
el cauce a la orden `bc -l`.

Ejercicio 5

**Calcule con decimales el resultado de la expresión aritmética (3-2)/5. Escriba todas las
expresiones que haya probado hasta dar con una respuesta válida. Utilizando una solución válida, compruebe qué
sucede cuando la expresión aritmética se acota entre comillas dobles; ¿qué ocurre si se usan comillas simples?, ¿y
si se ponen apóstrofos inversos?**

Probando como en el ejercicio anterior:

```
$ echo (3-2)/5 | bc -l
bash: error sintáctico cerca del elemento inesperado `3-2'
```

Como vemos la expresión (3-2)/5 no se ha evaluado ya que no
iba acotada por $(()) o por $[].

```
```

```
$ echo $[(3-2)/5] | bc -l
0
```

En este caso la expresión sí se ha ejecutado, pero la divisón entre 1 y 5 se realiza de forma entera, es decir resultado 0. Por tanto, la orden bc -l no devuelve otra expresión decimal que el propio 0.

```
$ echo "(3-2)/5" | bc -l
.20000000000000000000
```

En este caso, la solución sí es la correcta.

Probemos ahora qué sucede con comillas simples:

```
$ echo '(3-2)/5' | bc -l
.20000000000000000000
```

Es decir, las expresiones son equivalentes.

En el caso de apóstrofos inversos.

```
$ echo `(3-2)/5` \ bc -l
bash: command substitution: línea 1: error sintáctico cerca del elemento inesperado `/5'
bash: command substitution: línea 1: `(3-2)/5'
```

Resulta error, ya que los apóstrofos inversos intentan ejecutar la orden `(3-2)/5`, la cual hemos visto que era posible evaluar debido a la falta de `$(())` o `$[]`.

Ejercicio 6

Consulte la sintaxis completa de la orden `let` utilizando la orden de ayuda para las órdenes empotradas (`help let`) y tome nota de su sintaxis general.

`let arg [arg ...]`

Evalúa cada ARG como una expresión aritmética. La evaluación se hace con enteros de longitud fija, sin revisar desbordamientos, aunque la la división por 0 se captura y se marca como un error. La siguiente lista de operadores está agrupada en niveles de operadores de la misma prioridad. Se muestran los niveles en orden de prioridad decreciente.

Ejercicio 7

Al realizar el ejercicio anterior habrá observado que la orden `let` admite asignaciones múltiples y operadores que nosotros no hemos mencionado anteriormente. Ponga un ejemplo de asignación múltiple y, por otra parte, copie en un archivo el orden en el que se evalúan los operadores que admite.

```console
$ let a=1,b=2
$ echo "a es igual a $a y b es igual a $b".
a es igual a 1 y b es igual a 2
```

El orden en el se que se evalúan los operadores:

```console
    id++, id-- post-incremento, post-decremento de variable
    ++id, --id pre-incremento, pre-decremento de variable
    -, +               menos, más unario
    !, ~               negación lógica y basada en bits
    **          exponenciación
    *, /, %            multiplicación, división, residuo
    +, -               adición, sustracción
    <<, >>             desplazamientos de bits izquierdo y derecho
    <=, >=, <, >       comparación
    ==, !=             equivalencia, inequivalencia
    &           AND de bits
    ^           XOR de bits
    |           OR de bits
    &&          AND lógico
    ||          OR lógico
    expr ? expr : expr
                operador condicional
    =, *=, /=, %=,
    +=, -=, <<=, >>=,
    &=, ^=, |= asignación
```

Ejercicio 8

**Haciendo uso de las órdenes conocidas hasta el momento, construya un guion que admita dos
parámetros, que compare por separado si el primer parámetro que se le pasa es igual al segundo, o es menor, o es
mayor, y que informe tanto del valor de cada uno de los parámetros como del resultado de cada una de las
evaluaciones mostrando un 0 o un 1 según corresponda.**

Script compara

```bash
#!/bin/bash
# Titulo:       compara
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Compara los dos argumentos dados
# Opciones: ninguna
# Uso: compara <num1> <num2>

echo "El valor del primer parámetro es $1 y del segundo $2"

let mayor="$1 > $2"
```

```
let menor="$1 < $2"
let iguales="$1 == $2"

echo "¿$1 es mayor que $2?: $mayor"
echo "¿$1 es menor que $2?: $menor"
echo "¿$1 es igual que $2?: $iguales"
```

Procedamos a ejecutarlo

```console
$ chmod +x compara
$ ./compara 1 2

El valor del primer parámetro es 2 y del segundo 1
¿2 es mayor que 1?: 1
¿2 es menor que 1?: 0
¿2 es igual que 1?: 0
```

```console
$ ./compara 2 1

El valor del primer parámetro es 1 y del segundo 2
¿1 es mayor que 2?: 0
¿1 es menor que 2?: 1
¿1 es igual que 2?: 0

$ ./compara 1 1

El valor del primer parámetro es 1 y del segundo 1
¿1 es mayor que 1?: 0
¿1 es menor que 1?: 0
¿1 es igual que 1?: 1
```

Ejercicio 9

**Usando `test` , construya un guion que admita como parámetro un nombre de archivo y realice las
siguientes acciones: asignar a una variable el resultado de comprobar si el archivo dado como parámetro es plano
y tiene permiso de ejecución sobre él; asignar a otra variable el resultado de comprobar si el archivo es un enlace
simbólico; mostrar el valor de las dos variables anteriores con un mensaje que aclare su significado. Pruebe el
guion ejecutándolo con /bin/cat y también con /bin/rnano.**

```bash
#!/bin/bash
# Titulo:       comprueba
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el archivo es plano y tienes
#               permiso de ejecución sobre él o si es un enlace
#               simbólico
# Opciones: ninguna
# Uso: comprueba <archivo>
```

```
planoejecucion=`([ -x $1 ] && [ -f $1 ]) && echo "es" || echo "no es"`
enlacesimbolico=`[ -h $1 ] && echo "es" || echo "no es"`

echo "El archivo $1 $planoejecucion un archivo plano ejecutable"
echo "El archivo $1 $enlacesimbolico un enlace simbólico"
```

Nota: hemos usado la sintaxis [ ] en lugar del comando test.

```console
$ chmod +x comprueba
```

Funcionamiento:

```console
$ ./comprueba /bin/cat
El archivo /bin/cat es un archivo plano ejecutable
El archivo /bin/cat no es un enlace simbólico

$ ./comprueba /bin/rnano
El archivo /bin/rnano es un archivo plano ejecutable
El archivo /bin/rnano es un enlace simbólico
```

Ejercicio 5.10)

**Ejecute `help test` y anote qué otros operadores se pueden utilizar con la orden `test` y para
qué sirven. Ponga un ejemplo de uso de la orden `test` comparando dos expresiones aritméticas y otro
comparando dos cadenas de caracteres.**

`test` puede operar con strings:

```console
    String operators:

      -z STRING      True if string is empty.

      -n STRING
         STRING      True if string is not empty.

      STRING1 = STRING2
                     True if the strings are equal.
      STRING1 != STRING2
                     True if the strings are not equal.
      STRING1 < STRING2
                     True if STRING1 sorts before STRING2
lexicographically.
      STRING1 > STRING2
                     True if STRING1 sorts after STRING2
lexicographically.

    arg1 OP arg2   Arithmetic tests.  OP is one of -eq, -ne,
                         -lt, -le, -gt, or -ge.
```

Compración de cadenas:

```console
$ echo `[ "hola" = "hola" ] && echo "hola y hola son cadenas
idénticas"`
 hola y hola son cadenas idénticas
```

Comparaciones aritméticas

```console
$ echo `[ 3 -ge 2 ] && echo "3 es mayor que 2"`
3 es mayor que 2
```

Ejercicio 11

Responda a los siguientes apartados:

a) Razone qué hace la siguiente orden:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe"; fi
```

Esta orden imprime por pantalla que el archivo sesion5.pdf del
directorio
donde es ejecutada la orden existe, si este, además de existir, es un
archivo
plano `(-f)`

**b) Añada los cambios necesarios en la orden anterior para que
también muestre un mensaje de aviso en caso
de no existir el archivo. (Recuerde que, para escribir de forma
legible una orden que ocupe más de una
línea, puede utilizar el carácter " \ " como final de cada línea que
no sea la última.)**

Para mostrar un mensaje en caso de que el archivo no exista:

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe"; fi
```

**c) Sobre la solución anterior, añada un bloque elif para que, cuando
no exista el archivo ./sesion5.pdf,
compruebe si el archivo /bin es un directorio. Ponga los mensajes
adecuados para conocer el resultado
en cada caso posible.**

```console
$ if [ -f ./sesion5.pdf ]; then printf "El archivo ./sesion5.pdf
existe\n"; \
```

```
  elif [ -d /bin ]; then printf "/bin es un directorio\n"; \
  else printf "\n\nEl archivo ./sesion5.pdf no existe y /bin no es un
\
directorio"; fi

/bin es un directorio
```

**d) Usando como base la solución del apartado anterior, construya un guion que sea capaz de hacer lo mismo
pero admitiendo como parámetros la ruta relativa del primer archivo a buscar y la ruta absoluta del
segundo. Pruébelo con los dos archivos del apartado anterior.**

```bash
#!/bin/bash
# Titulo:       existe
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el primer argumento en un archivo plano
#               y existe o si el segundo es un directorio
# Opciones: ninguna
# Uso: existe <ruta_relativa_posible_archivo>
<ruta_absoluta_posible_directorio>

if [ -f $1 ]; then
    printf "El archivo $1 existe\n";
elif [ -d $2 ]; then
    printf "El archivo $1 no existe pero $2 es un directorio\n";
else
    printf "El archivo $1 no existe ni $2 es un directorio\n";
fi
```

Probemos teniendo en cuenta que el archivo `./ca` existe

```console
$ chmod +x existe

$ ./existe ./ca /bin
El archivo ./ca existe

$ ./existe ./co /bin
El archivo ./co no existe pero /bin es un directorio

$ ./existe ./co /bon
El archivo ./co no existe ni /bon es un directorio
```

Ejercicio 12

**Construya un guion que admita como argumento el nombre de un archivo o directorio y que
permita saber si somos el propietario del archivo y si tenemos permiso de lectura sobre él.**

La opción `-O` de `test`nos permite conocer si el archivo pertenece al usuario.
Y la opción `-r` de `test` permite conocer si podemos leer el archivo.

```bash
#!/bin/bash
# Titulo:       lecturaypropio
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si un archivo pertenece al usuario y
#               si este tiene permisos de lectura sobre él.
# Opciones: ninguna
# Uso: lecturaypropio <archivo>

if [ -O $1 ]; then
    echo "Eres el propietario del archivo $1";
else
    echo "No eres el propietario del archivo $1";
fi

if [ -r $1 ]; then
    echo "Tienes permisos de lectura sobre el archivo $1";
else
    echo "No tienes permisos de lectura sobre el archivo $1";
fi
```

```console
$ chmod +x lecturaypropio

$ ./lecturaypropio ca
```

Eres el propietario del archivo ca
Tienes permisos de lectura sobre el archivo ca

Ejercicio 13

**Escriba un guion que calcule si el número de días que faltan hasta fin de año es múltiplo de cinco
o no, y que comunique el resultado de la evaluación. Modifique el guion anterior para que admita la opción -h de
manera que, al ejecutarlo con esa opción, muestre información de para qué sirve el guion y cómo debe ejecutarse.**

```bash
#!/bin/bash
# Titulo:       diasmultiplo
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
# Descripción:  Comprueba si el número de días restantes para fin de año
#               el múltiplo de 5
# Opciones: ninguna
# Uso: diasmultiplo [-h]

if [ "$1" == "-h" ]; then
```

```
    echo "Este programa comprueba si el número de días restantes para
\
fin de año es múltiplo de 5."
    echo "Para ejecutarlo simplemente ejecute ./diasmultiplo";
else
    dias_restantes=$[365 -$(date +%j)]
    echo "Quedan $dias_restantes días para el fin de año."

    if [ $[dias_restantes % 5] == 0 ]; then
        echo "Y $dias_restantes es múltiplo de 5!";
    else
        echo "Pero $dias_restantes no es múltiplo de 5";
    fi

fi
```

Funcionamiento

```console
$ ./diasmultiplo -h
Este programa comprueba si el número de días restantes para fin de año
es múltiplo de 5.
Para ejecutarlo simplemente ejecute ./diasmultiplo

$ ./diasmultiplo
Quedan 73 días para el fin de año.
Pero 73 no es múltiplo de 5
```

Ejercicio 14

¿Qué pasa en el ejemplo anterior si eliminamos la redirección de la orden `if`?

Lo que ocurre al eliminar la redirección del if es que, en caso de error,
se mostrará tanto nuestro mensaje propio como el error de la propia orden rm.

Ejercicio 15

Haciendo uso del mecanismo de cauces y de la orden **echo , construya un guion que admita un
argumento y que informe si el argumento dado es una única letra, en mayúsculas o en minúsculas, o es algo
distinto de una única letra.**

```bash
#!/bin/bash
# Titulo:       unicaletra
# Fecha:        19/10/2017
# Autor:        Ricardo Ruiz
# Version:      1.0
```

```
# Descripción:  Informa si el argumento dado es una única letra
#               en mayúsculas o en minúsculas o es algo distinto
#               de una única letra
# Opciones: ninguna
# Uso: unicaletra <algo>

if  echo $1 | grep '^[a-Z]\{1\}$' 2> /dev/null ; then
    echo "Es una única letra";
else
    echo "Es algo distinto de una única letra";
fi 2> /dev/null
```

Ejercicio 16

**Haciendo uso de expresiones regulares, escriba una orden que permita buscar en el árbol de
directorios los nombres de los archivos que contengan al menos un dígito y la letra e. ¿Cómo sería la orden si
quisiéramos obtener los nombres de los archivos que tengan la letra e y no contengan ni el 0 ni el 1?**

Ejercicio 17

Utilizando la orden `grep` , exprese una forma alternativa de realizar lo mismo que con `wc -l`.