PRÁCTICA 4

Ejercicio 4.1: **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 $[…].**

Declaramos las variables:

~$ anyo=365
~$ dia=`date +%j`

Posteriormente, mediante la orden echo y $[...], mostramos el resultado por pantalla:

~$ echo "Faltan $[(anyo - dia)/7] semanas hasta el fin de año"
Faltan 11 semanas hasta el fin de año

Si hacemos uso de ((...)):

~$ echo "Faltan $(((anyo-dia)/7)) semanas hasta el fin de año"
Faltan 11 semanas hasta el fin de año

Usando expr:

~$ op=`expr $anyo - $dia`
~$ echo "Faltan `expr $op / 7` semanas hasta el fin de año"
Faltan 11 semanas hasta el fin de año

Combinandolas::

~$ echo "Faltan `expr $[$anyo - $dia] / 7`semanas hasta el fin de año"
Faltan 11 semanas hasta el fin de año

Ejercicio 4.2. **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

Declaramos la variable v, a la cual le asignamos el valor de 1.

$ echo $v
1

Mostramos por pantalla el valor de la variable v.

$ echo $((v++))
1

Hacemos primero lo que deseamos hacer con la variable (en este caso, mostrarla por pantalla) y posteriormente incrementamos en una unidad su valor.

$ echo $v
2

Mostramos por pantalla el valor de la variable, la cual ya ha visto incrementada su valor.

$ echo $((++v))
3

Primero incrementamos la variable y luego se hace lo que se desee con ella (mostrarla por pantalla).

$ echo $v.
3

Mostramos el valor de la variable por pantalla.

Ejercicio 4.3. **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**

Si asignamos los valores 30 y 6 a x e y respectivamente y realizamos la asinación abreviada:

~$ x=30
~$ y=6
~$ echo $[x /= y]
5
~$ echo $x
5

Obtenemos que x = 5.

Si asignamos los valores de 30 y 6 a x e y respectivamente y realizamos la asignación completa:

~$ x=30
~$ y=6

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

También obtenemos x = 5.

Queda así demostrado que ambas asignaciones son equivalentes.

Ejercicio 4.4. **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?**

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

Se muestra literalmente la expresión entrecomillada, pues con el uso de comillas dobles (acotación débil) esta orden queda protegida.

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

Se muestra literalmente la expresión entrecomillada, pues con el uso de comillas simples (acotación fuerte) esta orden queda protegida.

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

Se muestra el resultado, pues no se protege la orden que nos interesa, |bc -l.

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

Se muestra el resultado, pues no se protege la orden que nos interesa, |bc -l.

Ejercicio 4.5. **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?**

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

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

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

Encontramos la solución correcta. Ésta realiza la operación completa en el orden correcto antes de ejecutar bc -l.

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

Nos da la solución correcta pues realiza la operación completa en el orden correcto antes de ejecutar bc -l.

```
~$ 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'
```

Nos da error sintáctico porque lo entrecomillado no es un comando.

Ejercicio 4.6. **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
let: let arg [arg ...]
    Evalúa expresiones aritméticas.

    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.

        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

Se permiten las variables del intérprete como operandos. Se reemplaza el nombre de la variable por su valor (coercionado a un entero de longitud fija) dentro de una expresión. La variable no necesita tener activado su atributo integer para ser usada en una expresión.

Los operadores se evalúan en orden de prioridad. Primero se evalúan las sub-expresiones en paréntesis y pueden sobrepasar las reglas de prioridad anteriores.

Estado de salida:
Si el último ARGumento se evalúa como 0, let devuelve 1; de otra forma, let devuelve 0.

Ejercicio 4.7. **Con la orden let es posible realizar asignaciones múltiples y utilizar 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. Apóyese a través de la ayuda que ofrece help let.**

Ejemplo de asignación múltiple:

```
~$ let cinco=2+3 diez=5*2
~$ echo $cinco $diez
5 10
```

La siguiente lista de operadores está agrupada en niveles de operadores de la misma prioridad. Se muestran los niveles en orden de prioridad decreciente :

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

Primero se evalúan las sub-expresiones en paréntesis, las cuales pueden sobrepasar las reglas de prioridad anteriores.

Ejercicio 4.8. **Probad los siguientes ejemplos y escribir los resultados obtenidos con la evaluación de expresiones**

echo ejemplo1
ejemplo1
valor=6
if [$valor = 3]; then echo si; else echo no; fi
no
echo $valor
6

echo ejemplo2
ejemplo2
valor=5
if [$valor = 3] && ls; then echo si; else echo no; fi
no
echo $valor
5

echo ejemplo3
ejemplo3
valor=5
if [$valor = 5] && ls; then echo si; else echo no; fi
archivosP
'are taken to be the expression describing what is to be'
Descargas
'diagnostic information relating to the optimisa-'

Documentos
'earch for files in a directory hierarchy'
'e arguments are processed. Specifically, there are a number of'
ejercicio1
Ejercicio1.6
Escritorio
examples.desktop
FP
FS
Imágenes
'ions of Unix, file types are returned by readdir()'
'is the default optimisation level and corresponds to'
LMD
'manual page documents the GNU version of find. GNU find searches'
Música
'peed up execution while preserving the overall effect'
Plantillas
Público
'tats files during the processing of the command'
'ts based only on the names of files (for example'
Vídeos
si

echo $valor
5

echo ejemplo4
ejemplo4
valor=6
if ((valor==3)); then echo si; else echo no; fi
no
echo $valor
6

echo ejemplo5
ejemplo5
valor=5
if ((valor==3)) && ls; then echo si; else echo no; fi
no
echo $valor
5

echo ejemplo6
ejemplo6
valor=5
if ((valor==5)) && ls; then echo si; else echo no; fi

archivosP
'are taken to be the expression describing what is to be'
Descargas
'diagnostic information relating to the optimisa-'
Documentos
'earch for files in a directory hierarchy'
'e arguments are processed. Specifically, there are a number of'
ejercicio1
Ejercicio1.6
Escritorio
examples.desktop
FP
FS
Imágenes
'ions of Unix, file types are returned by readdir()'
'is the default optimisation level and corresponds to'
LMD
'manual page documents the GNU version of find. GNU find searches'
Música
'peed up execution while preserving the overall effect'
Plantillas
Público
'tats files during the processing of the command'
'ts based only on the names of files (for example'
Vídeos
si
echo $valor
5

echo ejemplo7
ejemplo7
echo $((3>5))
0
echo $?
0

echo ejemplo8
ejemplo8
((3>5))
echo $?
1

echo ejemplo9
ejemplo9
if ((3>5)); then echo 3 es mayor que 5; else echo 3 no es mayor que 5; fi
3 no es mayor que 5.

Ejercicio 4.9. **Haciendo uso de las órdenes conocidas hasta el momento, construya un guión 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.**

Realizamos el siguiente guión

```
#!/bin/bash

echo "Los valores con los cuales vamos a trabajar son $1 y $2";

echo "¿Es $1 = $2?";

if  (($1==$2));

        then
                echo  "Sí"

        else
                echo  "No"

fi


echo "¿Es $1 < $2?";

if  (($1<$2));

        then
                echo  "Sí"

        else
                echo "No"

fi


echo "¿Es $1 > $2?";

if  (($1>$2));

        then
                echo  "Sí"
```

```
        else
                echo  "No"

fi
```

Para ejecutarlo en la terminal primero debemos otorgarle permiso de ejecución mediante la siguiente orden :

```
~$ chmod -x ej9
```

Posteriormente, lo ejecutamos con los argumentos que deseemos :

```
~$ bash ej9 1 2
Los valores con los cuales vamos a trabajar son 1 y 2
¿Es 1 = 2?
No
¿Es 1 < 2?
Sí
¿Es 1 > 2?
No
```

Ejercicio 4.10. **Usando test, construya un guión 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 guión ejecutandolo con /bin/cat y también con /bin/rnano.**

Realizamos el siguiente guión:

```
#!/bin/bash

plano=`test -f $1 -a -x $1 && echo true || echo false`
simbolico=`test -h $1 && echo true || echo false`

echo "El archivo $1";

if (($plano=="true"));

        then
                echo "$1 es un archivo regular o plano y tiene permiso de ejecución sobre él."
```

```
        else
                echo "$1 no es un archivo plano y no tiene permiso de ejecución."

fi

if (($simbolico=="true"));

        then
                echo "$1 es un enlace simbólico."

        else
                echo "$1 no es un enlace simbólico."

fi
```

A continuación, en el terminal, usamos el siguiente comando para otorgarle permiso de ejecución:

```
~/Escritorio$ chmod -x Ej10
```

Para ejecutarlo, por ejemplo, con /bin/cat :

```
~/Escritorio$ bash Ej10 ./bin/cat
El archivo ./bin/cat
./bin/cat es un archivo regular o plano y tiene permiso de ejecución sobre él.
./bin/cat es un enlace simbólico.
```

Ejercicio 4.11. **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.**

```
~/Escritorio$ help test
test: test [expresión]
    Evaluate conditional expression.

    Exits with a status of 0 (true) or 1 (false) depending on
    the evaluation of EXPR.  Expressions may be unary or binary.  Unary
    expressions are often used to examine the status of a file.  There
    are string operators and numeric comparison operators as well.

    The behavior of test depends on the number of arguments.  Read the
    bash manual page for the complete specification.

    File operators:
```

-a FILE True if file exists.
-b FILE True if file is block special.
-c FILE True if file is character special.
-d FILE True if file is a directory.
-e FILE True if file exists.
-f FILE True if file exists and is a regular file.
-g FILE True if file is set-group-id.
-h FILE True if file is a symbolic link.
-L FILE True if file is a symbolic link.
-k FILE True if file has its `sticky' bit set.
-p FILE True if file is a named pipe.
-r FILE True if file is readable by you.
-s FILE True if file exists and is not empty.
-S FILE True if file is a socket.
-t FD True if FD is opened on a terminal.
-u FILE True if the file is set-user-id.
-w FILE True if the file is writable by you.
-x FILE True if the file is executable by you.
-O FILE True if the file is effectively owned by you.
-G FILE True if the file is effectively owned by your group.
-N FILE True if the file has been modified since it was last read.

FILE1 -nt FILE2 True if file1 is newer than file2 (according to
modification date).

FILE1 -ot FILE2 True if file1 is older than file2.

FILE1 -ef FILE2 True if file1 is a hard link to file2.

All file operators except -h and -L are acting on the target of a symbolic link, not on the symlink itself, if FILE is a symbolic link.

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.

Other operators:

-o OPTION True if the shell option OPTION is enabled.
-v VAR True if the shell variable VAR is set.
-R VAR True if the shell variable VAR is set and is a name reference.
! EXPR True if expr is false.
EXPR1 -a EXPR2 True if both expr1 AND expr2 are true.
EXPR1 -o EXPR2 True if either expr1 OR expr2 is true.

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

Arithmetic binary operators return true if ARG1 is equal, not-equal, less-than, less-than-or-equal, greater-than, or greater-than-or-equal than ARG2.

See the bash manual page bash(1) for the handling of parameters (i.e. missing parameters).

Exit Status:
Returns success if EXPR evaluates to true; fails if EXPR evaluates to false or an invalid argument is given.

- Ejemplo del uso de la orden test comparando dos expresiones aritméticas:

```
~$ test $[10-5] == $[20-15]
~$ echo $?
0                                   #Verdadero pues 5 = 5

~$ test $[10-5] == $[20+15]
~$ echo $?
1                                   #Falso pues 5 != 35
```

- Ejemplo del uso de la orden test comparando dos cadenas de caracteres:

```
~$ test hola = hola
~$ echo $?
0                                   #Verdadero

~$ test hola = adios
~$ echo $?
```

1 #Falso

Ejercicio 4.12. **Responda a los siguientes apartados:**

a) Razone qué hace la siguiente orden: if test -f ./sesion5.pdf ; then printf "El archivo ./sesion5.pdf existe\n"; fi

Primero comprueba mediante la orden test si "sesion5.pdf" es un archivo regular o plano y, posteriormente, si resulta cierto que cumple esta condición escribe por pantalla que el archivo "sesion5.pdf" existe.

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.)

if test -f ./sesion5.pdf ; then printf "El archivo ./sesion5.pdf existe\n"; else printf "El archivo sesion5.pdf no existe."; fi

Mediante la orden else conseguimos que si no se cumple la primera condición establecida se lleve a cabo la indicada dentro de esta.

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.

if test -f ./sesion5.pdf && test -d ./bin ; then printf "El archivo ./sesion5.pdf existe y el archivo ./bin es un directorio\n" ; elif ! test -f ./sesion5.pdf && test -d ./bin ; then printf "El archivo ./sesion5.pdf no existe y el archivo ./bin es un directorio\n" ; elif test -f ./sesion5.pdf && ! test -d ./bin ; then printf "El archivo ./sesion5.pdf existe y el archivo ./bin no es un directorio\n" ; else printf "El archivo ./sesion5.pdf no existe y el archivo ./bin no es un directorio\n" ; fi

d) Usando como base la solución del apartado anterior, construya un guión 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.

Realizamos el siguiente guión :

```
#!/bin/bash

if test -f ./$1 && test -d $2; then
```

```
        printf "El archivo ./$1 existe y el archivo $2 es un directorio\n"

elif ! test -f ./$1 && test -d $2 ; then
        printf "El archivo ./$1 no existe y el archivo $2 es un directorio\n"

elif test -f ./$1 && ! test -d $2 ; then
        printf "El archivo ./$1 existe y el archivo $2 no es un directorio\n"

else
        printf "El archivo ./$1 no existe y el archivo $2 no es un directorio\n"

fi
```

En la terminal ejecutamos la siguiente orden para otorgarle permisos de ejecución:

```
~$ chmod -x guionEj12
```

A continuación, ejecutamos el guión con los archivos del apartado anterior:

```
~$ bash guionEj12 sesion5.pdf ./bin
El archivo ./sesion5.pdf no existe y el archivo ./bin no es un directorio
```

Ejercicio 4.13. **Construya un guión 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.**

Creamos el siguiente guión:

```
#!/bin/bash

if test -o $1 ; then
        printf "Somos el propietario del archivo $1\n"

else
        printf "No somos el propietario del archivo $2\n"

fi

if test -r $1 ; then
        printf "Tenemos permiso de lectura sobre el archivo $1\n"

else
        printf "No tenemos permiso de lectura sobre el archivo $1\n"

fi
```

Para ejecutarlo en la terminal primero le damos permiso de ejecución mediante la siguiente orden :

```
~$ chmod -x ej13
```

A continuación, lo ejecutamos con el archivo que deseemos :

```
~$ bash ej13 ~/Escritorio/condicional
No somos el propietario del archivo
Tenemos permiso de lectura sobre el archivo /home/bece/Escritorio/condicional
```

Ejercicio 4.14. **Escriba un guión 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 guión anterior para que admita la opción -h de manera que, al ejecutarlo con esa opción, muestre información de para qué sirve el guión y cómo debe ejecutarse.**

Realizamos el siguiente guión:

```
#!/bin/bash

dias=$[365-`date +%j`]

if [[ $1 == "-h" ]]; then
        printf "Este guión calculará el número de días que faltan hasta fin de año y los mostrará por pantalla además de decir si dicho número de días es múltiplo de 5 o no.\n\n"

fi

if (( (dias%5) == 0 )); then
        printf "Quedan $dias hasta fin de año y dicho número es múltiplo de 5.\n"

else
        printf "Quedan $dias hasta fin de año y dicho número no es múltiplo de 5.\n"

fi
```

Para ejecutarlo en la terminal debemos de otorgarle antes permisos de ejecución mediante la orden:

```
~$ chmod -x ej14
```

A continuación, comprobamos de que funcione correctamente :

~$ bash ej14 -h
Este guión calculará el número de días que faltan hasta fin de año y los mostrará por pantalla además de decir si dicho número de días es múltiplo de 5 o no.

Quedan 67 hasta fin de año y dicho número no es múltiplo de 5.

Ejercicio 4.15. **¿Qué pasa en el ejemplo anterior si eliminamos la redirección de la orden if?**

Si se borra la redirección de la orden if se mostrará por pantalla el mensaje de error que emite el sistema operativo, sea el caso por ejemplo de que el archivo no exista, además de nuestro mensaje. Si no eliminamos el if no se mostrará por pantalla ya que se envía a /dev/null.

Ejercicio 4.16. **Haciendo uso del mecanismo de cauces y de la orden echo, construya un guión 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.**

Realizamos el siguiente guión:

```
#!/bin/bash

if [ `echo $1 | grep -e '^.$'` ]; then
        echo "Es un único carácter."

        if [[ $1 =~ [A-Z] ]]; then
                echo "Y es una letra mayúscula."

        elif [[ $1 =~ [a-z] ]]; then
                echo "Y es una letra minúscula."

        else
                echo "No es una letra, es otro carácter."

fi

else echo "No es un único carácter."

fi
```

Para ejecutarlo en la terminal primero debemos otorgarle permisos de ejecución mediante la orden:

~$chmod -x guionEj16

Comprobamos que funcione correctamente:

~$ bash guionEj16 a
Es un único carácter.
Y es una letra minúscula.

~$ bash guionEj16 A
Es un único carácter.
Y es una letra mayúscula.

~$ bash guionEj16 hola
No es un único carácter.

~$ bash guionEj16 8
Es un único carácter.
No es una letra, es otro carácter.

Ejercicio 4.17. **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?**

Para que contengan al menos un dígito y la letra e :

~$ find / -regex '.*\([0-9]+.*e\).*'

Para obtener los nombres de los archivos que tengan la letra e y no contengan ni al 0 ni al 1 :

~$ find / -regex '.*e.*' -and -not -regex '.*[01].*'

Ejercicio 4.18. **Utilizando la orden grep, exprese una forma alternativa de realizar lo mismo que con wc -l.**

Haciendo uso de grep -c obtenemos lo mismo que con wc -l.

s4_Complementos.txt

A continuación ejecute los siguientes ejemplos y anote una descripción del resultado que obtenga:

```
grep    .          p4_datos      #Busca todas las líneas que contengan cualquier cosa
grep    \.         p4_datos      #Igual que el anterior
grep  p.p.         p4_datos      #Busca las líneas que contengan palabras como papa, pepa..
grep  P.p.         p4_datos      #Busca las líneas que contengan palabras como Papa, Pepa...
grep  pi           p4_datos      #Busca las líneas que contengan pi
grep  pi\.         p4_datos      #Igual que el anterior
grep  'l?'         p4_datos      #Busca las líneas que contengan l?
grep  l?           p4_datos      #Igual que el anterior
grep  l\?          p4_datos      #Igual que el anterior
grep  “\(ll\)\?”   p4_datos      #Busca las líneas en las que aparezca el 11 o no, es decir, todas las líneas
grep  'pa\?'       p4_datos      #Busca las líneas que contienen una p y pueden tener o no una a
grep  'pa\?p'      p4_datos      #Igual que el anterior solo que puede o no tener una a y luego una p
grep  '\(pa\)+'    p4_datos      #Busca las líneas que contienen pa+
grep  pa*p         p4_datos      #Busca las líneas que contienen pa[lo que sea]p
grep  'pa\{2\}p'   p4_datos      #Busca las líneas que contienen a p y a a dos veces seguido de una p
grep  [1-6]        p4_datos      #Busca las líneas que contienen algún número del 1 al 6
grep  [6-0]        p4_datos      #Rango inválido
grep  [q-t]        p4_datos      #Busca las líneas que contienen una letra que va de la q a la t
grep  [Q-T]        p4_datos      #Igual que la anterior pero en mayúscula
grep  [q-sy]       p4_datos      #Busca las líneas que contienen una letra entre la q y la s, o aquellas que contienen una y
grep  [^QT]        p4_datos      #Busca las líneas que empiezan por Q o T
grep  ^P           p4_datos      #Busca las líneas que empiezan por P
grep  -v  ^[^p]    p4_datos      #Busca las líneas que empiezan por p
grep  ^[^p]        p4_datos      #Busca las líneas que no empiezan por p
grep  e$           p4_datos      #Busca las líneas que acaban en e
grep  'e\b'        p4_datos      #Busca las líneas que tienen una e al principio o final de una palabra
grep  “e\>”        p4_datos      #Busca las líneas que tienen una e al final de una palabra
grep  'e\B'        p4_datos      #Busca las líneas que tienen una e entre otros dos caracteres
grep  '\<e'        p4_datos      #Busca las líneas que tienen una e al principio de una palabra
grep  -E 'e$|s$'   p4_datos      #Busca las líneas que tienen alguna palabra que acaba en e o s
grep  -E 'e$ | s$'  p4_datos     # hay un hueco en blanco a cada lado del símbolo de cauce, ¿qué sucederá? Si hay espacios no hace nada porque buscas espacios literalmente
```

Realizado por Alejandro Becerra Burgos - 1º DGIIM Grupo 1 de FS