Tutor de PERL



Manual de PERL

Introducción

Perl *(Practical Extraction and Report Language) es un lenguaje de programación desarrollado por Larry Wall (lwall@netlabs.com) a partir otras herramientas de UNIX como son: ed,grep,awk,c-shell, para la administración de tareas propias de sistemas UNIX.*

No establece ninguna filosofía de programación concreta. No se puede decir que sea orientado a objetos, modular o estructurado aunque soporta directamente todos estos paradigmas y su punto fuerte son.las labores de procesamiento de textos y archivos.

No es ni un compilador ni un interprete, esta en un punto intermedio, cuando mandamos a ejecutar un programa en Perl, se compila el código fuente a un código intermedio en memoria que se optimiza como si fuésemos a elaborar un programa ejecutable pero es ejecutado por un motor, como si se tratase de un interprete

Variables y Datos

CAPITULO 1

En PERL no hay que declarar los tipos de las variables.

Van precedidas por unos caracteres especiales como: $,@,%, que ya indican su tipo.

Se distingue entre minúscula y mayúsculas, por tanto, $A y $a son distintas

 **Variables Escalares:**

*Comienzan con el signo **$**y contienen datos de tipo **escalar**. Por escalar entendemos valores numéricos y/o alfanuméricos. Para la asignación se usa el signo igual (=) como en la mayoría de los lenguajes. En el caso de ser un valor alfanumérico (texto) este se escribe entres comillas. Al final de la asignación se debe escribir ";".*

Ejemplos:

 # se asigna un valor número con decimales
***$edad**=1.5;*

 # se asigna un valor número con decimales. Incluso en notación científica.
***$edad**=1.5; **$notacion**=1.2E10;*

 # números octales y hexadecimales
***$Octal=**033;*
***$Hex=**0x19;*

Se pueden asignar valores entre variables:

***$animal1**="Gato"; **$animal2**="Perro";*
***$animal2**=**$animal1**;*
*quedaría así: **$animal2**="Gato"*

Interpolar variables dentro de otra:
***$animal**="Gato $color de $edad años";*
*Sería igual que **$animal**="Gato blanco y negro de 1.5 años"*

Asignar multiples valores a multiples variables:
***$A**="Hola"; **$B**=","; **$C**="PERL";*
*(**$a**,**$b**,**$c**)=(**$A**,**$B**,**$C**);*

Las variables pueden actuar como números o alfanuméricas, según el caso. El cambio se realiza en función del signo más (+) o el signo punto (.).

- *Texto a número (**+**):*

  ***$A**="12"; **$B**=3; o **$A**="12"; **$B**="3";*

$C=$A +$B sería igual a $C=15.

- *Número a texto (.):*

$A="12"; $B=3; o $A=12; $B=3;
$C=$A.$B sería $C="123";

Se puede hacer asignación multiple de variables:
$A*="Hola"; *$B=","; *$C*="PERL";
(**$a**,**$b**,**$c**)=(*$A*,*$B*,*$C*); quedaría así: *$a*="Hola"; *$b*=","; *$c*="PERL";

Variables de matriz: arreglos

Comienzan con el signo @ y contienen una lista de valores escalares (números y/o textos). Al ser una lista de valores para acceder a cada de uno estos se utiliza un número que indica su posición dentro de la lista. La numeración empieza en 0, y se coloca entre corchetes [ ]. A este número que indica la posición de un valor dentro de la lista del arreglo se denomina ***indice***

@animal=("gato","perro","vaca");
$animal[0] sería "gato"
$animal[1] sería "perro" y así hasta el último valor.
También podemos hacer referencia a varios valores así:
$animal[1..2] equivale a los valores de $animal[1] y $animal[2]
$animal[1,2] equivale a los valores de $animal[1] y $animal[2]

Como vemos para la asignación se usa el caracter @ y para acceder a un valor individualmente se utiliza el signo $, ya que en definitiva un arreglo está formado por distintas variables escalares.

Una vez asignados los valores en esa lista estos pueden ser cambiados asignando nuevos valores a las distintas variables que la forman.

$animal[0]="ratón". Ahora el arreglo quedaría así: @animal=("ratón","perro","vaca");

Igual que con las variables escalares los arreglos también aceptan la interpolación de otros arreglos.

@pajaro=("gorrión","paloma","halcón","golondrina");
@animal=("ratón","perro","vaca",@pajaro);

Ahora @animal valdría:
@animal=("ratón","perro","vaca","gorrión","paloma","halcón","golondrina");

Asociada a las matrices (arreglos), existen algunas ***funciones*** *que ayudan a poner o sacar elementos de la lista.*

Colocar nuevos elementos al final de la lista

```
push(@animal,"toro","oveja");
push(@animal,@pajaro);
```

Sacar el último elemento del arreglo. Se asigna a una variable escalar.

```
$sacado=pop(@animal);
```

Reemplazar el primer elemento del arreglo.

```
unshift(@animal,"caballo");
```

Sacar el primer elemento del arreglo. Se asigna a una variable escalar.

```
$sacado=shift(@animal);
```

Tenemos una notación especial para conocer **el último índice** del arreglo.

Del arreglo **@animal**=("ratón","perro","vaca","gorrión","paloma","halcón");
$#animal sería igual a **4.**

Variables matrices asociadas: "hashes"

Comienzan con el signo **%** y se tratan de matrices que se referencian por el par **clave/valor**. Como veremos a continuación los valores se introducen manteniendo una relación a pares. El primer valor es la clave o referencia del siguiente.

Se pueden asociar los valores a las matrices asociadas de 2 modos:

```
%dia=(Lun,"Lunes",Mar,"Martes",Mie,"Miércoles",Jue,"Jueves",Vie,"Viernes",Sab,"Sábado",Dom,"Domingo");
%dia=(
        Lun=> "Lunes",
        Mar=> "Martes",
        Mie=> "Miércoles",
        Jue=> "Jueves",
        Vie=> "Viernes",
        Sab=> "Sábado",
        Dom=> "Domingo"
       );
```

La referencia a los valores se realiza mediante la variable escalar del la matriz asociada y sus claves (Lun,Mar,Mie,...).

$dia{**Lun**}** equivale a "Lunes", **$dia{Sab**}** equivale "Sábado".

La asignación de valores individuales se realiza así:

***$dia{**Lun**}=**"LUNES" o así **$dia{"**Lun"**}=**"LUNES"**

*Si recorremos la matriz asociada mediante algún algoritmo, los pares de valores no se muestran en el mismo orden de introducción; **sólo** se mantiene el orden del par clave/valor.*

Al igual que con los arreglos, las matrices asociadas también tienen una serie de funciones asociadas, que facilitan su utilización:

***delete**($matriz**{**clave**}**): Para borrar un par clave/valor de un "hash", utilizamos la función delete que usa como argumento la variable escalar y la clave del par a borrar.*

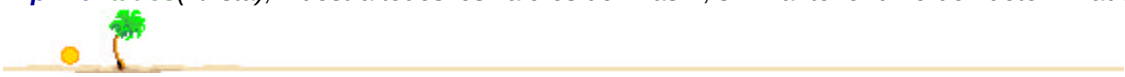
*Ej: **%lista**(uno,"1",dos,"2",tres,"3");*

***delete**($lista**{**tres**}**);*

***values**(Hash): Muestra todos los valores de la matriz asociada.*

*Ej: **%lista**(uno,"1",dos,"2",tres,"3");*

***print values**(%lista); muestra todos los valores del "hash", sin mantener un orden determinado.*



***keys**(Hash): Muestra las claves de la matriz asociada.*

*Ej: **%lista**(uno,"1",dos,"2",tres,"3");*

***print keys**(%lista); muestra todos las claves del "hash", sin mantener un orden determinado.*

***each**(Hash): Muestra un par clave/valor.*

*Ej:**%lista**(uno,"1",dos,"2",tres,"3");*

***print each**(%lista); muestra **sólo un par** clave/valor de la matriz asociada.*

***exists $lista{**clave**}**: idem.*

*Ej: **$existe**=**exists** $lista**{**dos**}**; Si existe, la variable $existe contendrá el valor **1**.*
*Ej: **$existe**=**defined** $lista**{**dos**}**; Si existe, la variable $existe contendrá el valor **1**.*

***defined $lista{**clave**}**: indica si existe o está definida una determinada clave dentro de la matriz asociada.*

*NOTA: Para mostrar ordenados los valores o claves se usa la función **sort**(...).*

*Ej: **print** **sort**(**keys**(%lista));*



Entrada/Salida

CAPITULO 2

Hasta ahora sólo hemos estudiado el tipo de variables. En este capítulo vamos a estudiar el modo de capturar/mostrar esos valores a través del teclado/pantalla.

La instrucción básica para mostrar el valor de las variables en pantalla es ***print****.*

Nuestro primer programa en PERL, podría ser este:

print *"Hola Perl";*

Su salida en pantalla sería: Hola Perl.

Existen muchos ***modificadores*** *para la salida de los valores de las variables. A continuación vamos a estudiar algunas de ellos.*

"\n" *Nueva línea.*
"\t" *Tabulacíon.*
"\r" *Retorno de carro.*
"\f" *Nueva hoja (formfeed).*
"\b" *Espacio atrás (backspace).*
"\e" *Secuencia de ESCape.*
"\u" *Pasa a mayúscula el primer caracter de texto siguiente".*
"\U" *Pasa a mayúscula todo el texto siguiente".*
"\l" *Pasa a minúscula el primer caracter de texto siguiente".*
"\L" *Pasa a minúscula todo el texto siguiente.*
"\E" *Fin del efecto de los modificadores* \U,\L.
"\a" *Emite un pitido.*
"\cC" *Combinación de Control+Letra. En este caso Control-C .*
xN *el signo por (x) seguido de un número N repite un caracter o texto anterior N veces.*

Algunos ejemplos del efecto de estos modificadores serían estos:

print *"Hola PERL \n"; Su salida es:* ***Hola PERL*** *seguido de nueva línea.*
print *"\UHola Perl"; Su salida es:* ***HOLA PERL****.*
print *"\lHola \LPERL \n"; Su salida es:* ***hola perl*** *seguido de nueva línea.*
print3*x4; Muestra:* ***3333****. NO CONFUNDIR x con el operador * (multiplicar).*
print *"Hola "x3. Muestra:****Hola Hola Hola Hola****.*

*Los **operadores aritméticos** son modificadores que sólo afectan a los valores númericos:*

***print 3 + 4**; Imprime el resultado de la **Suma** de 3 y 4.*
***print 4 - 3**; Imprime el resultado de la **Resta** de 4 y 3.*
***print 3 * 4;** Imprime el resultado de la **Multiplicación** de 3 por 4.*
***print 8 / 4**; Imprime el resultado de la **División** de 8 entre 4.*
***print 3 ** 3**; Imprime el resultado de la **Elevación** de 3 al cubo.*
***print 4 % 2**; Imprime el resultado del **Módulo** de 4.*

*Lo realmente particular del lenguaje PERL es el método que usa para la entrada de datos desde el teclado. Para asignar un valor desde el teclado a una variable, se asigna a la variable la representación del teclado **<STDIN>** (**ST**and**ar**D**IN**P**ut**).*
*Ej: $var=**<STDIN>** **.***

```
print "¿Cómo te llamas?: ";
$nombre=<STDIN>;
print "Tu nombre es:$nombre";
```

*Este pequeño programa pide un nombre de persona y su resultado final es: **Tu nombre es:** seguido de ese nombre.*

Existe una función asociada a la entrada de datos desde el teclado que suele usarse para eliminar el último caracter de la entrada, normalmente nueva línea.

***chop(...)**: elimina el último caracter de una entrada por teclado.Ej: **$nombre**=**chop**($nombre);*

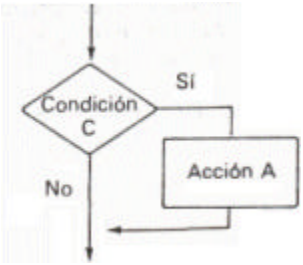
Estructuras de control

CAPITULO 3

La mayoría de las estructuras de control en PERL, han sido heredadas del lenguaje C/C++, por tanto, para aquellos que conozcan este lenguaje será fácil adaptarse.

Estructuras Condicionales Simples:

Las distintas intrucciones de un programa o algoritmo se ejecutan secuencialmente; pero en ocasiones es necesario bifurcar o desviar momentaneamente esa ejecución en función de nuestras necesidades. La estructura condicional más simple que permite esta bifurcación es el ***if...*** *(Si...) y permite realizar una acción o grupo de acciones en función de la condición.*




```
if (condición)
  {
     intrucciones...
  }
```

En el caso del ***if*** *en PERL existen 2 variantes en las expresiones que usan en las condiciones de comparación:*

1*.-* ***Comparación númerica****:*

- **==** *igual que...*
- **!=** *distinto de...*
- **<** *menor que...*
- **>** *mayor que...*
- **>=** *mayor o igual que...*
- **<=** *menor o igual que...*

```
$edad=22;

if ($edad == 22)
 {
   print "Tu  edad es: ",$edad;
 }
```

2.- Comparación alfanumérica, literal o de textos.

- **eq** *igual que...*
- **ne** *distinto de...*
- **lt** *menor que...*
- **gt** *mayor que...*
- **ge** *mayor o igual que...*
- **le** *menor o igual que...*

```
$saludo="hola";

if ($saludo eq "hola")
 {
  print $saludo;
 }
```

Existe otra estructura condicional simple propia del lenguaje PERL que es ***unless*** *(hacer a menos que...). Se usa cuando queremos que se ejecute una instrucción o grupo de instruciones en el caso de que una variable no esté definida o que la condición no sea verdad, es decir, equivaldría a* ***Si no existe... o Si no es...***

Si no existe:

```
unless ($saludo)
 {
  print "Hola";
 }
```

```
if ( ! $saludo)
 {
  print "Hola";
 }
```

Sólo se mostrará el saludo si la variable $saludo no está definida.

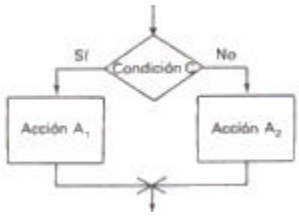
Si no es:

```
$saludo="Adiós";

unless ($saludo eq "Hola")
 {
  print "Hola";
 }
```

```
$saludo="Adiós";

if ($saludo ne "Hola")
 {
  print "Hola";
 }
```

Sólo se mostrará el saludo si la variable $saludo no es "Hola".

En el caso de que la condición de lugar a 2 posibilidades se pueden realizar 2 acciones o grupos de acciones. En este caso se utiliza ***if...else...*** *(Si...sino...)*




```
if (condición)
  {
    instrucciones...
  }
 else
  {
    instrucciones...
  }
```

Ejemplo:

```
print"¿Qué edad tienes?";$edad=<STDIN>;
chop($edad);

if ($edad >= 18 )
 {
   print "Eres mayor de edad\n";
 }
else
 {
   print "No eres mayor de edad\n";
 }
```

El programa pide la edad de una persona y le indica si es mayor de edad o no. Observamos que tras la entrada de la edad usamos la función ***chop****( ) para eliminar el retorno de carro de la edad; sino lo hacemos así el programa no funcionará, ya que, el valor de la edad terminaría con "\n" y ese caracter no lo comprobamos en las condiciones.*

Estructura Condicional Múltiple:

Hasta este momento hemos estudiado el caso de que una condición nos lleve a realizar una o dos acciones, pero se puede dar el caso de que la condición nos pueda ofrecer más de 2 resultados. En este caso el PERL dispone de una intrucción ideal para este fin, ***if...elsif...else...*** *(Si...sino-si...sino)*

```
print "¿A qué país pertenece la moneda: ";
$moneda=<STDIN>; chop($moneda);

if ($moneda  eq "peseta")
  {
   print "El país es  España" ;
  }
elsif ($moneda eq "dolar")
  {
  print "El país es  EEUU" ;
  }
elsif ($moneda eq "escudo")
 {
 print "El país es  Portugal";
 }
else
 {
 print "No conozco esa moneda";
 }
```

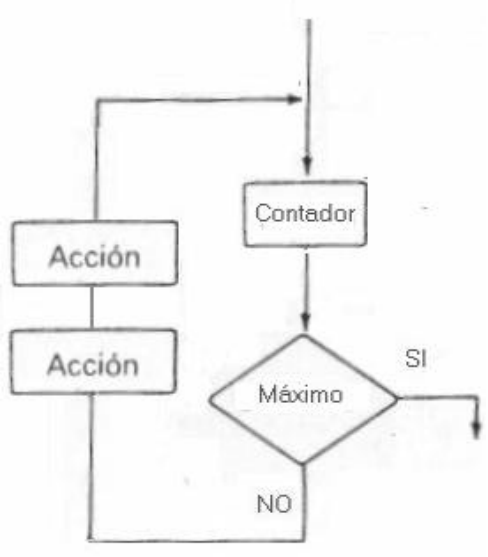
El programa es muy simple y pide el nombre de una moneda y muestra el país a que pertenece; siempre que sean las monedas de ciertos paises (España,EEUU y Portugal), en caso contrario muestra no conocer dicha moneda.

Estructuras Repetitivas *:*

Hasta este momento sólo hemos empleado estructuras que permiten decidir si se realiza una u otra acción o varias acciones en función de las distintas posibilidades que permite la condición. En el caso de que queramos ***repetir*** *una acción o grupo de acciones dependiendo de una condicion, podemos usar* ***for....****,****foreach...*** *y* ***while...*** *Todas estas instrucciones producen los denominados* ***bucles.***

for...(*para...*):

La instrucción ***for*** *es la estructura repetitiva más se usada para crear bucles. Se basa en la repetición N veces, definidas por el programador, de una serie de acciones o instrucciones. Cuando la condición alcanza el máximo de veces definido, concluye esta repetición.*



Los bucles de tipo ***Para*** *necesitan de unos valores numericos definidos que indican las veces que se repetirá un bloque de instrucciones.Entendemos por* ***bloque de instrucciones*** *a todas las acciones o intrucciones que se encuentren entre las llaves "***{ }***" que delimitan el ámbito de una estructura de control.*

El bucle se ejecutará N veces que indica una variable que actúa como contador, indicando el valor inicial, el final y el incremento de conteo. El incremento se hace mediante esta nomenclatura:

Incremento de 1 en 1

```
$var=$var+1;
$var++;
$var+=1;
```

Incremento de 2 en 2

```
$var=$var+2;
$var+=2;
```

y así sucesivamente para otros incrementos.

PERL	SIGNIFICADO

for ($i=1; $i<=10; $i++)
{
print "$i\n";
}
	Para (valor inicial; valor final; incremento)
	{
	Escribe "$i\n";
	}

Este simple programa muestra en pantalla una lista de números desde el 1 al 10.

En ocasiones nos puede interesar que el bucle termine antes del número de veces de repetición estimado o que al alcanzar cierto número de repeticiones se produzca una alteración del bucle...Para permitir estas alteraciones del flujo normal de las repeticiones, tenemos las instrucciones ***next*** *(siguiente)* y ***last*** *(último)*.

next *permite saltar la cuenta una vez en un punto determinado del proceso del bucle.*
last *concluye la cuenta y las repeticiones, terminando el bucle.*

```
for ($i=1; $i<=10; $i++)
{
  if ($i == 3)
   {
    next;
   }

  print "$i\n";

}
```

```
for ($i=1; $i<=10; $i++)
{
  if ($i == 3)
   {
    last;
   }

  print "$i\n";

}
```

El primer ejemplo muestra una lista de números del 1 al 10, saltando el número 3.
El segundo ejemplo muestra una lista de números del 1 al 3.

foreach...(*para cada...*):

Esta instrucción se usa para ***recorrer los valores de las matrices****. La expresión dentro del paréntesis (matriz) es evaluada para producir una lista con sus valores, que actúan como contadores del bucle, ejecutandose el bucle una vez por cada uno de estos elementos.*

En este ejemplo el bucle se ejecutará una vez por cada uno de los elementos de la matriz ***@nombres****. Se utiliza una variable contenedor* ***$persona*** *que contiene, en cada pasada, cada uno de los valores de la matriz.*

```
@nombres=("Juan","Antonio","Luis","Marcos");

foreach $persona (@nombres)
{
 print "$persona\n";
}
```

*En este caso se usan las claves (****keys****) y los valores (****values****) de una matriz asociada como contador del bucle y se asignan a variables* ***contenedores*** *como* ***$letra*** *y* ***$nombre*** *que podemos usar para mostrar las iniciales de cada una de las personas o los nombres.*

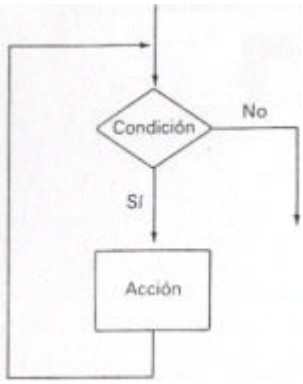
```
%nombres=("J","Juan","A","Antonio","L","Luis");

foreach $letra  (keys %nombres)
{
 print "$letra\n";
}
```

```
%nombres=("J","Juan","A","Antonio","L","Luis");

foreach $nombre  (values %nombres)
{
 print "$nombre\n";
}
```

while...(mientras que...)

*Como indica su nombre, un bucle se repetirá **mientras que** se cumpla o deje de cumplirse una condición. La nomenclatura de esta estructura repetitiva es:*

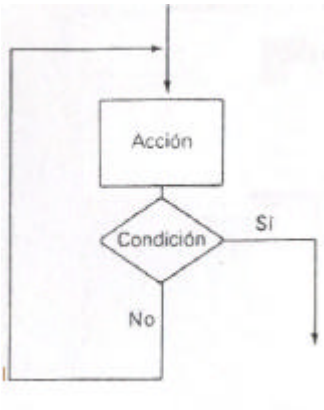
while *(**condición**)*
{
intrucciones...
}



*Ejemplo: Este programa mostraría el saludo de "**Hola**", mientras el contador no superase las 5 veces, pero hemos incluido una condición (**if** (**$n**==**3**)) para mostrar otro mensaje "**Adiós**" y salir (**exit**) al llegar a la **3** repetición.*

```
$n=1;
while ($n <= 5)
{
 print "Hola\n";

  if ($n==3)
    {
   print "Adiós";
   exit;
    }

 $n++;

}
```

do
{
intrucciones...
} ***until*** *(**condición**)*



*Ejemplo: Este programa mostraría el saludo de "**Hola**", hasta (**until**) que el contador no llegue a 3 veces.*

```
$n=0;
do
{
 print "Hola\n";
 $n++;

} until ($n == 3)
```



EXPRESIONES REGULARES
Y
PATRONES DE BUSQUEDA

CAPITULO 4

Una de las características más potente y usada en PERL es la manipulación de cadenas. Para ello existen una una gran variedad de ***expresiones y patrones*** *que nos facilitan esta tarea. Para estas expresiones se usan una serie de caracteres especiales que juntos o por separado actúan como patrones de busqueda, comparación o sustitución en una cadena. Estas expresiones se colocan entre 2 barras "/", así: /Hola/ y algunos caracteres con un significado especial que actual como patrones de busqueda, como:* ***^,*,$,?****,etc...*

Nota: Ya que "/" se usa como delimitador de expresiones regulares, si queremos buscar algún caracter como éste se escribe así: "\/", es decir, anteponiendo el signo de caracter de control "\".
Ej. Si buscamos una fecha de tipo 01/02/2000, el primer caracter "/" tras 01, podría entenderse como final de la expresión regular. Se escribiría así "01\/02\/2000".

BUSQUEDAS

Ej: Teniendo una cadena como "Hola Mundo", queremos comprobar si dentro de ella se encuentra la palabra "Hola".

```
$cadena="Hola Mundo";

if ($cadena =~ /Hola/)
   {
   print "Si existe";
  }
```

 # Podríamos traducirlo como "Si "Hola Mundo" contiene "Hola".

Comprobamos que aparece un signo especial ***=~*** *(contiene) para las* ***comparaciones*** *de expresiones. El signo* ***~*** *no aparece como una tecla normal en el teclado español. Se obtiene manteniendo pulsada la tecla* ***Alt*** *y pulsando* ***126*** *en el teclado numérico.*

La situación contraria sería:

```
$cadena="Hola Mundo";

if ($cadena !~ /Planeta/) {
   print "No existe";
  }
```

 # Podríamos traducirlo como Si "Hola Mundo" no contiene "Planeta".

Aparece otro signo especial ***!~*** *(no contiene) para las* ***comparaciones*** *de expresiones.*

En el ejemplo anterior la comparación es menos restrictiva, o sea, que buscamos la expresión en toda la cadena, sin tener en cuenta mayúsculas y minúsculas, si la expresión está al principio o final de la cadena, etc... Para realizar una comparación más exacta, usamos otros metacaracteres patrones.

Los ***metacaracteres*** *son una serie de caracteres que se usan con un significado especial y distinto del que representan. Así un asterisco " * " tiene un significado especial y definido por el programador. A continuación veremos los distintos significados que tienen algunos de estos signos.*

Ej: Buscamos la palabra "Hola" SOLO ***al principio*** *de la cadena (****^****).*

```
$cadena="Hola Mundo";

if ($cadena =~ /^Hola/)
 {
   print "Existe";
 }
```

 # Podríamos traducirlo como ***Si*** *"Hola Mundo"* ***contiene*** *"Hola"* ***al principio****.*

Ej: Buscamos la palabra "Mundo" SOLO ***al final*** *de la cadena (****$****).*

```
$cadena="Hola Mundo";

if ($cadena =~ /$Mundo/)
    {
    print "Existe";
   }
```

 # Se traduce como ***Si*** *"Hola Mundo"* ***contiene*** *"Mundo"* ***al final****.*

Una combinación de SOLO ambos metacaracteres ***^$*** *significa* ***línea en blanco****.*

```
$cadena="Hola Mundo";

if ($cadena =~ /^$/)
    {
    print '';
   }
```

 # Podríamos traducirlo como ***Si*** *"Hola Mundo"* ***contiene*** *ninguna palabra.*

Ej: Buscamos una palabra que empieza por "M" (********).

```
$cadena="Hola Mundo";
```

```
if ($cadena =~  /M*/)
 {
   print "Existe";
  }
```

 # Traducido como Si "Hola Mundo" contiene una palabra que empieza por "M".

Otra variante sería con el metacaracter **+**.

```
$cadena="Hola Mundo";

if ($cadena =~  /M+/)
   {
   print "Existe";
  }
```

 # Traducido como Si "Hola Mundo" contiene una palabra que empieza por "M".

Ej: Buscamos cualquier caracter (? o .) en una cadena.

```
$cadena="Hola Mundo";
if ($cadena =~  /M?ndo/)
 {
  print "Existe";
 }
```

 # Si "Hola Mundo" contiene palabras como "Mundo","Mendo",Mondo", etc....

Ej: Buscar una cadena dentro de otra o sola; usando \b y \B

```
$cadena="El oso es perezoso";

if ($cadena =~  /\boso/)
 {
  print "Palabra suelta: oso";
 }

if ($cadena =~  /\Boso/)
 {
  print "Forma parte de una palabra: perezoso ";
 }
```

 # Si "El oso perezoso" contiene la cadena "oso" como palabra suelta o formando otra palabra.

Ej: Podemos hacer una busqueda doble o alternativa (|), es decir, buscar una cadena u otra.

```
$cadena="El oso y el tigre están durmiendo";

if ($cadena =~  /oso|tigre/)
 {
  print "Existen";
 }

# Si  "El oso y el tigre están durmiendo" contiene la cadena "oso" o "tigre".
```

Ej: En ocasiones es preferible usar una variable implícita $_ para realizar las busquedas, de este modo no es necesario incluir la variable en la comparación.

```
$_="Hola Mundo";

if ( /Hola/)
 {
  print "Existe";
 }

# Si  "Hola Mundo" contiene la cadena "Hola". Como observamos en la comparación SOLO se incluye la expresión regular.
```

Hay otras combinaciones de caracteres que actúan como patrones. Se colocan entre corchetes.

[qjk] # Las letras **q** o **j** o **k**
[^qjk] # Ni **q** ni **j** ni **k.** En estos casos el signo ^ al principio indica NO.
[a-z] # Cualquier caracter desde **a** hasta **z** inclusive y en minúscula.
[^a-z] # Ninguna letra en minúscula.
[a-zA-Z] # Cualquier letra mayúscula o minúscula.

Recordemos los modificadores y caracteres de control que empiezan por "\".

"**\n**" Nueva línea.
"**\t**" Tabulacíon.
"**\s**" Espacio.
"**\S**" No espacio.
"**\r**" Retorno de carro.
"**\f**" Nueva hoja (formfeed).
"**\e**" Secuencia de ESCape.
"**\u**" Pasa a mayúscula el primer caracter de texto siguiente".
"**\U**" Pasa a mayúscula todo el texto siguiente".
"**\l**" Pasa a minúscula el primer caracter de texto siguiente".

"**\L**" Pasa a minúscula todo el texto siguiente.
"**\E**" Fin del efecto de los modificadores \U,\L.
"**\a**" Emite un pitido.
"**\cC**" Combinación de Control+Letra. En este caso Control-C .
xN el signo por (x) seguido de un número N repite un caracter o texto anterior N veces.

SUSTITUCIONES EN EXPRESIONES REGULARES

*Hasta ahora hemos usado las expresiones regulares SOLO para realizar busquedas, pero también se pueden usar para realizar sustituciones de cadenas. En las sustituciones anteponemos a los delimitadores de expresión regular "/../" la letra "**s**" de sustitución. Ej: **s**/Hola/Adiós/. Sustituimos la palabra "Hola" por "Adiós".Usando la variable implícita **$_** realizamos sustituciones rápidas.*

```
$_="Hola Mundo";

s /Hola/Adiós/;
print $_;

# Cambiamos en "Hola Mundo" la cadena "Hola" por "Adiós". La variable implícita $_ contendría "Adiós Mundo".
```

Reemplazamos todas las "o" por "a".

```
$_="Hola Mundo";

s /[o]/a/g;
print $_;

# Cambiamos en "Hola Mundo" todas las "o" por "a". La variable implícita $_ contendría "Adias Munda".
# En este caso incluimos otro modificador que es "g" que significa, globalmente.
```

Reemplazamos todas los espacios entre palabras por un guión "-".

```
$_="Hola Mundo";

s /\s/-/g;
print $_;

# Cambiamos en "Hola Mundo" todas los espacios por un guión "-". Quedaría así: "Hola Mundo".
```

Otros modificadores son:

i .- No tiene en cuenta mayúsculas y minúsculas.

```
$_="Oh, Hola Mundo";

s /o/a/gi;
```

```
print $_;

# Cambiamos en "Oh, Hola Mundo" todas las "o" por "a", incluso la que está en mayúscula.
# La variable implícita $_ contendría "ah, Hala Munda".
```

Existen una gran cantidad de posibilidades de patrones y modificadores por lo que no se pueden incluir en este tutor por su extensión. En el libro ***Programando PERL de la editorial O'Reilly****, así como en otros se pueden encontrar muchos ejemplos.*

FUNCIONES APLICADAS A EXPRESIONES REGULARES

Además de las expresiones regulares y los distintos patrones de busqueda, PERL incluye una serie de funciones especiales para el tratamiento de cadena.

 ***SPLIT.*** *Esta función se basa en la división o troceado de una cadena en distintos campos, usando como delimitador un caracter dado y que formará parte de una expresión regular de busqueda. La función devuelve una* ***matriz o array*** *con los distintos campos obtenidos.*

*Ej: Disponemos de un registro de personas, delimitados por el signo de 2 puntos "**:**". La función SPLIT nos va a dar el registro divido en campos, usando como patrón de busqueda el delimitador de los campos, es decir, los 2 puntos "**:**".*

```
$usuarios="Juan:Pedro José:Carlos:José María";
@personas=split(/:/,$usuarios);
foreach$nombre (@personas)
{
 print "$nombre\n";
}
```

*Si no se quiere cargar en la matriz obtenida del "troceado" de la cadena, los campos nulos, o sea, aquellos que están en blanco entre 2 delimitadores Ej: "Juan**::**Pedro:María". Usaremos esto:* ***split****(/**:+**/,**$usuarios**).*

 ***JOIN.*** *Esta función es contraria a la anterior y consigue componer un registro con distintos campos, usando como delimitador de cada campo un caracter dado. La función devuelve una cadena con campos delimitados.*

```
@personas=("Juan","Pedro José","Carlos","José María");
@registro=join(":",@personas);

foreach$cadena (@registro)
{
 print "$cadena";
}
```



EL MANEJO DE ARCHIVOS

CAPITULO 5

Hasta este momento no hemos realizado ningún programa que dejase grabado en disco o que leyese desde éste ningún dato. En este capítulo nos vamos a centrar en los modos de lectura y/o escritura de ficheros o archivos, es decir, del tratamiento de datos grabados en un soporte magnético, generalmente el disco duro.

CREACIÓN DE FICHEROS

Después de realizar cierto proceso con datos, tenemos que grabar estos para posteriormente usarlos. En este caso se nos presenta la necesidad de ***crear un archivo****.*

Ej: Imaginemos una serie de datos (personas y edades) y queremos almacernarlos en un soporte magnético como puede un disquete o un disco duro. Necesitamos grabarlos, ya que, será necesario actualizar estos datos cada año, al menos.

```
# Introducimos los datos.
@datos=("Juan",22,"Pedro",18,"Carlos",33,"Rosa",31,"Isabel",25);

# Delimitamos los datos con comas.
@registro=join(",",@datos);

# Damos nombre logico EMPLEADOS y físico "empleados.txt" al archivo.
# Como vamos a crear el archivo por primera vez, usamos el signo de ">"
open (EMPLEADOS,">empleados.txt");

#Tomamos cada campo, uno por uno, y los grabamos en el archivo.
foreach $campo (@registro)
{
 print EMPLEADOS $campo;
}

# Fin de línea y nueva línea para un posible nuevo registro.
 print EMPLEADOS "\n";

# Cerramos el fichero abierto
close (EMPLEADOS);
```

Entrada directa desde el teclado **(I)**.*Si la entrada de datos hubiese sido desde la consola , la matriz* ***@datos*** *se igualaría a la entrada genérica en PERL, es decir,* ***<STDIN>****.Ej:* ***@datos****=****<STDIN>****;*

```
# Indicamos cual será el caracter de fin de archivo
print "Al terminar de introducir los datos pulsar:\n";
print "Ctrl+D en Unix\n";
print "Ctrl+Z en MS-DOS\n";
```

```
print "--------------------------\n";

# Los datos introducidos mediante el teclado formarán una matriz
@datos=<STDIN>;

# Creamos el archivo.
open (EMPLEADOS,">empleado.txt");

# Grabamos el archivo con los datos
foreach $dato (@datos)
{
 print EMPLEADOS $dato;
}
# Cerramos el fichero abierto
close (EMPLEADOS);
```

Entrada directa desde el teclado (II).*Otra variante de la entrada desde el teclado sería esta:*

```
# Indicamos cual será el caracter de fin de archivo
print "Al terminar de introducir los datos pulsar:\n";
print "Ctrl+D en Unix\n";
print "Ctrl+Z en MS-DOS\n";
print "--------------------------\n";

# Abrimos un fichero que usa como entrada el teclado.
# Usamos un signo de guión "-" como representación del teclado
open (TECLADO,"-");

# Creamos el archivo.
open (EMPLEADOS,">empleado.txt");

# Los datos introducidos desde el teclado formarán una matriz
@datos=<TECLADO>;

# Grabamos el archivo con los datos
foreach $dato (@datos)
{
 print EMPLEADOS $dato;
}
# Cerramos los archivos abierto
close (TECLADO);
close (EMPLEADOS);
```

*Si una vez creado el archivo, **añadimos** nuevos datos, SÓLO hay que modificar esta línea en el programa anterior:*

```
# Añadimos nuevos datos al fichero ya creado.
open (EMPLEADOS,">>empleado.txt");
```

*Comprobamos que incluimos un signo ">" más a la hora de hacer referencia al archivo "**empleado.txt**"*

Metacaracteres	Significado
>	Nuevo...
>>	Añadir a...

LECTURA DE FICHEROS

Una vez que ya tenemos los datos grabados, será necesario en más de una ocasión su actualización, por lo que tendremos que leerlos.

```
# Podemos usar el signo "<" o ninguno delante del fichero, para lectura
open (EMPLEADOS,"empleados.txt");

#Añadimos cada línea de éste en la matriz.
@registros=<EMPLEADOS>;

# Mostramos los datos en pantalla
foreach $empleados (@registros)
{
 print $empleados;
}
# Cerramos el fichero abierto
close (EMPLEADOS);
```

La salida del programa sería todas la líneas o registros del archivo.

Leer un fichero pasado como parámetro desde la línea de comandos.

Ej: ***Perl leer.pl empleados.txt***

*En estos casos dentro de la codificación de un programa en PERL, se utiliza "***<>***" como referencia a nuestro archivo ,****empleados.txt****, que se escribe desde la línea de comandos.*

```
# La matriz recoge las líneas leidas, del archivo, desde la línea de comandos
@lineas=<>;

# Mostramos los registros en pantalla.
foreach $registro (@lineas)
{
print $registro;
}
```

SUBRUTINAS

CAPITULO 6

Como cualquier otro lenguaje, PERL permite al programador definir sus propias funciones llamadas ***subrutinas****. Se pueden colocar en cualquier lugar dentro de un programa, pero es aconsejable colocarlas todas al principio o final. Para llamar a un subrutina usamos el signo* ***&****.*

```
# Inicio de programa con subrutina.
# Declaración de la subrutina.
sub saludo
{
print "Hola Mundo\n";
}

# Llamada a la subrutina.
&saludo;
```

PARÁMETROS

A nuestras subrutinas podemos pasar distintos parámetros de modo que ***todos ellos forman parte de una matriz especial*** *representada como* @_ *(signos de arroba y subrayado). Si recordarmos las matrices o arreglos veremos que cada uno de los parámetros será una variable dentro de la matriz que se direcciona así* **$_[0]**, **$_[1]**,*etc...*

Ejemplo:

```
# Inicio del programa

# Declaración de la subrutina.
sub saludo
{
print "@_\n";
}
# Equivale a @_=("Hola","Mundo\n")
#$_[0]="Hola"  y  $_[1]="Mundo\n"

# Llamada a la subrutina con 2 parámetros.
&saludo ("Hola","Mundo\n");
```

La salida del programa será: ***Hola Mundo***

Otra variante del programa anterior sería:

```
# Inicio del programa
# Declaración de la subrutina.
```

```
sub saludo
{
print "$_[0] $_[1]";
}
#$_[0]="Hola"  y  $_[1]="Mundo\n"

# Llamada a la subrutina con 2 parámetros.
&saludo ("Hola","Mundo\n");
```

La salida del programa será también: ***Hola Mundo***

RETORNO DE VALORES

Como cualquier función de cualquier otro lenguaje, PERL también nos permite retornar valores y lo hace ***tomando como retorno el último valor escalar que se emplee*** *en la subrutina.*

Ejemplo: Subrutina de suma de 2 números

```
# Inicio del programa
# Declaración de la subrutina.
sub Suma
{
$Total=$_[0] + $_[1]; #$_[0]=2  y  $_[1]=3
}

# Llamada a la subrutina con 2 parámetros.
print &Suma (2,3);
```

La salida del programa será también: ***5***

ATENCIÓN*:* ***si tras la variable $Total tuviesemos otra variable,el valor final retornado sería el valor de esta última variable.***

Ejemplo:

```
# Inicio del programa
# Declaración de la subrutina.
sub Suma
{
$Total=$_[0] + $_[1]; #$_[0]=2  y  $_[1]=3
$saludo = "Hola Mundo\n";
}

# Llamada a la subrutina con 2 parámetros.
print &Suma (2,3);
```

La salida programa sería,: ***Hola Mundo***