INTRODUCCIÓN A PERL Y CPAN

Mariam Amrhar Martín

Department of Computer Science and Automation, University of Salamanca

Plaza de la Merced, s/n, 37008, Salamanca, Spain

mariam\_amartn@usal.es

# 1. Introduction

* Las siglas de PERL significan “*Practical Extracting and Reporting Language”* que en español lo podremos traducir como “*Lenguaje práctico para la extracción e informe”* y por estas siglas, podemos deducir que es trata de un lenguaje de programación práctico para poder extraer información de archivos de texto y generar informes a partir del contenido de los ficheros.
* El uso de Perl se centra principalmente en la creación de aplicaciones CGI para entornos web.
* Perl es software libre y está licenciado bajo la Licencia Artística y la GNU.
* Existen distribuciones disponibles para la mayoría de sistemas operativos.
* Las extensiones que podremos encontrar son *pl, pm y cgi*.
* Ha influido a otros lenguajes de programación como Python, PHP, Ruby o JavaScript.
* En un principio, el lenguaje Perl sólo funcionaba en entornos UNIX pero actualmente lo podremos encontrar en otras plataformas.
* El lenguaje Perl se simboliza por un dromedario, como podemos observar en el logo, sin embargo, en la actualidad para evitar problemas con la licencia comercial con el autor del logo, podemos encontrar nuevos logotipos basados en una cebolla debido a que al nombre que reciben las conferencias anuales que ofrece Larry Wall, el creador de Perl, que *es State of the Onion*, en español, estado de la cebolla cuya inspiración radica en *State of the Nation* el nombre del evento donde el presidente de EEUU informa del estado del país a los ciudades.
* Como dato curioso, la comunidad Perl está formada por grupos que a su vez conforman los *Perl Mongers.*

# 2. Evolución histórica

El lenguaje de programación Perl fue desarrollado por Larry Wall en 1987 mientras trabajaba en una empresa llamada *Unisys* como programador. La primera versión, la versión 1.0, se lanzó el 18 de diciembre de 1987.

**Figura 1** Larry Wall

El nombre original de “Perl” era “Pearl” ya que Wall quería un nombre corto y con connotaciones positivas (aunque también quería llamarlo Gloria, como su esposa), sin embargo, ya existía un lenguaje llamado así, entonces tuvo que cambiar el nombre al que conocemos actualmente.

En los siguientes años, el uso del lenguaje Perl se extendió y en 1988 se publicó Perl 2, dónde encontramos un motor mejor en el procesado de expresiones regulares. Al siguiente año, se publicó Perl, dónde se añadió un soporte para datos binarios

La documentación de Perl era una página del manual de Unix, hasta 1991 con la publicación de *Programming Perl* o también conocido como el libro del camello y se convirtió en la referencia más destacada del lenguaje. Paralelamente, salió al mercado Perl 4 y esta versión no supuso un gran cambio en el lenguaje, sino que esta versión es aquella documentada en el libro, aunque, también podremos añadir sucesivos lanzamientos de mantenimiento, culminando en Perl 4.036, lanzada al mercado en 1993. En este punto, Larry Wall abandonó Perl 4 para empezar a desarrollar Perl 5.

El 17 de Octubre de 1997 salió al aire Perl 5, siendo una reescritura del interprete, donde se añadieron nuevas características al lenguaje como paquetes o módulos. Esto permitó a los programadores añadir nuevas características. Aproximadamente el año después se creó lo que conocemos como CPAN (*Comprehesive* *Perl* *Archive* *Network***). CPAN es una colección de sitios web que almacenan y distribuyen fuentes en Perl, documentación, scripts y módulos.** En su origen, para acceder a cualquier sitio CPAN se debía usar su propia URL, pero en la actualidad, se redirecciona automáticamente al sitio donde deseamos acceder.

En 2008, Perl 5 continúa siendo mantenido y se han añadido características importantes y algunas construcciones esenciales como soporte Unicode hilos o un soporte para la programación orientada a objetos.

La última versión estable de Perl se lanzó en 29 de Noviembre del año pasado y es la versión 5.28, caracterizada por centrarse en la optimización, en mejorar la ejecución, (donde podremos ver la eliminación de los atributos :*locked* y :*unique*), solucionar problemas de seguridad o actualizaciones en los módulos.

Finalmente, hace más de un mes se ha desarrollado una última versión que es la 5.29.7, pero que actualmente se encuentra en pruebas.

Finalmente, encontramos Perl 6, del cual salió al aire a finales del año pasado y la cual refina las características vistas en Perl 5.

# 3. Lenguaje Perl

## 3.1 Características del lenguaje

* Perl es un lenguaje de programación, pero no como los habituales que todos conocemos; Java, C …, sino que, se trata de un lenguaje de script. Se caracteriza por ser un **lenguaje multipropósito**, ya que con éste podemos hacer desde scripts muy simples hasta administrar servidores basados en UNIX o GNU/Linux.
* El lenguaje Perl es un lenguaje imperativo con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.
* Perl combina varias características de los lenguajes más usados por los programadores donde podremos encontrar, C, la Shell de UNIX (.sh), *awk* para el formateo e incluso Pascal (en el tratamiento de ficheros).
* Otra característica a destacar sobre el lenguaje Perl, es que es un **lenguaje de programación interpretado** como *Javascript*, esto significa que el código no se compila sino que cuando queremos ejecutarlo, leemos el código y lo interpretamos.
* Es un lenguaje fácil de usar, excelente para el tratamiento de cadenas y que no tiene fronteras.
* Es un lenguaje extensible a partir de otros lenguajes, puesto que podremos hacer llamadas a subprogramas escritos en otros lenguajes y podremos ejecutar código Perl desde otros lenguajes*.*
* Las prestaciones de Perl son similares a las de otros lenguajes como Python, PHP o Ruby, sin embargo, es más lento que la mayoría de los lenguajes compilados (ya que como vimos anteriormente, Perl es un lenguaje de programación interpretado).
* Con el lenguaje Perl, podremos programar prácticamente cualquier cosa, pero tenemos que tener en cuenta que habrá programas que requieran mucha rapidez y será más conveniente utilizar otro tipo de lenguajes, los que no sean interpretados, ya que como hemos dicho anteriormente, Perl es un lenguaje más lento.
* Perl es un lenguaje muy utilizado para la creación de aplicaciones CGI para entornos web.
* El mantenimiento y la depuración de un programa en lenguaje PERL es más sencillo que en cualquier programa en lenguaje C.
* Junto al interprete, podremos encontrar un gran conjunto de librerías y módulos.
* Se destacará el uso de variables especiales, comunes en los lenguajes de tipo script en los sistemas de tipo Unix y éstas permiten la realización de una serie de operaciones sobre los datos y los archivos que aportando gran potencia y flexibilidad.

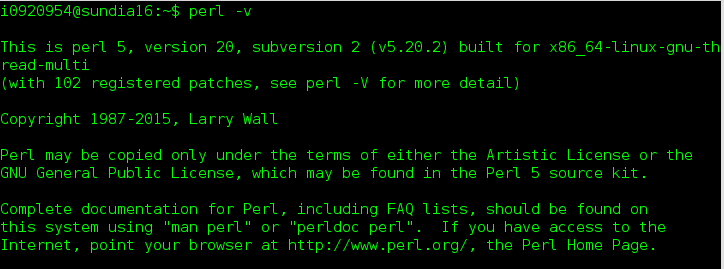
## 3.2 Intérprete de Perl

La misión del interprete es convertir en tiempo real el código Perl en un ejecutable, pasando por una traducción transitoria a código C, que también es compilado en tiempo real y éste posee ciertas opciones que nos proporcionan información sobre el mismo o facilidades en la ejecución de los *scripts*.

Lo dicho anteriormente, lo llevaremos a la práctica desde la terminal.

A continuación, procedemos a mostrar una serie de ejemplos de comandos:

* perl -v : nos muestra la versión del interprete de Perl que estamos utilizando y nos muestra la información sobre la configuración del intérprete de perl



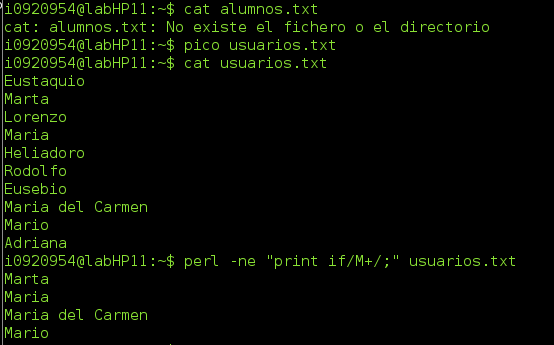
**Figura** 2 Información sobre el intérprete de Perl

* perl -e expresión: ejecutará la expresión como su fuera un ejemplo, esto lo veremos con el siguiente ejemplo:

perl -e “print ‘holita :D’;”

**Figura 3** resultado del comando realizado en la ventana de comandos

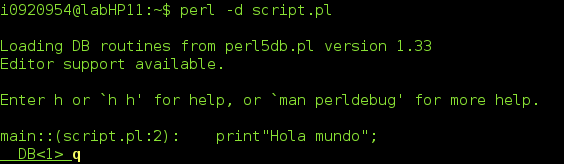
* perl -ne expresión: ejecutará la expresión como si fuera un bucle repetitivo, esto lo vemos de una manera más rápida con un ejemplo :
  + perl -ne “print if/M+/;”usuarios.txt



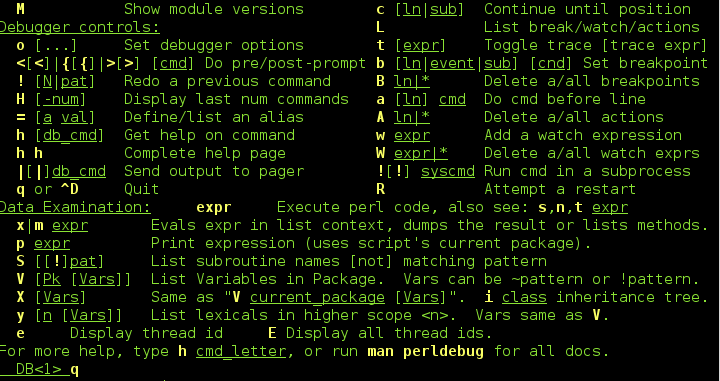
**Figura 4** resultado del comando “perl -ne “print if/M+/;”usuarios.txt”

Como Podemos observar, utilizando este comando imprimiremos por pantalla los usuarios que comiencen por ‘M’

* perl -d script: ejecutará el script bajo el depurador



**Figura 5** ejecutamos el script bajo el depurador



**Figura 6** la guía que encontramos cuando ejecutamos h bajo el depurador

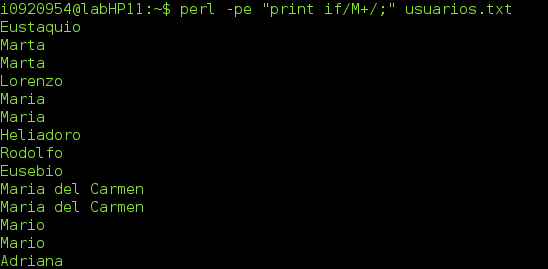
* perl -w script: nos dará *warnings* sobre las construcciones con errores



**Figura 7** resultado obtenido en la ventana de control de la ejecución

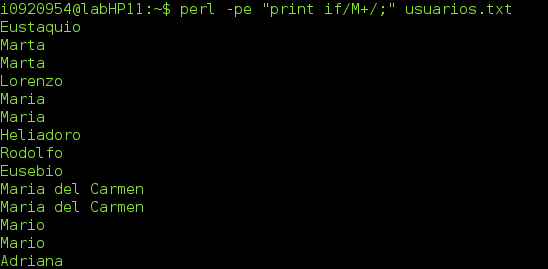
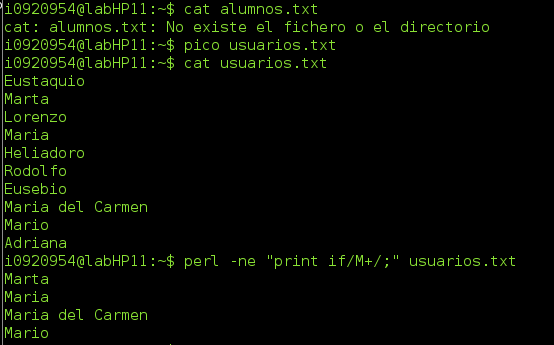
En este caso no tenemos *warnings* sobre las construcciones con errores puesto que este script consta de un print

* perl -pe expresión: ejecutará la expresión como si fuera un bucle repetitivo e imprime cada línea, lo veremos con el siguiente ejemplo
  + perl -pe “printf if/M+/;”usuarios.txt



**Figura 8** resultado de la ejecución del comando

* + A diferencia del comando perl -ne expresión mencionado anteriormente, es que este comando es que, en el mismo ejemplo puesto en *perl -ne expresión*, se imprimirán todos los nombres salvo los que comienzan con M que se imprimirán 2 veces y lo podemos ver en la comparación de estas 2 imágenes que son el resultado de lo que obtenemos en cada uno de los 2 comandos mencionados anteriormente.

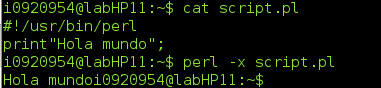


**Figura 10** resultado del comando “perl -ne “print if/M+/;”usuarios.txt”

**Figura 9** resultado del comando anterior

* perl -x script: se empieza a interpretar el fichero que contiene el script cuando encuentra la referencia al interprete, como vemos en el siguiente ejemplo

#!/usr/bin/perl



**Figura 11** resultado del comando anteriormente

* perl -i archivo: nos permite modificar los archivos, como veremos en el siguiente ejemplo
* Para poder ejecutar un programa tendremos que seguir los siguientes pasos:
  + Tenemos que navegar hacia el directorio donde está el script a ejecutar
  + Escribimos el comando perl nombreArchivo.pl para ejecutar el script perl

## 3.3 Tipos de datos

Perl permite la representación de los tipos de datos que pasaremos a ver más a fondo, a continuación:

### 3.3.1 Los tipos básicos de datos

3.3.1.1 Los tipos numéricos

Los valores numéricos expresados literalmente se presentan como valores reales codificaos en doble precisión.

Debemos tener en cuenta que las variables que contengan un $ representarán un escalar, que veremos que es más a fondo.

En la siguiente tabla veremos una serie de ejemplos.

|  |  |
| --- | --- |
| $a = 0.582 | Real |
| $b= 1.23e-78 | Real |
| $c= 7412 | Entero |
| $i= -85 | Entero |

Sin embargo, debemos tener en cuenta que los números no pueden comenzar por 0 porque que empiece por 0 nos permite especificar la codificación octal o hexadecimal.

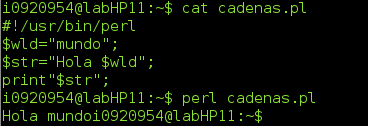
Que sea un tipo de codificación u otra dependerá de lo siguiente:

|  |  |
| --- | --- |
| Comienza por 0 | Codificación octal |
| Comienza por 0x/0X | Codificación hexadecimal |

3.3.1.2 Las cadenas de caracteres

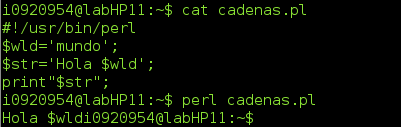
Las cadenas de caracteres se especifican por medio de una sucesión de caracteres limitados por unas comillas dobles (“ ”) o también por apostrofes. Perl distinguirá entre estos dos tipos de interpretaciones.

Cuando están delimitadas por comillas sucederá que toda variable referenciada en el interior se evalúa y se reemplaza por su valor, veámoslo con un ejemplo



**Figura 12** resultado de la declaración de las cadenas con dobles comillas

Sin embargo, cuando están delimitadas por apóstrofes, las cadenas se quedan intactas



**Figura 13** resultado de la declaración de las cadenas con comillas simples

A parte de estos tipos de delimitación mencionados anteriormente, también podremos delimitar una cadena de caracteres utilizando varias líneas, comillas o apóstrofes.

$str= <<etiqueta

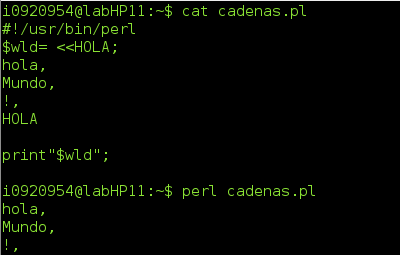
…

etiqueta

Cadena de caracteres cualquiera

Figura 2 Sintaxis del tercer tipo

Vamos a ver todo esto con un ejemplo



**Figura 14** resultado de la declaración de cadenas con varias líneas

3.3.1.3 El tipo booleano

El tipo booleano será similar al que encontramos en C, ya que se cumplirá que cuando un número es cero es falso y en cualquier otro caso es verdadero.

Tenemos que tener en cuenta que el cero está asociado a una ristra vacía, por lo tanto, la ristra vacía equivale a falso.

### 3.3.2 Las representaciones de datos

El lenguaje Perl posee tres tipos de representaciones de datos que veremos a continuación. Estas representaciones permiten la asociación de cada variable a cada tipo para poder ser utilizada.

La asignación de las variables se hace de manera dinámica y les asignamos un valor determinado en función del contexto.

3.3.2.1 Los escalares

El escalar permite la representación del tipo básico en Perl, donde representamos enteros, reales y cadenas de caracteres.

Como hemos mencionado anteriormente, las variables de tipo escalar van precedidas del carácter ‘$’.

Veamos algunos ejemplos:

$entero=-98;

$real=2.15423;

$natural=5;

$cadena=”Hola”;

$booleano=””

La asignación en las variables es dinámica, como hemos dicho anteriormente, y en un contexto numérico, el valor predeterminado es el 0 y en uno de una cadena de caracteres es una cadena vacía, “”.

En Perl, las variables no se tienen porque declarar, pero si las declaremos tendremos que usar my.

Como hemos mencionado anteriormente, la asignación es dinámica y por lo tanto, el tipo y valor de las variables en Perl se determinan a partir del contexto.

Caracteres especiales que podemos encontrar en Perl:

* \n (retorno de carro a una nueva línea)
* \b (retorno a la posición anterior)
* \a (pitido)
* \t (tabulación, que son 8 espacios)
* \\ (el carácter \)
* \”(las comillas dobles)
* \’ (las comillas simples)

Operaciones de datos escalares

El operador de asignación es el mismo que el que encontramos en C, es decir, el operador de asignación se representa por ‘=’.

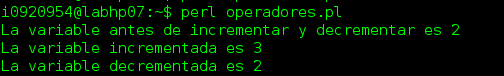
Los operadores de comparación, al igual que el operador de asignación, son similares a los que encontramos en C y varían dependiendo si estamos comparando números o cadenas. Estos operadores son:

|  |  |  |
| --- | --- | --- |
| **Comparación** | **Números** | **Cadenas** |
| Igual | = | eq |
| No igual | != | ne |
| Menor que | < | lt |
| Mayor que | > | gt |
| Menor o igual que | <= | le |
| Mayor o igual que | >= | ge |

En Perl, también podremos encontrar operadores de autoincremento (++) y autodecremento (--) , al igual que en C y a parte de estos dos, también podremos encontrar el operador potencia (\*\*)

Vamos a ver el funcionamiento de estos operadores con un ejemplo práctico.

1. #!/usr/bin/perl
2. $variable=2;
3. **print** "La variable antes de incrementar y decrementar es $variable\n";
4. variable++;
5. **print** "La variable incrementada es $variable\n";
6. variable--;
7. **print** "La variable decrementada es $variable\n";

Obtendremos el siguiente resultado:

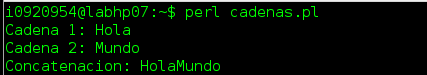
**Figura 15** resultado de la ejecución del script

Finalmente, para concatenar cadenas utilizaremos el operador punto “.” y para quitar el ultimo carácter a una cadena utilizaremos la función chop

Veámoslo con un ejemplo práctico

1. #!/usr/bin/perl
2. $cadena1="Hola";
3. **print**"Cadena 1: $cadena1\n";
4. $cadena2="Mundo";
5. **print**"Cadena 2: $cadena2\n";
6. $cadenafinal=$cadena1.$cadena2;
7. **print**"Concatenacion: $cadenafinal\n";

Obtendremos el siguiente resultado:



**Figura 16** resultado de la ejecución del script

3.3.2.2 Los arrays

Los arrays son listas de datos de tipo escalar y cada elemento de la lista es una variable escalar a la que le asociamos un determinado valor.

Las variables de tipo array se identifican con el prefijo ‘@’

Veamos algunos ejemplos:

@numeros= (23,17,20,9,12);

@animales=(“conejo”, ”gato”, ”perro”);

@mezcla=(7,”silla”,16, ”conejo”);

También se pueden definir los elementos de un array a partir de los valores de sus extremos como vemos en estos ejemplos:

@serienumeros= (0 .. 9);

@abecedario(a ..z);

Al igual que en C o en Java, los elementos de un array se referencian mediante índices y al igual que estos, el primer elemento se referencia por el índice 0.

$numero[1]; 🡪 vale 17

$animales[0]; 🡪 vale “conejo”

$mezcla[2]; 🡪 vale 16

Podemos acceder a un array utilizando un operador que define un conjunto de elementos del vector al que deseamos acceder con sus índices. Vamos a verlo con una serie de ejemplos:

@num=@numeros[0..2];🡪 @num= (23,17,20)

@cad=@animales[0,2];🡪@cad=(“consejo”,”gato”)

($cadena,$digito)=@mezcla;🡪$cadena=7,

$digito=”silla” (asignamos los 2 primeros valores del vector a las variables)

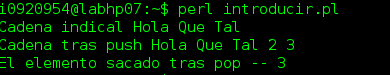
Como podemos observar, el tercer vector tendrá los elementos de los primeros vectores al hacer que el tercer vector tenga los primeros igualados. Algo que tenemos que tener en cuenta es que en Perl no ocurre que cuando incluimos un array dentro de otro se puede ver como un array de arrays, si no que, Perl introduce los elementos del array en la posición indicada.

Para sacar o introducir elementos podremos utilizar las funciones: pop & push, y shift & unshift. Con las funciones pop & push, sacaremos o introduciremos un elemento al final del array, como si fuera una pila y con las funciones shift & unshift, sacaremos o introduciremos un elemento al principal del array.

Veámoslo con un ejemplo práctico

1. #!/usr/bin/perl
2. @cadena=("Hola","Que","Tal");
3. **print**"Cadena inicial @cadena\n";
4. push(@cadena,2,3);
5. **print**"Cadena tras push @cadena\n";
6. $temp=pop(@cadena);
7. **print**"El elemento sacado tras pop -- $temp\n";

El resultado obtenido será el siguiente:



**Figura 17** resultado de la ejecución del script

Para poder ver el tamaño del array, utilizaremos el símbolo ‘#’ entre el símbolo ‘$’ y el nombre del array. El tamaño obtenido nos da el último índice del array, es decir, empieza contando por 0 y, si el array no tiene ningún elemento, el tamaño es -1.

Arrays Asociativos

Un array asociativo o tabla hash es aquel donde los elementos son referenciados mediante claves en vez de una posición.

En los arrays asociativos utilizaremos el símbolo ‘%’ para la representación.

En una línea similar a los arrays normales, para acceder a un elemento del array asociativo debemos referenciar como un escalar y la indexación por clave se debe hacer utilizando ‘{}’.

Podemos destacar también podemos hacer utilizando la operación de asociación con ‘=>’

Vamos a ver una serie de funciones propias de los arrays asociativos:

* keys: la función devolverá una array con las claves de un array asociativo.
* values: la función devuelve un array con los valores de un array asociativo.
* exits($hash{clave}): con esta función podremos ver si existe una clave o no.
* delete($hash{clave}): con esta función se elimina un elemento de un array asociativo.

3.3.2.3 Las listas asociativas

En las listas asociativas podremos destacar que la lista está indexada por cadenas en lugar de por números.

Para definir el tipo de lista asociativa utilizaremos ‘%’.

Debemos saber que un elemento está indexado por el anterior formando parejas del tipo (clave,valor). Entonces dada una lista asociativa de ejemplo:

%cuota=(“root”,1500,”pat”,500,”jen”,250);

Para poder acceder a cualquiera de los elementos de la lista cuota debemos conocer su clave, entonces:

$id=$cuota{“pat”}

$id=500

Podemos añadir nuevos valores y asociando a cada clave, su valor correspondiente, por lo tanto:

$couta{“dave”}=150; (la clave es “dave” y el valor asociado es 150)

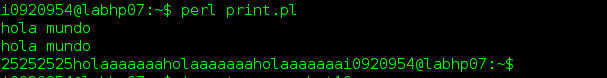
### 3.4 Entrada/Salida de Perl

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

El siguiente código se compone de una serie de **prints**

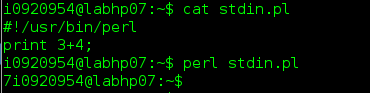
1. #!/usr/bin/perl
2. **print** "Hola mundo\n"; 🡪 imprime por pantalla “Hola mundo
3. **print** "hola \LMUNDO\n"; 🡪 imprime por pantalla “Hola mundo, porque \L hace que las letras que lo siguen sean minúsculas
4. **print** "25"x4;  🡪 imprime por pantalla 4 veces el número 3
5. **print** "holaaaaaaa"x3; 🡪 imprime por pantalla 3 veces “holaaaaaa”

Y el resultado que obtenemos será el siguiente



**Figura 18** resultado obtenido tras la ejecución del script

También podremos realizar operaciones aritméticas y mostrar el resultado final con print



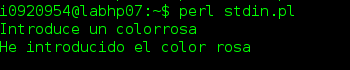
**Figura 19** resultado de la operación aritmética realizada

El método usado para la entrada de datos desde teclado es muy particular, puesto que para asignar un valor desde el teclado a una variable se asigna a la variable de representación del teclado **<STDIN>**

El siguiente código es un ejemplo de cómo utilizaremos la función <STDIN>

1. #!/usr/bin/perl
2. **print** "Introduce un color";
3. $color=<STDIN>;
4. **print** "He introducido el color $color"\n;

Y obtendremos el siguiente resultado:



**Figura 20** resultado obtenido tras la ejecución del script

### 3.5 Las referencias

Las referencias equivalen a los punteros en C, puesto que las referencias son un tipo de datos que permiten referenciar datos contenidos en otra entidad.

No forman una nueva representación de datos ya que las referencias son tratados como un dato escalar.

Para definir las referencias utilizaremos el símbolo ‘/’ y no existe ningún operador que anule la referencia

Las referencias en Perl consisten en que un registro posee las diversas referencias a cada elemento de memoria y las destruye cuando nadie hace referencia a esta, a diferencia de C, donde se ocupa un espacio de memoria independientemente del uso del puntero.

## **3.5.1 Creación de referencias**

Para crear una referencia, podremos utilizar un operador de referenciación en una variable o valor, cuando se trata de una variable, creamos un medio alternativo de acceder al valor de la variable.

$vescalar = \$vescalar;

$array= \@array;

$listaaso= \%listaaso;

Un dato a tener en cuenta es que, cuando utilizamos un operador de referenciación con un valor, creamos objetos anónimos que sólo pueden ser accedidos por medio de la referencia, pudiéndose construir estructuras más complejas.

$rescalar=\”hola”; 🡪 referencia a la cadena anónima “hola”

$array=[0,1,2]; 🡪 referencia al array anónimo (1,2,3)

$rlista={“llave1”=>“dato1”,”llave2”=>“dato2”};

%rNumeros={"arabes"=> [1,2,3], "romanos" => [I,V,X]} (lista asociativa anónima donde las claves son cadenas de caracteres y los valores son referencias a arrays)

## **3.5.2 Uso de referencias**

Para acceder a las referencias y modificar su valor. Como veremos en el ejemplo que viene a continuación para acceder al valor referenciado se utiliza el operador $ por duplicado ‘$$.

$nombre="entero";  
$entero=5;  
$rentero=\$entero;  
$$nombre=6;  
$$rentero=7;

Para abreviar la referencia a las listas asociativas o a los arrays se añadió el operador “->”.

$rarray->[2]="hola";

#Coloca “hola” como el tercer elemento del array referenciado

$rlista->["clave1"]="valor1";

#Coloca el par (“clave1” -> “valor1”)

Podemos realizar referencias a referencias y arreglos a referencias, de tal manera, que podremos hacer arrays multidimensionales como en C.

## 3.6 Operadores

Hemos visto los **operadores de comparación** en los operadores en las variables escalares y hemos observado que son similares a los de C y varían dependiendo si estamos hablando de cadenas o de números.

Los **operadores de asignación** se utilizan en PERL para dar valores a cualquiera de las variables validas en PERL. Podremos encontrar 2 tipos de operadores de asignación:

* **‘=’** :su funcionamiento es similar al de otros lenguajes de programación como C o Java.
* **‘=~’** : con este operador llevaremos a cabo las funciones de verificación de existencia de un patrón en una cadena y de sustituir un patrón de comparación por otra cadena en una expresión regular.

$var =~ s/one/uno/;

se reemplaza la ocurrencia de la cadena

'one' por la cadena 'uno.

Los operadores aritméticos son por los cuales podremos realizar operaciones aritméticas necesarias para el manejo de las expresiones y encontraremos los siguientes:

* **‘\*\*’:** lo utilizamos para obtener la potencia de un número.
* **‘.’:** lo utilizamos para concatenar 2 cadenas de caracteres.
* **‘x’:** lo utilizamos para multiplicar n veces la cadena de caracteres especificada.
* **‘%’:** lo utilizamos para obtener el módulo del cociente de 2 números.
* **‘|’:** lo utilizamos para realizar la operación binaria ‘o’/’OR’ entre 2 números.
* **'&’:** lo utilizamos para realizar la operación binaria ‘y’/’AND’ entre 2 números.

Los **operadores lógicos** son iguales a los que hemos visto en C (||,&&,!...)

Finalmente, los operadores de archivo son aquellos utilizados para verificar en el sistema operativo los permisos de un archivo o la naturaleza de su ejecución, entre otros. Los operadores serán los siguientes:

* **-r:** indica si el archivo tiene permiso de lectura
* **-W:** indica si el archivo tiene permiso de escritura
* **-T:** indica si el archivo es de tipo texto
* **-e:** indica si el archivo existe
* **-z:** indica si el tamaño del archivo es 0
* **-s:** indica si el tamaño del archivo es mayor que 0
* **-f:** indica si el archivo es plano
* **-d:** indica si es un directorio
* **-M:** indica el número de días transcurridos tras la última modificación
* **-B**: indica si el archivo es binario
* **-t**: indica si el archivo está en un terminal

## 3.7 Estructuras de control & subrutinas

En Perl manejaremos las estructuras de control básicas de cualquier lenguaje de programación, como son las estructuras condiciones y los bucles.

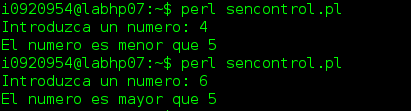
### Estructuras condicionales

Las sentencias que podremos encontrar son: if(),if() {} else(){} & if(){} elseif(){} else(){}

Veámoslo con un ejemplo

1. #!/usr/bin/perl
2. print"Introduzca un numero: ";
3. $numero=<STDIN>;
4. **if**($numero<5)
5. {
6. print"El numero es menor que 5\n";
7. }**else**{
8. print"El numero es mayor que 5\n";
9. }

Y obtendremos lo siguiente:



**Figura 21** resultado obtenido tras la ejecución del script

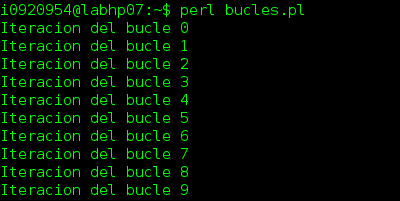
### Bucles

Las sentencias que podemos encontrar son: while(), for(), do(){} while() & foreach()

Lo vemos con un ejemplo

1. #!/usr/bin/perl
2. **for**($i=0;$i<10;$i++)
3. {
4. print"Iteracion del bucle $i\n";
5. }

Y obtenemos el siguiente resultado :



**Figura 22** resultado obtenido tras la ejecución del script

## 3.8 Ficheros

### 3.8.1 Creación de ficheros

### Para crear archivo tendremos que hacer lo siguiente

1. #!/usr/bin/perl
2. @datos=(); #ponemos los datos deseados
3. @registro=join(",",@datos); #delimitamos los datos con comas
4. open (NOMBREFICHERO,">nombrefichero.txt"); # Damos nombre logico y físico al archivo.
5. #Tomamos cada campo, uno por uno, y los grabamos en el archivo.
6. **foreach** $campo (@registro)
7. {
8. print NOMBREFICHERO $campo;
9. }
11. # Fin de línea y nueva línea para un posible nuevo registro.
12. print NOMBREFICHERO "\n";
14. # Cerramos el fichero abierto
15. close (NOMBREFICHERO);

Veamos esto con un ejemplo

1. #!/usr/bin/perl
2. @datos=("Juan",22,"Pedro",18,"Carlos",33,"Rosa",31,"Isabel",25);
3. @registro=join(",",@datos);
4. open (NOMBRES,">nombres.txt");
5. **foreach** $campo (@registro)
6. {
7. print NOMBRES $campo;
8. }
9. print NOMBRES "\n";
10. close (NOMBRES);

### 

### Como podemos ver, se ha creado el archivo correctamente

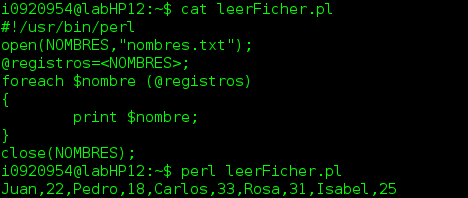
### 3.8.2 Lectura de ficheros

### Para poder leer un fichero, seguiremos la siguiente forma:

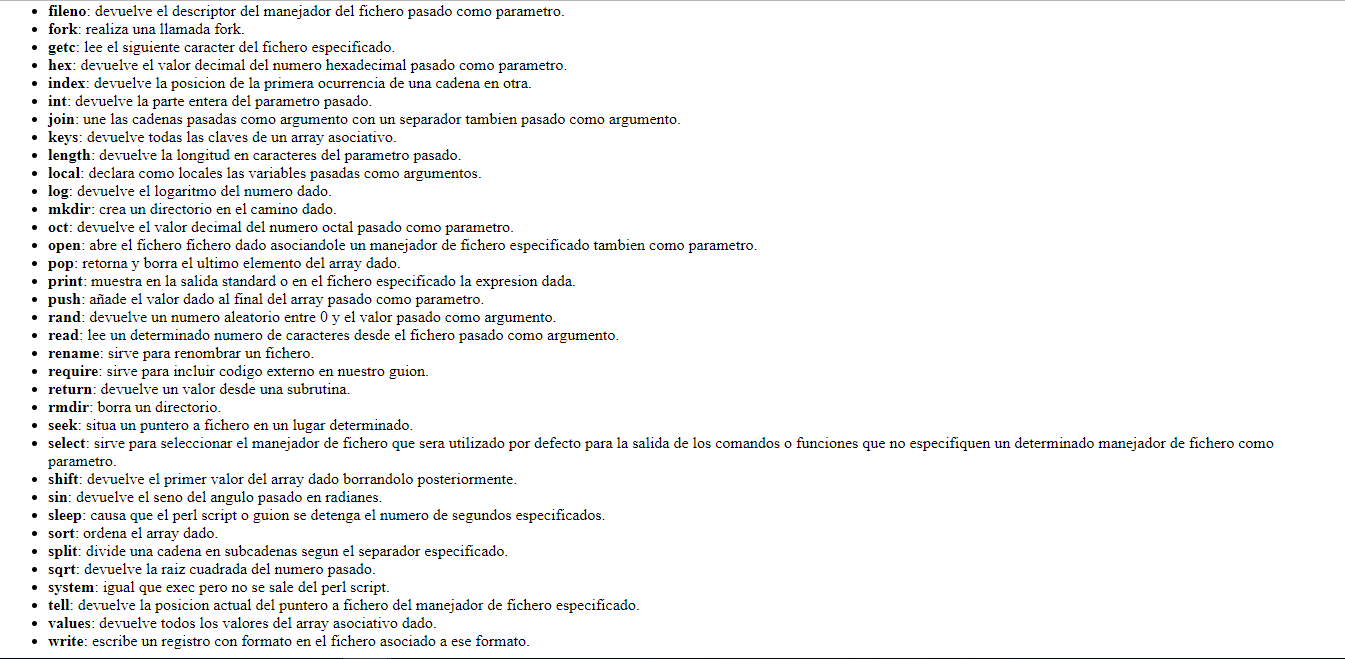
1. #!/usr/bin/perl
2. # Podemos usar el signo "<" o ninguno delante del fichero, para lectura
3. open (NOMBREFICHERO,"nombrefichero.txt");
5. #Añadimos cada línea de éste en la matriz.
6. @registros=<NOMBREFICHERO>;
8. # Mostramos los datos en pantalla
9. **foreach** $nombre (@registros)
10. {
11. print $nombre;
12. }
13. # Cerramos el fichero abierto
14. close (NOMBREIFICHERO);

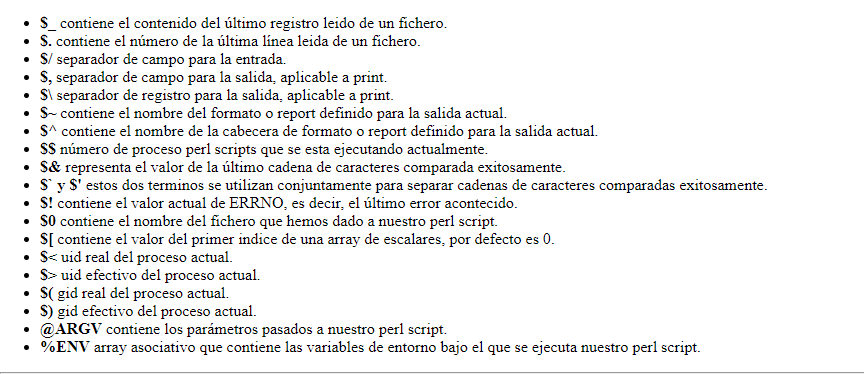
Veámoslo con un ejemplo

1. #!/usr/bin/perl
2. open (NOMBRES,"nombres.txt");
3. @registros=<NOMBRES>;
4. **foreach** $nombre (@registros)
5. {
6. print $nombre;
7. }
8. close (NOMBRES);



## 3.9 Ejemplo de algunas funciones & variables definidas en Perl importantes en Perl





**Figura 24** variables definidas

**Figura 23** funciones de Perl

# 4. Perl orientado a la administración de sistemas

Finalmente, vamos a ver como se utiliza este lenguaje a lo que nos compete, la administración de sistemas y es que Perl es uno de los lenguajes más utilizados en la administración de sistemas por su sencillez, flexibilidad, eficiencia y elevado rendimiento entre otras ventajas, a pesar de que como hemos mencionado anteriormente, Perl es más lento en comparación con otros lenguajes como Phyton.

## Añadir & gestionar usuarios

Vamos a ver una serie de ejemplos donde podremos encontrar distintas funciones que podremos hacer en Perl con añadir y gestionar usuarios.

## Vamos a añadir un grupo de usuario y será el grupo *“userss”*

1. #!/usr/bin/perl
2. system(‘addgroup userss’);

## Una vez creado el grupo usuario, vamos a introducir a un nuevo usuario al grupo.

1. #!/usr/bin/perl
2. system(‘adduser -ingroup userss Rodolfo’);

A este nuevo usuario vamos a moverlo a un grupo que ha sido creado previamente y lo llamaremos ‘grupo1’

Creamos otro grupo llamado ‘grupo1’ e introducimos a *“Rodolfo”* en el grupo creado y Rodolfo será un usuario secundario en el grupo1.

1. #!/usr/bin/perl
2. system(‘addgroup grupo1’);
3. system(‘adduser -ingroup grupo1 Rodolfo’);

Para poder meternos como *‘Rodolfo’* tendremos que hacer lo siguiente

1. #!/usr/bin/perl
2. system(‘su Rodolfo’);

Ponemos al grupo *‘userss’* una contraseña.

1. #!/usr/bin/perl
2. System(‘gpasswd userss’);

Continuamos creando un nuevo usuario en el grupo ‘users’ de manera que un nuevo usuario pueda utilizar la impresora y los dispositivos de audio y videos.

1. #!/usr/bin/perl
2. print“Users name ”
3. $name=<STDIN>
4. system(‘adduser -ingroup users ’.$name);
5. system(‘adduser $name ’ .users);
6. system(‘adduser $name ’.lp);
7. system(‘adduser $name ’.audio);
8. system(‘adduser $name ’.video);

Finalmente, vamos a ver cómo podemos eliminar usuario y para verlo, vamos a eliminar a ‘*Rodolfo’*

1. #!/usr/bin/perl
2. system(‘deluser -remove-home rodolfo’);

## Uso de cuotas

El uso de cuotas es bastante importante puesto que cualquier recurso que vaya a ser compartido tiene que ser administrado y de ahí su importancia y entonces, veremos una serie de ejemplos en Debian, comparándolos con Perl y veremos como es el uso de cuotas en Perl.

Vamos a obtener la información acerca de la ocupación del espacio de los discos del sistema

1. #!/usr/bin/perl
2. system (‘df -i /home’);

A continuación, obtendremos la información acerca del tamaño de los discos en home.

1. #!/usr/bin/perl
2. system (‘du -m/home);

## 4.3 Email

Vamos a ver una serie de ejemplos sobre la gestión de emails en Perl

**en Unix**

1. #!/usr/bin/perl
2. open my $SENDMAIL, '|-', '|/usr/sbin/sendmail -oi -t -odq' **or**
3. **die** "Can't fork for sendmail: $!\n";
4. print $SENDMAIL <<'EOF';
5. From: User Originating Mail <me@host>
6. To: Final Destination <you@otherhost>
7. Subject: A relevant subject line
8. Body of the message goes here after the blank line in **as** many lines **as** you like.
9. EOF
10. close(SENDMAIL) **or** warn "sendmail didn't close nicely";

## Vamos a ver como enviaríamos un correo en Windows y MacOS y para eso buscamos en CPAN los módulos necesarios

**En MacOs**

1. #!/usr/bin/perl
2. **use** MacPerl;
3. my $to      = 'user@example.com';
4. my $subject = 'Hi there';
5. my $body    = 'message body';
6. MacPerl::DoAppleScript(<<EOAS);
7. tell application "Mail"
8. set theNewMessage
9. subject:"$subject", content:"$body", visible:true}
10. tell theNewMessage
11. {address:"$to"}
12. send  end tell end tell EOAS

**en Windows**

1. #!/usr/bin/perl
2. **use** Win32::OLE;
3. **use** Win32::OLE::Const 'Microsoft Outlook';
4. my $outl = Win32::OLE->**new**('Outlook.Application');
5. my $ol   = Win32::OLE::Const->Load($outl);
6. my $message = $outl->CreateItem(olMailItem);
7. $message->Recipients->Add('user@example.edu');
8. $message->{Subject} = 'Perl to Outlook Test'; $message->{Body}    = "Hi there!\n\nLove,\nPerl\n";
9. $message->Send;

## 4.4 Ficheros de log

Vamos a ver una serie de ejemplos comparando lo que hacemos en Debian con lo que haremos en Perl para poder ver la gestión de usuarios utilizando Perl.

## Primero vamos a ver como leemos ficheros log y vamos a buscar por ejemplo la palabra ‘error’ en el fichero

1. #!/usr/bin/perl
2. open my $LOG, '<', "$logfile" **or** **die** "Unable to open $logfile:$!\n";
3. **while**(my $line = <$LOG>){
4. print **if** $line =~ /\berror\b/i;
5. }
6. close $LOG;

A continuación, vamos a ver como escribiremos un fichero log

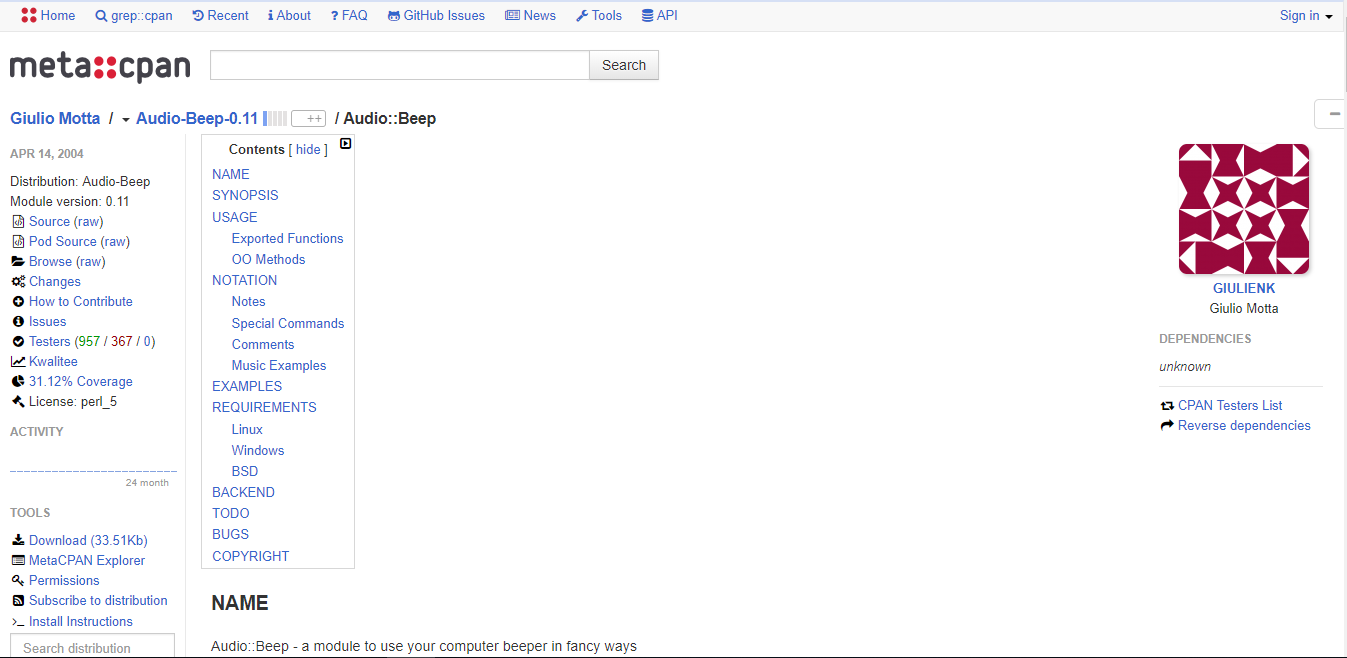
1. #!/usr/bin/perl
2. open my $LOGFILE, '>>', 'logfile' **or**
3. **die** "can't open logfile for append: $!\n";
4. print $LOGFILE 'began logfile example: ' .
5. scalar localtime . "\n";
6. close $LOGFILE;

Podremos observar que no necesitamos ningún método o modulo especial para escribir un fichero log.

Además de funciones, podemos encargarnos de la seguridad o de la configuración de ficheros y podremos registrar múltiples servicios, a parte del realizado y por esto decimos que Perl es un lenguaje muy útil en la administración de sistemas.

## 4.5 Registrar un servicio

A continuación, vamos a ver cómo podemos registrar un servicio en Perl y tendrá un funcionamiento similar a la práctica obligatoria pedida y es que vamos a hacer que al arrancar el sistema se reproduzca un sonido.

Tenemos que tener en cuenta que ***use Audio::Beep*** es un módulo que debemos instalar para el correcto funcionamiento del programa.

Página CPAN

Dónde se descarga

Este el código del sonido que se reproducirá

1. #!/usr/bin/perl
2. **use** Audio::Beep;
3. $frec=261.63;
4. $milliseconds=200;
5. beep ($freq, $milliseconds);
6. beep ($freq, $milliseconds);
7. beep ($freq, $milliseconds);
8. $frec=440.00;
9. beep ($freq, $milliseconds);
10. beep ($freq, $milliseconds);

Luego accedemos a la carpeta system y para eso haremos el siguiente comando:

*cd /etc/systemd/system*

Luego tenemos que hacer pico *cucarachaP.service* con lo siguiente:

*[Unit]*

*Description= reproduce la melodia*

*After=network.target*

*[Service]*

*ExecStart=/usr/bin/cucaracha.sh*

*[Install]*

*WantedBy=multi-user.target;*

Después tendremos que dar permisos de ejecución al programa y para eso haremos este comando

*$chmod +x cucarachaP.service*

A continuación, pondremos el siguiente comando para habilitar el servicio

*system enable cucarachaP.service*

Y como ya sabemos, generamos un link para poder hacer que se ejecute al comenzar el sistema.

Finalmente, reiniciamos el sistema o ejecutamos el comando *systemctl start cucaracha.service* y esto hará que ejecutemos la melodía de la cucaracha al arrancar el sistema.

# 5. Referencias

(Tom Christiansen, Brian d foy & Larry Wall con Jon Orwant) Programming Perl, 5th Edition O'Reily

Automating System Administration with Perl

<http://aplicaciones-web-lenguajes-programaci.blogspot.com/2011/12/perl.html>

<http://www2.iib.uam.es/bioinfo/curso/perl/tutoriales/cicei/cap2.htm>

<http://www.pablin.com.ar/computer/cursos/varios/perl1.htm>

<https://blog.desdelinux.net/introduccion-al-lenguaje-de-programacion-perl-parte-1/>

<http://flanagan.ugr.es/perl/index2.htm>

<https://desarrolloweb.com/articulos/541.php>

<https://blog.dinahosting.com/lenguaje-te-conviene-hoy-hablamos-perl/>

<https://lenguajesdeprogramacion.net/perl/>

<https://metacpan.org/pod/release/ENELL/POD2-ES-5.24.0.5/lib/POD2/ES/perl.pod>

<https://es.wikipedia.org/wiki/Perl>

<https://www.ecured.cu/Perl>

<https://www.i-programmer.info/news/222-perl/11937-perl-528-released.html>

<http://www2.iib.uam.es/bioinfo/curso/perl/tutoriales/cicei/cap3.htm>

<http://www6.uniovi.es/perl/tutor/cap02.htm>

<http://www6.uniovi.es/perl/tutor/cap05.htm>