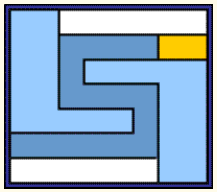
escuela técnica superior de ingeniería informática
Universidad de Sevilla



ALFA
Laboratorio

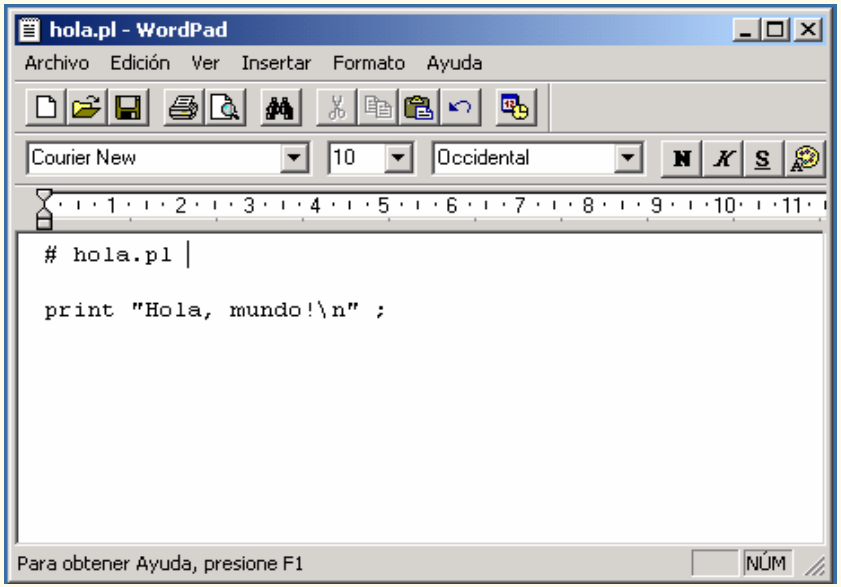
Víctor J. Díaz Madrigal
(vjdiaz@lsi.us.es)
Departamento de Lenguajes y Sistemas Informáticos
Sevilla, 2006

ÍNDICE

Lenguaje de programación Perl

INTRODUCCIÓN: Hola, mundo

Edición del script hola.pl



Ejecución del script hola.pl

Características generales de **Perl**

- Lenguaje multiplataforma y de distribución gratuita
- Lenguaje imperativo y de formato libre
- Las sentencias acaban con punto y coma
- Las sentencias se ejecutan en el orden en el que aparecen en el fuente
- Los comentarios empiezan con # y acaban con la línea
- Programación basada en **scripts** (guiones)

TIPO ESCALAR: Definición

Escalares (Números y cadenas)

Números: `12` `-5.4`

Cadenas: `"Hola, $nombre!\n"` `'Hola, $nombre!\n'`

Variables `$id`

Declaración por defecto con ámbito global (En id se distingue entre mayúsculas y minúsculas)

Inicialización por defecto: **`undef`** Según el contexto será el número 0 o la cadena vacía ""

Interpolación de variables dentro de literales cadenas enmarcadas entre dobles comillas "..."

Sentencia de asignación

```
$id =EXPRESION;
```

Operadores para números

Suma	+	Producto	*	Modulo	%
Resta	-	División	/	Potencia	**

```
# escalares.pl

$autor="Richard Wagner";
$ciudad="Leipzig";
$operas=11;
$nace=1813;
$muere="1883";

print "$autor nace en $ciudad\n";
print "Compuso $operas operas\n";
print "Vivió ",$muere-$nace," años\n";
```

[temperaturas.pl] Convierta una temperatura F medida en grados Fahrenheit a otra C medida en grados Celsius [Nota: C=5(F-32)/9]

TIPO ESCALAR: Condiciones

Sentencia Condicional

```
if (COND) {BLOQUE} [ [elsif (COND) {BLOQUE}]*  else {BLOQUE} ]
```

```
SENTENCIA if COND;
```

Expresiones Lógicas

Valor Falso				Conjunción		Disyunción		Negación	
"0"	""	0	**undef**	&&	**and**	||	**or**	!	**not**

Operadores Relacionales

	Números	Cadenas
Mayor que	>	**gt**
Menor que	<	**lt**
Igual que	==	**eq**
Mayor o igual	>=	**ge**
Menor o igual	<=	**le**
Distinto	!=	**ne**

```
# condiciones.pl

$n=4;

if (($n>=0) && ($n%2==0)) {
    print "$n es positivo y par\n"
} else {
    print "$n es negativo o impar\n"
}

$u="Aire";
$v="Aria";

print "La cadena $u ";
print "NO " if ($u ge $v);
print " precede a la cadena $v\n";
```

[raices.pl] Indique de qué tipo son las raíces de una ecuación de segundo grado $ax^2+bx+c=0$. Muestre, si es posible, sus raíces reales. **Nota:** Use la función sqrt() para calcular la raíz cuadrada

TIPO ESCALAR: Bucles y rupturas

Bucle while

```
while (COND) {BLOQUE}
```

```
SENT while COND;
```

Bucle for

```
for (INICIO; COND_FIN; SIGUIENTE) {BLOQUE}
```

Rupturas

`die ($mensaje)`	Cesa la ejecución e imprime $mensaje	Acaba el bucle	`last [COND]`
`exit($código)`	Cesa la ejecución y devuelve $código	Siguiente iteración	`next [COND]`

```
# bucles.pl

$n= 4;

die ("El argumento no es un entero estrictamente positivo\n")
   if ($n<=0);

$fact=1;
$i=$n;
while ($i != 1) {
   $fact=$fact*$i;
   $i--;            #  i-- equivale a $i=$i-1;
}

print "El factorial de $n es $fact\n";
```

Operadores incremento/decremento

Suma 1	`++`
Resta 1	`--`
Suma n	`+=`
Resta n	`-=`

[primo.pl] Indique si un determinado número n, mayor que 2, es primo o no

TIPO ESCALAR: Ficheros de texto

Apertura del fichero

```
open(MANIPULADOR,$fichero)
```

Modo de apertura ($fichero)

Lectura	Escritura (creación)	Escritura (adición)
`"fichero"`	`">fichero"`	`">>fichero"`

Lectura de una línea

```
$cadena=<MANIPULADOR>
```

Escritura de una línea

```
print MANIPULADOR LISTA
```

Cierre del fichero

```
close(MANIPULADOR)
```

```
# ficheros.pl

$entrada="ficheros.in";
$salida="ficheros.out";
open(IN,$entrada)
   or die("No puedo abrir $entrada!\n ");
open(OUT,">$salida")
   or die("No puedo crear $salida!\n ");

$num=0;
while ($linea = <IN>) {
     print OUT  $num++,":$linea";
}
close(IN);
close(OUT);
```

```
while (<IN>) {
     print OUT $num++,":$_";
}
```

`$_` Variable sobre la que se realiza la lectura (y escritura) por defecto

[acumula.pl] Suma los números de un fichero de texto. Nota: Cada número está en una línea distinta
NOTA: La función **chomp**(CADENA) suprime, si existe, el último \n de CADENA

EXPREGs: Definición

Una **expresión regular** (patrón) es una notación para representar un conjunto de cadenas mediante una sola cadena

`/PATRON/`

`$c=~m/PATRON/`

Busca de izquierda a derecha en la cadena

`$_`

`$c`

e indica si contiene una subcadena que encaje con PATRON

Meta-caracteres
Caracteres que actúan como operadores

`\ . ^ $ * + ? { } [ ] ( ) |`

Expresiones regulares básicas

a	El símbolo a (salvo meta-carácter)
\x	El meta-carácter x (como símbolo)
abc	La cadena abc

`$&` Guarda la **primera** subcadena que encajó en PATRÓN

`$`` Guarda la subcadena previa a $&

`$'` Guarda las subcadena posterior a $&

```
# expreg.pl

$_="Para todo x: |x|=|-x|";

print "[$`]\n[$&]\n[$']\n" if /x/; # imprime [Para todo ]
                                   #         [x]
                                   #         [: |x|=|-x|]

print "[$&]\n" if m/todo/;          # imprime [todo]

$e="la menor: la si do re mi fa(#) sol(#) la";

print "[$&]\n" if $e=~/\(#\)/;     # imprime [(#)]
```

EXPREGs: Secuencias de escape y comodín

Secuencia de escape: Cadena precedida por una barra \ y sujeta a interpretación en expregs y literales cadenas

Caracteres no alfanuméricos	Interpretación textual (Permite el uso de meta-caracteres como símbolos en expregs)
Caracteres alfanuméricos	La mayoría de ellos (meta-símbolos) son reinterpretados

Meta-símbolos para representar símbolos (especiales) en cadenas dobles o en expregs

`\n`	Salto de línea	`\r`	Retorno de carro	`\e`	Escape	`\a`	Alarma
`\t`	Tabulador	`\f`	Alimentación de hojas	`\cx`	Control-x	`\b`	Espacio atrás

`\nnn`	Carácter Octal (ASCII) (\000 a \377)	`\xnn`	Carácter Hexadecimal (ASCII) (\x00 a \xff)

Comodín	`.`	Cualquier símbolo salvo \n

```
# escape.pl

print 'La cadena simple no interpreta (salvo \' y \\) secuencias de escape',"\n";

$_="\a\aFalta de ortograf\xa1a:\t la e de America se acent\243a";
print $_;

print "[$&]\n"  if /.cent\243./;   # imprime [acentúa]
```

[**sílabas.pl**] Un fichero contiene una palabra (con sus sílabas divididas con guiones) en cada una de sus líneas. Muestra las palabras, de más de una sílaba, cuya última sílaba está formada por tres letras.

EXPREGs: Concatenación y alternancia

$p_1 \ldots p_n$

Concatenación: Una cadena que encaje sucesivamente con los patrones p_1, p_2, ..., p_n

Las cadenas son un caso particular de concatenación donde los patrones que la forman son caracteres (o secuencias de escape)

$p_1|p_2 \ldots|p_n$

Alternancia: Una cadena que encaje en alguno de los patrones p_1, p_2, ..., p_n

Los patrones en una alternancia se aplican en el orden en el que figuran
La búsqueda se detiene con la primera cadena que encaje en algún patrón.

Paréntesis `()`

Modifica la precedencia de operadores
La alternancia es menos prioritaria que la concatenación

```
# alternancia.pl

$_="En el priorato en el que estaba aquel prior";

print "[$`][$&]\n"  if /aquel|e(n|l)/;     # imprime [En ][el]

print "[$`][$&]\n"  if /priorato|prior/;   # imprime [En el ][priorato]

print "[$`][$&]\n"  if /pri(or|orato)/;    # imprime [En el ][prior]
```

[sufijos.pl] Un fichero contiene en cada una de sus líneas una palabra escrita en minúsculas. Muestra todas las palabras formadas con los sufijos dad, endo o mente.

EXPREGs: Clases

`x-y`	**Rango:** Un carácter incluido entre los caracteres x e y
`[a1..an]`	**Clase:** Un carácter del conjunto {a_1, ... , a_n}
`[^a1...an]`	**Clase Complementaria:** Un carácter no incluido en [$a_1a_2..a_n$]

Abreviaturas	Dígitos		Alfanuméricos		Espacios	
	`\d`	[0-9]	`\w`	[A-Za-z0-9_]	`\s`	[\t\n \r\f]
	`\D`	[^0-9]	`\W`	[^A-Za-z0-9_]	`\S`	[^\t\n \r\f]

Dentro de las clases se admiten rangos, secuencias de escape y abreviaturas (Mediante \^, \- y \] podemos incluir los símbolos ^, - y] dentro de las clases)

```
# clases.pl

$_="Vivaldi (1678-1741) compuso alrededor de 500 conciertos.";

print "[$&]\n"  if /\w[aeiou][^A-Z\d][a-z]/; # imprime [Viva]

print "[$&]\n"  if /\d\d[\s.()\-]\d\d/;      # imprime [78-17]
```

[horas.pl] Las líneas de un fichero contienen una hora enmarcada entre barras (formato |HH:MM| o |H:MM|). Muestra las líneas que contienen una hora incoherente. Por ejemplo: |24:00| o |9:65|.

EXPREGs: Cuantificadores

Cuantificadores: Representan cadenas donde se repite un mismo patrón un cierto número de veces

	Cero o más	Una o más	Cero o una	Entre n y m	Justo n	Al menos n
Cadena máxima	`p*`	`p+`	`p?`	`p{n,m}`	`p{n}`	`p{n,}`
Cadena mínima	`p*?`	`p+?`	`p??`	`p{n,m}?`	`p{n}?`	`p{n,}?`

No es factible apilar cuantificadores. Ej: **p*+** es un error sintáctico.
Los cuantificadores son más prioritarios que la alternancia y que la concatenación

```
# cuantificadores.pl

$_="QUINTAS: fa do sol re la mi si fa# do# sol# re# la# (mi#)";

print "[$&]\n" if /[a-z]{1,5}/;   # imprime [fa]
print "[$&]\n" if /[a-z]{3}/;     # imprime [sol]
print "[$&]\n" if /[#a-z]{4,}/;   # imprime [sol#]

print "[$&]\n" if /[a-z]*/;       # imprime []
print "[$&]\n" if /[a-z]+/;       # imprime [fa]
print "[$&]\n" if /do#?/;         # imprime [do]

print "[$&]\n" if /#.*#/;         # imprime [# do# sol# re# la# (mi#]
print "[$&]\n" if /#.*?#/;        # imprime [# do#]
```

[binarios.pl] Muestra aquellas líneas de un fichero constituidas exclusivamente por tres números binarios de los que el segundo es, además, un número par. Dichos tres números pueden estar precedidos o sucedidos por un número arbitrario de espacios en blanco.

EXPREGs: Interpolación y agrupamiento

Interpolación	Las variables que figuran en PATRON son interpoladas. (Con \$ se evita una interpolación no deseada)	
Agrupamiento	Cada subexpresión regular encerrada entre paréntesis en PATRÓN determina un grupo.	
	Los grupos se numeran según el orden en que figuran los paréntesis izquierdos.	
	`$1, $2, ..., $n`	Variables que guardan la subcadena que encajó con el i-ésimo grupo.
	`\1, \2, ..., \n`	Aserciones para acceder, DENTRO de PATRÓN, a $1, $2, ..., $n.

```
# grupos.pl

$_="Renacimiento: Victoria (1548-1611), Guerrero (1527-1599) y Morales (1500-1553)";

$fecha="\\d\\d\\d\\d";                          # Cadena alternativa  '\d\d\d\d'

print "[$1 - $2]\n" if /($fecha)-($fecha)/;     # imprime [1548-1611]

print "[$1-$3]\n" if /((\d\d)00)-\2(\d\d)/;     # imprime [1500-53]
```

[mesas.pl] Las líneas de un fichero de texto contienen datos sobre las dimensiones (medidas en cms) y el precio (en euros con un máximo de dos decimales) de varios modelos de mesas. Los datos en las líneas se ajustan al siguiente formato:

modelo, altura, anchura, profundidad, precio

Imprima la marca y el precio de todas las mesas que sean cuadradas.

EXPREGs: Modificadores

Modificadores para m//		
	`/i`	Ignora la diferencia entre mayúsculas y minúsculas al emparejar
	`/s`	El operador . encaja en cualquier símbolo (incluido \n)
	`/x`	Ignora espacios y comentarios de línea en el PATRÓN (Las clases o la barra eliminan dicho efecto)
	`/o`	Compila sólo una vez el PATRÓN (aunque contenga variables cuyo contenido puede haber variado)
	`/g`	Búsqueda global (iterativa) hasta que no empareja PATRÓN

Función	`pos($cadena)`	devuelve el último índice accedido por $cadena=~m//g

```
# modificadores.pl

$c1="Verde que te quiero verde.\nVerde viento. Verdes ramas.\nVerde";

print "La cadena original es\n[$c1]\n\n";

$cadena="verde";
while ($c1=~m/($cadena)/igo) {
    print "m//gi encaja con [$1] y sigue buscando desde ",pos($c1),"\n";
}

print "\nm//sx encaja con\n[$1]" if ($c1=~m/(Verde   # Primer Verde
                                          .*      # lo que sea en medio
                                          Verde)  # Ultimo Verde
                                         /sx);
```

[faqs.pl] Imprime las preguntas que figuran en un fichero de texto en español de FAQs (Frequently Asked Questions). Nota: Una misma pregunta puede ocupar más de una línea en un mismo párrafo

EXPREGs: Sustitución

Sustituye las subcadenas que encajan en la expreg PATRÓN por la cadena REMPLAZO

`s/PATRON/REMPLAZO/`

`$c=~s/PATRON/REMPLAZO/`

Sustituye de izquierda a derecha la primera subcadena de

`$_`

`$c`

y devuelve el número de sustituciones realizadas

Modificadores

`/i` `/s` `/x` `/o` El mismo efecto que cuando modifica m//

`/g` Sustituye global de todas las subcadenas que encajan en PATRÓN

`/e` Sustituye el resultado de evaluar REMPLAZO

```
# sustitucion.pl

$c0=$c1="Hay un tono salvo de mi a fa, y de si a do.";
$r=$c1=~s/do|re|mi|fa|sol|la|si/NOTA/g;
print "Original   [$c0]\nResultante [$c1]";
print "\nSe han realizado $r sustituciones\n\n";

$c0=$_="El resultado del partido ha sido de 2-2";
s/(\d+)/$1+1/e;
print "Original   [$c0]\nResultante [$_]"
```

[intercambio.pl] Las líneas de un fichero son secuencias de pares de la forma N:M, donde N y M son dos números naturales. Crea un fichero donde los números figuran intercambiados. Es decir, M:N

EXPREGs: Anclas

Ancla: Es una **restricción** sobre la posición en la que se debe buscar un patrón. **OJO:** No encaja con **ningún** símbolo

\A	Denota el principio de una cadena
\Z	Denota el final de la cadena o justo delante de \n si éste es el último carácter
\z	Denota el final de la cadena

Operadores

\A es equivalente a ^
\Z es equivalente a $

\b	Denota el límite de un identificador (\w\W o \W\w)
\B	Denota el interior de un identificador (\w\w)

Un \b en una clase (Ej: [\b]) representa *backspace*
Los límites incluyen el inicio y el fin de una línea

```
# anclas.pl

$_="Aceite y Vinagre";

print "Ancla B [$1]\n" while /(.\Be\B.)/gi;  # imprime [cei]  de aceite
print "Ancla b [$1]\n" while /(..\Be\b)/gi;  # imprime [ite] de aceite y [gre] de vinagre

print "Ancla A [$1]\n" if /\A(\w+)/;         # imprime [Aceite]
print "Ancla Z [$1]\n" if /(\w+)\Z/;         # imprime [Vinagre]
print "Ancla z [$1]\n" if /(\w+)\z/;         # imprime [Vinagre]

$_="Aceite\nVinagre\nSal\n";

print "Ancla Z [$1]\n" if /(\w+)\Z/;         # imprime [Sal]
print "Ancla z [$1]\n" if /(\w+)\z/;         # !!!!!! NO IMPRIME NADA !!!!!!
```

[palabra.pl] Cuenta las apariciones de una palabra en un texto

[quita_s] Elimine todos los espacios innecesarios de un texto almacenado en un fichero.

LISTAS: Listas y arrays

Listas (Secuencia de escalares)

Literales	`(1,2,3)`	`("uno", 2, "tres")`
Rangos	`(1..10)`	
Asignación Múltiple	`($a,$b,$c) = (0,34, "abc");`	
Iterador	`foreach VAR (LISTA) { BLOQUE }`	

Arrays (Listas con nombre)

Variables	`@array`
Acceso al i-ésimo elemento	`$array[$i]`
Longitud	`$l=@array`

OJO :
La lista vacía () representa también el valor lógico falso

Los elementos de un array valen por defecto undef

```
# arrays.pl

($nacimiento,$defuncion)=(1813,1883);

@ciudades=("Leipzig","Paris","Dresde","Bayreuth","Venecia");

print "Wagner nace en $ciudades[0] y muere en $ciudades[3] en $defuncion\n";

$num_ciudades=@ciudades;
print "Las $num_ciudades ciudades más importantes en su vida fueron:\n";
foreach $ciudad (@ciudades) { print "$ciudad\n" };
```

[media.pl] Calcula la media de un array de números

LISTAS: Conversión entre listas y cadenas

Mezcla	`join(CADENA,LISTA)`
División	`split(/PATRON/,CADENA)`

```
# split.pl

$nif="28.198.654-W";

@lista=split(/[.-]/,$nif);

print "El nif original era $nif \n";

print "Al dividirlo quedan los siguientes elementos:\n";

foreach $e (@lista) {print "\"$e\"\n";}

$nuevo_nif=join("",@lista);

print "Tras mezclarlo queda: $nuevo_nif\n";
```

[notas.pl] Un fichero de texto incluye en cada una de sus líneas información sobre las notas obtenidas en tres ejercicios por un alumno. El formato de cada línea es el siguiente:

Apellidos : Nombre : DNI : Nota_Ejercicio_1 : Nota_Ejercicio_2 : Nota_Ejercicio_3

Cree un fichero cuyas líneas tendrán el siguiente formato:

Apellidos, Nombre --- Nota

donde Nota es la nota media obtenida en los 3 ejercicios.

LISTAS: Actualización de arrays

	Por el principio	Por el final
Inserción de elementos	`unshift(ARRAY,LIST)`	`push(ARRAY,LIST)`
Eliminación de un elemento	`shift(ARRAY)`	`pop(ARRAY)`

```
# push_pop.pl

@gazpacho=("pepino","tomate","pimiento","ajo","pan");
print "Ingredientes para hacer un gazpacho: @gazpacho\n";

print "Si no quieres que se repita tanto, quita ",shift(@gazpacho),"\n";

unshift(@gazpacho,"sal","vinagre","aceite");
print "Para darle sabor hay que aderezar: @gazpacho\n";

print "Para no engordar, reduce el ", pop(@gazpacho), " y echa más tomate\n";

print "Algunos le ponen pimiento rojo para darle más color\n";
push(@gazpacho,"pimiento rojo");

print "En resumen, los ingredientes son: @gazpacho\n ";
```

[vacia.pl] Vacía un array de números @array de forma que su contenido sea repartido entre tres arrays @positivos, @negativos y @nulos que contendrán respectivamente los elementos de @array que son positivos, negativos y ceros. Dichos tres arrays se construirán según se vacía @array.

LISTAS: Mapas

Mapas
Listas de pares de escalares clave-valor
(la clave es una cadena y el valor un escalar)

Literales

```
("uno", 1, "dos",2,"tres",3)
( uno=> 1, dos => 2, tres =>3)
```

Variables

```
%mapa
```

Acceso al valor asociado a $clave

```
$mapa{$clave}
```

OJO :
El valor de una clave no creada es por defecto undef

Proyección

```
keys (%mapa)
```

Supresión de una par

```
delete($mapa{$clave})
```

Existencia de una par

```
exists($mapa{$clave})
```

```
# mapas.pl

%operas =(Bartok => 1, Falla=> 1, Brahms => 0);

foreach $autor (keys %operas) {
   print "$autor compuso $operas{$autor} operas\n";
}

delete($operas{Falla});

if (!exists($operas{Berg})) {
    $operas{Berg}=2;
}

foreach $autor (keys %operas) {
   print "$autor compuso $operas{$autor} operas\n";
}
```

[conjunto.pl] Elimina los elementos repetidos de una lista

LISTAS: Ordenación de arrays

Ordena lexicográficamente los elementos de ARRAY

```
sort {BLOQUE} (ARRAY)
```

Cambio del criterio de comparación

Operador de comparación	Números	<=>	Cadenas	cmp
Variables de comparación	$a	$b		

Invierte el orden de los elementos de ARRAY

```
reverse(ARRAY)
```

```
# sort.pl

@coro=("Soprano", "Contralto", "Tenor","Bajo");

foreach $c (@coro) {print "$c\n";}

@ordenada=sort(@coro);
print "Orden ascendente: @ordenada\n";

@ordenada_por_longitud=sort {length($a) <=> length($b); } @coro;
print "Ordena por longitud de cadena: @ordenada_por_longitud\n";

@invertida=reverse sort(@coro);
print "Orden descendente: @invertida\n";
```

[maximo.pl] Calcula el máximo de un array de números