Introducción

Perl (Practical Extraction Report Language).

Perl es un lenguaje de script (o de guiones), lo que quiere decir que no hace falta un fichero binario para poder ejecutar las instrucciones que hemos codificado usando este lenguaje, es decir, es interpretado, aunque el intérprete de Perl "compila" los programas antes de ejecutarlos. Perl tiene características de muchos lenguajes de programación (buscando lo mejor de cada uno), pero al que más se parece es al C.

Ejemplo 1:

```
#!/bin/perl
print "Hola Mundo\n";
```

La primera línea indicamos donde está el intérprete de Perl, después de `#!`. Esto no haría falta si, al ejecutar el script, lo hacemos usando el interprete. Por ejemplo:

```
/usr/bin/perl hola.pl
```

Ahora lo guardamos en un fichero con extensión **.pl** que es la extensión de los scripts en Perl. Para ejecutarlo, basta con llamarlo por su nombre (si se tiene los permisos de ejecución adecuados):

```
./hola.pl
```

Variables

Los datos escalares son las variables simples y pueden contener enteros, reales, cadenas o referencias. Este tipo de variables van precedido siempre por **$**. Esta es una de las mayores diferencias con lenguajes como el C y es que a cualquier variable escalar se le antepone dicho signo (**$**). En Perl las variables no se tienen porque declarar, aunque se declaran con **my**. Además la interpretación del tipo de variable se hace en función del contenido, aunque todas la variables numéricas internamente se tratan como reales.

Ejemplo 2:

```
#!/usr/bin/perl

#Declaro una variable
my $hola;

#Asigno valores
$hola="Hola Mundo\n";
$adios="adios\n";

#Escribimos un poco en pantalla
print $hola;
$hola=23;
print "Mira un número: $hola \n";
print "Y $adios";
```

Como podemos ver en el anterior ejemplo las cadenas se encierran entre comillas dobles (" "), aunque también se pueden cerrar entre comillas simples (' '). Existen caracteres especiales.

- **\n** retorno de carro a una nueva línea
- **\b** retorno a la posición anterior
- **\a** pitido
- **\t** tabulación (8 espacios)
- **** el caracter \

Operadores de Comparación

Son similares a los de C, pero depende si estamos comparando cadenas o datos numéricos.

Comparación	Números	Cadenas
igual	==	eq
no igual	!=	ne
menor que	<	lt
mayor que	>	gt
menor o igual que	<=	le
mayor o igual que	>=	ge

Para concatenar cadenas tenemos el operador punto (**.**) y la función `chop` para quitar el último carácter a una cadena.

Ejemplo 3:

```
#!/usr/bin/perl

#Asigno valores a variables
my $cad1="Hola";
my $cad2="Mundo";
my $cad3=$cad1." ".$cad2;

#Metemos una nueva línea y un caracter raro en $cad3
$cad3=$cad3."\n=";

#Y le quitamos el caracter raro
chop($cad3);

#Escribimos en pantalla
print $cad3;
```

Arrays

Un Array en Perl es como los de C, pero con la diferencia de que van precedidos del símbolo arroba **@**. (como las variables de **$**). Hay que tener en cuenta que cuando se accede a un elemento de un array, ya no se está haciendo referencia a un array sino a un dato escalar, por lo que debe ir precedido del símbolo $.

Ejemplo 4:

```
#!/usr/bin/perl

#Declaramos la variable primer_array como un array
my @primer_array;

#asignamos unos cuatro valores al array
@primer_array=(1,"dos",3,"cuatro");

#Añadimos un quinto de forma individual
$primer_array[4]=5.5;

#Mostramos el tercer elemento del array
print "El tercero es= ".$primer_array[3]." \n";
```

Para sacar/insertar elementos se pueden usar las funciones **pop** y **push**. Que sacan o insertan, respectivamente, un elemento al final, es decir, tratan el array como una pila. También podemos utilizar **shift** y **unshift** para sacar o insertar, respectivamente, un elemento del principio del array.

Para ver el tamaño (número de elementos) de un array se utiliza el símbolo de sostenido (#) entre el símbolo $ y el nombre del array, es decir, con **`$#array`**. Este tamaño nos lo da **contando desde 0** o, lo que es lo mismo, realmente nos da el último índice que existe en el array. Si el array no tuviese ningún elemento, su tamaño sería -1.

Ejemplo 5:

```
#!/usr/bin/perl

#asignamos unos cuatro valores al array
@matriz=(1,"dos",3,"cuatro");

#Añadimos con Push
push(@matriz, 5, 6, "siete");

#Mostramos el último elemento
print "El último es ". $matriz[$#matriz]."\n";

#sacamos con Pop
$uno=pop(@matriz);

print "He sacado $uno\n";

#Añadimos con unshift
unshift(@matriz, "cero", -1 );

#Mostramos el primer elemento
print "El primero es ". $matriz[0]."\n";

#sacamos con shift
$uno=shift(@matriz);

print "He sacado $uno\n";
print "La matriz tiene ".$#matriz." elementos\n";
```

En todo script de Perl existe el array **@ARGV** que contiene los parámetros de entrada.

Entradas y Salidas

Para leer de teclado se utiliza <STDIN>, que es un manejador de ficheros (Filehandle), que por costumbre se ponen en mayúsculas. (También existe <STDOUT> y <STDERR> para la sálida estándar y la sálida de errores respectivamente).

Ejemplo 6:

```
  #!/usr/bin/perl

  print "Hola. ¿Cómo te llamas?\n";
  $nombre=<STDIN>;

  #le quitamos el caracter de retorno de carro
  chop($nombre);

  if ($nombre eq "")
  {
        print STDERR '¿Qué pasa? ¿eres mudit@?'."\n";
  }
  else
  {
   print STDOUT "Hola $nombre, yo me llamo Flanagan. ¿trabajas o
vives?\n";
  }
```

Ficheros

Para trabajar con un fichero hay que abrirlo, escribir/leer de él y cerrarlo. Para abrir un fichero se utiliza la función `open` y para cerrarlo con `close`. El formato para abrir un fichero es **`open`** `(Manejador_de_fichero, Modo_y_NombreFichero)`. Lo modos de abrir un fichero se muestran en la siguiente tabla.

Modo	Significado
"Nombre"	Abre **Nombre** para leer.
"<Nombre"	Abre **Nombre** para leer. Igual que el anterior
">Nombre"	Abre **Nombre** para escribir. Si no existe lo crea
">>Nombre"	Abre **Nombre** para añadir al final
"+>Nombre"	Abre **Nombre** para lectura/escritura.

Ejemplo 7:

```
  #!/usr/bin/perl

  my $entrada="entrada.txt";
  my $salida ="salida.txt";

  open (ENTRADA,"<$entrada") || die "ERROR: No puedo abrir el fichero
$entrada\n";
  open (SALIDA,">$salida") || die "ERROR: No puedo abrir el fichero
$salida\n";

  while ($linea=<ENTRADA>)
  {
    print SALIDA $linea;
  }

  close (ENTRADA);
  close (SALIDA);
```

La función `die` seguido de un cadena lo que hace es finalizar (die=morir) el programa si la parte de la izquierda del O lógico es falsa (no funciona correctamente) y, antes de finalizar, muestra la cadena que le sigue a `die`.