Perl a vuelo de pájaro.

1 primeros ejemplos

Primer ejemplo.

```
camelot:~/lugro# perl
print "hola mundo\n";

^D

hola mundo
camelot:~/lugro#
```

Segundo ejemplo.

```
$nro = 123;
$txt = "hola mundo";
print $nro . " " . $txt . "\n";
print "$nro  $txt\n";
print '$nro  $txt\n';
```

Lo anterior muestra

```
 123 hola mundo          conversión y concatenación.
 123 hola mundo                  interpolación.
 $nro $txt\n       los apóstrofes impiden la interpolación.
```

Tercer ejemplo.

```
@l1 = (1,2,3,4,5);
@l2 = ('A','B','C','D','F');
@l3 = (@l1, @l2);
$l4 = @l2;
$l5 = (@l1, @l2);
print "@l1\n";
print "@l2\n";
print "@l3\n";
print "$#l1 $#l3\n";
print "$l4\n";
print "$l5\n"; # toma el array que esta mas a la derecha
```

Salida correspondiente.

```
1 2 3 4 5
A B C D F
1 2 3 4 5 A B C D F
4 9
5
5
```

Veamos qué resultan estas expresiones.

```
print 100*2 . "\n";
```

Resultado:

```
200
```

```
print 100*"2" . "\n";
```

Resultado:

```
200
```

```
print 100x"2" . "\n";
```

Resultado:

```
100100
```

2 algo de expreciones regulares

Las expresiones regulares de Perl son derivadas de las usadas por sed, AWK.

```
while(<>) {
    print if /^[A-F].*/;
}
```

Con la siguiente entrada

```
Esta es una
prueba para
ver
Cuales son
mostradas
```

produce

```
Esta es una
Cuales son
```

y

```
while(<STDIN>) {
    s/(\S+)\s+(\S+)/$2 $1/;
    print;
}
```

con

```
dia noche
sol luna
cosa golda
re ves
```

produce

```
noche dia
luna sol
golda cosa
ves re
```

3 hashes

Perl tiene *hashes*.

```
%h1 = (hola => 'que tal', hasta => 'la vista, baby');

$h2{hola} = 'que tal';
$h2{hasta} = 'la vista, baby';

foreach $i (keys %h1) {
    print "h1: a $i le corresponde %h1{$i}\n";
}
foreach $i (keys %h2) {
    print "h2: a $i le corresponde %h2{$i}\n";
}

print "claves: " . keys(%h1) . "\n";
print "valores: " . values(%h1) . "\n";
print "claves: ";
print  keys(%h1); print "\n";
print "valores: ";
print values(%h1); print "\n";
```

Salida:

```
h1: a hasta le corresponde la vista, baby
h1: a hola le corresponde que tal
h2: a hasta le corresponde la vista, baby
h2: a hola le corresponde que tal
claves: 2
valores: 2
claves: hastahola
valores: la vista, babyque tal
```

4 Contextos de evaluación

Las evaluaciones cambian mucho según el contexto. Por ejemplo, en un contexto de lista,

```
@x = (11, 222, 3333);
print "@x\n";
```

da

```
11 222 3333
```

Pero en un contexto escalar como

```
$x = (11, 222, 3333);
print "$x\n";
```

da

```
3333
```

5 Referencias

Perl tiene referencias, similares (aunque no son lo mismo) a los punteros de C. Ejs.

```
$n = 10;
$rn = \$n;
@l = ("hola", "que", "tal");
$rl = \@l;
%h = ("Laurel" => "Hardy", "Thompson" => "Williams");
$rh = \%h;

$$rn++;
print "\$n = $n\n";

push @$rl, "que hago?";
print @l; print "\n";

$rl->[$#$rl+1] = "ahora?"; # similar a $l[$#l+1] = "ahora?"
print @l; print "\n";

$$rh{"Sunday Horse"} = "IMF";
foreach $i (keys %h) {
    print "$i -> $h{$i}\n";
}

$rh->{"INFLACION CERO"} = "DONDE?";
foreach $i (keys %h) {
    print "$i -> $h{$i}\n";
}
```

6 eval

```
while(<>) {
    print eval($_) . "\n";
}
```

El siguiente programa

```
$a = 1;
$b = 0;
$c = $a / $b;
print "$c\n";
```

da

```
Illegal division by zero at excep0.pl line 3.
```

En cambio el que sigue es más civilizado.

```
$a = 1;
$b = 0;
$c = eval("$a / $b");
if($c) { print "Bien!. c=$c\n"; }
else { print "Ooops! Divisor nulo!\n"; }

$b = 2;
```

```
$c = eval("$a / $b");
if($c) { print "Bien! c=$c\n"; }
else { print "Ooops! Divisor nulo!\n"; }
```

```
Ooops! Divisor nulo!
Bien! c=0.5
```

7 Sockets

7.1 Servidor

```
use IO::Socket;

$sock = IO::Socket::INET->new(
    Listen    => 5,
    LocalAddr => 'localhost',
    LocalPort => 8000,
    Proto     => 'tcp')
or die "IO::Socket!";
$otro_sock = $sock->accept();
$mens = <$otro_sock>;
chop $mens;
print "[$mens]\n";
eval($mens);
close $otro_sock;
close $sock;
```

7.2 Cliente 1

```
use IO::Socket;

$sock = IO::Socket::INET->new(
    PeerAddr => 'localhost',
    PeerPort => 8000,
    Proto => 'tcp'
) or die "IO::Socket!";
print $sock "print \"hola mundo!\\n\";";
close $sock;
```

7.3 Cliente 2

Otro ejemplo más convincente es correr –con el mismo servidor– este otro cliente:

```
use IO::Socket;

$sock = IO::Socket::INET->new(
PeerAddr => 'localhost',
PeerPort => 8000,
Proto => 'tcp'
) or die "IO::Socket!";
print "que quiere que haga?...";
print $sock (<>);
close $sock;
```

8 Regex Plus

8.1 Clases de caracteres abreviados

Constructor	Clase Equiv.	Constructor Negado	Clase Equiv. Negada
\d	[0-9]	\D	[^0-9]
\w	[a-zA-Z0-9_]	\W	[^a-zA-Z0-9_]
\s	[\r\t\n\f]	\S	[^ \r\t\n\f]

Un ejemplo puede ser:

```
[\da-fA-F] # matchea un digito hexadecimal
```

Secuencias

simbolo	accion
*	cualquier cantidad de caracteres
?	uno o cero caracteres
+	uno o mas caracteres
{n,m}	minimo n, y maximo m

* es equivalente a {0,}, + es equivalente a {1,}, y ? es quivalente a {0,1}

8.2 Greedy o Non-Greedy?

```
$_ = "a xxx c xxxxxxxxxxxx c xxx d ";
/a.*c.*d/;
```

`a.*c` va a matchear hasta la segunda c

```
$_ = "a xxx c xxxxxxxxxxxx c xxx d ";
/a.*?c.*d/;
```

`a.*?c` va a matchear hasta la primera c

8.3 Parentesis

```
/hola(.)quetal\1/;
```

El parentesis actuaria de memoria, pudiendo asi matchear el mismo caracter al final de la linea. Por otra parte este puede ser el intento de como matchear doble palabras:

```
/(\w+) +\1/;
```

El siguiente es un intento de matchear la doble palabra y llvarla a una sola.

```
$_ = "Hola Hola que tal";
s/(\w+) +\1/$1/;
print;
```

el resultdo de eso seria: Hola que tal. Por otro lado:

```
$_ = "<tag>Hola que tal</tag>";
print $2 if /\<([^>]+)\>([^<]+)\<\/\1\>/;
```

La idea de esta linea es captura lo que tenemos dentro ¡tag¿ y ¡/tag¿, no importa cual sea el "tag" en cuestion, queremos capturar lo que tenemos dentro de los mismos. Un ultimo ejemplo de los parentesis y de su efecto:

```
abc*       # matchea ab, abc, abcc, abccc, etc
(abc)*     # matchea "", abc, abcabc
^a|b       # matchea a al principio de la linea y a b en cualquier lado
^(a|b)     # matchea a o b al principio de la linea
(a|b)c     # matchea ac o bc
```

8.4 Split & Join

```
$line = "jdiaz::517:100:Jose Luis Diaz:/var/home/jdiaz:/bin/bash";
@fields = split(/:/, $line);
# ahora @fields es ("jdiaz","","517", "100", "Jose Luis Diaz",
                    "/var/home/jdiaz" , "/bin/bash")
```

Cambiar por `/:+/` en el ejemplo para evitar tener el segundo elemento del array en blanco.

```
$line = join(/-/, @fields);
# ahora $line es "jdiaz--517-100-Jose Luis Diaz-/var/home/jdiaz-/bin/bash"
```

9 Modulos

Basicamente lo que define la clasula **package** es similar a un namespace en C++.

```
package A;
$a = 10;
package B;
$a = 20;

package A;
print $a;
```

esto va a imprimir

```
10
```

Las ultimas dos lineas pueden ser reemplazadas por:

```
print $A::a;
```

La siguiente linea:

```
Class->method(@args)
```

Intenta invocar:

```
Class::method("Class", @args);
```

Esto es util para definir un modulo de la siguiente manera:

```
package Test;

sub new {
        my $arg = $_[0];
        my $var = "just a test";
        return bless \$var, $arg;
}

sub print {
        my $arg = $_[0];
        print "Package " . $ { $arg } . "\n";
}


1;
```

La siguientes lineas:

```
Test->print();
$test = Test->new();
$test->print();
```

Tienen la siguiente salida:

```
Package
Package just a test
```

Ya que **bless** devuelve un una referencia al modulo instanciando el valor del escalar en cuestion. Esto se utiliza para guardar informacion asociada al modulo.

10 OOprograming

Una forma de heredar otra "modulos" seria la siguiente:

```
package Animal;
sub speak {
     my $class = shift;
     print "a $class goes ", $class->sound, "!\n"
}

package Cow;
@ISA = qw(Animal);
sub sound { "moooo"; }
```

Cuando se llama Cow→s, al no encontrar dentro del package Cow, el metodo asocidado intenta hacer @ISA::speak, dentro de Animal, $class es "Cow" (el primer argumento) despues intenta hacer $class→sound, va a buscar por Cow→sound, entonces imprime "a Cow goes moooo!".

Un ejemplo un poco mas interesante de como trabajar con objetos:

```
package StoreItem;

my $_sales_tax = 8.5;  # 8.5% added to all components's post rebate price

sub new {
    my ($pkg, $name, $price, $rebate) = @_;
    bless {
        _name => $name, _price => $price, _rebate => $rebate
    }, $pkg;
}

# Accessor functions
sub sales_tax {shift; @_ ? $_sales_tax = shift : $_sales_tax};

sub name {my $obj = shift; @_ ? $obj->{_name} = shift : $obj->{_name}};

sub rebate {my $obj = shift; @_ ? $obj->{_rebate} = shift : $obj->{_rebate}};

sub price {my $obj = shift;
              @_ ? $obj->{_price} = shift
                 : $obj->{_price} - $obj->rebate
}
```

```
sub net_price {
    my $obj = shift;
    return $obj->price * (1 + $obj->sales_tax / 100);
}

1;

#-----------------------------------------------------------------------------
package Component;
@ISA = qw(StoreItem);
1;

#-----------------------------------------------------------------------------
package Monitor;
@ISA = qw (StoreItem);
# Hard-code prices and rebates for now
sub new { $pkg = shift; $pkg->SUPER::new("Monitor", 400, 15)}
1;

#-----------------------------------------------------------------------------
package CDROM;
@ISA = qw (StoreItem);
sub new { $pkg = shift; $pkg->SUPER::new("CDROM", 200, 5)}
1;

#-----------------------------------------------------------------------------
package Computer;
@ISA = qw (StoreItem);

sub new {
    my $pkg = shift;
    my $obj = $pkg->SUPER::new("Computer", 0, 0); # Dummy value for price
    $obj->{components} = [];        # list of components
    $obj->components(@_);
    $obj;
}

sub components {
    my $obj = shift;
    @_ ? push (@{$obj->{components}}, @_)
       : @{$obj->{components}};
}

sub price {
    my $obj = shift;
    my $price = 0;
    foreach my $component ($obj->components()) {
        $price += $component->price();
    }
    $price;
}
```

11 Referencias

- Programming Perl, 3nd Edition. Larry Wall, Tom Christiansen & Randall Schwartz. O'Reilly & Associates.
- Perl Resource Kit – UNIX Edition. Larry Wall et al. O'Reilly & Associates.
- Advanced Perl Programming. Siriam Srinivasan. O'Reilly & Associates.
- Etc., etc., etc.