1 Perl

1.1 Sumario

1.2 Introdución a Perl

Instalación de Perl en Windows:

- Baixar e instalar Perl para Windows

Para ver a versión de Perl instalada:

```
C:\scripts> perl -v

This is perl, v5.10.1 built for MSWin32-x64-multi-thread
(with 2 registered patches, see perl -V for more detail)

Copyright 1987-2009, Larry Wall

Binary build 1007 [291969] provided by ActiveState http://www.ActiveState.com
Built Jan 27 2010 14:12:21

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl".  If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.
```

O meu primeiro *script* en perl:

Creamos o arquivo **proba.pl** e o editamos có noso editor favorito, o contido do arquivo debe ser o seguinte:

```
#!/usr/bin/perl
print "Ola Caracola!\n"
```

Logo para executalo: - Se é dende Linux hai que configurar o arquivo como executable:

```
# chmod 755 proba.pl
# ./proba.pl
```

E logo executalo:

```
# ./proba.pl
```

- Se é dende Windows, acceder á liña de comandos e escribir:

```
> perl proba.pl
```

- Un exemplo mais elaborado será o seguinte:

```
print "Valor del inmueble? ";
chop( $valor = <STDIN> );
$comision = $valor * 0.25;
print "Comision = $comision\n";
```

- Para poñer unha línea de comentarios empregar o símbolo **#**.

1.3 Variables e Tipos

Perl emprega variables para representar cousas que poden ter diferentes valores. En Perl temos tres grandes grupos de variables: escalares ($), arrays (@) e *hashes* (%). Ter en conta que se distingue entre minúsculas e maiúsculas.

1.3.1 Escalares

◊ Esta variable pode conter calquera tipo de información xa sexan números, letras, cadenas de texto, signos, código ASCII, etc.
◊ Para definir unha variable escalar todo o que temos que facer é:

```
$variable
```

◊ Por escalares enténdense valores numéricos e alfanuméricos, no caso dos valores alfanuméricos (texto) este escríbese entre comiñas.
◊ Ao final da asignación débese escribir ";".
◊ As variables non poden comezar por un número.

Exemplos:

```
# Número enteiro
$num1=127;
# Número decimal
$num2=2.5;
# Número Octal
$Octal1=044;
# Número Hexadecimal
$Hex1=0xFF;
# Interpolar variables dentro doutra
$Frase1="O usuario $nome $apelidos de $idade anos...";
```

Pódese definir tamén o alcance (*scope*) da variable: Se non se indica nada as variables son sempre globais.

◊ **my** - Unha variable declarada con **my** non pertence a ningún paquete, pertence só ao seu bloque.
◊ **our** - Para definir variables globais empregamos a palabra clave **our**.

```
...
my $age = 40;
our $count = 1;

...
```

1.3.2 *Arrays*

◊ Comezan có signo @.
◊ Conteñen unha lista de valores escalares (números e textos).

Exemplos:

```
# Definir un array
@provincia=("A Coruña","Lugo","Ourense","Pontevedra");
# Devolve "A Coruña"
$provincia[0]
# Devolve "Lugo"
$provincia[1]
# Cambiar o valor gardado
$provincia[0]="Oviedo";
# Agora @provincia=("Oviedo","Lugo","Ourense","Pontevedra")
```

◊ Como vemos, para a asignación emprégase o carácter **@** e para acceder a un valor individual, tanto para ler como para escribir nel, emprégase o signo **$**.
◊ Os *arrays* tamén aceptan interpolación con outros *arrays*.
◊ Existen algunhas funcións que axudan a poñer ou sacar elementos da lista:

```
# Saber o índice do último elemento da lista:
$#provincia
# E mostrar por pantalla ese último elemento da lista:
print $provincia[$#provincia];
# Colocar novos elementos ao final da lista:
push(@provincia, 'Cadiz', 'Sevilla');
# Sacar o último elemento do ''array'' e asignarllo a unha variable:
$oultimoera=pop(@provincia);
# Remprazar o primeiro elemento do ''array'':
unshift(@provincia, 'Oviedo');
# Sacar o último elemento do ''array'' e asignarllo a unha variable:
$oprimeiroera=shift(@provincia);
```

1.3.3 Hashes

◊ Trátase de matrices que se referencian polo par clave/valor.
◊ Comezan có signo **%**.
◊ Os valores introdúcense mantendo unha relación a pares, o primeiro valor é a clave do seguinte.

```
# Definir un hash:
%dia=(Lun,"Luns",Mar,"Martes",Mer,"Mércores",Xov,"Xoves",Ven,"Venres",Sab,"Sábado",Dom,"Domingo");
# Outro modo de definir o mesmo hash:
%dia=(
Lun=> "Luns",
Mar=> "Martes",
Mer=> "Mércoles",
Xov=> "Xoves",
Ven=> "Venres",
Sab=> "Sábado",
Dom=> "Domingo"
);
# Acceder a un dos elementos:
print $dia{Lun};
# Modificar o contido dun dos elementos do hash:
$dia{Lun}=LUNS;
```

◊ Os *hashes* tamén teñen unha serie de funcións asociadas, que facilitan a súa utilización:

```
# Para borrar un par clave/valor utilizamos a función delete:
delete($dia{Lun});
# A función values mostra todos os valores da matriz asociada pero sen manter un orden determinado:
print values(%dia);
# A función keys mostra todas as claves do hash sen manter un orden determinado:
print keys(%dia);
# Para mostrar os elementos dun xeito ordenado empregaremos a función sort:
print sort(values(%dia));
print sort(keys(%dia));
```

1.4 Expresiones regulares en Perl

Las expresiones regulares en Perl son una parte importante del trabajo con los operadores *m//*, *s///*. *qr//* y *split*.

Perl es ampliamente conocido por su excelente procesamiento de textos, y las expresiones regulares son uno de los factores que le dan esta fama.

A partir de ahora "expresión regular" será abreviado como "regexp".

- **Coincidencias de una palabra simple:**

  Para buscar la coincidencia de un *string* en una frase:

```
#!/usr/bin/perl
if ("Hola mundo" =~ /mundo/) {
```

```
        print "Coincidencia!!!\n";
}
else {
        print "No hay coincidencias!!!\n";
}
```

En este caso sí hay coincidencia.

- **NO coincidencia de una palabra simple:**

En este caso lo haríamos del siguiente modo:

```
#!/usr/bin/perl
if ("Hola mundo" !~ /mundo/) {
        print "Coincidencia!!!\n";
}
else {
        print "No hay coincidencias!!!\n";
}
```

En este caso NO hay coincidencia.

1.5 Aprender con exemplos

1.5.1 Pedir nombre, guardarlo en variable y luego imprimirlo. Ver también las diferencias entre utilizar las " " y las ' '

```
#!/usr/bin/perl
#Ejemplo sencillo
#
print"¿Cómo te llamas? ";
chop( $nombre = <STDIN> );

print "Tu nombre es $nombre\n";
print 'Tu nombre es $nombre\n';

#END
```

Salida:

¿Cómo te llamas? Manuel
Tu nombre es Manuel
Tu nombre es $nombre\n

Aclaraciones:

◊ Función print.
◊ Función chop.

1.5.2 Trabajo con variables de entorno y variables de usuario

```
#!/usr/bin/perl
#Ejemplo de variables
#
$saludo = "Bienvenido Sr. ";
$nombre = getlogin();
print $saludo . " " . $nombre . "\n";

# Función específica para obtener
# toda la información de un usuario: getpwnam
($name, $pass, $uid, $gid, $quota, $comment, $gcos, $dir, $shell, $expire) = getpwnam($nombre);
print "Su directorio home es $dir\n\n";

# Las variables $ también son muy útiles
# Y también el carácter de escape \
print "Está usted ejecutando el script \"$0\"\n";

# Argumentos:
```

```
$NumArg = @ARGV;
print "con $NumArg argumentos, que son: \n";
$i = 0;
while ($i < $NumArg) {
print "\t- Argumento Núm. ", $i, " ---> ", $ARGV[$i], "\n";
$i++;
}
print "\n";

# Para el PID de un proceso:
print "El PID de este proceso es " . getpgrp(0) . "\n\n";

# Ejecutar una función de shell bash desde el script perl:
$fecha=`/bin/date`;
print "Por último le indico la fecha actual: $fecha \n";

# Si la última sentencia se ejecutó con éxito devolverá un 0
print "La función print se ejecutó: $?\n";

#END
```

Aclaraciones:

- ◊ Variables predefinidas
- ◊ Usuarios, Grupos y Procesos

1.5.3 Trabajar con la función tr

```
#!/usr/bin/perl
#Ejemplo con tr
#
while (<STDIN>){
        $texto = $_;
        $texto =~ tr/a-z/A-Z/;
        print "$texto\n";
}
```

Aclaraciones:

- ◊ Función tr

1.6 Enlaces interesantes

- Manuales de Perl