programa de BECAS
DGTIC
de FORMACIÓN en TIC

Seguridad Informática

Programación con Perl

Cervantes Varela Juan Manuel
Vallejo Fernández Rafael Alejandro

Función qx

- Una alternativa a utilizar los ``` para ejecutar comandos del sistema operativo es utilizar la función qx.

```
qx comando;
qx DELIM comando DELIM;
DELIM puede ser cualquier caracter
```

- Al igual que con ``` podemos manipular la salida en un contexto escalar o de lista

```
$salida = qx”comando”;
@salida = qw(comando);
```

- https://www.webtrafficexchange.com/perl-string-quotation-operators-q-qq-qr-and-qx

\1 y $1

\1

Se guarda en él el grupo encontrado desde el bloque de match, y es accesible únicamente desde la misma línea de la expresión regular.

$1

Se guarda en él el grupo encontrado después del bloque de match, y es accesible después del match y en el resto del código hasta que su valor cambie.

Ejercicio 14

Realizar un programa que obtenga todas los enlaces de una pagina web y los guarde en un archivo txt.

Práctica 4

Hacer un crawler en perl, que analice hasta dos niveles y genere un reporte con los enlaces encontrados en estructura de arbol.

Entrada y salida de directorios

Del mismo modo que existen handles para trabajar con archivos, también existen otros para trabajar con directorios.

Los principales son:

- opendir (DIR, $directory)

Abre un directorio

- readdir DIR

Lee el contenido de un directorio, linea por linea o todo el contenido en una arreglo (dependiendo su contexto).

- closedir(dir)

Cierra un directorio

- Chdir($dir)

Cambia el directorio de trabajo.

Entrada y salida de directorios

```
opendir DIR, ".";

# Lectura en contexto escalar
$cont = readdir DIR;
print "$cont\n";

# lectura en contexto de lista
#@cont = readdir DIR;
#print "@cont\n";

close DIR;
```

```
use Cwd;

print cwd, "\n";

chdir ("./nuevo_directorio");

print cwd, "\n";

open FILE, ">poc.txt";
print FILE cwd;
close FILE;

system("touch poc1.txt");
```

Módulos (continuación)

Cuando se tiene código que quiere reutilizarse en otro script podemos meter ese código en un “módulo de Perl” y posteriormente utilizarlo en otro mediante la sentencia “use”.

- **Módulo**

Su contenido es cualquier código de perl válido y al final de todo el código debe ponerse el valor “1” para no generar errores de compilación.

La extension del modulo debe ser “.pm” (abreviación de perl module) para que los scripts puedan importarlos.

Módulos (continuación)

Por convención, la primera línea de un módulo es la declaración de “package” que indica que el espacio de nombres del paquete es el mismo que el nombre del módulo.

Después de la declaración de paquetes viene el uso de módulos externos, declaración e inicialización de variables, subrutinas y al final:

1;

Este valor indica que devuelve un valor verdadero en lugar de una cadena vacía, que sería falso.

Módulos (continuación)

Los nombres de los módulos creados por nosotros tienen la convención de que deben ser mezclados, ya que todo en minúsculas está reservado para módulos propios de Perl y en mayúsculas puede ser confuso.

Módulos (continuación)

Archivo: User.pm

```perl
#!/usr/bin/perl
package User;

use warnings;

sub habla { print "Hola a todos\n"; }

sub nombre { print "Nombre: ". shift ."\n"; }

sub info {
    print "Descripcion: ". shift . "\n";
    print "Edad: ". shift . "\n";
    print "Cumpleaños: " . shift . "\n";
}

1;  # Siempre debe ser la última instrucción del módulo.
    # En caso de no ponerlo siempre dará un error de compilación
    # porque devolvería un valor falso "".
```

Use

La sentencia “use” permite importar el código de un módulo para poder utilizarlo en el script.

La sintaxis es:

use <NombreModulo>;

Cuando son módulos creados por nosotros, los dos archivos (el módulo .pm y el script .pl) deben estar en el mismo directorio.

Nota: En algunos casos no funciona cuando están en el mismo directorio, por lo que es necesario indicarlo explícitamente.

Use

Para acceder a los elementos de un módulo es necesario indicar el nombre del módulo :: elemento.

Archivo: usamodulo.pl

```
1   ### Archivo: usamodulo.pl
2   #!/usr/bin/perl
3   #use lib './';
4   #BEGIN { push(@INC, './'); }
5   use User;
6
7   User::nombre ("root");
8   User::habla;
9   User::info ("Superusuario", "30", "11 de septiembre de 1990");
```

@INC

Es el arreglo global donde se encuentran definidos los directorios para buscar los módulos.

Si se requiere añadir un directorio es necesario añadirlo mediante el bloque BEGIN y con la operación push.

```
BEGIN { push(@INC, './'); }
```

Use lib

Esta instrucción permite añadir también directorios al arreglo @INC sin la necesidad del bloque BEGIN.

```
use lib './';
```

Para interpretar “~” como bash, se puede utilizar la función glob(); de Perl que lo traduce.

```
use lib glob('~/modulos');
```

Bloques begin / end

Existen los bloques de código denominados BEGIN y END, que actúan como constructores y destructores respectivamente.

BEGIN

Se ejecuta después de que se cargue y compile el script perl, pero antes de que se ejecute cualquier otra instrucción.

END

Se ejecuta justo antes de que salga el intérprete de Perl.

Los bloques BEGIN y END son particularmente útiles al crear módulos Perl.

Bloques begin /end

```
print "Ejecución de código principal 1\n";

BEGIN {
    print "Esto es el bloque BEGIN\n"
}

print "Ejecución de código principal 2\n";

END {
    print "Este es el bloque END\n"
}

print "Ejecución de código principal 3\n";
```

Sockets

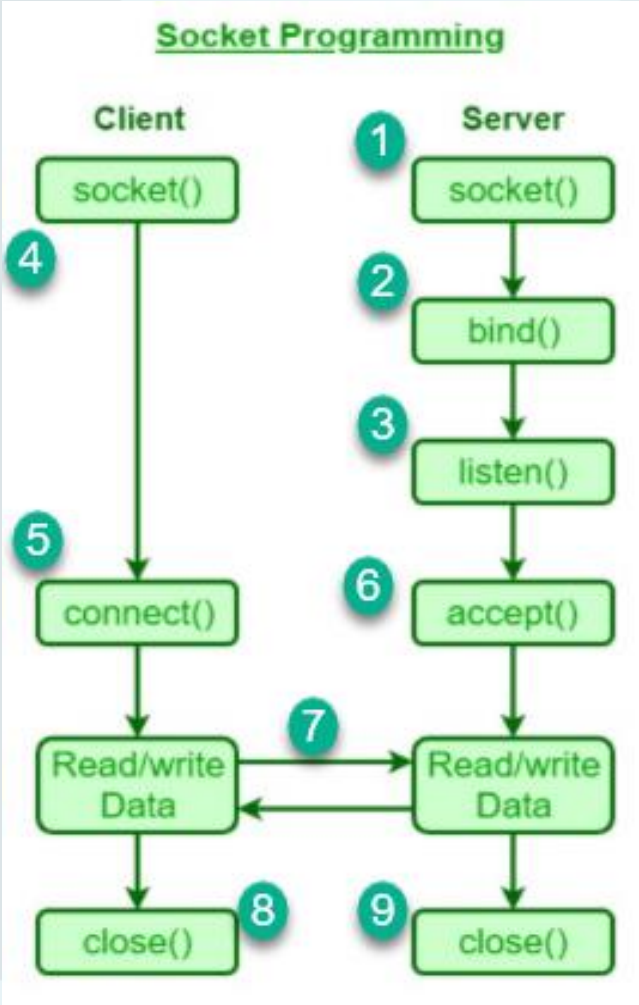
La programación de sockets es una forma de conectar dos nodos en una red para comunicarse entre sí.

Funcionan mediane una arquitectura Cliente-Servidor, donde un Cliente se conecta, envía mensajes al servidor y el servidor procesa la información, pudiendo o no retornar una respuesta.

Pueden generarse sockets TCP/UDP dependiendo de la funcionalidad y utilidades que el usuario les quiera dar.

```
use Socket;
```

Ejemplo de proceso de comunicacion entre sockets cliente-servidor 1 a 1



Sockets - funciones importantes

- New

Para crear un nuevo puerto se usa la función new, especificando como argumentos y usando el operador flecha, las configuraciones necesarias.

```
# Servidor
my $socket = new IO::Socket::INET (     # IO::Socket::INET->new (
    LocalHost => 'localhost',
    LocalPort => '12345',
    Proto => 'tcp',                     # 'tcp', 'udp', IPPROTO_TCP, IPPROTO_UDP
    Listen => 1,                        # 5 por defecto
    ReuseAddr => 1,
) or die "Can't bind : $@\n";
```

```
# Cliente
my $socket = new IO::Socket::INET (     # IO::Socket::INET->new (
    PeerAddr => 'localhost',
    PeerPort => '12345',
    Proto => 'tcp',
) or die "Can't connect : $@\n";
```

Sockets - funciones importantes

Para este modulo, con el simple hecho de crear el socket se pone en modo escucha (bind, en el caso del servidor) o se conecta (connect, en el caso del cliente), por lo que no se detallarán esas funciones.

- Accept

Normalmente va dentro de un bucle genera un nuevo descriptor de archivo basándose en el descriptor del socket cuando una conexión entrante es aceptada.

```
my $new_socket = $socket->accept();
```

Sockets - funciones importantes

- Close

Termina la conexión y ciera el descriptor de archivo del socket.

```
close($socket);
```

Para enviar y recibir datos se hace de la misma forma en que se escriben o leen datos de un archivo

```
# Enviar
print $socket "Mensaje";
# Recibir
$mensaje = <$socket>;
```

Sockets

```perl
my $socket = new IO::Socket::INET (
    LocalHost => 'localhost',
    LocalPort => '6666',
    Proto => 'tcp',
    Listen => 1,
    Reuse => 1,
) or die "no se puede crear el socket: $!\n";

print "Esperando datos del cliente\n";
my $new_socket = $socket->accept();
while(<$new_socket>) {
    print $_;
}

close($socket);
```

```perl
use strict;
use warnings;
use IO::Socket;

my $socket = new IO::Socket::INET (
    PeerAddr => 'localhost',
    PeerPort => '12345',
    Proto => 'tcp',
) or die "No se puede crear el socket: $!\n";

print "Ingresa cadena a enviar:\n";
my $data = <STDIN>;
chomp $data;
print $socket "Cadena enviada: '$data'\n";

close($socket);
```

Ejercicio

- **Ejercicio 16**

Utilizando el ejercicio 15 implementar la calculadora mediante sockets donde el cliente envía la expresión a evaluar al servidor y el servidor le devuelve el resultado al cliente.

Práctica

Práctica 5

Realizar una calculadora científica que contenga las funciones:

Seno,Coseno,Tangente,Cotangente,Operaciones aritmeticas(/ + - *), Raiz cuadrada, Potencia (Sin usar función predefinida), Factorial (Sin usar función predefinida).

- Tomar en cuenta la jerarquía de operaciones y uso de ().
- Los datos se ingresan por entrada estándar o como argumento (elección libre).

Práctica

Práctica 6

Instalar y configurar un servidor de IRC

Instalar un cliente y conectarse al servidor IRC a través de la terminal

Conectarse a un canal (el nombre de canal es su nombre de usuario)

Generar un bot simple de IRC