programa de BECAS
DGTIC
de FORMACIÓN en TIC

Seguridad Informática

Programación con Perl

Cervantes Varela Juan Manuel
Vallejo Fernández Rafael Alejandro

Funciones de interacción con sistema

Funciones para manipulación de archivos y directorios propias de perl.

- **Unlink('ruta_archivo')**

Borra un archivo

- **Rename('nombre antiguo', 'nombre nuevo')**

Cambia el nombre de un archivo

- **Mkdir('Dir', permisos_octal)**

Crea un directorio

- **Rmdir('Directorio')**

Borra un Directorio Vacio

- **Chmod(permisos, archivo)**
- **Chmod(permisos, archivo1,…,archivon)**

Modifica los permisos de 1 a n archivos

Funciones de interacción con sistema

```
archivo1.txt
archivo2.txt
archivo3.txt
directorio
└──archivo.txt
directorio1
```

```
drwxr-xr-x  4 kali kali  4096 Sep 10 14:32 .
drwxr-xr-x 10 kali kali  4096 Sep 10 14:05 ..
-rw-r--r--  1 kali kali     0 Sep 10 14:07 archivo1.txt
-rw-r--r--  1 kali kali     0 Sep 10 14:07 archivo2.txt
-rw-r--r--  1 kali kali     0 Sep 10 14:08 archivo3.txt
drwxr-xr-x  2 kali kali  4096 Sep 10 14:07 directorio
drwxr-xr-x  2 kali kali  4096 Sep 10 14:32 directorio1
-rw-r--r--  1 kali kali   235 Sep 10 14:26 funcs.pl
-rw-------  1 kali kali 12288 Sep 10 14:32 .funcs.pl.swp
```

Funciones de interacción con sistema

```php
unlink("archivo1.txt");
rename("archivo2.txt", "nuevo_archivo.txt");
mkdir("nuevo_directorio", 0777);
rmdir("directorio");
rmdir("directorio1");
chmod(0777, "nuevo_archivo.txt");
chmod(0700, "archivo3.txt", "nuevo_archivo.txt");
```

Funciones de interacción con sistema

```
archivo3.txt
directorio
└──archivo.txt
nuevo_archivo.txt
nuevo_directorio
```

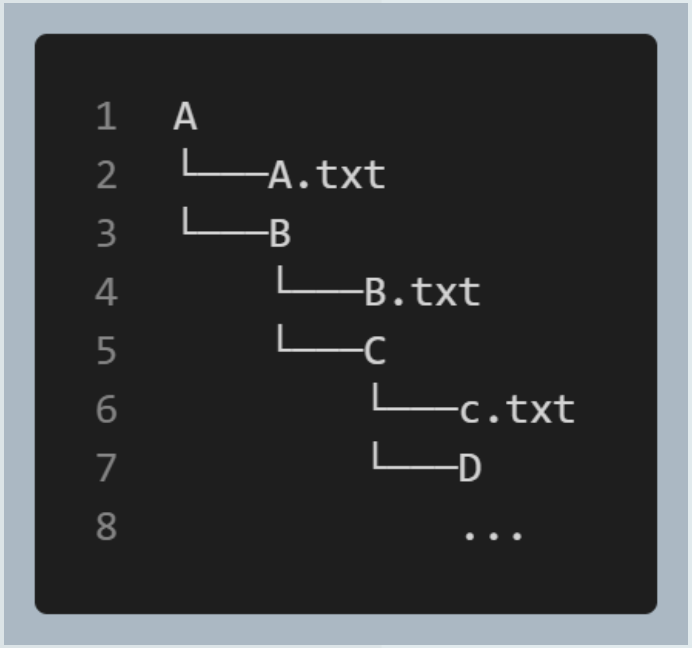
```
drwxr-xr-x  4 kali kali  4096 Sep 10 14:40 .
drwxr-xr-x 10 kali kali  4096 Sep 10 14:05 ..
-rwx------  1 kali kali     0 Sep 10 14:08 archivo3.txt
drwxr-xr-x  2 kali kali  4096 Sep 10 14:07 directorio
-rw-r--r--  1 kali kali   229 Sep 10 14:40 funcs.pl
-rw-------  1 kali kali 12288 Sep 10 14:40 .funcs.pl.swp
-rwx------  1 kali kali     0 Sep 10 14:07 nuevo_archivo.txt
drwxr-xr-x  2 kali kali  4096 Sep 10 14:40 nuevo_directorio
```

Funciones de interacción con sistema

Ejercicio 9

Crear de forma anidada directorios cuyo nombre vayan de A a la K, y en cada carpeta un archivo txt con el mismo nombre de la carpeta contenedora.

Ej.



Subrutinas de ordenamiento - grep

Devuelve una lista con los valores encontrados en un arreglo mediante el uso de la expresión.

```
1  @nombres = ("Carlos", "Regina", "Andrea", "Pepe", "Rodrigo");
2  @res = grep(/^R/, @nombres);
3  print "@res\n";      # Regina Rodrigo
4  @nombres = (1, "dos", 3, "cuatro", 5);
5  @res = grep(/\d/, @nombres);
6  print "@res\n";      # 1 3 5
```

Funciones de ordenamiento - map

Ejecuta una expresión en cada elemento de una matriz y devuelve una nueva matriz con los resultados.

```perl
my @numbers = (1..5);
print "@numbers\n";          # 1 2 3 4 5
my @squares = map {$_ ** 2} @numbers;
print "@doubles\n";          # 1 4 9 16 25
```

```perl
@nombres = ("Carlos", "Regina", "Andrea", "Pepe", "Rodrigo");
my %tams = map {$_ => length($_)} @nombres;
foreach (keys %tams) {
    print "$_ => $tams{$_}\n";
}
```

Funciones de ordenamiento - map

```
1  @nombres = ("Carlos", "Regina", "Andrea", "Pepe", "Rodrigo");
2  my %tams = map {$_ => $_ =~ /^.[ae].*/ ? uc($_) : $_ } @nombres;
3  foreach (keys %tams) {
4      print "$_ => $tams{$_}\n";
5  }
```

Expresiones regulares

El operador más utilizado es match.

- **Grupos**

Es posible capturar cadenas que coinciden con los patrones especificados entre paréntesis. Se les llama grupos.

Los valores capturados se asignan a las variables especiales $1,$2,$3,…

```
1  $alumno = "Alumno: Perez Vargas Patricio Identificador: 1000120";
2  $alumno =~ m#Alumno: (\w+) (\w+) (\w+) Identificador: (\d+)#;
3  $apaterno = $1;
4  $amaterno = $2;
5  $nombre = $3;
6  $id = $4;
7  print "El ID:$id es del alumno: $nombre $apaterno $amaterno\n";
```

Expresiones regulares

- **Grupos**

También es posible agrupar sin capturar mediante (?:patron).

Útil cuando se require buscar otro sub-patrón dentro de la cadena.

```
1  $saludo = "buenas tardes";
2  $saludo = "buenos dias";
3  $saludo =~ m#(buen(?:os|as)) (\w+)#;
4  #$saludo =~ m#(buen(os|as)) (\w+)#;
5  print "1.Se captura: $1 $2\n";
6  #print "2.Se captura: $1 $2 $3\n";
```

Expresiones regulares

- **Variables especiales**

Estas variables pueden asignarse a un escalar (string).

$1, $2, $3, …

En los grupos se vio una de estas variables, las cuales son $1,$2,$3,… cuyo valor es el grupo capturado.

\1, \2, \3, …

Similar al anterior, pero captura los grupos durante el match. Útil cuando se quiere reformatear una cadena mientras se parsea.

$&

Es la variable que contiene la cadena de coincidencia con el patrón.

Expresiones regulares

- **Variables especiales**

$`

Es la variable que contiene la cadena que está antes (izquierda) de la coincidencia con el patrón.

$’

Es la variable que contiene la cadena que está después (derecha) de la coincidencia con el patrón.

Expresiones regulares

- **Variables especiales**

```
1  $cad = "Se requieren de 20 numeros enteros";
2  $cad =~ m#(\d+)#;
3  print "Cadena: $cad\n";
4  print "Antes match: $`\n";
5  print "Match: $&\n";
6  print "Después match: $'\n";
```

Expresiones regulares

- **Caracteres \Q y \E**

Lo que se ponga (cadena y valor de variable) entre estos caracteres será interpretado tal cual.

Ejemplo:

```
1  $str = "¡Hola!.¡adios!"; # ¡Hola!.¡adios!
2  print "Patron: $str\n";
3  $prueba = "¡Hola!/¡adios!";
4  print "Prueba: $prueba\n\n";
5  print "Sin \\Q \\E (interpetra el patrón como REGEX): ";
6  $prueba =~ m#$str# ? print "Coincide" : print "NO coincide"; # Coincide
7  print "\nCon \\Q \\E (interpreta el patrón como una cadena): ";
8  $prueba =~ m#\Q$str\E# ? print "Coincide\n" : print "NO coincide\n"; # NO coincide
```

Clases de caracteres

```
[def.]      Caracteres d,e,f o .
[d-f]       Lo mismo que [def].
[a-z]       Letras minúsculas.
[^bx]       Todo excepto b o x.
\w          caracteres de palabras: [a-zA-Z0-9_].
\d          Números: [0-9]
\s          [\f\t\n\r ]
\W          Lo que no sean palabras; equivalente a: [^\w]
\D          Lo que no sean numeros; equivalente a: [^\d]
\S          Lo que no sean espacios; equivalente a: [^\s]
[:class:]   POSIX character classes (alpha, alnum...)
\p{...}     Definiciones de clase de caracteres
            Unicode (IsAlpha, IsLower, ...)
\P{...}     Complemento de clase de caracteres Unicode
```

Ejercicio

- **Ejercicio 10**

Hacer un script que obtenga información de los usuarios del equipo que cuenten con un intérprete de comandos.

Por cada usuario imprimir su nombre, ID de usuario, ID de grupo, su directorio home y el intérprete de comandos que utiliza.

- **Ejercicio 11**

Hacer un script que valide direcciones de correo electrónico leídos desde un archivo.

Una vez validados, agrupar por dominio los correos electrónicos y mostrarlos en salida estándar.

Práctica

Práctica 2

Utilizando el reporte generado de la practica anterior, generar estadisticas detalladas:

- Número de hosts que utilizan cada puerto detectado
- Número de puertos segun su clasificación (bien conocidos, registrados y de proposito general)
- Número de puertos segun su estado (abierto, filtrado, etc)
- Número de puertos según el protocolo (tcp y udp)
- Número de puertos activos por host
- SO de cada host (usar ping)
- Número de host por SO (Linux, Windows, MAC)

Guardando los resultados en un nuevo archivo.

Práctica

Práctica 3

Con el archivo generado en la práctica anterior, generar las gráficas necesarias.

Hint (Instale módulos de CPAN para generar graficas).

Pragmas

Son modificaciones a las directivas del compilador de perl modificando como debe actuar al compilar un programa.

. Para activarlas se hace de la siguiente forma:

```
1   use <directiva>;
```

Donde <directiva> es el pragma a habilitar. Existen 5 principales.

Pragma strict

Genera errores en caso de que se encuentren prácticas de programación inseguraso dificiles de debuggear. Los principales ejemplos son los siguientes:

- Variables no inicializadas o que no sean definidas con ámbito léxico (my, our).
- Referencias irreales (donde se busque hacer referencias donde no existen),
- Palabras que no tienen comillas sin ser subrutinas o identificadores de archivos.

Su alcance (scope) abarca el archivo o el bloque de código en el que se encuentre.

Pragma warnings

Trabaja del mismo modo que el pragma strict, pero unicamente mostrando alertas en lugar de manejarlos como errores.

Ambos son flexibles pudiendo especificar que tipo de alertas habilitar y en que partes del programa.

Existen muchos pragmas más, pero estos son los dos principales y de uso recomendado para no incurrir en malas prácticas o programación insegura.

* https://perldoc.perl.org/index-pragmas.html

Pragmas

```
1  use strict;
2  use strict "vars";
3  use strict "refs";
4  use strict "subs";
5
6  {
7      use strict;
8      no strict "vars";
9  }
```

```
1  use warnings;
2  use warnings "all";
3
4  {
5      no warnings;
6  }
```

Contextos

En perl, las llamadas a funciones, los términos y las declaraciones tienen explicaciones inconsistentes que dependen de su contexto.

Existendos contextos importantes, los cuales son:

- Contexto escalar
- Contexto de lista

Contexto escalar

Cuando se habla de un contexto escalar se refiere a que los datos serán manejados como escalares.

Es por esta razón que al asignar $tam = @array nos devuelve la cantidad de elementos del arreglo.

Es possible indicar de forma explicita que algo sea evaluado como un dato escalar mediante el operador “scalar”.

```
1   $tam = scalar @array;
```

Contexto de lista

En un contexto de lista, Perl proporciona la lista de elementos, donde la lista puede tener cualquier cantidad de elementos, uno solo o ninguno.

```
1  @array = (1..9);
2  @array = grep(/^s/, @array);
3  @otro_array;
```

```
1  @nums = (1..9);
2  $uno, $dos, $tres = @nums;
3  print "$uno, $dos, $tres\n";
4  ($uno, $dos, $tres) = @nums;
5  print "$uno, $dos, $tres\n";
```

Localtime

Funcion de perl que devuelve la fecha dado un epochtime, retorna un arreglo con la siguiente estructura:

```
1  ($segundos, $minutos, $horas, $dia_mes, $mes, $anio, $dia_semana, $dia_anio, $bisiesto)
```

* NOTAS:

- El año devuelve la cantidad de años transcurridos a partir de 1900.
- El mes se devuelve en un rango de 0..11
- El día se devuelve en un rango de 0..364 (0..365 en años bisiestos)

```perl
@localtime = localtime();
print "@localtime\n";
print scalar localtime(), "\n";
```

```
kali@kali:~/14g/funcs$ perl lt.pl
5 58 15 10 8 120 4 253 1
Thu Sep 10 15:58:05 2020
kali@kali:~/14g/funcs$
```

Ejercicio 12

Genere una función date
que devuelva la fecha con el siguiente formato

AAAA-MM-DD Hora:Minutos:Segundos

Ej. 2020-12-26 16:54:23

Ejercicio 13

Genere una función date que devuelva la fecha con el siguiente formato

Día_semana DD de Mes de AAAA,
Horas:Minutos:Segundos

Ej. Sabado 26 de Diciembre de 2020, 16:54:23