

Perl: Archivos Directorios Procesos Formatos

Archivos

- La lectura desde un file handle es similar a la lectura desde la entrada estándar:

```
open(F, "listado.txt");
while (<F>){
    chomp;
    print "El archivo contiene: $_\n";
}
```

- De la misma manera se puede escribir en un archivo:

```
print F $linea;
```

Archivos

- Copiar el contenido de un archivo a otro:

```
open (IN, $a) || die "Error $a: $!\n";
open (OUT, ">$b") || die "Error $b: $!\n";
while (<IN>){
    print OUT $_;
}
close(IN) || die "No puedo cerrar $a: $!\n";
close(OUT) || die "No puedo cerrar $b: $!\n";
```

Archivos

- Dependiendo del contexto se puede leer un escalar o una lista desde un archivo:

```
$linea = <FILE>; # lee una línea
@lineas = <FILE>; # lee TODAS las líneas
```

```
# ejemplo de comando cat
@lista = <>;
print @lista;
```

```
#cat resumido aprovecha contexto de lista
print <>;
```

Directorios

- Para “moverse” entre directorios existe la función “chdir”:

```
chdir("/pepino")
|| die "no puedo ir a /pepino ($!)";
```

- Retorna verdadero en caso exitoso, los paréntesis son opcionales
- Cada proceso tiene su propio CWD (directorio de trabajo actual)
- Cuando se lanza un nuevo proceso, hereda el directorio del padre

Directorios

- Si un programa Perl modifica el directorio de trabajo, esto no afectará al shell que ejecuta el proceso Perl
- “chdir” sin parámetros, usará el directorio “home”, tal como el comando “cd”
- Tal como existen los “file handle”, acá se tienen los “directory handle”
 - Se recomienda utilizar mayúsculas
 - Se puede tener el file handle “CC” y el directory handle “CC”
 - Representa una conexión a un directorio en particular
 - Permite leer una lista de nombres de archivos que están en el directorio

Directorios

- No se puede utilizar un directory handle para cambiar el nombre a un archivo o borrarlo
- Para abrir y cerrar un directorio:

```
opendir(D, "/home/cc31a")
|| die "Inaccesible /home/cc31a: $!\n";
# operaciones utilizando D
closedir(D);
```

- Tal como “close”, “closedir” es innecesario pero recomendable utilizarlo

Directorios

- Para leer desde un directorio:

```
opendir(D, "/home/cc31a") || die "error: $!\n";
while ($name = readdir(D)){
    print $name, "\n";
}
closedir(D);
```

- Utilizando “sort”, se pueden imprimir ordenados:

```
foreach $name (sort readdir(D)){
    print $name, "\n";
}
```

Directorios

- Comúnmente en un intérprete de comandos como un shell el carácter * representa una lista de nombres de archivos
 - Al ejecutar “rm ” el “*” se reemplaza por los nombres de los archivos
 - “/etc/host*” representa a una lista de archivos que están en el directorio “/etc” y su nombre comienza con “host”
- La expansión de * en una lista de nombres de archivo se conoce como “globbing”
- Perl soporta globbing, ubicando el patrón entre “<” y “>”

Directorios

- Ejemplo:

```
while ($nextname = </etc/host*>){
    print "Un archivo es: $nextname\n";
}
```

- También se permiten múltiples argumentos:

```
@98_99_files = <98* 99*>;
```

- Entrega una lista con los nombres de archivos que comienzan con “99” concatenada con la lista de archivos que comienzan con “98”

Directorios

- Existe una excepción para el patrón <\$var>, la cual debe ser escrita como <\${var}>
 - El problema es que la notación <\$var> es usada para leer una línea desde el file handle “\$var”
- Borrar archivos “.o”:

```
unlink <*.o>;
```

- Verificando errores:

```
foreach (<*.o>){
    unlink || warn "Problemas para borrar $_: $!";
}
```

Directorios

- También se pueden crear y borrar directorios:

```
mkdir("tarea4", 0755)
|| die "No puedo crear tarea4/: $!\n";
```

```
rmdir("tarea4")
|| die "No puedo borrar tarea4/: $!\n";
```

- Cambiar permisos:

```
chmod(0666, "tarea4")
|| die "No cambie permisos tarea4: $!\n";
```

Procesos

- Una forma simple de lanzar procesos es utilizando "system":

```
system("date");
```

- La función "system" hereda desde el proceso padre (programa Perl) los 3 archivos estándares:
 - Entrada y salida estándar y salida de errores
- Ejecutar escribiendo a un archivo:

```
system("date > data.txt")
&& die "No se puede escribir data.txt ${!}\n";
```

Procesos

- Típicamente los comandos retornarán "0" cuando este todo OK
 - Si retorna 0, no se evalúa el segundo argumento del "&&"
 - No se ejecuta "die"
 - Si retorna distinto de cero, quiere decir que falló el comando, por lo que el primer argumento es verdadero y evalúa el "die"
- Si se agrega un "&" al final no se espera el retorno del comando:

```
system "(date; who) > data.txt &";
```

Procesos

- Cuando se utiliza "&":
 - El valor de retorno de "system" es el valor de retorno del shell
 - Si el valor es verdadero indicaría que el proceso se lanzó exitosamente
 - No se puede suponer que el proceso terminó exitosamente
- Se puede llamar a "system" con una lista de argumentos:

```
system "grep 'Oct 5' data.txt";
system "grep", "Oct 5", "data.txt";
```

Procesos

- Otra forma de ejecutar comandos es usar 'backquotes':


```
$now = "La fecha y hora es: `date`";
```

 - La diferencia es que se captura la salida del comando en el programa, incluyendo el salto de línea
- Si se usan backquotes en un contexto de lista, se obtendrán una lista de strings
 - Cada uno vendrá de la salida estándar del comando, finalizado con salto de línea

Procesos

- La salida de who es del estilo:

```
jourzua  tty1      Oct 27 19:30
merlyn   pts/0      Dec 25 19:31
cc31a    pts/1      Sep 11 19:50
```

- Para obtener la salida:

```
foreach $_ (`who`){
    ($who, $where, $when) = /(\S+)\s+(\S+)\s+(.*)/;
    print "$who on $where at $when\n";
}
```

Procesos

- Se pueden lanzar procesos haciéndolos parecer filehandles:

```
open(WHOP, "who|"); #abre "who" para lectura
```

- El "|" al lado derecho del comando indica que no es un archivo y que un comando es ejecutado
- Como el "|" está al lado derecho, el filehandle es de lectura, la salida estándar del comando será "capturada"
- Para el resto del programa WHOP es un filehandle de lectura, por lo que puede utilizar todas las funciones de filehandles

Procesos

- Se podría leer el filehandle a un arreglo:

```
@salidawho = <WHOP>;
```

- De la misma manera, si un comando espera una entrada:

```
open(LPR, "|lpr -Phpalumnos");
print LPR @reporte;
close(LPR);
```

- Abrir un proceso con un filehandle, permite que el comando se ejecute en paralelo con el programa Perl

Procesos

- Al hacer "close" del filehandle del proceso se obliga a a esperar a que el proceso termine
 - Si no se llama a "close" el proceso podría seguir ejecutando por más tiempo que el programa Perl
- Al llamar un proceso para escritura, se cambia la entrada estándar de comandos por el filehandle
- Se pueden redirigir todos los mensajes desde el comando:

```
open(LPR, "|lpr -Php212 >/dev/null 2>&1");
```

Procesos

- Combinación de procesos:

```
open(WHO, "who|");
open(LPR, "|lpr -Phpalumnos");
while(<WHO>){
    unless /cc31a/{ # evitar "cc31a"
        print LPR $_;
    }
}
close(WHO);
close(LPR);
```

Procesos

- Se pueden utilizar varios "|":

```
open(WHOP, "ls | tail -r |");
```

- En Perl también existe fork:

```
if (!defined($child_pid = fork())){
    die "No pude hacer fork: $!\n";
} elsif ($child_pid != 0){
    # padre
}else{
    # hijo
}
```

Formatos

- Dentro de las facilidades que otorga Perl para manipular texto, está el generar reportes
- Perl ofrece la noción de un reporte simple, mediante la escritura de un template llamado "format"
 - Un formato define una parte constante y una parte variable
- Para usar "format" son necesarias 3 cosas:
 - Definir el formato
 - Asignar los datos a la parte variable del formato
 - Invocar el formato

Formatos

- Definir el formato:
 - Se puede realizar en cualquier parte del código:

```
format FORMATNAME =
definicion_de_primera_linea
$valores, $primera, $linea
definicion_de_segunda_linea
$valores, $segunda, $linea
.
```

- La definición del formato termina con una línea que tiene solamente 1 punto
- Cada línea que define el formato es conocida como fieldline

Formatos

- El filehandle del archivo debe ser **idéntico** al nombre del formato
- Luego se abre un archivo con la información que se desea formatear (datos.txt), el contenido de este archivo puede ser:

```
Juan Urdemales:Blanco Encalada 2120:Santiago:RM:1234
Justo Estay:San Ignacio 460:Rancagua:VI:9876
```

- Al realizar “split(/:/)” se obtiene cada elemento
- En el ciclo while se lee cada línea del archivo de datos

Formatos

- Es importante que las variables que toman los valores del archivo de datos sean **idénticas** a las definidas en el formato
- Cuando se tienen todas las variables se llama a la función “write”, que invoca al formato
 - El parámetro que recibe write es el filehandle en donde se escribirá, por omisión utiliza el formato del mismo nombre
- Cada campo del formato (fieldholder) es reemplazado por el valor correspondiente de la variable

Formatos

- Después de procesar el contenido del archivo labels será:

```
=====
|Juan Urdemales      |
|Blanco Encalada 2120|
|Santiago           , RM 1234 |
|=====
|Justo Estay         |
|San Ignacio 460     |
|Rancagua           , VI 9876 |
|=====
```

Formatos

- En el ejemplo se utilizó el fieldholder @<<<<, el cual asignaba texto alineado a la izquierda
 - Similarmente @>>>> asigna texto alineado a la derecha
 - @||| texto centrado
- Para valores numéricos se utiliza @#####.##
 - 5 enteros y 2 decimales
- Con @* se permite fieldholder de varias líneas
- Se puede definir el formato para STDOUT, dando como nombre del formato STDOUT
- Mas información: perldoc perlform y perldoc -f format