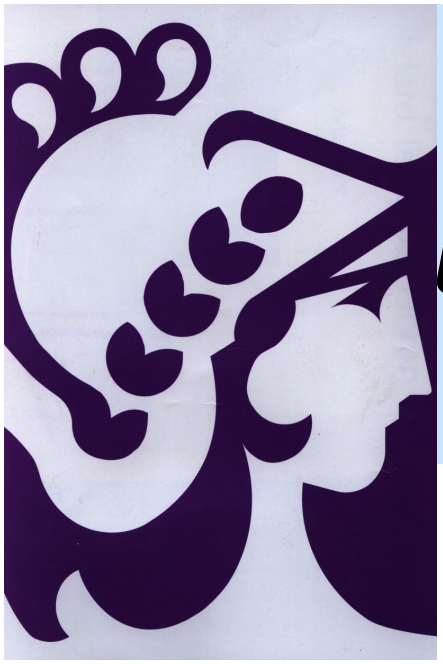

Facultad de Ingeniería
Universidad de Buenos Aires

75.08 Sistemas Operativos

Prof. Lic. Ing. Osvaldo Clúa

Jefe T.P. Lic. Adrián Muccio

Guía de Ejercicios de la Práctica

Autores

Lic. Sandra Abraham

Lic. Guido Fernandez

Ing. Tomás Rinke

Lic. Adrián Muccio

Versión 1.0

Índice

I. Introducción

La presente Guía contiene ejercicios relacionados a temas propios de la Práctica

El listado de ejercios no contempla el alcance completo de los temas, simplemente tiene el fin de servir como herramienta de apoyo a la cursada regular.

No es obligatoria la resolución de los ejercicios

Se incluye la resolución de algunos ejercicios a modo ejemplificativo sin perjucio de que existan otras posibles soluciones.

Los ejercicios se encuentran ordenados por tema y nivel de dificultad.

Los ejercicios tipo parcial no tienen orden definido.

II. Shell Scripting

1 Fecha Actual

Obtenga la siguiente salida (tal cual está, en dos líneas):

Son las hh:mm:ss del día: dd

del mes: mm del año: aa.

Solución:

date +"Son las %T del día %e %n del mes %m del año %y."

2 Directorio

Realizar un shell script que dado un directorio pasado por parámetro, cree un archivo tar comprimido con gzip y con nombre igual a la fecha en formato yyyymmdd seguido de guión y seguido del nombre del directorio terminado en .tar.gz.

Ejemplo: aplicado sobre home obtendríamos -? 2004-04-03-home.tar.gz.

3 Tabla de multiplicar

Realizar un shell script que dado un número 'n' pasado por parámetro muestre los diez primeros elementos de su tabla de multiplicar, mostrando el resultado en la forma: i x n = resultado.

4 Divisible

Realizar un shell script que, dado un número pasado por parámetro, indique si es o no divisible entre 101. Si no se proporciona un número debe mostrar como usar el programa.

5 Comprimir

Realizar un shell script que dado una lista de directorios, cree un archivo tar comprimido con gzip con nombre igual a la fecha en formato yyyy-mm-dd.tar.gz. Además se generará un archivo yyyy-mm-dd.lst con los nombres de los directorios contenidos en el archivo tar, UNO POR LINEA usando un bucle. Si el archivo lst existe, mostrar un error y terminar el programa. Si alguno de los elementos no es un directorio, mostrar un error y finalizar el programa.

6 PID

Realizar un shell script que permita adivinar al usuario cual es su PID. El script pide un número al usuario y cada vez que lo haga debe indicar al usuario si el PID es mayor o menor que el número introducido. Cuando se adivina el valor, se deben mostrar los intentos empleados.

7 Verificación

Realizar un shell script que verifique cada 30 segundos si existe en el directorio actual un archivo prueba.txt. Para probar este guión es necesario ejecutarlo en segundo plano.

8 Directorios

Crear un shell script que liste todos los directorios y subdirectorios recursivamente de uno dado. El directorio será introducido como argumento y el guión lo primero que hará será verificar si es precisamente un directorio.

9 Proceso

Hacer un shell-script llamado 'proceso.sh' para ser lanzado en background que como maximo permanecera vivo 30 segundos.

Podra ser lanzado varias veces en background y cada vez generará un shell-script distinto 'stop_proceso.sh.' que al ser ejecutado matara el proceso lo origino y despues se borrara a si mismo.

10 Pirámide

Realice un shell script 'piramide.sh' que reciba por entrada std (no por parámetro) un número 'N' y que optenga a la salida una serie de 'N' filas de forma triangular.

Para ./piramide.sh 12 la salida sería.

01

02 02

03 03 03

04 04 04 04

05 05 05 05 05

06 06 06 06 06 06

07 07 07 07 07 07 07

08 08 08 08 08 08 08 08

09 09 09 09 09 09 09 09 09

10 10 10 10 10 10 10 10 10 10

11 11 11 11 11 11 11 11 11 11 11

12 12 12 12 12 12 12 12 12 12 12 12

11 Igualdad

Realice un shell-script que admita tres palabras como argumentos y que muestre un mensaje informando de las relaciones de igualdad y desigualdad entre esas palabras.

"Las tres son iguales"

"Son iguales primera y segunda"

"Son iguales primera y tercera"

"Son iguales segunda y tercera"

"Son las tres distintas"

12 LOG

Asumiremos que tenemos en un directorio una serie de archivos de log que van creciendo de forma ilimitada con el uso regular de ciertos programas.

Realizar un shell script que actue sobre los archivos con nombre tipo '*.log' del directorio actual de forma tal, que si alguno de ellos supera en tamaño las 2000 lineas, dejará solo las últimas 1000 líneas del archivo y las restantes serán guardadas en un directorio old_rot en formato

comprimido.

En dicho directorio habrá que conservar en formato comprimido no solo la última porción eliminada del original, sino las cuatro últimas porciones eliminadas. Para ello será necesario ir rotando los nombres de estos archivos comprimidos incorporando en los mismos un digito de secuencia.

(parte eliminada) --> *.log.rot1.gz --> *.log.rot2.gz --> *.log.rot3.gz --> *.log.rot4.gz --> eliminado

El proceso durante su ejecución irá mostrando los archivos encontrados y señalará aquellos que por su tamaño sea necesario rotar.

13 Contenido

Escribir un shell-script denominado fileodir que compruebe que el parámetro que se le ha pasado es un archivo y en caso afirmativo muestre su contenido. Si el parámetro es un directorio deberá mostrar los archivos que contiene. También, debe aceptar más de un parámetro.

14 PATH

Hacer un shell-script que busque la presencia del comando pasado como argumento en alguno de los directorios referenciados en la variable $PATH, señalando su localización y una breve descripción del comando caso de existir su página man.

15 Rango

Realizar un shell-script que reciba como argumentos numeros comprendidos entre 1 y 75. Dará error en caso de que algun argumento no este dentro del rango y terminará sin hacer nada. En caso contrario generará una linea por cada argumento con tantos asteriscos como indique su argumento.

16 Archivos

Realizar un shell-script que acepte como argumentos nombres de

archivos y muestre el contenido de cada uno de ellos precedido de una cabecera con el nombre del archivo

17 Media Aritmética

Hacer un shell-script que calcule la media aritmética de todos los argumentos pasados como parámetros con una precisión de 40 digitos decimales despues de la coma.

III. Expresiones Regulares

Solo se pueden utilizar comandos sed, grep, wc, echo.

1 Capicúas

Se tiene un archivo con números enteros de 3 dígitos, se desea generar otro archivo con los capicúas de cada uno de los números

2 Reformatear Fecha

Se tiene un archivo de texto en el que aparecen fechas con el siguiente formato mm/dd/aaaa se desea cambiarle el formato a dd/mm/aaaa

Solución Ejemplo

```
                  1          2          3
            -----mes----  -----día-----  -----año----
sed 's-\([0-1] [0-9]\)/\([0-3][0-9]\)/\([0-9] \{4\}\)-\2/\1/\3-g'
```

3 IP

Realizar un shell script que reciba por parámetro una ip (EJ: 192.168.1.1) e indique si es válida o no.

4 FECHA

Realizar un shell script que tome por entrada std una fecha (formato YYYY-MM-DD) e indique si

es válida o no.

5 EMAIL

Realizar un shell script que reciba por parámetro un email e indique si es válido o no.

6 DEBITO

Realizar un shell script que busque dentro del archivo debito-automatico.txt que solo contiene números de tarjetas de crédito de Visa (empiezan con 4 y tienen 16 dígitos), Mastercard (empiezan del 51 al 55 inclusive y tienen 16 dígitos), American Express (empiezan del 34 al 37 y tienen 15 dígitos) y muestren por pantalla el número con su respectiva denominación.

Ejemplo:

4312-4311-4311-4123:VISA

5123-1231-1231-1231:MASTERCARD

7 DECIMALES

Realizar un shell script que busque dentro del archivo datos.txt los números decimales mayores a 7.534

ejemplo linea datos.txt

1:Juan Perez:9.321

2:Juan Paso:3.321

3:Matias Perez:7.534

4:Fernando Poso:7.999

8 /ETC/PASSWD

a. Mostrar los nombres de todos los usuarios de la máquina:

b. Mostrar los nombres de los usuarios, pero sólo los que usan bash (/bin/bash):

9 CANCIONES

Dado una duración de canción pasada por parámetro con el siguiente formato (hh:mm:ss) indicar si es mayor o menor al límite establecido:

LIMITE="00:03:12"

10 TEXTO

Dado el siguiente archivo de texto:

lipsum.txt

"Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Duis suscipit dictum urna et luctus. Ut non elementum urna, ac ultricies justo. Nulla a lacus rutrum, sollicitudin neque interdum, scelerisque mauris.

Mauris tempor rhoncus tincidunt. Mauris dignissim venenatis risus, sit amet tincidunt risus laoreet vehicula.

Etiam volutpat libero ac ipsum vestibulum elementum. Morbi eget diam non tellus mattis malesuada.

Sed fermentum felis tempus nisi venenatis dictum.

Nam suscipit lacinia nisl ut pulvinar. Proin id enim condimentum, ultrices leo quis, auctor tellus.

Aliquam porttitor nibh felis. Integer pharetra elementum libero rhoncus egestas.

Etiam at aliquet elit, ac feugiat nunc. Proin id lorem viverra mi consectetur tempus ut sed neque.

Nullam scelerisque congue accumsan. Nunc tincidunt tellus odio, eu fermentum mi tempor at.

Etiam et ullamcorper elit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos."

a. Realizar un shell script que busque dentro del archivo de texto palabras en latín pasadas por parámetro.

b. Reemplace en las lineas que empiezan con E, la quinta palabra por el texto "ELIMINADO"

c. Elimine las líneas que terminan tienen la palabra "sed"

d Inserte comentarios con la siguiente notación: "#COMENTARIO: " al final de cada linea

IV. PERL

1. Escriba un programa en Perl que lea dos números de entrada standard, los multiplique e imprima el resultado.
2. Escriba un programa en Perl que utiliza la sentencia while para imprimir los primeros 10 números (1-10) en orden ascendente.
3. Mostrar por salida standard los elementos de un array (usar sentencia for).
4. Llenar dos vectores A y B de 45 elementos cada uno, sumar el elemento uno del vector A con el elemento uno del vector B y así sucesivamente hasta 45; almacenar el resultado en un vector C, e imprimir el vector resultante.
5. Cargar un vector con 10 elementos y calcular el promedio de los valores almacenados. Determinar además cuántos son mayores del promedio, imprimir el promedio, el número de datos mayores que el promedio.
6. Mostrar por salida standard los elementos de un hash (usar sentencia foreach).
7. Leer del achrivo facturas.txt, cuyo formato es :nro de factura, fecha,sucursal, importe. Mostrar por salida stantard el total de facturas por sucursal (usar hash)
8. A traves de los siguientes ejercicios se mostraran
   - diferentes maneras de cargar/usar hashes simples
   - diferentes maneras de leer/recorrer un archivo
   - diferentes maneras de leer/recorrer un directorio
   - diferentes file test (-e, -r, -s)
   - diferentes funciones de manipulacion de registros/campos
     - Split
     - Substr
     - Length
     - Join
     - index
   - otras funciones complementarias utiles en el procesamiento de archivos
     - localtime
     - getlogin
     - chomp

Ejercicio a)

```
#!/usr/local/bin/perl
# PARTE A
# Armar un arreglo que traduzca el  numero del mes a su correspondiente
```

```
nombre y mostrar el resultado

%month = ('01', 'enero',  '02', 'febrero',  '03', 'marzo',
     '04', 'abril',  '05', 'mayo',     '06', 'junio',
     '07', 'julio',  '08', 'agosto',   '09', 'septiembre',
     '10', 'octubre','11', 'noviembre','12', 'diciembre');


foreach $i (keys %month) {
   printf "\n $i es $month{$i} ";
   }

# PARTE B - acumular las ventas por mes (todas las ventas informadas
corresponden al mismo año)
# imprimir un listado ordenado por mes con: "Las ventas de <nombre del mes>
fueron de: <valor> pesos"
# Archivo de input en el directorio corriente con nombre: ventas2.csv y
formato: fecha(aaaammdd);monto
$archivo = "ventas2.csv";
open (ARCHIN, "<$archivo");
@input = <ARCHIN>;
foreach $unregistro (@input) {
   chomp ($unregistro);
   @reg = split (";","$unregistro");
   $mes = substr($reg[0],4,2);
   $monto = $reg[1];
   $ventas{$mes} = $ventas{$mes} + $monto;
}
close(ARCHIN);
foreach $i ( sort keys %ventas) {
   printf "Las ventas de $month{$i} fueron de $ventas{$i} pesos\n";
   }
```

Ejercicio b)

```
#!c:\Users\Sandra\Perl\bin\perl
#---- PARTE A: leer primer parametro y validar que exista el directorio
$directorio = $ARGV[0];
print "el nombre del directorio donde se encuentran los archivos es:
$directorio \n";
if (!-d $directorio) {print "directorio NO encontrado"; exit; }
#---- PARET B: leer el directorio y listar SOLO los nombres de los archivos
que empiecen con "venta"
# USO opendir, readdir, closedir
opendir (DIR,$directorio);
@archivos = readdir (DIR);
closedir (DIR);
foreach $f (@archivos) {
   if ($f =~ "venta*"){
       print " nombre del archivo de venta: $f \n";
       }
}
#---- PARTE C: leer el directorio y listar path completo y nombre de los
archivos que empiecen con "venta"
# USO del operador diamante: <>
@archivos_de_ventas = <$directorio/venta*>;
foreach $f (@archivos_de_ventas) {
       print " path completo y nombre del archivo de venta: $f \n";
}
```

Ejercicio c)

```
#!/usr/local/bin/perl
#Ej. 1.   Generar un archivo (segundo parametro) a partir de otro
(primer parametro) que cumpla las siguientes condiciones:
#  campo 1: registro de input HASTA 80 caracteres: si el registro de
input tiene mas, truncar en 80 caracteres
#  campo 2: usuario, sin formato ni longitud especificado
#  campo 3: fecha, sin formato ni longitud especificado
#  separador de campos: ; (carácter punto y coma)
# Validar que el archivo de input exista
@ARGV == 2 || die "Cantidad de argumentos incorrecta\n";
($archin, $arch1) = @ARGV;
-e $archin || die "El Archivo de input no existe\n";
open (ARCHIN, "<$archin");
open (ARCHOUT1, ">$arch1");
@input = <ARCHIN>;
foreach $unregistro (@input) {
   chomp ($unregistro);
   $campo80 = substr($unregistro,0,80);
   $campo2 = getlogin;
   $campo3 = localtime;
   $salida1 = join(";",($campo80,$campo2,$campo3));
   print ARCHOUT1 "$salida1\n";
}
close(ARCHIN);
close(ARCHOUT1);
```

Ejercicio d)

```
#!/usr/local/bin/perl
#Ej. 2.   Idem ejercicio 1 pero agregando las siguientes condiciones:
#• Si la longitud del registro es mayor a 80, grabar solo el resto del
registro en otro archivo (pasado como parametro 3)
#• Si la longitud del registro en cero, no grabar ninguna salida
#• El formato de la fecha debe ser: d/m/aaaa
# Validar que el archivo de input sea de lectura

$cp = $#ARGV + 1;
$cp == 3 || die "Cantidad de parametros incorrecta\n";
$archi = $ARGV[0];
$arch1 = $ARGV[1];
$arch2 = $ARGV[2];
if (!-r $archi) {
   print STDERR ("Archivo de input no es de lectura\n");
   exit;
}
open (ARCHIN, "<$archi");
open (AROUT1, ">$arch1");
open (AROUT2, ">$arch2");
foreach $unregistro (<ARCHIN>) {
   chomp ($unregistro);
   $campo80 = substr($unregistro,0,80);
   $campo_resto = substr($unregistro,81);
   $l= length($campo80);
   if ($l == 0) {
      next;
   }
   $campo2 = getlogin;
```

```
    ($s,$m,$h,$dia,$mes,$anio) = localtime;
    $anio += 1900;
    $mes++;
    $campo3 = "$dia/$mes/$anio";

    print AROUT1 "$campo80;$campo2;$campo3\n";

    if (length($campo_resto)) {
        print AROUT2 "$campo_resto\n";
    }
}
close(AROUT1,AROUT2);
```

Ejercicio e)

```
#!/usr/local/bin/perl
#Ej. 3.   Leer el archivo "C:/Users/Sandra/Perl/Ejemplos/catalogo.txt"
#       Grabar un archivo que cumpla las siguientes condiciones:
#         campo 1= tercer campo del archivo de input cuando contiene en
alguna parte el string 'azul'. Si no contiene la palabra 'azul', grabar la
leyenda: 'Color Pendiente'
#         campo 2= (segundo campo del archivo de input x 60) + (primer
campo del archivo de input)
#         campo 3= fecha de grabacion con formato dd-mm-aaaa
#  El separador de campos (del archivo de input y del de output) es punto y
coma
#       Validar que el archivo de input no este vacio
#       Solicitar al usuario el nombre del archivo de salida

-e "C:/Users/Sandra/Perl/Ejemplos/catalogo.txt" || die "Archivo de input no
encontrado\n";


if (!-s "C:/Users/Sandra/Perl/Ejemplos/catalogo.txt") {
   print "El archivo de input esta vacio\n";
   exit;
}
open (FDIN, "C:/Users/Sandra/Perl/Ejemplos/catalogo.txt") || die print
"ERROR en el open-input del archivo de entrada ";

print "por favor ingrese el nombre del archivo de salida:";
$salida = <STDIN>;
chomp($salida);

open (FDOUT, ">$salida") || die print "ERROR en el open-output del archivo
$salida ";

while (<FDIN>){
   chomp;
   @reg = split(";", $_); print "Archivo de entrada $reg[0] $reg[1]
$reg[2]\n";
   $hayazul = index($reg[2],'azul');
   if ($hayazul >= 0) {
      $campo1=$reg[2];
   } else {
      $campo1='Color Pendiente';
   }
   $campo2=$reg[0]+$reg[1]*60;
   my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime;
   $year += 1900;
```

```
    $mon++;
    $mon = "0$mon" if $mon < 10;
    $mday = "0$mday" if $mday < 10;
    $campo3 = "$mday-$mon-$year";
    print FDOUT ($campo1.";".$campo2.";".$campo3."\n");
}
close FDOUT;
```

V. Ejercicios Tipo Parcial

ER-Pay Per View

Una empresa operadora de televisión para su servicio de canales con cargo adicional, conocido como “Pay Per View”, desea que sus clientes puedan adquirir este servicio por medio de un mensaje SMS.

Nos pide desarrollar un script que sea invocado por una operadora telefónica y nuestro script registre la venta invocando a su vez a un API de su sistema CRM.

El script debe recibir como parámetros de entrada:

- Número de teléfono origen
- Código del cliente
- Canal

El Número de teléfono origen se envía con el siguiente formato fijo *(nn)(nnnnnn)nnnn*.

Donde ‘*n’:* significa dígito

Los caracteres 6 a 11 contienen el código de área

Los caracteres 13 a 16 contienen el número

El API a invocar es un programa que se llama RegistrarVentaPPV y recibe como parámetros el código de cliente y la señal PPV.

Por restricciones técnicas, en un mismo canal, no todos los clientes ven las mismas señales.

La señal se determina en base al código de área del teléfono y el canal recibidos.

Las relaciones se encuentran en el archivo Signals&Chanels.dat, que posee el siguiente formato:

Nombre de Campo	Descripción	

ID_RELACION	Identificación de Relación (*)	Todos los campos se encuentran separados por el caracter ;
SEÑAL	Señal (*)	El (*) indica que se desconocen tipo y formato del campo. Existe a lo sumo un canal por cada dupla de SEÑAL y CODIGO_AREA
NOMBRE_LARGO	Nombre largo de la señal (*)	
CODIGO_AREA	Código de área. Tipo: numérico de 6 posiciones Formato: de ser necesario se rellena con 0s a la izquierda.	
CANAL	Código del canal (*)	
COMENTARIOS	Comentarios varios (*)	

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, bc, let y echo

ER-Centro de Despacho

En una empresa de servicios, el Centro de Despacho asigna las órdenes de trabajo a los técnicos y controla su ejecución. Este sector requiere un script que reciba como parámetro un identificador de técnico y devuelva por salida std el código de la orden de trabajo que tiene asignada y la hora con minutos y segundos en que fue planificada. A lo sumo existe una orden asignada para un técnico en un día.

Para esto se cuenta con el archivo de asignaciones del día, que posee el siguiente formato:

ID_ASIGNACION	Identificador de Asignación. Tipo Alfanumerico. Se desconoce formato y longitud	Todos los campos se encuentran separados por el caracter ;
COD_OT	Código de Orden de Trabajo (*)	El (*) indica que se desconocen tipo y formato del campo
TIPO_OT	Tipo de Orden de Trabajo (*)	
ID_CLIENTE	Identificador de Cliente (*)	
DIRECCION	Dirección (*)	
ID_TECNICO	Identificador del Técnico (*)	
FECHA_PLAN	Fecha Planificada. Tipo Fecha. Formato: dd/mm/aaaa hh:mi:ss	
ESTADO	Estado de la Orden (*)	
FECHA_REAL	Fecha Real. Tipo Fecha Formato: dd/mm/aaaa hh:mi:ss	
COMENTARIO	Comentarios varios (*)	

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, sort y echo

ER-Juegos Panamericanos

Para evaluar los resultados de cada uno de los países participantes de los Juegos Panamericanos se nos pide desarollar un script que reciba como parámetros un nombre de país, un nombre de disciplina y muestre por salida std el mensaje:

"<Nombre_de_país> obtuvo una medalla <metal_de_medalla> en <nombre_de_disciplina>" si ganó una medalla.

O el mensaje "<Nombre_de_país> no obtuvo ninguna medalla en <nombre_de_disciplina>"

Para esto se cuenta con el archivo Resultados.dat, que posee un registro por cada combinación de país-disciplina y cuenta con el siguiente formato:

Nombre de Campo	Descripción	
CODIGO_PAIS	Código de país (*)	Todos los campos se encuentran separados por el caracter ;
NOMBRE_PAIS	Nombre de país (*)	El (*) indica que se desconocen tipo y formato del campo
CODIGO_DISCIPLINA	Código de disciplina (*)	
NOMBRE_DISCIPLINA	Nombre de disciplina(*)	
FECHA_FINAL	Fecha de última prueba(*)	
ORO	Caracter S o N	
PLATA	Caracter S o N	
BRONCE	Caracter S o N	
BATIO_RECORD	Caracter S o N	
COMENTARIOS	Comentarios varios (*)	

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, bc, let y echo

ER-Clasificador de Errores

Se nos pide realizar un script para clasificar errores llamado ClasificaErrores.sh que recibe como parámetro un código de error y devuelve por salida std la clasificación que le corresponda.

Para clasificar los códigos se debe aplicar la siguiente lógica según el código enviado.

Buscar en archivo CLASE_DE_ERRORES.dat, ingresando por el campo ERROR_CODE y obteniendo el campo "ERROR_CLASS"

En caso de no encontrar clasificación para el código, debe devolver el código encerrado entre "**<>**" y seguido del mensaje " ***- No Clasificado***"

El archivo CLASE_DE_ERRORES.dat posee el siguiente formato:

ID_CLASS	(*)	Todos los campos se encuentran separados por el caracter :
ERROR_CODE	(*) (**)	(*) indica que se desconocen tipo y formato del campo
ERROR_DESCRIPCION	(*)	(**) indica que no hay más de una línea con el mismo valor en ese campo
ERROR_CLASS	(*)	
LAST_UPDATED_BY	(*)	
LAST_UPDATE_DATE	(*)	

Ejemplo de invocación del script:

> ClasificaErrores.sh "*código de error*"

<*código de error*> - No Clasificado

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, sort y echo

ER-Actualizador de Precios

Un supermercado nos pide realizar la integración para actualizar los precios entre los sistemas ERP y Flejes (que muestra los precios de los productos en góndola).

Para esto desarrollaremos el script ActualizarPrecioERP.sh que recibirá por parámetro el identificador de artículo del ERP, el precio y el id de sucursal destino.

Nuestro script traducirá el identificador de artículo al código standard internacional EAN 13, enviará el id de sucursal destino y cambiará el formato del precio para finalmente invocar al API SetPriceFlejes.

El código de retorno del script debe ser el mismo código que devuelve el API.

El formato de precio de que envía el ERP no tiene separador de decimales, interpreta los últimos dos caracteres como los centavos, pero el formato esperado por Flejes tiene el carácter '.' como separador. Ejemplo '1000' à '10.00'

Para obtener el código EAN13 se cuenta con el archivo ARTICULOS.dat, que posee el siguiente formato:

ID_ARTICULO	Identificador de artículo (*)	Todos los campos se encuentran separados por el caracter ;
NOMBRE	Nombre del artículo (*)	El (*) indica que se desconocen tipo y formato del campo
DESCRIPCION	Descripción del artículo (*)	
COD_EAN13	Código EAN 13 (*)	
COD_EAN7	Código EAN 7 (*)	
OTROS	(*)	

Ejemplo de invocación del script:

ActualizarPrecioERP.sh <id_artículo> <precioERP> <id_sucursal>

Ejemplo de invocación del API:

SetPriceFlejes <*EAN_13*> < *id_sucursal*> <*precioFLEJES*>

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, sort y echo

ER-Medidor de Distancia

La Agencia Nacional de Vialidad nos pide desarrollar un script llamado ***distancia.sh*** que muestre por salida standard la cantidad de kilómetros de distancia entre dos localidades unidas por una ruta.

El script recibe por entrada standard una línea que contiene el nombre de la localidad origen, el nombre de la localidad destino y como tercer valor, el número de ruta.

Los valores se encuentran separados por el caracter “%”

Contamos con el archivo Info_rutas.dat que contiene en que kilómetro se encuentra cada una de todas las localidades según las rutas de las que formen parte.

El archivo Info_rutas.dat posee el siguiente formato de campos separados por el caracter **'-'**’

- ID
- Ruta
- Localidad
- Kilómetro relativo al inicio de la ruta

El script ***distancia.sh*** va a ser invocado en forma automática. Es por eso que todos los valores de la entrada se consideran válidos (cantidad, formato y valores existentes en archivo Info_rutas.dat). Se utiliza el mecanismo de pipe “|” de manera de optimizar los tiempos de procesamiento.

Ejemplo de invocación:

$ Cmd1.exe | ***distancia.sh***

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, bc y echo

ER-Disponibilidad Hotelera

Una agencia de turismo nos pide desarrollar un script que muestre por salida std **SOLO** la cantidad de habitaciones disponibles en los hoteles a la fecha de inicio del próximo carnaval, esto es el 17/02/2013

El script recibe por entrada std la cantidad de huéspedes y el número de estrellas ambos campos separados por el carácter **','**

Para obtener la cantidad de habitaciones disponibles se cuenta con el archivo disponibilidad.dat que posee el siguiente formato de campos separados por el caracter **'-'**'

- ID
- Hotel
- Estrellas
- Habitación
- Capacidad de huéspedes de la habitación
- Fecha #formato mm/aa/aaaa
- Estado # el estado disponible tiene como valor **'DISP'**

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, bc y echo

ER-Filtro Musical

Se desea realizar un script llamado FiltroMusical que reciba de la entrada std un género musical y muestre por salida std todos los autores y los nombres de los temas cuya duración se encuentre dentro del rango entre 30 minutos y una hora con 20 minutos inclusive.

La información de los temas musicales se encuentra en un archivo llamado Lista.Musical con el siguiente formato:

FECHA DE CREACION	
ALBUM	
NOMBRE DEL TEMA	
DURACION	Formato: HH:MM:SS
INTERPRETE	
AUTOR	
GENERO	

Todos los campos se encuentran separados por el caracter ;

Ejemplo de invocación del script:

> *> echo "Opera" | Filtro.Musical*

Ejemplo de salida:

Mozart-Las bodas de Figaro

Verdi-La traviata

IMPORTANTE: Solo se permite el uso de los comandos grep, sed, wc, y echo

Shell- Facturas

Se reciben en un directorio `/input` archivos de diversos proveedores con la información de las facturas por estos emitidas, el nombre de los archivos `<cod.prov>.<aaaammdd>.dat` (ej. `K234.20100910.dat`), con la siguiente información:
`nro factura,cod.producto,cantidad,importe`
Además se cuenta con el archivo de proveedores (`/tablas/provema.txt`)
`cod. proveedor, tipo producto, cuit, razón social`
También tenemos el archivo de productos (`/tablas/productos.txt`)
`cod. producto, tipo producto, descripción`
Se pide realizar un script que valide estos archivos, teniendo las siguientes consideraciones:

- El código de proveedor debe ser valido (debe existir en prevema.txt).
- El producto debe pertenecer al tipo de productos que trabaje ese proveedor, para eso utilizar los archivos de proveedores y productos.

Las facturas validadas deben guardase en el directorio `/ok` con el nombre de archivo `FACTURAS.aaaamm.txt` con los siguientes campos:
`Cod. Proveedor,nro factura,cod. producto,cantidad`
Las facturas erróneas deben guardarse con el mismo formato en `/error` con el nombre `facturas.error.aaaamm.err`

Shell- Validar archivos.

Se tienen n archivos en el directorio seteado en la variable de ambiente DIR_PROC, se pide validar que los archivos tengan todos sus registros con el formato correcto, para esto se deberá validar la cantidad de campos de cada registro, la misma se obtendrá el archivo REGISTROS.DAT que tiene el siguiente formato:
Extensión|separador| cantidad de campos (ej. txt|;|5)

Tomando esta información validar los archivos, solo se debe validar los archivos cuyas extensiones existen en el archivo REGISTROS.DAT.
Por cada archivo mostrar por salida standard nombre del archivo cantidad de registros totales, erróneos y buenos.

Se deben copiar en el directorio de la variable DIR_PROC_OK los registros que cumplen la cantidad de campos, los registros los erróneos copiarlos en DIR_PROC_ERR.

Shell- Pedidos.

Se reciben archivos de Pedidos de diferentes agencias (agencia.sec.dat ej.SOLDATI.003.dat), con el siguiente formato:
Nro pedido, fecha pedido, producto, cantidad, fecha_entrega
Ej. 12154,20111030,A45,152,20111105

Se pide realizar un script que realice las siguientes validaciones:
- La secuencia del archivo debe ser válida (tiene que ser mayor que la secuencia que le corresponde a la agencia Agencia.dat). El formato de AGENCIA.DAT es:

Agencia, descripción, dirección, última secuencia
Ej. SOLDATI,Agencia Soldati, Moreno 456,2

- La fecha pedido debe ser menor que la fecha de entrega
- El producto tiene que tener stock (verificar en Productos.DAT cantidad debe ser mayor a cantidad producto) . El formato de PRODUCTOS.DAT es:

Producto, descripción ,cantidad producto

Ej.A45, Producto A,300

Generar para los registros ok un archivo agregando la extensión ok y para los errores la extensión err (ej. SOLDATI.003.dat.ok y/o SOLDATI.003.dat.err)

PERL- Parcial 1

Se pide desarrollar un comando perl denominado “Acumulador” que tenga la capacidad de sumar las cantidades vendidas por artículo. Para ello debe procesar un archivo de entrada cuyo nombre es pasado como parámetro y dejar el resultado en otro archivo cuyo nombre también es pasado como parámetro.

Además se pasan como parámetro la ubicación del campo artículo y la ubicación del campo cantidad dentro del archivo de entrada. El separador de campos del archivo de entrada es el punto y coma.

Es obligatorio el uso de una estructura de hash para efectuar la acumulación de cantidades por artículo. El campo artículo es el que se emplea como clave y el campo cantidad es el que se debe acumular.

Salida: se debe grabar en un archivo de salida todos los artículos con su cantidad acumulada. El separador de campos del archivo de salida debe ser la coma y su formato: artículo, cantidad acumulada

Parámetro 1: ubicación del campo artículo

Parámetro 2: ubicación del campo cantidad

Parámetro 3: Nombre del archivo de Input

Parámetro 4: Nombre del archivo de Output

Validaciones a efectuar:

1. Que sean 4 parámetros, si se pasan mas ignorar los que sobran, si se pasan menos mostrar por pantalla un mensaje descriptivo del error.

2. Que el archivo de entrada exista. Si no existe mostrar por pantalla un mensaje descriptivo del error.

3. Que "ubicación del campo artículo" sea un numero >= 1 y <= cantidad total de campos del registro de entrada pero distinto a "ubicación del campo cantidad"

4. Que "ubicación del campo cantidad" sea un numero >= 1 y <= cantidad total de campos del registro de entrada

PERL- Parcial 2

*Desarrollar un comando perl denominado "**parcial.pl**" cuyo propósito es procesar archivos con datos del ultimo censo económico. El nombre de estos archivos es: **<cod.provincia>-<cód.localidad>.dat***

El separador de registros es el new line, el separador de campos es el **; (punto y coma)**

La cantidad de campos dentro de un registro es desconocida pero los primeros 5 campos siempre son:

Fecha; población urbana; PBI urbano; población rural; PBI rural

El comando puede ser invocado con la **opción –s o con la opción –f.** Sin parámetros o cualquier otra opción debe considerarse error, en cuyo caso se debe mostrar un mensaje explicativo y terminar el programa.

Opción -s: Si el comando fue invocado con la opción –s, entonces parcial.pl es un proceso que nunca termina. Se ejecuta, duerme 30 minutos, se ejecuta, duerme 30 minutos nuevamente, y así sucesivamente. En Perl la función sleep tiene la siguiente sintaxis: sleep <cantidad de segundos>.

1. Por cada Ciclo debe leer todos los archivos *.dat que haya en el directorio corriente.
   1.1. Por cada archivo debe validar que el cód de la provincia y el cód de la localidad existan en la tabla Localidades.tab cuyo formato es el siguiente: cód Provincia; cód Localidad
      1.1.1. Si existe, el archivo es procesable, entonces:
         1.1.1.1. Acumula en una estructura Hash la **población total de la provincia (urbana + rural)**
      1.1.2. Si no existe, el archivo no es procesable, entonces:
         1.1.2.1. muestra el mensaje: el archivo <nombre del archivo> no será procesado
   1.2. borra el archivo para evitar leerlo nuevamente
2. Si pudo procesar algo en ese ciclo, entonces debe grabar en **salida.txt**: Fecha de Proceso;

Provincia; población total. Si el archivo **salida.txt** ya existe, no se deben perder los registros existentes, agregarle los nuevos.

3. Duerme 30 minutos y vuelve a 1.

Opción -f: si el comando fue invocado con la opción –f se debe realizar el mismo trabajo de la opción –s pero una sola vez (no duerme y termina)

Condiciones de Resolución:

Abrir y cerrar adecuadamente todos los archivos. En caso de error en alguna de estas operaciones, mostrar un mensaje explicativo y terminar el programa.

No usar archivos auxiliares bajo ninguna hipótesis.

Bajo ningún concepto se podrán utilizar comandos Unix dentro del script de Perl.

Es obligatorio: El uso de Hash (para acumular) y de Split (para procesar los registros)

PERL- Parcial 3

1. Se tienen n archivos de beneficiarios (“benef<numero cualquiera>” ejemplo: benef1, benef22, benef111, etc) con el siguiente formato: oficina,zona,plan,persona,mes,monto pagado.

2. Se pide acumular en un hash los montos pagados por oficina (el campo oficina es el que se emplea como clave) solo para la zona y el plan pasados como parámetro, es decir, que se debe invocar al script con los siguientes parámetros: zona y plan (el plan se valida con el archivo plan.tab, la zona no se valida)

3. Cuando se terminan de procesar todos los registros de los n archivos de beneficiarios, solicitar al usuario que ingrese por estándar input el nombre de un archivo de salida

4. Grabar en este archivo los montos acumulados por oficina (un registro por oficina) Formato de salida: Oficina, monto

5. Mientras se graba, si se detecta que el monto acumulado de alguna oficina es menor o igual a los $1.000 se debe mostrar un mensaje por estándar output: Para el plan <descripción del plan> en la zona xxxx, la oficina yyyy no supero los $1000

6. Todos los archivos mencionados se encuentran en el directorio corriente

7. Abrir y cerrar adecuadamente todos los archivos. Siempre el separador de campos es la coma y el de registros es el new line.

8. Verificar que se pasen 2 parámetros de input, si se pasan mas, ignorarlos, pero si se pasan menos, cancelar el programa

9. Verificar que el Plan ingresado como parámetro exista en plan.tab (formato de registro: plan,descripción)

10. Si el archivo de salida existe, solicitar al usuario confirmación para sobreescribirlo. Si el usuario no quiere sobreescribirlo, terminar el programa sin grabar.

11. No usar archivos auxiliares, comandos Unix (bajo ningún concepto) ni expresiones

regulares (excepto las expresiones simples de las funciones predefinidas de perl)