Actividad 4

Luis Aarón Cerón Ramírez

March 4, 2018

1 Introducción

El objetivo de esta actividad fue el de utilizar los difernetes comandos, aplicados en la manipulacion de datos, los cuales son de gran importancia pues facilitan el trabajo a realizar. entre los comandos utilizados se encuentran el cat, less, grep, de los cuales se hara una explicacion. La actividad consistio en utilizar los diferentes comandos a archivos obtenidos de la base de datos de la Universidad de Wyoming.

2 Lista de comandos

2.1 cat

Concatanea ficheros en otro dado. Si la salida de la concatenacion se omite, cat la muesta por pantalla. Si la entrada se omite, el fichero consiste en los caracteres que se introuducen por la terminal.
Este comando sirve por tanto, para ver ficheros y crear ficheros.

2.2 chmod

El comando chmod cambia los permisos de un archivo. Para que este comando funcione, necesitas permisos de busqueda en el directorio que contenga al archivo.

2.3 echo

Este comando escribe a la salida estándar la cadena de texto que se le pasa como parámetro. Generalmente se utiliza sin opciones, es por eso que no se nombrarán en este texto.

2.4 grep

Este comando busca en uno o más archivos las líneas que contenga un objetivo, e imprime todas las líneas que encuentra. Por ejemplo, la siguiente orden imprime todas las líneas del archivo que contiene la palabra lugar. Se utiliza con frecuencia para buscar información en archivos estructurados o bases de datos simples.

2.5 ls

Si se ejecuta ls sin argumentos, dará como resultado un listado de todos los archivos (incluyendo directorios) del directorio donde el usuario está posicionado.

2.6 wc

El nombre del comando wc proviene de word count, y como es de suponer, sirve para contar palabras.

3 Sintesis de shell script

Las dos primeras secciones de este texto da una introducción y habla acerca de la filosofia de este lenguaje, por lo que lo omitire y hablare directamente de las demas subsecciones

3.1 A first script

En esta primer seccion nos ensenan a hacer nuestro primer script, muy basico por cierto

```
#!/bin/sh
# This is a comment!
echo Hello World        # This is a comment, too!
```

La primera línea le dice a Unix que el archivo debe ser ejecutado por / bin / sh.
La segunda línea comienza con un símbolo especial: #. Esto marca la línea como un comentario, y shell lo ignora por completo.
La tercera línea ejecuta un comando: echo, con dos parámetros o argumentos: el primero es "Hello"; el segundo es "Mundo". Tenga en cuenta que echo colocará automáticamente un espacio único entre sus parámetros. El símbolo # aún marca un comentario; el # y todo lo que sigue es ignorado por el shell.
Despues se ejecuta el comando chmod 755 para hacerlos ejecutable

```
$ chmod 755 first.sh
$ ./first.sh
Hello World
$
```

La sección termina haciendole algunos cambios a la primer script.

3.2 Variable

En esta sección nos enseña como usar variables con el siguiente script

```
#!/bin/sh
MY_MESSAGE="Hello World"
echo $MY_MESSAGE
```

Esto asigna la cadena "Hello World" a la variable MY_MESSAGE y luego repite el valor de la variable. Al shell no le importan los tipos de variables; pueden almacenar cadenas, enteros, números reales, cualquier cosa.

```
$ x="hello"
$ expr $x + 1
expr: non-numeric argument
$
```

Esto se debe a que el programa externo expr solo espera números, sin embargo, tenga en cuenta que los caracteres especiales deben escaparse correctamente para evitar la interpretación por parte del intérprete de comandos.

3.3 Escape characters

ciertos caracteres son importantes para shell, es por eso que en esta seccion nos enseñan como se debe utilizar por ejemplo que el uso de caracteres de comillas dobles (") afecta la forma en que se tratan los espacios y los caracteres TAB. La mayoría de los caracteres (*, ', etc.) no se interpretan (es decir, se toman literalmente) colocándolos entre comillas dobles (""). Se toman como están y se pasan al

comando que los llama. Sin embargo, ; $, ỳ backlash aún son interpretados por el shell, incluso cuando están entre comillas dobles. El carácter de barra invertida (backlash) se utiliza para marcar estos caracteres especiales para que el intérprete no los interprete, sino que los pase al comando que se está ejecutando. Hemos visto por qué el "es especial para preservar el espaciado. El dólar ($) es especial porque marca una variable, por lo que $ X es reemplazado por el caparazón con el contenido de la variable X.

3.4 Loops

La mayoría de los lenguajes tienen el concepto de bucles: si queremos repetir una tarea veinte veces, no queremos tener que escribir el código veinte veces, con un ligero cambio cada vez. Como resultado, tenemos ciclos while y while en el shell Bourne. En esta seccion se aprende a utilizar estos importantes comandos, asi como su sintaxis.
For loops:
Este iterar a través de un conjunto de valores hasta que se agote la lista

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

While loops:
Lo que ocurre aquí es que las instrucciones de echo y lectura se ejecutarán indefinidamente.

```
#!/bin/sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
  echo "Please type something in (bye to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

3.5 Test

El test es utilizada por prácticamente todos los guiones de shell escritos. Puede que no parezca así, porque el test a menudo no se llama directamente. El test se llama con más frecuencia como [. [es un enlace simbólico para probar, solo para hacer que los programas de shell sean más legibles.

```
$ type [
[ is a shell builtin
$ which [
/usr/bin/[
$ ls -l /usr/bin/[
lrwxrwxrwx 1 root root 4 Mar 27 2000 /usr/bin/[ -> test
$ ls -l /usr/bin/test
-rwxr-xr-x 1 root root 35368 Mar 27  2000 /usr/bin/test
```

Esto significa que '[' es en realidad un programa, al igual que ls y otros programas, por lo que debe estar rodeado de espacios

3.6 Case

La declaración case guarda pasar por un conjunto completo de sentencias if .. then .. else. Su sintaxis es realmente bastante simple:

```
$ ./talk.sh
Please talk to me ...
hello
Hello yourself!
What do you think of politics?
Sorry, I don't understand
bye
See you again!

That's all folks!

echo "That's all folks!"
```

La línea case en sí siempre tiene el mismo formato, y significa que estamos probando el valor de la variable INPUT_STRING.

3.7 External programs

Los programas externos a menudo se usan en scripts de shell; hay algunos comandos integrados (echo, que, y prueba comúnmente están integrados), pero muchos comandos útiles son en realidad utilidades de Unix, como tr, grep, expr y cut.
El backtick (`) también se asocia a menudo con comandos externos.
Eejmplo:

```
#!/bin/sh
HTML_FILES=‘find / -name "*.html" -print‘
echo "$HTML_FILES" | grep "/index.html$"
echo "$HTML_FILES" | grep "/contents.html$"
```

3.8 Variable 2

Existen otro tipo de variable ya establecidas las cuales no pueden tener un valor asignado, estas pueden contener informacion util para el script.
El primer conjunto de variables que veremos son $ 0 .. $ 9 y $. La variable $ 0 es el nombre base del programa como se lo llamó. $ 1 .. $ 9 son los primeros 9 parámetros adicionales con los que se invocó el script. La variable $ @ es todos los parámetros $ 1 ... lo que sea. La variable $ *, es similar, pero no conserva ningún espacio en blanco, y las comillas, por lo que "Archivo con espacios" se convierte en "Archivo" "con" "espacios".

3.9 Function

Una característica que a menudo se pasa por alto de la programación de guiones de shell de Bourne es que puede escribir fácilmente funciones para usar en su secuencia de comandos.Esto generalmente se hace de una de dos maneras; con un script simple, la función simplemente se declara en el mismo archivo como se llama. Sin embargo, al escribir un conjunto de secuencias de comandos, a menudo es más fácil escribir una "biblioteca" de funciones útiles, y el origen de ese archivo al inicio de los otros scripts que utilizan las funciones.

```
. ./library.sh
```

Para empezar el script Una función puede devolver un valor en una de cuatro formas diferentes:
Cambiar el estado de una variable o variables.
Use el comando exit para finalizar el script de shell.
Utilice el comando de retorno para finalizar la función y devuelva el valor proporcionado a la sección de llamada del script de shell.
echo output to stdout, que será capturado por la persona que llama al igual que c = ‘expr $ a + $ b‘ está atrapado.
Esto es más bien como C, en esa salida se detiene el programa y la devolución devuelve el control a la persona que llama. La diferencia es que una función de shell no puede cambiar sus parámetros, aunque puede cambiar los parámetros globales.
ejemplo usando funciones:

```
#!/bin/sh
# A simple script with a function...

add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
  echo "Adding user $USER ..."
  echo useradd -c "$COMMENTS" $USER
  echo passwd $USER $PASSWORD
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

###
# Main body of script starts here
###
echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script..."
```

La línea 4 se identifica como una declaración de función al terminar en (). Esto es seguido por , y todo lo que sigue al emparejamiento se toma como el código de esa función. Este código no se ejecuta hasta que se llama a la función. Las funciones se leen, pero básicamente se ignoran hasta que realmente se llaman. Otro punto importante es las funciones pueden ser recursivas.