Universidad de Sonora

Departamento de ciencias exactas

Física Computacional

Introducción a la porgramación de los intérpretes de comandos

Alumno:
Martha Anahí Iñiguez Beltrán

214202804

March 8, 2018

Contents

1 Introducción

En la siguiente actividad aprenderemos la programación de los intérpretes de comandos, en este caso interactuar y programar Scripts. Específicamente los Scripts estarán dirigidos a la manipulación de datos extraídos de bases de datos que contienen también información acerca del medio donde se tomaron los datos y otro tipo de especificaciones son innecesarias.

Los sistemas operativos UNIX y sistemas derivados (Linux y macOS), estan apoyados en un intérprete de comandos llamado Shell, el cual es intermediario entre el usuario y el sistema operativo.

Algunos intérpretes de comandos que existen son: -C Shell (/bin/csh),
-Bourne Shell (/bin/sh),
-Korn Shell (/bin/ksh/),
-Bourne Again Shell (/bin/bash),
- y otros.

Los intérpretes que utilizaremos serán /bin/sh y /bin/bash.

Algunos de los comandos a utilizar serán cat, chmod, echo, ls, etc. y se irá explicando su función.

2 Comandos

A continuación se enlistan los comandos utilizados para interactuar con nuestro script y se explica la función de cada uno de ellos.

2.1 cat

El comando cat enlaza archivos y los muestra como salida en la terminal. Lo que hacemos es explorar los datos de un archivo en la terminal sin la necesidad de un editor

Ejemplo

Supongamos que existe un archivo texto.txt el cual contiene datos (no está vacío). Al usar el comando cat podemos explorar cada uno de los datos que contiene. La sintáxis es la siguiente:

```
\$ cat texto.txt
```

2.2 chmod

El comando chmod nos permite cambiar los permisos del archivo a manipular. Los permisos pueden ser leer, editar, ejecutar o combinados entre ellos (leer-ejecutar, leer-editar, editar-ejecutar, leer-editar-ejecutar, etc). Cada tipo de permisos es llamado usando la notación numérica en base 8.

Ejemplo

Supongamos que tenemos un archivo llamado script.sh el cual solo tiene permisos para leer-editar, al usar el chmod podemos cambiar los permisos para ejecutar. La sintaxis es la siguiente:

```
\$ chmod 755 script.sh
```

2.3 echo

El comando echo 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.

Ejemplo

Si queremos imprimir la frase "Hello World" directamente en la terminal, la sintáxis será la siguiente:

```
\$ echo Hello World
```

2.4 grep

El comando grep sirve para buscar, en uno o más archivos, líneas que contengan una característica específica (frase, número, etc) e imprime todas las líneas donde se encuentre.

Ejemplo

Supongamos que tenemos un archivo texto.txt el cual contiene los siguientes datos:
12345
Hello World
Comment something
Computer
Operating System

Comment
Thing

Si queremos buscar la palabra Comment la sintaxis será la siguiente:

```
\$ grep Comment texto.txt
```

2.5 ls

El comando ls (del ingles list) nos muestra una lista con los archivos y directorios dentro de un directorio. El comando ls sin argumento muestra el listado de los elementos del directorio desde donde se encuentra.

Ejemplo

Tenemos un directorio que contiene los subdirectorios y archivos subdir1 subdir2 archivo1 archivo2. Al usar el comando ls seguido de los nombres de los archivos y subdirectorios especificados veremos una lista con el nombre de los archivos y los archivos dentro de los subdirectorios. La sintaxis será la siguiente:

```
\$ ls archivo1 subdir1 subdir2
```

Al este comando se le puede agregar argumento para que especifíque las características del listado que hará. Los siguientes son ejemplos de argumentos:

-a: muestra un listado en el formato largo, con información de permisos, número de enlaces asociados al archivo, usuario, grupo, tamaño y fecha de última modificación además del nombre.

-d: muestra solamente el nombre del subdirectorio, sin entrar en él ni dar un listado del contenido.

-L: en los enlaces simbólicos, muestra los datos del archivo referenciado en vez de los del link.

-1: muestra el listado en una sola columna. Sin la opción -1 el listado se muestra en varias columnas, tantas como permita el ancho de la terminal.

2.6 wc

El comando wc (del ingles word count) permite realizar diferentes conteos desde la entrada estándar, ya sea de palabras, caracteres o saltos de líneas.

El programa lee la entrada estándar o una lista concatenada y genera una o más de las estadísticas siguientes: conteo de líneas, conteo de palabras, y conteo de bytes. Si se le pasa como parámetro una lista de archivos, muestra estadísticas de cada archivo individual y luego las estadísticas generales.

Ejemplo

Supongamos que tenemos dos archivos, ideas.txt y excerp.txt. La sintaxis será la siguiente:

```
\$ wc ideas.txt excerpt.txt
```

Este comando, al igual que ls, al estar acompañado de argumento especifíca el conteo con características específicas. Algunos de los argumentos son:

-l: cuenta el número de líneas.

-c: cuenta el número de bytes.

-m: imprime el número de caracteres.

-L: imprime la longitud de la línea más larga.

-w: imprime el número de palabras.

3 Shell Script

A continuación se muestra una síntesis de las notas de Steve Parker: Shell Script Tutorial.

3.1 Introducción

Éste tutorial está escrito para ayudar a las personas a entender algunos de los conceptos básicos de la programación de scripts de shell. Existe una serie de factores básicos para hacer scripts de la manera correcta: los criterios más importantes deben diseñarse clara y legiblemente y debemos evitar comandos que no sean necesarios.

3.2 A first script

En la primera sección nos enseña a hacer un script básico.

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

La primera línea ordena a Unix que el archivo debe ser ejecutado por / bin / sh. La segunda línea utiliza el símbolo # para marcar la línea como comentario y así shell lo ignora al momento de compilar. La tercera línea ejecuta el comando echo que contiene dos parámetros: el primero es "Hello"; el segundo es "Word". echo colocará un espacio entre sus parámetros automáticamente. Despues se ejecuta el comando chmod 755 para hacer el script ejecutable y luego lo ejecuta.

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

Al final se termina esta sección editando el script.

3.3 Variables (Parte 1

A continuación veremos como agregar variables a un script

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

Se le asigna la cadena 'Hello World' a la variable MY_MESSAGE y con echo repite el valor de la variable. Al shell no le importan los tipos de variables; éstas pueden almacenar cadenas, enteros, números, reales, etc.

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

El anterior ejemplo muesta como en la tercera línea el mensaje "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.

3.4 Wildcards

Los comodines no son de mucha utilidad en la elaboración de scripts de shell. Esta sección va dedicada a como editar o mover archivos en la terminal. Estos son más utilizados cuando hay loops.

El siguiente ejemplo muestra como copiar los archivos de a hacia b.

```
$ cp /tmp/a/* /tmp/b/
$ cp /tmp/a/*.txt /tmp/b/
$ cp /tmp/a/*.html /tmp/b/
```

El agregar .txt despues del * significa que copiara todos los archivos de texto hacia b.

3.5 Escape Characters

Ciertos caractéres son importantes para el shell; hemos visto, por ejemplo, que el uso de la comilla doble ("") afecta la forma en que se tratan los espacios y caractéres.

La mayoría de los caracteres (*, ', etc) no se interpretan (es decir, que se toman literalmente) por medio de la colocación entre comillas dobles (""). Se toman como están y se pasan al comando que se llama. Un ejemplo usando el asterisco (*) es:

```
$ echo * caso . shtml escapar . shtml primero . shtml

funciones . shtml consejos . índice shtml . shtml
ip - primer . txt raid1 + 0.txt
$ echo * txt
ip - primer . txt raid1 + 0.txt
$ echo "*" *
$ echo "* txt" * txt
```

En el primer ejemplo, * se expande para indicar todos los archivos en el directorio actual.

En el segundo ejemplo, * txt significa todos los archivos que terminan en txt. En el tercero, ponemos * entre comillas dobles, y se interpreta literalmente. En el cuarto ejemplo, lo mismo se aplica, pero hemos agregado txt a la cadena.

3.6 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 for-while bucles en el shell Bourne.

for Loops

Los bucles for iteran a través de un conjunto de valores hasta que se agote la lista:

```
#! / bin / sh para i en 1 2 3 4 5 do
  echo "Looping ... número $ i" hecho
```

While Loops

Un ejemplo del While bucle es el siguiente:

```
#! / bin / sh
INPUT_STRING = hola
 mientras [ "$ INPUT_STRING" ! = "bye" ] no
  echo "Por favor escriba algo en (adiós a dejar de fumar)"

  leer INPUT_STRING
  echo "Escribiste: $ INPUT_STRING" hecho
```

Lo que ocurre aquí es que las instrucciones de eco y lectura se ejecutarán indefinidamente hasta que escriba "bye" cuando se le solicite.

3.7 Test

El Test es utilizado por prácticamente todos los guiones de shell escritos. Puede que no parezca así, porque a menudo no se llama test durectamente. test es más frecuentemente llamado así [. [es un enlace simbólico test, solo para hacer que los programas shell sean más legibles. También es normalmente un shell incorporado (lo que significa que el intérprete de comandos interpretará el [como test.

```
$ type [ [ es un shell incorporado

$ 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 raíz 35368 27 de marzo de 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:

```
if [ $ foo = "bar" ]
```

No funcionará; se interpreta como if test$foo = "bar"], que es un]"sin comienzo [".

3.8 Case

La declaración case guarda pasar por un conjunto completo de las declaraciones if.. then.. else. Su sintaxis es realmente sencilla:

```
#!/bin/sh

echo "Please talk to me ...""Please talk to me ..."
while :while :
dodo
  read INPUT_STRING
  case $INPUT_STRING incase $INPUT_STRING in
hello))
echo "Hello yourself!""Hello yourself!"
;;;;
bye))
echo "See you again!""See you again!"
breakbreak
;;;;
*)*)
echo "Sorry, I don't understand""Sorry, I don't understand"
;;;;
  esacesac
donedone
echo
echo "That's all folks!""That's all folks!"
```

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

Las opciones que entendemos están listadas y seguidas por un corchete derecho, como hello) y bye). Esto significa que si INPUT_STRING coincide con hello, se ejecuta esa sección de código, hasta el punto y coma doble. Si INPUT_STRING

coincide con bye, se imprime el mensaje de despedida y termina el ciclo. Tenga en cuenta que si quisiéramos salir del script completamente, usaríamos el comando exit en lugar de break.

3.9 External Programs

Los programas externos a menudo se usan en scripts de shell; hay algunos comandos internos (echo, which, y test son comúnmente incorporados), pero muchos comandos útiles son en realidad utilidades Unix, tales como tr, grep, expry cut.

El backtick (‘) también se asocia a menudo con comandos externos. Debido a esto, discutiremos primero el backtick.

El backtick se usa para indicar que el texto adjunto se debe ejecutar como un comando. Esto es bastante simple de entender. Primero, use un intérprete interactivo para leer su nombre completo desde /etc/passwd:

```
$ grep "^ $ {USER}:" / etc / passwd | corte - d : - f5
 Steve Parker
```

Ahora tomaremos esta salida en una variable que podemos manipular más fácilmente:

```
$ MYNAME = ‘grep" ^ $ {USER}: "/ etc / passwd | corte -d: -f5‘
$ echo $ MYNAME
Steve Parker
```

Entonces vemos que el backtick simplemente captura el resultado estándar de cualquier comando o conjunto de comandos que decidamos ejecutar.

3.10 Functions

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.

Una función puede devolver un valor en cuatro formas diferentes:

-Cambiar el estado de una variable o variables
-Use el comando exit para finalizar el script de shell
-Utilice el comando return para finalizar la función y devolver 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.

Un script simple que usa una función se vería así:

```
#! / bin / sh
# Un script simple con una función ...# Un script simple con una función ...

add_a_user ()()
{{
  USUARIO = $ 1= $ 1
  CONTRASEÑA = $ 2= $ 2
  cambio; cambio;; cambio ;
  # Habiendo cambiado dos veces, el resto ahora son comentarios ...# Habiendo cambi
  COMENTARIOS = $ @= $ @
  echo "Agregar usuario $ USER ...""Agregar usuario $ USER ..."
  echo useradd -c "$ COMENTARIOS" $ USUARIO- c "$ COMENTARIOS" $ USUARIO
  echo passwd $ USER $ PASSWORD
  echo "Usuario agregado $ USER ($ COMENTARIOS) con pass $ PASSWORD""Usuario agrega
}}

######
# El cuerpo principal del script comienza aquí# El cuerpo principal del script comi
######
echo "Inicio del script ...""Inicio de la secuencia de comandos ..."
add_a_user bob letmein Bob Holness el presentadorBob Holness el presentador
add_a_user fred badpassword Fred Durst el cantanteFred Durst el cantante
add_a_user bilko worstpassword Sgt. Bilko el modelo a seguirSgt . Bilko el modelo a
echo "Fin del script ...""Fin del guión ..."
```

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.

4 Conclusión

Como conclusión, se anexa el script que editamos con las instrucciones que se nos dieron. El manejo de Scripts de shell ahora es más sencillo pues conocemos los comándos, para que funciona cada uno de ellos, así como la manipulación correcta para que el script haga exactamente lo que se le pide sin errores.

5 Bibliografía

- The Linux Information Project (2018) En linfo.org. Recuperado el 05 de marzo del 2018: http://www.linfo.org/index.html

- Parker, S. (2017) Shell Script Tutorial, shellscript.sh. Recuperado el 05 de marzo del 2018: https://www.shellscript.sh/index.html

- Atmospheric soundings (2018) Weather.uwyo.edu. Recuperado el 05 de marzo del 2018: http://weather.uwyo.edu/upperair/sounding.html

6 Apéndice

A continuación se responden algunas preguntas respecto a la actividad.

¿Qué fue lo que más te llamó la atención en esta actividad?

El manejo de shell y como el script puede programarse para que las bases de datos se descargaran automaticamente al ejecutarlo desde la terminal.

¿Qué consideras que aprendiste?

La edición de scripts y los diversos comandos que se pueden utilizar en la terminal.

¿Cuáles fueron las cosas que más se te dificultaron?

Siento que no se me dificulto nada en ésta actividad.

¿Cómo se podría mejorar en esta actividad?

Haciendo scripts más complejos o más elaborados.

¿En general, cómo te sentiste al realizar en esta actividad?

Siento que ahora tengo mayor conocimiento del uso de shell.