



"El saber de mis hijos
hará mi grandeza"

UNIVERSIDAD DE SONORA

División de Ciencias Exactas y Naturales

Licenciatura En Física

Física Computacional I

Reporte de Actividad 4

*"Introducción a la Programación de los Intérpretes
de Comandos"*

César Omar Ramírez Álvarez

Profr. Carlos Lizárraga Celaya

Hermosillo, Sonora

Febrero 21 de 2018

Introducción

Los sistemas operativos UNIX y otros sistemas derivados, como lo es Linux y macOS, se apoyan con un intérprete de comandos (Shell), quien juega el papel de intermediario entre el usuario y el sistema operativo. Un *Shell* es una interfaz de usuario para acceder a los servicios de un sistema operativo.

En general, las shells del sistema operativo utilizan una interfaz de línea de comandos (CLI) o una interfaz gráfica de usuario (GUI), dependiendo del rol de la computadora y de la operación particular. Se denomina shell porque es la capa más externa alrededor del núcleo del sistema operativo.

En la práctica de la semana se realizaron distintas acciones con el intérprete de comandos Shell. Fue utilizado para descarga de datos en conjunto, organización y filtración de los mismos, así como para crear archivos.

Durante el desarrollo del presente reporte se mostrará de forma general los aspectos prácticos necesarios para la realización de la actividad con los diferentes intérpretes de comandos, como lo fueron el uso del cat, chmod, echo, grep, etc. Incluyendo una descripción teórica a grandes rasgos de éstos. Por último a manera de síntesis se presenta un tutorial de Shell mostrando varios ejemplos rescatados de una página web recomendada.

Práctica y Resultados

Como una primera acción fue descargar y guardar en la carpeta Actividad 4 el archivo proporcionado por el profesor. El archivo contiene un script llamado "script.sh" que permite descargar de manera automática los registros de todos los días del año de una estación seleccionada de los sondeos atmosféricos.

Entrando en contexto, en la práctica anterior se trabajó con registros de solo dos meses, los cuales fueron descargados de manera individual. En esta ocasión, la descarga será automática y en conjunto por lo que se editó el script descargado colocando el número de la estación deseada. La estación con la que he trabajado es *Camborne Observations*.

Los registros de sondeos atmosféricos son tomados de la Universidad de Wyoming en la página web siguiente (<http://weather.uwyo.edu/upperair/sounding.html>).

A continuación se presenta el script utilizado:

```
#!/bin/bash

# Archivo "script1.sh"
# Script para bajar automáticamente los datos de sondeos atmosféricos de la
# Universidad de Wyoming (http://weather.uwyo.edu/upperair/sounding.html)
# para un rango de tiempo. Sustituir el valor de la estación de interés:
# Por ejemplo Chihuahua es la estación número: 76225

STATION=03808

# Despues de editar este archivo, ejecuta el comando: chmod 755 script1.sh
# Para ejecutar el script: ./script1.sh

# Definimos el separador de valores de las variables, en este caso es ":"
IFS=":"

# Por ejemplo nos interesan bajar los datos de: 2013-2017
LISTYs="2017"

# Lista de meses por días
LISTM31="01:03:05:07:08:10:12"
LISTM30="04:06:09:11"
LISTM28="02"

for j in $LISTYs ; do
# Meses 31 días
for i in $LISTM31 ; do
    /usr/bin/wget "http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=$j&MONTH=$i&FROM=0100&TO=3112&STNM=$STATION"
# Reposo 5 segundos
    /bin/sleep 5
done
# Meses 30 días
for i in $LISTM30 ; do
    /usr/bin/wget "http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=$j&MONTH=$i&FROM=0100&TO=3012&STNM=$STATION"
# Reposo 5 segundos
    /bin/sleep 5
done
# Feb. 28 días
for i in $LISTM28 ; do
    /usr/bin/wget "http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=$j&MONTH=$i&FROM=0100&TO=2812&STNM=$STATION"
```

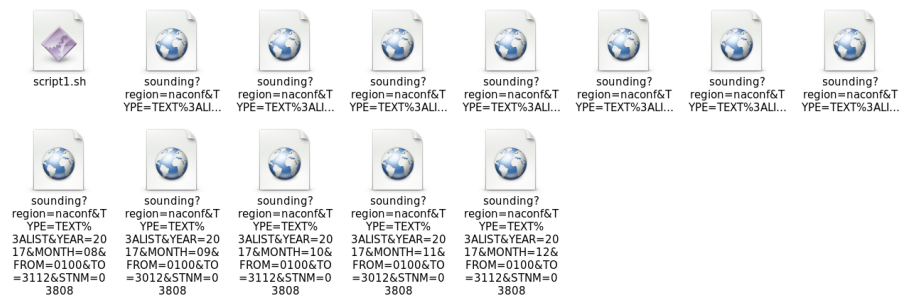
La finalidad del script.sh, era descargar automáticamente los registros de todo un año de los sondeos meteorológicos de Camborne Observations , la cuál, es la estación con la que he trabajado. En el contenido del script es visible como estan declaradas las cosas, pues son variables tanto la estación como los meses (aclarando que para estos se definen la cantidad de días correspondientes, así evitamos datos basura) y se realizan ciclos "DO" dependiendo de los días que contiene cada mes, dejando descanso de 5 segundos entre mes y mes. Descarga los datos usando un comando llamada wget.

Éste script descargado con cuenta con el permiso para ser ejecutado, desde nuestra terminal con el comando "ls -alg" hacemos que se muestren los archivos y el tipo de permiso que éstos cuentan, notamos que nuestro archivo tiene los permisos: -rw-r--r--, que significa que es solamente de lectura para el dueño, posteriormente con el comando "chmod" cambiamos los permisos del archivo. Esto se hace posible mediante un cambio a notación en base 8, donde para este caso se utiliza 755. Usamos de nuevo el comando "ls -alg" y verificamos que ya es ejecutable.

```
cesaromar97@ltsp26:~/Computacional1/Actividad4$ ls -alg
total 12
drwxr-xr-x 1 users 4096 mar  4 12:43 .
drwxr-xr-x 1 users 4096 mar  4 12:42 ..
-rw-r--r-- 1 users 1416 mar  4 12:43 script1.sh
cesaromar97@ltsp26:~/Computacional1/Actividad4$ chmod 755 script1.sh
cesaromar97@ltsp26:~/Computacional1/Actividad4$ ls -alg
total 12
drwxr-xr-x 1 users 4096 mar  4 12:43 .
drwxr-xr-x 1 users 4096 mar  4 12:42 ..
-rwxr-xr-x 1 users 1416 mar  4 12:43 script1.sh
cesaromar97@ltsp26:~/Computacional1/Actividad4$
```

Siendo ejecutable el script podemos correrlo y observamos el progreso de descarga de cada uno de los 12 archivos correspondientes.

```
cesaromar97@ltsp26:~/Computacional1/Actividad4$ ./script1.sh
--2018-03-04 12:50:05-- http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&
TYPE=TEXT%3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=3112&STNM=03808
Resolviendo weather.uwyo.edu (weather.uwyo.edu)... 129.72.27.74
Conectando con weather.uwyo.edu (weather.uwyo.edu)[129.72.27.74]:80... conectado
.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 1359825 (1.3M) [text/html]
```



```
cesaromar97@ltsp26:~/Computacional1/Actividad4$ emacs script1.sh
cesaromar97@ltsp26:~/Computacional1/Actividad4$ cat sounding* > sondeos.txt
cesaromar97@ltsp26:~/Computacional1/Actividad4$ ls -alg
total 28388
drwxr-xr-x 1 users 4096 mar  4 12:56 .
drwxr-xr-x 1 users 4096 mar  4 12:42 ..
-rwxr-xr-x 1 users 1398 feb 15 17:07 script1.sh
-rw-r--r-- 1 users 14519583 mar  4 12:56 sondeos.txt
-rw-r--r-- 1 users 1359825 mar  4 12:50 sounding?region=naconf&TYPE=TEXT%3ALIST
&YEAR=2017&MONTH=01&FROM=0100&TO=3112&STNM=03808
-rw-r--r-- 1 users 1080283 mar  4 12:52 sounding?region=naconf&TYPE=TEXT%3ALIST
&YEAR=2017&MONTH=02&FROM=0100&TO=2812&STNM=03808
-rw-r--r-- 1 users 1232395 mar  4 12:50 sounding?region=naconf&TYPE=TEXT%3ALIST
&YEAR=2017&MONTH=03&FROM=0100&TO=3112&STNM=03808
```

Continuando, para el caso de lectura y revisión de los datos, hicimos uso de dos comandos muy parecidos en cierta forma, estos comandos son "less" y "cat". Ambos muestran en la terminal todos los datos de un archivo sin la posibilidad de editarlo, solo examinar datos se podría decir. La única diferencia entre estos dos comandos es que el cursor cuando aplicamos less aparece al inicio del archivo y cuando aplicamos cat lo manda al final.

```
<H2>03808 Camborne Observations at 00Z 01 Jan 2017</H2>
<PRE>
```

PRES	HGHT	TEMP	DWPT	RELH	MIXR	DRCT	SKNT	THTA	THTE	THTV
hPa	m	C	C	%	g/kg	deg	knot	K	K	K
1015.0	88	8.8	7.7	93	6.53	230	7	280.8	298.8	281.9
1014.0	96	9.0	7.9	93	6.63	231	8	281.0	299.4	282.2
1000.0	207	8.6	7.3	92	6.45	240	15	281.8	299.7	282.9
995.0	248	8.3	7.1	92	6.39	240	18	281.8	299.6	282.9
984.0	340	7.6	6.6	93	6.25	247	19	282.1	299.4	283.1
965.0	501	6.7	3.9	82	5.26	260	22	282.7	297.5	283.6
959.0	553	6.4	3.0	79	4.98	260	22	282.9	297.0	283.8
951.0	621	5.8	3.6	86	5.24	260	21	283.0	297.8	283.9
943.0	690	5.2	3.4	88	5.21	260	21	283.1	297.8	284.0

Otro comando utilizado es el "grep" que nos permite tomar cierta información en específico de un archivo renglón por renglón (si la hay). La manera de hacerlo es primeramente indicando el comando, la palabra o frase a buscar y el archivo de donde quieres que se busque.

```
cesaromar97@ltsp26:~/Computacional/Actividad4$ grep CAPE sounding\?region\=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=3112&STNM=03808
```

```
CAPE using virtual temperature: 31.97
CAPE using virtual temperature: 2.42
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.09
CAPE using virtual temperature: 0.14
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
CAPE using virtual temperature: 0.00
```

Ahora bien, teniendo los archivos descargados verificamos a qué tipo pertenecen con el comando "file" esta vez agregando sounding* (por el hecho de que sounding* es el inicio de todos los archivos descargados y queremos saber el tipo de todos ellos) y notamos que son archivos de texto ASCII.

```
cesaromar97@ltsp26:~/Computacional/Actividad4$ file sounding*
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=3112&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=02&FROM=0100&TO=2812&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=3112&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=04&FROM=0100&TO=3012&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=05&FROM=0100&TO=3112&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=06&FROM=0100&TO=3012&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=07&FROM=0100&TO=3112&STNM=03808: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=08&FROM=0100&TO=3112&STNM=03808: HTML document, ASCII text
```

Nos piden juntar todos los archivos en uno solo, por lo que usaremos el comando cat pero agregandole un redirector ">" para indicar que todos estos archivos irán a un archivo .txt. Con esto ahora tenemos un archivo "sondeos.txt". A continuación, hicimos uso de grep, para filtrar los renglones de información que se quiere conservar (la deseada). Esto crea un archivo llamado df2017.csv, éstas ultimas dos acciones realizadas pueden ser automatizadas mediante el uso de un scripts. En la actividad nos piden crear un script que contenga estas dos indicaciones con ayuda de él estas acciones podrán hacerse rápidamente

sin la necesidad de hacerlo manualmente. Se crea el script con el editor de textos emacs colocando los comandos necesarios (los anteriores), con un cambio solamente en el archivo de destino, llamándolo df2017_2.csv.

```
egrep -v 'PRES|hPa' sondeos.txt | egrep '03808|Showalter|...|Precip' > df2017.csv
```

```
#!/bin/bash
#Archivo "filtro.sh"
#Script utilizado para la optimización del proceso de datos usado en clase.
#Datos anuales obtenidos de la estación 03808

cat sounding* > sondeos.txt #Recopilación de los archivos en uno.

egrep -v 'PRES|hPa' sondeos.txt | egrep '03808|Showalter|LIFT|SWEAT|K|Totals|CAPE|
CINS|LFCT|CAPV|Temp|Pres|thick|Precip' > df2017_2.csv #Filtro de datos.
```



```
<LINK REL="StyleSheet" HREF="/resources/select.css" TYPE="text/css">
<H2>03808 Camborne Observations at 00Z 01 Jan 2017</H2>
      Showalter index: 8.48
      LIFT computed using virtual temperature: 6.91
      SWEAT index: 79.02
      K index: 1.30
      Totals totals index: 43.20
      CAPE using virtual temperature: 31.97
      CINS using virtual temperature: -3.62
      LFCT using virtual temperature: 917.06
      Bulk Richardson Number using CAPV: 2.75
      Temp [K] of the Lifted Condensation Level: 278.73
      Precipitable water [mm] for entire sounding: 13.55
```

Teniendo estos dos archivos, como no cambio nada en su contenido, solo la forma de hacerlo (manual y automático) no debe existir ninguna diferencia. Para comprobar eso se hace uso del comando "diff", si existe tal diferencia la muestra en la terminal y si no, no muestra nada.

```
cesaromar97@ltsp26:~/Computacional1/Actividad4$ diff df2017.csv df2017_2.csv
cesaromar97@ltsp26:~/Computacional1/Actividad4$ █
```

Breve Descripción de Comandos Utilizados

- **cat:** Muestra el contenido de archivos y concatena archivos.
- **ls:** Lista archivos y directorios. Al agregar "-alg" muestra los permisos de los archivos y directorios.

- ***chmod***: Cambia los permisos de un archivo o carpetas.
- ***diff***: Busca y muestra diferencias entre archivos.
- ***echo***: Imprime una línea de texto, variables, o contenido a un archivo.
- ***file***: Determina el tipo de archivo.
- ***grep***: Busca patrones de cadenas dentro de archivos.
- ***less***: Muestra el contenido de un archivo, permite búsquedas y movimiento hacia atrás y adelante.
- ***wget***: Descargador de archivos desde Internet, no interactivo.
- ***wc***: Cuenta palabras, líneas, caracteres de un archivo o listado.
- ***redirectores***: "|" separa datos, ">" envía información (datos o texto) a un archivo.

Síntesis *"Shell Script Tutorial"*

Introducción

Este tutorial está escrito para ayudar a las personas a comprender algunos de los conceptos básicos de la programación de scripts de shell (también conocido como shell scripting), y con suerte para presentar algunas de las posibilidades de la programación simple pero potente disponible bajo el shell Bourne. Está dirigido a la audiencia con conocimientos básicos en programación en Unix/ Linux Shell, es preferiblemente necesario conocer algunos comandos más comunes (*ls*, *echo*, *cp*, *etc.*).

Filosofía

La programación de script de Shell tiene una mala impresión entre algunos administradores de sistemas de Unix. Esto debido a la velocidad a la que se ejecutará un programa interpretado en comparación con un programa C y a la existencia de muchos scripts de shell de mala calidad. Es de suma importancia conocer los comandos para realizar un buen script y mantenerlo limpio.

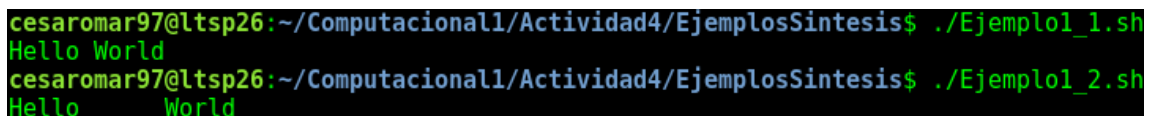
Un Primer Script

Para nuestro primer script de shell, solo escribiremos un script que diga "Hello World". El código es:

```
#!/bin/sh
# Este es un comentario!
echo Hola Mundo          # Este también es un comentario!
```

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 el caparazón lo ignora por completo. La única excepción es cuando la primera línea del archivo comienza con `#!`- como lo hace la nuestra. Por ultimo el tercer renglón contiene el comando `echo`, el cual imprime en pantalla lo que se escribe después del punto. Podemos observar que `echo` automáticamente puso un espacio entre las palabras, pero si colocamos mas espacios shell lo interpretará como solamente uno, para poder realizar cambios de este estilo, debemos de colocar el texto entre comillas:

```
#!/bin/sh
# Este es un comentario!
echo "Hola          Mundo"          # Este también es un comentario!
```



```
cesaromar97@ltsp26:~/Computacional1/Actividad4/EjemplosSintesis$ ./Ejemplo1_1.sh
Hello World
cesaromar97@ltsp26:~/Computacional1/Actividad4/EjemplosSintesis$ ./Ejemplo1_2.sh
Hello      World
```

Variables I

En casi todos los lenguajes de programación existe el concepto de variables, un nombre simbólico a un trozo de memoria al cual podemos asignarle un valor, leer y manipular su contenido. Para asignar variables se debe de igualar el nombre de la variable a lo que se desea almacenar, pero sin espacios.

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

A shell no le importa el tipo de variable, pueden contener caracteres, enteros, reales, lo que se necesite. Pero si sabe diferenciar entre ellos. Por lo que éstas se debe mantener del mismo tipo y no intentar mezclar números con letras. Inclusive se puede utilizar variables junto con el comando `read` para leer un dato insertado de entrada y depositarlo en la variable.


```
#!/bin/sh
echo What is your name?
read MY_NAME
echo "Hello $MY_NAME - hope you're well."
```

Cabe resaltar que las variables se reinician cada vez que ejecutamos el script. Además si queremos juntar nuestra variable con otro dato, se hace entre corchetes curvos (llaves) para anunciar cuando inicia y acaba la variable, para que no lo considere todo.

```
#!/bin/sh
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I will create you a file called ${USER_NAME}_file"
touch "${USER_NAME}_file"
```

Wildcards (Comodines)

Son solamente comandos para utilizar en la terminal con la finalidad de facilitar el movimiento y edición de archivos de distintas sintaxis. Como un ejemplo, si queremos copiar los archivos de la carpeta "a" a la carpeta "b", es necesario usar el siguiente código:

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

Por lo general, se usan los wilcards para movimiento de archivos, y nos es tan común su uso en los scritps.

Caracteres de Escape

Algunos caracteres tienen significado en shell, como ejemplo tenemos a las dobles comillas que afectan en como se tratan los espacios en el comando echo. Pero si deseamos utilizar las comillas como texto es necesario hcaer uso de la barra diagonal inversa, esto no solo aplica para comillas si no para cualquier caracter especial que shell no interpreta. Lo siguiente imprime : Hello "World"

```
$ echo "Hello \"World\""
```

La mayoría de los caracteres no son interpretados como texto, para eso tienen que ser colocados entre comillas, por el contrario se toman como código.

```
echo *  
echo *txt  
echo "*"   
echo "*txt"
```

Lo que nos va a imprimir según los casos es lo que indica lo siguiente: en el primer renglón, * se expande para indicar todos los archivos en el directorio actual. En el segundo renglón, * 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. Sin embargo, ", \$, ', y \ todavía son interpretados por el shell, incluso cuando están entre comillas dobles. El carácter de barra invertida (\) se utiliza para marcar estos caracteres especiales para que el intérprete no los interprete, sino que los pase al comando que se está ejecutando.

Ciclos

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. Existen dos tipos de ciclos, los "for" y los "while".

Ciclos For

Estos ciclos iteran a través de un conjunto de valores hasta que se agote la lista. Tenemos el siguiente ejemplo:

```
#!/bin/sh  
for i in hello 1 * 2 goodbye  
do  
echo "Looping ... i is set to $i"  
done
```

Este ejemplo nos imprime en pantalla con un carácter (hello) y número (1), después los archivos del directorio actual, número (2) y por último carácter (goodbye).

Ciclos While

Estos ciclos repiten una sentencia que se ejecuta al menos una vez y es reejecutada cada vez que la condición se evalúa a verdadera.

```
#!/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

```

Este ejemplo seguirá imprimiendo hola, hasta que el usuario escriba "bye".

Test

Test es usado en casi todos los scripts, pero esto no parece ser así debido a que usualmente no se le llama mediante el comando test. Este comando es llamado frecuentemente con el símbolo '[', que hace referencia a este comando. Test nos sirve para comparar y revisar archivos y su contenido. Debe estar rodeado de espacios, por el contrario se interpretará como texto. A menudo se invoca indirectamente a través de las instrucciones if y while. Se puede dar mejor orden a las indicaciones del test, dando restricciones a lo que puede o no dar el usuario, mediante el uso del comando grep, un ejemplo:

```

#!/bin/sh
echo -en "Please guess the magic number: "
read X
echo $X | grep "[^0-9]" > /dev/null 2>&1
if [ "$?" -eq "0" ]; then
    # If the grep found something other than 0-9
    # then it's not an integer.
    echo "Sorry, wanted a number"
else
    # The grep found only 0-9, so it's an integer.
    # We can safely do a test on it.
    if [ "$X" -eq "7" ]; then
        echo "You entered the magic number!"
    fi
fi

```

En el ejemplo anterior, se nos pide insertar un número en el intervalo del [0,9], si el número es correcto te dice que encontraste el número mágico. En el siguiente ejemplo, tiene como finalidad indicar si el texto que ingresaste realmente es texto, si solo damos enter acaba el proceso.

```
#!/bin/sh
X=0
while [ -n "$X" ]
do
    echo "Enter some text (RETURN to quit)"
    read X
    echo "You said: $X"
done
```

Case

El comando CASE permite ahorrarnos el uso del conjunto completo "if-else-if...". Su sintaxis es realmente bastante simple:

```
#!/bin/sh

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

Las opciones válidas del ejemplo son "hello" y "bye", si lo introducido coincide con alguna de estas dos cadenas, se muestra en pantalla lo correspondiente a su opción, mientras que la última opción *), es en caso de que no coincida ninguna de las dos.

Variables II

Existen un conjunto de variables establecidas, y la mayoría de ellas no pueden tener valores asignados. Contienen información útil referente que el script puede utilizar para conocer el entorno en el que se está ejecutando. 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. Como ejemplo:

```
#!/bin/sh
echo "I was called with $# parameters"
echo "My name is $0"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are @$0"
```

Las otras dos variables principales que le asigna el entorno son \$\$ y \$!. Estos son ambos números de proceso. Otra variable es interesante IFS. Este es el separador de campo interno. El valor predeterminado es SPACE TAB NEWLINE, pero si lo está cambiando, es más fácil tomar una copia, como se muestra a continuación:

```
#!/bin/sh
old_IFS="$IFS"
IFS=:
echo "Please input some data separated by colons ..."
read x y z
IFS=$old_IFS
echo "x is $x y is $y z is $z"
```

Variables III

Como ya se ha mencionado anteriormente, las llaves {} alrededor de una variable evitan confusiones:

```
foo=sun
echo $fooshine      # $fooshine is undefined
echo ${foo}shine    # displays the word "sunshine"
```

Aquí las llaves permiten que se imprima "sunshine" en el segundo renglón, teniendo valor nulo en el primero. En general se explica como lidiar con variables indefinidas y nulas. Con esto y usando "-" se puede especificar valor default de una variable si no se encuentra definida.

```
#!/bin/sh
echo -en "What is your name [ 'whoami' ] "
read myname
if [ -z "$myname" ]; then
    myname='whoami'
fi
echo "Your name is : $myname"
```

Programas Externos

Los programas externos a menudo se usan en scripts de shell; hay algunas órdenes internas (echo, which, y test son comúnmente incorporados), pero muchos comandos útiles son en realidad utilidades Unix, tales como tr, grep, expr y cut. El backtick (comilla inversa (`)) se usa para indicar que el texto adjunto se debe ejecutar como un comando.

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

Este ejemplo imprime en pantalla la información y archivos con terminación .html

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

Funciones

Las funciones son fáciles de usar dentro del script del Bourne Shell. 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 "librería" 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 una de cuatro formas diferentes:

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 return para finalizar la función y devolver el valor proporcionado a la sección de llamada del script de shell.
- Con una salida echo a una entrada estándar.

```
#!/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..."
```

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. En este caso, la función `add_a_user` se lee y se comprueba la sintaxis, pero no se ejecuta hasta que se llame explícitamente.

Los programadores acostumbrados a otros lenguajes pueden sorprenderse con las reglas de alcance para las funciones de shell. Básicamente, no hay una definición del alcance, aparte de los parámetros (`$1`, `$2`, `$@`, etc.). Tomando el siguiente segmento de código simple:

```
#!/bin/sh

myfunc()
{
    echo "I was called as : $@"
    x=2
}

### Main script starts here

echo "Script was called with $@"
x=1
echo "x is $x"
myfunc 1 2 3
echo "x is $x"
```

La variable @ cambia con la función para reflejar que la función fue llamada. Pero la variable x es una variable global, el cambio aún es efectivo a pesar de regresar al script principal.

Consejos y Sugerencias

Se presentarán algunos tips de comandos a manera de ejemplificación con la finalidad de facilitar la creación y uso de los scripts.

Telnet es una técnica útil, aunque telnet ya no se usa en los servidores, todavía lo utilizan algunos dispositivos de red, como los concentradores de terminales y similares. Al crear un script como este, su propio script o desde una línea de comando, puede ejecutar.

```
#!/bin/sh
host=127.0.0.1
port=23
login=steve
passwd=hellothere
cmd="ls /tmp"

echo open ${host} ${port}
sleep 1
echo ${login}
```



```
sleep 1
echo ${passwd}
sleep 1
echo ${cmd}
sleep 1
echo exit
```

Otra utilidad útil es sed: el editor de flujo. El cual cambia, borra o reemplaza palabras de un archivo en específico.

```
SOMETHING="This is a bad word."
echo ${SOMETHING} | sed s/"bad word"/g
```

Referencias

Esta es una guía de referencia rápida sobre el significado de algunos de los comandos y códigos menos fáciles de adivinar de los scripts de shell.

Command	Description	Example
&	Run the previous command in the background	ls &
&&	Logical AND	if ["\$foo" -ge "0"] && ["\$foo" -le "9"]
	Logical OR	if ["\$foo" -lt "0"] ["\$foo" -gt "9"] (not in Bourne shell)
^	Start of line	grep "^foo"
\$	End of line	grep "foo\$"
=	String equality (cf. -eq)	if ["\$foo" = "bar"]
!	Logical NOT	if ["\$foo" != "bar"]
\$\$	PID of current shell	echo "my PID = \$"
\$_	PID of last background command	ls & echo "PID of ls = \$_"
\$_	exit status of last command	ls ; echo "ls returned code \$_"
\$0	Name of current command (as called)	echo "I am \$0"
\$1	Name of current command's first parameter	echo "My first argument is \$1"
\$9	Name of current command's ninth parameter	echo "My ninth argument is \$9"
\$@	All of current command's parameters (preserving whitespace and quoting)	echo "My arguments are \$@"
\$*	All of current command's parameters (not preserving whitespace and quoting)	echo "My arguments are \$*"
-eq	Numeric Equality	if ["\$foo" -eq "9"]
-ne	Numeric Inequality	if ["\$foo" -ne "9"]
-lt	Less Than	if ["\$foo" -lt "9"]
-le	Less Than or Equal	if ["\$foo" -le "9"]
-gt	Greater Than	if ["\$foo" -gt "9"]
-ge	Greater Than or Equal	if ["\$foo" -ge "9"]
-z	String is zero length	if [-z "\$foo"]
-n	String is not zero length	if [-n "\$foo"]
-nt	Newer Than	if ["\$file1" -nt "\$file2"]
-d	Is a Directory	if [-d /bin]
-f	Is a File	if [-f /bin/ls]
-r	Is a readable file	if [-r /bin/ls]
-w	Is a writable file	if [-w /bin/ls]
-x	Is an executable file	if [-x /bin/ls]
(...)	Function definition	function myfunc() { echo hello }

Shell Interactivo

Shell tambien puede ser usado mediante otras formas que no son scripts. Primeramente bash tiene algunas herramientas de búsqueda de historia muy prácticas; las teclas de flecha hacia arriba y hacia abajo se desplazarán por el historial de comandos anteriores. Más útilmente, Ctrl + r hará una búsqueda inversa, haciendo coincidir cualquier parte de la línea de comando. Presione ESC y el comando seleccionado se pegará en el shell actual para que pueda editarlo según sea necesario. Por otra parte el ksh se le pueden agregar comandos del historial, es posible abrir una nueva sesión ksh desde otro shell interactivo, y luego regresar al anterior.

Conclusión

En general el realizar tanto la práctica como el presente reporte fue ininteresante, ya que se enlazó bastante la teoría con lo práctico, además de que es la primera vez que se trabaja con shell script. Algo que en lo personal era desconocido en un principio para mi, por que se habia estado trabajando con lenguajes interpretados. Sin pensar ya estabamos haciendo scripts. Con bastante esfuerzo y sobre todo análisis e interpretación de teoría se llegó a concluir con la actividad.

Bibiografía

- Atmospheric Soundings. (2018). Weather.uwyo.edu. Recuperado el 16 de Febrero de 2018, desde <http://weather.uwyo.edu/upperair/sounding.html>
- Comandos de Linux mas frecuentemente utilizados. (2018). Linuxtotal.com.mx. Recuperado el 20 de Febrero de 2018, desde https://www.linuxtotal.com.mx/?cont=info_admon_002
- Shell de Unix. (2018). Es.wikipedia.org. Recuperado el 19 de Febrero de 2018, desde https://es.wikipedia.org/wiki/Shell_de_Unix
- Shell Scripting Tutorial. (2018). Shellscrip.sh. Recuperado el 21 de Febrero de 2018, desde <https://www.shellscrip.sh/index.html>

Apéndice

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

Primeramente el uso de nuevos comandos con el interprete de comandos de Unix, recuerdo que en el curso de Programación y Lenguaje Fortran en las prácticas introductorias hablamos sobre algunos de ellos que generalmente eran para la creación de directorios, los vistos aquí se podría decir que son de mas nivel. Claro, destacando tambien que el uso de scripts es una herramienta muy util para automatizar el uso de comandos que fueron presentados.

2. ¿Qué consideras que aprendiste?

En general, el uso de scripts como una forma de automatizar y también de forma manual.

3. ¿Cuáles fueron las cosas que más se te dificultaron?

La manera de utilizar los comandos, ya que cambian de una forma a otra y es diferente como se trabajan.

4. ¿Cómo se podría mejorar en esta actividad?

Pienso que es necesario practicar con los comandos y tener una explicación teórica (origen, uso y manera de abreviación) de ellos, en las que se propongan ejercicios interactivos para su comprensión.

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

Tenía inseguridad en un principio, pues me di cuenta que sería complicado por que me iba a enfrentar a algo nuevo, por el lado de la síntesis, la información de ella estaba en inglés y se me iba a complicar por mi nivel de comprensión, pero conforme el desarrollo, aunque estuvo bastante larga logré terminarla.