Universidad de Sonora

Licenciatura en Física

Física Computacional I

Actividad 4 - Introducción a la programación de los intérpretes de comandos.

Alumno:
José Gabriel Navarro I.

Profesor:
Carlos Lizarraga Celaya

21 de Febrero de 2018

1 Introducción

En el presente reporte se habla acerca de la cuarta actividad realizada para la clase de Física Computacional I, el cual abarca el uso de un interprete de comandos de UNIX llamado Shell. En la actividad se realizaron distintas acciones con este interprete como es la descarga de datos, su organización y filtración, además de la creación de archivos.

En el reporte se habla acerca del proceso realizado en la practica ademas de su resultado al utilizar varios comandos en el interprete, como lo fue el uso de cat, chmod, echo, grep entre otros. También se da una pequeña descripción de cada uno de ellos de manera teórica. Además de esto se presenta una síntesis acerca de un tutorial de Shell, en donde se muestran varios ejemplos presentados en la pagina web.

2 Actividades realizadas

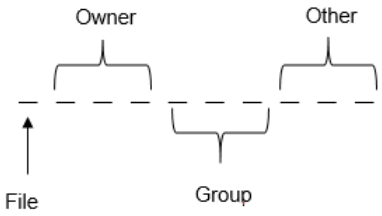
Primeramente se descargo el archivo proporcionado por el profesor. Este archivo es un script que permite descargar todos los registros de los días en un año de una estacion seleccionada de los sondeos climáticos. El script fue editado de manera que la estación seleccionada fuera la misma estación que la practica anterior: Observatorio de Valentia.

El script se muestra a continuación:

```
#!/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=03953

# 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 dias
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"
# Reposa 5 segundos
        /bin/sleep 7
done
# Meses 30 dias
for i in $LISTM30 ; do
        /usr/bin/wget "http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=$j&MONTH=$i&FROM=0100&TO=3012&STNM=$STATIO
# Reposa 5 segundos
        /bin/sleep 7
done
# Feb. 28 dias
for i in $LISTM28 ; do
```

La función de este script, como ya se había mencionado, es descargar todos los registros del año del sondeo meteorológico de una estación, el Observatorio de Valentia en mi caso. Podemos observar como en el script, primero se declaro como variable a la estación y los distintos meses dependiendo del numero de días que tiene, esto para que al descargar los datos no se descargue datos basura. Posteriormente en los ciclos 'DO', podemos observar como dependiendo del mes que es es el numero de veces que se repite el proceso, cambiando el URL de descarga dependiendo de la estación y del mes. Para descargar los datos se utiliza el comando wget.

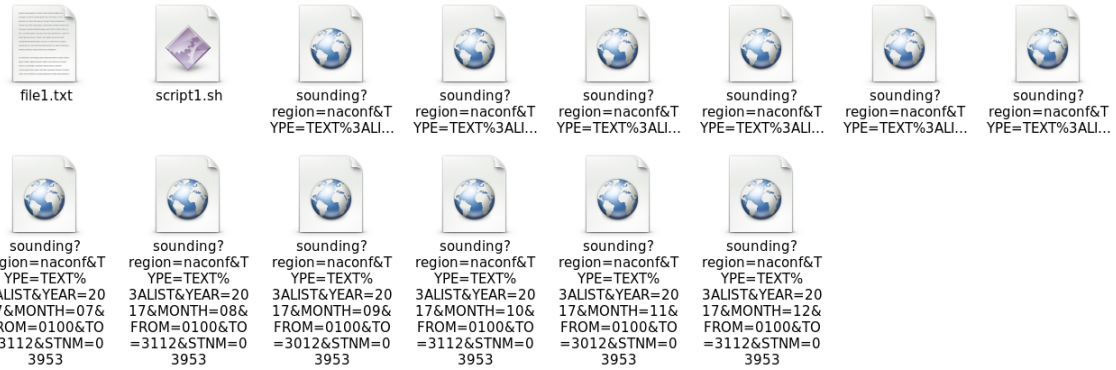
Sin embargo, el script aun no es un ejecutable, utilizando el comando ***ls- alg*** nos muestra los permisos del archivo, en donde podemos observar que solo es para lectura y edición. Los permisos del archivo se clasifican de la siguiente manera:



Al realizar este comando, observamos que el archivo tiene los permisos: -rw-r–r–, que significa que es solamente de lectura para el dueño. Para cambiarlo a ejecutable se utiliza el comando ***chmod***, que permite cambiar los permisos de un archivo, usando la notación numérica en base 8. Entonces, para cambiarlo a un ejecutable usamos la notación 755, en donde el resultado será:

```
drwxr-xr-x 1 users    4096 feb 19 11:40 .
drwxr-xr-x 1 users    4096 feb 15 16:14 ..
-rw-r--r-- 1 users       5 feb 15 16:57 file1.txt
-rwxr-xr-x 1 users    1401 feb 15 17:12 script1.sh
-rwxr-xr-x 1 users    1401 feb 15 17:08 script1.sh~
```

Ahora que es un ejecutable, al correrlo se empieza a descargar todos los archivos, un total de 12 archivos, mostrando el progreso de descarga en cada uno y tomándose un descanso cada 7 segundos entre archivos. El resultado obtenido fue:



```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ emacs script1.sh
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ cat sounding* >sondeos.txt
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ ls -alg
total 9868
drwxr-xr-x 1 users    4096 feb 19 11:40 .
drwxr-xr-x 1 users    4096 feb 15 16:14 ..
-rw-r--r-- 1 users       5 feb 15 16:57 file1.txt
-rwxr-xr-x 1 users    1401 feb 15 17:12 script1.sh
-rwxr-xr-x 1 users    1401 feb 15 17:08 script1.sh~
-rw-r--r-- 1 users 5032188 feb 19 11:40 sondeos.txt
-rw-r--r-- 1 users  441893 feb 15 17:08 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=01&FROM=0100&TO=3112&STNM=03953
-rw-r--r-- 1 users  388774 feb 15 17:11 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=02&FROM=0100&TO=2812&STNM=03953
-rw-r--r-- 1 users  428528 feb 15 17:08 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=03&FROM=0100&TO=3112&STNM=03953
-rw-r--r-- 1 users  429917 feb 15 17:10 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=04&FROM=0100&TO=3012&STNM=03953
-rw-r--r-- 1 users  417479 feb 15 17:08 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=05&FROM=0100&TO=3112&STNM=03953
-rw-r--r-- 1 users  405361 feb 15 17:11 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=06&FROM=0100&TO=3012&STNM=03953
-rw-r--r-- 1 users  420244 feb 15 17:09 sounding?region=naconf&TYPE=TEXT%3ALIST&
YEAR=2017&MONTH=07&FROM=0100&TO=3112&STNM=03953
```

Para la lectura y revisión, se utilizaron dos comandos, uno de ellos es ***less***. Este comando muestra en la terminal todos los datos de un solo archivo en pantalla, pero muestra todo seguido, es decir, la terminal no baja hasta el punto final, permite examinar todos los datos de inicio a fin, pero solamente permite la visualización de la información, no permite editarlos. Por otro lado el comando ***cat*** muestra todos los datos en pantalla, al igual que less, pero este pone el cursor al final del archivo, de manera que permite visualizar desde del fin del archivo hacia arriba en si concatena los datos para mostrar la información del archivo.

```
<H2>03953  Valentia Observatory Observations at 12Z 01 Dec 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
-----------------------------------------------------------------------------
1030.0     30    7.0    2.3     72   4.40    105      4  277.8  290.1  278.5
1028.0     45    8.4  -15.6     17   1.11    101      4  279.3  282.7  279.5
1025.0     69    6.6  -11.4     26   1.57     96      5  277.8  282.4  278.1
1000.0    264    4.6   -3.4     56   2.99     50      8  277.8  286.2  278.2
 927.0    876   -0.6   -3.4     81   3.22     10     23  278.6  287.7  279.1
 925.0    893   -0.7   -3.4     82   3.23     10     23  278.6  287.8  279.1
 903.0   1085   -2.3   -3.6     91   3.26     14     24  278.9  288.1  279.4
 876.0   1326   -0.5  -14.5     34   1.43     20     25  283.2  287.5  283.4
 863.0   1446    0.8  -16.2     27   1.26     22     25  285.7  289.6  285.9
 850.0   1568    1.0  -15.0     29   1.41     25     25  287.2  291.5  287.4
 818.0   1877    1.6  -16.4     25   1.30     22     27  291.0  295.1  291.2
 777.0   2289   -0.5  -19.5     22   1.06     18     29  293.0  296.4  293.2
 758.0   2487    0.4  -13.6     34   1.77     16     30  296.1  301.7  296.4
 700.0   3120   -4.5  -14.5     46   1.78     10     33  297.5  303.1  297.8
 596.0   4362  -13.9  -17.1     77   1.69      8     40  300.6  306.0  300.9
 566.0   4752  -15.9  -22.9     55   1.08      7     42  302.7  306.3  302.9
 514.0   5468  -22.3  -25.0     79   0.98      5     46  303.4  306.7  303.6
:
```

El comando ***grep*** permite tomar información en especifico de un archivo por renglón. Esto se hace indicando el comando, la palabra a buscar y el archivo de donde se obtendrá esa información. Esto puede hacerse con varias palabras, de distintos archivos e

incluso limitarlo aun cierto numero de renglones. grep significa global regular expression print, ya que se usa en su mayoría para buscar texto o buscar en archivos lineas especificas.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ grep SWEAT sounding\?region\=na
conf\&TYPE\=TEXT%3ALIST\&YEAR\=2017\&MONTH\=02\&FROM\=0100\&TO\=2812\&STNM\=0395
3
                                SWEAT index: 159.05
                                SWEAT index: 323.85
                                SWEAT index: 321.92
                                SWEAT index: 254.61
                                SWEAT index: 172.90
                                SWEAT index: 316.16
                                SWEAT index: 294.05
                                SWEAT index: 285.47
                                SWEAT index: 194.56
                                SWEAT index: 252.05
                                SWEAT index: 277.73
                                SWEAT index: 241.69
                                SWEAT index: 265.99
                                SWEAT index: 258.22
                                SWEAT index: 66.05
                                SWEAT index: 223.34
                                SWEAT index: 182.17
                                SWEAT index: 116.55
                                SWEAT index: 89.36
                                SWEAT index: 27.20
                                SWEAT index: 52.45
```

Ahora vamos a juntar todos estos archivos en uno solo. Utilizando el comando *file sounding**, podremos ver el tipo de archivo que hay en nuestra carpeta enfocándose solamente en los archivos que comiencen con sounding, en este caso es un archivo de texto ASCII:

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ file sounding*
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=3112&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=02&FROM=0100&TO=2812&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=3112&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=04&FROM=0100&TO=3012&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=05&FROM=0100&TO=3112&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=06&FROM=0100&TO=3012&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=07&FROM=0100&TO=3112&STN
M=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=08&FROM=0100&TO=3112&STN
M=03953: HTML document, ASCII text
```

Para juntar todos los archivos en uno solo usaremos el comando cat de nuevo pero esta vez usaremos el redirector $>$ para indicar que todos estos archivos irán a un archivo .txt. De esta manera queda creado el archivo.

Ahora para obtener solamente los renglones que deseamos, utilizaremos el comando grep, seleccionando solamente los renglones que estan al final de la tabla. Como ya se había mencionado, grep permite elegir renglones con distintas palabras, esto mediante el uso de |, separando las palabras a buscar. El comando en si es:

```
egrep -v 'PRES|hPa' sondeos.txt | egrep '76225|Showalter...Precip' > df2017.csv \\
```

En donde el indicador -v significa que va tomar todos los datos excepto los que contengan las palabras PRES y hPa. Mientras que el resto del código toma los renglones que presenten las palabras Sowalter, Precip, etc.

Estas ultimas dos acciones realizadas pueden ser automatizadas mediante el uso de un script. Si creamos un script que contenga estas dos indicaciones, y cambiándolo a un ejecutable, estas acciones podrán hacerse rápidamente sin la necesidad de hacerlo manualmente. Para esto solo se creo un archivo de emacs con terminación sh, se puso los comandos anteriormente mencionados, con un cambio solamente en el archivo de destino, ahora siendo df2017_2.csv. Posteriormente se convirtió a un ejecutable y se corrió. Esto genero el archivo que se muestra a continuación.

```
<LINK REL="StyleSheet" HREF="/resources/select.css" TYPE="text/css">
<H2>03953  Valentia Observatory Observations at 00Z 01 Jan 2017</H2>
                              Showalter index: 7.76
   LIFT computed using virtual temperature: 12.15
                                 SWEAT index: 99.07
                                     K index: 20.60
                         Totals totals index: 44.10
            CAPE using virtual temperature: 0.00
            CINS using virtual temperature: 0.00
          Bulk Richardson Number using CAPV: 0.00
  Temp [K] of the Lifted Condensation Level: 275.03
Precipitable water [mm] for entire sounding: 18.79
<H2>03953  Valentia Observatory Observations at 12Z 01 Jan 2017</H2>
                              Showalter index: 13.05
   LIFT computed using virtual temperature: 12.60
                                 SWEAT index: 159.29
                                     K index: -15.50
                         Totals totals index: 36.80
            CAPE using virtual temperature: 6.10
            CINS using virtual temperature: -1.41
            LFCT using virtual temperature: 899.30
          Bulk Richardson Number using CAPV: 0.12
  Temp [K] of the Lifted Condensation Level: 270.23
Precipitable water [mm] for entire sounding: 8.20
<H2>03953  Valentia Observatory Observations at 00Z 02 Jan 2017</H2>
                              Showalter index: 19.03
   LIFT computed using virtual temperature: 18.28
                                 SWEAT index: 137.92
                                     K index: -60.70
                         Totals totals index: 6.60
            CAPE using virtual temperature: 0.00
            CINS using virtual temperature: 0.00
          Bulk Richardson Number using CAPV: 0.00
```

Este archivo deberia ser exactamente igual al creado anteriormente mediante el método manual, para comprobar esto se uso el comando ***diff***, el cual indica la diferencia entre dos archivos. Si existe una diferencia se mostrara en pantalla, si no lo hay, no se muestra nada como se observa a continuación:

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ diff df2017.csv df2017_2.csv
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$
```

Algunos comandos utilizados también fueron los de ls, wc y echo. ***echo*** permite escribir o mandar información aun archivo, por ejemplo, se puede crear un archivo y dentro de el se puede escribir algo utilizando este comando, además también permite visualizar la ubicación en donde se encuentra actualmente la terminal:

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ echo 1234 > file1.txt
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ echo $PATH
/opt/anaconda3/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Ls se utilizo mucho al revisar los archivos y sus permisos, esto es porque ls permite ver el contenido de los directorios, y agregando -alg, permite vernos también los permisos que tiene.

wc imprime en pantalla el numero de lineas, palabras o bytes en un archivo, esto indicado ya sea por c, l o w, seguido por el nombre del archivo, por ejemplo, en este archivo hay 5632 lineas:

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$ wc -l sounding\?region\=naconf\
&TYPE\=TEXT%3ALIST\&YEAR\=2017\&MONTH\=02\&FROM\=0100\&TO\=2812\&STNM\=03953
5632 sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=02&FROM=0100&TO=281
2&STNM=03953
jose_navarro@ltsp26:~/ComputacionalI/Actividad4$
```

3 Síntesis

A continuación se presenta la síntesis de la pagina Shell Script, donde se muestra un poco de teoría acerca del tema y posteriormente algunos ejemplos:

3.1 Introducción y Filosofía

La programación de Shell tiene un poco de mala fama entre algunos sistemas administrativos de Unix. Esto se debe a la rapidez en la que un programa interpretado corre a comparación de un programa en C, en C es mas rápido y por tanto preferible, y como es muy fácil escribir este tipo de scripts, existen muchos scripts de shell de poca calidad.

Es por eso que es importante que el script a realizar sea claro y legible, tratando de evitar comandos innecesarios. Esto ya que el sistema operativo tarda mas, entre mas comandos tenga que realizar.

3.2 Un primer script

Para el primer script a realizar, haremos el programa clásico: "Hola Mundo". Para esto solamente crearemos un script con lo siguiente:

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

La primera linea le dice a Unix que el archivo debe ser ejecutado en \bin\sh, La segunda linea indicada por un #, es un comentario, y es ignorada por Shell, siendo la única

excepción cuando este símbolo esta acompañado por !, lo cual significa que no importa que shell estemos utilizando este debe de ser interpretado por el Shell original. Por ultimo el tercer renglón contiene el comando echo, el cual imprime en pantalla lo que se escribe después del punto.

Algo a notar, es que echo automáticamente puso un espacio entre las palabras, pero si colocamos mas espacios shell lo interpretara como solamente uno, para poder realizar cambios de este estilo, debemos de colocar el texto entre comillas:

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

Aquí se puede notar la diferencia:

3.3 Variables

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.

A shell no le importa el tipo de variable, pueden contener caracteres, enteros, reales, lo que se necesite. Pero si sabe diferenciar entre ellos; no se puede hacer la suma de un carácter y un numero. Inclusive se puede utilizar variables junto con el comando read para interactuar con la computadora:

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

Si se declaran mal las variables el script no marcara error, si no que se imprimirá en pantalla la cadena de carácter vació, es decir, nada. Si se quiere utilizar variables declaradas DESPUÉS de su impresión, es necesario exportarlas, ya que cada vez que corremos nuestro script, el shell se reinicia y pierde el valor.

Ahora bien, no se puede usar el nombre o contenido de una variable uniéndola a otra, es decir, no podemos nombrar algo con nuestra variable a menos que se utilicen corchetes como se indica:

```
#!/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
```

3.4 Wildcards (Comodines)

Estos son solamente comandos para utilizar en la terminal para facilitar el movimiento y edición de archivos de distintas sintaxis. Por ejemplo, si queremos copiar todos los archivos de una carpeta 'a' otra carpeta 'b' podemos utilizar:

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

O si queremos imprimir en pantalla la lista de archivos en una carpeta, pero sin utilizar ls, podemos usar un echo \tmp\a*.

Generalmente se utilizan las "wildcards" para facilitar el movimiento de archivos, y casi no se usan en scripts.

3.5 Caracteres de Escape

Algunos caracteres tienen significado en shell, por ejemplo, las dobles comillas afectan en como se tratan los espacios en el comando echo. Pero si deseamos utilizar las comillas como texto, ¿cómo lo hacemos?

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

Esto permite imprimir: Hello "World", que era lo deseado.

La mayoría de los caracteres no son interpretados como texto, para eso tienen que ser colocados entre comillas, si no el interpretador los toma como código.

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

En el primer renglón, * imprimirá todos los archivos en el directorio, y *txt imprimirá todos los archivos .txt en pantalla. Mientras que en el renglón 3 y 4 se imprimirá el carácter. Sin embargo, algunos caracteres como $ y ' aun son interpretados por shell. Es por eso que se utiliza el carácter \para su uso, tal como se muestra en el primer ejemplo.

3.6 Ciclos

La mayoría de los lenguajes tienen lo que son los ciclos. Si queremos repetir algo veinte veces, no tenemos que escribir el código 20 veces cambiándolo un poco cada vez. Existen dos tipos de ciclos, los "for" y los "while"

3.6.1 Ciclos For

Estos ciclos iteran hasta que la lista de valores se acabe, los valores pueden ser cualquier cosa, por ejemplo, el siguiente código imprime en pantalla con un carácter, numero, después toma todos los archivos del directorio actual, numero y por ultimo carácter:

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

3.6.2 Ciclos While

En estos ciclos, las instrucciones dentro de este se repetirán indefinidamente hasta que se cumpla la condición de salida, por ejemplo, el siguiente código muestra un ciclo que seguirá hasta que el usuario escriba "bye", si no, el ciclo sigue 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
```

3.7 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. Permite revisar y comparar tipos de archivos y su contenido.

Para que funcione este debe de estar separado entre espacios, si no lo interpreta como texto y no funciona correctamente. Este comando generalmente es llamado por los comandos if y while. Para terminar esta sintaxis, if se escribe al revés: 'fi'. Un ejemplo de esto puede ser:

```
#!/bin/sh
x= VALOR
if [ "$X" -lt "0" ]
then
  echo "X is less than zero"
fi
```

```
if [ "$X" -gt "0" ]; then
  echo "X is more than zero"
fi
[ "$X" -le "0" ] && \
      echo "X is less than or equal to  zero"
[ "$X" -ge "0" ] && \
      echo "X is more than or equal to zero"
[ "$X" = "0" ] && \
      echo "X is the string or number \"0\""
[ "$X" = "hello" ] && \
      echo "X matches the string \"hello\""
[ "$X" != "hello" ] && \
      echo "X is not the string \"hello\""
[ -n "$X" ] && \
      echo "X is of nonzero length"
[ -f "$X" ] && \
      echo "X is the path of a real file" || \
      echo "No such file: $X"
[ -x "$X" ] && \
      echo "X is the path of an executable file"
[ "$X" -nt "/etc/passwd" ] && \
      echo "X is a file which is newer than /etc/passwd"
```

Dependiendo del valor que tiene x, los corchetes evalúan el valor de esta variable e imprime en pantalla lo correspondiente a su valor. En este caso, x tiene un valor numérico:

Como se puede notar, este valor numérico cumple con varios de los exámenes, ya que este es mayor a 0, es mayor igual a 0, no es igual a la cadena de caracteres "hello" y no es ningún tipo de archivo. Antes de buscar en los tipos de archivo, se pueden poner distintas restricciones, como lo es "nt", que significa "newer than", que busca un archivo mas nuevo que el que se indica. Cabe mencionar que si estas restricciones deben de ser correspondientes a el tipo de variable que se usa, no se puede usar lt o gt, con archivos, ya que solo son para numéricos.

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!"
   else
    echo "That's not the magic number!"
  fi
fi
```

La indicación de grep[0,9] encuentra las lineas de texto que contiene los dígitos (0-9) y otros caracteres, y con el símbolo de ^encuentra las lineas que no consisten de solamente números. El apartado de >/dev/null dirige la salida de error a la parte especial "null", en vez de que aparezca en pantalla.

3.8 Case

El comando CASE permite ahorrarnos el uso de una dacena de "if-else-if..." para hacer un menu de opciones. Un ejemplo simple es:

```
#!/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!"
```

El comando de case siempre sigue la misma sintaxis, las opciones que aceptamos de entrada son seguidas por un paréntesis derecho, siendo en nuestro caso: 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 ultima opción *), es en caso de que no coincida ninguna de las dos. De igual manera que en el comando if, para cerrar el comando case, se escribe al revés: esac.

3.9 Variables 2

Existe un conjunto de variables que ya están definidas y no se le pueden asignar valores. Contienen información útil referente al archivo o al directorio.

El primer conjunto de variables son del $0 a $9 y $#. La variable $0 es la base de donde el programa fue llamado, mientras que las variables del 1 al 9 son parametros adicionales con la que fue llamado el script. Por 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 $@"
```

Al correrlo, se mostraran los parámetros correspondientes a la variable. Las otras dos variables principales son $$ y $!. Ambas son números de proceso. Otra variable interesante es IFS. Este es el "Internal Field Separator". Su valor default es el espacio, pero si se desea cambiar es mejor hacer una copia:

```
#!/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"
```

3.10 Variables 3

Anteriormente ya se habia mencionado el uso de { } para evitar confusiones entre las variables:

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

Estos corchetes permiten que se imprima "sunshine" en la segunda linea, mientras que la primera tendrá un valor nulo. Con este atributo y el uso de "-" podemos tambien podemos especificar un valor default a una variable si esta no esta definida:

```
#!/bin/sh
echo -en "What is your name [ `whoami` ] "
read myname
echo "Your name is : ${myname:-`whoami`}"
```

3.11 Programas externos

Los programas externos se usan normalmente dentro de los scripts, algunos de los comandos ya vienen preparados, como lo es el caso para echo y test, pero algunos de los comandos mas usados como grep y cut son utilidades de Unix.

La comilla inversa `, indica que el texto que esta encerrado debe ser ejecutado como un comando, por ejemplo:

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

Ademas de que se puede utilizar para imprimir archivos de una manera mas eficiente:

```
#!/bin/sh
VAR=`grep -w Uno JoseNavarro_file.txt`
echo $GREP
```

3.12 Funciones

Una utilidad que la mayoría de veces se pasa por alto en Bourne Shell, es que se pueden escribir funciones para el uso de los scripts. Esto se hace generalmente de dos formas, se puede declarar en el mismo archivo o se le puede llamar a una librería. Estas funciones pueden devolver un valor en cuatro maneras distintas: cambiar variables, usar el comando exit para salir del script, usar el comando return para terminar la función y darle este valor al script, o una salido echo a stdout. Un ejemplo simple es:

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

En donde se puede observar que la función es llamada add_a_user, y es llamada en la parte principal del script.

El alcance de las variables en shel es distinto a en otros lenguajes, basicamente no existe, solo para parametros $1, $2, @, etc. Por ejemplo:

```
#!/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 mundial, el cambio aun es efectivo a pesar de regresar al script principal.

3.13 Tips y trucos

A continuación se presentan algunos tips de algunos comandos en forma de ejemplo, para facilitar mas el uso de scripts:

Telnet es la interfaz de usuario para utilizar el protocolo TELNET, a pesar de no ser usado mucho ya en servidores.

```
#!/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
```

Otro comando que puede ayudar, es el de sed, que puede cambiar, borrar o reemplazar palabras en especifico en un archivo. Esto es distinto a grep ya que grep borraria toda la linea en vez de solamente la palabra:

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

3.14 Shell Interactivo y referencias

En esta sección se presentan nombres de variables utilizadas ya por shell y algunos de las especificaciones que se le pueden dar a los comandos como lo son -eq, -lt, etc.

Shell tambien puede ser usado mediante otras formas que no son scripts, como lo son comandos de teclado, or ejemplo, las flechas de arriba y hacia abajo muestral la lista de codigos usados recientemente, ctrl+r hace una busqueda en reversa, y si usamos la tecla ESC el comando seleccionado sera pegado en el shell actual para ser editado.

4 Bibliografía

- Ls command in Linux.(2018) Recuperado de: www.rapidtables.com/code/linux/ls.html
- Wc. (2014) Recuperado de: www.tutorialspoint.com/unix_commands/wc.htm
- Shell Scripting Tutorial. (2018). Recuperado de: www.shellscript.sh/index.html

5 Apéndice

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

Utilizar nuevos comandos en el interprete de comandos de UNIX. Anteriormente en Fortran las primeras practicas consistieron en el uso de estos comandos para creación de carpetas y archivos, pero esto comandos son mucho mas avanzados. Esto sin mencionar el uso de los scripts una herramienta muy util para automatizar estos mismo comandos.

2. ¿Qué consideras que aprendiste?

El uso de scripts para automatizar procesos y aprender a hacer manualmente estos procesos.

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

Entender la sintaxis de los comandos, cada uno es distinto y tiene su forma de trabajar, lo cual dificulta un poco en aprenderse que hace cada uno y su forma de usarlo.

4. ¿Cómo se podría mejorar en esta actividad?

Una pequeña introducción teórica al uso de los comandos y de donde se originan, para así tener contexto de su uso y abreviaciones.

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

Me sentí bien, un poco inseguro debido a que es algo que no había visto antes y me daba miedo no saber manejarlo, pero con la investigación y bibliografía, ademas de la ayuda del profesor y compañeros salio todo bien.