

Reporte de Actividad 4

Introducción a la programación de los intérpretes de comandos

Michelle Contreras Cossio

21 de Febrero del 2018

1 Introducción

En el siguiente reporte resume los resultados obtenidos al realizar la Actividad #4 de la clase de Física Computacional 1. En esta tenemos una introducción a "Shell", donde usamos y definimos sus comandos principales y aprendemos a escribir scripts que automatizan algunas acciones. Además, se realizó una síntesis del tutorial "Shell Scripting Tutorial", escrito por Steve Parker.

2 Fundamentos

El sistema operativo Linux, con el cual trabajamos en el centro de cómputo, utiliza un intérprete de comandos, llamado Shell, el interprete de comandos es el programa que recibe lo que se escribe en la terminal y lo convierte en instrucciones para el sistema operativo.

Existe una gran variedad de intérpretes de comandos, como son: C Shell (/bin/csh), Bourne Shell (/bin/sh), Korn Shell (/bin/ksh), Bourne Again Shell (/bin/bash), y otros. En este caso trabajamos con /bin/bash/ y /bin/sh (Bourne Again Shell y Bourne Shell, respectivamente).

Como se mencionó, trabajaremos con scripts, un script, archivo de órdenes, archivo de procesamiento por lotes que por lo regular se almacena en un archivo de texto plano. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario.

3 Resumen de Comandos

- **cat**: Lee archivos y los muestra en pantalla (pero los muestra rápidamente, no es para la lectura del usuario), puede además concatenarlos en un solo archivo de salida.
- **chmod**: Este comando permite cambiar los permisos de acceso (leer, editar, ejecutar), sobre el archivo, por parte del dueño, del grupo de usuarios registrados y de cualquier usuario en general.
- **echo**: Es un comando para la impresión de un texto en pantalla. Es comúnmente utilizado dentro de scripts
- **grep**: Significa global regular expression print, es usado para buscar la ubicación o renglones que contengan cierta cadena de texto dentro de un archivo y los muestra.
- **less**: Permite ver, de manera detallada, el contenido de un archivo, pero no editarlo.
- **ls**: Enlista el contenido (archivos o directorios) que se tiene en el directorio asignado. Al añadirle la bandera "-alg", nos permite ver los permisos de acceso que se tienen sobre los archivos.
- **wc**: Realiza un conteo, de palabras, caracteres o saltos de línea, del archivo indicado.

- **Redirectores:** |, >: El símbolo "|" crea "pipes", que conectan dos comandos, donde la salida de un programa es la entrada del otro. Por otro lado, el caracter ">" redirecciona la salida de un comando ejecutado a un archivo que se creará.

4 Actividad Realizada

En esta sección se añade la actividad que se realizó para conocer los comandos principales del shell.

1. Se inició utilizando un script proporcionado y adecuándolo a la estación de sondeos atmosféricos con la que trabajamos la semana pasada, la cual fue del Aeropuerto Internacional de Adelaide, en Australia.
2. Posteriormente, se utilizó el comando `ls -alg` para observar los permisos con los que contaba este archivo.

```
michellecc@ltsp24:~/Computacional1/Actividad4$ ls -alg
total 16
drwxr-xr-x 1 users 4096 feb 15 16:37 .
drwxr-xr-x 1 users 4096 feb 15 16:15 ..
-rw-r--r-- 1 users 1416 feb 15 16:37 script1.sh
-rw-r--r-- 1 users 1436 feb 15 16:28 script1.sh~
```

3. Al no ser un archivo ejecutable, se cambió el permiso con el comando `chmod` y se procedió a ejecutar el script, el cual descargó 12 archivos (uno por mes) de la base de datos de la universidad de Wyoming.

```
michellecc@ltsp24:~/Computacional1/Actividad4$ chmod 755 script1.sh
michellecc@ltsp24:~/Computacional1/Actividad4$ ls -alg
total 16
drwxr-xr-x 1 users 4096 feb 15 16:37 .
drwxr-xr-x 1 users 4096 feb 15 16:15 ..
-rwxr-xr-x 1 users 1416 feb 15 16:37 script1.sh
-rw-r--r-- 1 users 1436 feb 15 16:28 script1.sh~
```

```
michellecc@ltsp24:~/Computacional1/Actividad4$ ./script1.sh
--2018-02-15 17:08:57-- http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672
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: 532997 (521K) [text/html]
Grabando a: "sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672"
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672 [532997/532997] 520.50K 313KB/s in 1.7s
--2018-02-15 17:08:59 (313 KB/s) - "sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672" guardado [532997/532997]
--2018-02-15 17:09:04-- http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=0100&STNM=94672
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: 582385 (569K) [text/html]
Grabando a: "sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=0100&STNM=94672"
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=0100&STNM=94672 [582385/582385] 568.66K 425KB/s in 1.3s
```

4. El comando `less` permitió observar con detenimiento un archivo. El comando `cat`, en cambio, lo mostró más rápido.
5. El comando `grep` permitió buscar renglones

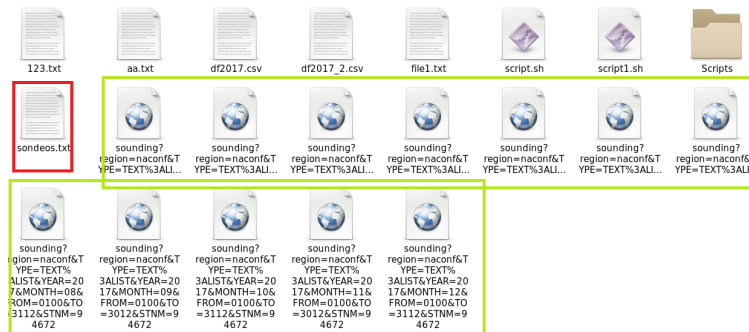
```
michellecc@ltsp51:~/Computacional1/Actividad4$ grep Observation sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672
<H2>94672 YPAD Adelaide Airport Observations at 00Z 01 Jan 2017</H2>
Observation time: 170101/0000
<H2>94672 YPAD Adelaide Airport Observations at 00Z 02 Jan 2017</H2>
Observation time: 170102/0000
```

6. El comando `file` nos dejó ver el tipo de archivo que eran los archivos descargados con el script.

```
michellecc@ltsp51:~/Computacional1/Actividad4$ file sounding*
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=01&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=04&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=05&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=06&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=07&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=08&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=09&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=10&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=11&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT&3ALIST&YEAR=2017&MONTH=12&FROM=0100&TO=0100&STNM=94672: HTML document, ASCII text
michellecc@ltsp51:~/Computacional1/Actividad4$
```

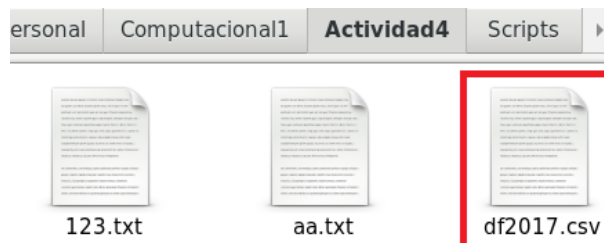
7. El comando cat se utilizó nuevamente para juntar los 12 archivos en uno txt. Con:

```
cat sounding* > sondeos.txt
```



8. Posteriormente, se utilizó el siguiente comando (en una sola linea):

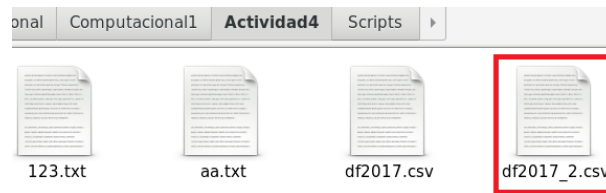
```
egrep -v 'PRES|hPa' sondeos.txt | egrep '94672|Showalter|LIFT|SWEAT|K|Totals  
|CAPE|CINS|LFCT|CAPV|Temp|Pres|thick|Precip' > df2017.csv
```



El cual retira los renglones que empiecen con 'PRES|hPa' del archivo creado en el paso 7. Posteriormente dejaba únicamente los renglones que comenzaban con alguna de las siguientes opciones: '94672|Showalter|LIFT|SWEAT|K|Totals|CAPE|CINS|LFCT|CAPV|Temp|Pres|thick|Precip' y el resultado lo mandó al archivo df2017.csv.

```
<LINK REL="StyleSheet" HREF="/resources/select.css" TYPE="text/css">
<H2>94672 YPAD Adelaide Airport Observations at 00Z 01 Jan 2017</H2>
  Station number: 94672
  Showalter index: 7.86
  LIFT computed using virtual temperature: 7.57
  SWEAT index: 246.31
  K index: 7.80
  Totals totals index: 36.90
  CAPE using virtual temperature: 0.34
  CINS using virtual temperature: -5.49
  LFCT using virtual temperature: 876.01
  Bulk Richardson Number using CAPV: 0.00
  Temp [K] of the Lifted Condensation Level: 284.54
  Precipitable water [mm] for entire sounding: 21.08
<H2>94672 YPAD Adelaide Airport Observations at 00Z 02 Jan 2017</H2>
  Station number: 94672
  Showalter index: 18.63
  LIFT computed using virtual temperature: 16.62
  SWEAT index: 41.01
  K index: -14.70
  Totals totals index: 13.80
  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: 280.72
  Precipitable water [mm] for entire sounding: 13.03
<H2>94672 YPAD Adelaide Airport Observations at 12Z 02 Jan 2017</H2>
  Station number: 94672
  Showalter index: 18.12
  LIFT computed using virtual temperature: 16.51
  SWEAT index: 36.99
  K index: -16.90
  Totals totals index: 15.20
  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: 282.46
  Precipitable water [mm] for entire sounding: 13.19
<H2>94672 YPAD Adelaide Airport Observations at 00Z 03 Jan 2017</H2>
  Station number: 94672
  Showalter index: 14.89
  LIFT computed using virtual temperature: 16.60
```

9. Se creó un script que automatizara el paso 7 y 8, creando un archivo llamado df2017_2.csv. Estos fueron comparados con el comando diff, verificando que no existiera ninguna diferencia entre ambos.



```
<LINK REL="StyleSheet" HREF="/resources/select.css" TYPE="text/css">
<H2>94672 YPAD Adelaide Airport Observations at 00Z 01 Jan 2017</H2>
      Station number: 94672
      Showalter index: 7.86
      LIFT computed using virtual temperature: 7.57
      SWEAT index: 246.31
      K index: 7.80
      Totals totals index: 36.90
      CAPE using virtual temperature: 0.34
      CINS using virtual temperature: -5.49
      Bulk Richardson Number using CAPV: 0.00
      Temp [K] of the Lifted Condensation Level: 284.54
      Precipitable water [mm] for entire sounding: 21.08
<H2>94672 YPAD Adelaide Airport Observations at 00Z 02 Jan 2017</H2>
      Station number: 94672
      Showalter index: 18.63
      LIFT computed using virtual temperature: 16.62
      SWEAT index: 41.01
      K index: -14.70
      Totals totals index: 13.80
      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: 280.72
      Precipitable water [mm] for entire sounding: 13.03
<H2>94672 YPAD Adelaide Airport Observations at 12Z 02 Jan 2017</H2>
      Station number: 94672
      Showalter index: 18.12
      LIFT computed using virtual temperature: 16.51
      SWEAT index: 36.99
      K index: -16.90
      Totals totals index: 15.20
      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: 282.46
      Precipitable water [mm] for entire sounding: 13.19
<H2>94672 YPAD Adelaide Airport Observations at 00Z 03 Jan 2017</H2>
      Station number: 94672
      Showalter index: 14.89
      LIFT computed using virtual temperature: 16.60
df2017.csv
```

```
michellecc@ltsp51:~/Computacional1/Actividad4$ diff df2017.csv df2017_2.csv
michellecc@ltsp51:~/Computacional1/Actividad4$
```

5 Síntesis de "Shell Scripting Tutorial"

NOTA: Todos los scripts creados como ejemplos se anexan en la carpeta "Scripts", con la misma numeración aquí mencionada.

5.1 Introducción

El propósito del tutorial es ayudar a programar utilizando scripts en el shell y enseñar lo básico por medio de ejemplos y ejercicios. No obstante, el tutorial no es para cualquier individuo, este asume conocimiento básico del uso del Shell, de programación y comandos de unix. Bourne Shell fue creado por Steve Bourne, este y el Bourne Again Shell son los que cubre el tutorial, sin embargo, existen otros como Korn Shell (ksh), C Shell (csh), entre otros.

Script #1:

Se creó un script usando los siguientes comandos:

```
echo '#!/bin/sh' > my-script.sh
echo 'echo Hello World' >> my-script.sh
```

Que obtuvo como salida:

```
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ chmod 755 script1.sh
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ ./script1.sh
Hello World
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$
```

5.2 Filosofía

La programación de Shell Scripts tiene un poca mala fama, debido a la que no es tan veloz, en comparación con C o Perl; la otra razón importante es que no es tan estricto con su sintaxis, lo que provoca que existan scripts con poca calidad.

Los criterios que se toman en cuenta para calificar un script como bueno, son: que tenga un diseño claro y legible, y que se evite el uso de comandos innecesarios, ya que esto provoca lentitud.

Sin embargo, cuando estos criterios se toman en cuenta al escribir el script, se tienen ciertas ventajas. Por ejemplo, que el diseño sea claro y legible permite que siga creciendo el script y que lo pueda mantener cualquier usuario, no solo el autor.

Algunas debilidades de la programación por Shell Scripting es que no se hace muy buen uso de la indentación y resultan más difíciles de leer. Por otro lado, otra debilidad, es que hay diferentes maneras de realizar una misma acción, sin embargo, a veces una es más eficiente que la otra, y si bien es cierto, si queremos realizar solo una acción, no es tan significativo, el problema se registra cuando la tenemos dentro de un ciclo.

Como recomendación, se añade que es necesario saber que los shell scripts siempre son susceptibles a errores y es importante siempre pedir retroalimentación cuando lo compartes.

5.3 Primer Script

Script #2:

```
#!/bin/sh
# Comentario
echo Hello World
```

La primera línea indica que el archivo se ejecuta con `/bin/sh`, es decir el Bourne Shell. La segunda línea empieza con `#`, que indica un comentario y se ignora lo que incluya ese renglón, a excepción del primer renglón que se empieza con `#!`, que indica que se debe interpretar lo que siga por el Bourne Shell.

La tercera línea muestra al comando `echo` con dos parámetros "Hello" y "World", `echo` pone automáticamente espacio entre los parámetros.

En la terminal se utiliza el comando `chmod` para cambiar los permisos sobre el archivo y permitir ejecutarlo, posteriormente se ejecuta:

```
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ chmod 755 script2.sh
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ ./script2.sh
Hello World
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$
```

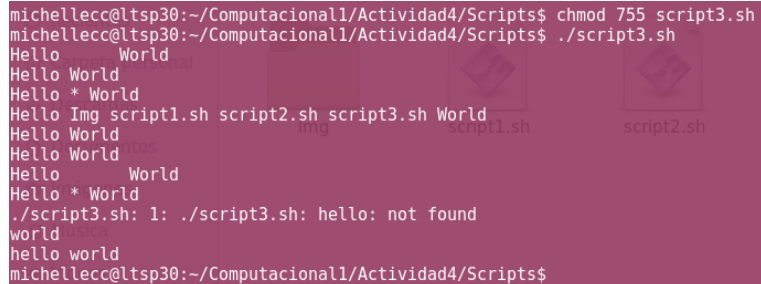
Es necesario diferenciar como funciona el comando `echo`, si se agregaran espacios a la línea de código `echo Hello World`, daría la misma salida que en la imagen anterior. Por otro lado, si se da la orden: `echo "Hello World"`, el resultado si nos daría los espacios exactamente como los ingresamos, esto debido a que pasa la cadena como un solo parámetro.

Script #3:

```
#!/bin/sh
#Comentario
echo "Hello      World"
echo "Hello World"
echo "Hello * World"
echo Hello * World
echo Hello      World
echo "Hello" World
echo Hello "      " World
echo "Hello "*" World"
```

```
echo 'hello' world
echo 'hello' world
```

Y dio como resultado, que permite diferenciar entre como deben ser ingresados los parámetros para que nos de el resultado que queramos:

A terminal window showing the execution of a script. The prompt is michellecc@ltsp30:~/Computacionall/Actividad4/Scripts\$. The command chmod 755 script3.sh is entered. Then, the command ./script3.sh is entered, and the output is: Hello World, Hello World, Hello * World, Hello Img script1.sh script2.sh script3.sh World, Hello World, Hello World, Hello * World, Hello * World, ./script3.sh: 1: ./script3.sh: hello: not found, world, hello world. The prompt returns to michellecc@ltsp30:~/Computacionall/Actividad4/Scripts\$.

```
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ chmod 755 script3.sh
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ ./script3.sh
Hello World
Hello World
Hello * World
Hello Img script1.sh script2.sh script3.sh World
Hello World
Hello World
Hello * World
Hello * World
./script3.sh: 1: ./script3.sh: hello: not found
world
hello world
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$
```

5.4 Variables (parte 1)

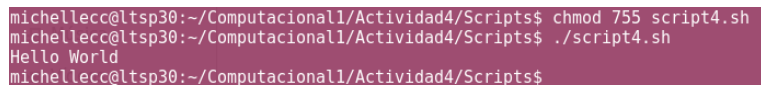
Como en todo lenguaje de programación, se tienen variables, a las cuales se les puede asignar valor y manipular su contenido.

Para que las variables funcionen, no deben existir espacios al momento de asignarles valor, es decir: VAR=value

Script #4:

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

Este script da el mismo resultado que los anteriores con los que hemos estado trabajando, sin embargo, este asigna a la variable "MY_MESSAGE" la cadena "Hello World", posteriormente lo imprime a la pantalla. Es necesario añadir que las variables solo admite un parámetro, por lo que si se tienen varias palabras, se deben poner entre comillas.

A terminal window showing the execution of script4.sh. The prompt is michellecc@ltsp30:~/Computacionall/Actividad4/Scripts\$. The command chmod 755 script4.sh is entered. Then, the command ./script4.sh is entered, and the output is: Hello World. The prompt returns to michellecc@ltsp30:~/Computacionall/Actividad4/Scripts\$.

```
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ chmod 755 script4.sh
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$ ./script4.sh
Hello World
michellecc@ltsp30:~/Computacionall/Actividad4/Scripts$
```

El shell permite que las variables sean de cualquier tipo, cadenas, enteros, reales. De hecho, todos los guarda como cadenas, así que si se busca operar con estas, no se podrá.

También se puede obtener el valor de las variables desde el usuario, usando el comando read.

Las variables en el Bourne Shell no necesitan ser declaradas, sin embargo, si se utiliza una variable que no está declarada, no marcará error, solo estará vacía, esto puede causar muchos errores. Para arreglar un poco eso existe el comando *export*, que, como su nombre lo dice, exporta la variable por otro programa y permite darle valor(momentáneo) a la variable desde fuera. Otra opción es al ejecutar el archivo poner ". ./".

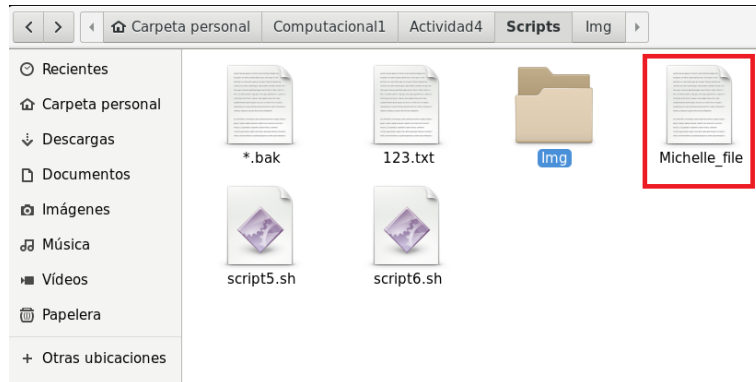
Script #5: Otra cosa que se puede hacer con las variables es usarlas como nombre de archivo, notese que se deben agregar llaves, ya que se añade "_file" al nombre de la variable:

```
#!/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"
```

Que da como resultado:

```
michellecc@ltsp30:~/Computacional1/Actividad4/Scripts$ chmod 755 script5.sh
michellecc@ltsp30:~/Computacional1/Actividad4/Scripts$ ./script5.sh
What is your name?
Michelle
Hello Michelle
I will create you a file called Michelle_file
michellecc@ltsp30:~/Computacional1/Actividad4/Scripts$
```

Y crea el archivo:



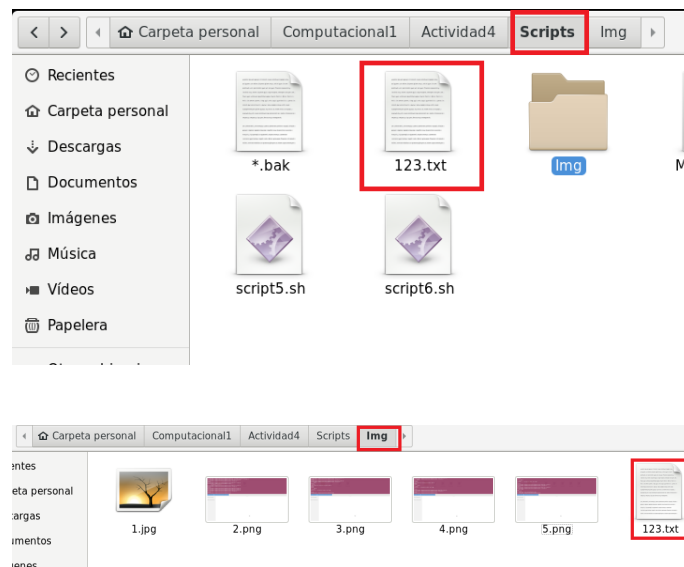
5.5 Wildcards

En español se conocen como *caracteres comodín*.

Script #6:

```
#!/bin/sh
cp /home/michellecc/Computacional1/Actividad4/Scripts/ *.txt/home/michellecc
/Computacional1/Actividad4/Scripts/Img
```

Este script mueve los archivos txt encontrados en el primer directorio, al segundo:

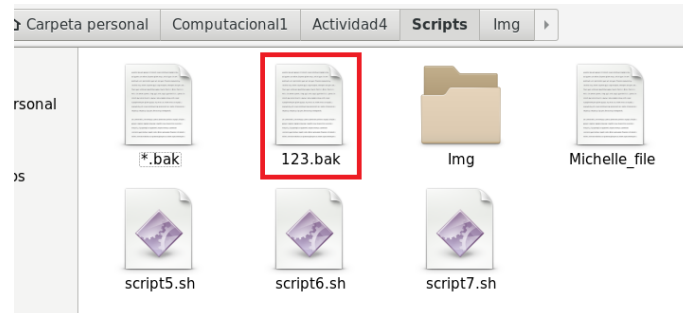


Script #7:

```
#!/bin/sh
files='ls -1 *.txt'
for f in *.txt; do
```

```
mv "$f" "${basename "$f" .txt}.bak"
done
```

Este script convierte a todos los archivos txt en bak:



5.6 Secuencias de escape

Existen algunos caracteres que ya tienen funciones dentro del Shell, por ejemplo, las comillas " sirven para formar cadenas, el asterisco * muestra los archivos dentro del directorio, el signo de pesos \$ llama a alguna variable, entre otros. El problema surge cuando queremos usar esos caracteres pero no para su función dentro del shell, solo como caracteres. Para ello se usa el backslash \.

Script #8: Se muestran las diferencias al trabajar con el asterisco:

```
#!/bin/sh
echo *
echo *txt
echo "*"
echo "*txt"
```

Que tiene como salida:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script8.sh
*
*txt
*
*txt
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Script #9: Se muestra como se utiliza el backslash para poder usar los caracteres mencionados:

```
#!/bin/sh
echo "A quote is \", backslash is \\, backtick is \`."
echo "A few spaces are   ; dollar is \$. \ $X is ${X}."
```

Que tiene como salida:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script9.sh
A quote is ", backslash is \, backtick is `
A few spaces are   ; dollar is $. $X is $X
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

5.7 Ciclos

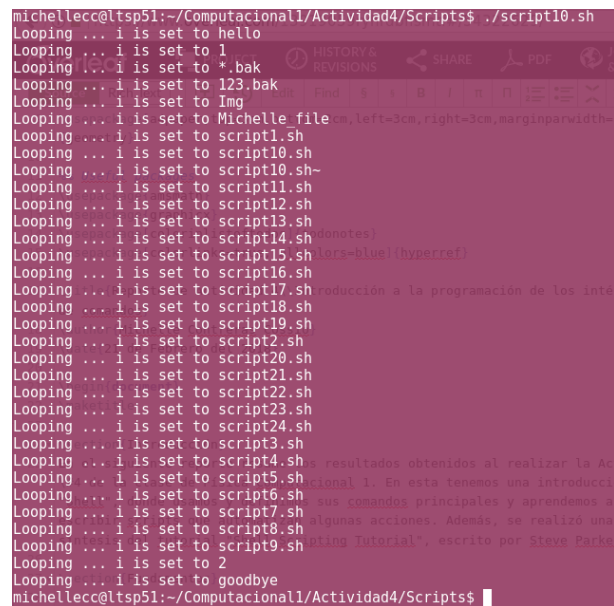
Los ciclos o "loops" son comunes en los lenguajes de programación, para repetir cierta tarea una cierta cantidad de veces. Los loops que maneja el Bourne shell son for y while.

- **For loops:** Estos iteran en valores ya establecidos hasta que la lista se acaba, sin importar si son números o cualquier entrada que se le de.

Script #10:

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

Que da como salida:



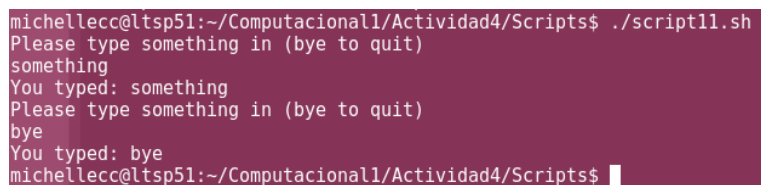
```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script10.sh
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to *.bak
Looping ... i is set to 123.bak
Looping ... i is set to Img
Looping ... i is set to Michelle.file
Looping ... i is set to script1.sh
Looping ... i is set to script10.sh
Looping ... i is set to script11.sh
Looping ... i is set to script12.sh
Looping ... i is set to script13.sh
Looping ... i is set to script14.sh
Looping ... i is set to script15.sh
Looping ... i is set to script16.sh
Looping ... i is set to script17.sh
Looping ... i is set to script18.sh
Looping ... i is set to script19.sh
Looping ... i is set to script2.sh
Looping ... i is set to script20.sh
Looping ... i is set to script21.sh
Looping ... i is set to script22.sh
Looping ... i is set to script23.sh
Looping ... i is set to script24.sh
Looping ... i is set to script3.sh
Looping ... i is set to script4.sh
Looping ... i is set to script5.sh
Looping ... i is set to script6.sh
Looping ... i is set to script7.sh
Looping ... i is set to script8.sh
Looping ... i is set to script9.sh
Looping ... i is set to 2
Looping ... i is set to goodbye
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

- **While Loops:** Estos tienen una condición de salida y hasta que esa se cumpla, el ciclo se acaba, aunque tambien se puede salir de ellos con el comando *ctrl+c*.

Script #11:

```
#!/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
```

Da como salida:



```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script11.sh
Please type something in (bye to quit)
something
You typed: something
Please type something in (bye to quit)
bye
You typed: bye
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

5.8 Test

Test se llama haciendo uso de "[", y es como un programa por sí solo. Este es parte de la estructura del if (elif y else se pueden omitir):

```
if [ something ]; then
    echo "Something"
elif [ something_else ]; then
    echo "Something else"
else
    echo "None of the above"
fi
```

El constructo if realiza una acción dependiendo si lo que se encuentra dentro de los corchetes es cierto.

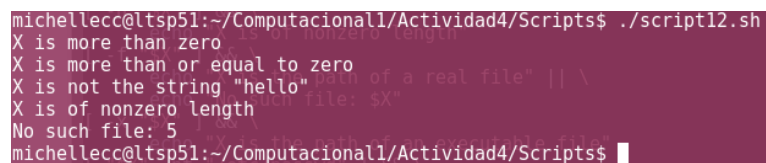
Es necesario agregar que el punto y coma se utiliza para unir dos líneas en un solo renglón. El backslash \ se usa para indicar que no es el final de la línea y que el siguiente renglón se debe usar como continuación de la línea.

En lugar de if se puede usar && o ||, que producirán un segmento de código si el comando es verdadero o falso, respectivamente.

Script #12: Este script ejemplifica lo anteriormente mencionado

```
#!/bin/sh
X=5
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"
```

Con salida:



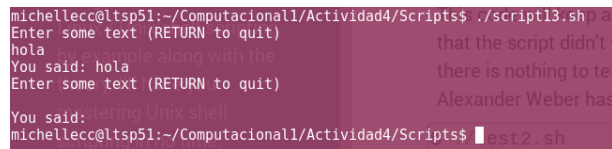
```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script12.sh
X is more than zero
X is more than or equal to zero
X is not the string "hello"
X is of nonzero length
No such file: 5
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Tambien el test se puede usar para ciclos while:

Script #13: Este ciclo se repetirá hasta que lo que está en el test se cumpla.

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

Con salida:



```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script13.sh
Enter some text (RETURN to quit)
hola
You said: hola
Enter some text (RETURN to quit)
by example along with the
You said: by example along with the
Enter some text (RETURN to quit)
Alexander Weber has
You said: Alexander Weber has
Enter some text (RETURN to quit)
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

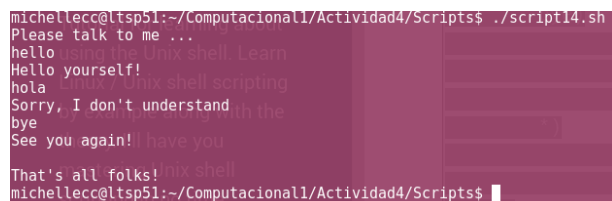
5.9 Case

Como en otros lenguajes de programación, el case es un conjunto de if/then/else y funciona exactamente igual; escoge una variable y la compara con opciones (o casos) de posibles valores de la variable y de no ser así, la opción default "*" contiene todos los casos no listados. Para iniciar un case se pone case y la variable a evaluar. Para terminarlo se pone esac (case al revés).

Script #14: El script muestra un case dentro de un while.

```
#!/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!"
```

Y esta es su salida:



```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script14.sh
Please talk to me ...
hello
You said: hello
hola
You said: hola
Sorry, I don't understand
You said: Sorry, I don't understand
bye
You said: bye
See you again!
You said: See you again!
That's all folks!
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

5.10 Variables (parte 2)

Existen variables que ya tienen valor, unas pocas son:

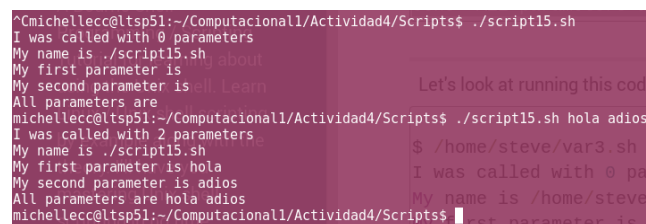
- **\$0**: nombre del programa donde se ubica.
- **\$1**: nombre del primer parámetro.
- **\$2**: nombre del segundo parámetro.
- **\$3...9**: nombre del (número) parámetro.
- **\$@**: nombre de todos los parámetros.
- **\$#**: número de parámetros que contiene.
- **\$?**: el status de salida del último comando.
- **\$\$**: PID (Process IDentifier), identificador de proceso del shell que esta corriendo.
- **\$_**: PID del último proceso en el fondo.

(Los últimos tres comandos no me quedaron muy claros)

Script #15:

```
#!/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 $@"
```

Y su salida es:



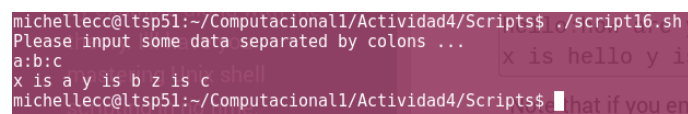
A terminal window showing the execution of script15.sh. The first run shows the script being called with 0 parameters, displaying its name and no arguments. The second run shows it being called with 'hola' and 'adios', displaying each argument separately. The third run shows it being called with 'hola', 'adios', and 'first parameter is', displaying all arguments together. A text box on the right says 'Let's look at running this code'.

Una última variable importante es IFS que por sus siglas en inglés es Internal Field Separator, se podría considerar como un separador de variables, el siguiente script lo ejemplifica mejor:

Script #16:

```
#!/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"
```

Su salida es:



A terminal window showing the execution of script16.sh. It prompts the user to input data separated by colons. The user enters 'a:b:c'. The script then outputs 'x is a y is b z is c'. A text box on the right says 'What if you en'.

5.11 Variables (parte 3)

Las llaves se usan para evitar confusiones en las variables, por ejemplo:

Script #17:

```
#!/bin/sh
foo=sun
echo $fooshine
echo ${foo}shine
```

Que tiene como salida lo siguiente, donde se observa la diferencia, la variable fooshine no se encuentra definida y no da valor, por otro lado, al agregarle las llaves, nos despliega la palabra sunshine. :

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script17.sh
sunshine
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Script #18: El siguiente código nos da un valor default en caso de que no se ingrese un nombre por el usuario:

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

Con el símbolo ":" estamos dándole un valor default a la variable, en caso de que el usuario no ingrese uno, su salida es:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script18.sh
-en What is your name [ michellecc ]
Michelle
Your name is : Michelle
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script18.sh
-en What is your name [ michellecc ]

Your name is : michellecc
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

5.12 Programas externos

Son comandos de Unix que se usan en el shell. Algunos ejemplos son echo, which, test, tr, grep, expr, cut.

Para hablar de comandos externos se utiliza la comilla invertida ('). Esta nos indica que el texto encerrado dentro se ejecutará como comando, un ejemplo:

Script #19:

```
#!/bin/sh
MYNAME='grep "^${USER}:" /etc/passwd | cut -d: -f5'
echo $MYNAME
```

Que tiene como salida:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script19.sh
Michelle Contreras Cossio
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Script #20:

```
#!/bin/sh
MYNAME='grep "^${USER}:" /etc/passwd | cut -d: -f5'
echo $MYNAME
```

Con salida:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script20.sh
Michelle Contreras Cossio
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

5.13 Funciones

Programar scripts con Bourne shell nos da la opción de crear funciones, ya sea con un simple script, declarando la función en el archivo donde se manda o con el comando de library.

En shell no existe diferencia entre procedures y funciones, una función en shell puede ser cualquiera de las dos o las dos. Las funciones dan un valor de cuatro maneras diferentes:

- Cambia el estado de una o muchas variables.
- Usa el comando exit para terminar el script.
- Usa el comando return para terminar la función y regresa el valor a la sección de donde fue llamada.
- echo output a stdout (estándar output).

Un ejemplo de función:

Script #21:

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

Su salida es:

```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script21.sh
Start of script...
Adding user bob ...
useradd -c Bob Holness the presenter bob
passwd bob letmein
Added user bob (Bob Holness the presenter) with pass letmein
Adding user fred ...
useradd -c Fred Durst the singer fred
passwd fred badpassword
Added user fred (Fred Durst the singer) with pass badpassword
Adding user bilko ...
useradd -c Sgt. Bilko the role model bilko
passwd bilko worsepassword
Added user bilko (Sgt. Bilko the role model) with pass worsepassword
End of script...
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Para declarar la variable se utiliza () al final del renglón. Seguido por llaves, donde todo lo que estas encierran es el código de la función. Este código no correrá hasta que se llame la función, en el ejemplo, esto ocurre hasta que el script inicia.

Existen variables globales, que pueden usarse tanto en el script principal como en la función y esta puede cambiar su valor, al menos que se le restrinja hacerlo, usando un subshell. Sin embargo, no puede cambiar los parámetros con los que fue llamada, al menos que los asigne a una variable, esa si la puede manipular.

Otra propiedad de las variables es que pueden ser recursivas, es decir, que se llame a si misma. Un ejemplo es con la función factorial:

Script #22:

```
#!/bin/sh

factorial()
{
    if [ "$1" -gt "1" ]; then
        i='expr $1 - 1'
        j='factorial $i'
        k='expr $1 \* $j'
        echo $k
    else
        echo 1
    fi
}

while :
do
    echo "Enter a number:"
    read x
    factorial $x
done
```

Y esta nos regresa el factorial de un número ingresado:

Como fue mencionado con anterioridad, también se pueden crear funciones con librerías. estas lo que hacen es separar la función en otro archivo con extensión .lib y al llamarla desde un script, se le llama como un archivo ejecutable, básicamente lo que hace es separar en múltiples archivos, para poder usarla en más de un script.

5.14 Tips y consejos

En esta sección se ven algunos consejos rápidos y básicos para mejorar la redacción de scripts.

Existen algunos comandos importantes, como grep (definido en la sección 3), cut que remueve líneas o pedazos de algun archivo; otro es tr que cambia o sustituye un set de caracteres en otro.

Otro tip es hacer trampa (cheating), esto es para poder remediar cosas que el shell no hace bien. Existen dos formas de cheating, con sed o con awk.

La función awk es que encuentra y reemplaza texto. Por su parte, sed elimina o sustituye cadenas o líneas, similar a grep, pero grep borra las líneas completas.

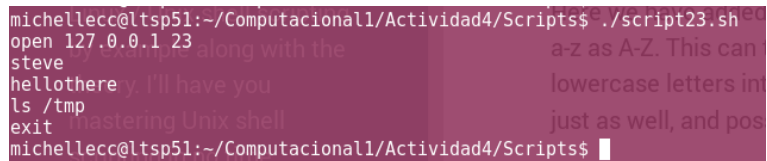
Telnet es otro truco que se usa, aquí dos ejemplos:

Script # 23:

```
#!/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
```

Con salida:



```
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$ ./script23.sh
open 127.0.0.1 23
steve
hellothere y I'll have you
ls /tmp
exit
michellecc@ltsp51:~/Computacional1/Actividad4/Scripts$
```

Script #24:

```
#!/bin/sh
# telnet2.sh | telnet > FILE1
host=127.0.0.1
port=23
login=steve
passwd=hellothere
cmd="ls /tmp"
timeout=3
file=file1
prompt="$"

echo open ${host} ${port}
sleep 1
tout=${timeout}
while [ "${tout}" -ge 0 ]
do
    if tail -1 "${file}" 2>/dev/null | egrep -e "login:" > /dev/null
    then
        echo "${login}"
        sleep 1
        tout=$((tout-1))
        continue
    else
        sleep 1
        tout=$((tout-1))
    fi
done

if [ "${tout}" -ne "-5" ]; then
```



```

        exit 1
    fi

    tout=${timeout}
    while [ "${tout}" -ge 0 ]
    do
        if tail -1 "${file}" 2>/dev/null | egrep -e "Password:" > /dev/null
        then
            echo "${passwd}"
            sleep 1
            tout=-5
            continue
        else
            if tail -1 "${file}" 2>/dev/null | egrep -e "${prompt}" > /dev/null
            then
                tout=-5
            else
                sleep 1
                tout='expr ${tout} - 1'
            fi
        fi
    done

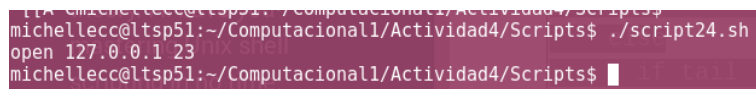
    if [ "${tout}" -ne "-5" ]; then
        exit 1
    fi

    > ${file}

    echo ${cmd}
    sleep 1
    echo exit

```

Con salida:



```

michellecc@ltsp51:~/Computacionall/Actividad4/Scripts$ ./script24.sh
open 127.0.0.1 23
michellecc@ltsp51:~/Computacionall/Actividad4/Scripts$

```

5.15 Guía rápida de comandos

Esta guía resume algunos comandos, no tan triviales, y que considero que serán de nuestro uso en la redacción de scripts:

Comando	Descripción
&	Corre el comando anterior en el fondo.
&&	AND Lógico
	OR Lógico
^	Inicio de la línea
\$	Final de la línea
=	Igualdad de cadenas
!	NOT lógico
\$0	Nombre del comando actual
\$1	Nombre del argumento del comando actual
-eq	Igualdad numérica
-ne	Desigualdad numérica
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que que

-ge	Mayor o igual que
-nt	Más nuevo que
-d	Directorio
-f	Archivo
-r	Archivo legible
-w	Archivo editable
-x	Archivo ejecutable

5.16 Shells interactivos

- **bash:**

Cuenta con herramientas muy útiles, por ejemplo, puedes ver el historial con las flechas de arriba y abajo. Existen también otros comandos que sirven para repetir comandos previamente corridos, todo esto hace de bash un shell bastante útil.

- **ksh:**

Ksh también permite trabajar con el historial, sin embargo, se tiene que hacer uso de emacs o vi. Si presionas ESC + k, navegas por el historial.

Si te encuentras en otro shell, puedes iniciar ksh y también salir de este para regresar al anterior.

6 Conclusión

Como conclusión me gustaría añadir que me pareció bastante digerible la programación con shell scripts, de hecho, más sencillo que los otros lenguajes que conozco. Probablemente esto se deba a que todos los lenguajes de programación son parecidos y guardan una sintaxis similar, unos más estrictos que otros, que no es este el caso. Pero una vez que ya llevaste un curso de cualquier otro lenguaje de programación, irse a otro es prácticamente lo mismo, solo cambian los nombres de los comandos y funciones.

Trabajar con el shell me gustó porque automatizó algo que hubiera tenido que hacer a mano y vi otra cara de la programación.

7 Bibliografía

- Script (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: <https://es.wikipedia.org/wiki/Script>
- Interprete de comandos: Shell (2000). Consultado: 21 de Febrero del 2018, de ibiblio. Sitio web: https://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/CURSOLINUX/curso_linux/node66.html
- Echo (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: [https://es.wikipedia.org/wiki/Echo_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Echo_(inform%C3%A1tica))
- Cat (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: [https://en.wikipedia.org/wiki/Cat_\(Unix\)](https://en.wikipedia.org/wiki/Cat_(Unix))
- Chmod (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: <https://en.wikipedia.org/wiki/Chmod>
- Linux grep command (20187). Consultado: 21 de Febrero del 2018, de HowtoForge. Sitio web: <https://www.howtoforge.com/tutorial/linux-grep-command/>
- Less (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: [https://en.wikipedia.org/wiki/Less_\(Unix\)](https://en.wikipedia.org/wiki/Less_(Unix))
- ls command in Linux/Unix. Consultado: 21 de Febrero del 2018, de Rapid Tables. Sitio web: <https://www.rapidtables.com/code/linux/ls.html>

- wc (2018). Consultado: 21 de Febrero del 2018, de Wikipedia. Sitio web: [https://es.wikipedia.org/wiki/Wc_\(Unix\)](https://es.wikipedia.org/wiki/Wc_(Unix))
- Unix / Linux - Pipes and Filters. Consultado: 21 de Febrero del 2018, de Tutorials Point. Sitio web: <https://www.tutorialspoint.com/unix/unix-pipes-filters.htm>
- Working With the Unix Shell (2010). Consultado: 21 de Febrero del 2018, de Univesity of Washington. Sitio web: <https://www.washington.edu/computing/unix/startdoc/shell.html>
- Shell Scripting Tutorial - The Shell Scripting Tutorial(2018). Consultado: 25 de Febrero del 2018, de Shell Scripting Tutorial. Sitio web: <https://www.shellscript.sh/index.html>

8 Apéndice

1. ¿Qué fue lo que más te llamó la atención en esta actividad?
Los scripts, que permiten automatizar, por ejemplo, descargar muchos archivos desde una base de datos en línea, en lugar de hacerlo manualmente, uno por uno.
2. ¿Qué consideras que aprendiste?
Aprendí o conocí otra cara de la programación, donde podemos trabajar directamente con la máquina para que haga lo que queramos, de una manera más cómoda.
3. ¿Cuáles fueron las cosas que más se te dificultaron?
Entender el nuevo lenguaje y sintaxis al trabajar con bash.
4. ¿Cómo se podría mejorar en esta actividad?
Dar más tiempo y ver un poco más de comandos.
5. ¿En general, cómo te sentiste al realizar en esta actividad?
Bien, aunque el tiempo no me alcanzó muy bien, pero estuvo digerible todo el material.