

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=":"

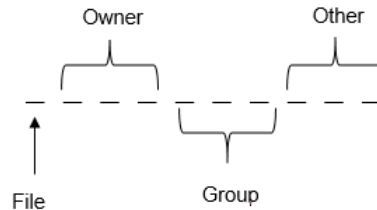
# Por ejemplo nos interesan bajar los datos de: 2013-2017
LISTYs="2017"

# 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 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 7
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 7
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=$i&FROM=0100&TO=2802&STNM=$STATION"
# Reposo 5 segundos
/bin/sleep 7
done
done
```

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 declara 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.

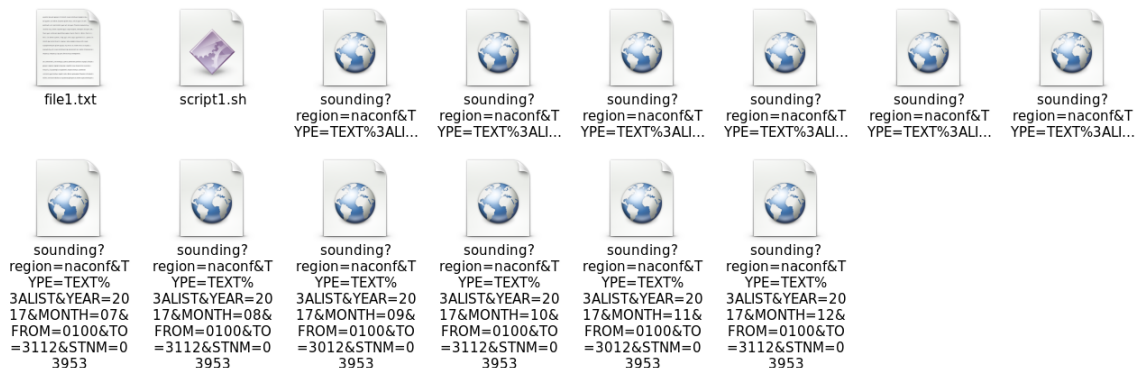
Sin embargo, el script aun no es un ejecutable, utilizando el comando *ls -alg* nos muestra los permisos del archivo, en donde podemos observar que solo es para lectura y edición. Los permisos del archivo se clasifican de la siguiente manera:



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 hPa	HGHT m	TEMP C	DWPT C	RELH %	MIXR g/kg	DRCT deg	SKNT knot	THTA K	THTE K	THTV 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 específico 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 específicas.

```
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&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=02&FROM=0100&TO=2812&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=03&FROM=0100&TO=3112&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=04&FROM=0100&TO=3012&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=05&FROM=0100&TO=3112&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=06&FROM=0100&TO=3012&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=07&FROM=0100&TO=3112&STNM=03953: HTML document, ASCII text
sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=08&FROM=0100&TO=3112&STNM=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 Showalter, 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 debería 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\&T0\=2812\&STNM\=03953
5632 sounding?region=naconf&TYPE=TEXT%3ALIST&YEAR=2017&MONTH=02&FROM=0100&T0=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
# Este es un comentario!
echo Hola Mundo          # Los comentarios en casi cualquier lugar!
```

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.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo1_1.sh
Hola Mundo
```

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
# Este script muestra las diferencias al usar las comillas en el comando echo
echo "Hola Mundo"
echo Hola mundo
```

Aquí se puede notar la diferencia:

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo1_2.sh
Hola      Mundo
Hola mundo
```

### 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 ¿Cuál es tu nombre?
read NOMBRE
echo "Hola $NOMBRE - espero estes bien!"
```


```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo2_1.sh
¿Cuál es tu nombre?
José Navarro
Hola José Navarro - espero estes bien!
```



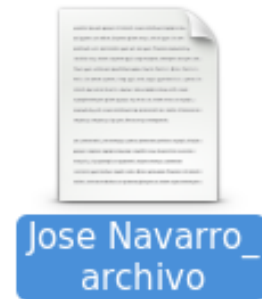
Si se declaran mal las variables el script no marcará error, si no que se imprimirá en pantalla la cadena de carácter vacío, es decir, nada. Si se quiere utilizar variables declaradas DESPUÉS de su impresión, es necesario exportarlas, ya que cada vez que corremos nuestro script, el shell se reinicia 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 "¿Cuál es tu nombre?"
read USUARIO
echo "Que onda $USUARIO"
echo "Crearé un archivo con el nombre ${USUARIO}_archivo"
touch "${USUARIO}_archivo"
```

A terminal window showing the execution of a shell script. The prompt is 'jose.navarro@lts26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos ./Ejemplo2\_2.sh'. The script asks '¿Cuál es tu nombre?', the user enters 'Jose Navarro', and the script prints 'Que onda Jose Navarro' and 'Crearé un archivo con el nombre Jose Navarro archivo'.

(a) Impresión en pantalla



(b) Archivo creado

### 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.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ cp a/* b/
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$
```

(a) Impresión en pantalla



1.txt

(b) Archivo de carpeta A a B

### 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.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ chmod 755 Ejemplo4_1.sh
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo4_1.sh
Hello    "World"
```

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.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo4_2.sh
Ej1_1.png Ej1_2.png Ej2_1.png Ej2_21.png Ej2_22.png Ej3_11.png Ej3_12.png Ej4_1.png Ejemplo1_1.sh Ejemplo1_2.sh Ejemplo2_1.sh Ejemplo2_2.sh Ejemplo4_1.sh
arro_archivo.txt a b
Jose Navarro_archivo.txt
*
*txt
```

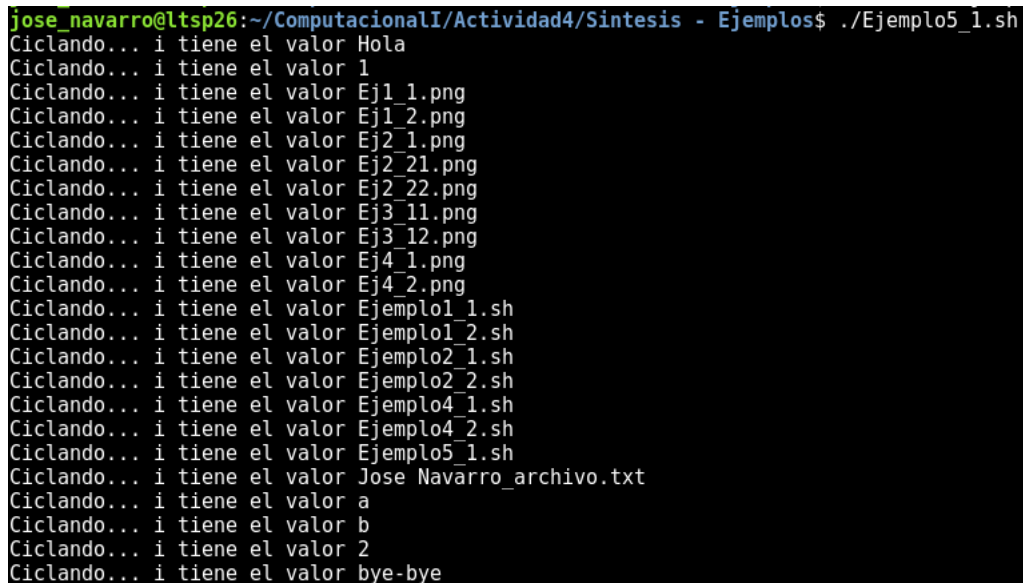
### 3.6 Ciclos

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 Hola 1 * 2 bye-bye
do
    echo "Ciclando... i tiene el valor $i"
done
```

A terminal window with a black background and green text. The prompt is 'jose\_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos\$'. The command executed is './Ejemplo5\_1.sh'. The output shows a series of lines: 'Ciclando... i tiene el valor Hola', 'Ciclando... i tiene el valor 1', 'Ciclando... i tiene el valor Ej1\_1.png', 'Ciclando... i tiene el valor Ej1\_2.png', 'Ciclando... i tiene el valor Ej2\_1.png', 'Ciclando... i tiene el valor Ej2\_21.png', 'Ciclando... i tiene el valor Ej2\_22.png', 'Ciclando... i tiene el valor Ej3\_11.png', 'Ciclando... i tiene el valor Ej3\_12.png', 'Ciclando... i tiene el valor Ej4\_1.png', 'Ciclando... i tiene el valor Ej4\_2.png', 'Ciclando... i tiene el valor Ejemplo1\_1.sh', 'Ciclando... i tiene el valor Ejemplo1\_2.sh', 'Ciclando... i tiene el valor Ejemplo2\_1.sh', 'Ciclando... i tiene el valor Ejemplo2\_2.sh', 'Ciclando... i tiene el valor Ejemplo4\_1.sh', 'Ciclando... i tiene el valor Ejemplo4\_2.sh', 'Ciclando... i tiene el valor Ejemplo5\_1.sh', 'Ciclando... i tiene el valor Jose Navarro\_archivo.txt', 'Ciclando... i tiene el valor a', 'Ciclando... i tiene el valor b', 'Ciclando... i tiene el valor 2', and finally 'Ciclando... i tiene el valor bye-bye'.

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo5_1.sh
Ciclando... i tiene el valor Hola
Ciclando... i tiene el valor 1
Ciclando... i tiene el valor Ej1_1.png
Ciclando... i tiene el valor Ej1_2.png
Ciclando... i tiene el valor Ej2_1.png
Ciclando... i tiene el valor Ej2_21.png
Ciclando... i tiene el valor Ej2_22.png
Ciclando... i tiene el valor Ej3_11.png
Ciclando... i tiene el valor Ej3_12.png
Ciclando... i tiene el valor Ej4_1.png
Ciclando... i tiene el valor Ej4_2.png
Ciclando... i tiene el valor Ejemplo1_1.sh
Ciclando... i tiene el valor Ejemplo1_2.sh
Ciclando... i tiene el valor Ejemplo2_1.sh
Ciclando... i tiene el valor Ejemplo2_2.sh
Ciclando... i tiene el valor Ejemplo4_1.sh
Ciclando... i tiene el valor Ejemplo4_2.sh
Ciclando... i tiene el valor Ejemplo5_1.sh
Ciclando... i tiene el valor Jose Navarro_archivo.txt
Ciclando... i tiene el valor a
Ciclando... i tiene el valor b
Ciclando... i tiene el valor 2
Ciclando... i tiene el valor bye-bye
```

### 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
CARACTER=hola
while [ "$CARACTER" != "bye" ]
do
    echo "Escribe algo (escribe bye para salir)"
    read LETRAS
    echo "You typed: $LETRAS"
done
```

```
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ ./Ejemplo5_2.sh
Escribe algo (escribe bye para salir)
Hola
Tu escribiste: Hola
Escribe algo (escribe bye para salir)
Que onda
Tu escribiste: Que onda
Escribe algo (escribe bye para salir)
bye
Tu escribiste: bye
jose_navarro@ltsp26:~/ComputacionalI/Actividad4/Sintesis - Ejemplos$ █
```

## 4 Bibliografía

- Ls command in Linux. (2018) Recuperado de: [www.rapidtables.com/code/linux/ls.html](http://www.rapidtables.com/code/linux/ls.html)
- Wc. (2014) Recuperado de: [www.tutorialspoint.com/unix\\_commands/wc.htm](http://www.tutorialspoint.com/unix_commands/wc.htm)
- Shell Scripting Tutorial. (2018). Recuperado de: [www.shellscript.sh/index.html](http://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 estos 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, además de la ayuda del profesor y compañeros salió todo bien.