Actividad 4

Jesús Adrián Zatarain Alvarado

March 8, 2018

1 Introducción

En esta actividad se plantea el cómo aprender ciertos intérpretes de comandos, se verán algunos de los más útiles e importantes; es una pequeña introducción para lo que se verá en las siguientes actividades.

2 Fundamentos

Unix: Es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de ATT, entre los que figuran Dennis Ritchie, Ken Thompson y Douglas McIlroy.

UNIX es un sistema operativo, es decir, es una colección de programas que ejecutan otros programas en una computadora. UNIX nació en los Laboratorios Bell de ATT en 1969, desarrollado por Ken Thompson y Dennis Ritchie (también creador del lenguaje de programación C). UNIX ofrecía solo una serie de pequeños programas, tan poco como fuese posible con respecto a sus predecesores, y tal vez esperando que la gente querría unir todos esos programas.

El sistema provee un serie de herramientas, cada una realiza una función limitada y bien definida, utiliza un sistema de archivos unificado como medio de comunicación, y un lenguaje de comandos llamado "shell" que permite combinar esas herramientas para realizar acciones complejas.

Intérprete de comandos SHELL: El Shell es un programa que se encuentra en el directorio /bin. El resto de este tutorial trata únicamente del shell Shell bash que es el Shell por defecto en las distribuciones GNU/Linux.

El Shell permite ejecutar comandos, explorar el árbol de directorios del sistema, crear, editar y eliminar archivos, etc.

Una gran cantidad de soluciones están dadas en línea de comandos, no es que GNU/Linux no tenga una interfaz gráfica, pero en ciertas tareas, el uso de la línea de comando es muchos más práctico y potente que el famoso ratón.

Un comando es un archivo ejecutable. La ejecución de un comando puede ser diferente dependiendo del caso.

Los comandos que puede ejecutar desde su terminal se encuentran en ciertos directorios del sistema. La variable PATH (en español: "ruta") contiene una lista de directorios, los cuales contienen los comandos a los que puede acceder.

Para poder acceder a todos los comandos es necesario ser root. Para encontrar la ubicación de un comando, utilizamos "whereis" (en español “dónde está”):

3 Resumen de la actividad

Primero se inicia descargando un archivo script1.sh que contiene las instrucciones que se van a ejecutar para completar la actividad. Lo que contiene es el código necesario para descargar los datos metereológicos correspondientes de una estación, de 12 meses de un año en el sitio web Wyoming. El archivo contiene algunos datos innecesarios y a corregir. Como la presencia de un /local que se repite reiterada veces. Para ello, se utiliza el editor emacs, donde donde se usan comandos para poderlos eliminar por medio del teclado. También se modifica la cantidad de datos por años que se van a descargar, al igual que el tiempo que descargará entre uno y otro; para no saturar los servidores.

A continuación, se usa el comando "ls -alg", para observar los permisos del archivo. El archivo tiene la particularidad de que no es ejecutable,sólo tiene permisos de lectura, tiene los permisos 644; que es una notación de tipo base 8, que sirve para atribuirle ciertos aspectos a los permisos del archivo. Se usó el comando 755 para concederle los permisos de lectura, escritura y ejecución. Luego, se verifica si los permisos están de acuerdo a lo que se espera usando nuevamente el comando "ls -alg".

Cumplido todos esos requerimentos, se pasa a ejecutar el script. En donde se observará que hay 12 archivos correspondientes a los datos de meses descargados, más los dos archivos que ya había, en la carpeta donde está localizado el script.

A continuación se utilizó el comando "less" que sirve para ver de manera ordenada en la terminal, los datos que contiene cada página de un archivo de texto. El comando "cat" funciona de una manera similar, éste despliega todo el archivo de texto. El comando grep funciona como un filtro; sólo selecciona los archivos que coincidan con los carácteres condicionados por el usuario.

Después, se verifica que todos los archivos descargados son del mismo tipo (ASCll), se usa el comando file, que desplegará los 12 archivos correspondientes. Se prosigue recolectando todos los archivos en uno sólo con extensión de texto, usando el comando "cat". EL archivo contendrá todos los datos descargados por el script. Luego, se filtran los datos por medio de la instrucción "grep" dejando sólo a los que tengan relación directa con la estación seleccionada.

A continuación, se usa un comando del tipo "grep" para crear un archivo que contiene todos los datos ordenados en una tabla, con cada variable con su respectiva cantidad.

Para finalizar, se pide realizar un script que automatice las últimas instrucciones realizadas. Por medio del emacs, se pega el último código escrito en la terminal y se le modifican los permisos para poder ejecutarlos. Se comparan los dos archivos generados manualmente y por medio del script, y se llega a la conclusión que los dos son idénticos; para ello se utilizó el comando "diff" para ver las similitudes y diferencias que hay entre los dos archivos creados.

4 Comandos de Linux utilizados

Cat: Este comando sirve para desplegar todas los datos contenidos en un archivo de texto por medio de la terminal, sin entrar a un editor de texto. Lo despliega en forma de lista, página por página, haciendo que la única manera de interactuar con los archivos, sea visualizándolos, sin poder editarlos. También se pueden desplegar la cantidad de archivos necesarios escribiendo el nombre de los archivos con su correspondiente extensión. Si se coloca el prefijo "-n", enumera todas las líneas del archivo selecionado. El prefijo "-b", enumera las líneas sin espacios en blanco.

Chmod: Cada script tiene la particularidad de que tienen permisos. Su ejecución, modificación o lectura está restringida al usuario. Por defecto, los script tienen los permisos 644, que son los que permiten la visualización y escritura en el archivo, más no la ejecución del mismo. Por tanto, se tienen que cambiar los permisos a 755, para que se puedan hacer las tres acciones.

La escritura del comando es como sigue:

```
chmod [modificadores] permisos fichero/directorio
```

modificadores es opcional, y puede tomar los valores:

- -f: no visualiza los posibles mensajes de error que puedan ocurrir debido a conflictos en la asignación de permisos.
- -v: lista los ficheros y directorios a los que se les va aplicando el comando a medida que el mismo se ejecuta
- -h:
- -R: aplica el comando chmod recursivamente a todos los ficheros y de los subdirectorios.
- -H:
- -L:
- -P:
- -C: igual a -v con excepción que solo lista los ficheros modificados.
- -E:

Esto significa que f, h y v pueden ser usados todos a la vez, e independientemente de los valores de los demás modificadores; H, L y P son ignorados salvo que se especifique explícitamente la opción R

permisos corresponde a uno de los modos que se describen a continuación, y enumera los tipos de permisos que se brindan a las clases de usuarios.

fichero/directorio fichero o directorio al cual se otorga el permiso.

Donde los permisos deben estar en una notación en base ocho

Existen tres permisos independientes, llamados permisos básicos, que pueden ser permitidos (estado 1) o denegados (estado 0) a un fichero y/o directorio

- r - lectura: Ver el nombre de los ficheros dentro del directorio (pero sin poder saber nada más sobre ellos como: el tipo de archivo, tamaño, propietario, permisos, etc.)
- w - escritura: Agregar, eliminar y renombrar ficheros del directorio
- x - ejecución: Recorrer su árbol para acceder ficheros y subdirectorios, pero no ver los ficheros dentro del directorio (excepto que se le de el permiso de lectura)

Los permisos de sistemas UNIX se dividen en cuatro clases, conocidas como usuario, grupo, otros y todos.

Por lo tanto, las clases de usuarios a los cuales se les puede asignar los permisos básicos anteriormente mencionados son:

- u – dueño: dueño del fichero o directorio
- g – grupo: grupo al que pertenece el fichero
- o – otros: todos los demás usuarios que no son el dueño ni del grupo
- a – todos: incluye al dueño, al grupo y a otros

Los permisos efectivos aplicados a un determinado usuario en relación a un fichero se determinan en un orden lógico de precedencia. Por ejemplo, el usuario propietario del fichero tendrá los permisos efectivos dados a la clase de usuario, sin importar los asignados a la clase de grupo o a la clase de otros.

Para asignar permisos se tiene vomo resultado de la combinación de los tres tipos de permisos (lectura, escritura y ejecución), con las tres clases de usuarios (dueño, grupo y otros), se obtiene

$2^3 = 8$

permisos en total que pueden ser asignados o denegados de forma independiente.

La base 8 se utiliza habitualmente para que exista un dígito por cada combinación de permisos (un bit a modo de bandera por cada permiso, con valor 1 o 0 según el permiso esté concedido o denegado).

Así, las posibles combinaciones se resumen en números octales de tres dígitos del 000 al 777, cada uno de los cuales permite establecer un tipo de permiso distinto a cada clase de usuario:

El primer dígito establece el tipo de permiso deseado al dueño; el segundo al grupo; y el tercero al resto de los usuarios.

Echo: Es un comando para la impresión de un texto en pantalla. Es utilizado en

las terminales de los sistemas operativos como Unix, GNU/Linux, o MS-DOS; dentro de pequeños programas llamados scripts; y en ciertos lenguajes de programación tales como PHP.

Grep: Es una utilidad de la línea de comandos escrita originalmente para ser usada con el sistema operativo Unix.

Usualmente, grep toma una expresión regular de la línea de comandos, lee la entrada estándar o una lista de archivos, e imprime las líneas que contengan coincidencias para la expresión regular.

Su nombre deriva de un comando en el editor de texto ed que tiene la siguiente forma: g/re/p y significa «hacer una búsqueda global para las líneas que encajen con la expresión regular (regular expression en inglés), e imprimirlas». Hay varios argumentos que se pueden usar con grep para modificar el comportamiento por defecto.

Para mostrar todas las líneas que contienen la cadena «tal» en una lista de archivos (donde «*» representa todos los archivos en el directorio actual):

```
grep tal *
```

Para mostrar todas las líneas que no contengan la cadena «tal», se usa «-v»:

```
grep -v tal *
```

Para mostrar sólo el nombre de tales archivos, se usa «-l»:

```
$ grep -l tal *
```

Para mostrar sólo el nombre de los archivos que no contienen la cadena, se usa «-L»:

```
$ grep -L tal *
```

Para buscar recursivamente, no sólo en los archivos del directorio actual sino también en los de sus subdirectorios (donde "." representa el directorio actual), se usa «-r»:

```
$ grep -r tal *
```

Existen muchos derivados de grep; por ejemplo agrep, que significa approximate grep (grep aproximado), que sirve para hacer una búsqueda aproximada de cadenas; fgrep para buscar patrones fijos; egrep para búsquedas que involucren expresiones regulares más complejas; y tcgrep, que utiliza la sintaxis de expresiones regulares de Perl. Todas estas variantes de grep han sido llevadas a diversos sistemas operativos.

Muchos otros comandos contienen la cadena «grep». Por ejemplo pgrep, que muestra los procesos cuyos nombres encajan con cierta expresión regular.

En Perl existe la función grep, que recibe una expresión regular y una lista, y devuelve los elementos de la lista que encajan con dicha expresión.

Less: lEs un visualizador de archivos de texto que funciona en intérpretes de comando.

A diferencia de otros programas similares (como more), less permite una completa navegación por el contenido del archivo, utilizando un mínino de recursos del sistema.

Ls: Es un comando del sistema operativo Unix y derivados que muestra un listado con los archivos y directorios de un determinado directorio. Los resultados se muestran ordenados alfabéticamente.

Los archivos y directorios cuyo nombre comienza con . (punto) no se muestran con la instrucción ls, por lo que se suelen denominar «archivos ocultos». La opción -a de ls inhibe este comportamiento, y muestra todos los archivos y subdirectorios, incluso los que comienzan con punto.

ls es una de las herramientas más básicas de los sistemas operativos Unix, por lo que forma parte del paquete GNU Coreutils.

- -l muestra un listado en el formato largo, con información de permisos, número de enlaces asociados al archivo, usuario, grupo, tamaño y fecha de última modificación además del nombre.
- -h con -l imprime el tamaño de los archivos de forma entendible para los humanos (ej. 1K 234M 2G).
- -d muestra solamente el nombre del subdirectorio, sin entrar en él ni dar un listado del contenido.
- -t muestra ordenado por la fecha de última modificación.
- -c muestra ordenado por la fecha de última modificación del estado del archivo.
- -r cuando el listado se hace por orden temporal, los archivos más recientes van al principio. Si se indica la -r se invierte el orden, mostrando los más recientes al final.
- -L en los enlaces simbólicos, muestra los datos del archivo referenciado en vez de los del link.

- -1 muestra el listado en una sola columna. Sin la opción -1 el listado se muestra en varias columnas, tantas como permita el ancho de la terminal (generalmente controlado con la variable de entorno.
- -i muestra el número del i-nodo antes del nombre de archivo.
- -m muestra los archivos en una línea y separados por comas.
- -R hace un listado recursivo. Lista primero los archivos del directorio en curso, luego los de los subdirectorios de éste, luego los de los subdirectorios contenidos en ellos (nietos) y así sucesivamente.
- -s muestra delante del nombre del fichero el tamaño en kilobytes del mismo.
- –color muestra cada tipo de archivo de un color distinto: un color para los directorios, otro para los archivos regulares, otro para los enlaces simbólicos, otro para los sockets, otro para las tuberías FIFO, etc. Este parámetro no se acepta en todas las versiones de ls y, por supuesto, requiere que la terminal sea capaz de mostrar distintos colores o intensidades.
- - a muestra los archivos ocultos.
  Al igual que en la mayoría de las órdenes unix, las opciones se pueden agrupar, de manera que es lo mismo poner ls -li que ls -l -i . O también es lo mismo ls -ltra que ls -l -t -r -a

  Wc: wc (word count) es un comando utilizado en el sistema operativo Unix que permite realizar diferentes conteos desde la entrada estándar, ya sea de palabras, caracteres o saltos de líneas.

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

  Modo de uso:

  ```
  wc -l <archivo> número de lineas
     wc -c <archivo> número de bytes
     wc -m <archivo> imprime el número de caracteres
     wc -L <archivo> imprime la longitud de la línea más larga
     wc -w <archivo> imprime el número de palabras
  ```

  Redirectores: |, > : Un comando normalmente lee su entrada desde la entrada estándar, que es su terminal por defecto. De forma similar, un comando normalmente escribe su salida en salida estándar, que de nuevo es su terminal por defecto.

5 Síntesis-Steve Parker

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

Modo de operar

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

Primeros pasos

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

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

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

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

Variables I

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

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

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

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

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

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

Wildcards (Comodines)

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

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

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

Carácteres de Escape

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

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

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

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

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

Ciclos

La mayoría de los lenguajes tienen el concepto de bucles: si queremos repetir una tarea veinte veces, no queremos tener que escribir el código veinte veces, con un ligero cambio cada vez. Exiasten dos tipos de ciclos, los "for" y los "while".

Ciclos For

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

```
#!/bin/sh
for i in hello 1 * 2 goodbye
```

```
do
echo "Looping ... i is set to $i"
done
```

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

Ciclos While

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

```
#!/bin/sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
echo "Please type something in (bye to quit)"
read INPUT_STRING
echo "You typed: $INPUT_STRING"
done
```

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

Test

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

```
#!/bin/sh
echo -en "Please guess the magic number: "
read X
echo $X | grep "[^0-9]" > /dev/null 2>&1
if [ "$?" -eq "0" ]; then
  # If the grep found something other than 0-9
  # then it's not an integer.
  echo "Sorry, wanted a number"
else
```

```
  # The grep found only 0-9, so it's an integer.
  # We can safely do a test on it.
  if [ "$X" -eq "7" ]; then
    echo "You entered the magic number!"
  fi
fi
```

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

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

Case

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

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

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

Variables II

Existen un conjunto de variables establecidas, y la mayoría de ellas no pueden tener valores asignados. Contienen información útil referente que el script puede utilizar para conocer el entorno en el que se está ejecutando. El primer conjunto de variables que veremos son $0 .. $9 y $#. La variable $0 es el nombre base del programa como se lo llamó. $1 .. $9 son los primeros 9 parámetros adicionales con los que se invocó el script. Como ejemplo:

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

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

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

Variables III

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

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

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

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

Programas Externos

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

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

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

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

Funciones

Las funciones son fáciles de usar dentro del script del Bourne Shell.Esto generalmente se hace de una de dos maneras; con un script simple, la función simplemente se declara en el mismo archivo como se llama. Sin embargo, al escribir un conjunto de secuencias de comandos, a menudo es más fácil escribir una "librería" de funciones útiles, y el origen de ese archivo al inicio de los otros scripts que utilizan las funciones. Una función puede devolver un valor en una de cuatro formas diferentes:
Una función puede devolver un valor en una de cuatro formas diferentes:

- Cambiar el estado de una variable o variables.

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

```
#!/bin/sh
# A simple script with a function...
add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
  echo "Adding user $USER ..."
  echo useradd -c "$COMMENTS" $USER
  echo passwd $USER $PASSWORD
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}
###
# Main body of script starts here
###
echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script..."
```

Este código no se ejecuta hasta que se llama a la función. Las funciones se leen, pero básicamente se ignoran hasta que realmente se llaman. En este caso, la función add_a_userse lee y se comprueba la sintaxis, pero no se ejecuta hasta que se llame explícitamente.
Los programadores acostumbrados a otros lenguajes pueden sorprenderse con las reglas de alcance para las funciones de shell. Básicamente, no hay una definición del alcance, aparte de los parámetros ($1, $2, $@, etc.). Tomando el siguiente segmento de código simple:

```
#!/bin/sh
myfunc()
{
  echo "I was called as : $@"
  x=2
}
### Main script starts here
echo "Script was called with $@"
```

```
x=1
echo "x is $x"
myfunc 1 2 3
echo "x is $x"
```

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

Consejos y Sugerencias

Se presentarán algunos tips de comandos a manera de ejemplificacion con la finalidad de facilitar la creación y uso de los scripts.
Telnet es una técnica útil, aunque telnet ya no se usa en los servidores, todavía lo utilizan algunos dispositivos de red, como los concentradores de terminales y similares. Al crear un script como este, su propio script o desde una línea de comando, puede ejecutar.

```
#!/bin/sh
host=127.0.0.1
port=23
login=steve
passwd=hellothere
cmd="ls /tmp"
echo open ${host} ${port}
sleep 1
echo ${login}
sleep 1
echo ${passwd}
sleep 1
echo ${cmd}
sleep 1
echo exit
```

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

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

Referencias

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

6 Conclusión

Fue una actividad interesante. Se concoció mucho sobre cómo operan los scripts, y cómo automatizar tareas tediosas. Se aprendieron muchos comandos útiles para usar en la terminal.

7 Bibliografía

www.shell.blogspot.com

8 Apéndice

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

La utlización de scripts que sirven para hacer procesos automatizados y el descubrimiento de los comandos Shell

2.- ¿Qué consideras que aprendiste? A cómo modificar scripts y usar mejor la terminal

3.- ¿Cuáles fueron las cosas que más se te dificultaron? Apreder todo sobre la actividad, fue muy densa

4.- ¿Cómo se podría mejorar en esta actividad? Está perfecta como está

5.- ¿En general, cómo te sentiste al realizar en esta actividad? Muy bien, fue fácil, pero densa. Funciona como buena introducción a los comandos venideros a aprender.