**LABORATORIO No 10: Shell y administración**

**Objetivos**

Al finalizar el taller, se espera que el estudiante esté en capacidad de:

* Utilizar el bash a través de línea de consola.
* Identificar elementos para realizar scripts de administración en el bash.
* Utilizar estructuras de control a través scripts de administración en el bash.
* Responder a requerimientos a través de la implementación de scripts de administración en el bash.

# Primer contacto

# Para averiguar cuál es el shell de trabajo que le ha asignado su administrador tan sólo tiene que ejecutar en el terminal el siguiente comando que muestra en pantalla el contenido de la variable $SHELL. Esta es la variable es la que el sistema utiliza de forma estándar para guardar el nombre del shell de trabajo.

# $ echo $SHELL

# Desvío de la salida y la entrada de los comandos

# El bash shell permite desviar la entrada y la salida de todos los comandos de forma que estos puedan leer o escribir sus datos en un archivo, en la pantalla, en una línea de comunicaciones o en cualquier otro dispositivo sin que sea preciso cambiar una sola línea del código fuente. Para desviar, por ejemplo, la salida del comando ls a un archivo llamado lst basta con teclear el siguiente comando:

# $ ls -al > lst

# Este comando crea primero el archivo lst y a continuación ejecuta ls, pero desviando toda la salida que produzca hacia este archivo. También es posible desviar la salida de un comando añadiéndola a un archivo ya existente. Por ejemplo, si quisiéramos añadir al archivo anterior la frase "estos eran mis archivos", bastaría con teclear el siguiente comando:

# $ echo "estos eran mis archivos" >> lst

# Si ahora mostramos en pantalla el contenido de lst obtendríamos el siguiente resultado:

# $ cat lst

# O editar el contenido del archivo, se digita

# $ nano lst

# Tuberías y comunicaciones: pipes

# Situaciones en las que la salida de un comando se utiliza como la entrada de otro son muy frecuentes en la práctica. Por ejemplo, supongamos que deseamos obtener un listado ordenado alfabéticamente con información acerca de todos los usuarios conectados actualmente en la máquina. Sabemos que el comando who permite obtener la información acerca de los usuarios conectados y que sort puede ordenarla alfabéticamente. Una posible solución es:

# $ who > tmp

# $ sort < tmp

# usuario1 :0 Sep 09 19:53

# usuario2 :1 Sep 09 15:45

# $ rm tmp

# En este caso hemos usado el archivo tmp para guardar de forma temporal la información proporcionada por who antes de usar el comando sort. Una forma simplificada de hacer esto es usar tuberías, tal y como se muestra a continuación:

# $ who | sort

# usuario1 :0 Sep 09 19:53

# usuario2 :1 Sep 09 15:45

# El efecto es similar a la creación de un archivo temporal, con la única diferencia de que el uso de las tuberías es mucho más eficiente.

# Variables

# Definición y consulta: En bash shell no es necesario declarar las variables. Para introducir una nueva tan sólo es necesario asignarle un valor utilizando la siguiente sintaxis (fíjese en que no existe ningún espacio en blanco al rededor del signo =):

# $ mi\_nombre=juan

# $ mi\_maquina=aramis.co

# Para la consulta de variables es necesario preceder el nombre de las variables del signo $. Por ejemplo:

# $ mi\_direccion=$mi\_nombre@$mi\_maquina

# $ mi\_cuenta=file://$mi\_maquina/$LOGNAME

# Variables especiales

# Figura1. Argumentos de entrada

# 

# Visualice el contenido de cada variable.

# Cuando set se ejecuta con parámetros, lo que hace es asignarlos a las variables $0, $1, $2, etcétera. Por ejemplo, a veces suele ser preciso determinar la hora actual a partir del comando date. El problema es que este comando muestra en su salida estándar una cadena que informa no sólo de la hora actual sino también del día de la semana, el mes, el año etcétera.

# $ date

# lun sep 12 13:08:54 CEST 2005

# Para obtener sólo la hora podríamos ejecutar el siguiente comando:

# $ set ‘date‘

# $ echo $4

# Nota: Se debe recordar que para asignar a los argumentos de entrada el resultado del comando date, se debe colocar entre el símbolo de la tilde en sentido contrario (`)

# Programación en el Shell

# Ahora se iniciará con el proceso de elaboración de programas que se pueden ejecutar en el Shell. Para desarrollar este proceso se hace necesario cumplir con 3 actividades.

# La *primera* actividad consta de dar respuesta a un requerimiento funcional elaborando un programa a través de un editor de texto.

# La *segunda* actividad consiste en asignar permisos de ejecución al programa creado.

# La *tercera* actividad permite ejecutar el programa creado y verificar el funcionamiento.

# A continuación se presenta el resumen de un requerimiento funcional al cuál se debe dar respuesta a través de un programa para el Shell.

|  |  |
| --- | --- |
| Nombre: | Editar el archivo |
| Resumen: | Se requiere abrir con el editor nano el archivo cuyo nombre se pasa como parámetro a través de la línea de argumentos. |
| Entradas: | Palabra |
| Situaciones anómalas: | Si no se pasó desde línea de argumentos ningún valor, se debe mostrar un mensaje informando del hecho.Si el archivo no existe, se debe mostrar un mensaje informando del hecho. |
| Resultado: | Se ha abierto el archivo para su edición. |

# Cumpliendo con las actividades planteadas anteriormente, inicialmente se debe crear mediante un editor de texto un archivo.

# $ nano edicion.sh

# Como buena práctica los programas que se construyen para ser ejecutados en el Shell deben llevar la extensión (.sh).

# Cuando ya se encuentre creado el archivo, aplicando otra buena práctica, se debe colocar el tipo de Shell que se debe utilizar para ejecutar el programa. Para este laboratorio se hará uso del bash.

# Código fuente

# #!/bin/bash

# if [ -n $1 ]

# then

# if [ -f $1 ]

# then

# if [ -w $1 ]

# then

# nano $1

# else

# echo “El archivo no tiene permisos para ser modificado”

# fi

# else

# echo “El archivo no existe”

# fi

# else

# echo “El nombre del archivo es requerido desde línea de parámetros”

# fi

Para comprender que hace el código, se hace necesario describir que hacen las opciones que se están utilizando en el código fuente:

1. Para evaluar si un archivo existe, se utiliza la expresión [ -f nombre del archivo ], es decir, retorna verdadero si el archivo existe.
2. Para evaluar si un parámetro no está vacío se utiliza la expresión [ -n nombre del parámetro ], es decir, retorna verdadero si el parámetro evaluado, no está vacío.
3. Para evaluar si un archivo tiene permisos de modificación [ -w nombre del archivo ], es decir, retorna verdadero si el parámetro evaluado, tiene permisos de modificación.

# A continuación como parte de la segunda actividad, se debe otorgar los permisos de ejecución al archivo *palabra.sh.* Para asignar los permisos se debe utilizar el comando chmod. Antes de presentar la forma como se realiza, se describirá brevemente algunos elementos teóricos del comando.

En primera instancia se hace necesario identificar que los archivos tienen las siguientes propiedades: ¿De quién es? ¿A qué grupo pertenece? Y ¿Qué permisos tiene?.

Entre los permisos que tiene un archivo se encuentran que: 1.) Pueden ser leídos (Read), 2.) Pueden ser ejecutados (Execute), y 3.) Pueden ser escritos o modificados (Write).

El comando chmod permite cambiar permisos de lectura, escritura y ejecución. Para cambiar los permisos de un archivo la sintaxis es:

chmod [quien] [permiso] [usuario]

Donde quien es el archivo al cual se cambiará los permisos, esto porque pueden ser:

* El propietario ( u )
* El grupo ( g )
* Los otros usuarios del sistema ( o )
* Todos los usuarios incluso el dueño ( a )

En permisos

Existen los símbolos + (Otorgar) – (Quitar) los cuales se combinan con

* ( r ) lectura
* ( x ) ejecución
* ( w ) escritura

Por ejemplo:

Si se quiere que el archivo “ documento” pueda ser solo leído por el usuario propietario, el comando será.

chmod u+r-w documento

Otra forma de asignar permisos es a través de la conversión numérica, de la siguiente forma:

* El permiso de ejecución, tiene el valor 1
* El permiso de escritura, tiene el valor 2
* El permiso de lectura, tiene el valor 4

Por lo tanto asignado los valores respectivos en la Tabla 1, se puede conocer los valores a ser asignados.

Tabla 1. Permisos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Read** | **Write** | **Execute** | ***Total*** |
| **User** |  |  |  |  |
| **Group** |  |  |  |  |
| **Others** |  |  |  |  |

Por ejemplo, si al archivo palabra.sh se desea asignar los permisos de ejecución, lectura y escritura para el propietario, esto significa que a User se le debe asignar los respectivos valores en la fila que le corresponde como se puede observar en la Tabla 2.

Tabla 2. Permisos del archivo edicion.sh

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Read** | **Write** | **Execute** | ***Total*** |
| **User** | 4 | 2 | 1 | 7 |
| **Group** | 0 | 0 | 0 | 0 |
| **Others** | 0 | 0 | 0 | 0 |

Por lo tanto, la asignación de los permisos al propietario, corresponden con la suma de valores registrados en la columna Total. La ejecución mediante el comando chmod, se presenta a continuación:

$ chmod 700 edicion.sh

Finalmente, para probar el funcionamiento del programa, simplemente se antepone ./ al programa a ejecutar, como se muestra a continuación:

$ ./edicion.sh prueba.txt

En este caso verificar que en la ruta donde se encuentra ubicado existe un archivo prueba.txt, si el archivo tiene permisos de modificación, lo abre con el editor nano.