

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

LABORATORIO: Sistemas Operativos

Práctica 10

Programación del Shell. Scripts y procesos.

LosDos

Integrantes:

Amanda Velasco Gallardo - 154415
Carlos Octavio Ordaz Bernal - 158525

Fecha(s) de elaboración de la práctica:

5 de abril de 2019

Introducción

El **shell** es un intérprete de comandos. Es el encargado de validar las instrucciones introducidas por el usuario y cuenta con variables de ambiente que permiten personalizar el entorno de trabajo. Una característica que distingue a los **shell** del resto de los intérpretes es que éste es un lenguaje de programación que cuenta con instrucciones para la ejecución selectiva y repetitiva de grupos de comandos, cada programa escrito en un **shell** se le conoce como guión (*script*).

La principal ventaja de utilizar un *script* de **shell** es la capacidad de ejecutar instrucciones secuenciales, en vez de tener que estar ingresando una a una dentro de la línea de texto del intérprete de comandos. Además, facilita las tareas que se deben realizar ya que se pueden programar un conjunto de instrucciones de una sola vez.

Las variables de **shell** son aquellas que permiten personalizar el ambiente de trabajo. Existe un conjunto de variables predefinidas y el usuario es capaz de definir sus propias variables. Existen dos tipos de variables de **shell**: las booleanas y las que pueden recibir un valor. Además, existe otro tipo de variables conocidas como variables de entorno o globales que son heredadas a cualquier **subshell** creado. Si una variable de entorno y una variable de **shell** tienen el mismo nombre, la de mayor precedencia es la segunda.

La principal ventaja de utilizar **shell** es la personalización, cada intérprete puede contener variables definidas por el usuario dependiendo de las tareas que se vayan a realizar. Es posible decir que los **shell** vienen en varios sabores por lo diferentes que pueden ser en términos de la forma en que trabajan y sus variables definidas.

Desarrollo

2. Elabore el script *ej1*, ejecútelo y muestre lo desplegado. ¿Por qué `ps -l` despliega a *ej1* como un proceso? ¿Cuál proceso es, en realidad, *ej1*? ¿Cuáles son su PID y su PPID? ¿Cuál es su proceso padre? ¿Cuál es el proceso padre de `ps`?

El resultado de ejecutar el script *ej1* se muestra a continuación en la figura 1.

```

ubuntu:~> ej1
ps -l
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  3012  3005  0  80   0 -  1438 rt_sig pts/0    00:00:00 tcsh
0 S  1000  3020  3012  0  80   0 -  1337 rt_sig pts/0    00:00:00 ej1
0 R  1000  3022  3020  0  80   0 -  1177 -      pts/0    00:00:00 ps
ls -al
total 476
drwxr-xr-x 38 sisops sisops 4096 Apr  5 09:59 .
drwxr-xr-x  3 root  root  4096 Jun  4 2012 ..
-rw-rw-r--  1 sisops sisops  15 Feb 22 09:22 12.txt
-r--r--r--  1 sisops sisops 3562 Feb 15 09:48 154415_158525_p04.txt
-rw-rw-r--  1 sisops sisops  150 Feb 15 09:19 9b.txt
-rw-rw-r--  1 sisops sisops  320 Jan 14 04:30 argumentos.cc
-rw-----  1 root  root   524 Feb 26 2013 .bash_history
-rw-r--r--  1 sisops sisops  220 Jun  4 2012 .bash_logout
-rw-r--r--  1 sisops sisops 3486 Jun  4 2012 .bashrc
drwx----- 16 sisops sisops 4096 Mar  7 11:59 .cache
drwx----- 14 sisops sisops 4096 Mar  7 11:52 .config
drwx-----  3 sisops sisops 4096 Jun  4 2012 .dbus
drwxr-xr-x  2 sisops sisops 4096 Mar 21 12:48 Desktop
-rw-r--r--  1 sisops sisops   26 Apr  5 09:18 .dmrc
drwxr-xr-x  2 sisops sisops 4096 Feb 19 11:15 Documents

```

Fig. 1: Despliegue de *ej1*.

El comando `ps -l` muestra a *ej1* como un proceso porque es un shell. Entonces, *ej1* es en realidad un proceso `tcsh`. En la figura se puede ver que el PID de *ej1* es 3020 y que su PPID es 3012. De aquí se puede inferir que su padre es el proceso `tcsh` y que, a su vez, el proceso padre de `ps -l` es *ej1*.

3. ¿Cuál es la función de la variable especial `$$`? ¿Qué valor arroja el siguiente comando y a qué se refiere dicho valor: `prompt>echo $$`? Agregue el comando anterior a *ej1*, ejecútelo y diga a qué se refiere el valor arrojado por `echo $$`.

La variable `$$` contiene el PID del proceso actual. El comando `prompt>echo $$` arroja como resultado el valor 3012, el cual, como ya habíamos visto, es el PID de `tcsh`. Al agregar el comando anterior a *ej1*, éste arroja 3059, que es el PID de *ej1*.

4. Agregue los comandos necesarios al script *perin* para que los datos sean pedidos, al usuario, adecuadamente.

En la figura 2, se puede ver que se agregaron dos comandos `echo` antes de solicitar los valores para que dicha solicitud fuera visible al momento de ejecutar el script.

```
#!/bin/tcsh
set i = 3
set j = $i
echo "i es " $i "j es " ${j}
echo "Ingrese v1"
set v1 = $<
echo "Ingrese v2"|
set v2 = $<
echo "v1 es " ${v1} "v2 es " $v2
```

Fig. 2: Contenido del script *perin*.

5. Del ejemplo de la página 4.19, diga qué significa el valor 0 y qué significa el valor 1 devueltos por `$status`. Usando los comandos `cp` y `ls`, provoque un acierto y un error en cada caso.

Un 1 significa que la operación dio falso u ocasionó un error, mientras que un 0 significa que la operación se realizó correctamente o dio verdadero. En las figuras siguientes se muestran ejemplos de comandos `cp` (figura 3) y `ls` (figura 4) que resultaron en distintos valores de `$status`.

```
ubuntu:~/ud> cp x y; echo $status
cp: cannot stat `x': No such file or directory
1
ubuntu:~/ud> cp perin pilin; echo $status
0
```

Fig. 3: Provocando un error y un acierto con `cp`.

```

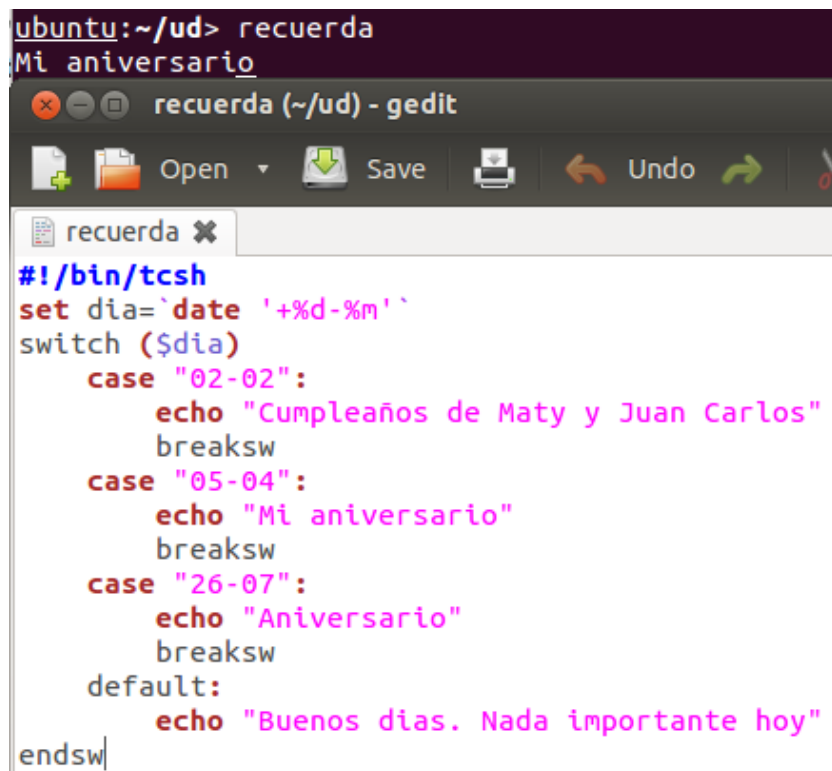
ubuntu:~/ud> ls z; echo $status
ls: cannot access z: No such file or directory
2
ubuntu:~/ud> ls ; echo $status
arbol      arroba      checa      dir      parpos      perin      pilin      recuerda      sol
arbol.txt  Bienvenida  cx         fact     parpos2     perin~     planta     saludo       sustcm
0

```

Fig. 4: Provocando un error y un acierto con ls.

6. Modifique el script de la página 4.24 de tal manera que al ser ejecutado imprima “Mi Aniversario” solo para este día.

Para esto fue necesario solamente agregar otro `case` con la fecha del día deseado en el formato que se especificó al obtenerla con `date`. El script, así como el resultado de su ejecución, se pueden ver en la figura 5.



```

ubuntu:~/ud> recuerda
Mi aniversario

```

```

#!/bin/tcsh
set dia=`date +%d-%m`
switch ($dia)
    case "02-02":
        echo "Cumpleaños de Maty y Juan Carlos"
        breaksw
    case "05-04":
        echo "Mi aniversario"
        breaksw
    case "26-07":
        echo "Aniversario"
        breaksw
    default:
        echo "Buenos días. Nada importante hoy"
endsw

```

Fig. 5: Uso de la estructura switch.

7. ¿En qué es diferente foreach, en cuanto a funcionamiento, de la

tradicional instrucción repetitiva `for` del lenguaje C, C++ o Java?

El `for` recibe un contador y va a ejecutar las instrucciones en su interior iterando sobre el contador. Es decir, el valor del contador se va a ir modificando con cada ejecución, y la ejecución va a continuar hasta que el contador se salga del rango especificado. Por el otro lado, el `foreach` itera sobre una estructura de datos para ejecutar un conjunto de instrucciones sobre cada uno de los datos.

8. ¿Cuál es el objetivo de la estructura `if-then-else`? ¿Cómo se obtuvo el valor que se asignó a `cta` en el script *fact*?

Esta estructura sirve para ejecutar instrucciones dependiendo de una condición. Esta condición se incluye dentro del `if`. El `then` contiene lo que se deberá ejecutar en caso que la condición especificada se cumpla mientras que el `else` contiene aquello que se deberá ejecutar en caso contrario. En el script *fact*, el valor de `cta` corresponde al primer argumento que reciba el script o, si no recibió ninguno, al valor que se ingrese en ese momento.

9. Explique el programa del script *parpos2*.

En este script se van recorriendo los argumentos recibidos una posición a la izquierda cada vez. Con esto se logra que el número de argumentos vaya decrementando de a uno en uno ya que el argumento `$1` se pierde después de cada `shift`. Entonces, se puede imprimir el valor del argumento en la posición 1 en repetidas veces y éste será distinto en cada iteración debido a los `shifts`. Así, con este script podemos imprimir los n argumentos recibidos imprimiendo solamente el primero y ejecutando `shift` n veces.

10. En el script *arbol*, ¿cuál es el objetivo de la variable `cabeza`?

En esta variable se coloca la primera línea del archivo si es que ésta contiene el texto `"tcsh"`. Si no es vacía, significa que el archivo encontrado es un script de `tcsh`. En caso contrario, el archivo no es un script y se puede descartar.

Conclusiones

La presente práctica nos permitió comprender de manera más tangible el concepto de **shell**, ya que teóricamente no se había profundizado acerca de sus funciones y potencial como intérprete de comandos. Pudimos trabajar, activar y desactivar, crear, y visualizar el estado de las diferentes variables de **shell**. Además, a diferencia de otras prácticas en esta ocasión trabajamos con guiones o *scripts* que nos permitieron ejecutar más de una instrucción a la vez. Finalmente, aprendimos la importancia de generar *scripts* para tareas que pueden ser muy tardadas o repetitivas como para ser ejecutadas línea a línea dentro del **shell**.

Referencias

- Ríos, J. (2019). Notas del curso de Sistemas Operativos. Recuperado el 03 de abril de 2019, del sitio web: Comunidad ITAM.