Interprete de comandos

1. Introducción

El intérprete de comandos permite la comunicación entre el sistema y el usuario. Aunque actualmente las interfaces gráficas de usuario (GUI) facilitan el trabajo cotidiano, todavía existen funciones que se resuelven mejor desde la línea de comandos. Algunas ventajas de los interpretes de comando son las siguientes:

- Menor consumo de recursos (muy importante si se actúa sobre sistemas remotos mediante una conexión lenta).
- Posibilidad de programar scripts para automatizar tareas administrativas.

Existe una gran variedad de interpretes de comandos, algunos ejemplos son los siguientes:

- Bourne Again Shell (bash)
- Korn Shell (ksh)
- C Shell (csh)
- Bourne Shell (sh)
- command.com
- cmd.exe

En la siguiente sección se muestra algunos comandos que podemos ejecutar en el interprete *bash*.

2. Comandos

Las Tablas 1 y 2 muestran algunos ejemplos de comandos en Bash. Para cada comando se muestra, el nombre del comando, su descripción, algunos de sus parámetros y un ejemplo.

Comando	Descripción	Parámetros	Ejemplo
ls	Lista los archivos y directorios.	▪ -a: Todos loa archivos ▪ -l: Listar con detalles ▪ -R: Listar recursivamente los subdirectorios	ls -l
cd	Permite cambiar de directorio.	▪ $\sim$: Regresa al Home ▪ ..: Regresa a un directoro atras.	cd ..
pwd	Permite conocer la ubicación actual en el sistema de archivos.		
clear	Limpia pantalla.	▪ -x: No elimina el contenido del scrollback	clear -x
cat	Muestra el contenido de un archivo de texto.	▪ -n: Muestra el número de línea. ▪ -e: Muestra un $ al final de cada línea.	cat -n file.txt
more	Muestra el contenido de un archivo de texto por partes.	▪ -n: Número de líneas a desplegar	more -n 2 file.txt

Tabla 1: Ejemplos de comandos (Parte 1)

Comando	Descripción	Parámetros	Ejemplo
cp	Copiar un archivo o directorio.	▪ -r: Copia de forma recursiva (para directorios) ▪ -i: Copia de forma interactiva	cp -r directorio rutaACopiar
ps	Despliega información de los procesos actuales	▪ -e: Muestra todos los procesos. ▪ -f: Muestra información detallada de los procesos. ▪ -o: Muestra los procesos en un formato dado por el usuario.	ps -eo user,pid
top	Despliega una fotografía de los procesos en tiempo real.	▪ -u/-U: Muestra los procesos de un usuario ▪ -p/-pid: Monitorea solo el proceso con un pid particular.	ps -u invitado
pstree	Muestra el árbol de procesos	▪ -c: Muestra el árbol sin compactar. ▪ -p: Muestra el árbol de un proceso particular.	pstree -c
man	Despliega la ayuda de un comando.		man ls

Tabla 2: Ejemplos de comandos (Parte 2)

3. Parámetros de un comando

Los comandos como varios programas pueden recibir parámetros que permiten cambiar su comportamiento por default o bien su salida en pantalla. Algunos

parámetros de comando son: 

3.1. ls

¿Qué se muestra al ejecutar los siguientes comandos?:

1. ls -l
2. ls -a
3. ls -la

3.2. date

¿Qué se muestra al ejecutar los siguientes comandos?:

1. date -d "tomorrow"(date -v +1d)
2. date -d "1 weeks ago"(date -v +1w)
3. date +%A-%B-%Y

3.3. touch

¿Qué se muestra al ejecutar los siguientes comandos?:

1. touch -m –date=”anio-mes-dia” archivo
2. touch -t AAAAMMDDhhmm archivo

3.4. Tuberías y redireccionamiento

Las tuberías permiten que la salida de un comando sea la entrada de otro. El símbolo para representar las tuberías en el bash es |. Ejemplo:

```
ls -l | wc -l
```

Muestra la cantidad de líneas que despliega el comando ls.

Ubuntu:

```
grep "MemTotal" /proc/meminfo | awk {'print $2'}
```

Filtra la salida de:

```
grep "MemTotal" /proc/meminfo
```

y muestra únicamente la segunda columna.

macOS:

```
sysctl -n hw.memsize | awk '{print $0/1024/1024/1024 " GB"}'
```

Filtra la salida de:

```
    sysctl -n hw.memsize
```

Tomando el valor obtenido de sysctl, lo divide por 1024 tres veces (para convertir de bytes a gigabytes) y luego agrega "GB."al final para indicar la unidad.

En una termina compile y ejecute el siguiente programa:

```
#include <sys/types.h>
#include <unistd.h>

int main(){
   int i;
   if(fork())
       for(i=0;i<2 && fork();i++)
           fork();
   sleep(1000);
   return 0;
}
```

Después, en otra terminal ejecute la secuencia de comandos:

```
ps -eo pid,user,comm | grep "fork" | head -n 1 | awk {'print $1'} |
    xargs pstree -c
```

¿Qué observa?

4. Salida a un archivo

Para enviar la salida de un comando a un archivo se utiliza el símbolo >. Ejemplo:

```
date > salida
```

Si la salida se desea enviar a un archivo existente se utiliza:

```
date >> salida
```

5. Script

Un *script* es un archivo de texto que contiene comandos a ejecutar. Se ejecuta de forma similar a un ejecutable C, usando ./. La primer línea del script hace referencia al interprete de comando.

```
#!/bin/bash
```

Si la ruta es incorrecta, se puede utilizar el comando *whereis* para buscar el ejecutable:

```
whereis bash
```

Posteriormente se tienen que asignarle permisos de ejecución, lo cual se puede realizar de la siguiente forma:

```
chmod +x nombre del archivo script
```

5.1. Ejemplo 1

Escribir un archivo llamado ejemplo1.sh y agregarle el texto:

```
#!/bin/bash
echo "Hola mundo"
```

¿Qué se muestra al ejecutar el script?

5.2. Ejemplo 2

Escribir un archivo llamado ejemplo2.sh y agregarle el texto:

```
#!/bin/bash

if [ "$1" = "" ]
then
  echo "Debe indicar el nombre del directorio a utilizar."
  exit
fi

if [ -e $1 ]
then
    echo "Ok: existe el directoio"
else
    mkdir $1
    echo "Creando el directorio: "$1
fi

echo "Accediendo al directorio...."
cd $1

path="cdn.mindmajix.com/blog/images/what-is-linux-120619.png"

curl -O $path

if [ $? -ne 0 ]
then
```

```
   echo "Archivo no descargado...Error"
else
   echo "Archivo descargado..."
fi
```

```
#!/bin/bash

if [ "$1" = "" ]
then
  echo "Debe indicar el nombre del directorio a utilizar."
  exit
fi

if [ -e $1 ]
then
    echo "Ok: existe el directoio"
else
    mkdir $1
    echo "Creando el directorio: "$1
fi

echo "Accediendo al directorio...."
cd $1

path="cdn.mindmajix.com/blog/images/what-is-linux-120619.png"

wget -q $path

if [ $? -ne 0 ]
then
   echo "Archivo no descargado...Error"
else
   echo "Archivo descargado..."
fi
```

¿Qué se ocurre al ejecutar el script?

5.3. Ejemplo 3

Escribir un archivo llamado ejemplo3.sh y agregarle el texto:

```
#!/bin/bash
n=1
while [ $n -le 6 ]; do
        echo $n
        let n++
done
```

¿Qué se muestra al ejecutar el script?

5.4. Ejemplo 4

Considere que trabaja en red (con mascara 255.255.255.0/24) y tiene un archivo llamado máquinas con la información:

```
192.168.2.1
192.168.2.2
192.168.2.3
192.168.2.4
192.168.2.5
```

y tiene el script:

```
#!/usr/bin/bash

for maquina in $(cat maquinas);
do
 online=$(nc -z $maquina 22 > /dev/null)
   if [ $? == 0 ]; then
      l1=$(ssh $maquina "lscpu | grep 'L1' | awk '{print \$3}'")
      l2=$(ssh $maquina "lscpu | grep 'L2' | awk '{print \$3}'")
      l3=$(ssh $maquina "lscpu | grep 'L3' | awk '{print \$3}'")
      echo "Maquina: "$maquina" L1:"$l1" L2:"$l2" L3:"$l3
   else

      echo "Maquina offline: "$maquina
   fi
done
```

¿Qué se muestra al ejecutar el script?

5.5. Ejemplo 5

Considere el programa

```
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv){
   int i;
   int n;
   n=atoi(argv[1]);
   for(i=1;i<n;i++)
   if(fork())
       if(fork()){

    break;
       }
   sleep(1000);

   return 1;
}
```

y tiene el script:

```
#!/bin/bash

echo "Compilando el programa binario"
gcc binario.c -o binario
for (( k=1; k<=$1; k++ ))
do
     echo "Ejecutando el programa con "
     ./binario $k &
     pid=$(ps -eo pid,user,comm | grep 'binario' | head -n 1 | awk
    {'print $1'})
     echo "Pid de la Raiz:"$pid
     pstree -cp $pid
     killall binario
done
```

¿Qué se muestra al ejecutar el script?

6. Ejercicios

Resolver los siguientes problemas mediante script de bash. La salida de cada script debe ser FORMATEADA para que sea amigable, es decir, no solo se trata de desplegar la información.

1. Escribir un script que permita monitorear los usuarios en el sistema cada determinado tiempo. La información de la hora de monitoreo y los usuarios se guardan en una bitácora.

2. Escribir un script que genere N cadenas aleatorias. Las cadenas deben cumplir con ciertas restricciones. Entre 8 y 12 caracteres, dos símbolos numéricos, al menos dos letras mayúsculas, al menos dos letras minúsculas y al menos dos de los siguientes símbolos: $@, \#, \$, \%, \&$. Las cadenas se almacenan en un archivos (si lo especifica el usuario) o se muestran en pantalla.

3. Debe implementar el árbol n-ário de procesos fork de profundidad p. Cada proceso debe ejecutar un sleep muy grande (no se requieren operaciones como wait). Por medio de un script debe ejecutar su programa. Después de que el árbol ha sido creado (todos los procesos están en el sleep), el mismo script debe matar los procesos de las hojas a la raíz. Antes de que un nivel sea eliminado, el script debe mostrar los procesos (su identificador) que siguen formando parte del árbol binario.

4. Realizar un árbol de procesos que represente un flor. Se debe indicar el número de procesos que integran los TALLOS, las FLORES y los PÉTALOS de las flores (ver Figura 1). Después de que el árbol ha sido creado (todos los procesos están en el sleep), el mismo script debe matar los procesos de las hojas a la raíz. Antes de que un nivel sea eliminado, el script debe mostrar los procesos (su identificador) que siguen formando parte del árbol binario.

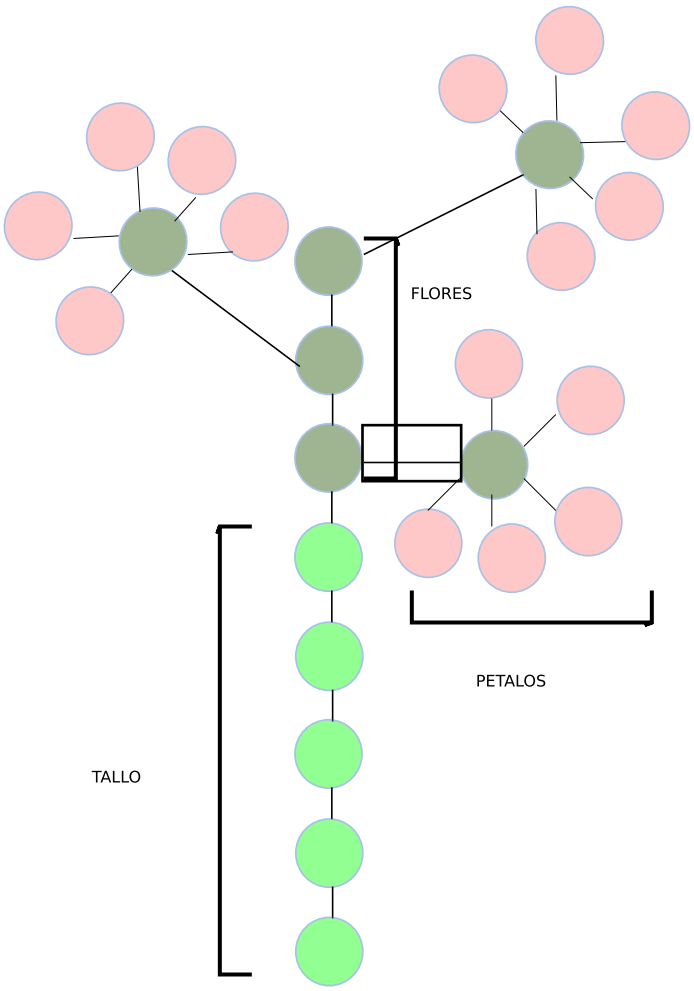

Figura 1: Árbol de procesos que representan una flor.

5. Por cada directorio en la ruta actual, desplegar el tamaño de cada subdirectorio en unidades legibles por una persona (bytes,kbytes, megabytes, etc.).