

Practica 1: Interprete de comandos

Adriana Pérez Espinosa
José Luis Quiroz Fabián

Enero 2017

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:

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

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

2 Comandos

2.1 date

Comando que permite obtener la fecha y hora del sistema.

2.2 whoami

Muestra el usuario que se encuentra ejecutando los comandos.

2.3 pwd

Permite conocer la ubicación actual en el sistema de archivos.

2.4 cd

Permite cambiar de directorio. ¿Qué pasa si ejecuta `cd~`?

2.5 ls

Muestra el contenido de un directorio. ¿Qué se muestra al ejecutar los siguientes comandos?:

1. `ls -l`
2. `ls -a`
3. `ls -la`
4. `man ls`

2.6 find

Permite buscar archivos/directorios. Ejemplo:

```
find . -name '*.jpg'
```

2.7 cat

Muestra el contenido de un archivo de texto.

2.8 more

Muestra el contenido de un archivo de texto por partes.

2.9 grep

Permite buscar/analizar un archivo tomando en cuenta parte de su información. Ejemplo,

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

2.10 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`.

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
```

Ejecute la secuencia de comandos

```
textitps -eo pcpu,user | grep "joseluis" | awk '{print $1}' | paste -sd+ | bc
```

¿Qué hace esta secuencia?

Ejecute el siguiente programa:

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

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

Después ejecute la secuencia de comandos:

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

3 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. En nuestro caso agregar:

```
#!/usr/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
```

3.1 Ejemplo 1

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

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

.

¿Qué se muestra al ejecutar el script?

3.2 Ejemplo 2

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

```
#!/usr/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="psdsuite.com/wp-content/uploads/2015/07/operating-system-icon-5746821.jpg"

wget -q $path

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

```

·
¿Qué se ocurre al ejecutar el script?

3.3 Ejemplo 3

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

```

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

```

·
¿Qué se muestra al ejecutar el script?

4 Ejercicios

Resolver los siguientes problemas.

1. Escribir un script que muestre la cantidad de cores y memoria (en Gbytes) que tiene la máquina donde están trabajando.
2. Escribir un script que muestre los procesos de un usuario X, pasado por parámetro, que consumen más CPU y más memoria.
3. Escribir un script que muestre la cantidad de procesos que hay en cada estado.
4. Escribir un script que muestre cuántos procesos tienen N hijos.
5. Escribir un script que muestre que proceso es el que tiene más hilos y cuántos son.
6. Escribir un script que muestre los primeros 10 procesos que consumen mayor CPU y cuánto consume cada uno.
7. Escribir un script para válida si una contraseña tiene un buen formato: mínimo 8 caracteres, al menos una símbolo numérico, al menos unos de los siguientes símbolos: @, #, \$, %, &, *, +, -, =
8. Escribir un script para que se encuentre monitorizando cada X segundos (X un parámetro) los procesos en un sistema. Si un proceso tiene más de 5 hijos, entonces el script envía un mail a un correo Y (Y un parámetro) con la información relevante: el tiempo, el PID del padre, PID de los hijos. El script genera una bitácora.
9. Un script que permita crear usuarios con contraseña (requiere permisos de administrador). La contraseña se tiene que validar como el punto 7. No se deben ver los caracteres cuando se introduce la contraseña.