Sistemas Operativos

Práctica 1. Introducción al interprete de comandos.

Prof. Jorge Matadamas Hernández

Octubre del 2022

1. Objetivo

Que el alumno practique la interacción con el sistema operativo mediante el interprete de comandos.

2. Introducción

2.1. Shell

Una de las funciones del sistema operativo consiste en ser un intermediario entre los recursos de un sistema de cómputo y los usuarios humanos. Las dos principales formas en que los usuarios pueden interactuar con el sistema operativo son: la interfaz gráfica de usuario (GUI) y la interfaz de líneas de comandos (CLI o shell). El shell predeterminado para muchas distribuciones de Linux es GNU Bourne-Again Shell (bash). Cuando una shell se utiliza de manera interactiva, muestra un *shell prompt* que termina con el carácter `$` o `#`, dependiendo de si el usuario es un usuario convencional o el superusuario *root* con mayores privilegios.

```
Ejecutar una shell           (ambiente gráfico vs linea de comandos)
Obtener privilegios de root  (su -, sudo, sudo su, $ vs #)
Instalar vim y git           (apt-get install, gestores de paquetes)
Salir de la shell de root    (exit o ctrl+d)
```

2.2. Comandos

La interacción se da cundo el usuario escribe un comando y el sistema operativo responde con una serie de acciones y/o resultados, según el comando. Los comandos están compuestos por tres partes:

- Comando. Es el comando a ejecutar.
- Opciones. Permiten ajustar el comportamiento del comando.
- Argumentos. Generalmente son los destinos del comando.

Las opciones, por lo general, son precedidas por uno o dos guiones (`-a o --all`) para que se distingan de los argumentos. Por ejemplo, el comando "ls" muestra los archivos y directorios del directorio actual; si adicionalmente se le agregan los modificadores "l" y "a" los muestra en forma de lista e incluye los archivos y directorios ocultos; si adicionalmente se le pasa un argumento mostrará los archivos y directorios del directorio que se le pase como argumento:

```
ls
ls -l -a                    (ocultos, extensiones)
ls -la Escritorio/
```

Para conocer las opciones y los argumentos que algún comando acepta y el orden en que se espera que se introduzcan, se puede consultar el manual, ejecutando el comando `man` seguido del comando sobre el cuál se desea saber, por ejemplo:

```
man ls
man man
```

Nota: Para salir del manual se presiona la tecla "q".

El manual, entre otras cosas, muestra una descripción del comando, la forma en que se utiliza y una lista de argumentos; usa las siguientes convenciones:

- Las palabras en `negritas` se escriben tal cual.
- Los corchetes "`[ ]`" indican elementos opcionales.
- Lo que va seguido de "`...`" se puede repetir.
- Si hay múltiples elementos separados por "`|`", únicamente se puede especificar uno de ellos.

Ejemplos de comanods:

```
clear            (ctrl+l)
cd ruta          (rutas absolutas y relativas)
                 (sin ruta, ..,~,-)
pwd
mkdir            (-p)
cp               (-r)
mv
rm               (-r)
touch            (extensiones, ocultos)

find . -name '*.jpg'     (comodínes * y ?)
head -n 3 /etc/passwd
tail -n 3 /etc/passwd
wc                       (renglones, palabras, bytes)
wc -l /etc/passwd
history
vim .bash_history

exit              (ctrl+d)
```

Además del manual, en Internet hay mucha información sobre cómo utilizar un comando en particular o qué comando utilizar para una tarea específica.

2.3. Estructura del sistema de archivos

El sistema de archivos son las reglas y la forma en que el sistema operativo almacena y gestiona los archivos. La estructura de los archivos en linux es muy parecida a la que se muestra en la Figura 1 (falta el directorio `/var`).

2.4. Redireccionamiento

El redireccionamiento de E/S reemplaza los canales predeterminados (stdin, stdout, stderr) por otros archivos o dispositivos. Así es posible enviar la salida de un proceso a un archivo utilizando ">" (si el archivo existe se sobre escribe) y "> >" (si el archivo existe se agrega al final).

```
date > fecha.txt
cat fecha.txt
date >> fecha.txt
ls -l ~ > salida.txt
```

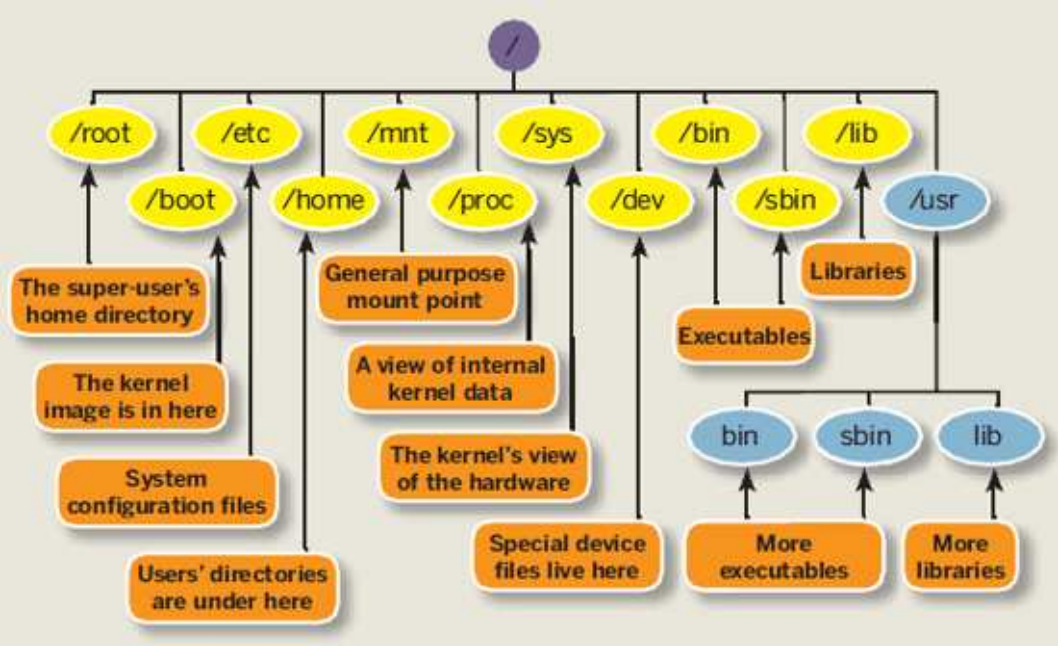

Figura 1: Estructura del sistema de archivos Linux.

El archivo `/dev/null` es un archivo especial que siempre está vacío, a pesar de que se escriba en él cualquier cantidad de información, por lo general se utiliza para redireccionar a él la salida estándar o el error estándar de algunos procesos, sin preocuparse por ocupar espacio en el disco.

```
ls -l ~ > /dev/null
```

2.5. Tuberías

Las tuberías (pipe) nos permiten ejecutar secuencias de comandos, permiten que la salida de un comando sea la entrada de otro. Los comandos se escriben separados por "|".

```
history | grep cp
ps aux | grep bash
```

2.6. vim

Vim es un editor de textos que funciona desde el shell, es decir que no necesita una interfaz gráfica. Se puede utilizar para editar archivos de configuración del sistema y para hacer programas o scripts. Una forma de ejecutarlo es tecleando en el shell:

```
vim nombredelarchivo
```

Donde "`nombredelarchivo`" es el nombre de un archivo, si el archivo no existe se crea.

Cuando vim arranca lo hace en modo "comando", desde el modo "comando" se puede cambiar al modo "edición" prsionando la tecla "i" o al modo "comando extendido" presionando la tecla ":". Para regresar al modo "comando" se presiona la tecla "esc".

En el modo "comando" se pueden teclear comandos para navegar el archivo. En el modo "edición" se pueden utilizar las teclas: flechas, enter, backspace, shift, letras y números para navegar y editar el archivo. Mientras que en el modo "comando extendido" se pueden teclear comandos para hacer búsquedas, editar y guardar el archivo (entre otros).

El flujo básico de trabajo es:

- Abrir un archivo con: `vim nombredelarchivo`
- Presionar `i` para entrar al modo edición
- Editar el archivo
- Presionar `esc` para vlolver al modo comando
- Escribir `:w` para guardar los cambios
- O escribir `:wq` para guardar y salir vim
- O escribir `:q!` para descartar los últimos cambios y salir

Ejemplos:

```
vim hola.txt      (se crea y se edita)
cat hola.txt      (se revisa su contenido)

vim hola.c        (escribir un hola mundo en C)
gcc hola.c        (compilación, opción -o)
./a.out           (ejecución)

vim Hola.java     (escribir un hola mundo en Java)
javac Hola.java   (compilación, .class, bytecode/JVM)
java Hola         (ejecución)
```

2.7. Scripts bash

Un *script bash* es un archivo de texto con comandos bash, los cuales serán ejecutados en el shell sin tener que teclearlos uno a uno. Es costumbre que el archivo tenga la extensión `.sh` y que la primera linea sea el siguiente comentario:

```
#!/bin/bash          (ruta donde se encuentra bash)
```

El script se ejecuta escribiendo en el shell:

```
bash nombreArchivo.sh
```

o

```
./nombreArchivo.sh
```

En el segundo caso es necesario que el archivo tenga permisos de ejecución (`chmod +x nombreArchivo.sh`).

Ejemplos de programas bash.

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

for i in {1..10}
do
    echo Hola Mundo $i
done


*******************************
#!/bin/bash

n=1
while [ $n -le 6 ]
do
  echo $n
  n=$(( $n + 1 ))
done

*******************************
#!/bin/bash

echo -n "Enter Number: "
read x

if [ $((x%2)) == 0 ]; then
  echo "Number is Even"
else
  echo "Number is Odd"
```

```
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 directorio"
else
        mkdir $1
        echo "Creando el directorio: " $1
fi

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

dir="https://www.debian.org/logos/openlogo-nd-100.jpg"

wget -q $dir

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

Referencia Bash scritping: `https://devhints.io/bash`

2.8. awk

El comando awk (que también es un lenguaje) es útil para procesar y analizar texto que está organizado en líneas y columnas, se utiliza en los *scripts* para filtrar los resultados de otros comandos y el contenido de archivos. La forma básica de un comando awk es:

```
awk [condición] '{comandos}' archivo
```

Ejemplos de uso de `awk`

```
awk -F : '{print $1}' /etc/passwd

Imprime la primera columna del archivo /etc/passwd considerando que el carácter
de separación es ":"


ps -eo pid,user,comm | grep "bash" | head -n 1 | awk '{print $1}'

¿Que significa el resultado y cómo se obtiene?
comandos utilizados: ps, grep, head, awk
```

3. Actividades

Conteste brevemente las siguientes preguntas:

1. ¿Qué es un sistema operativo?
2. ¿Qué es un intérprete de comandos?
3. ¿Cómo se llama el shell por defecto de Ubuntu, Red-hat, MacOS y Windows?
4. ¿Además de bash qué otros shell existen?
5. ¿Qué es un gestor de paquetes? y ¿Cuál es el nombre del gestor de paquetes por defecto de Ubunto y de Fedora?
6. ¿Cual es la relación entre Debian y Ubunto? y entre ¿Red-Hat y Fedora?
7. ¿Qué información se guarda en el directorio `/var`?
8. Investigue para que sirven los siguientes comandos: `whoami`, `grep`, `less`, `more`, `uname`, `top`, `du`, `df`, `ps`, `kill`, `find`, `chmod`, `chown`, `ssh`, `scp`, `uptime`, `w`, `wget`, `curl`, `mount`, `tree`
9. Resuelva los siguientes problemas haciendo un *script bash* con los comandos y tuberías necesarias (Incluir en la práctica la captura de pantalla de cada código y sus resultados).
    - obtener una lista de todos los procesos de tu usuario
    - Mostrar la cantidad de procesadores y cores que tiene la computadora.
    - Mostrar la cantidad de memoria (en GBytes) que tiene la computadora.
    - Mostrar el número de procesos que se están ejecutando en el sistema.

- Mostrar cuantos procesos corresponden al usuario root y cuantos a su usuario.
- Mostrar la información (idUsuario, nombre, % de consumo de CPU) de los 3 procesos que consumen más CPU.
- Validar si una contraseña tiene un buen formato: mínimo 8 caracteres, al menos un símbolo numérico, al menos uno de los siguientes símbolos: `@, #, $, %, &, *, +, -, =`