Práctica 3: comandos básicos de Shell

Dr. Hermes Francisco Montes Casiano

1. Introducción

Un *Sistema Operativo Linux* (ver Figura 1) se compone de cuatro partes:

- El núcleo
- Las utilidades de GNU
- El entorno gráfico
- Aplicaciones

Las utilidades de GNU tuvieron como principal objetivo de dotar a un Linux de un entorno similar a Unix, trasladando muchas de las utilidades de línea de comandos del sistema Unix (*CoreUtils*): utilidades para el manejo de archivos, utilidades para manejo de texto y utilidades para la gestión de procesos.

El shell bash fue desarrollado por el proyecto GNU como reemplazo del shell estándar de Unix, llamado shell Bourne (en honor a su creador). El nombre del shell bash es un juego de palabras, conocido como el *shell de Bourne back*. Las distintas distribuciones de Linux traen al menos una shell, en la tabla 1 se muestran algunas de las más comunes.

Con todas las nuevas funciones gráficas de escritorio de Linux, a veces encontrar una manera de obtener una *CLI* en una distribución de Linux no es una tarea fácil. Una forma de

Shell	Descripción
ash	Un shell simple, requiere poca memoria pero tiene compatibilidad total con shell bash.
korn	Un shell programable compatible con el shell Bourne pero con soporte para funciones de programación avanzadas como matrices asociativas y aritmética de punto flotante.
tcsh	Un shell que incorpora elementos del lenguaje de programación C en scripts de shell.
zsh	Un shell avanzado que incorpora funciones de bash, tcsh y korn, que proporciona funciones de programación avanzadas, historial compartido y *prompts* con temas visuales avanzados.

Tabla 1: Shells más comunes en las distribuciones de linux

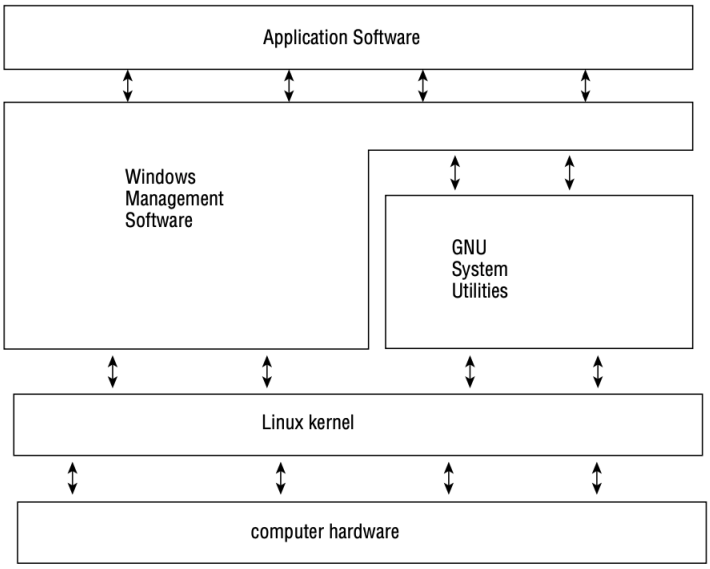

Figura 1: Sistema operativo linux

llegar a una CLI es sacar el sistema Linux del modo de escritorio gráfico y colocarlo en modo de texto. Esto no proporciona nada más que un shell de CLI simple en el monitor, como en los días anteriores a los escritorios gráficos.

El shell predeterminado que se usa en todas las distribuciones de Linux es el shell bash de GNU. El shell bash de GNU es un programa que proporciona acceso interactivo al sistema Linux. Se ejecuta como un programa regular, normalmente se inicia cada vez que un usuario inicia sesión en una terminal.

El shell que inicia el sistema depende de la configuración de su ID de usuario. El archivo */etc/passwd* contiene una lista de todas las cuentas de usuario del sistema, junto con información de configuración básica sobre cada usuario. Aquí hay una entrada de muestra de un archivo */etc/passwd*:

```
1 alumno:x:501:501:Juan Carlos Gallardo Barrera:/home/juan:/bin/
    bash
```

Dónde:

- Nombre del usuario
- Digesto del password o el indicador de que el password se ha almacenado en otro archivo
- Identificador del usuario
- Identificador del grupo al que pertenece el usuario

- Nombre completo del usuario
- El directorio *home*
- El shell por defecto para el usuario

2. Comandos básicos

Los comandos básicos de shell se listan en la tabla 2:

Tabla 2: Comandos básicos

Comando	**Descripción**	**Uso**
man	Permite recorrer las páginas de manual completas del shell bash. Es extremadamente útil al crear scripts, ya que no es necesario volver a consultar libros o sitios de Internet para buscar formatos específicos para los comandos.	$ man bash
cd	Permite cambiar el directorio de trabajo por otro, especificando un nuevo directorio mediante una ruta absoluta o relativa	$ cd destino
ls	Lista el contenido de un directorio	$ ls directorio
cat	El comando cat es una herramienta útil para mostrar todos los datos dentro de un archivo de texto	$ cat archivo
more	Muestra un archivo de texto, pero se detiene después de mostrar cada página de datos	$ more archivo
tail	Muestra el último grupo de líneas de un archivo, por defecto muestra las últimas 10 líneas, pero es configurable mediante parámetro	$ tail archivo
head	Muestra el primer grupo de líneas al comienzo de un archivo, por defecto mostrará las primeras 10 líneas de texto, pero es configurable mediante parámetro	$ head archivo
less	Proporciona una funcionalidad similar al comando *more*, además de funciones para desplazarse adelante y atrás de un archivo de texto, además de funciones de búsqueda avanzada	$ less archivo
ps	La salida básica muestra el ID de proceso (PID) de los programas, el terminal (TTY) desde el que se ejecutan y el tiempo de CPU que ha utilizado el proceso	$ ps
top	Funciona igual que el comando *ps* , muestra la información del proceso, con la diferencia que éste lo hace en tiempo real	$ top

Continua en la siguiente página

Tabla 2 – continuación de la página anterior

Comando	Descripción	Uso
kill	Permite enviar una señal a un proceso mediante su identificador de proceso. Por defecto, el comando *kill* envía la señal de término, pero se le puede especificar mediante parámetro la señal que se desea envíar.	$ kill -9 identificador
chmod	Modifica los bits de modo de un archivo o conjunto de archivos	$ chmod +x archivo
wc	Cuenta el numéro de palabras, líneas, caracteres y bytes de un archivo	$ wc -l archivo
echo	Escribe los operandos especificados separados por un espacio y seguido por el carácter de fin de línea '\n'.	$ echo hola mundo

Actividad 1: Ejecución de comandos

1. Abra una terminal
2. Ejecute cada uno de los comandos que se listan en la tabla 2.
3. Para obtener mayor detalle de la ejecución, conocer cuáles son las opciones y parámetros de cada comando, utilice el comando *man*.

```
$ man cat
...
cat [-benstuv] [file ...]
...
```

Ejemplo 1: Consultar el manual

4. Pruebe cada comando con al menos tres opciones y describa el resultado.

Cómo pudo darse cuenta, a la mayoría de los comandos se les puede especificar una funcionalidad en particular mediante opciones y argumentos, por ejemplo:

```
$ ls -lah
```

Ejemplo 2: Ejecución de un comando indicando opciones

Pregunta 1

¿Qué es una opción y un argumento y cuál es su función para la ejecución de un comando?

3. Ejecutando más de un comando

Hasta ahora, ha visto cómo usar el indicador de la interfaz de línea de comandos (CLI) del shell para ingresar comandos y ver los resultados de los comandos. La clave de los scripts de shell es la capacidad de ingresar múltiples comandos y procesar los resultados de cada comando, incluso posiblemente pasando los resultados de un comando a otro. El shell le permite encadenar comandos en una misma línea, separandolos por un punto y coma:

```
1 $ date ; pwd
2 Sun Apr 10 21:48:50 CDT 2022
3 /home/alumno
```

Ejemplo 3: Uso de múltiples comandos en una sóla línea

Actividad 2: Ejecución de múltiples comandos

1. Abra una terminal
2. Elija cinco combinaciones de al menos dos comandos
3. Ejecute las combinaciones de comandos en la terminal, separándolos por una coma, y describa la salida

Pregunta 2

¿Identifica alguna problemática o inconveniente con ésta técnica de ejecución de comandos?,¿por qué?

4. Creando un *script de shell*

En lugar de tener que ingresar manualmente los comandos en una línea de comandos, puede combinar los comandos y guardarlos en un archivo de texto simple. Cuando necesite ejecutar los comandos, simplemente ejecute el archivo de texto.

Cuando se crea un de script de shell, debe especificar el shell que está utilizando en la primera línea del archivo. El formato para esto es:

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

Ejemplo 4: Definición del shell a utilizar

En un script de shell el carácter # se utiliza para comentar una línea, sin embargo, su uso en la primera línea seguido del carácter *!* del archivo es un caso especial, ya que le indica al shell con qué shell ejecutar el script.

```
#!/bin/bash
# Creado mi primer script de shell
date
pwd
```

Ejemplo 5: Definición del shell a utilizar

Por otro lado, en un script de shell puede combinar comandos separándolos con un punto y coma o en diferentes líneas; shell procesará los comandos en el orden en el que se encuentren en el archivo. Las líneas que comienzan con el carácter # no son interpretadas.

Actividad 3: Ejecución de un script de shell

1. Abra el editor de texto *Visual Studio Code*.
2. Escriba el código del script 5 y guárdelo.
3. Abra una terminal y cambie el directorio de trabajo a la ubicación del script.
4. Ejecute el script que acaba de crear. Recuerde que si un ejecutable no se encuentra en el *PATH*, para ejecutarlo debe colocar el prefijo *./* al nombre del binario ejecutable.
5. En caso de que el script no se pueda ejecutar, realice las acciones necesarias para poder ejecutarlo. Ver el comando *chmod*.
6. Escriba sus observaciones