Linux y el interprete comandos

Dr. José Luis Quiroz Fabián

1 Introducción a Linux y sus distribuciones

Linux es un sistema operativo de código abierto basado en Unix, creado originalmente por Linus Torvalds en 1991. A diferencia de otros sistemas operativos comerciales, Linux se distribuye bajo la licencia GNU GPL, lo que permite a cualquier persona utilizar, modificar y distribuir su código fuente de forma libre.

Linux es ampliamente utilizado en servidores, supercomputadoras, sistemas embebidos, dispositivos móviles (como Android) y también en equipos de escritorio.

Dado que el núcleo de Linux (llamado kernel) solo proporciona las funcionalidades básicas, distintas comunidades y organizaciones han creado distribuciones, o distros, que combinan el kernel con herramientas, entornos gráficos, gestores de paquetes y otras utilidades.

Cada distribución tiene un propósito o enfoque distinto: algunas son ideales para usuarios novatos, otras para servidores, educación, seguridad informática o desarrollo de software. Ejemplos populares incluyen:

- Debian: Conocida por su estabilidad y filosofía de software libre.
- Ubuntu: Muy usada en entornos educativos y personales. Amigable para principiantes. Basada en Debian.
- Fedora: Similar a Ubuntu pero basada en RedHat.
- Arch Linux: Para usuarios avanzados que desean un control total del sistema.
- Kali Linux: Enfocada en pruebas de penetración y análisis forense digital.

2 Principales directorios del sistema Linux

El sistema de archivos en Linux sigue una jerarquía estándar definida por el Filesystem Hierarchy Standard (FHS). A continuación se presentan los directorios más importantes:

/ (raíz): Directorio principal del sistema. Todos los demás directorios cuelgan de este.

/bin Contiene los ejecutables esenciales del sistema accesibles por todos los usuarios. Incluye comandos básicos como `ls`, `cp`, `mv`, etc.

/boot Almacena los archivos necesarios para el arranque del sistema, como el kernel y el gestor de arranque.

/dev Contiene archivos de dispositivos que representan hardware del sistema.

/etc Archivos de configuración del sistema. Es crucial para el desarrollo web cuando se configuran servidores como Nginx, Apache o bases de datos.

/home Directorios personales de los usuarios. Aquí se desarrollan y almacenan proyectos en muchos casos.

/lib Bibliotecas compartidas necesarias para ejecutar programas en `/bin` y `/sbin`.

/media y

/mnt Puntos de montaje temporales para dispositivos externos como discos duros y memorias USB.

/opt Software adicional instalado manualmente. Algunos frameworks o entornos de desarrollo pueden instalarse aquí.

/proc Sistema de archivos virtual que expone información del kernel y procesos en tiempo real.

/root Directorio personal del usuario `root`.

/run Archivos de estado del sistema desde el arranque.

/sbin Comandos del sistema usados principalmente por el administrador (`root`).

/srv Contiene datos para servicios del sistema como `ftp`, `http`, etc. Muy relevante si se alojan sitios web directamente.

/tmp Archivos temporales. Pueden ser usados por aplicaciones web para almacenar archivos subidos o sesiones.

/usr Aplicaciones de usuario y sus dependencias. Contiene subdirectorios como `/usr/bin`, `/usr/lib`, `/usr/share`.

/var Archivos variables como logs, colas de impresión, cachés y archivos temporales de aplicaciones. Es vital para monitoreo y diagnóstico de aplicaciones web.

3 Interprete de comando

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*.

4 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 sub-directorios	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: Muestra los procesos de un usuario	ps -U invitado
man	Despliega la ayuda de un comando.		man ls

Tabla 2: Ejemplos de comandos (Parte 2)

5 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:

5.1 ls

¿Qué se muestra al ejecutar los siguientes comandos?:

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

5.2 date

¿Qué se muestra al ejecutar los siguientes comandos?:
macOS

1. date -v +1d

2. date -v -1w
3. date + %A- %B- %Y

Ubuntu

1. date -d "tomorrow"
2. date -d "6 weeks ago"
3. date + %A- %B- %Y

5.3 touch

¿Qué se muestra al ejecutar los siguientes comandos?:

Ubuntu

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

Ubuntu

5.4 apt-get

Comando para instalar paquetes en sistemas basados en Debian (ejecutar como root o con sudo). ¿Qué se muestra al ejecutar los siguientes comandos?:

sudo apt-get install cmatrix

5.5 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.

5.6 Compilación y ejecución

Un elemento fundamental es compilar y ejecutar en terminal. Por ejemplo, el siguiente programa en Java:

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola mundo");
    }
}
```

Compilación

```
javac HolaMundo.java
```

Ejecución

```
java HolaMundo
```

6 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
```

7 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
```

7.1 Ejemplo 1: Hola mundo

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

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

¿Qué se muestra al ejecutar el script?

7.2 Ejemplo 2: basico.sh

Ejemplo básico con comandos.

```
#!/bin/bash


if [ "$#" -ne 1 ]; then
  echo "Uso: $0 <nombre_dir>" >&2
  exit 1
fi

DIR="$1"
FILE="info.txt"

# 1) crear directorio si no existe
mkdir -p "$DIR" || { echo "No se pudo crear $DIR" >&2; exit 2; }
echo "Directorio creado/confirmado: $DIR"

# 2) entrar al directorio
cd "$DIR" || { echo "No se puede acceder a $DIR" >&2; exit 3; }

# 3) mostrar directorio actual
echo "Ahora en: $(pwd)"

# 4) crear un archivo simple con cat heredoc
cat > "$FILE" <<EOF
Este es el archivo $FILE
Creado por: $(whoami)
Ruta actual: $(pwd)
Fecha: $(date '+%F %T')
EOF

echo "Archivo creado: $FILE"

# 5) mostrar contenido con cat
echo
echo "Contenido de $FILE:"
cat "$FILE"

exit 0
```

7.3 Ejemplo 3: ciclos.sh

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

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

```
# Demostración de estructuras de repetición en Bash
set -euo pipefail

# --- Configuración inicial ---
MAX="${1:-6}"            # Máximo por defecto (puede pasarse como argumento)



# 1) while: contador ascendente
echo "while (contador ascendente)"
n=1
while [ "$n" -le "$MAX" ]; do
    printf "while: %d\n" "$n"
    ((n++))
done

# 2) until: contador descendente (ejecuta hasta que la condición sea verdadera)
echo "until (contador descendente)"
m=$MAX
until [ "$m" -le 0 ]; do
    printf "until: %d\n" "$m"
    ((m--))
done

# 3) for clásico (estilo C): índices y uso de continue/break
echo "for (( ... ))  con continue y break"
for ((i=1; i<=MAX; i++)); do
    if (( i % 2 == 0 )); then
        echo "for: $i (par)  continue"
        continue     # salta la impresión siguiente para números pares
    fi
    if (( i == 5 )); then
        echo "for: $i  alcanzado 5, break"
        break        # corta el ciclo cuando i == 5
    fi
    echo "for: $i (impar)"
done


# 4) ciclo anidado: tabla de multiplicar pequeña
echo "Ciclo anidado: tabla de multiplicar 1..5"
for ((r=1; r<=5; r++)); do
    for ((c=1; c<=5; c++)); do
        printf "%2d " $((r*c))
    done
    printf "\n"
done
```

¿Qué se muestra al ejecutar el script?

7.4 Ejemplo 4: filtrar.sh

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

```
#!/bin/bash

# Archivo de entrada con datos (uno por línea)
INPUT="datos.txt"

```

```
# Expresiones regulares
regex_email='^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
regex_tel='^[0-9]{3}-[0-9]{3}-[0-9]{4}$'

while IFS= read -r line; do
  if [[ $line =~ $regex_email ]]; then
    echo "EMAIL válido: $line"
  elif [[ $line =~ $regex_tel ]]; then
    echo "TELÉFONO válido: $line"
  else
    echo "Formato desconocido: $line"
  fi
done < "$INPUT"
```

¿Qué se muestra al ejecutar el script?