



# HERRAMIENTAS DE LÍNEA DE COMANDOS


## MÓDULO III

Sistemas Abiertos – Administración de SO I

1



### Contenido del Módulo III.

- **Introducción a las herramientas de línea de comandos.**
    - ¿Qué es una CLI? Ventajas de usar CLI. Shells disponibles en GNU/Linux. El shell Bash.
  - **Estructura general de un comando.**
    - Sintaxis. Componentes. Ejecución paralela y secuencial. Variables de entorno
  - **Comandos básicos.**
    - Navegación por el sistema de archivos. Gestión de archivos y directorios. Info del sistema. Permisos y usuarios.
  - **Redirección de entrada y salida.**
    - Redirección de salida. Redirección de entrada. Redirección de errores. Uso de /dev/null para descartar salidas.
  - **Tuberías y pipes.**
    - Concepto de tubería. Uso del operador |. Combinación de comandos con pipes.
  - **Expresiones regulares.**
    - Sintaxis básica. Caracteres especiales y clases de caracteres. Repeticiones y patrones. Casos de uso.
  - **Comodines.**
    - Uso de comodines. Diferencia entre comodín y expresión regular. Ejemplos.
- 

2

## Introducción a las herramientas de CLI.

- **¿Qué es una línea de comandos?**

- La línea de comandos (**CLI**, Command-Line Interface), es una interfaz de usuario que permite interactuar con el SO mediante el ingreso de comandos de texto.
- En los primeros días de Unix y GNU/Linux, el acceso a la línea de comandos se realizaba a través de **terminales electromecánicas**.
- Un ejemplo famoso de estas terminales es la **VT100**, desarrollada por Digital Equipment Corporation (DEC).
- Hoy en día, estas terminales físicas han sido reemplazadas **por emuladores de terminal**.
- La línea de comandos es un componente fundamental que permite realizar operaciones a nivel del sistema con mayor precisión y flexibilidad.

3

## Introducción a las herramientas de CLI.

- **Ventajas del uso de la terminal frente a interfaces gráficas.**

- Las GUI son más accesibles para usuarios menos técnicos, pero la línea de comandos ofrece ventajas, especialmente para tareas avanzadas o repetitivas:
  - **Eficiencia:** permite realizar operaciones complejas con unos pocos comandos.
  - **Control total:** ofrece acceso directo a casi todas las funciones del sistema.
  - **Automatización:** es más fácil automatizar tareas repetitivas mediante scripts.
  - **Uso de recursos:** consumen menos recursos en comparación con las GUI.
  - **Acceso remoto:** se puede acceder a la línea de comandos de un sistema remoto de manera eficiente.
  - **Compatibilidad universal:** tienden a ser más uniformes y consistentes entre diferentes distribuciones o versiones del SO.

4

## Introducción a las herramientas de CLI.

- **Shells disponibles en GNU/Linux.**

- Un intérprete de comandos, o shell, es un programa cuya principal función es:
  - recibir las órdenes del usuario
  - interpretar esa cadena de texto
  - ejecutar los programas correspondientes
  - mostrar el resultado.
- Algunos intérpretes disponibles para GNU/Linux son:
  - **Bash (Bourne Again Shell):** es el más utilizado.
  - **Zsh (Z shell):** similar a Bash, con algunas mejoras.
  - **Fish (Friendly interactive shell):** scripts con sintaxis más simple.
  - **Dash (Debian Almquist shell):** ligero y rápido, enfocado en scripts POSIX.
  - **Tcsh:** derivado de Csh, con sintaxis similar al lenguaje C.

5

## Introducción a las herramientas de CLI.

- **El Shell Bash (Bourne Again Shell).**

- Shell por defecto en la mayoría de las distros GNU/Linux y uno de los más populares y ampliamente utilizados.
- Desarrollado como mejora sobre el Bourne Shell original (sh) de Unix, hereda sus características, y añade funcionalidades más potentes y flexibles.
- Creado en 1989 por Brian Fox como parte del proyecto GNU con el objetivo de proporcionar una alternativa libre al shell Bourne de Unix.
- De ahí su nombre, Bourne Again Shell, un juego de palabras con "born again" (nacer de nuevo).
- Desde entonces, Bash ha evolucionado hasta convertirse en un componente clave de la mayoría de los sistemas basados en Unix.

6

## Introducción a las herramientas de CLI.

### • El Shell Bash (Bourne Again Shell).

- Características más importantes de Bash que lo diferencian de otros shells:
  - **Historial de comandos:** mantiene un registro de los comandos ejecutados en sesiones anteriores, lo que facilita la reutilización de comandos sin tener que reescribirlos.
  - **Redirección de entrada/salida:** permite redirigir la salida de los comandos a archivos o incluso a otros comandos. Esto es fundamental para la creación de flujos de trabajo eficientes.
  - **Alias:** permite crear alias para abreviar comandos largos o complejos. Por ejemplo, puedes crear un alias para `ls -la` con el nombre `ll`.
  - **Variables:** se pueden definir variables para almacenar información temporalmente dentro de una sesión de Bash, lo cual es muy útil en scripts y automatización.

7

## Introducción a las herramientas de CLI.

### • El Shell Bash (Bourne Again Shell).

- Características más importantes de Bash que lo diferencian de otros shells:
  - **Scripting avanzado:** no solo es un shell interactivo, sino también un potente lenguaje de scripting. Se puede escribir scripts complejos con soporte para variables, condicionales, bucles, y funciones.
  - **Autocompletado:** al presionar la tecla `Tab`, Bash puede completar automáticamente comandos, nombres de archivos, y directorios.
  - **Expansión de comodines (globbing):** soporta el uso de comodines como `*`, `?`, y `[]` para trabajar con múltiples archivos al mismo tiempo.
  - **Substitución de comandos:** permite ejecutar un comando dentro de otro. Esto es útil para usar la salida de un comando como entrada para otro.

8

## Introducción a las herramientas de CLI.

- **El Shell Bash (Bourne Again Shell).**

- **Configuración de Bash.**

- Bash es altamente configurable y personalizable. Algunas formas en las que se puede ajustar incluyen:
    - **Archivos de configuración:** tiene varios archivos de configuración, como `.bashrc` y `.bash_profile`, que permiten a los usuarios definir alias, variables de entorno, y personalizar su entorno de trabajo:
      - `~/ .bashrc`: Este archivo de configuración se ejecuta cada vez que se abre una nueva terminal. Pueden agregarse alias, funciones y otras personalizaciones aquí.
      - `~/ .bash_profile`: Usado al iniciar sesión en Bash, contiene configuraciones para sesiones de login.
    - **Alias:** se pueden crear alias personalizados para ejecutar comandos largos con nombres más cortos. Por ejemplo: `alias ll='ls -la'`. Esto permite ejecutar `ll` en lugar de escribir `ls -la` cada vez.

9

## Introducción a las herramientas de CLI.

- **El Shell Bash (Bourne Again Shell).**

- **Configuración de Bash.**

- **Prompt personalizado:** permite personalizar el prompt, esto es, la cadena de texto que se muestra antes de que el usuario escriba un comando. El prompt se configura a través de la variable de entorno `PS1`. Un ejemplo básico de un prompt personalizado es:

```
export PS1="\u@\h:\w\$ "
```

Este prompt muestra el nombre de usuario (`\u`), el nombre del host (`\h`), el directorio actual (`\w`), seguido del símbolo `$`.

10



# Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Sintaxis de los comandos.**

- Cada comando tiene una estructura básica, que puede variar en complejidad según las necesidades. La sintaxis básica de un comando es:

`comando [opciones] [argumentos]`

- **comando:** Es el nombre del programa o instrucción que a ejecutar. Por ejemplo, `ls`, `cp`, `mkdir`, etc.
    - **opciones:** Son parámetros que modifican el comportamiento del comando. Las opciones usualmente se anteceden con un guion simple (-) o doble (--) para diferenciarlas de los argumentos.
      - Ejemplo con guion simple: `ls -l`
      - Ejemplo con guion doble: `ls --long`
    - **argumentos:** Son los valores que el comando necesita para ejecutarse sobre un archivo, directorio o dato específico. Por ejemplo, al copiar un archivo, el archivo de origen y el destino serían los argumentos.

11

# Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Ejecución de comandos secuenciales y paralelos.**

- Se pueden ejecutar múltiples comandos de manera secuencial o en paralelo, lo que ofrece flexibilidad al usuario para automatizar tareas.

- **Ejecución secuencial:** Se ejecutan comandos uno tras otro, en el orden en que aparecen. Se utiliza el punto y coma (;):

`comando1 ; comando2 ; comando3`

- Por ejemplo:

`mkdir nuevo_directorio ; cd nuevo_directorio ; touch archivo.txt`

- Aquí, se crean secuencialmente un nuevo directorio, se cambia a ese directorio, y luego se crea un archivo en el mismo.

12

## Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Ejecución de comandos secuenciales y paralelos.**

- **Ejecución paralela:** Los comandos se pueden ejecutar en paralelo usando el operador `&`. Esto es útil cuando los comandos no dependen entre sí y pueden ejecutarse simultáneamente.

```
comando1 & comando2 &
```

- Por ejemplo:

```
ping google.com & ping bing.com &
```

- Esto ejecuta ambos comandos ping de manera simultánea, cada uno **en su propio proceso**.

13

## Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Variables de entorno y su influencia en la ejecución de comandos.**

- Son un conjunto de variables del sistema que influyen en el comportamiento de los comandos en la terminal.
    - Se almacenan en la memoria y contienen información que los programas y procesos utilizan para funcionar correctamente.
    - Algunas variables importantes son:
      - **PATH:** Define los directorios en los que el sistema busca comandos ejecutables.
      - **HOME:** Define el directorio personal del usuario actual.
      - **USER:** Contiene el nombre de usuario actual en la sesión de la terminal.
      - **PS1:** Define el aspecto del prompt.
      - **SHELL:** Indica qué shell se está utilizando en la sesión actual.

14

## Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Variables de entorno y su influencia en la ejecución de comandos.**

- Las variables de entorno se pueden establecer o modificar en tiempo real dentro de la sesión actual con el comando `export`:

```
export NOMBRE="valor"
```

- Por ejemplo:

```
export EDITOR=nano
```

- Esto indica que cualquier programa que consulte qué editor de texto utilizar debe usar `nano`.
  - Algunas variables de entorno, como `PATH`, se pueden modificar para que los cambios persistan entre sesiones. Para ello, puedes agregarlas a los archivos de configuración de Bash, como `.bashrc` o `.bash_profile`.

15

## Introducción a las herramientas de CLI.

- **Estructura general de un comando.**

- **Variables de entorno y su influencia en la ejecución de comandos.**

- Para listar todas las variables de entorno se puede utilizar el comando:

```
printenv
```

- Si se desea consultar el valor de una variable de entorno específica:

```
printenv NOMBRE_VARIABLE
```

- Por ejemplo, para ver el valor de la variable `PATH`, se puede ejecutar:

```
printenv PATH
```

- Otra alternativa para listar las variables de entorno es el comando:

```
env
```

- Ambos comandos son útiles para verificar las variables que están influyendo en el comportamiento del sistema y sus comandos en la terminal.

16



# Introducción a las herramientas de CLI.

## • Comandos básicos.

- Veremos una parte de los comandos básicos de GNU/Linux, útiles tanto para principiantes como para administradores.
- Podríamos clasificar los comandos básicos en cuatro categorías:
  - **Navegación por el sistema de archivos:** para acceder a los directorios, ver sus contenidos, etc.
  - **Gestión de archivos y directorios:** para copiar, mover, renombrar, etc., tanto archivos como directorios.
  - **Información del sistema:** conocer datos sobre los componentes hardware, como las unidades de disco, uso de memoria, etc.
  - **Manejo de permisos y usuarios:** modificar los permisos de usuarios sobre archivos o directorios, cambiar la pertenencia a grupos, etc.

17

# Introducción a las herramientas de CLI.

## • Comandos básicos.

- **Navegación por el sistema de archivos.**
- **Comando ls:** lista el contenido de un directorio.
  - Opción -l: listado detallado.

```
marcos@vbox: $ ls -l
total 40
-rw-r--r-- 1 marcos marcos 13 Sep 20 11:35 archivo.txt
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:36 carpeta
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Desktop
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Documents
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Downloads
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Music
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Pictures
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Public
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Templates
drwxr-xr-x 2 marcos marcos 4096 Sep 20 11:31 Videos
marcos@vbox: $
```

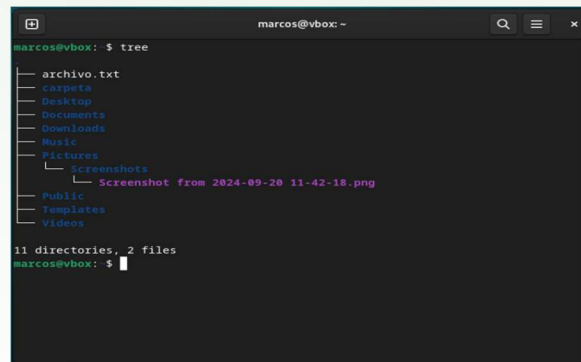
- Opción -a: muestra los archivos ocultos.
- Opción -h: combinado con -l, muestra los tamaños en formato legible.
- Opción -R: muestra el contenido de manera recursiva.

18

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Navegación por el sistema de archivos.**
- **Comando tree:** lista el contenido de un directorio.



```
marcos@vbox: ~  
$ tree  
.  
├── archivo.txt  
├── carpeta  
├── Desktop  
├── Documents  
├── Downloads  
├── Music  
├── Pictures  
│   └── Screenshots  
│       └── Screenshot from 2024-09-20 11-42-18.png  
├── Public  
├── Templates  
└── Videos  
  
11 directories, 2 files  
marcos@vbox: ~
```

19

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Navegación por el sistema de archivos.**
- **Comando cd:** cambia de directorio.
  - Sintaxis: `cd /ruta/a/directorio.`
  - Sin argumentos nos lleva al directorio personal.
  - Utilizando `~` también nos lleva al directorio personal. Se puede combinar con otros directorios: `cd ~/Documentos.`
  - Directorios especiales: `.` y `..`: el `.` es el directorio personal, y el `..` es el directorio por arriba de la jerarquía de directorios.
  - La barra `/` es el separador de directorios. Se combina con los directorios especiales:
    - `./archivo_ejecutable`
    - `../..` sube dos directorios en la jerarquía.

20

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Navegación por el sistema de archivos.**
- **Comando pwd:** imprime en pantalla el directorio de trabajo actual.

```
marcos@Casa-PC:~$ pwd
/home/marcos
marcos@Casa-PC:~$ |
```

- No confundir con el directorio de trabajo por defecto.

21

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Gestión de archivos y directorios.**
- **Comando cp:** copia archivos y directorios.
  - Sintaxis: `cp archivo_origen archivo_destino.`
  - `-r` permite copiar directorios con su contenido, sin eso no se puede.
  - `-i` pregunta antes de copiar.
  - `-u` copia sólo si el archivo destino es más antiguo que el origen o si no existe.
  - `-v` muestra en pantalla lo que va copiando.
  - `-p` preserva los atributos originales.
  - `-a` combinación de las opciones `-r`, `-p` y `-d`, hace una copia exacta de un directorio.

22

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Gestión de archivos y directorios.**

- **Comando mv:** mueve archivos y directorios.

- Sintaxis: `mv archivo_origen directorio_destino.`
    - Si se especifican dos archivos, renombra el primero con el segundo.
    - `-i` pregunta antes de mover.
    - `-n` no sobrescribe.
    - `-b` hace un backup del origen con el carácter `~` al final.
    - `-S` cambia el sufijo para el backup, se usa con `-b`.

23

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Gestión de archivos y directorios.**

- **Comando rm:** elimina archivos.

- Sintaxis: `rm archivo.`
    - `-i` pregunta antes de eliminar.
    - `-f` fuerza la eliminación.
    - `-d` elimina directorios vacíos.
    - `-r` elimina un directorio no vacío.

24

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Gestión de archivos y directorios.**

- **Comando mkdir:** crea directorios.

- Sintaxis: `mkdir directorio`.
    - `mkdir directorio{1,2,3}`.
    - `mkdir -p proyecto/{src,bin,doc}`.
    - `mkdir -p ruta/a/nuevo/directorio`.

25

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Gestión de archivos y directorios.**

- **Comando rmdir:** elimina directorios vacíos.

- Sintaxis: `rmdir directorio`.
    - `-r` elimina el directorio con su contenido.
    - `-p` elimina directorios padres si están vacíos.

26



## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Información del sistema.**

- **Comando df:** muestra el uso de espacio en disco total.

- Sintaxis: df.
    - -h muestra los tamaños en formato legible.

```
marcos@Casa-PC:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
none            7.8G  0  7.8G  0% /usr/lib/modules/5.15.153.1-microsoft-standard-WSL2
none            7.8G  4.0K  7.8G  1% /mnt/wsl
drivers         465G  89G  377G  20% /usr/lib/wsl/drivers
/dev/sdc        1007G  763M  955G  1% /
none            7.8G  76K  7.8G  1% /mnt/wslg
none            7.8G  0  7.8G  0% /usr/lib/wsl/lib
rootfs          7.8G  2.2M  7.8G  1% /init
none            7.8G  0  7.8G  0% /dev
none            7.8G  4.0K  7.8G  1% /run
none            7.8G  0  7.8G  0% /run/lock
none            7.8G  0  7.8G  0% /run/shm
none            7.8G  0  7.8G  0% /run/udev
tmpfs           7.8G  0  7.8G  0% /sys/fs/cgroup
none            7.8G  92K  7.8G  1% /mnt/wslg/versions.txt
none            7.8G  92K  7.8G  1% /mnt/wslg/doc
C:\             465G  89G  377G  20% /mnt/c
D:\             1.9T  312G  1.6T  17% /mnt/d
E:\             448G  228G  220G  51% /mnt/e
F:\             448G  270G  178G  61% /mnt/f
marcos@Casa-PC:~$
```

27

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Información del sistema.**

- **Comando du:** muestra el uso de espacio por archivos y directorios específicos.

- Sintaxis: du.
    - -h muestra los tamaños en formato legible.

```
marcos@Casa-PC:~$ du
4  ./local/share/nano
8  ./local/share
12 ./local
116 .
```

28

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Información del sistema.**
- **Comando free:** muestra el uso de memoria.
  - Sintaxis: `free`.
  - `-h` muestra los tamaños en formato legible.

```
marcos@Casa-PC:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:            15Gi       584Mi        15Gi        2.5Mi        97Mi        15Gi
Swap:           4.0Gi          0B         4.0Gi
```

29

## Introducción a las herramientas de CLI.

- **Comandos básicos.**

- **Información del sistema.**
- **Comando uname:** muestra el información del sistema operativo.
  - Sintaxis: `uname`.
  - `-a` muestra información más detallada.

```
marcos@Casa-PC:~$ uname -a
Linux Casa-PC 5.15.153.1-microsoft-standard-WSL2 #1 SMP Fri Mar 29 23:14:13 UTC 2024 x86_64 GNU/Linux
marcos@Casa-PC:~$
```

30

## Introducción a las herramientas de CLI.

- **Redirección de entrada y salida.**

- **Entrada, salida y error estándar.**

- **Entrada estándar (stdin):** La **entrada estándar** es el flujo de datos que un comando o programa **recibe**.

- Por defecto es el teclado, pero puede ser un archivo o la salida de un comando.

- Utiliza el descriptor de archivo 0.

- **Salida estándar (stdout):** La **salida estándar** es el flujo de datos que un programa o comando **envía como resultado de su ejecución**.

- Por defecto es la pantalla, pero se puede redirigir a un archivo.

- Utiliza el descriptor de archivo 1.

- **Error estándar (stderr):** El **error estándar** es el flujo donde los programas o comandos **envían los mensajes de error**.

- Por defecto es la pantalla, pero se puede redirigir a un archivo.

- Utiliza el descriptor de archivo 2.

31

## Introducción a las herramientas de CLI.

- **Redirección de entrada y salida.**

- **Redirección de salida.**

- Con el operador `>`, podemos redirigir la salida de un comando a un archivo.

- `comando > archivo.txt`

- Si queremos **agregar** la salida de un comando a un archivo sin sobrescribir su contenido, utilizamos el operador `>>`

- `echo "Nuevo contenido" >> archivo.txt.`

- **Redirección de entrada.**

- Utilizamos el operador `<` para tomar la entrada de un archivo en lugar de ingresarla manualmente.

- `cat < datos.txt`

32

## Introducción a las herramientas de CLI.

- **Redirección de entrada y salida.**

- **Redirección de error.**

- Usamos el descriptor 2 seguido del operador >.
    - `ls /directorio_no_existe 2> error.log`
  - Podemos redirigir la salida estándar y los errores a archivos diferentes usando ambos descriptores (1 y 2):
    - `comando > salida.txt 2> error.log`
  - Si queremos redirigir tanto la salida estándar como los errores al mismo archivo, usamos el operador &>.
    - `ls /directorio /directorio_no_existe &> salida_y_errores.txt`

33

## Introducción a las herramientas de CLI.

- **Redirección de entrada y salida.**

- **Redirección hacia /dev/null.**

- El archivo especial /dev/null se utiliza para descartar cualquier salida o error.
  - Es conocido como el “agujero negro” del sistema.
    - `comando > /dev/null`
  - Esto descarta cualquier salida. Si queremos descartar los mensajes de error:
    - `comando 2> /dev/null`
  - La salida estándar se mostrará normalmente, mientras que los errores se descartan.

34

## Introducción a las herramientas de CLI.

### • Tuberías (pipes).

#### • Concepto de tubería.

- Permiten conectar la salida de un comando con la entrada de otro comando.
- Podemos combinar varios comandos para realizar tareas complejas en una sola línea de comandos.
- Permite procesar datos de forma fluida sin necesidad de utilizar archivos temporales.
- Se utiliza el operador |. Por ejemplo:
  - `ls | grep archivo`
- El comando `ls` genera una lista de archivos, que se redirige al comando `grep`, el cual filtra la lista y muestra en pantalla los nombres de archivos que contienen la palabra “archivo”.

35

## Introducción a las herramientas de CLI.

### • Tuberías (pipes).

#### • Uso del operador |.

- Cuando colocamos el operador entre dos comandos, la salida del primer comando se convierte en la entrada del segundo:
  - `comando1 | comando2`
- Un ejemplo sencillo:
  - `cat archivo.txt | sort | uniq`
- `Cat` muestra el contenido de `archivo.txt`.
- `Sort` ordena alfabéticamente el contenido del archivo.
- `Uniq` elimina las líneas duplicadas de la salida ordenada del archivo.
- Todo esto se realiza en una sola línea, sin necesidad de crear archivos temporales.

36



## Introducción a las herramientas de CLI.

### • Tuberías (pipes).

- **Combinación de comandos con pipes: flujos de trabajo eficientes.**
- Mostrar los 5 procesos más consumidores de CPU:
  - `ps aux | sort -nrk 3,3 | head -n 5`
- Encontrar archivos grandes en un directorio:
  - `du -h /ruta/del/directorio | sort -hr | head -n 10`
- Monitorear procesos en particular.
  - `ps aux | grep apache | awk '{print $2, $3, $4}'`
- Automatización de backups:
  - `tar czf - /ruta/a/backup | ssh usuario@servidor "cat > /ruta/de/destino/backup.tar.gz"`

37

## Introducción a las herramientas de CLI.

### • Expresiones regulares (RegEx).

- **Introducción: sintaxis básica.**
- Secuencia de caracteres que define un **patrón de búsqueda**:
- A través de este patrón, podemos buscar coincidencias en cadenas de texto.
- Por ejemplo, para encontrar la palabra "error" en un archivo, usaríamos la expresión regular:
  - `grep "error" archivo.log`
- Esto buscará todas las líneas del archivo que contengan la palabra error.

38

## Introducción a las herramientas de CLI.

- **Expresiones regulares (RegEx).**

- **Caracteres especiales.**

- **Punto (.):** Representa cualquier carácter excepto una nueva línea.
      - `grep "a.b" archivo.txt`
    - Esto coincidirá con cualquier cadena que tenga una "a", seguida de cualquier carácter, y luego una "b", como "acb", "a7b", o "a-b".
    - **Caret (^):** Marca el inicio de una línea. Si queremos buscar líneas que comiencen con una palabra específica, usamos ^.
      - `grep "^inicio" archivo.txt`
    - Esto solo mostrará las líneas que comienzan con la palabra "inicio".
    - **Signo pesos (\$):** Marca el final de una línea.
      - `grep "fin$" archivo.txt`
    - Esto buscará líneas que terminen con la palabra "fin".

39

## Introducción a las herramientas de CLI.

- **Expresiones regulares (RegEx).**

- **Caracteres especiales.**

- **Asterisco (\*):** Coincide con cero o más ocurrencias del carácter o patrón anterior.
      - `grep "a*b" archivo.txt`
    - Esto coincide con cualquier número de "a" seguido de una "b", por ejemplo, "b", "ab", "aaab", etc.
    - **Signo más (+):** Coincide con una o más ocurrencias del carácter anterior.
      - `grep "a+b" archivo.txt`
    - Esto coincidirá con "ab", "aab", "aaab", etc., pero no con "b" solo.
    - **Signo de interrogación (?):** Coincide con cero o una ocurrencia del carácter o patrón anterior.
      - `grep "colou?r" archivo.txt`
    - Esto coincidirá con "color" y "colour", haciendo opcional la "u".

40

## Introducción a las herramientas de CLI.

- **Expresiones regulares (RegEx).**

- **Clases de caracteres.**

- Permiten definir un conjunto de caracteres entre los cuales podemos hacer coincidir.
    - Se encierran entre corchetes.

- **Rango de caracteres:** define rangos utilizando guiones. Por ejemplo, [a-z] coincidirá con cualquier letra minúscula.

- `grep "[0-9]" archivo.txt`

- Esto buscará cualquier línea que contenga un número.

- **Clases predefinidas:** útiles para patrones específicos:

- `\d`: Coincide con cualquier dígito (equivalente a [0-9]).
    - `\w`: Coincide con cualquier carácter de palabra (letras, dígitos, y guiones bajos).
    - `\s`: Coincide con cualquier espacio en blanco (espacios, tabulaciones, nuevas líneas).

- Un ejemplo con `\d` para encontrar números:

- `grep "\d" archivo.txt`

41

## Fin del Módulo III

42