Como crear un script:

`cd /dirección` : Ubicación en donde se guarda el script.

`sudo su` : Para tener acceso de superusuario.

`nano nombre.sh` : Nombre del script y abre el editor.

`#!/bin/bash` : Indica que es un script.

`echo "texto"` : Imprime una línea de texto.

`ctrl + o / intro` : Guardar cambios.

ctrl + x : Salir de nano.

`chmod 777 nombre.sh` : Otorgar permisos al script.

`./nombre.sh` : Para ejecutar el script.

ctrl + k : Para borrar líneas.

Comandos básicos:

`grep [opciones] patrón [archivo...] :` Permite buscar cadenas de texto o patrones dentro de archivos.

Ejemplos:
`-grep "palabra" archivo.txt :` Muestra las líneas que contienen esa palabra en archivo.txt
`-grep -r "palabra" /ruta/del/directorio` : Busca palabra en todos los archivos dentro del directorio
`-grep -i "palabra" archivo.txt :` Ignora minúscula y mayúsculas
`-grep -n "palabra" archivo.txt` : Lineas con la palabra “palabra” y el número de la línea
`-grep -v "palabra" archivo.txt` : Lineas que no contienen la palabra “palabra”
`-grep -e "patrón1" -e "patrón2" archivo.txt` : Esto buscará líneas que contengan cualquiera de los patrones especificados.

`cat` : Concatenate FILE(s), or standard input, to standard output.

Ejemplos:

`-cat archivo.txt` Esto muestra el contenido del archivo `archivo.txt`.
`-cat archivo1.txt archivo2.txt` Esto mostrará el contenido de `archivo1.txt` seguido del contenido de `archivo2.txt`.
`-cat > nuevo_archivo.txt` Después de ejecutar este comando, puedes escribir texto que se guardará en `nuevo_archivo.txt`. Para finalizar, presiona `Ctrl+D`.
`-cat >> archivo_existente.txt` Similar al comando anterior, escribirás el texto que deseas agregar al final de `archivo_existente.txt` y luego presionas `Ctrl+D` para finalizar.
`-cat -n archivo.txt` Puedes mostrar el contenido de un archivo con los números de línea precediendo cada línea.
`-cat -A archivo1.txt archivo2.txt` : Para mostrar el contenido de varios archivos con nombres de archivo precediendo cada sección, puedes usar la opción `-A` o `--show-all`.
`-cat archivo.txt | grep -v '^$'` : Aunque `cat` no tiene una opción directa para eliminar líneas vacías, puedes combinarlo con otras herramientas como `grep` para lograr esto.

`sort` se utiliza para ordenar líneas de texto en archivos o en la entrada estándar.

Ejemplos:

-sort archivo.txt : Puedes ordenar las líneas de un archivo y mostrar el resultado en la salida estándar.

-sort -r archivo.txt : Por defecto, `sort` organiza las líneas en orden ascendente. Para ordenarlas en orden descendente, usa la opción `-r`.

-sort -k 2 archivo.txt : Puedes ordenar por una columna o campo específico usando la opción `-k` seguida del número de campo.

-sort -k 2 archivo.txt : Esto ordenará las líneas de `archivo.txt` basándose en el segundo campo.

-sort -n archivo.txt : Si necesitas ordenar líneas numéricamente en lugar de alfabéticamente, usa la opción `-n`.

-sort -M archivo.txt : Para ordenar fechas y otros datos con formato específico, usa la opción `-M` para meses o `-t` para especificar delimitadores.

-sort -t ',' -k 3 archivo.txt : Aquí, `-t` especifica la coma como delimitador y `-k 3` indica que se debe ordenar por el tercer campo.

-sort -u archivo.txt : Puedes eliminar líneas duplicadas usando la opción `-u` (único).

-sort archivo.txt | less : Para manejar grandes volúmenes de datos, puedes combinar `sort` con `less` para visualizar el resultado en páginas.

-sort archivo.txt > archivo_ordenado.txt : Puedes redirigir el resultado de la ordenación a un nuevo archivo.

El comando `head` se utiliza para mostrar las primeras líneas de uno o más archivos.

Ejemplos:

-head archivo.txt : Por defecto, `head` muestra las primeras 10 líneas de un archivo.

-head -n 20 archivo.txt : Puedes ajustar la cantidad de líneas que deseas ver usando la opción `-n` seguida del número de líneas.

-head archivo1.txt archivo2.txt : Cuando se proporcionan varios archivos, `head` muestra las primeras líneas de cada archivo, precedidas por el nombre del archivo.

-ls -l | head : `head` puede leer desde la entrada estándar, lo que permite usarlo en combinación con otros comandos mediante tuberías. Esto muestra las primeras 10 líneas de la salida del comando `ls -l`.

-head -c 50 archivo.txt : En lugar de mostrar un número específico de líneas, también puedes mostrar un número específico de bytes usando la opción `-c`.

El comando `wc` se utiliza para contar líneas, palabras y caracteres en uno o más archivos.

Ejemplos:

wc archivo.txt : Este comando muestra el número de líneas, palabras y caracteres en el archivo `archivo.txt`.

wc -l archivo.txt : `-l` Muestra solo el número de líneas :

`wc -w archivo.txt : -w` Muestra solo el número de palabras.

`wc -c archivo.txt : -c` Muestra solo el número de caracteres.

wc -l -w -c archivo.txt

Uso en combinación con otros comandos: `wc` se puede usar en tuberías para contar el contenido de la salida de otros comandos. Por ejemplo, para contar el número de líneas en la salida del comando `ls`:

ls | wc -l

En resumen, el comando `wc` es una herramienta versátil para obtener estadísticas básicas sobre el contenido de archivos o de la salida de otros comandos.

El comando `awk` es una herramienta poderosa en sistemas Unix/Linux utilizada para el procesamiento y análisis de texto. Es especialmente útil para manipular datos organizados en columnas y líneas, y se usa ampliamente en scripts de shell para tareas como la extracción, filtrado y formateo de datos.

awk 'patrón { acción }' archivo

`patrón`: Expresión que determina qué registros deben ser procesados. Si se omite, la acción se aplica a todas las líneas.

`Acción`: Comandos que se ejecutan para cada línea que cumple con el patrón. Si se omite el patrón, se aplican a todas las líneas.

Variables Internas

- **`$0`**: La línea completa.
- **`$1`**, **`$2`**, **...**, **`$n`**: Campos individuales en una línea (el número del campo empieza en 1).
- **`NR`**: Número de registros (líneas) procesados.
- **`NF`**: Número de campos en la línea actual.
- **`FS`**: Separador de campos (por defecto, espacio o tabulación).
- **`OFS`**: Separador de campos de salida (por defecto, espacio).

Conector `&`

El símbolo `&` se utiliza para ejecutar comandos en segundo plano. Cuando pones un `&` al final de un comando, el sistema lo ejecuta en segundo plano, permitiéndote seguir usando la terminal mientras el comando se sigue ejecutando.

sleep 60 &

En este ejemplo, el comando `sleep 60` se ejecutará en segundo plano, lo que significa que puedes seguir usando la terminal para otros comandos mientras `sleep` sigue contando el tiempo en el fondo.

Características:

- **No bloqueante**: La terminal no espera a que el comando termine para que puedas ejecutar otros comandos.
- **Job ID**: Cuando ejecutas un comando con `&`, se te asigna un ID de trabajo (job ID) que puedes usar para controlar el proceso (por ejemplo, para detenerlo o traerlo al primer plano).

Conector `;`

El símbolo `;` se utiliza para ejecutar múltiples comandos en secuencia, uno después del otro, sin importar si el comando anterior tuvo éxito o falló. Cada comando se ejecuta de manera independiente del éxito o fracaso del comando anterior.

comando1 ; comando2 ; comando3

echo "Hola" ; ls -l ; echo "Adiós"

En este ejemplo, primero se ejecuta `echo "Hola"`, luego `ls -l` y, finalmente, `echo "Adiós"`. Cada comando se ejecuta en el orden en que se escriben, y todos se ejecutan independientemente del resultado de los comandos anteriores.

Características:

- **Secuencial**: Los comandos se ejecutan uno tras otro en el orden en que aparecen.
- **Independiente**: No depende del éxito o fracaso de los comandos anteriores.

Diferencias y Usos

- `&`: Usa este conector cuando quieras ejecutar un comando en segundo plano y continuar trabajando en la terminal sin esperar a que termine.
- `;`: Usa este conector cuando quieras ejecutar múltiples comandos secuencialmente, sin importar si los comandos anteriores tienen éxito o no.

Ejemplo de Uso Combinado

Puedes combinar ambos conectores para ejecutar un comando en segundo plano y luego ejecutar otros comandos en la secuencia:

```
comando1 & comando2 ; comando3
```

Aquí, `comando1` se ejecuta en segundo plano, y luego `comando2` se ejecuta en primer plano, seguido por `comando3` después de que `comando2` ha terminado.

En resumen, `&` es útil para tareas en segundo plano que no necesitan tu atención constante, mientras que `;` es útil para ejecutar una serie de comandos de manera secuencial.

El símbolo de **pipe** (`|`) en sistemas Unix y Linux se utiliza para redirigir la salida de un comando como entrada para otro comando. Este mecanismo permite encadenar varios comandos y crear tuberías de procesamiento de datos de manera eficiente.

¿Qué Hace el Pipe (`|`)?

1. **Redirige la Salida a la Entrada**:
    - La salida estándar (stdout) de un comando se pasa como entrada estándar (stdin) a otro comando. Esto permite realizar operaciones complejas mediante la combinación de comandos simples.
2. **Encadenar Comandos**:
    - Puedes usar pipes para combinar varios comandos y procesar datos de manera secuencial. Esto facilita la creación de comandos más poderosos y flexibles.

comando1 | comando2

En este caso, la salida de `comando1` se convierte en la entrada de `comando2`.

Ejemplos Comunes

1. **Buscar Texto en la Salida de Otro Comando**:
    - Puedes usar `grep` para buscar texto específico en la salida de otro comando.

ls -l | grep "archivo"

Aquí, `ls -l` lista los archivos en el directorio, y `grep "archivo"` filtra las líneas que contienen la palabra "archivo".

Contar el Número de Líneas en la Salida de un Comando:

- Usa `wc -l` para contar el número de líneas en la salida de otro comando.

ps aux | wc -l

Esto cuenta el número de líneas en la salida del comando `ps aux`, que muestra información sobre los procesos en ejecución.

Ordenar y Filtrar Resultados:

- Puedes combinar `sort` y `uniq` para ordenar y eliminar duplicados en los resultados.

cat archivo.txt | sort | uniq

Aquí, `cat` muestra el contenido de `archivo.txt`, `sort` ordena las líneas y `uniq` elimina las duplicadas.

ver el Contenido de un Archivo de Log en Tiempo Real:

- Usa `tail` con `-f` para ver el contenido en tiempo real y `grep` para filtrar resultados específicos.

tail -f /var/log/syslog | grep "error"

Esto muestra en tiempo real las líneas que contienen "error" en el archivo de log `/var/log/syslog`.

Redirección de Salida `>`

El operador `>` se usa para redirigir la salida estándar (stdout) de un comando a un archivo. Si el archivo ya existe, se sobrescribirá con la nueva salida.

comando > archivo

Ejemplos

- **Redirigir la salida de un comando a un archivo**:

echo "Hola Mundo" > archivo.txt

Esto crea el archivo `archivo.txt` (o lo sobrescribe si ya existe) con el texto "Hola Mundo".

Guardar la lista de archivos en un directorio en un archivo:

ls -l > listado.txt

Esto guarda la salida del comando `ls -l` en el archivo `listado.txt`.

Redirección de Entrada `<`

El operador `<` se usa para redirigir la entrada estándar (stdin) desde un archivo. En lugar de leer desde la entrada estándar (como el teclado), el comando lee los datos del archivo especificado.

comando < archivo

Ejemplos

- **Leer el contenido de un archivo como entrada para un comando**:

sort < archivo.txt

Esto toma el contenido de `archivo.txt` y lo pasa como entrada al comando `sort`.

- **Usar un archivo de datos como entrada para un script**:

python script.py < datos.txt

Esto pasa el contenido de `datos.txt` como entrada estándar al script de Python `script.py`.

Redirección de Salida y Entrada Simultáneamente

Puedes usar ambos operadores juntos para redirigir la entrada desde un archivo y la salida a otro archivo.

comando < archivo_entrada > archivo_salida

Ejemplos

- **Ordenar un archivo y guardar el resultado en otro archivo**:

sort < archivo_entrada.txt > archivo_salida.txt

Esto toma el contenido de `archivo_entrada.txt`, lo ordena y guarda el resultado en `archivo_salida.txt`.

Redirección Adicional `>>` y `2>`

- `>>`: Redirige la salida y la agrega al final de un archivo en lugar de sobrescribirlo.

echo "Nueva línea" >> archivo.txt

Esto agrega "Nueva línea" al final del archivo `archivo.txt`.

- `2>`: Redirige la salida de errores estándar (stderr) a un archivo.

comando 2> errores.txt

Esto guarda cualquier mensaje de error producido por `comando` en `errores.txt`.

Resumen

- `>`: Redirige la salida estándar a un archivo, sobrescribiendo el archivo si ya existe.
- `<`: Redirige la entrada estándar desde un archivo.
- `>>`: Redirige la salida estándar a un archivo, agregando al final del archivo en lugar de sobrescribir.
- `2>`: Redirige la salida de errores estándar a un archivo.