Módulo 4: Comandos avanzados

4.1. Procesamiento de texto

4.1.1. Comando *grep*: búsqueda avanzada de patrones en archivos.

Introducción

El comando `grep` es una de las herramientas más poderosas y comunes para buscar cadenas de texto dentro de archivos. Su nombre proviene de una antigua expresión en editores de texto que significa **Global Regular Expression Print**. `grep` permite realizar búsquedas simples y complejas utilizando **expresiones regulares**, lo que lo convierte en una herramienta fundamental para filtrar grandes cantidades de texto de manera eficiente.

Uso básico: búsqueda de cadenas

El uso más simple de `grep` es buscar una cadena de texto dentro de un archivo. La sintaxis básica es:

```
grep "patrón" archivo
```

Ejemplo:

```
grep "error" /var/log/syslog
```

Esto buscará la palabra "error" en el archivo `/var/log/syslog`. Si la cadena está presente, se mostrarán todas las líneas que la contienen.

Opciones básicas

- **-i**: Hace que la búsqueda sea **insensible a mayúsculas y minúsculas**.
  - `grep -i "error" archivo.txt`
- **-n**: Muestra el **número de línea** donde se encuentra el patrón.
  - `grep -n "error" archivo.txt`
- **-v**: Muestra todas las líneas que **no** contienen el patrón.
  - `grep -v "error" archivo.txt`
- **-r**: Realiza una búsqueda **recursiva** dentro de todos los archivos de un directorio.
  - `grep -r "error" /var/log`

Expresiones regulares

El verdadero poder de `grep` radica en su capacidad para trabajar con **expresiones regulares**. Estas permiten hacer búsquedas mucho más avanzadas que simples cadenas de texto.

1. Buscar líneas que comiencen con un patrón:
   - `grep "^patrón" archivo.txt`

   El símbolo `^` indica el inicio de la línea.
2. Buscar líneas que terminen con un patrón:
   - `grep "patrón$" archivo.txt`

El símbolo `$` indica el final de la línea.

3. Buscar líneas que contengan un patrón opcional:
   - `grep "opci(on|ión)" archivo.txt`

   Esto buscará "opcion" u "opción".

4. Buscar patrones repetidos:
   - `grep "ho\+" archivo.txt`

   Buscará "ho", "hoo", "hooo", etc.

Modificadores y búsqueda recursiva

1. **Buscar en múltiples archivos**:
   - `grep "patrón" archivo1 archivo2`
2. **Buscar archivos que contienen el patrón**: El flag `-l` muestra solo los nombres de los archivos donde se encuentra el patrón:
   - `grep -l "error" *.log`
3. **Búsqueda recursiva en directorios**: La opción `-r` permite buscar dentro de todos los archivos en un directorio de forma recursiva. Esto es útil cuando no se conoce el archivo exacto donde está el patrón.
   - `grep -r "error" /var/log`
4. **Colorear los resultados**: grep permite resaltar el patrón encontrado en los resultados usando la opción `--color`. Esto es útil cuando se analiza un archivo grande con muchas coincidencias.
   - `grep --color "error" archivo.txt`

Ejemplos avanzados

- **Buscar varias palabras**:
  - `grep -E "palabra1|palabra2" archivo.txt`

  El modificador -E permite usar **expresiones regulares extendidas**, como `|` para indicar "o" lógico.

- **Buscar líneas vacías**:
  - `grep "^$" archivo.txt`

  Esto muestra todas las líneas vacías en el archivo.

Combinación con otros comandos

El comando `grep` es a menudo utilizado en combinación con otros comandos para filtrar la salida de los mismos. Por ejemplo:

```
ps aux | grep "apache"
```

Esto mostrará solo las líneas del comando `ps aux` que contienen la palabra "apache".

4.1.2. Comando *awk*: Procesamiento y análisis de archivos

Introducción

El comando `awk` es una poderosa herramienta de procesamiento de texto y análisis de datos. Su nombre proviene de las iniciales de sus creadores: **Aho, Weinberger y Kernighan**. Este comando permite analizar y manipular texto de forma eficiente, trabajando línea por línea y dividiendo las líneas en campos. Es ampliamente utilizado para tareas que involucran archivos delimitados por espacios, tabulaciones u otros caracteres, como archivos de registro o tablas de datos.

Uso básico de `awk`

La sintaxis básica de `awk` es la siguiente:

```
awk 'condición {acción}' archivo
```

- **condición**: Define qué líneas o registros deben ser procesados.
- **acción**: Lo que se hará con las líneas que cumplen la condición.

Por ejemplo, si queremos imprimir todas las líneas de un archivo:

```
awk '{print}' archivo.txt
```

Esto imprimirá cada línea del archivo tal como está.

Selección de campos

Con `awk` se divide las líneas de un archivo en **campos** (o columnas) usando un delimitador por defecto (el espacio o tabulador). Los campos se acceden mediante **$1**, **$2**, etc., donde **$1** es el primer campo, **$2** es el segundo, y **$0** representa la línea completa. Por ejemplo: imprimir el segundo campo de cada línea de un archivo:

```
awk '{print $2}' archivo.txt
```

Si un archivo contiene líneas como estas:

```
Juan Pérez 25
Ana Gómez 30
```

el comando imprimirá:

```
Pérez
Gómez
```

Separador de campos

Puedes cambiar el delimitador que usa `awk` con la opción **`-F`**. Por ejemplo, si tienes un archivo CSV donde los campos están separados por comas:

```
awk -F ',' '{print $1, $3}' archivo.csv
```

Esto imprimirá el primer y tercer campo de cada línea de un archivo CSV.

Condiciones básicas

El comando `awk` permite aplicar condiciones para realizar acciones solo sobre determinadas líneas o campos. Por ejemplo, puedes imprimir solo las líneas donde el valor de un campo cumple una condición:

```
awk '$3 > 20 {print $1, $2}' archivo.txt
```

Este comando imprimirá las dos primeras columnas de las líneas donde el tercer campo es mayor a 20.

Operaciones con expresiones regulares

Al igual que `grep`, `awk` soporta el uso de expresiones regulares para buscar patrones en un archivo. Puedes usar expresiones regulares dentro de la condición para seleccionar las líneas que contengan una coincidencia. Por ejemplo, imprimir las líneas que contengan la palabra "error":

```
awk '/error/ {print}' archivo.txt
```

También puedes combinar expresiones regulares con otras condiciones:

```
awk '$3 > 20 && /error/ {print $0}' archivo.txt
```

Esto imprimirá las líneas que contengan "error" y cuyo tercer campo sea mayor a 20.

Modificación y transformación de campos

El comando `awk` no solo se limita a leer y filtrar datos, sino que también permite modificar el contenido de los campos. Por ejemplo, puedes sumar valores o concatenar cadenas:

- **Sumar valores**:
  - `awk '{suma = $3 + 5; print $1, suma}' archivo.txt`

  Esto sumará 5 al valor del tercer campo y mostrará el resultado junto al primer campo.

- **Concatenar cadenas**:
  - `awk '{nombre_completo = $1 " " $2; print nombre_completo}' archivo.txt`

  Esto concatenará el primer y segundo campo con un espacio en el medio, creando un nombre completo.

4.1.3. Comando *wc*: Conteo de líneas, palabras y caracteres

Introducción

El comando **`wc`** (word count) cuenta líneas, palabras y caracteres en un archivo de texto. Es útil cuando se necesita un resumen rápido del tamaño o la longitud de un archivo. La sintaxis básica es:

```
wc [opciones] archivo
```

Opciones más comunes:

- **`-l`**: Cuenta solo el número de líneas del archivo.

- `wc -l archivo.txt`

Esto devolverá el número total de líneas en archivo.txt.

- **-w**: Cuenta el número de palabras en el archivo.
  - `wc -w archivo.txt`

Esto muestra la cantidad de palabras en el archivo.

- **-c**: Cuenta el número de bytes o caracteres.
  - `wc -c archivo.txt`

Devuelve el número total de caracteres (incluyendo espacios).

- **-m**: Cuenta el número de caracteres, pero sin considerar bytes multi-octeto (para soportar archivos codificados con UTF-8).
  - wc -m archivo.txt

Devuelve el número total de caracteres.

Un ejemplo combinado es:

```
wc archivo.txt
```

Este comando mostrará el conteo de líneas, palabras y caracteres del archivo:

```
10 50 300 archivo.txt
```

Esto significa que archivo.txt tiene 10 líneas, 50 palabras y 300 caracteres.

4.1.4. Comando *cat*: Concatenación y visualización de archivos

Introducción

El comando `cat` es uno de los comandos más utilizados para visualizar el contenido de un archivo directamente en la terminal. Su nombre proviene de "concatenate" (concatenar), y su principal función es mostrar el contenido de uno o varios archivos o concatenarlos.

Sintaxis básica

La sintaxis básica del comando es:

```
cat archivo1 archivo2 > archivo3
```

Este comando concatena `archivo1` y `archivo2` y escribe el resultado en `archivo3`. Si `archivo3` no existe, `cat` lo crea. Si ya existe, lo sobrescribe.

Visualizar un archivo completo

Para visualizar el contenido de un archivo, presentándolo en la salida estándar, la sintaxis es simplemente:

```
cat archivo.txt
```

Esto mostrará el contenido completo de `archivo.txt` en la terminal.

Concatenar varios archivos:

Si lo que se quiere hacer es concatenar el contenido de dos archivos, en el orden en que se especifican, la sintaxis es:

```
cat archivo1.txt archivo2.txt
```

Esto mostrará en la terminal el contenido de ambos archivos concatenados.

También es posible redirigir la salida del comando a un archivo:

```
cat archivo1.txt archivo2.txt > archivo3.txt
```

El contenido de `archivo1.txt` y `archivo2.txt` será unido y guardado en `archivo3.txt`.

Si de desea añadir contenido a un archivo existente, sin sobrescribir su contenido, se puede usar el operador de redirección `>>` para agregar al final del archivo:

```
cat archivo_nuevo.txt >> archivo_existente.txt
```

Uso especial para crear archivos

Si no se especifica ningún archivo como argumento y se redirige la salida a un archivo, esto permitirá ingresar texto por teclado, y al cortar la ejecución del comando (con la combinación de teclas `Ctrl+C`) se creará el archivo (si no existía) al que se redirige la salida:

```
cat > archivo.txt
```

El comportamiento aquí sería el siguiente: el comando recibe al menos un archivo como argumento; si no se especifica ninguno, toma como archivo la entrada estándar, que es el teclado.

4.1.5. Comandos *less* y *more*: Visualización paginada de archivos

Los comandos `more` y `less` son utilizados para visualizar archivos grandes de manera paginada, lo que permite desplazarse por el contenido sin que toda la información sea mostrada de golpe.

Comando `more`

Este comando muestra el contenido de un archivo, pantalla por pantalla. La sintaxis básica es:

```
more archivo.txt
```

Esto abrirá el archivo en modo de lectura, permitiendo avanzar a través del archivo utilizando el teclado:

- **Espacio**: Avanza una pantalla completa.
- **Enter**: Avanza una línea.
- **b**: Retrocede una pantalla.
- **q**: Sale del modo de lectura.

Comando `less`

Es una versión más avanzada que permite desplazarse tanto hacia adelante como hacia atrás dentro de un archivo, además de permitir búsquedas. La sintaxis básica es:

```
less archivo.txt
```

Dentro de `less`, los controles son:

- **Espacio, b, Page Up / Page Doown**: avanzar o retroceder una página.
- **Flechas arriba/abajo**: moverse línea por línea.
- **G**: ir al final del archivo.
- **g**: ir al principio del archivo.
- **/palabra**: Busca una palabra dentro del archivo.
- **n**: Ir al siguiente resultado de búsqueda.
- **q**: Salir.

Diferencias entre `more` y `less`.

El comando `more` carga el archivo completo en memoria de una sola vez. Esto puede ser un problema si el archivo es muy grande, ya que requiere más memoria y tiempo para procesar archivos extensos. En cambio, `less` carga el archivo **de manera dinámica**, es decir, lee y muestra solo lo que necesita en la pantalla y carga más contenido a medida que te desplazas. Esto lo hace mucho más eficiente al trabajar con archivos grandes, ya que no necesita cargar todo el archivo en memoria al inicio.

Por otro lado, cuando llegas al final del archivo con `more` y sales del comando (presionando la tecla **q**), la pantalla vuelve a la terminal y **limpia el contenido** del archivo que estabas visualizando. Es decir, una vez que sales, ya no verás el contenido que habías estado viendo, sino que regresarás a la línea de comandos limpia. A diferencia de `more`, `less` **no limpia la pantalla** al salir. Esto significa que el contenido del archivo sigue visible en la terminal incluso después de haber salido del comando. El comportamiento de `less` es más útil si quieres consultar el archivo, salir y seguir viendo alguna parte del contenido sin necesidad de volver a ejecutarlo.

4.1.6. Comandos *head* y *tail*: Ver las primeras y últimas líneas de un archivo

Estos comandos se utilizan para obtener una vista rápida de las primeras o últimas líneas de un archivo. Son especialmente útiles cuando trabajas con archivos grandes o quieres extraer información rápidamente sin tener que abrir archivos completos o cargar grandes volúmenes de datos en memoria. También son imprescindibles para la depuración de sistemas y la supervisión en tiempo real de registros del sistema o de aplicaciones.

Comando `head`

El comando `head` muestra las primeras líneas de un archivo de texto. La sintaxis básica es:

```
head -n N archivo.txt
```

La opción `-n N` especifica cuántas líneas quieres mostrar. Si no se indica `N`, muestra las primeras 10 líneas de forma predeterminada. Esto resulta útil cuando se quiere chequear el formato de los datos de un archivo CSV, o bien las primeras líneas de un archivo de configuración de un servicio.

Comando `tail`

Muestra las últimas líneas de un archivo de texto. Su sintaxis es:

```
tail -n N archivo.txt
```

Con la opción `-n N` se muestra las últimas N líneas del archivo. Si no se especifica N, se mostrarán las últimas 10 líneas por defecto. Una opción destacable es la `-f`. Esta permite monitorear en tiempo real un archivo de texto, mostrando las nuevas líneas que se agregan. Esto es especialmente útil para seguir logs del sistema o registros de aplicaciones en tiempo real.

Si queremos extraer una parte intermedia del archivo, por ejemplo, las líneas 11 a la 20, se pueden combinar los comandos `head` y `tail` de la siguiente manera:

```
head -n 20 archivo.txt | tail -n 10
```

Aquí, `head` obtiene las primeras 20 líneas, y luego `tail` muestra solo las últimas 10 de esas 20, que corresponden a las líneas 11 a 20 del archivo.

4.2. Manipulación de archivos

4.2.1. Comando *tar*: Creación y gestión de archivos empaquetados

El comando `tar` se utiliza para empaquetar archivos o desempaquetarlos. Aunque por sí mismo no comprime, suele usarse en combinación con comandos de compresión como `gzip` o `bzip2`. El archivo resultante de `tar` suele tener la extensión .tar. La sintaxis básica del comando es:

```
tar [opciones] archivo.tar [archivos o directorios]
```

Algunas de las opciones más comunes son:

- **-c**: Crea un nuevo archivo .tar.
- **-x**: Extrae archivos de un archivo .tar.
- **-v**: Muestra el progreso del comando.
- **-f**: Especifica el nombre del archivo .tar.

Por ejemplo, para crear un archivo .tar, la línea de comando será la siguiente:

```
tar -cvf archivo.tar archivo1 archivo2
```

Este comando crea un archivo llamado `archivo.tar` que contiene `archivo1` y `archivo2`. Si queremos extraer el contenido de un archivo .tar:

```
tar -xvf archivo.tar
```

Esto extrae el contenido del archivo archivo.tar en el directorio actual.

4.2.2. Comandos *gzip* y *bzip2*: Compresión y descompresión

Tanto `gzip` y `bzip2` son comandos que comprimen archivos. La diferencia principal entre ambos es que `bzip2` suele ofrecer una mayor compresión a cambio de un proceso más lento en comparación con `gzip`.

Comando `gzip`

La sintaxis de este comando es:

```
gzip archivo
```

Este comando comprime archivo y crea uno nuevo con la extensión `.gz`. El archivo original es reemplazado por el archivo comprimido. Para descomprimir un archivo .gz:

```
gzip -d archivo.gz
```

Tanto la compresión como la descompresión se realizan en el directorio de trabajo actual.

Comando `bzip2`

Este comando tiene como sintaxis básica:

```
bzip2 archivo
```

Este comando comprime archivo y crea uno nuevo con la extensión `.bz2`. Similar a `gzip`, el archivo original es reemplazado por el comprimido. Para descomprimir un archivo `.bz2`:

```
bzip2 -d archivo.bz2
```

Combinación de los comandos con `tar`

El comando `tar` se utiliza para empaquetar varios archivos y directorios en un solo archivo, preservando la estructura y los metadatos, pero sin comprimirlos. Para reducir el tamaño de estos archivos empaquetados, es común combinar `tar` con herramientas de compresión como `gzip` y `bzip2`, que aplican distintos algoritmos para disminuir el espacio que ocupan. Entonces, en el comando `tar`, se utilizan las siguientes opciones adicionales:

- **-z**: Usa `gzip` para comprimir el archivo.
- **-j**: Usa `bzip2` para comprimir el archivo.

Por ejemplo, para crear un archivo comprimido con `gzip` utilizaremos:

```
tar -czvf archivo.tar.gz archivo1 archivo2
```

Esta combinación es extremadamente útil para realizar copias de seguridad, transferir archivos de gran tamaño o archivar datos de manera eficiente, logrando tanto la organización de los archivos como su compresión en un solo paso.

4.2.3. Comando *touch*: Crear o actualizar la fecha de modificación de archivos

El comando `touch` tiene dos funciones principales: crear archivos vacíos y actualizar las marcas de tiempo de los archivos existentes (fecha de modificación y acceso). La sintaxis básica es:

```
touch archivo
```

Esto crea un archivo vacío con el nombre especificado si no existe. Si el archivo ya existe, actualiza su marca de tiempo (fecha de modificación y acceso) a la fecha y hora actuales, sin modificar el contenido del archivo.

También es posible cambiar la fecha de modificación por una específica. Este comando cambia la fecha de modificación de archivo.txt a las 12:00 del 1 de enero de 2024:

```
touch -t 202401011200 archivo.txt
```

Cambiar la marca de tiempo de un archivo con el comando `touch` puede sonar "turbio" a primera vista, pero en realidad tiene usos legítimos y prácticos en diversas situaciones, especialmente en administración de sistemas, desarrollo de software, y automatización de tareas. Algunos casos en los que es útil cambiar la marca de tiempo de un archivo:

- Fuerza la recompilación de archivos en proyectos de software: En muchos entornos de desarrollo, los compiladores o herramientas de construcción (como `make`) dependen de la marca de tiempo de los archivos para decidir si necesitan recompilarse. Al modificar la marca de tiempo de un archivo fuente con `touch`, puedes forzar la recompilación de un archivo o incluso de todo el proyecto sin modificar su contenido.
- Actualización de la fecha de acceso o modificación: A veces resulta necesario actualizar la marca de tiempo de un archivo para mantener coherencia en sistemas que dependen de estos datos. Por ejemplo, en scripts de administración de backups o sincronización de archivos, modificar la marca de tiempo puede evitar que un archivo sea sobrescrito innecesariamente.
- Crear archivos vacíos como marcadores: `touch` también se usa para crear archivos vacíos sin contenido. Esto puede ser útil para marcar eventos o como "flags" en scripts. Por ejemplo, un archivo vacío llamado `done.flag` podría indicar que una tarea se ha completado.
- Simular fechas para pruebas: Si estás desarrollando o probando software que gestiona archivos en función de sus marcas de tiempo (como software de respaldo o sincronización), puedes usar `touch` para simular archivos creados en el pasado o en fechas específicas para asegurarte de que el sistema se comporte correctamente.
- Administración de archivos temporales: Algunos procesos generan archivos temporales que son monitoreados o eliminados según su antigüedad. Al actualizar la marca de tiempo con `touch`, puedes mantener esos archivos "frescos" para evitar que sean eliminados prematuramente.

4.2.4. Comando *find*: Búsqueda de archivos en el sistema

El comando `find` es extremadamente poderoso para localizar archivos y directorios en un sistema de archivos, permitiendo realizar búsquedas basadas en diversos criterios como el nombre del archivo, la fecha de modificación, el tamaño, y más. Su sintaxis básica es:

```
find [ruta] [opciones] [criterio]
```

- **[ruta]**: Especifica el directorio en el que se iniciará la búsqueda. Si no se especifica, find comenzará en el directorio actual.
- **[opciones]**: Permite especificar opciones como profundidad de búsqueda, etc.

- **[criterio]**: El criterio utilizado para buscar, como el nombre del archivo, la fecha de modificación, etc.

Ejemplos de búsqueda:

- Buscar archivos por nombre:

```
find /ruta -name "archivo.txt"
```

  Esto buscará archivos llamados `archivo.txt` en el directorio `/ruta` y sus subdirectorios.

- Buscar archivos por extensión:

```
find /ruta -name "*.txt"
```

  Esto buscará todos los archivos con la extensión `.txt` en el directorio `/ruta`.

- Buscar archivos modificados en los últimos 7 días:

```
find /ruta -mtime -7
```

  Esto mostrará los archivos que fueron modificados en los últimos 7 días.

- Buscar archivos mayores a un tamaño específico:

```
find /ruta -size +50M
```

  Este comando busca archivos que tengan más de 50 MB en el directorio especificado.

- Ejecutar una acción sobre los archivos encontrados: `find` también permite ejecutar comandos sobre los archivos que encuentra. Por ejemplo, para eliminar todos los archivos `.tmp` en el directorio `/tmp`:

```
find /tmp -name "*.tmp" -exec rm {} \;
```

  El comando `-exec` ejecuta la acción (`rm` en este caso) sobre cada archivo encontrado. La secuencia `{}` se reemplaza por el nombre de cada archivo y `\;` indica el fin del comando.

4.3. Gestión de procesos

4.3.1. Comando *ps*: Visualización de procesos activos

El comando `ps` es utilizado para mostrar información sobre los procesos que están corriendo en el sistema. Muestra detalles como el ID del proceso (PID), el usuario que lo está ejecutando, el uso de CPU, y otros datos útiles. La sintaxis básica es:

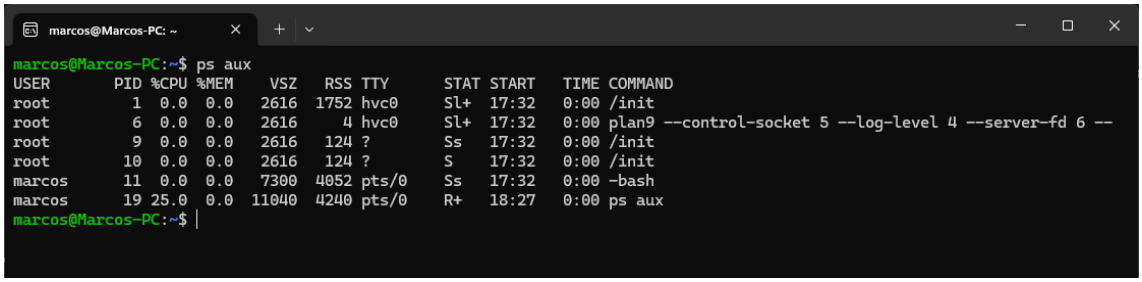
```
ps [opciones]
```

Por ejemplo:

```
ps aux
```

- **a**: Muestra los procesos de todos los usuarios.
- **u**: Muestra los procesos con un formato de usuario.

- **x**: Incluye procesos que no están conectados a una terminal.



Este comando proporciona una lista detallada de todos los procesos que están corriendo en el sistema, independientemente de quién los haya iniciado, mostrando información útil como el PID, el porcentaje de uso de CPU, memoria, y más.

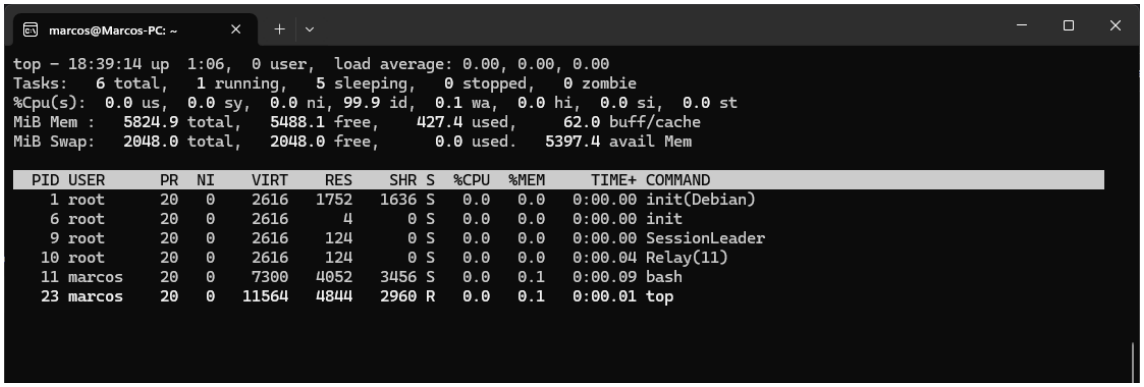
Si solo se quiere filtrar por un usuario en particular, se utiliza la opción `-u`, y si se quiere filtrar por el PID, entonces se utiliza la opción `-p`:

```
ps -p 1234
```

Esto muestra únicamente el proceso con PID 1234.

4.3.2. Comando *top*: Monitorización en tiempo real de procesos

El comando `top` es una herramienta interactiva que permite ver en tiempo real los procesos que están corriendo en el sistema, así como el uso de recursos (CPU, memoria, etc.). Su sintaxis es simplemente: `top`. Al ejecutarlo, se abre una interfaz en la terminal que se actualiza constantemente con información sobre los procesos en ejecución.



Para salir de la interfaz, simplemente se pulsa la tecla *q*.

4.4. Gestión avanzada de procesos

4.4.1. Comando *kill*: Finalización de procesos

El comando `kill` se utiliza para enviar señales a un proceso en ejecución, usando su **PID**. Aunque el nombre del comando sugiere que su propósito es "matar" o finalizar un proceso, en realidad es mucho más versátil, ya que permite enviar diferentes tipos de señales a los procesos, no solo la de finalización. La sintaxis es:

```
kill [señal] PID
```

En Unix/Linux hay una serie de señales predefinidas que se pueden enviar a los procesos. Las más comunes son:

1. **SIGTERM (Señal 15) – Terminar el proceso de forma educada**: Esta es la señal predeterminada que se envía cuando ejecutas `kill` sin especificar ninguna señal. SIGTERM le pide al proceso que termine de manera ordenada, permitiéndole liberar recursos o guardar datos si es necesario.
2. **SIGKILL (Señal 9) – Matar el proceso inmediatamente**: A veces, un proceso puede no responder a SIGTERM (por ejemplo, si está atascado o en un estado no manejable). En estos casos, puedes usar SIGKILL, que fuerza la finalización inmediata del proceso sin darle oportunidad de limpiar recursos. Es más drástico, pero efectivo.
3. **SIGHUP (Señal 1) – Reiniciar o recargar la configuración del proceso**: La señal SIGHUP es útil para decirle a un proceso que recargue su configuración sin detenerse. Por ejemplo, algunos servicios como servidores web o de bases de datos pueden recibir esta señal para recargar sus archivos de configuración sin tener que ser reiniciados por completo.
4. **SIGSTOP (Señal 19) – Pausar un proceso**: SIGSTOP detiene (suspende) un proceso en su lugar sin finalizarlo. Este proceso no puede hacer nada mientras esté en estado suspendido, pero puede ser reanudado con SIGCONT.
5. **SIGCONT (Señal 18) – Reanudar un proceso suspendido**: SIGCONT permite reanudar un proceso que fue pausado con SIGSTOP. Este comando se usa, por ejemplo, si has suspendido temporalmente un proceso que deseas retomar más tarde.

4.4.2. Comando *jobs*: Visualización de procesos en segundo plano o suspendidos

El comando `jobs` muestra una lista de procesos que se están ejecutando en segundo plano o que han sido suspendidos dentro de la terminal actual. La sintaxis básica es:

```
jobs [opciones]
```

Algunas opciones útiles son:

- **-p:** muestra únicamente los PID.
- **-r:** limita la salida a sólo los procesos en ejecución.
- **-s:** limita la salida a sólo los procesos detenidos.

4.4.3. Comandos *fg* y *bg*: Control de procesos en segundo y primer plano

Cuando ejecutas un comando y lo suspendes con **`Ctrl+Z`**, puedes enviarlo al segundo plano con `bg` (background) o traerlo de vuelta al primer plano con `fg` (foreground).

Por ejemplo, supongamos que tenemos dos procesos en segundo plano:

Con el comando `fg %1` traeremos el trabajo con el número `[1]` al primer plano:

Y para enviarlo de nuevo al segundo plano, entonces ejecutamos `bg %1`:

En todos los casos `%N` representa el número de trabajo en segundo plano, que es el que está encerrado entre corchetes cuando los visualizamos con el comando `jobs`.

4.5. Editor de texto *nano*

Introducción

El comando `nano` es un editor de texto ligero y fácil de usar que funciona directamente en la terminal. Es una alternativa más simple y amigable al editor `vi` o `vim`, y está orientado a usuarios que necesitan editar archivos rápidamente sin una curva de aprendizaje pronunciada. Aunque nano es básico en comparación con otros editores, ofrece todas las funciones esenciales para editar archivos de texto desde la terminal.

Para abrir un archivo con nano, la sintaxis básica es:

```
nano archivo.txt
```

Si el archivo no existe, nano creará un archivo vacío con el nombre especificado. Si el archivo existe, su contenido será mostrado para editar.

Interfaz de *nano*

Al abrir nano, verás una pantalla dividida en varias partes:

1. **Área de edición**: Aquí puedes ver y modificar el contenido del archivo. Es posible navegar por el texto tanto con las flechas, como utilizar atajos tales como Ctrl+Flechas para posicionarte al comienzo de cada palabra, o utilizar las teclas Inicio y Fin para ubicarte al comienzo o fin de una línea. Esta es una diferencia enorme respecto del editor `vi`, ya que éste último utiliza comandos para posicionarte dentro del texto.
2. **Línea de estado**: Muestra información sobre el archivo actual (nombre del archivo, número de líneas, etc.).
3. **Comandos de control**: Al final de la pantalla verás una lista de comandos rápidos disponibles. Los comandos se ejecutan mediante combinaciones de teclas que incluyen la tecla **Ctrl**. Por ejemplo, **Ctrl + O** se usa para guardar, y **Ctrl + X** para salir.

Funciones principales de *nano*

Crear y editar archivos

Puedes escribir directamente en el área de edición para modificar el archivo de texto. Si has creado un archivo vacío, simplemente comienza a escribir.

Guardar cambios

Para guardar el archivo, utiliza el siguiente comando:

- **Ctrl + O**: Guarda los cambios en el archivo. Al presionar esta combinación, nano te pedirá que confirmes el nombre del archivo. Presiona **Enter** para guardar los cambios.

Salir de `nano`

Una vez que termines de editar el archivo, puedes salir del editor con:

- **Ctrl + X**: Salir de nano. Si has hecho cambios, nano te preguntará si quieres guardarlos antes de salir.

Buscar texto

Puedes buscar texto dentro del archivo utilizando la combinación:

- **Ctrl + W**: Inicia una búsqueda de texto. Ingresa la cadena de búsqueda y presiona **Enter**. Para buscar la siguiente aparición de la cadena, puedes presionar **Ctrl + W** nuevamente y luego **Enter**.

Cortar, copiar y pegar texto

- **Ctrl + K**: Corta una línea de texto.
- **Ctrl + U**: Pega la línea cortada o copiada.

- **Ctrl + ^**: Marca un bloque de texto para cortar o copiar. Coloca el cursor en el inicio del bloque y presiona **Ctrl + ^** para marcar el punto de inicio. Luego desplázate al final del bloque y usa **Ctrl + K** para cortar o **Ctrl + U** para pegar.

Navegación

Puedes desplazarte por el archivo con las teclas de dirección, pero también tienes las siguientes combinaciones útiles:

- **Ctrl + A**: Mueve el cursor al inicio de la línea.
- **Ctrl + E**: Mueve el cursor al final de la línea.
- **Ctrl + Y**: Desplaza una pantalla hacia arriba.
- **Ctrl + V**: Desplaza una pantalla hacia abajo.

Justificar texto

Para justificar un párrafo o el texto que estás escribiendo:

- **Ctrl + J**: Justifica el texto.

Otras características de *nano*

Abrir múltiples archivos

El comando nano también permite abrir varios archivos al mismo tiempo. Puedes navegar entre ellos con **Ctrl + X**, luego **N** o **Y** para guardar y cambiar entre los archivos.

Mostrar números de línea

Si necesitas saber en qué línea te encuentras, puedes iniciar `nano` con la opción `-c`:

```
nano -c archivo.txt
```

Esto habilitará la visualización de números de línea en la interfaz.

Salir sin guardar

Si accidentalmente abres un archivo o haces cambios que no deseas guardar, puedes salir sin guardar utilizando:

- **Ctrl + X**, luego presiona **N** cuando te pregunte si deseas guardar los cambios.