

# Comandos

Aprendiendo a usar comandos de Linux.

- Tail
- Instalar SSH
- Borrar archivos
- Redireccionar el resultado de un comando a un archivo.
- Manual
  - Comando grep
  - Comando nano
  - Comando sed
  - Comando for
- Linux+
  - Gestión de archivos y directorios

# Tail

En esta guía, aprenderás cómo usar el comando tail. Usar tail es una manera sencilla de mostrar los finales de archivos, por ejemplo, al analizar registros y otros archivos de texto que cambian con el tiempo. También puede combinarse con otras herramientas para un monitoreo selectivo y en tiempo real. Al realizar tareas administrativas en tu Linode, tail es una de las herramientas más útiles disponibles.

- Ingresa el comando tail, seguido por el archivo que te gustaría ver:

```
tail /var/quest/kace/user/KAgent.log
```

Esto imprimirá las últimas diez líneas del archivo /var/log/auth.log en la salida de tu terminal.

- Para cambiar el número de líneas mostradas, usa la opción -n:

```
tail -n 50 /var/quest/kace/user/KAgent.log
```

En este ejemplo, se mostrarán las últimas 50 líneas, pero puedes modificar este número para mostrar tantas o tan pocas líneas como necesites.

- Para mostrar una salida en tiempo real, de un archivo que está cambiando, usa las opciones -f o --follow:

```
tail -f /var/quest/kace/user/KAgent.log
```

Esto imprimirá el final del archivo en tu pantalla y lo actualizará a medida que el archivo cambie. Por ejemplo, puedes usar esta opción con /var/log/auth.log (en sistemas Debian y Ubuntu) para mostrar tu registro de acceso en tiempo real. Esto se ejecutará como un proceso en primer plano, así que para cancelarlo, presiona CTRL+C.

- Tail incluso puede combinarse con otras herramientas como grep para filtrar los resultados:

```
tail /var/quest/kace/user/KAgent.log | grep 127.0.0.1
```

Este comando buscaría las últimas diez líneas de tu registro de acceso y solo mostraría aquellas que contienen la dirección IP 198.51.100.1. También puedes aplicar opciones a tail para mostrar más o menos líneas, ver los resultados filtrados en tiempo real y más.

# Instalar SSH

Para instalar el servidor SSH en Ubuntu, puedes usar el siguiente comando en la terminal:

```
sudo apt update  
sudo apt install openssh-server
```

Esto instalará el servidor SSH y lo iniciará automáticamente. Puedes verificar si el servicio está funcionando correctamente con el siguiente comando:

```
sudo systemctl status ssh
```

Si necesitas habilitar o deshabilitar el servicio de SSH para que se inicie automáticamente al arrancar el sistema, puedes usar los siguientes comandos:

- Para habilitar:  

```
sudo systemctl enable ssh
```
- Para deshabilitar:  

```
sudo systemctl disable ssh
```

Una vez instalado, puedes conectarte a tu máquina desde otro equipo mediante un cliente SSH, usando el comando `ssh usuario@dirección_ip` desde la terminal del otro equipo.

# Borrar archivos

## Borrar archivos duplicados

Digamos que en una carpeta hay varios archivos duplicados, en mi caso estos archivos fueron creados con "(1)" al final del nombre. Y como es comun, eso siga una secuencia.

Se puede usar el comando `find` combinado con `rm` para eliminar archivos que contengan "(1)" en su nombre. Aquí está el comando para eliminar esos archivos:

```
find /home/user/directorio-con-archivos -type f -name "*\\(1\\)*" -exec rm {} \\;
```

- `find /home/user/directorio-con-archivos`: Busca en el directorio especificado.
- `-type f`: Asegura que solo se encuentren archivos (no directorios).
- `-name "*\\(1\\)*"`: Busca archivos que tengan "(1)" en su nombre. Los paréntesis se escapan usando `\\`.
- `-exec rm {} \\`: Ejecuta el comando `rm` (eliminar) en cada archivo encontrado (`{}` representa el archivo encontrado).

## Borrar archivos excepto...

Para eliminar todo excepto el archivo ejemplo.txt en un directorio, puede usar el siguiente comando en cualquier sistema Linux:

```
find /$path/to/files/ ! -name 'ejemplo.txt' -type f -delete
```

### Explicación:

- `find`: comando usado para buscar archivos y directorios.
- `/$path/to/files/`: la ruta donde se aplica la búsqueda.
- `! -name 'ejemplo.txt'`: excluye el archivo llamado ejemplo.txt.
- `-type f`: solo afecta archivos (no directorios).
- `-delete`: elimina los elementos encontrados.

# Borrar archivos subdirectorios y su contenido

Para eliminar todo (archivos y carpetas) excepto gitlab.rb, puede usar:

```
find /etc/gitlab/ ! -name 'ejemplo.txt' ! -path /$path/to/files/ -delete
```

⚠ **Tenga mucho cuidado** al usar comandos con -delete. Siempre es recomendable hacer primero una prueba sin eliminar nada, para revisar qué se va a borrar:

```
find /$path/to/files/ ! -name 'ejemplo.txt' -type f
```

# Redireccionar el resultado de un comando a un archivo.

En Linux, la **redirección de archivos** es una forma de controlar hacia dónde van los datos de entrada y salida al ejecutar comandos. Puedes redirigir la entrada estándar (stdin), la salida estándar (stdout) y los errores estándar (stderr) hacia o desde archivos u otros comandos. Esto permite manejar flujos de datos de manera eficiente, como guardar la salida en un archivo o leer la entrada desde un archivo.

## Tipos de redirección:

1. **Salida estándar (stdout)** (`>` y `>>`)
2. **Entrada estándar (stdin)** (`<`)
3. **Error estándar (stderr)** (`2>`, `2>>`, y `2>&1`)

### 1. Redirección de la salida estándar (stdout)

Por defecto, la salida de un comando se muestra en la terminal. Puedes redirigir esta salida a un archivo.

- **Sobrescribir archivo** (`>`): Redirige la salida estándar a un archivo, sobrescribiendo el contenido del archivo si ya existe.

```
comando > archivo.txt
```

Ejemplo:

```
echo "Hola, mundo" > salida.txt
```

Esto escribe "Hola, mundo" en el archivo `salida.txt`, reemplazando su contenido si ya existe.

- **Agregar al archivo** (`>>`): Redirige la salida estándar a un archivo, pero en lugar de sobrescribir, **agrega** el contenido al final del archivo si ya existe.

```
comando >> archivo.txt
```

Ejemplo:

```
echo "Texto adicional" >> salida.txt
```

Esto agrega "Texto adicional" al final del archivo `salida.txt` sin sobrescribir su contenido.

## 2. Redirección de la entrada estándar (stdin)

Por defecto, muchos comandos toman entrada desde el teclado. Puedes redirigir la entrada para que venga de un archivo en lugar del teclado.

- **Redirigir la entrada desde un archivo (`<`):** Redirige la entrada estándar desde un archivo.

```
comando < archivo.txt
```

Ejemplo:

```
cat < archivo.txt
```

Esto usa el contenido de `archivo.txt` como entrada para el comando `cat`.

## 3. Redirección de errores estándar (stderr)

Los errores producidos por los comandos suelen mostrarse en la terminal. Puedes redirigir estos mensajes de error a un archivo.

- **Redirigir los errores (`2>`):** Redirige la salida de errores a un archivo, sobrescribiendo el archivo si ya existe.

```
comando 2> errores.txt
```

Ejemplo:

```
ls archivo_inexistente 2> errores.txt
```

Este comando enviará el mensaje de error (porque el archivo no existe) a `errores.txt`.

- **Agregar los errores (`2>>`):** Redirige los errores estándar a un archivo y agrega el contenido en lugar de sobrescribirlo.

```
comando 2>> errores.txt
```

- **Redirigir stdout y stderr al mismo archivo (`&>` o `2>&1`):** Redirige tanto la salida estándar como los errores estándar al mismo archivo.

```
comando > archivo.txt 2>&1
```

Ejemplo:

```
ls archivo_existente archivo_inexistente > salida.txt 2>&1
```

Esto enviará tanto la salida (listado del archivo existente) como el mensaje de error al archivo `salida.txt`.

## 4. Pipes (`|`)

Puedes usar un **pipe** (`|`) para enviar la salida de un comando como entrada a otro comando. Es una forma de redirección útil para encadenar comandos.

```
comando1 | comando2
```

Ejemplo: ▣

```
ls | grep "archivo"
```

Esto toma la salida de `ls` (listar archivos) y la pasa como entrada al comando `grep` para filtrar y mostrar solo archivos que coinciden con "archivo".

# Ejemplos comunes de redirección de archivos

- **Redirigir salida a un archivo:**

```
echo "Hola, mundo" > hola.txt
```

- **Agregar salida a un archivo:**

```
echo "Texto adicional" >> hola.txt
```

- **Redirigir stdout y stderr al mismo archivo:**



```
comando > salida.txt 2>&1
```

- **Redirigir stderr a un archivo:**

```
comando 2> errores.txt
```

- **Usar un archivo como entrada para un comando:**

```
cat < entrada.txt
```

## Resumen de símbolos:

- `>`: Redirigir stdout a un archivo (sobrescribe).
- `>>`: Redirigir stdout a un archivo (agregar).
- `<`: Redirigir stdin desde un archivo.
- `2>`: Redirigir stderr a un archivo (sobrescribe).
- `2>>`: Redirigir stderr a un archivo (agregar).
- `2>&1`: Redirigir stderr al mismo lugar que stdout.
- `|`: Encadenar la salida de un comando como entrada a otro.

La redirección de archivos en Linux es una herramienta muy poderosa para gestionar la entrada y salida de datos, especialmente en la automatización de scripts o manejo de grandes volúmenes de información.

# Manual

# Comando grep

El comando `grep` es una herramienta muy utilizada en sistemas Linux para buscar cadenas de texto o patrones específicos dentro de archivos o salidas de otros comandos. Su función principal es filtrar líneas que coinciden con el patrón de búsqueda proporcionado.

## Comando ejemplo:

```
grep tcp /etc/services | awk '{print $1}' | sort | less
```

## Desglose:

1. `grep tcp /etc/services` :
  - `grep` busca líneas que contienen el texto `tcp` en el archivo `/etc/services`.
  - El archivo `/etc/services` es una lista de servicios de red y sus puertos asociados, incluyendo si usan **TCP** o **UDP**.
  - Esto filtra todas las líneas donde aparece "tcp", que corresponde a servicios que usan el protocolo TCP.
2. `awk '{print $1}'` :
  - `awk` es una herramienta para procesar texto. En este caso, toma la salida de `grep` y extrae la **primera columna** (campo), que en el archivo `/etc/services` es el **nombre del servicio**.
  - Por ejemplo, si una línea contiene `http 80/tcp`, el `awk '{print $1}'` extraerá solo `http`.
3. `sort` :
  - `sort` ordena alfabéticamente los nombres de los servicios extraídos por `awk`. Esto organiza los resultados para que se muestren en orden.
4. `less` :
  - `less` es un paginador que permite visualizar la salida de manera interactiva. Como la lista de servicios puede ser larga, `less` facilita la navegación por la salida paginada (arriba/abajo con las teclas de flecha o espacio).

# Comando nano

## Abrir un archivo:

Para abrir un archivo en Nano, escribe `nano -u nombre_del_archivo.txt` en la terminal. Si el archivo no existe, Nano lo creará cuando lo guardes.

## Comandos básicos:

- **Guardar:** Presiona **Ctrl + O** para guardar los cambios.
- **Salir:** Presiona **Ctrl + X** para salir de Nano.
- **Ayuda:** Presiona **Ctrl + G** para abrir la guía de ayuda.

## Editar texto:

- **Cortar:** Usa **Ctrl + K** para cortar texto (piensa en "K" de "cut").
- **Pegar:** Usa **Ctrl + U** para pegar texto (piensa en "U" de "uncut").
- **Copiar:** Marca el texto con **Ctrl + 6**, luego presiona **Alt + 6** para copiarlo.

## Deshacer/Rehacer:

Si iniciaste Nano con la opción `-u`, puedes deshacer con **Alt + U** y rehacer con **Alt + E**.

## Buscar y reemplazar:

- **Buscar:** Presiona **Ctrl + W** (piensa en "Where") para buscar texto.
- **Reemplazar:** Presiona **Alt + R** para buscar y reemplazar texto.

## Números de línea:

Presiona **Alt + C** para activar o desactivar los números de línea.

# Comando sed

`sed` es un editor de flujo utilizado para editar texto a medida que se procesa. Aquí están sus usos principales:

- **Print** (Imprimir): Muestra la salida basada en un patrón.
- **Delete** (Eliminar): Borra el texto que coincide con un patrón.
- **Substitute** (Sustituir): Reemplaza un patrón con otro.

Puede manejar expresiones regulares básicas y extendidas, lo que lo convierte en una herramienta poderosa para la manipulación de texto en Linux.

Considere el archivo anexo.

A estos numero de telefonos, podemos agregarles codigo de pais, parentesis y guion para reformarlo apropiadamente.

Si `telefonos.txt` tiene una línea como: `1123456789` después de ejecutar el comando, se transformaría en: `+55(11)2345-6789`

```
sed -E 's/([0-9]{2})([0-9]{4})([0-9]{4})/+55(\1)\2-\3/' telefonos.txt
```

Vamos a desglosarlo:

1. **`sed -E`**:
  - La opción `-E` habilita expresiones regulares extendidas, lo que permite patrones de coincidencia más potentes sin la necesidad de escapar caracteres como `+` o `?`.
2. **`s/([0-9]{2})([0-9]{4})([0-9]{4})/`**:
  - La estructura `s/.../.../` es la sintaxis para sustitución en `sed`, donde el patrón a la izquierda es reemplazado por el formato especificado a la derecha.
  - **`([0-9]{2})`**: Coincide con los dos primeros dígitos, que representan el código de área del estado.
  - **`([0-9]{4})`**: Coincide con los siguientes cuatro dígitos, que representan el prefijo.
  - **`([0-9]{4})`**: Coincide con los últimos cuatro dígitos, que representan el número final.

Cada `([0-9]{...})` captura el grupo de números correspondiente (1, 2 y 3) para usarlos en la parte de reemplazo. (esto es secuencial y por lo que vi no se puede reemplazar)
3. **Patrón de reemplazo `+55(\1)\2-\3/`**:
  - **`+55`**: Agrega el prefijo de código de país al inicio.
  - **`(\1)`**: Coloca el primer grupo (el código de área) entre paréntesis.
  - **`\2`**: Inserta el segundo grupo (el prefijo) tal como está.

- `-`: Coloca un guion entre el segundo y el tercer grupo.
- `\3`: Inserta el tercer grupo (el número final).

4. `telefonos.txt`:

- Es el archivo de entrada que contiene los números de teléfono. `sed` aplica el formato especificado a cada línea del archivo.

# Comando for

Estaba bajando muchos archivos .wav con su licencia .pdf. Y estos los estaba guardando en una carpeta.

Resulta que estos archivos viene con un nombre que usa esta expresion "

Eso es muy incomodo e innecesariamente repetitivo, asi que usando el comando `for` pude renombrar cientos de archivos para que solo contengan la descripcion y borrar el resto.

```
for file in *.wav *.pdf; do
    if [[ "$file" =~ ^[0-9]{8}__ ]]; then
        mv "$file" "${file:10}"
    fi
done
```

## Explicación

- `for file in *.wav *.pdf; do` : Itera sobre todos los archivos `.wav` y `.pdf` en el directorio actual.
- `if [[ "$file" =~ ^[0-9]{8}__ ]]; then` : Comprueba si el nombre del archivo comienza con 8 dígitos seguidos de `__`.
- `mv "$file" "${file:10}"` : Usa `mv` para renombrar el archivo, quitando los primeros 10 caracteres ( `${file:10}` ).

# Linux+



# Gestión de archivos y directorios

Desde siempre que existió Linux se ha tenido que interactuar con el Shell. Es una herramienta que nos permite administrar archivos, texto, programas, procesos entre otras.

Y una forma fácil podríamos decir que todo en Linux es un archivo. Inclusive el Flash Drive que conectemos en la computadora es un archivo. Es cierto que profundiza más allá de eso pero por el momento podemos quedarnos con ese concepto.

Hay miles de comandos que podemos utilizar y la combinación con cada programa incrementa exponencialmente la cantidad de variaciones que podemos hacer. Pero para efectos de nuestro trabajo diario como administradores de un sistema podríamos reducirlo a unos pocos cientos.

## [Administrando archivos y directorios]

Los archivos en un sistema Linux son guardados como una única estructura.

Normalmente lo primero que se aprende cuando trabajamos con Linux es navegar entre los directorios, y manejar archivos.

Así que para efectos de este video podemos comenzar por ahí:

```
pwd
```

```
ls
```

```
ls --help
```

```
❏ syntax=
```

```
❏ ls [OPTION]... [FILE]...
```

entre corchetes se llaman argumentos. [OPTION] significa que es opcional

[FILE] significa que podemos un directorio al comando o al archivo para ver la metadata.

```
ls -l (organizado ascendentemente)
```

```
ls -lr (organizado descendentemente)
```

```
ls -lh vs ll (is an alias)
```

```
man ls
```

Un alias es una representación corta de un comando que usamos muy seguido, por lo que tiene sentido para simplificar las cosas al máximo.

[vista grafica]

Si estamos trabajando con directorio los archivos ayuda mucho sobre las cosas gráficamente. Si no tenemos una interfase gráfica, todavía tenemos la oportunidad de ver las cosas en una forma pseudo gráfica utilizando el comando tree (tres es la traducción en inglés para árbol)

Eso significa que de una forma ramificada podemos ver los directorios y archivos dentro de ellos.

```
tree
```

[espacios en los nombres de un directorio]

En Linux el uso de la barra invertida se debe a que Shell utiliza los espacios como separadores entre argumentos de un comando.

Para que Shell interprete correctamente el nombre de un archivo o directorio con espacios como un solo argumento, es necesario identificar esos espacios con el símbolo `(\)`

```
cd Directorio\ con\ ejemplos
```

Para que no tengan que escribir manualmente algo tan confuso, Pueden utilizar la tecla tab `cd` `Dire<Tab>`

Alternativamente los directorios se pueden manejar con 'comillas' simples o "comillas" dobles

```
cd "Directorio con ejemplos"
```

```
cd 'Directorio con ejemplos'
```

[change directory]

Si estamos cambiando de directorio y anteriormente no habíamos corrido el comando LS, y no recordamos cuál es el siguiente directorio al que queremos viajar, mientras corremos el comando CD podemos apretar la tecla tab para que nos sugiere cuál es el siguiente directorio de esa lista

```
→ ~ cd 'directorio con ejemplos'<tab>  
documentos/ informacion/ libros/
```

[touch]

Usando el comando Touch se puede crear un archivo o múltiples archivos.

Para crear múltiples archivos lo único que tenemos que hacer es listarlos: → ~ touch archivo1.txt  
archivo2.txt archivo3.txt

```
demo@ubuntu20:~$ mkdir directorioejemplo && touch archivocreado.txt
```

```
demo@ubuntu20:~$ ls
```

```
archivocreado.txt 'directorio con ejemplos' directorioejemplo
```

```
demo@ubuntu20:~$ rm -r directorioejemplo/ && rm archivocreado.txt
```

```
demo@ubuntu20:~$ mkdir directorioejemplo && touch directorioejemplo/archivocreado.txt
```

```
demo@ubuntu20:~$ ls
```

```
'directorio con ejemplos' directorioejemplo
```

```
demo@ubuntu20:~$ tree
```

[rm and rmdir]