MANIPULACION DE

DATOS EN BASH PARA PENTESTERS EN 5 MINUTOS

MIGUEL ANGEL VILLALOBOS GARCIA

Manipulación de datos en Bash para pentesters en 5 minutos

Por Miguel Ángel Villalobos García
https://www.linkedin.com/in/m7villalobos/

Introducción

La línea de comandos es una herramienta muy potente para trabajar manipulando datos de todo tipo. Imagina que tienes archivos enormes con miles de líneas (logs del sistema, resultados de programas, listas). Ver esos datos o buscar algo específico puede ser difícil incluso empleando programas gráficos. Aquí es donde la línea de comandos puede echarte un cable, usando pequeñas herramientas que pueden trabajar juntas. Este manual te enseñará a usar estas herramientas paso a paso.

Parte 1: Fundamentos - Ver y Contar Datos

Empecemos por lo más básico: cómo ver el contenido de tus archivos y obtener información simple sobre ellos.

1.1 Ver Archivos Completos: `cat`

- **Para qué sirve:** `cat` (viene de "concatenate", concatenar) te muestra todo el contenido de uno o más archivos en la pantalla.
- **Ejemplo:** Mostrar el contenido del archivo `mi_archivo.txt`.

```
cat mi_archivo.txt
```

- **Ejemplo:** Mostrar el contenido de dos archivos, uno después del otro.

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

1.2 Ver el Principio de un Archivo: `head`

- **Para qué sirve:** A veces los archivos son muy grandes y solo quieres ver las primeras líneas para hacerte una idea (por ejemplo, ver las cabeceras de una tabla). `head` hace eso.
- **Cómo funciona:** Por defecto, muestra las 10 primeras líneas.
- **Ejemplo:** Ver las primeras 10 líneas de `log_muy_grande.log`.

```
head log_muy_grande.log
```

- **Opción útil (`-n`):** Puedes decirle cuántas líneas quieres ver con `-n` .
- **Ejemplo:** Ver las primeras 5 líneas.

```
head -n 5 log_muy_grande.log
```

1.3 Ver el Final de un Archivo: `tail`

- **Para qué sirve:** Similar a `head` , pero `tail` muestra las últimas líneas. Muy útil para ver lo más reciente en archivos que se actualizan constantemente, como los logs.
- **Cómo funciona:** Por defecto, muestra las 10 últimas líneas.
- **Ejemplo:** Ver las últimas 10 líneas de `log_del_sistema.log` .

```
tail log_del_sistema.log
```

- **Opción útil (`-n`):** Igual que `head` , `-n` te permite elegir cuántas líneas ver.
- **Ejemplo:** Ver las últimas 20 líneas.

```
tail -n 20 log_del_sistema.log
```

- **Opción clave (`-f`):** ¡Esta es muy importante! `tail -f` (follow) se queda "enganchado" al archivo y te muestra las nuevas líneas que se añaden en tiempo real. Ideal para monitorizar. Para detenerlo, pulsa `Ctrl + C` .
- **Ejemplo:** Monitorizar un archivo de log en vivo.

```
tail -f /var/log/syslog
```

1.4 Contar Cosas en Archivos: `wc`

- **Para qué sirve:** `wc` (word count) cuenta líneas, palabras y caracteres/bytes en un archivo.
- **Ejemplo:** Obtener el recuento completo (líneas, palabras, bytes) de `mi_documento.txt` .

```
wc mi_documento.txt
```

(La salida será algo como: `15 85 450 mi_documento.txt` -> 15 líneas, 85 palabras, 450 bytes)

- **Opciones útiles:**
  - `-l`: Contar solo las líneas.
  - `-w`: Contar solo las palabras.
  - `-c`: Contar solo los bytes.
- **Ejemplo:** Contar cuántas líneas tiene una lista de usuarios.

```
wc -l lista_usuarios.txt
```

- **Ejemplo:** Contar cuántas palabras tiene un informe.

```
wc -w informe.txt
```

Parte 2: Primeros Pasos en Manipulación - Redirección y Tuberías

Ahora que sabemos ver y contar, veamos cómo guardar los resultados y cómo hacer que los comandos trabajen juntos.

2.1 Guardar la Salida de un Comando: Redirección (`>` y `>>`)

- **Para qué sirve:** En lugar de ver la salida en pantalla, puedes guardarla en un archivo.
- **Operador `>` (Mayor que):** Guarda la salida en un archivo. **¡Cuidado! Si el archivo ya existe, lo borrará y creará uno nuevo.**
  - **Ejemplo:** Guardar la lista de archivos del directorio actual en `contenido_directorio.txt`.

```
ls > contenido_directorio.txt
```

- **Operador `>>` (Doble mayor que):** Guarda la salida **añadiéndola al final** de un archivo existente. Si el archivo no existe, lo crea.
  - **Ejemplo:** Añadir la fecha y hora actual a un archivo de registro `eventos.log`.

```
date >> eventos.log
```

2.2 Conectar Comandos: Tuberías (`|`)

- **Para qué sirve:** ¡Este es el concepto más potente! La tubería (`|`, a veces llamada "pipe") conecta la salida de un comando con la entrada del siguiente. Es como una cadena de montaje: lo que produce el primero, lo procesa el segundo.
- **Ejemplo simple:** Contar cuántos archivos hay en el directorio actual.

```
ls | wc -l
```

Explicación: `ls` lista los archivos (la salida va a la tubería), y `wc -l` recibe esa lista como entrada y cuenta las líneas.

- **Ejemplo:** Ver las primeras 5 líneas de un log (usando `cat` y `head`).

```
cat log_muy_grande.log | head -n 5
```

Explicación: `cat` envía todo el contenido del log a la tubería, y `head -n 5` solo toma las primeras 5 líneas de esa entrada.

2.3 Usar un Archivo como Entrada: Redirección (`<`)

- **Para qué sirve:** Le dice a un comando que lea su entrada desde un archivo, como si estuvieras escribiendo el contenido de ese archivo directamente en el comando.
- **Ejemplo:** Contar las líneas de `lista_usuarios.txt` usando `<`.

```
wc -l < lista_usuarios.txt
```

(Funciona igual que `wc -l lista_usuarios.txt`, pero ilustra cómo funciona `<`)

2.4 Manejar Errores: Redirección (`2>` y `&>`)

- **Concepto:** A veces los comandos producen errores (ej: archivo no encontrado). Estos errores van a un canal diferente llamado "salida de error" (stderr). La salida normal va a "salida estándar" (stdout). Por defecto, ambos se ven en la pantalla. Podemos redirigir los errores.
- **Operador `2>`:** Redirige solo los mensajes de error (del canal 2) a un archivo.
  - **Ejemplo:** Intentar ver un archivo que no existe y guardar el error.

```
cat archivo_que_no_existe 2> errores.log
```

(No verás nada en pantalla, pero `errores.log` contendrá el mensaje de error).

- **Operador `&>`:** Redirige **todo** (salida normal y errores) al mismo archivo.
  - **Ejemplo:** Ejecutar un comando y guardar absolutamente toda su salida.

```
./script_complejo.sh &> salida_completa.log
```

Parte 3: Filtrar y Transformar Datos

Ahora vamos a modificar y seleccionar la información que nos interesa.

3.1 Buscar Texto: `grep`

- **Para qué sirve:** `grep` es como un buscador súper potente para encontrar líneas que contengan un texto (o patrón) específico dentro de archivos o de la salida de otros comandos.
- **Ejemplo simple:** Buscar la palabra "error" en el archivo `sistema.log`.

```
grep "error" sistema.log
```

- **Opción `-i`:** Buscar sin distinguir mayúsculas/minúsculas.
  - **Ejemplo:** Buscar "Usuario" o "usuario".

```
grep -i "usuario" accesos.log
```

- **Opción `-v`:** Invertir la búsqueda. Muestra las líneas que *NO* contienen el texto.
  - **Ejemplo:** Mostrar todas las líneas de `config.txt` que *no* son comentarios (no empiezan por `#`).

```
grep -v "^#" config.txt
```

  (`^` significa "inicio de línea")
- **Uso con Tuberías:** `grep` es muy útil para filtrar la salida de otros comandos.
  - **Ejemplo:** Ver los procesos del usuario "apache".

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

- **Opción `-o`:** Muestra *solo* la parte de la línea que coincide con el patrón, no la línea entera. Útil para extraer datos.
  - **Ejemplo:** Extraer solo las direcciones IP (patrón simple) de un archivo.

```
grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"
texto_con_ips.txt
```

  (Esto es una "expresión regular", un patrón avanzado para buscar texto complejo)
- **Opción `-E`:** Permite usar expresiones regulares más potentes y fáciles de leer.
  - **Ejemplo:** Buscar líneas que contengan "error" o "warning".

```
grep -E "error|warning" system.log
```

  (`|` significa "o")

3.2 Transformar Caracteres: `tr`

- **Para qué sirve:** `tr` (translate) sirve para cambiar o eliminar caracteres individuales. No busca palabras, solo caracteres sueltos.
- **Ejemplo:** Convertir un texto de minúsculas a mayúsculas. (Necesita recibir la entrada de otro comando o archivo).

```
cat mi_texto.txt | tr 'a-z' 'A-Z'
```

- **Opción `-d`:** Eliminar caracteres.
  - **Ejemplo:** Eliminar todos los espacios de un texto.

```
cat texto_con_espacios.txt | tr -d ' '
```

- **Opción `-s`:** Reducir secuencias de caracteres repetidos a uno solo.
  - **Ejemplo:** Reemplazar múltiples espacios seguidos por uno solo.

```
echo "Texto   con    espacios   extra" | tr -s ' '
# Salida: Texto con espacios extra
```

Parte 4: Trabajar con Columnas y Ordenar

Muchos datos vienen en forma de tabla o columnas. Estas herramientas nos ayudan a manejarlos.

4.1 Cortar Columnas: `cut`

- **Para qué sirve:** `cut` permite "cortar" y quedarse solo con ciertas columnas (campos) o caracteres de cada línea.
- **Opción `-d`:** Especifica qué carácter separa las columnas (el "delimitador"). Si no se usa, asume que es el tabulador.
- **Opción `-f`:** Indica qué columnas (campos) queremos. Se numeran desde 1.
  - **Ejemplo:** Sacar la primera columna (nombres de usuario) del archivo `/etc/passwd`, que usa `:` como separador.

```
cut -d':' -f1 /etc/passwd
```

  - **Ejemplo:** Sacar la primera y tercera columna de un archivo CSV (separado por comas).

```
cut -d',' -f1,3 datos.csv
```

- **Opción `-c`:** Indica qué posiciones de caracteres queremos.
  - **Ejemplo:** Sacar los caracteres del 1 al 10 de cada línea.

```
cut -c1-10 cualquier_archivo.txt
```

4.2 Ordenar Datos: `sort`

- **Para qué sirve:** `sort` ordena las líneas de un archivo o de la entrada que recibe.
- **Ejemplo simple:** Ordenar alfabéticamente una lista de nombres.

```
sort nombres.txt
```

- **Opción `-r`:** Ordenar en orden inverso (descendente).
  - **Ejemplo:** Ordenar números de mayor a menor.

```
sort -r numeros.txt
```

- **Opción `-n`:** Ordenar numéricamente (si no, ordena "10" antes que "2").
  - **Ejemplo:** Ordenar correctamente una lista de números.

```
sort -n numeros.txt
```

- **Uso con Tuberías:** Ordenar los resultados de otro comando.
  - **Ejemplo:** Obtener los usuarios y ordenarlos alfabéticamente.

```
cut -d':' -f1 /etc/passwd | sort
```

4.3 Datos Únicos: `uniq`

- **Para qué sirve:** `uniq` filtra líneas duplicadas... pero **solo si están juntas (adyacentes)**. Por eso, ¡casi siempre se usa después de `sort`!
- **Ejemplo simple:** Obtener una lista de elementos únicos (eliminando duplicados).

```
sort lista_con_duplicados.txt | uniq
```

- **Opción `-c`:** Contar cuántas veces aparece cada línea consecutiva. ¡Muy útil!

- **Ejemplo:** Contar cuántas veces ha accedido cada IP a un servidor (necesita extraer IPs y ordenar primero).

```
# (Asumiendo que $1 es la IP en el log)
cat access.log | awk '{print $1}' | sort | uniq -c
```

(Esto mostrará algo como: `15 192.168.1.5`, `5 10.0.0.2`)

- **Patrón Común:** Un uso muy frecuente es encontrar los elementos más comunes: extraer datos -> ordenar -> contar únicos -> ordenar por contador.
  - **Ejemplo:** Encontrar las 5 IPs que más veces aparecen en el log anterior.

```
cat access.log | awk '{print $1}' | sort | uniq -c | sort -nr |
head -n 5
```

Explicación paso a paso:

1. `cat access.log | awk '{print $1}'`: Extrae las IPs.
2. `sort`: Ordena las IPs para que `uniq` funcione.
3. `uniq -c`: Cuenta cuántas veces seguidas aparece cada IP.
4. `sort -nr`: Ordena numéricamente (`-n`) y en reverso (`-r`) basado en el contador. Los más frecuentes quedan arriba.
5. `head -n 5`: Muestra solo los 5 primeros (los más frecuentes).

Parte 5: Edición y Procesamiento Avanzado

Estas herramientas son más potentes y permiten transformaciones más complejas.

5.1 Editor de Flujo: `sed`

- **Para qué sirve:** `sed` es como un robot editor que puede hacer cambios en un archivo o flujo de texto automáticamente, línea por línea, siguiendo tus instrucciones.
- **Comando Básico (`s/buscar/reemplazar/`)**: La instrucción más común es sustituir (`s`).
  - **Ejemplo:** Cambiar la primera "manzana" por "pera" en cada línea.

```
sed 's/manzana/pera/' mi_lista.txt
```

- **Sustitución Global (`g`)**: Para cambiar *todas* las "manzanas" por "peras" en cada línea, añade `g` al final.
  - **Ejemplo:**

```
sed 's/manzana/pera/g' mi_lista.txt
```

- **Comando Eliminar (`d`)**: Elimina las líneas que cumplan una condición.
  - **Ejemplo:** Eliminar todas las líneas que contengan la palabra "temporal".

```
sed '/temporal/d' archivo.log
```

  - **Ejemplo:** Eliminar líneas vacías.

```
sed '/^$/d' texto_con_lineas_vacias.txt
```

  (`^$` es un patrón que significa línea vacía)
- **Imprimir solo lo necesario (`-n` y `p`)**: Normalmente `sed` imprime todas las líneas (modificadas o no). Con `-n` no imprime nada por defecto, y con `p` (print) le dices qué líneas imprimir.
  - **Ejemplo:** Imprimir solo las líneas que contienen "IMPORTANTE".

```
sed -n '/IMPORTANTE/p' documento.txt
```

- **Aplicar a Rangos (Direcciones)**: Puedes decirle a `sed` que aplique un comando solo a ciertas líneas (por número o por patrón).
  - **Ejemplo:** Eliminar las líneas 1 a 5.

```
sed '1,5d' archivo_largo.txt
```

  - **Ejemplo:** Sustituir solo entre las líneas que contienen "START" y "END".

```
sed '/START/,/END/s/viejo/nuevo/g' bloque_de_texto.txt
```

5.2 Procesador de Texto Avanzado: `awk`

- **Para qué sirve:** `awk` es como tener una mini-hoja de cálculo en la línea de comandos. Es genial para archivos con columnas, porque entiende de campos y puede hacer cálculos o tomar decisiones basadas en ellos.
- **Concepto Clave:** `awk` lee línea por línea. Cada línea la divide en "campos" (columnas), que se llaman `$1` (el primero), `$2` (el segundo), `$3`, etc. `$0` es la línea entera.
- **Acción por Defecto (Imprimir todo):** Si no le dices qué hacer, `awk` simplemente imprime cada línea, como `cat`.

```
awk '{ print $0 }' mi_archivo.txt
```

- **Imprimir Campos Específicos:** Puedes decirle qué campos imprimir.

- **Ejemplo:** Imprimir el primer y tercer campo de cada línea.

```
awk '{ print $1, $3 }' datos.txt
```

- **Especificar el Separador (`-F`):** Igual que `cut -d`, puedes decirle a `awk` qué carácter separa los campos con `-F`.
  - **Ejemplo:** Imprimir el nombre de usuario (campo 1) del `/etc/passwd` (separador `:`).

```
awk -F':' '{ print $1 }' /etc/passwd
```

- **Hacer Cosas si se Cumple una Condición:** Puedes poner una condición antes de la acción `{ }`. La acción solo se ejecuta si la condición es verdadera para esa línea.
  - **Ejemplo:** Imprimir solo las líneas donde el tercer campo sea mayor que 100.

```
awk '$3 > 100 { print $0 }' datos_numericos.txt
```

  - **Ejemplo:** Imprimir el primer campo si el segundo campo es exactamente "ERROR".

```
awk '$2 == "ERROR" { print $1 }' log_aplicacion.log
```

- **Bloques Especiales (`BEGIN` y `END`):**
  - `BEGIN { ... }`: Código que se ejecuta *antes* de leer ninguna línea (ej: imprimir una cabecera).
  - `END { ... }`: Código que se ejecuta *después* de leer todas las líneas (ej: imprimir totales).
  - **Ejemplo:** Calcular la suma de los valores en la primera columna.

```
awk 'BEGIN { suma = 0 } { suma += $1 } END { print "Suma total:", suma }' numeros.txt
```

    Explicación: Antes de empezar (`BEGIN`), pone `suma` a 0. Por cada línea, añade el valor del primer campo (`$1`) a `suma`. Al final (`END`), imprime el resultado.

Parte 6: Combinando Todo - Flujos de Trabajo

La verdadera magia ocurre al combinar estas herramientas para resolver problemas reales.

- **Flujo 1: Análisis Básico de Logs Web**

- **Objetivo:** Encontrar las 5 páginas más visitadas en un log de acceso simple (campo 7 es la ruta).
- **Pasos:**
  1. Extraer la ruta solicitada (campo 7).
  2. Ordenar las rutas para agrupar las iguales.
  3. Contar cuántas veces aparece cada ruta única.
  4. Ordenar por el número de visitas (descendente).
  5. Mostrar las 5 primeras.
- **Comando:**

```
cat access.log | awk '{print $7}' | sort | uniq -c | sort -nr |
head -n 5
```

- **Flujo 2: Encontrar Usuarios con Shell Inválida en `/etc/passwd`**
  - **Objetivo:** Listar usuarios cuyo shell (último campo) no sea uno de los válidos (ej: `/bin/bash`, `/bin/sh`, `/usr/sbin/nologin`).
  - **Pasos:**
    1. Asegurarse de que el archivo `/etc/passwd` existe.
    2. Extraer el último campo (shell) usando `awk`.
    3. Filtrar usando `grep -v` para excluir los shells válidos.
    4. Ordenar y obtener únicos para tener una lista limpia.
  - **Comando:**

```
cat /etc/passwd | awk -F':' '{print $NF}' | grep -v -E
'^(/bin/bash|/bin/sh|/usr/sbin/nologin)$' | sort | uniq
```

Explicación del `grep`: `-v` invierte, `-E` usa regex extendida, `^` inicio de línea, `(...)` agrupa, `|` es "o", `$` fin de línea. Busca líneas que _no sean exactamente uno de esos shells._

¡Felicidades! Si has llegado hasta aquí, seguramente hayas aprendido bastantes comandos, trucos, y los fundamentos de herramientas muy potentes como: `cat`, `head`, `tail`, `wc`, `grep`, `tr`, `cut`, `sort`, `uniq`, `sed` y `awk`, además de cómo conectarlas con tuberías (`|`) y redirecciones (`>`, `>>`, `<`).

La clave ahora es practicar. Intenta usar estos comandos con tus propios archivos o con la salida de otros programas. Empieza con tareas simples y poco a poco combina más herramientas. ¡Verás cómo puedes analizar y manipular texto de forma increíblemente rápida y eficiente! ¡Hasta la próxima!