💻 Examen 103 - Comandos GNU y Unix

103.4 Usar flujos, pipes y redirecciones

Teoría

Como mencionamos en el objetivo 103.2, los comandos en Linux interactúan con tres flujos estándar: entrada estándar (stdin), salida estándar (stdout) y error estándar (stderr). Por defecto, stdin viene del teclado y stdout/stderr van a la pantalla. Las redirecciones y las tuberías te permiten cambiar estos destinos u orígenes.

Redirecciones:

Las redirecciones cambian el origen de stdin o el destino de stdout/stderr.

1. **Redirección de Salida (`>` y `>>`):**
    - `comando > archivo`: Redirecciona la salida estándar (`stdout`) del `comando` al `archivo`. **Si el `archivo` ya existe, su contenido se sobrescribe.**
    - `comando >> archivo`: Redirecciona la salida estándar (`stdout`) del `comando` al `archivo`. **Si el `archivo` ya existe, la salida se añade al final (append).**
2. **Redirección de Entrada (`<`):**
    - `comando < archivo`: Redirecciona el contenido del `archivo` para que sea la entrada estándar (`stdin`) del `comando`. Muchos comandos (como `grep`, `sort`, `wc`) aceptan un archivo como argumento, pero esta es la forma explícita de usar stdin.
3. **Redirección de Error Estándar (`2>` y `2>>`):**
    - `comando 2> archivo_error`: Redirecciona la salida de error estándar (`stderr`) del `comando` al `archivo_error`.
    - `comando 2>> archivo_error`: Redirecciona la salida de error estándar (`stderr`) del `comando` al `archivo_error`, añadiendo al final si ya existe.
    - El número `2` es el descriptor de archivo asociado con stderr. `1` es para stdout (implícito con `>`) y `0` para stdin (implícito con `<`).
4. **Redirección de Salida Estándar y Error Estándar Juntos (`&>` y `&>>` o variantes antiguas):**
    - `comando &> archivo_todo`: Redirecciona tanto `stdout` como `stderr` al `archivo_todo`. **Sobrescribe.** (Sintaxis de Bash)
    - `comando &>> archivo_todo`: Redirecciona tanto `stdout` como `stderr` al `archivo_todo`. **Añade.** (Sintaxis de Bash)
    - Alternativas más antiguas (compatibles con otras shells):
        - `comando > archivo_todo 2>&1`: Redirecciona `stdout` a `archivo_todo`, y luego redirecciona `stderr` (`2`) al mismo destino que `stdout` (`&1`). El orden es importante aquí.
        - `comando >> archivo_todo 2>&1`: Similar, pero añadiendo.

5. **Redirección a `/dev/null`:**
    - `/dev/null` es un archivo especial (el "agujero negro" del sistema) que descarta cualquier dato escrito en él y proporciona un flujo vacío cuando se lee.
    - `comando > /dev/null`: Descarta la salida estándar.
    - `comando 2> /dev/null`: Descarta la salida de error estándar.
    - `comando &> /dev/null`: Descarta ambas salidas. Útil para ejecutar comandos en silencio, sin que muestren nada en la terminal.

Tuberías (Pipes - `|`):

Una tubería conecta la salida estándar (`stdout`) de un comando a la entrada estándar (`stdin`) de otro comando. Permite encadenar comandos para procesar datos secuencialmente.

- `comando1 | comando2`: La salida de `comando1` se convierte en la entrada de `comando2`. `comando2` procesa los datos que recibe de `comando1`. Puedes encadenar múltiples comandos: `comando1 | comando2 | comando3 | ...`.

Combinación de Redirecciones y Pipes:

Puedes usar redirecciones y pipes en la misma línea de comandos. Por ejemplo:

- `comando1 archivo_entrada | comando2 > archivo_salida`: `comando1` lee de `archivo_entrada`, su salida se envía a `comando2`, y la salida de `comando2` se guarda en `archivo_salida`.
- `comando1 | comando2 2> archivo_error`: La salida de `comando1` es la entrada de `comando2`. Los errores de `comando2` van a `archivo_error`, mientras que su salida estándar sigue yendo a la pantalla (o a la siguiente pipe si la hay).

Dominar redirecciones y pipes es esencial para el scripting de shell y la administración eficiente de Linux. Te permite usar comandos pequeños y especializados para construir soluciones complejas.