PRÁCTICA DE LABORATORIO 6

REDIRECCIÓN DE E/S

En repetidas ocasiones en la vida de un sistema es mejor tener todo en archivos, ya sea para guardar algún historial o para automatizar ciertas funciones dentro de scripts.

Para almacenar o sacar información de archivos y vincularlas con entradas o salidas estándares se utilizan las Redirecciones.

La redirección se expresa con los símbolos "Mayor" > y "Menor" <. Y se pueden utilizar en forma simple o doble.

Utilizando el comando cat se puede hacer una copia de arch1.txt a arch2.txt utilizando redirección.

```
$ cat arch1.txt > arch2.txt
```

O se puede redireccionar un archivo para visualizarlo con el comando less.

```
$ less < arch1.txt
```

Redirección de escritura

Para escribir un archivo se utiliza >. Hay que tener mucho cuidado de no borrar un archivo sobre-escribiéndolo. Cuando se utilizan redirecciones, debido a su utilidad en los scripts, "no se realizan confirmaciones".

Si el archivo a escribir ya existe desde antes, el redireccionador > lo sobreescribe con flujo de texto nuevo.

En cambio, el operador >> realiza un agregado de texto en el flujo existente.

No hay nada mejor que un ejemplo clarificador:

```
$ escribe-en-salida-estandar > archivo.txt
```

El (falso) comando escribe-en-salida-estándar justamente hace eso, escribe unas cuantas cosas en salida estándar.

Puede ser un comando ls, un comando cal (calendario) o cualquier comando antes visto, así como también una combinación de comandos por tuberías.

En este punto, el contenido de archivo.txt es lo mismo que saldría en pantalla. Si ejecutamos otro comando redireccionado a archivo.txt (nuevamente), éste pierde su contenido y el resultado de la operación pasa a estar en el mismo.

Cuando se necesita tener una lista de acontecimientos, no se quiere que un acontecimiento nuevo borre a todos los anteriores. Para lograr esto agregamos en vez de sobreescribir.

```
$ echo Este es el acontecimiento Nro. 1 > bitacora.log
$ echo Este es el segundo acontecimiento >> bitacora.log
```

Va a escribir dos líneas en el archivo bitacora.log sin eliminar nada.

Ejemplo: Si queremos combinar el ejemplo de las tuberías con lo aprendido recientemente podríamos escribir:

```
$ cat archivo.txt | sort | sacar-repetidas | diccionario >> glosario.txt
```

Redirección de lectura

Para la lectura es el símbolo menor < y se utiliza de la siguiente manera:

```
$ comando-que-acepta-stdin < archivo-de-entrada.txt
```

Como por ejemplo:

```
$ mail usuario1 usuario2 < correo.txt
```

Dónde correo.txt podría ser un archivo que se genere automáticamente... así como su contenido. Otra facilidad para redireccionar entrada estándar es <<, que después de un comando, permite ingresar, por teclado, un texto que se constituirá en la entrada estándar.

A continuación de << debe ponerse una palabra, que indicará fin de entrada (en nuestro ejemplo, adiós). La entrada estándar constará de las líneas que se digiten a continuación hasta la primera que contenga sólo la palabra que indicaba fin de entrada. Por ejemplo:

```
$ sort <<fin
> Perú
> Argentina
> Brasil
> fin
Argentina
Brasil
Perú
```

ordenará las palabras dadas (excepto chau que indica el fin de la entrada). Así, << es equivalente a editar un archivo y después redireccionarlo a la entrada estándar de un programa.

TUBERÍAS

Podríamos representar cada programa como una caja negra que tiene una entrada y una salida que se pueden unir entre ellos.

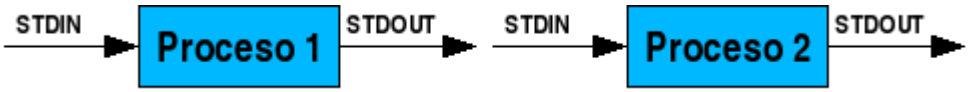
Debido a que la entrada por defecto es el teclado y la salida por defecto es terminal, graficaremos cuando sea necesario ambos.

El ejemplo que utilizamos se encuentra esquematizado en la figura

Siendo la entrada estándar el teclado y la salida estándar el terminal o por simplicidad la pantalla.

Vamos a suponer un caso ficticio donde necesitamos todas las definiciones de cada palabra en un texto. Primero las ordenamos alfabéticamente, luego utilizamos un comando ficticio llamado diccionario que toma palabras de la entrada estándar y las reescribe junto a su significado en la salida estándar.

Su esquema se ve en la figura



En este caso nombramos por separado las entradas y salidas estándares de los dos programas, pero la unión entre ambos programas se puede considerar como un sólo tubo.

En ese tubo, el flujo está en un estado intermedio, donde está ordenado, pero no tiene las definiciones de diccionario.

En la línea de comandos esto se escribe de la siguiente manera:

```
$ sort | diccionario
```

Donde el caracter " | " representa la conexión entre la salida estándar de un programa y la entrada estándar de otro.

Con este fuerte y simple concepto se pueden concatenar gran cantidad de programas como si fuera una línea de producción en serie para generar resultados complejos.

Para mejorar nuestro ejemplo sacaremos las palabras repetidas, antes de mostrarlas con definiciones. Suponiendo que exista un programa llamado sacar-repetidas, la línea de comando sería:

```
$ sort | sacar-repetidas | diccionario
```

Simple, utilizando herramientas sencillas logramos algo un poco más complicado. El inconveniente que tenemos en este ejemplo es que hay que escribir aquello a procesar. Normalmente queremos utilizar archivos como entrada de nuestros datos.

Es necesario un comando que envíe a la salida estándar un archivo, así se procesa como la entrada estándar del sort y continúa el proceso normalmente. Este comando es cat. La sintaxis es simple

```
cat nombre-de-archivo
```

Quedando nuestro ejemplo:

```
$ cat archivo.txt | sort | sacar-repetidas | diccionario
```

... esto crea un glosario de las palabras que se encuentren en archivo.txt

La combinación de comandos es incalculable y brinda posibilidades enormes. Veamos algunos ejemplos.

En el caso que se quieran buscar procesos con el string http:

```
$ ps ax | grep http
3343 ?        S      0:00 httpd -DPERLPROXIED -DHAV
3344 ?        S      0:00 httpd -DPERLPROXIED -DHAV
3975 ?        S      0:00 httpd -DPERLPROXIED -DHAV
12342 pts/6    S      0:00 grep http
```

Si queremos eliminar la última línea podemos volver a usar grep con la opción -v

```
$ ps ax | grep http | grep -v grep
3343 ?        S      0:00 httpd -DPERLPROXIED -DHAV
3344 ?        S      0:00 httpd -DPERLPROXIED -DHAV
3975 ?        S      0:00 httpd -DPERLPROXIED -DHAV
```

Se pueden filtrar las líneas que contengan la palabra linux del archivo arch1.txt y luego mostrarlas en un paginador como less.

```
$ cat arch1.txt | grep linux | less
```

Podemos enviar los resultados por correo a un amigo, con un asunto que diga "Tu archivo".

```
$ cat arch1.txt | grep linux | mail -s 'Tu archivo' amigo@email.com
```

EJERCICIOS

1. En su directorio home crear el directorio `lab6`
2. Dentro de `lab6` crear dos directorios llamados: `uno` y `dos` (hacerlo con un solo comando)
3. Escribir UN comando que cree un archivo oculto llamado secreto dentro de `desarrollo_lab6` con el contenido "Este es un secreto". Los archivos ocultos en Linux deben empezar con un punto, ejemplo `.oculto`
4. Estando dentro de `lab6`, escribir un comando que cree un archivo llamado `ficheros_bin_descendente` que contenga el listado de todos los ficheros que se encuentran en el directorio `/bin` ordenados de manera descendente (de z - a).
5. Ingresar al directorio `uno`, Estando en `uno`, escribir un comando que utilice tuberias y filtros para crear, dentro del directorio `dos`, un fichero llamado `ficheros_bin_ascendente` que contenga una copia del texto del archivo `ficheros_bin_descendente` pero ordenado de manera ascendente.
6. Estando en el directorio `uno`, escribir un comando que mueva el archivo `ficheros_bin_descendente` al directorio `dos`
7. Estando en el directorio `uno`, escribir un comando que nos permita mover hacia el directorio `dos`
8. Estando en `dos`, escribir un comando que cree un archivo oculto llamado `tmp`
9. Estando en `dos`, escribir un comando que elimine el directorio `uno`
10. Estando en `dos`, escribir un comando que mueva todos los ficheros en el directorio actual hacia la carpeta `lab6`
11. Estando en `dos`, eliminar el directorio `dos` (si, el directorio actual)
12. Sin moverse del directorio actual, crear un archivo llamado `tmp`. ¿Qué sucede?
13. Moverse al directorio `lab6`.
14. Escribir un comando que cree un fichero llamado `ficheros` que contenga el listado de todos los ficheros (incluidos los archivos ocultos) que se encuentren en el directorio actual.
15. Escribir un comando que cree un fichero llamado `historial_mk` que contenga el historial de comandos ejecutados que contienen la palabra `mkdir`.

Referencias

Erazo, H. (2016). *Tutorial del Editor de Texto Nano*. *Nanotutoriales.com*. Retrieved 25 May 2016, from https://www.nanotutoriales.com/tutorial-del-editor-de-texto-nano

Manual para aprender a utilizar VIM | Emezeta. (2008). *Emezeta.com*. Retrieved 25 May 2016, from http://www.emezeta.com/articulos/manual-para-aprender-a-utilizar-vim