Preparación para el examen LPI 101

Tema 103.4

Usando streams, pipes y redirecciones

Créditos y licencia de uso

Coordinación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Oscar Casal (ocs) oscar@glug.es

Traducción:
Pere Catalan (arGos) pecat@cetib.ictnet.es
Manuel Guillán (xLekOx) lpi@xlekox.org
Al H. Carril (Mandraker) spoofff@hotmail.com

Maquetación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net

Versión 1.1 (05-08-2004 10:30)

ÍNDICE

Índice de contenido

Tema 103.4

Introducción

En este capítulo se verá como usar Streams, Tuberías (pipes) y Redirecciones Unix. Conectar archivos a comandos y comandos a otros comandos para procesar datos de texto de forma eficaz. Incluye la redirección de la entrada estándar, salida estándar y error estándar; y canalizar la salida de un comando como entrada de otro comando o como argumento (usando xargs); enviando la salida hacia stdout (salida estándar) y a un archivo (usando tee).

Los comandos que se verán en este tema son:

- tee
- xargs
- <
- <<
- >
- >>
- |

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 5 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Uso de los Streams (flujos de datos), Pipes (tuberías), y Redireccionamientos

Una de las muchas ventajas de los sistemas Linux y Unix es la noción de que cualquier cosa es un fichero. Un disco y sus particiones, cintas, terminales (pantallas), puertos serie, el ratón, e incluso el audio están mapeados (organizados y jerarquizados) en el sistema de ficheros. Este mapeo permite a los programas interactuar con cualquier tipo de dispositivos y ficheros del mismo modo, lo cual simplifica su interfaz enormemente.

Cada dispositivo que está representado por un fichero es llamado dispositivo de fichero (device file o dev), lo cual lo convierte en un objeto especial en el sistema de ficheros que proporciona un interfaz hacia el dispositivo. El kernel asocia los drivers del dispositivo (el físico) con varios dispositivos de ficheros, así es como el sistema da la apariencia de que a los dispositivos se puede acceder como si fuesen ficheros. Utilizando una terminal (pantalla) como ejemplo, un programa lee del dispositivo fichero del terminal que recibirá los caracteres tecleados en el teclado. La escritura en el terminal (fichero) provoca que los caracteres aparezcan en pantalla. Aunque parece raro pensar en un terminal (pantalla) como un fichero, este mismo concepto unifica y dota de simplicidad la programación de Linux y Unix.

Entrada y salida estándar and File Descriptors por defecto

La entrada/salida estándar (standard I/O) es responsabilidad del shell, habitualmente se usan en todas las utilidades basadas en consola de GNU/Linux para controlar y dirigir las entradas de programa, las salidas y la información de errores. Cuando un programa (proceso) se ejecuta tiene disponibles tres *streams*, o tipos de flujo de datos: uno para la entrada, otro para la salida y otro para mostrar mensajes de error o de diagnóstico, se describen a continuación:

Entrada estándar (abreviada *stdin*)
Este "file descriptor" es una entrada de flujo de texto. Por defecto está asociada al teclado. Cuando se teclea en un programa interactivo de texto, se está alimentando la entrada estándar. Como se puede ver, algunos programas emplean uno o más nombres de ficheros como argumentos de la línea de comandos e ignoran la entrada estándar. La entrada estándar está asignada al *file descriptor 0*.

Salida estándar (abreviada *stdout*)
Este "file descriptor" es la salida por defecto del flujo de caracteres de un programa. Por defecto está asociada al terminal (o ventana). La salida generada a través de los comandos es escrita en la salida estándar para ser reflejada en la pantalla. La salida estándar está asignada al *file descriptor 1*.

Error estándar (abreviada *stderr*)
Este "file descriptor" es también una salida de flujo de caracteres, pero se usa exclusivamente para mostrar errores u otra información no relacionada con los resultados normales del comando. Por defecto esta salida está asociada al terminal de salida, igual que la salida estándar. Ello implica que tanto stdout como stderr convergen en la pantalla, de modo que pueden ser confundidas. El modo de controlar esto se tratará más adelante. La salida de error estándar está asociada al *file descriptor 2*.

Los descriptores de fichero de la entrada/salida estándar se usan en el momento de creación y posterior ejecución de un programa para leer y escribir ficheros del disco. Ello permite asociar comandos a ficheros y a dispositivos, gestionando los comandos de entrada y salida exactamente

como se desee. La diferencia es que los programas ya tienen los descriptores de fichero -asignados por defecto gracias a la shell- y no es necesario crearlos explícitamente.

Los datos enviados a las utilidades que se verán en este capítulo suelen venir tanto de la stdin como de un archivo. Muchas utilidades escriben la salida y los errores hacia la stdout y stderr respectivamente. Esto es adecuado para ver los resultados; sin embargo, si se desea guardar los resultados en un archivo al que se pueda acceder más tarde, Linux tiene una solución para ello: la Redirección.

Redirecciones

Es posible cambiar la stdin, stdout y stderr cuando se trabaja con una shell de Linux. El stdin se redirige por medio del símbolo <. Muchas utilidades trabajan usando información proveniente desde el stdin o desde un archivo, pero también se pueden enviar datos a esas aplicaciones usando las redirecciones.
Un ejemplo del uso de la redirección de stdin es el siguiente, en el que el comando sort, recibe la entrada de datos desde un fichero 'listaNombres' en lugar de hacerlo por teclado (entrada estándar):

```
$ sort < listaNombres
```

Mucho más a menudo se querrá redirigir la stdout. Esto se hace por medio del símbolo >.
Ver el contenido de un archivo en orden alfabético puede no tener tanto valor como poder guardar esa lista ordenada para futuras consultas. Usando la redirección se puede guardar la salida en otro archivo.
El siguiente ejemplo usa la redirección para enviar el resultado del comando sort a un archivo llamado “alfaNombres”:

```
$ sort listaNombres > alfaNombres
```

Además de redirigir stdout para crear un archivo nuevo, se puede querer redirigir stdout para añadir datos a un archivo ya existente. Esto se hace por medio del símbolo >>.
En el siguiente ejemplo, el contenido del archivo “nicks” se ordena y el resultado se añade al archivo “alfaNombres”:

```
$ sort nicks >> alfaNombres
```

Es importante remarcar que cuando se crean ficheros, los operadores de redirección de salida son interpretados por el shell después de que los comandos se hayan ejecutado. Ello implica que ningún fichero creado con un comando de redirección puede ser abierto antes. Por ello, no se puede modificar un fichero según la siguiente secuencia:

```
$ grep "stuff" file1 > file1  ¡¡¡¡¡NO hacerlo!!!!!
```

Si file1 contiene algo importante, este comando será desastroso ya que un file1 vacío sobreescribirá el original. El comando grep será el último en ejecutarse, resultando de esto una total pérdida de datos del fichero original file1 ya que el fichero que lo reemplaza está vacío. Para evitar este problema, sencillamente se debe usar un fichero intermedio y renombrarlo:

```
$ grep "stuff" file1 > file2
$ mv file2 file1
```

También se puede redirigir stderr junto con stdout. Para redirigir stderr y stdout hacia un archivo nuevo, se usa el símbolo >&. Por ejemplo:

```
$ sort listaNombres >& listaNombresOrdenada
```

Si no hay errores, el contenido del archivo de destino será el mismo que si no se hubiera redirigido stderr. Sin embargo, usando >&, cualquier error que ocurra será incluido en el archivo.

Si se quiere redirigir únicamente stderr hacia un archivo, se puede usar el símbolo 2>. Por ejemplo, para redirigir sólo los errores del comando sort usado antes, se usará:

```
$ sort listaNombres 2> erroresListaNombres
```

Esto también es útil si se quiere ejecutar un comando mientras se redirige stderr y stdout hacia dos archivos separados. El ejemplo siguiente enviará stderr hacia el archivo “erroresLista” y stdout hacia el archivo “listaOrdenada”:

```
$ sort listaNombres 2> erroresLista > listaOrdenada
```

Otro tipo de redirección es la llamada “herefile”(<<) la cual consiste en especificar la entrada a un comando en varias lineas después de la llamada al mismo, terminando con un valor centinela, se ve más fácil con un ejemplo:

Si se ejecuta por consola el siguiente comando:

```
$ sort << quietoparado
```

No saldrá el prompt PS2 (>) en el cual se pondrá la lista de cosas que se quieren ordenar:

```
>hola
>caracola
>zarpando
```

para terminar de meter el listado, se insertará la palabra quietoparado (puede ser cualquier otra indicada después del símbolo <<)

```
>quietoparado
```

inmediatamente después saldrá el resultado del comando:

```
>caracola
>hola
>zarpando
```

y devolverá al prompt $

En la tabla 4-1 se listan las redirecciones más comunes de I/O estandar para el bash shell, especificadas en los Objetivos del LPI.

Las sintaxis de la redirección puede ser significativamente distinta si se usa otro shell.

Tabla 4-1. Redirecciones de I/O estandar para el bash shell

Función de Redirección	Sintaxis de bash
Enviar stdout a file	$ cmd > file $cmd 1> file
Enviar stderr a file	$ cmd 2> file
Enviar stdout y stderr a file	$ cmd > file 2>&1
Enviar stdout a file1 y stderr a file2	$ cmd > file1 2> file2
Recibir por stdin de file	$ cmd < file
Recibir por stdin desde teclado en varias lineas	$ cmd <<valor_de_parada
Añadir stdout a file	$ cmd >> file $ cmd 1>> file
Añadir stderr a file	$ cmd 2>> file
Añadir stdout y stderr a file	$ cmd >> file 2>&1

En el Exámen:
Se ha de estar preparado para diferenciar entre nombres de ficheros y nombres de comandos en aquellos comandos donde se utilizan operadores de redirección. También es necesario entender la sintaxis en comandos con redirección para estar seguro sobre que comando o fichero es fuente de datos y cual de ellos es destino.

Tuberías (pipes):

Junto con las redirecciones, se puede querer usar otros comandos. Esto se puede hacer por medio del comando "pipe" (|). Esto permite usar una línea de comandos para enviar los datos de salida de un comando como datos de entrada de otro comando.
En el siguiente ejemplo, el archivo listaNombres se ordena alfabéticamente, y después se añaden números de línea a cada nombre usando el comando nl:

```
$ sort +1 listaNombres | nl
```

Estos datos también se pueden canalizar hacia otra utilidad o redirigir hacia un archivo:

```
$ sort +1 listaNombres | nl > nombresNumerados
```

Este ejemplo muestra como se pueden usar juntas las tuberías y las redirecciones para procesar datos utilizando distintas utilidades y después guardar el resultado en un archivo.
Así como stdout se puede redirigir para guardar el resultado en un archivo, también se puede incluir stderr utilizando el símbolo |&.

Por ejemplo, el siguiente comando ordena el contenido de "listaNombres", añade números de línea, y añade cualquier información de error:

```
$ sort +1 listaNombres |& nl
```

Otro ejemplo:

```
$ grep "01523" order* | less
```

Se buscan en los ficheros cuyo nombre empiecen por “order” las líneas que contienen la palabra “01523”. Creando este pipe, la salida estándar del *grep* es enviada a la entrada estándar del *less*. El mecanismo de esta operación es tratado por el shell y transparente al usuario.

Los pipes pueden ser usados con una serie de muchos comandos. Cuando más de dos comandos se unen, la operación resultante es conocida como pipeline o text stream (flujo de texto), implicando esto que el flujo de texto pasa de un comando al siguiente.

Bifurcaciones (tee)

La utilidad tee copia la entrada estándar hacia la salida estándar y también hacia otros archivos dados como argumentos, de modo que stdin aparece tanto en stdout como en los archivos. Esto es útil cuando se quiere enviar datos no sólo hacia una tubería, sino también guardar una copia en un archivo. Si el archivo que se está escribiendo no existe todavía, se crea. Si el archivo ya existe, se sobreescribe el contenido del mismo, a no ser que se utilice la opción de añadir.
Las opciones del comando tee son las siguientes:

-a	Añade la entrada estándar la archivo especificado
-f	Ignora las señales de interrupción que podrían ser utilizadas para parar o reiniciar el proceso.

En el siguiente ejemplo, el archivo “listaNombres” se ordena primero alfabéticamente y después se canaliza hacia tee de forma que la salida se guarda también en el archivo abcNombres. Los datos son enviados entonces al comando nl, de forma que se numeran las líneas, y la salida es entonces enviada a otro archivo llamado abc123Nombres:

```
$ sort +1 listaNombres | tee abcNombres | nl > abc123Nombres
```

Pasando multiples argumentos a otros comandos (xargs)

La utilidad xargs se utiliza para pasar un gran número de argumentos a otros comandos. La utilidad args lee argumentos desde la entrada estándar, delimitados por espacios en blanco (el espacio en blanco funciona como un carácter normal cuando se utiliza entre comillas simples o dobles o tras una barra invertida) o caracteres de salto de línea, y ejecuta el comando una o más veces con cualquier número de argumentos iniciales seguidos por los argumentos leídos desde la entrada estándar. Las líneas en blanco de la entrada estándar se ignoran. Esto permite a un comando procesar más argumentos de los que en condiciones normales podría manejar.

En el ejemplo siguiente se buscan todos los archivos “README” en la base de datos “locate”, usando el link locate. Usando xargs, los nombres de los archivos son enviados a la utilidad cat, que mostrará el contenido de los mismos en pantalla. Todo el texto de estos archivos se envía a la utilidad fmt, que formatea los datos hasta un máximo de 60 caracteres por línea. La salida se envía finalmente al archivo /home/angie/readmes:

```
$ locate README | xargs cat | fmt -60 > /home/angie/readmes
```

Preguntas pre-TEST

1. ¿Que caracteres se usan para redirigir la stdout?
2. ¿ Que caracteres se usan para redirigir la stdout y stderr ?
3. ¿ Que utilidad se usa para enviar las salidas de un comando a stdout y a un fichero?

Pregunta Escenario

1- Tu empresa almacena un gran número de pequeños archivos en un directorio dentro de un sistema Linux. Se necesita combinar los contenidos de estos archivos en un gran archivo. Posteriormente se necesita partir ese gran archivo en varios archivos de la misma longitud. ¿Como se podría hacer?

Preguntas TEST

1- ¿Cual de las siguientes respuestas enviará los datos de la salida del comando ls al archivo myfiles?

A. ls | myfiles
B. ls > myfiles
C. ls < myfiles
D. ls | xargs myfiles

2- ¿Como se redirecciona stdout y stderr a un archivo?

A. <&
B. >&
C. |&
D. &&

3- ¿Que respuesta ordenaría alfabéticamente el fichero mylist, numeraría las lineas y finalmente lo separaría en ficheros que contuvieran 60 lineas cada uno?

A. sort mylist | nl > -60 lists
B. sort mylist > nl > split -60 > lists
C. sort mylist | nl | split -60 lists
D. sort mylist | nl | tee lists | split -60 lists

4- La utilidad _____ facilita información para resolver problemas, guardando la salida de un comando que es usado en tubería (pipe) con otro?

Respuestas pre-TEST

1. El carácter “mayor que” (>) se usa para redireccionar la salida standard (stdout) hacia un archivo.
2. Los caracteres >& se usan para redireccionar la salida standard (stdout) y de errores (stderr) hacia un archivo.
3. La utilidad tee se usa para enviar los resultados de la ejecución de un comando a la salida standard (stdout) y a un archivo al mismo tiempo.

Respuesta Escenario

1. ls | xargs cat | tee largefile | split -60
La utilidad xargs se usa para trabajar con grandes cantidades de datos. El comando tee envía los datos a un fichero y a la salida standard. Finalmente los datos son divididos en paginas de 60 lineas cada una.

Respuestas TEST

1. A. Las tuberías se usan para enviar datos de la salida de un comando a la entrada de otro.
2. B. Los caracteres >& envían la salida standard y de errores (stdout y stderr) a un archivo.
3. D. Las tuberías se usan para enviar datos de un comando a otro, por lo tanto A y B son incorrectas. La respuesta C no crea dos archivos, así pues solo la D es la respuesta correcta.
4. tee. Con tee se envía la salida de un comando a un archivo y a la salida estandar (stdout) al mismo tiempo.

Bibliografia y enlaces recomendados

LPIC 1 Certification Bible (Bible) by Angie Nash, Jason Nash
John Wiley & Sons; Bk&CD-Rom edition (July 1, 2001) ISBN: 0764547720

LPI Linux Certification in a Nutshell by Jeffrey Dean
O'Reilly & Associates; 1st ed edition (May 15, 2001) ISBN: 1565927486

CramSession's LPI General Linux Part 1 : Certification Study Guide
CramSession.com; ISBN: B000079Y0V; (August 17, 2000)

Referencias Unix Reviews
http://www.unixreview.com/documents/s=7459/uni1038932969999/

Página LPI: www.lpi.org

Apuntes IBM: http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir21-i.html

Manuales GPL: http://www.nongnu.org/lpi-manuals/