Universidad Carlos III De Madrid

Sistemas Operativos

Práctica opcional: Bash Scripts

Cristhian Alcántara López

NIA: 100356418 Grupo: 81

100356418@alumnos.uc3m.es

Iván Estévez Albuja

NIA: 100346193 Grupo: 81

100346193@alumnos.uc3m.es

Grado en Ingeniería Informática

22 de abril de 2018

Índice

1. Introducción

El contenido de este documento son las respuestas a preguntas planteadas en los ejercicios 2 y 5 del enunciado. Estas explicaciones se detallan en las secciones siguientes.

2. Ejercicio 2

Tenga en cuenta el código copiado a continuación y responda a las preguntas de abajo.

```
#!/bin/bash
tr -c [:alnum:] [\\n\*] < $1 | sort | uniq -c | sort -nr | head -$2
```

(a) Describa cuál es el propósito del script y de cada una de las etapas del pipeline.

Explicaremos cada uno de los comandos por separado. Partimos de un fichero de texto `entrada1.txt` cuyo contenido se muestra a continuación.

```
$ cat entrada1.txt
B-3-tres-Y-2-dos
Y-1-uno-Y-3-tres
Y-2-dos-X-3-tres
B-3-tres-B-2-dos
```

El primer comando toma los datos de su entrada, que será el fichero introducido en el parámetro $1, primer argumento del script. Convierte todos los datos no alfanuméricos en un salto de línea. Para comprobar este comportamiento, probamos este comando en la consola, usando como fichero de entrada a `entrada1.txt`.

```
$ tr -c [:alnum:] [\\n\*] < entrada1.txt
B
3
tres
Y
2
dos
Y
1
uno
Y
3
tres
Y
2
dos
X
3
tres
B
3
tres
B
2
dos
```

El segundo comando (`sort`) ordena las líneas de texto de su entrada en orden alfabético ascendente. Probamos a ejecutar este comando usando como entrada la salida del primer comando, es decir, uniéndolos con un pipe:

```
$ tr -c [:alnum:] [\\n\*] < entrada1.txt | sort
1
2
2
2
3
3
3
3
B
B
B
dos
dos
dos
tres
tres
tres
tres
uno
X
Y
Y
Y
Y
```

El tercer comando (`uniq -c`) filtra las líneas de repetidas adyacentes de su entrada. La opción -c añade un prefijo a las líneas de salida indicando el número de ocurrencias. Comprobamos su funcionamiento utilizando como entrada las salida del segundo comando:

```
$ tr -c [:alnum:] [\\n\*] < entrada1.txt | sort | uniq -c
      1 1
      3 2
      4 3
      3 B
      3 dos
      4 tres
      1 uno
      1 X
      4 Y
```

El cuarto comando es otro sort (`sort -nr`). Compara según el valor numérico de la cadena (opción `-n`, ordenación numérica). Devuelve en orden inverso (descendente) el resultado de las comparaciones debido a la opción `-r`.

```
$ tr -c [:alnum:] [\\n\*] < entrada1.txt | sort | uniq -c| sort -nr
      4 Y
      4 tres
      4 3
      3 dos
      3 B
      3 2
      1 X
      1 uno
      1 1
```

Por último, el quinto comando (`head -$2`) imprima las primeras $2 líneas de su entrada a la salida estándar. Siendo $2 el segundo argumento del script. Comprobamos su funcionamiento añadiéndolo al pipeline, usaremos la opción `-4` para quedarnos solo con las 4 primeras líneas.

```
$ tr -c [:alnum:] [\\n\*] < entrada1.txt | sort | uniq -c| sort -nr | head -4
      4 Y
      4 tres
      4 3
      3 dos
```

Por lo tanto, el propósito del script es ver las $2 palabras (alfanuméricas) más frecuentes en el fichero de entrada $1, ordenadas según su número de ocurrencias. Las palabras con el mismo número de ocurrencias aparecen en orden alfabético inverso.

3. Ejercicio 5

(a) Compruebe el tamaño de `exercise5.sh`. ¿Qué ha ocurrido?

El tamaño pasa de ser 409 bytes a 54,6 megabytes. Lo que ha ocurrido es que el comando `tar` almacena varios ficheros en un mismo archivo. Este archivo se escribe al final del fichero `exercise5.sh`. Por lo tanto, las 7 primeras líneas son el código y las líneas en adelante son el archivo codificado en base 64.

(b) Intente ejecutar el script. ¿Qué hace? ¿Por qué es necesario el `exit 0;`? ¿Y el `2>&1` de la línea 5? Explique detalladamente cada línea del código dado.

Al intentar ejecutar el script, este muestra una ventana de diálogo donde seleccionar las carpetas del archivo. El script restaura las carpetas que se hubieran seleccionado, en el directorio `$HOME`.

El comando `exit 0;` de la linea 7 es necesario ya que las siguientes líneas del script no son código en bash y no deben ejecutarse, son el archivo codificado.

El `2>&1` de la línea 5 es necesario ya que el comando `pv` utiliza la salida de errores para mostrar el progreso. El comando `zenity` necesita los valores numéricos del progreso en su entrada estándar, al estar conectado con un pipe, su entrada será la salida del comando anterior. Por esta razón la salida estándar del comando anterior deberá ser el progreso dado por el comando `pv`, debido a esto, se redirige la salida de errores (`2>`) a la salida estándar (`&1`).

A continuación se explica cada línea del código.

La linea 2 guarda en la variable `WHAT` el resultado de la elección de la lista mostrada en la caja de `zenity`. Si se eligen distintos valores de la lista, estos se concatenan con el carácter `'|'`. Debido a esto, es necesario sustituir estos caracteres por espacios en blanco, lo que se consigue con el comando `tr \| \ .`

La linea 3 guarda en la variable `SIZE` el tamaño en bytes del propio script completo, incluida la porción que no es código.

La linea un 4 es el principio de un if, cuya condición es que el tamaño del string `WHAT` (opciones elegidas del diálogo de zenity) es mayor que 0, es decir, si se ha elegido al menos una opción.

La linea 5 contiene varios comandos en un pipeline.
El primero (`tail -n +8 $0`) devuelve desde la octava linea del script `ejercicio5.sh`, esto es, el archivo tar creado a partir del comando especificado en el enunciado.

El siguiente (`pv -n -s $SIZE -i 0.25`) muestra el progreso de los datos através de un pipeline. La opción `-n` hace que la salida sea numérica, en lugar de la barra de progreso visual, por defecto. La opción `-s` indica que el tamaño máximo será el contenido de la variable `SIZE` (para calcular porcentajes), `-i` indica que se actualice cada 0.25 segundos (4 veces por segundo).

El comando `base64 -d` decodifica los datos de su entrada, que será la salida del comando `tail`.

El comando `tar -xzf - $WHAT` extrae los ficheros especificados en la variable `WHAT` del archivo en su entrada.

Por último, el comando `zenity` muestra una caja de dialogo de tipo barra de progreso utilizando como datos la salida del comando `pv`. La opción `--time-remaining` daba el siguiente error: `Esta opción no está disponible. Por favor vea --help para todos los usos posibles.` por lo que se optó por borrarla, lo que solucionaba el problema.