Sistemas Operativos

Prácticas curso 2021-2022

Programación de shell-scripts
Apuntes de la práctica 1

Redirección de entrada/salida en la ejecución de programas

1. Redirección de entrada

Hace que la entrada estándar de un programa sea el fichero indicado.

Sintaxis:

<programa> **<** <ruta_fichero_lectura>

Ejemplo:

```
ls -l < entrada.txt
```

2A. Redirección de salida

Hace que la salida estándar de un programa sea el fichero indicado

Sintaxis:

<programa> **>** <ruta_fichero>

Ejemplo:

```
ls -l > salida.txt
```

2B. Redirección de salida doble

Igual que 2A pero en lugar de sobrescribir el fichero añade la información al final del mismo.

Sintaxis:

<programa> **>>** <ruta_fichero>

Ejemplo:

```
echo "Añadiendo texto" palabra fichero.txt >> mensaje.txt
```

Comunicación de programas mediante redirecciones y ficheros

Se trata de hacer una operación compleja uniendo la ejecución de varias órdenes unas detrás de otras y pasando mediante ficheros la salida de una orden a la entrada de la orden siguiente.

Ejemplo:

```
grep palabra fichero.txt > filtro.txt
sort < filtro.txt
```

Este ejemplo buscaría "palabra" en el fichero "fichero.txt" mediante el comando **grep**. Este comando mostrará a la salida estándar todas las líneas del fichero que contengan la búsqueda. Como hemos redireccionado la salida esas líneas se irán grabando en el fichero "filtro.txt". Posteriormente ordenamos dichas líneas mediante el comando **sort** y redireccionando su entrada estándar con el fichero "filtro.txt"

Problemas de comunicar de esta forma los programas:

1 El nombre del fichero debe inventarse y debemos tener cuidado de que no exista ya

2 Los programas se ejecutan en serie

3 Los comunico mediante un fichero → Memoria secundaria → Muy lento

4 Hay que borrar el fichero auxiliar creado

Comunicación de programas mediante PIPES

Soluciona los problemas vistos en el apartado anterior y es la técnica que se usa hoy en día.

Sintaxis:

[**<** <fichero_entrada>] <programa1> **|** <programa2> **|** .... **|** <programaN> [**>** <fichero>]

Cada pipe, comunica directamente la salida estándar del programa anterior a la pipe, con la entrada estándar del programa siguiente. Es decir, la salida de un programa se pasa directamente a la entrada del programa siguiente.

Nótese que el primer programa tiene la entrada estándar libre (por defecto el teclado) y por tanto se le puede hacer una redirección de entrada a un fichero para que lea de ahí la información.

Nótese que el último programa tiene la salida estándar libre (por defecto la consola) y por tanto se le puede hacer una redirección de salida a un fichero para que escriba ahí la información final.

Ejemplo (mismo ejemplo que en el apartado anterior pero ahora con pipes):

```
grep palabra fichero.txt | sort
```

Ventajas de comunicar programas mediante pipes

1 Ya no hacen faltan ficheros intermedios

2 La comunicación de los datos (la pipe) está en memoria principal (RAM)

3 Los comandos se ejecutan concurrentemente

¿Qué es un shell-script?

Es un fichero de texto en el que dentro hay órdenes de ejecución de programas (muchas veces comandos del SO)

(Dentro puede haber órdenes iguales a las que se pueden dar en una terminal)

¿Quién procesa esas órdenes?

Un intérprete de comandos

Intérpretes de comandos en UNIX

sh --> Estándar POSIX (usaremos éste para todo)

bash

csh

ksh

etc

¿Cómo se ejecuta un shell-script?

PRIMERA FORMA:

<interprete> <fichero_shell_script>

SEGUNDA FORMA (hace que el Shell-script parezca un ejecutable más del operativo):

./<fichero_shell_script>

Nótese que al final se va a ejecutar con la PRIMERA FORMA pero lo va a hacer automáticamente el SO.

Condiciones para ejecutarlo de la segunda forma:

1 LA PRIMERA LINEA DEL FICHERO DEBE INDICAR EL INTÉRPRETE CON EL QUE EJECUTARLO

```
#!<ruta_absoluta_al_interprete>
```

2 EL FICHERO SHELL_SCRIPT DEBE TENER PERMISO DE EJECUCIÓN, con la orden

```
chmod +x <fichero_shell_script>
```