13.Entrada/Salida

Hasta ahora sólo se vieron comandos que imprimen mensajes. Cuando un comando imprime un mensaje, el mensaje se llama “salida”.

En este capítulo, se verán diferentes formas de salida disponibles en el shell. Además, se introducirán los mecanismos utilizados para obtener entrada de los usuarios.

Específicamente las áreas que se cubrirán son

- Salida a la pantalla
- Salida a un archivo
- Entrada desde un archivo
- Entrada del usuario

Salida

Como se vio en capítulos anteriores, la maypría de los comandos producen salida. Por ejemplo el comando

```
$ date
```

produce la fecha actual en la terminal:

```
Thu Nov 12 16:32:35 PST 1998
```

Cuando un comando produce salida que se escribe en la terminal, se dice que el programa imprimió su salida a la Salida Estándar (STDOUT).

También se vieron comandos que producen mensajes de error, como:

```
$ ls -s invalidFile invalidFile1
ls: invalidFile: No such file or directory
ls: invalidFile1: No such file or directory
```

Los mensajes de error no son impresos en, en cambio estos son impresos a un tipo especial de salida llamada Error Estándar (STDERR), la cual es reservada para mensajes de error. La mayoría de los comandos utiliza STDERR para mensajes de error y STDOUT para mensajes de información.

Se verá más adelante el uso de STDERR. En esta sección se verá el uso de STDOUT en scripts para la salida de mensajes a:

- La terminal (STDOUT)
- Un archivo
- La terminal y un archivo

Salida a la Teminal

Existen 2 comandos que se utilzan generalmente para imprimir mensajes a la terminal (STDOUT):

- echo
- printf

El comando echo es utilizado mayormente para imprimir strings que requieren formateo simple. El comando printf es la versión del shell de la función del lenguaje C, printf. Provee una alto grado de flexibilidad para formatear la salida.

Primero se verá el echo

echo

El comando más utilizado para imprimir mensajes en la terminal es el comando echo. La sintaxis es

```
echo string
```

Donde string es el mensaje que se quiere imprimir. Por ejemplo, el comando

```
$ echo Hola
```

produce la salida:

```
Hola
```

También se pueden insertar espacios:

```
$ echo Hola mundo
Hola mundo
```

Además de espacios, se pueden insertar en un string lo siguiente:

- Símbolos de puntuación
- Substitución de variables
- Formateo de secuencias de escape

Embebiendo Signos de Puntuación

Los símbolos de puntuación son utilizados por ejemplo cuando se necesita hacer una pregunta al usuario o un mensaje de error:

```
echo ¿Desea instalar?
echo Error: archivo inválido!
```

Embebiendo Substituciones de Variables

Un uso común de esta técnica es para mostrar el nombres de paths:

```
echo Su directorio home es $HOME
Su directorio home es /home/frepond
```

Formateo de secuencias de Escape

Muchas veces, se necesita formatear la salida en colunas y líneas. Usando secuencias de escape se puede formatear la salida del echo. Una secuencia de escape es una secuencia especial de caracteres que representa otro caracter. Cuando el shel encuentra una secuencia de escape, la substituye por el carácter apropiado.

El comando echo entiende varias secuencias de escape. Las más comunes son las siguientes:

Table 13.1 Secuencias de Escape para el Comando Echo

Secuencia de Escape	Descripción
\n	Imprime un caracter de nueva línea
\t	Imprime un caracter tab
\c	Imprime un string sin la nueva línea por defecto

```
$ AUTO="ferrari"
$ echo -e "El auto de Carlo es:\n\t$AUTO"
El auto de Carlo es:
        ferrari
```

Importante: el quoteado con comillas dobles y la opción -e.

printf

El comando printf es similar al comando echo, desde el punto de vista que permite imprimir mensajes a STDOUT. En la forma más básica, su uso es idéntico al echo. Por ejemplo, el siguiente comando echo:

```
$ echo "Es la Ferrari de Carlo?"
```

es idéntico al comando printf:

```
$ printf "Es la Ferrari de Carlo?\n"
```

La mayor diferencia es que el string especificado por printf debe incluir la secuencia de escape \n al final para imprimir el carácter de nueva línea. El comando echo en cambio imprime la nueva línea automáticamente.

La potencia de printf reside en la capacidad de hacer formateo complejo usando especificaciones de formato. La sintaxis básica del comando es

printf formato argumentos

Donde formato es un string que contiene 1 o más secuencias de formato y argumentos son strings que corresponden a las secuencias de formateo incluídas en formato.

Las secuencias de formato son de la forma

%[-]m.nx

El símbolo % comienza la secuencia de formateo y x idencitica el tipo de formato.

Tabla 13.2 Tipos de Secuencias de Formato

Letra	**Descripción**
s	String
c	Caracter
d	Número decimal (entero)
x	Número hexadecimal
o	Número octal
e	Número de punto flotante exponencial
f	Número de punto flotante fijo
g	Número de punto flotante compacto

Dependiendo del valor de x, los enteros m y n son interpretados de forma diferente. Usualmente m es la mínima longitud del campo y n la máxima. Si se un número de punto flotante, n es interpretado como la presición. El guión (-) justifica a izquierda el campo. Por defecto todos los campos están justificados a derecha.

```
#!/bin/sh
echo -e "File Name\tType"

for i in *;
do
        echo -e "$i\t\c"
        if [ -d $i ]; then
                echo "directory"
        elif [ -h $i ]; then
                echo "symbolic link"
```

```
      elif [ -f $i ]; then
            echo "file"
      else
            echo "unknown"
      fi
done

File Name        Type
arguments.sh     file
array-test.sh    file
dir1     directory
dir2     directory
dir3     directory
dir6     directory
echo-out.sh      file
file1.txt        file
file1.txt.ln     symbolic link
```

Como se puede ver la salida no queda bien formateada. Las columnas no están bien formateadas. Esto se podría corregir con el echo utilizando espacios y tabs, sin embargo con printf la tarea es mucho más sencilla.

```
#!/bin/sh

printf "%32s %s\n" "File Name" "File Type"

for i in *;
do
      printf "%32s " "$i"
      if [ -d "$i" ]; then
            echo "directory"
      elif [ -h "$i" ]; then
            echo "symbolic link"
      elif [ -f "$i" ]; then
            echo "file"
      else
            echo "unknown"
      fi;
done
```

En este ejemplo tomamos como tamaño máximo de nombre de archivos 32 con lo que el formato quedó %32s.

```
                       File Name File Type
                    arguments.sh file
                   array-test.sh file
                            dir1 directory
                            dir2 directory
```

```
                dir3 directory
                dir6 directory
         echo-out.sh file
           file1.txt file
        file1.txt.ln symbolic link
```

En este ejemplo las columnas están alineadas pero incorrectamente justificadas, para corregir esto basta con modificar el script cambiando el formato por %-32s.

```
#!/bin/sh

printf "%-32s %s\n" "File Name" "File Type"

for i in *;
do
	printf "%-32s " "$i"
	if [ -d "$i" ]; then
		echo "directory"
	elif [ -h "$i" ]; then
		echo "symbolic link"
	elif [ -f "$i" ]; then
		echo "file"
	else
		echo "unknown"
	fi;
done
```

Ahora la salida es formateada correctamente:

```
File Name                        File Type
arguments.sh                     file
array-test.sh                    file
dir1                             directory
dir2                             directory
dir3                             directory
dir6                             directory
echo-out.sh                      file
file1.txt                        file
file1.txt.ln                     symbolic link
```

Redirección de Salida

Cuando se desarrolla un script, hay veces que se necesita capturar la salida de un comando y guardarla en un archivo.

En UNIX, el proceso de capturar la salida de un comando y guardarla en un archivo se llama redirección de salida porque redirecciona la salida de un

comando a un archivo en lugar de la pantalla. Para redireccionar la salida de un comando o script a un archivo en lugar de STDOUT, se utiliza el operador de redirección, >, de la siguiente manera:

```
command > file
list > file
```

La primera forma redirige la salida del comando command al file especificado, mientras que la segunda, redirige la salida de list a un file. Si file existe, su contenido es sobreescrito; si file no existe, se crea.

Por ejemplo el comando

```
date > now
```

redirige la salida de date al archivo now. La salida no sale en la terminal sino que se escribe en el archivo. Si se inspecciona el contenido del archivo now, se encuentra la salida de date:

```
$ cat now
Sat Nov 14 11:14:01 PST 1998
```

También se puede redireccionar la salida de una lista de comandos de la siguiente manera:

```
{ date; uptime; who ; } > mylog
```

En este caso la salida de los 3 comandos es redirigida al archivo mylog.

Agregando a un Archivo

Existen casos donde simplemente sobreescribir un archivo no es deseable. Para estos casos el shell provee una segunda forma de redirección con el operador >>, que agrega la salida a un archivo. La sintaxis básica es

```
command >> file
list >> file
```

De esta forma, la salida es agregada al archivo especificado, o es creado si no existe. Por ejemplo se puede guardar el contenido de mylog cada vez que una fecha es agregada, usando el siguiente comando:

```
{ date; uptime; who ; } >> mylog
```

Si se ve el contenido de mylog, se puede ver que ahora contiene todas las salidas de las listas ejecutadas:

```
Thu Sep 20 00:04:41 ART 2007
```

```
 00:04:41 up  3:52,  2 users,  load average: 0.01, 0.05, 0.07
frepond  tty1         2007-09-19 20:13
frepond  :0           2007-09-19 20:15
Thu Sep 20 00:04:44 ART 2007
 00:04:44 up  3:52,  2 users,  load average: 0.01, 0.05, 0.07
frepond  tty1         2007-09-19 20:13
frepond  :0           2007-09-19 20:15
```

Redirección de Salida a Archivos y Pantalla

En algunos casos, se necesita dirigir la salida de un script tanto a un archivo como a una terminal. Un ejemplo de esto son scripts que deben producir un log de sus actividades.

Para scripts interactivos, algunas salidas deberían también mostrarse en pantalla al usaurio además de loguearse a un archivo.

Para redirigir la salida a un archivo y pantalla, se usa el comando tee. La sintaxis básica es la siguiente:

command | tee file

Donde command es el nombre de un comando, como ls, y file es el nombre del archivo donde se quiere escribir la salida. Por ejemplo, el comando

$ date | tee now

produces la siguiente salida en la terminal:

Thu Sep 10 00:17:04 ART 2007

La misma salida es guardada en el archivo now.

Entrada

Muchos programas UNIX son interactivos y leen la entrada del usuario. Para utilizar estos programas en scripts, se debe proveer la entrada en forma no interactiva. También, hay veces donde se debe pedir que el usuario ingrese datos para ejecutar los comandos correctamente.

Para proveer entrada a programas interactivos o leer entrada del usuario, se necesita utilizar redirección de entrada.

- Redireción de entrada desde un archivo
- Lectura de la entrada de un usuario
- Redirección de la salida de un comando a la entrada de otro

Redirección de Entrada

Cuando se necesita usar un comando interactivo como mail en un script, se necesita proveer entrada al comando. Un método para hacer esto es guardando la entrada en un archivo y después indicarle al comando que tome la entrada de dicho archivo. Esto se consigue utilizando redirección de salida.

La entrada puede ser redirigida de forma similar a la redirección de salida:

comando < archivo

El contenido de archivo se convierte en la entrada de comando. Por ejemplo, el siguiente podría ser un ejemplo de redirección:

mailx tpi-siop@unq.edu.ar < Enunciado_con_respuestas_2do_parcial

En este ejemplo la entrada del comando mailx, el cuerpo del mensaje, es el archivo Enunciado_con_respuestas_2do_parcial.

Lectura de la Entrada del Usuario

Una tarea común en un script es pedir ingreso de datos al usuario y leer sus respuestas. Para esto, se utiliza el comando read, que permite guardar la entrada del usuario en una variable. Luego generalmente s e evalúa el valor de la variable con un case.

El comando read trabaja de la siguiente manera:

read var

Lee una linea de la entrada del usuario hasta que el usaurio presiona “Enter” y asigna lo ingresado a la variable var.

Un ejemplo:

```
SN=no

printf "Desea abandonar [$SN]? "

read SN

: ${SN:=si}
case $SN in
        [sS]|[sS][iI]) exit 0 ;;
        *) echo "Continúa" ;;
esac
```

Se pide al usuario que de una respuesta y se proveyendo un valor por defecto. Luego se lee y evalúa la respuesta del usuario utilizando un case.

Un uso común de redirección en conjunto con el comando read es leer una línea a la vez de un archivo utilizando un while.

```
while read LINE
do
        : # manipular el contenido
done < archivo
```

En el cuerpo del while, se puede manipular cada línea del archivo. Un ejemplo simple de esto

```
while read LINE
do
        case $LINE in
                *root*) echo $LINE ;;
        esac
done < /etc/passwd
```

Sólo las líneas que contienen el string root en el archivo /etc/passwd son mostradas:

```
root:x:0:0:root:/root:/bin/bash
```

Pipelines

La mayoría de los comandos UNIX que están diseñados para trabajar con archivos también puede leer la entrada de STDIN. Esto permite usar un programa para filtrar la entrada de otro. Esta es una de la tareas más comunes en los scripts: tener un programa que manipula la salida de otro programa. Se puede redirigir la salida de un comando a la entrada de otro utilizando el pipeline (|), que conecta varios comandos con pipes de la siguiente manera:

```
comando1 | comando2 | ...
```

El caracter pipe, |, conecta la salida estándar de comando1 a la entrada estándar de commando2, y así sucesivamente.

A continuación algunos ejemplos del uso de pipelines:

```
tail -f /var/adm/messages | more
ps -ael | grep "$UID" | more
```

En el primer ejemplo, la salida estándar de tail es conectada a la entrada estándar de more, que permite ver la salida de una pantalla a la vez.

En el segundo ejemplo, la salida estándar de ps se conecta a la entrada estándar de grep, y la salida estándar de grep se conecta a la entrada estándar de more, así la salida de grep puede ser visualizada de a una pantalla a la vez. Por ahora, sólo se introdujo esta técnica en los siguientes capítulos se verá el uso de esta para filtros de texto.

Descriptores de Archivos

Cuando se ejecuta un comando, 3 archivos son asociados con el comando. En el shell, cada uno de estos es representado por un número entero llamado descriptor de archivo.

Los 3 archivos abiertos por cada comando son

- Entrada Estándar (STDIN), 0
- Salida Estándar (STDOUT), 1
- Error Estándar (STDERR), 2

El entero que sigue a cada uno de estos archivos es el descriptor. Generalmente estos están asociados a la terminal del usuario pero pueden redirigirse.

Asociación de un Archivo a un Descriptor

Por defecto, el shell provee 3 descriptores de archivo estándar para todo comando. También se puede asociar cualquier archivo con un descriptor utilizando el comando exec.

Asociar un archivo con un descriptor es útil cuando se necesita redirigir la salida o entrada a un archivo muchas veces pero no se quiere repetir el nombre cada vez.

Para abrir un archivo para escritura, se usa una de las siguientes formas:

```
exec n>archivo
exec n>>archivo
```

Donde n es un entero, y archivo es el nombre del archivo que se quiere abrir para escritura. Por ejemplo, el siguiente comando

```
$ exec 4>fd4.out
```

asocia el archivo fd4.out con el descriptor 4.

Para abrir un archivo para lectura, se usa la siguiente forma:

```
exec n<file
```

Redirección Entrada/Salida General

Se puede realizar entrada/salida en forma general combinando un descriptor y un operador de redirección. La forma general es

```
comando n> archivo
comando n>> archivo
```

Donde comando es el nombre de un comando, como ls, n es un descriptor de archivo (entero), y archivo es el nombre del archivo.

Por ejemplo, se puede escribir la redirección de la salida estándar en la forma general como

```
ls -la 1> ls.out
ls -la 1>> ls.out
```

La forma general de redirección de entrada es similar a la de salida:

```
command n<file
```

Un ejemplo de eso es:

```
comando 0<archivo
```

Redirección de STDOUT y STDERR a Archivos Separados

Uno de los usos más comunes de los descriptores es para redireccionar STDOUT y STDERR a archivos separados. La sintaxis básica es

```
comando 1> archivo1 2> archivo2
```

En este caso la STDOUT del comando especificado es redirigido a archivo1, y la STDERR (mensajes de error) a archivo2. El descriptor de STDOUT (1) puede omitirse.

```
comando > archivo1 2> archivo2
```

En el siguiente ejemplo, se ilustra la el uso:

```
for FILE in $FILES
do
        ln -s $FILE ./docs >> /tmp/ln.log 2> /dev/null
done
```

En este caso la STDOUT del ln es agregada al archivo /tmp/ln.log, y la STDERR es redirigida al archivo /dev/null, para descartarla.

Redirección de STDOUT y STDERR al mismo Archivo

Hay casos donde se necesita redirigir STDOUT y STDERR la mismo archivo.

```
comando > archivo 2>&1
lista > archivo 2>&1
```

En este caso STDOUT (descriptor 1) Y STDERR (descriptor 2) son redirigidos al archivo especificado. El siguiente es un ejemplo donde es necesario redirigir ambos:

```
rm -rf /tmp/my_tmp_dir > /dev/null 2>&1 ; mkdir /tmp/my_tmp_dir
```

En este ejemplo no importa el mensaje de error o la información impresa por el comando rm. Sólo interesa eliminar el directorio, por lo tanto toda la salida es redirigida a /dev/null.

Imprimir un mensaje a STDOUT

También se puede utilizar esta forma de redirección de salida para imprimir mensajes en STDERR. La sintaxis básica es

```
echo string 1>&2
printf format args 1>&2
```

Como ejemplo, supongamos que se necesita mostrar un mensaje de error si un directorio es dado en lugar de un archivo. Se puede utilizar la siguiente sentencia if:

```
if [ ! -f $FILE ] ; then echo "ERROR: $FILE no es un archivo" >&2 ; fi
```

Redirección de 2 Descriptores de Archivo

También se puede redirigir STDOUT y STDERR a un único archivo utilizando la forma general de redirección para redirigir la salida de un descriptor de archivo a otro:

n>&m

Donde n y m son descriptores. Si se hace n=2 y m=1, se consigue que STDERR sea redirigido a STDOUT. Redirigiendo STDOUT a un archivo, también se redirige STDERR.

Si m es un guión (-) en lugar de un número, el archivo correspondiente al descriptor n es cerrado. Cuando se intenta leer o escribir de un descriptor de un archivo cerrado da error.