Entrada-Salida de a Caracteres y Redirección

Esp. Ing. José María Sola, profesor

v3.0.0, 2023-05-02

Tabla de contenidos

Lista de ejemplos

1

Introducción — Shells y CLI

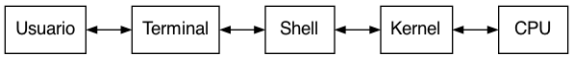
La **tecnología se crea para ser usada**, y las soluciones que aplican dispositivos tecnológicos requieren **interactuar con su entorno**. Las computadoras son una clase de dispositivo tecnológico; una clase muy amplia que, entre otros, incluye: servidores, PC de escritorio, notebooks, dispositivos móviles como celulares y tablets, consolas de videojuegos, y TV smart entre tantos otros.

Todos interactúan con su entorno, ya sea otros dispositivos o personas. La interacción tiene que darse en un marco que abstraiga el hardware y el software de base, un marco que permita el **input** de datos desde el entorno y el **output** devuelta al entorno como resultado del **procesamiento** de la computadora. Ese marco está dado por hawrdware y software especializados para el input, el output o ambos.

El hardware de I/O para el caso de la interacción con las personas, se implementa en general como un set de dispositivos que juntos permiten la entrada de datos y salida de resultados. Ejemplos de dispositivos de input: teclado, mouse, trackpad, pantalla táctil, micrófono. Ejemplos de salida de output: pantalla, parlantes, impresora. Al hardware de I/O que permite la interacción con la computadora se lo conoce como *Terminal* o *Console* (*consola*).

Al tipo de software especializado para esa interacción se lo conoce como *Shell*.

Los shells son programas que, aunque no forman parte del **kernel** del **sistema operativo**, son fundamentales para la interacción.



Forman parte del paquete software que es el sistema operativo, pero más allá de su papel fundamental, no tienen ninguna otra particularidad, es más, nosotros mismos podríamos escribir shells alternativos a los provistos por los sistemas operativos.

Hay dos tipos de shells:

- **GUI**: *Graphical User Interface* (Interfaz Gráfica de Usuario)[1].
- **CLI**: *Command-Line Interprter* (Intérprete de Línea de Comandos)[2].

La primera GUI fue creada en **Xerox** para su computadora **Alto**, y luego popularizada por la **Macintosh** de **Apple** y por **Windows** de **Microsoft**. La interacción se basa en apuntar y *clickear* -o *tapear* en el caso de pantalla táctiles- lo cual produce respuestas visuales de parte de la computadora. Requiere una terminal gráfica con hardware complejo que permita WIMP[3]. En el caso de Windows, el shell con GUI es `explorer.exe`, en Macintosh, `Finder.app`, y en sistemas basados en Unix hay varias opciones que corren sobre *X11*.

Las CLI requieren terminales con hardware mucho más simple y shells asimismo simples. Como input, un teclado es suficiente. Para el output, hardware con capacidad de presentar texto, como teletipos, impresoras, y por supuesto pantallas. En el caso de Microsofts, el shells son CLI `command.com`, `cmd.exe`, o `Powershell.exe`. En **macOS**--el sistema operativo de Macintosh—y en otros sistemas *unix-like* hay numerosos shells CLI, algunos populares son: `sh`, `csh`, `bash`, y `zsh`. Para usar las facilidades de las terminales de texto en hardware complejo como el que usamos generalmente, los sistemas operativos incluyen **emuladores de terminales**, que justamente emulan los *buffers* de teclado y pantalla, y desde donde podemos ejecutar uno o más shells tipo CLI.

Aunque con una curva de aprendizaje mayor, en algunos contextos los CLI son más eficientes y funcionales que las GUI, por eso su importancia. El objetivo de este texto este tener un primer contacto con shells tipo CLI.

[1] https://en.wikipedia.org/wiki/Graphical_user_interface
[2] https://en.wikipedia.org/wiki/Command-line_interface
[3] https://en.wikipedia.org/wiki/WIMP_(computing)

2

Anatomía y Uso del CLI

Una línea de comando arranca por un **prompt** que indica el contexto actual, y permite al usuario ingresar **comandos** que son pedidos a la computadora y **parámetros** que indica argumentos u opciones que afectan como realizar el comando:

```
prompt comando paramámetros
```

Los comandos se tipean y al pulsar enter o return se envía la solicitud de ejecución del comando a la computadora.

Ejemplo 2.1. Ejemplo de línea de comando

```
$ gcc hello.c -std=c18
```

- El prompt es `$`, indica que el shell está esperando un ingreso
- El comando es `gcc`.
- El primer parámetro es un argumento, indica con qué se va a operar.
- El segundo es una opción, indica como se va operar.

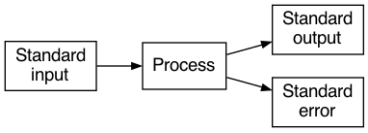
En este caso se pide que con `gcc` se compile `hello.c` siguiendo el estándar de C de 2018. El resultado va a ser la creación de un ejecutable.

Los shells de sistema operativo tienen sus peculiaridades, pero en general comparten las mismos conceptos y varias similitudes, inclusive entre shells de Microsoft y de sistemas unix-like.

3

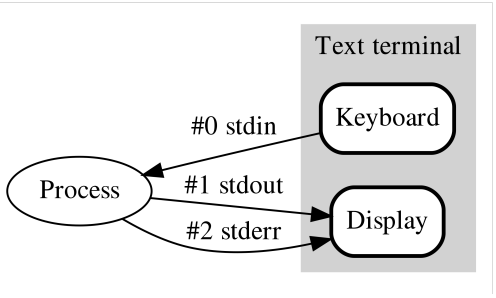
Standard Streams (Flujos Estándar)

Cuando nuestros programas se ejecutan se convierten en **procesos**. La comunicación desde y hacia un proceso puede hacerse mediante **streams** (flujos de datos). Cuando un programa inicia tiene tres **flujos estándar** ya abiertos y disponibles, todos conectados a la terminal:



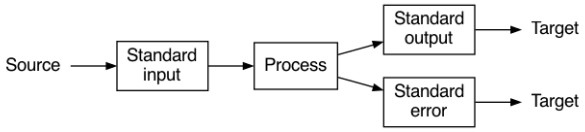
Los programas pueden abrir explícitamente otros flujos además de los estándar. Pero los programas que se basan los flujos estándar, permiten flexibilidad y familiaridad en su uso.

En C para usar esos flujos debemos incluir `<stdio>`, y se llaman `stdin`, `stdout`, y `stderr`.



Mientras que en C++ para usarlos debemos incluir `<iostream>`, y los flujos se llaman `cin`, `cout`, y `cerr`, todos son parte del *namespace* `std`.

Los shells asocian los flujos estándar a la terminal, pero permiten **redireccionar** de forma simple la entrada o la salida estándar a otros orígenes o destinos.



4

Caso de Estudio: Contar Líneas

El objetivo es comprender el uso de la terminal de texto mediante la creación y uso de un programa que cuenta líneas en su entrada presentrado en la sección "1.5.3 Conteo de Líneas" de ???.

4.1. Compilación

Vamos a escribir el programa en C y C++, *buildear* un ejecutable, y ejecutarlo para contar líneas de diferentes maneras.

Esta es la versión en C del libro:

```c
#include <stdio.h>
/* count lines in input */
main()
{
 int c, nl;
 nl = 0;

 while ((c = getchar()) != EOF)
  if (c == '\n')
   ++nl;
 printf("%d\n", nl);
}
```

Esta es una versión actualizada:

```c
// Count lines in input
#include <stdio.h>
```

```
int main(void){
 int nl=0;

 for(int c; (c = getchar()) != EOF;)
  if(c == '\n')
   ++nl;

 printf("%d\n", nl);
}
```

Ejercicio 1. Hipotetice las posibles razones de los cambios que la versión actualizada hace a la del libro.

Y esta es la versión en C++

```
// Count lines in input
#include<iostream>

int main(){
 int nl{};

 for(char c; std::cin.get(c);)
  if(c == '\n')
   ++nl;

 std::cout << nl << '\n';
}
```

Comenzamos creando el archivo `ContarLineas.c` para C ó `ContarLineas.cpp` para C++, y escribimos el código fuente del programa, utilizando cualquier editor de texto.

Luego compilamos con un comando *similar* al siguiente para C:

```
$ cc ContarLineas.c -std=c2x
```

Para C++ utilizamos un comando como el siguiente:

```
$ c++ ContarLineas.cpp -std=c++2b
```

4.2. Ejecución e Interacción con el CLI

Para ejecutarlo, ingresamos un comando similar al siguiente:

```
$ ./ContarLineas
```

A continuación, el shell le deice al sitema opertivo que inicie el programa, el proceso empieza a correr y el shell nos presenta un cursor, indicando que está a la espera de entrada desde la terminal. En ese momento comienza la interacción.

En este ejemplo de uso vamos a escribir tres líneas (i.e., renglones)

```
a
bc
def
```

Las teclas y combinaciones de teclas son las siguientes:

```
a Enter b c Enter d e f Enter ctrl+Z Enter
```

En los sistemas Microsoft la combinación ctrl+Z y luego la tecla Enter, **señaliza que no hay más datos para el flujo de entrada**; los programas Unix y sus derivados suelen utilizar ctrl+D.

La salida por pantalla esperada es

```
3
```

Es importante notar que la terminal posee un *buffer* para el dispositivo de entrda teclado, esto implica que a medida que pulsamos las teclas los caracteres se almacenan en una memoria intermedia y el programa *no* procesa la entrada hasta que recibe la indicación tratar los datos del buffer.

La forma de interactuar con la termainl varía entre los diferentes sistemas operativos, pero en general el contenido del buffer se envía al programa para que lo procese ante dos situaciones:

1. Al finalizar la línea, ó

2. Al finalizar el texto.

El fin de línea se indica pulsando Enter (o Return). Esto tiene **dos** efectos: primero, agrega al buffer un carácter *nueva línea* y, luego, hace que se envíe el contenido del buffer al proceso para que lo procese.

Como indicamos previamente, la señalización de que no hay más datos en la entrada varía entre las diferentes consolas, pero en ninguno de los casos la *señal* de fin de flujo se traduce un carácter que reciba el programa, como sí ocurre con el fin de línea.

4.3. Redirección de los Flujos Estándar

Existen situaciones donde queremos que nuestro input no sea la terminal, si no, un archivo. Esta *redirección* la indicamos desde la línea de comando anexando un símbolo de menor (<) seguida del nombre del archivo.

En este ejemplo, calculamos la cantidad de líneas que el propio código fuente de ContarLineas.c, posee:

```
$ ./ContarLineas < ContarLineas.c
12
```

Adicionalmente, si deseásemos que el output no se la terminal sea un archivo, por ejemplo resultado.txt, simplemente debemos indicarlo con un símbolo mayor (>) y el nombre del archivo.

```
$ ./ContarLineas < ContarLineas.c > resultado.txt
```

El anterior comando redirecciona la salida estándar al archivo `resultado.txt`, si no existe lo crea, si no, lo sobre escribe. En nuestro caso el contenido es un una única línea con:

```
12
```

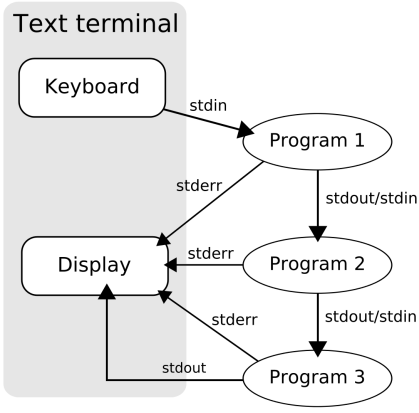
El contenido del archivo se puede visualizar de dos formas.

- Desde un shell GUI haciendo doble-click sobre el archivo haciedno click-dereceho para abrir el *menú contextual* y luego click en *open*.

- Desde un shell CLI con el comanfo `cat` para sistemas unix-like o `type` para sistemas Microsoft:

```
$ cat resultado.txt
12
```

Asimismo, la salida de un programa puede ser la entrada de otro, este control de flujo de datos se logra mediante la canalización utilizando *pipes* (tuberías) que proveen la mayoría de las consolas. El carácter que indica la conexión pipe es `|`.



Como ejemplo, el siguiente comando hace que la salida de contar la cantidad de líneas que hay en el archivo fuente actúe como la entrada a contar líneas.

```
$ ./ContarLineas < ContarLineas.c | ./ContarLineas
```

Sabemos que `./ContarLineas < ContarLineas.c` da como resultado la línea `12\n`. Luego, ese resultado se utiliza como la entrada para la segunda invocación a `ContarLineas`, así que el resultado del comando es la línea `1\n`.

5

Changelog

3.0.0+2023-05-20

- Se agregó una introducción detallada para Shells, CLIs, Terminales, y flujos estándar.
- Se agregó C++2b.
- Se actualizó a C2x.
- Se profundiza en los streams estándar.
- Se presenta el concepto de filtro.
- Se actualiza la Bibliografía.
- Se mueven los trabajos y a cada asignatura.

2.0.0+2012-03-28

- Mejoras.

1.0.0+2005-08-28

- Versión original con CC. Jorge Muchnik, profesor