



Cómo compilar y ejecutar programas en línea de comandos



Universidad Complutense
de Madrid



I.E.S. Antonio de Nebrija
(Móstoles)

<http://www.programa-me.com>

Cómo compilar y ejecutar programas en línea de comandos

En las modalidades presenciales de ProgramaMe, el entorno software que se pone a disposición de los participantes está basado en alguna distribución de GNU/Linux. Lo normal será que estén disponibles algunos entornos integrados de desarrollo, típicamente Eclipse y Netbeans, junto con múltiples editores de texto (vi, emacs, gedit, ...) ¹. En las ediciones on-line, son los propios participantes los que deciden qué software utilizan, pudiendo usar incluso sistemas operativos y entornos propietarios distintos. En ese caso, debe recordarse que el juez automático que evaluará los envíos se ejecutará sobre GNU/Linux, por lo que se deberían tener en cuenta las posibles diferencias entre los “dialectos” de los lenguajes, particularmente en C y C++.

En este documento se detalla cómo compilar programas escritos en C, C++ y Java en GNU/Linux utilizando la línea de comandos. Está especialmente orientado a los participantes de los concursos presenciales que decidan no hacer uso de ninguno de los entornos de desarrollo que se les proporcionen.

En la parte final, también se detalla cómo aprovechar los ficheros `sample.*` para probar, mínimamente, los programas realizados. Esta información puede resultar útil para todos los participantes, independientemente de la modalidad del concurso en la que participen.

Las capturas de pantalla mostradas asumen una distribución estándar basada en Gnome. No obstante, el procedimiento es independiente del escritorio que se esté usando.

1 ¿Por dónde empezar?

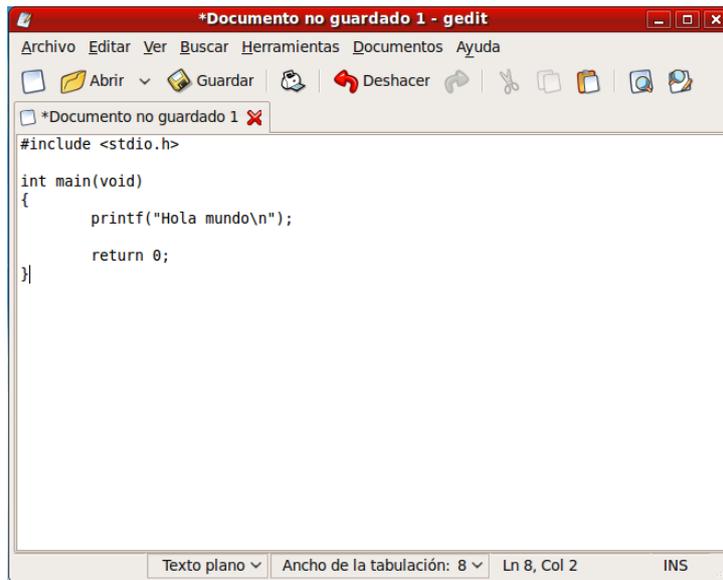
El primer paso es, naturalmente, escribir el código fuente que se quiere compilar. Si ya tienes destreza utilizando algún editor de texto (como emacs, vim, o cualquier otro), entonces puedes saltar a la siguiente sección.

Si no tienes preferencia por ningún editor, entonces consideramos que lo más sencillo es que te decantes por `gedit` (Gnome) o `Kwrite` o `Kate` (KDE) ². Búscalo en el menú de aplicaciones de tu distribución, normalmente en el apartado de Accesorios (quizá con el nombre genérico de “Editor de textos”).

Tras abrir el editor, el siguiente paso es escribir el programa. Utilizaremos como ejemplo el icónico *Hola mundo*, que se popularizara a raíz del libro de Dennis Ritchie:

¹ *Consulta* la información particular de la sede del concurso en el vayas a participar para estar seguro del software que tendrás disponible.

² Recuerda, esto es únicamente si habéis decidido no utilizar ninguno de los entornos de desarrollo (IDEs) disponibles.



The image shows a window titled "*Documento no guardado 1 - gedit". The menu bar includes "Archivo", "Editar", "Ver", "Buscar", "Herramientas", "Documentos", and "Ayuda". The toolbar contains icons for "Abrir", "Guardar", "Deshacer", and other editing functions. The main text area contains the following C code:

```
#include <stdio.h>

int main(void)
{
    printf("Hola mundo\n");
    return 0;
}
```

The status bar at the bottom indicates "Texto plano", "Ancho de la tabulación: 8", "Ln 8, Col 2", and "INS".

Tras escribir el código, graba el fichero en algún lugar. En este caso, que es un programa en C, usa la extensión `.c`. Usa `.cpp` y `.java` para cada uno de los otros dos lenguajes.

2 Ya tengo mi programa escrito ¿Y ahora qué?

Una vez tengamos escrito nuestro programa, es hora de compilarlo. Para eso, necesitamos una *consola* o *terminal*, que nos proporcione el interfaz en línea de comandos. De nuevo, busca la aplicación en el menú de aplicaciones de tu distribución. Normalmente aparecerá también en la sección de Accesorios. Al final, deberías conseguir una ventana como la siguiente:



The image shows a terminal window titled "madrid@ProgramaMe: ~". The menu bar includes "Archivo", "Editar", "Ver", "Terminal", and "Ayuda". The terminal prompt is "madrid@ProgramaMe:~\$".

Una vez abierta, utiliza el comando `cd` para moverte hasta el directorio donde guardaste tu archivo de código fuente. Cuando estés en él, escribe el comando necesario para compilarlo, que dependerá del lenguaje de programación usado:

- Si el programa lo hemos escrito en C:

```
gcc <archivo.c>
```

Ejemplo 1: compilar el programa `hola.c`, y generar el fichero ejecutable `a.out`:

```
gcc hola.c
```

Ejemplo 2: compilar el programa `hola.c`, y generar el fichero ejecutable `hola`:

```
gcc hola.c -o hola
```

- Si el programa lo hemos escrito en C++:

```
g++ <archivo.cpp>
```

Ejemplo 3: compilar el programa `hola.cpp` y generar el ejecutable `hola`:

```
g++ hola.cpp -o hola
```

- Si el programa lo hemos escrito en Java:

```
javac <archivo.java>
```

Ejemplo 4: compilar el programa (la clase) `Hola.java` y generar el fichero (ejecutable a través de la máquina virtual de Java) `Hola.class`:

```
javac Hola.java
```

Si el código fuente era correcto, se habrá generado el ejecutable y podremos pasar al siguiente punto. En otro caso, analiza los mensajes de error proporcionados por el compilador y solucionalos modificando el código fuente.

3 Mi programa ya compila, ¿qué es lo siguiente?

Ha llegado la hora de ejecutar el programa. Si escribiste el código fuente en C o C++, bastará con escribir `./a.out` (o el nombre que hayas usado con la opción `-o`, como `./hola` en el ejemplo).

Si el programa esta realizado en Java, habrá que escribir `java programa`, donde `programa` es el nombre que hemos dado a nuestro fichero de código, pero sin la extensión (java `Hola` en el ejemplo).

Con el programa en ejecución, toca probarlo introduciendo por teclado la entrada usando el formato definido en el enunciado del problema, y comprobando si la salida es la esperada.

4 ¿Cómo depuro mi programa?

En las modalidades presenciales del concurso, habitualmente tendrás disponible `gdb`, el depurador de GNU/Linux de consola³. No obstante, la depuración está fuera del alcance de este documento. Si no se conoce su uso, es preferible que los participantes se acostumbren a buscar los errores del código mediante la inclusión de código que vuelve en pantalla el estado de las variables importantes, con el fin de ir comprobando que es lo que está haciendo el programa.

5 ¿Para qué sirven los archivos `sample.in` y `sample.out`?

En los enunciados de los problemas de los concursos se proporciona siempre un ejemplo de entrada, con la salida esperada, para completar la explicación de lo que se está pidiendo. *Como mínimo*, los participantes deberían probar sus soluciones con esos casos de ejemplo.

Para eso, se puede lanzar la ejecución como se describió antes, y teclear cada una de las líneas del ejemplo mostrado en el enunciado. Sin embargo, para evitar tener que hacerlo continuamente, en los concursos suelen proporcionarse ficheros de texto con el contenido de los ejemplos que aparecen en los enunciados. En concreto, el archivo `sample.in` de un problema contiene el ejemplo de *entrada* y el fichero `sample.out` contiene la salida esperada para dicho ejemplo del enunciado. Como ejemplos, en la página web hay publicados varios `sample.*` de la sesión de prueba on-line del concurso regional de la Comunidad de Madrid del año 2011.

Para utilizar el fichero `sample.in` nos apoyamos en la *redirección de la entrada estándar* que proporcionan los interfaces en línea de comandos. Al usarla, cuando el programa al que se le está aplicando la redirección lee del teclado, lo que estará realmente haciendo es leer del fichero redirigido. Por tanto, en lugar de escribir manualmente el ejemplo, podremos utilizar:

```
./prog < sample.in
```

donde `prog` es el nombre del programa que hayamos. Si el programa está escrito en Java:

```
java prog < sample.in
```

En la pantalla veremos ahora únicamente la salida generada por el programa, que podrá comprobarse con el ejemplo de salida del enunciado.

Si la salida de ejemplo es muy larga, puede resultar tedioso comprobar si es exactamente igual que la del enunciado. Además, los espacios *no son visibles*. Si, por ejemplo, la solución escribe espacios adicionales al final de las líneas, pasará desapercibido en una comparación manual, pero el juez automático lo considerará un error.

³De nuevo, consulta la información específica del entorno software del concurso en el que vayas a participar.

Es por tanto interesante hacer también una comparación *automática* con el `sample.out`. Para eso, redirigimos también la *salida estándar* a un fichero, de modo que todo lo que escriba el programa no saldrá por pantalla, sino que terminará en disco:

```
./prog < sample.in > result.out
```

Si queremos comprobar si la salida del programa (`result.out`), es igual a la salida correcta del ejercicio `sample.out` no tendremos más que utilizar la orden `diff` para saber si hay diferencias y, de haberlas, dónde se encuentran. Si no se obtiene salida, significará que ambos ficheros son iguales, y nuestro programa ha funcionado bien con los casos de ejemplo.

Si estás probando los `sample.*` de la página web, asegúrate de utilizar los apropiados para el sistema operativo que uses. Los saltos de línea no son iguales en GNU/Linux y Windows, y si utilizas el fichero que no corresponda a tu sistema obtendrás falsos errores.

Por último, ten en cuenta que la comprobación exitosa con el ejemplo del enunciado *no* significa que la solución funcione correctamente para todas las posibles entradas. Es decir, la coincidencia entre la salida del programa y el fichero `sample.out` *no* es en medida alguna garantía de que la solución esté bien (pero sí al contrario; si el `sample.out` es distinto, entonces la solución está mal).

Los jueces automáticos utilizarán *otros casos de prueba distintos* para comprobar que el programa funciona bien. Es por eso por lo que los participantes deberían probar las soluciones que desarrollen con sus propios casos de prueba antes de enviarlos al juez automático.