TUTORIAL 4
GCC

Prof. Ing. Miguel Angel Aguilar Ulloa

TEC

LICENCIA

Usted es libre de:

✓ Copiar, distribuir y comunicar públicamente la obra.

✓ Hacer obras derivadas.

Bajo las siguientes condiciones:

✓ Reconocimiento — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

✓ No comercial — No puede utilizar esta obra para fines comerciales.

✓Compartir bajo la misma licencia — Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Texto de la licencia: http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode.

1. GCC y sintaxis.
2. Etapas de compilación.
3. Creación y enlace de bibliotecas estáticas y dinámicas.

1. GCC Y SINTAXIS

GNU Compiler Collection (GCC) - http://gcc.gnu.org/

• Es un conjunto de compiladores creados por el proyecto GNU, disponible bajo la licencia GPL.

• Lenguajes soportados: Ada, ANSI C, C++, Fortran, Java y Objective-C/C++.

• Arquitecturas Soportadas: ARM, AVR, Blackfin, MIPS, Motorola, PowerPC, SPARC, SuperH, x86.

• La sigla GCC significa "GNU Compiler Collection". Originalmente significaba "GNU C Compiler"; todavía se usa GCC para designar una compilación en C.

Sintaxis

• Las opciones van precedidas de un guión, como es habitual en UNIX, pero las opciones en sí pueden tener varias letras.

• No pueden agruparse varias opciones tras un mismo guión.

• Algunas opciones requieren después un nombre de archivo o directorio, otras no.

• Finalmente, pueden darse varios nombres de archivo a incluir en el proceso de compilación.

```
gcc [ opción | archivo ] ...
g++ [ opción | archivo ] ...
```

Algunos ejemplos rápidos

✓ Compila el programa en C **hola.c** y genera un archivo ejecutable **a.out**.

```
gcc hola.c
```

✓ Compila el programa en C **hola.c** y genera un archivo ejecutable **hola**.

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

✓ Compila el programa en C++ **hola.c** y genera un archivo ejecutable hola.

```
g++ -o hola hola.c
```

✓ No genera el ejecutable, sino el código objeto, en el archivo **hola.o**. Si no s indica un nombre para el archivo objeto, usa el nombre del archivo en C y le cambia la extensión por **.o**.

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

Algunos ejemplos rápidos

✓ Genera el código objeto indicando el nombre de archivo.

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

✓ Igual para un programa en C++.

```
g++ -c hola.cpp
```

✓ Genera el ejecutable **hola** en el subdirectorio señalado.

```
g++ -o ~/bin/hola hola.cpp
```

✓ Indica dos directorios donde han de buscarse bibliotecas. La opción **-L** debe repetirse para cada directorio de búsqueda de bibliotecas.

```
gcc -L/lib -L/usr/lib hola.cpp
```

✓Indica un directorio para buscar archivos de encabezado (de extensión .h).

```
gcc -I/usr/include hola.c
```

Sufijos en nombres de archivos

Son habituales las siguientes extensiones o sufijos de los nombres de archivo:

Extensión	Descripción
`.c`	Código fuente C
`.C .cc .cpp .c++ .cp .cxx`	Código fuente C++, se recomienda .cpp
`.m`	Código fuente Objetive-C
`.i`	C preprocesado
`.ii`	C++ preprocesado
`.S`	Fuente en ensamblador
`.o`	Código objeto
`.h`	Archivo de encabezado

Opciones principales

Opción	Descripción
`-c`	Realiza preprocesamiento y compilación, obteniendo el archivo en código objeto; no realiza el enlazado
`-E`	Realiza solamente el preprocesamiento, enviando el resultado a la salida estándar.
`-o <archivo>`	Indica el nombre del archivo de salida, cualesquiera sean las etapas cumplidas
`-I<ruta>`	Especifica la ruta hacia el directorio donde se encuentran los archivos marcados para incluir en el programa fuente. No lleva espacio entre la `I` y la ruta, así: `-I/usr/include`
`-L<ruta>`	Especifica la ruta hacia el directorio donde se encuentran los archivos de biblioteca con el código objeto de las funciones referenciadas en el programa fuente. No lleva espacio entre la `L` y la ruta, así: `-L/usr/lib`
`-Wall`	Muestra todos los mensajes de error y advertencia del compilador, incluso algunos cuestionables pero en definitiva fáciles de evitar escribiendo el código con cuidado.

Opciones principales

Opción	Descripción
`-g`	Incluye en el ejecutable generado la información necesaria para poder rastrear los errores usando un depurador, tal como GDB (GNU Debugger).
-v	Muestra los comandos ejecutados en cada etapa de compilación y la versión del compilador. Es un informe muy detallado.

2. ETAPAS DE COMPILACIÓN

1. Preprocesado

• En esta etapa se interpretan las directivas al preprocesador. Entre otras cosas, las variables inicializadas con #define son sustituidas en el código por su valor en todos los lugares donde aparece su nombre.

• El preprocesado puede pedirse con cualquiera de los siguientes comandos; **cpp** alude específicamente al preprocesador.

```
gcc -E archivo.c > archivo.pp
cpp archivo.c > archivo.pp
```

2. Compilación

• La compilación transforma el código C en el lenguaje ensamblador propio del procesador .

```
gcc -S archivo.c
```

• Lo anterior realiza las dos primeras etapas creando el archivo **archivo.S**.

3. Ensamblado

- El ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje de máquina ejecutable por el procesador.
- El ensamblador se denomina **as**:

```
as -o archivo.o archivo.S
```

- Crea el archivo en código objeto **archivo.o** a partir del archivo en lenguaje ensamblador **archivo.S**. No es frecuente realizar sólo el ensamblado; lo usual es realizar todas las etapas anteriores hasta obtener el código objeto así:

```
gcc -c archivo.c
```

- Donde se crea el archivo **archivo.o** a partir de **archivo.c**. Puede verificarse el tipo de archivo usando el comando.

```
file archivo.o
  archivo.o: ELF 32-bit LSB relocatable, Intel 80386, version 1, not stripped
```

4. Enlazado

- Las funciones de C/C++ incluidas el código, tal como **`printf()`**, se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Es preciso incorporar de algún modo el código binario de estas funciones al ejecutable que se está desarrollando. En esto consiste la etapa de enlace, donde se reúnen uno o más módulos en código objeto con el código existente en las bibliotecas.

- El enlazador se denomina **`ld`**. El comando para enlazar:

```
ld -o ejecutable archivo.o -lc
  ld: warning: cannot find entry symbol _start; defaulting to 08048184
```

- El error anterior se da por falta de referencias.

4. Enlazado

- Es necesario escribir algo como lo siguiente para obtener un ejecutable.

```
ld -o ejecutable /usr/lib/gcc-lib/i386-linux/2.95.2/collect2 -m
elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o ejecutable
/usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc-lib/i386-
linux/2.95.2/crtbegin.o -L/usr/lib/gcc-lib/i386-linux/2.95.2
archivo.o -lgcc -lc -lgcc /usr/lib/gcc-lib/i386-
linux/2.95.2/crtend.o /usr/lib/crtn.o
```

- El uso directo de ld se hace cuando se desea un mayor nivel de configuración en el ensamblado, sin embargo, simplicidad el ensamblado normalmente se hace de la siguiente manera:

```
gcc -o ejecutable archivo.o
```

Todo en un solo paso

• En programa con un único archivo fuente todo el proceso anterior puede hacerse en un solo paso:

```
gcc -o ejecutable archivo.c
```

• No se crea el archivo **archivo.o**; el código objeto intermedio se crea y destruye sin verlo el operador, pero el programa ejecutable aparece allí y funciona.

• Es instructivo usar la opción **-v** de gcc para obtener un informe detallado de todos los pasos de compilación:

```
gcc -v -o ejecutable archivo.c
```

3. CREACIÓN Y ENLACE DE BIBLIOTECAS ESTÁTICAS Y DINÁMICAS

Bibliotecas

- Es un conjunto de subprogramas utilizados para desarrollar software.

- Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de éstos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular.

- La mayoría de los sistemas operativos modernos proporcionan bibliotecas que implementan la mayoría de los servicios del sistema. De esta manera, estos servicios se convierten en una "*materia prima*" que cualquier aplicación moderna espera que el sistema operativo ofrezca.

Bibliotecas

• Los programas escritos en C/C++ usan: bibliotecas y archivos de cabecera `.h`.

• Los archivos cabecera de las Librerías de C/C++ tienen las siguientes características:

✓ Contienen declaraciones, prototipos o cabeceras de las funciones incluidas en las bibliotecas.

✓ Contienen macros y definiciones de tipos que se usan en dichas funciones.

✓ El preprocesador se encarga de procesar las directivas: `#include`, `#define`, entre otras.

✓ Las cabeceras están: `/usr/include`, entre otras ubicaciones, y las bibliotecas en sí mismas están en `/lib`, `/usr/lib`, entre otras.

Ejemplo de una biblioteca en C

Archivo cabecera: **biblioteca.h**

```
#ifndef _BIBLIOTECA_H
#define _BIBLIOTECA_H

int funcion1(int a, int b);
void funcion2(void);

#endif
```

Archivo de código fuente: **biblioteca.c**

```
int función1(int a, int b)
{
    // Ejecuta algunas acciones
     return 0;
}
void función2(void)
{
   // Ejecuta algunas acciones
}
```

Conflicto empleando bibliotecas

• Un potencial conflicto se presenta cuando en un mismo programa se incluye más de una vez el archivo de cabecera de una misma biblioteca:

✓ En `archivo1.h` existe `#include <biblioteca.h>`.

✓ En `archivo2.h` existe `#include <biblioteca.h>`.

✓ En `archivo3.c` existe `#include <archivo1.h>` y `#include <archivo2.h>`.

• Para evitar el conflicto cuando se incluye la cabecera de la biblioteca por primera vez, `_BIBLIOTECA_H` no estará definido, así que se entrará dentro del bloque `#ifndef` - `#endif` y se definirán todos los tipos y prototipos de funciones, incluido el mismo `_BIBLIOTECA_H`. Cuando lo incluyamos por segunda vez, `_BIBLIOTECA_H` ya estará definido (de la inclusión anterior), por lo que no se entrará en el bloque `#ifndef` - `#endif`, y no se redefinirá nada por segunda vez.

Enlaces estáticos y dinámicos

• Existen dos modos de realizar el enlace:

✓ ***Estático:*** los binarios de las funciones se incorporan al código binario de nuestro ejecutable.

✓ ***Dinámico:*** el código de las funciones permanece en la biblioteca; nuestro ejecutable cargará en memoria la biblioteca y ejecutará la parte de código correspondiente en el momento de correr el programa.

El enlazado dinámico permite crear un ejecutable más pequeño, pero requiere disponible el acceso a las bibliotecas en el momento de ejecutar el programa. El enlazado estático crea un programa autónomo, pero al precio de agrandar el tamaño del ejecutable binario.

Creación de una biblioteca estática

1. Se crea el código objeto de **biblioteca.c**.

```
gcc -c biblioteca.c
```

2. **ar** crea una biblioteca, aunque en realidad sirve para empaquetar cualquier tipo de archivo.

```
ar rv libbiblioteca.a biblioteca.o
```

3. **ranlib** genera un índice de los contenidos de un archivo, y lo coloca al comienzo del archivo, y que contiene una descripción de los módulos y los identificadores que va a poder resolver el enlazador sin necesidad de leer toda la biblioteca.

```
ranlib libbiblioteca.a
```

Creación de una biblioteca estática

4. Para visualizar el índice:

```
nm -s libbiblioteca.a
```

5. Generar el ejecutable, que usa una función de la biblioteca:

```
gcc -o ejecutable aplicacion.c libbiblioteca.a
```

Ubicación de cabeceras y bibliotecas

• Normalmente GCC tiene varios directorios por defecto donde busca tanto los archivos de cabecera .h como las bibliotecas. Si las bibliotecas que se crean por usuario se encuentran en directorios diferentes a los estándar se le debe indicar al compilador y enlazador dichas rutas.

```
gcc -L/home/miaguilar/lib -I /home/miaguilar/include -o
ejecutable aplicacion.c -lbiblioteca
```

• Donde **-I<ruta>** cuando los **.h** están en otra carpeta distinta a **/usr/include**; **-L<ruta>** cuando los **.a** o **.so** están en otra carpeta distinta a **/lib** o **/usr/lib** y **-lbiblioteca** indica el nombre de la librería sin el prefijo **lib** ni la extensión **.a**, ya que, el compilador las pone automáticamente.

Creación de bibliotecas dinámicas

1. Se crea el código objeto de **biblioteca.c**.

```
gcc -c biblioteca.c
```

2. Se crea la biblioteca con el enlazador.

```
ld -o libbiblioteca.so biblioteca.o  -shared
```

- La opción **-o libgeneral.so** le indica el nombre que queremos dar a la biblioteca. La opción **-shared** le indica que debe hacer una librería y no un ejecutable.

3. Una vez generada la librería, para enlazarla con el programa, se debe hacer lo siguiente:

```
gcc -o ejecutable aplicacion.c -I/home/maguilar/include -
L/home/maguilar/lib -Bdynamic libbiblioteca.so
```

Creación de bibliotecas dinámicas

4. Una vez compilado el ejecutable hay que decirle al programa, al momento de la carga del mismo, dónde están las librerías dinámicas, puesto que las va a ir a buscar cada vez que se llame a una función de ellas, para tal efecto se debe definir la variable de entorno LD_LIBRARY_PATH, de la siguiente manera.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/maguilar/lib
```