Tutorial básico de Make

Nicolás Gómez

Basado en el tutorial de Hector Urtubia
(http://mrbook.org/blog/tutorials/make/)

Archivos usados

- main.c
- hello.c
- factorial.c
- functions.h

El proceso de construcción (build)

1. El compilador toma los códigos fuente y genera los códigos objeto.
2. El enlazador toma los código objeto y genera un ejecutable.

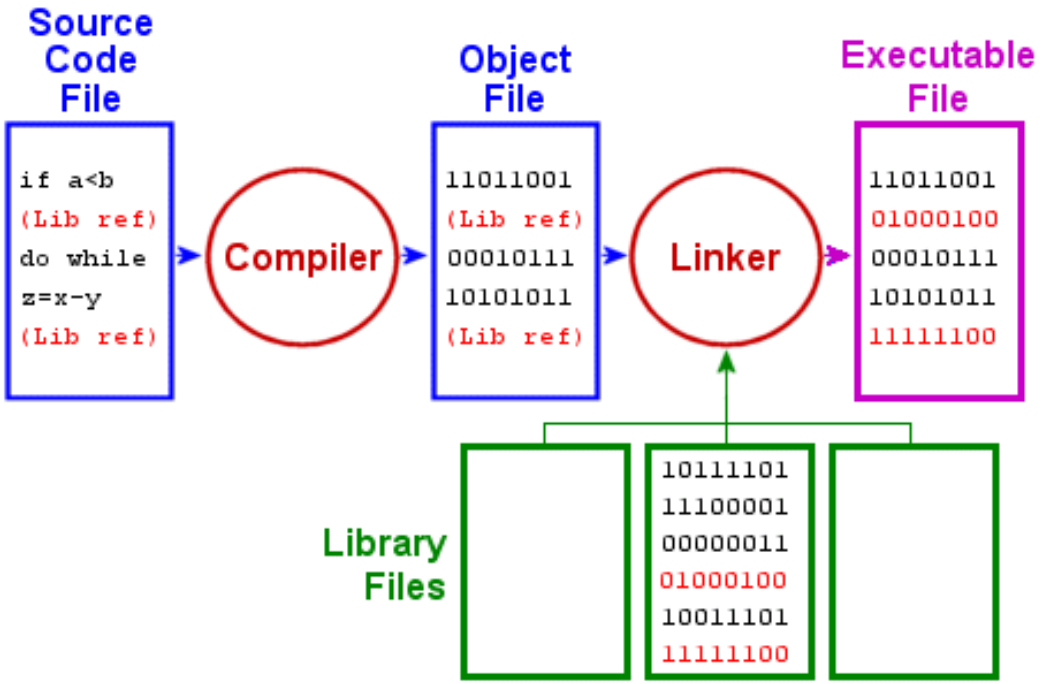

Imagen tomada de: http://www.aboutdebian.com/compile.htm

Compilando “a mano”

- Una forma rápida para compilar los archivos y obtener un ejecutable, es mediante el comando

```
gcc main.c hello.c factorial.c -o hello.x
```

  Otra forma alternativa

```
gcc -o hello.x main.c hello.c factorial.c
```

- Compilando cada archivo y enlazando todos los código objeto en un ejecutable final

```
gcc -c main.c hello.c factorial.c
gcc factorial.o hello.o main.o -o hello.x
```

La necesidad de make

- Compilar los codigos fuente puede llegar a ser **muy** tedioso, especialmente a la hora de incluir varios archivos fuente y se tiene que llamar al comando de compilación cada vez que se desee realizar dicha tarea cuando algún fichero se ha cambiado.
- Una solución para este problema es el uso de makefiles.
- Los makefiles son archivos con un formato particular, que junto con la utilidad `make` ayudan a construir y manejar proyectos de una forma “automágica”.

La necesidad de make

- Las herramientas de tipo make permiten establecer las relaciones entre ficheros de manera que sea posible determinar cuáles dependen de otros. Así, cuando detecta que alguno de los ficheros tiene una fecha y hora de modificación anterior a alguno de los que depende, se ejecuta el comando indicado para generarlos de nuevo.
- Así, no es necesario preocuparse de qué ficheros hay que generar y cuáles no hace falta actualizar. Por otra parte, evita tener que ejecutar individualmente una serie de comandos que, para programas grandes, puede ser considerable.

La utilidad make

- Al correr el comando

  ```
  make
  ```

  este buscará por un archivo llamado Makefile en el directorio actual y después lo ejecutará.
- Si se tienen varios makefiles, se pueden ejecutar con el comando

  ```
  make -f MyMakefile
  ```
- Make tiene un gran número de opciones, por lo que leer el manual de make no es mala idea

  ```
  man make
  ```

Makefiles

- Un makefile básico se compone de

```
target: dependencies
[tab] system command 1
[tab] system command 2
[tab] ...
```

- Aplicando a nuestro problema un makefile básico sería

```
all:
    gcc main.c hello.c factorial.c -o hello.x
```

Makefiles (continuación)

- Para ejecutar este makefile se debe ejecutar make

  ```
  make
  ```

- Make ejecutará por defecto el primer target a menos que se especifique otro. Para este caso, el target por defecto se se llama all, esto es algo estándar (pero no obligatorio) para makefiles.
- También se puede observar que no se usó ninguna dependencia para el target all, entonces lo ejecuta sin ninguna limitación.
- Make siempre muestra los comandos que este ejecuta acorde al Makefile.

Uso de dependencias

- Algunas veces es muy util usar diferentes targets. Esto debido a que si se modifica sólo un archivo en su proyecto, no es necesario recompilar todo, sólo lo que se modificó.

Uso de dependencias (cont.)

```
all: hello

hello: main.o factorial.o hello.o
    gcc main.o factorial.o hello.o -o hello.x

main.o: main.c
    gcc -c main.c

factorial.o: factorial.c
    gcc -c factorial.c

hello.o: hello.c
    gcc -c hello.c

clean:
    rm *o hello.x
```

Uso de variables y comentarios

```
# I am a comment, and I want to say that the variable CC will be
# the compiler to use.
CC=gcc
# Hey!, I am comment number 2. I want to say that CFLAGS will be the
# options I'll pass to the compiler.
CFLAGS=-c -Wall

all: hello

hello: main.o factorial.o hello.o
    $(CC) main.o factorial.o hello.o -o hello.x

main.o: main.c
    $(CC) $(CFLAGS) main.c

factorial.o: factorial.c
    $(CC) $(CFLAGS) factorial.c

hello.o: hello.c
    $(CC) $(CFLAGS) hello.c

clean:
    rm *o hello.x
```

Ejemplo de un Makefile “sofisticado”

```
CC=gcc
CFLAGS=-c -Wall
LDFLAGS=  #Puede estar en blanco, una opcion puede ser -lm
SOURCES=main.c hello.c factorial.c
OBJECTS=$(SOURCES:.c=.o) #Esto reemplaza el .c por .o
EXECUTABLE=hello       #El ejecutable terminara en .x

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
$(CC) $(LDFLAGS) $(OBJECTS) -o $@.x

%.o: %.c
$(CC) $(CFLAGS) $< -o $@

clean:
rm *.o *.x
```