

Creando Makefiles

Un minitutorial

Autor : Jaime López López

Creando Makefiles : Un minitutorial

Este artículo pretende ser una guía para crear tu propio archivo Makefile. Este artículo explica el porqué se necesitan archivos Makefiles y se listarán ciertos requisitos para crear un archivo Makefile.

Introducción

Imagínate que estás desarrollando un programa con nombre *foo*, que consiste de 5 cabeceras, *1.h*, *2.h*, *3.h*, *4.h* y *5.h*, y de seis archivos de código C llamados *1.cpp*, *2.cpp*, *3.cpp*, *4.cpp*, *5.cpp* y *main.cpp* (recordad, que no es recomendado utilizar este tipo de nombres para los archivos de un programa).

Supón que encuentras un bug en el archivo *2.cpp* y lo has reparado. Para conseguir un nuevo programa *foo* tienes que compilar de nuevo todos los archivos, los de cabecera y los de código, incluso cuando sólo has cambiado un archivo. Esto no es un trabajo divertido, esperando a que el ordenador termine de compilar el programa. En particular, si no tienes un ordenador rápido.

¿Qué puedes hacer entonces? ¿Hay alguna solución para este problema?

Por favor, no os preocupéis. Este tipo de problemas ya ha sido experimentado por compañeros hackers hace años. Para resolver este problema, se ha desarrollado un programa llamado *make*. En vez de construir todos los archivos de código fuente, este programa sólo construirá el archivo que se haya modificado. Si cambias el archivo *2.cpp*, *make* construirá este archivo solamente. ¿No es divertido?

Otras razones para utilizar *make* son las siguientes :

Un proyecto software que está compuesto de varios archivos de código, puede tener grandes y complejos comandos de compilación. Usando *make*, esto se ve reducido.

Los proyectos de programación algunas veces necesitan de opciones especiales del compilador que son raramente utilizadas en otras situaciones además de ser difíciles de recordar, con *make* este se ve facilitado.

Para mantenimiento de un ambiente de desarrollo consistente.

Automatizar el proceso de construcción, porque *make* puede ser llamado fácilmente desde un *script*, o una tarea *cron*.

¿Por qué necesitamos un Makefile?

Aunque *make* es muy útil, no puede hacer su trabajo sin que le demos una instrucciones nosotros. Las instrucciones de *make* son almacenadas en un archivo de texto. Este archivo es llamado Makefile y contiene comandos que deben ser procesados por *make*.

Este archivo es llamado normalmente Makefile o makefile. Como convención, los programadores llaman a su makefile, Makefile, porque es fácil de ver (si haces "ls" este archivo normalmente está en la parte alta de la lista). Si das otro nombre, asegúrate que introduces en *make* la opción *-f*.

Por ejemplo, si tenemos un makefile llamado bejo, entonces, el comando que usaremos para instruir *make* en el proceso ese archivo es :

```
make -f bejo
```

Estructura de makefile

Un makefile consiste de tres secciones : *target*, *dependencias* y *rules*. *Dependencias* son objetos o archivos de código fuente necesarios para crear un *target*; *target* es, normalmente, un archivo ejecutable o el nombre de un archivo objeto. *Rules* son comandos necesarios para crear un *target*.

Esta es una descripción de un archivo makefile :

```
target : dependencias
        command
        command
        ...
```

Un ejemplo de un archivo Makefile

Lo siguiente es un ejemplo simple de un archivo Makefile (los números de línea se han añadido para el artículo) :

```
1 client: conn.o
2     g++ client.cpp conn.o -o client
3 conn.o: conn.cpp conn.h
4     g++ -c conn.cpp -o conn.o
```

En el makefile de arriba, las *dependencias* están en la línea contenida en *client: conn.o*, mientras que *rules* están en la línea *g++ client.cpp conn.o -o client*. Fíjate que cada línea *rule* comienza con un tabulador, y no con espacios. Olvidar insertar un tabulador al principio de la línea *rule* es el error más común en la construcción de makefiles. Afortunadamente, esta clase de error es muy fácil de ver, porque el programa *make* nos lo indicará.

La descripción detallada de lo que hará el makefile anterior es la siguiente :

Crear un archivo ejecutable llamado *client* como el *target*, que depende del archivo *conn.o*

Las reglas para crear el *target* están en la línea 2.

En la tercera línea, para hacer el *target* *conn.o*, *make* necesita los archivos *conn.cpp* y *conn.h*

Las reglas para hacer el *target* *conn.o* están en la línea 4.

Comentarios

Para realizar comentarios en archivo makefile, solamente hay que poner el símbolo # en la primera columna de cada línea que sea comentada.

Debajo es un makefile de ejemplo que ha sido comentado :

```
#Crear un archivo ejecutable con nombre client
client: conn.o
    g++ client.cpp conn.o -o client

#Crear un archivo objeto "conn.o"
conn.o: conn.cpp conn.h
```

```
g++ -c conn.cpp -o conn.o
```

El target phony

El *target phony* es un nombre de archivo *fake*. Es sólo un nombre de un comando que será ejecutado cuando se realiza una petición explícita. Hay dos razones para usar un *target phony* : para evitar conflictos con archivos del mismo nombre.

Si escribes un *rule* cuyo comando no creará un archivo, estos comandos serán ejecutados cada vez que el *target* se haga. Por ejemplo :

```
clean:
    rm *.o temp
```

Debido a que el comando *rm* no creará ningún archivo llamado *clean*, este archivo nunca existirá. El comando *rm* siempre será ejecutado cada vez que realices la llamada *make clean*, porque *make* asume que el archivo *clean* es siempre nuevo.

El anterior *target* parará de trabajar si un archivo con nombre *clean* existe en el actual directorio. Debido a que no requiere de la parte *dependencies*, el archivo *clean* será considerado actualizado, y el comando “*rm *.o temp*” no será ejecutado. Para resolver este problema, puedes declarar explícitamente el *target* como *phony*, usando el comando especial *.PHONY*. Por ejemplo :

```
.PHONY : clean
```

En el makefile anterior, si no escribimos *make clean* desde la línea de comandos, el comando “*rm *.o temp*” correrá siempre, aunque exista o no un archivo con nombre *clean* en el directorio.

Variables

Para definir una variable en un archivo makefile, se usa el siguiente comando :

```
$VAR_NAME=valor
```

Como convención, un nombre de variable se escribirá en mayúsculas, por ejemplo :

```
$OBJECTS=main.o test.o
```

Para obtener el valor de una variable, sólo hay que poner el símbolo \$ antes del nombre de la variable, de esta manera :

```
$(NOMBRE_VARIABLE)
```

En un archivo makefile, hay dos tipos de variable, variables recursivamente expandidas, y variables simplemente expandidas.

En las variables recursivamente expandidas, *make* continuará expandiendo esa variable hasta que no pueda expandirse más, por ejemplo :

```
TOPDIR=/home/tedi/project
SRCDIR=$(TOPDIR)/src
```

La variable *SRC*DIR será expandida, primero expandiendo la variable *TOP*DIR. El resultado final es */home/tedi/project/src*

Pero, una variable recursivamente expandida, estos comandos estarán en una vuelta sin fin. Para estos problemas, usamos una variable simplemente expandida :

```
CC:=gcc -o
CC+=$(CC) -O2
```

El símbolo ‘:=’ crea una variable *CC* y le da un valor “*gcc -o*”. El símbolo ‘+=’ añade “*-O2*” al valor que tiene *CC*.

Comentarios finales

Espero que este pequeño tutorial sirva para dar conocimiento sobre la creación archivos makefile. Hasta entonces, feliz hackeo.

Bibliografía

GNU Make Documentation File, info make.
Kurt Wall, et.al., Linux Programming Unleashed, 2001.

Comentario de la traducción

Este documento ha sido traducido por Jaime López López. La fecha de la traducción fue el día 31/03/03. El documento original fue obtenido de la dirección electrónica <http://www.linuxgazette.com/issue83/heriyanto.html>. La página donde puedes obtener esta traducción es <http://www.dotlib.tk>.