Herramientas GNU/Linux para programadores

Jesús Manuel Mager Hois

25 de agosto de 2010

1. Introducción

GNU Compiler Collection es una colección de compiladores para casi todos los sistemas arquitecturas y sistemas operativos. Es una de las herramientas clave y fundamentales en el desarrollo de aplicaciones multiplataforma y de compilación curzada. Es desarrollada por el Proyecto GNU (GNU is Not Unix), núcleo del movimeinto del Software Libre (SoL). Esta colección soporta Ada, C, C++, Fortran, Java, Objective-C, Objective-C+ por defecto, pero se extiende a más lenguajes[1].

Es fundamental comprender esta herramienta para poder crear sistemas on casi cualquier lenguaje de programación, para cualquier tipo de computadora y cualquier sistema operativo. Nos centraremos en GNU+Linux.

Sin embargo, es necesario a su vez difundir las nuevas herramientas y nuevas tecnologías que nos permiten generar aplicaciones de manera efectiva y rápida. Tal es el caso de Python. El lenguaje de scripting es hoy en día la base de las tecnologías más poderosas como es Django Google App Engine, Google Maps, etc.

Nos sumergiremos, con una rápida introducción al mundo de la programación Python.

2. Introducción a la Programación en entornos GNU/Linux

2.1. Presentación de GCC

Como ya lo hemos mencionado, GCC es una colección de compiladores del proyecto GNU. Existe un compilador para los más diversos lenguajes, y una variedad aún más amplia en lenguajes experimentales. El potente enlazador y compiladres de la colección no se encuentran restringidos a una plataforma de Hardware ni a un sistema operativo. De esta manera podemos afirmar que GCC es el compilador estándar de los sistemas operativos tipo Unix, Mac OS X, y Android. Además, es posible compilar para sistemas Windows, PlayStation y muchos otros.

La compilación de un sencillo programa en C del tipo *hola mundo* es relativamente fácil y se lleva acabo con línea de comandos. En primer lugar expondremos el código fuente del programa a compilar.

```
#include <stdio.h>
int
main(int argc, char **argv)
 {
   printf("Hola mundo!\n");
   return 0;
 }
```

Este código, es un claro ejemplo de C estándar C99. El código es compilable en cualquier compilador estándar que soporte C99 e incluso C89. Pondremos como ejemplo una compilación y enlace del código en el archivo *hola.c* que generará el ejecutable *hola*.

```
$ gcc hola.c -o hola -Wall
```

Como podemos apreciar, el comando *gcc* es el comando que invoca el compilador de C. Si el código fuente fuese en C++, se utilizaría g++. Como primer argumento se incluye el nombre del archivo fuente, el argumento *-o* indica que el siguiente argumento será el nombre dej ejecutable, resultado del código objeto resultante, que en este caso es *hola*. *-Wall* indica al compilador que se espera ver todos los *Warnings* que aparezcan en el programa.

2.2. Utilizando GCC con Make y Automake

El procedimiento manual de compilación es relativamente fácil, sin embargo la complejidad de una compilación aumenta si se intentara compilar varios archivos fuentes, con varios ejecutables y algunas librerías incluidas. La herramienta *make* permite agilizar este método y automatizar la compilación. Con *make* es posible compilar, limpiar la compilación, instalar los ejecutables en la dirección correcta, etc... En general es un archivo script denominado *Makefile* que es interpretado por la herramienta y ejecutado. A continuación veremos un ejemplo simpe utilizando el ejemplo anterior:

```
CC=gcc
CFLAGS=-Wall

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

clean:
	rm -rf *o hola
```

2.3. Diferencias entre sistemas UNIX... e incluso Windows

Ahora bien, para poder hacer una distribución que se ajuste a cada computadora de nuestro programa es necesario utilizar un "toolchain.º cadena de herramientas que nos genere de forma automática el Makefile. De esta manera el Makefile podrá ser generado desde nuestro ./configure. Para crear el configure utilizaremos autotools.

Como es conocido por muchos, los sistemas UNIX o parecidos a el tienen gran cantidad de variantes, y cada una de estas variantes tendría que tener su propio sistema de generación Make. Sin embargo el proyecto Automake ha facilitado esto, creando una interáz de autogeneración de archivos Make y configure. Veamos como se crea esto.

Es común en los sistemas tipo Unix, la distribución de programas en forma de código, que es compilable por el usuario. Una de las grandes ventajas que tiene esta forma de distribuir es la adaptación óptima al hardware y al sistema operativo para aprovechar al máximo las características del equipo. Para poder distribuir de forma adecuada el código fuente se suelen comprimir

en archivos *tar.gz (o tarballs)* o cualquier otro tipo de compresión una serie de archivos que contienen el código fuente, otros archivos como instructivos, documentación, los archivos *Makefile* y *configure*. Se ejecutan de la siguiente forma:

```
$ ./configure
$ make
$ sudo make install
```

Los archivos *Makefile.am* y *configure.in* son archivos de entrada para automake. Un archivo básico configure.in puede ser el siguiente:

```
dnl Produce el archivo configure

AC_PREREQ(2.59)

AC_INIT([prueba], [1.0], [fongog@gmail.com])

AM_INIT_AUTOMAKE([1.9 foreign])

AC_PROG_CC
AM_PROG_LEX
AC_PROG_YACC

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Y un archivo Makefile.am se verá de la siguiente forma:

```
bin_PROGRAMS = prueba
prueba_SOURCES = main.c
prueba_LDADD = @LEXLIB@
```

Esto permitirá a autotools generar nuestros archivos configure,Makefile etc.. todo listo para que la compilación del programa se ajuste al sistema operativo sobre el cual queremos llevar acabo la compilación. Sólo necesitamos introducir las sigueintes líneas en el shell:

```
$ autoreconf --install
$ ./configure
$ make
```

Con esto no es necesario escribir un código distinto para uno u otro sistema, y podemos reciclar toda la cantidad de líneas código que queramos.

2.4. Utilizar bibliotecas gráficas (GTK)

Diversas librerías que nos permiten el uso de ambientes gráficos se encuentran disponibles no solo para sistemas GNU Linux, sino también para otros sistemas operativos. Estas librerías multiplataforma son GTK+ que es una librería bajo la licencia LGPL, la cual nos permite enlazar la librería a todo tipo de trabajos, incluidos comerciales de código cerrado y está escrita en C. Qt4 es una librería escita en C++ y tiene licencia dual, tanto en GPL (que no nos perite usarla en trabajos de código cerrado) y una licencia comercial.

```
#include <gtk/gtk.h>

static void hello( GtkWidget *widget,
                   gpointer   data )
{
   g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget,
                              GdkEvent  *event,
                              gpointer   data )
{
   g_print ("delete event occurred\n");
   return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
   gtk_main_quit ();
}

int main( int   argc,
```

```
          char *argv[] )
{

    GtkWidget *window;
    GtkWidget *button;

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    g_signal_connect (G_OBJECT (window), "delete_event",
                      G_CALLBACK (delete_event), NULL);
    g_signal_connect (G_OBJECT (window), "destroy",
                      G_CALLBACK (destroy), NULL);

    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    button = gtk_button_new_with_label ("Hello World");

    g_signal_connect (G_OBJECT (button), "clicked",
                      G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              G_OBJECT (window));

    gtk_container_add (GTK_CONTAINER (window), button);

    gtk_widget_show (button);
    gtk_widget_show (window);

    gtk_main ();

    return 0;
}
```

2.5. Programación rápida con Python

En esta aplicación pretendo introducir al usuario al desarrollo de aplicaciones de manera rápida con Python, introduciendo IDLE como interfaz de programación, enseñando algunas cuestiones básicas como son los tipos de datos disponibles, controles de flujo, todo a través del shell interactivo. A partir de aquí introduciré algunas cosas útiles como una creación de una UI.

Python es un lenguaje de alto nivel[2], útil para realizar código rápido y fácilmente comprensible. El lenguaje de programación incorpora una amplia serie de módulos que permiten trabajar en casi todas las áreas informáticas. Desde accesos a bases de datos, pasando por abstracciones orientadas a objetos de las mismas bases de datos hasta programación distribuida en clusters.

Su amplio uso, su capacidad casi ilimitada de trabajo ha hecho a este lenguaje la base de sistemas tan importantes como google. Pero sin duda alguna, nuevamente una de las mas grandes ventajas es que es un sistema multiplataforma. Una aplicación escrita en python es ejecutable tanto en Windows como en cualquier variante tipo unix, etc... El único defecto que se tiene en este lenguaje es la velocidad de ejecución en comparación con nuestro s ejemplos de C. Pero incluso eso no es problema, pues es posible utilizar rutinas que consumen mucho tiempo, desde C en Python.

Ahora bien, veamos un poco las características de este lenguaje.

```
>>> a = 1
>>> a = "Hola!"
>>> lista=[1,"Hola",a]
>>> lista.append("Adios")
```

Como hemos visto, la variable a puede representar desde un entero hasta una cadena, esto es conocido como un lenguaje sin tipos de datos. Otra gran característica es que Python cuenta con estructuras de datos optimizadas que se pueden acceder de forma completamente dinámica. Como vemos en el ejemplo anterior es posible introducir todo tipo de objetos a la lista.

```
>>> if a is not 10:
>>>     print "no es 10"
>>> for i in lista:
>>>   print i
```

En el ejemplo pasado logramos observar el comportamiento de una estructura de flujo. Esta estructura nos permite decidir el flujo que tendrá nuestro

programa. La pertenencia a una determinada estructura de flujo será determinada por la identación que tiene cada línea de código. Esto nos permite hacer programación sin corchetes. En el siguiente ejemplo utilizaremos la librería wxPython[3] para crear interfáz gráfica.

```
#!/usr/bin/env python
import wx

app = wx.App(False)
frame = wx.Frame(None, wx.ID_ANY, "Hello World")
frame.Show(True)
app.MainLoop()
```

Por último vemos como podemos crear fácilmente una ventana con muy poco código para ambientes gráficos.

Referencias

[1] http://gcc.gnu.org/

[2] http://www.python.org

[3] http://www.wxpython.org/tutorial.php