22

1

4

DD

Now
DD

Open source programming for beginners

Print Comment this

0 comments this.

by Kamal Dua

Sometimes even experienced developers can be a bit overwhelmed by Linux's extensive development capabilities. Sit back and soak up these tips to become a smarter and more productive Linux developer overnight...

This article originally appeared in issue#26 of Linux User & Developer magazine. SUBSCRIBE *Subscribe and save more than 30% and receive our exclusive money back guarantee - click here to find out more.*

This article is designed to look at a large range of tips and techniques – here's a breakdown of what this article touches on and is here:

Page 2
Running commands from C
Creating and applying patches
Conditional compilation

Page 3
Developing for the Text User Interface
Building multimedia applications

Page 4
Using an embedded SQL database

Page 5
Developing multicore applications
Utilising instead of just Making
Eclipse CDT tips

Page 6
6.Develop A Plasmoid plug-ins
Best practices for writing cross platform applications

Dividing your path
There are multiple ways by which you can program in Linux. Linux supports most of the modern types of programming environment available today. Based on your interest and objective, you can choose from the following programming methods.

Shell programming: A shell is the interface portion of Linux which provides the command-line interface for UNIX-like operating systems, including Linux. In addition to providing a command-line interface, shells are also capable of executing scripts called shell scripts. These scripts can be used for a wide variety of tasks, such as automating system administration, manipulating text files, and performing repetitive tasks. Shell programming is not meant such as math, looping variables etc, but what makes it different is the ease with which shell programming language is its ease of use and availability. Incidentally, shell programming is required for any Linux certification program, or for that matter any UNIX certification program.

Interpreted languages: A significant portion on Linux is made up of interpreted languages such as Perl, Python and Ruby. The same can also be used by you to program Linux. These languages are often dynamic and are less restrictive than the traditional programming languages, while providing the same level of functionality. Hence they became popular to many fields, such as system administration and web programming.

High-level compiled languages: They are the big boss of Linux development environments. More than 95% of the Linux OS and its applications are built using compiled languages. Linux supports most of the compiled languages – including the most popular, C and C++. Use this mode if you want to conquerer the Linux development world the good old traditional way.

This article is more inclined towards C/C++ due to the enormous open source adoption.

Finding programming languages supported by GCC
GCC is the GNU Project's C and C++ compiler. It compiles that is also native to the Linux programming environment. As the name suggests, GCC is not just a compiler for a language but rather a compiler collection that supports many programming languages. Depending upon the specific Linux distribution you are using, GCC's support for languages could vary. To find out which languages are supported, perform the following command:

```
$ gcc -v
Using built-in specs.
Target: i686-linux-gnu
Configured with: ../src/configure -v --with-pkgversion=Ubuntu 4.4.3-4ubuntu5 --with-bugurl=file:///usr/share/doc/gcc-4.4/ubuntu/bugs --enable-languages=c,c++,fortran,objc,obj-c++ --prefix=/usr --enable-shared --enable-multilib --enable-linker-build-id --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.4 --program-suffix=-4.4 --enable-nls --enable-clocale=gnu --enable-libstdcxx-debug --enable-plugin --enable-objc-gc --enable-targets=all --disable-werror --with-arch-32=i686 --with-tune=generic --enable-checking=release --build=i686-linux-gnu --host=i686-linux-gnu --target=i686-linux-gnu
Thread model: posix
gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)
```

This command gives a bunch of information, including the configure options used when building this compiler. "--enable-languages" indicates the languages supported by this compiler. In this case it is "--enable-languages=c,c++,fortran,objc,obj-c++", so the supported languages are C, C++, FORTAN, Objective C and Objective C++.

Coloured GCC output
GCC provides excellent information when your code is not compiling or linking properly. But when you are compiling a large project, GCC's output may not be very helpful. The colleges tool can come very handy in such cases. It is a wrapper for the compiler and linker that adds colour to the output of GCC, make, ld, source manager, system errors and linking errors to ease longer compiler outputs. Search for 'colleges' in your distribution's package manager, or you can download it from http://sublimity.de/colleges/ 1.3.2 tar. If you want to use colleges for all of your GCC based projects, you will need to set an alias mapping GCC to colleges.

Be alarmed
By default GCC prints a rather restrictive set of warnings and errors that inform you about the potential and final issues with your code. To get an even better perspective of the code, you can increase the level of warnings generated by GCC, by using the option '-Wall'.

For example, for the following code :

```
@code Systems =
//
//Include rand.h:
//Include rand.h.in
void main()
{
 printf ("Files in directory are /\n");
 system("ls -l");
}

when compiled without -Wall option
kamal@ubuntu:~/srcdaw:~/tipsf gcc system.c -o system
system.c:4: error: expected '=', ',', ';', 'asm' or '__attribute__' before '{' token
system.c:5: error: expected ';' before '}' token
kamal@ubuntu:~/srcdaw:~/tipsf gcc -Wall system.c -o system
system.c:4: warning: implicit declaration of built-in function 'printf'
system.c:5: warning: incompatible implicit declaration of built-in function 'printf'
system.c:5: warning: control reaches end of non-void function
kamal@ubuntu:~/srcdaw:~/tipsf gcc -Wall system.c -o system
system.c:4: warning: implicit declaration of function 'printf'
system.c:5: warning: incompatible implicit declaration of built-in function 'printf'
system.c:5: warning: control reaches end of non-void function
kamal@ubuntu:~/srcdaw:~/tipsf gcc -Wall system.c -o system
system.c:4: warning: implicit declaration of function 'printf'
system.c:5: warning: incompatible implicit declaration of built-in function 'printf'
system.c:5: warning: implicit declaration of function 'system'
```

Be disciplined
Experienced programmers will tell you that ignoring warnings is a bad practice. One way to get rid of warnings from your code is to pay attention to the warnings and fix the code. But if you get too lazy, this may method may not work. In such cases, GCC provides a option '-Werror' that can enable to treat warnings as errors. Hence, the compiler won't even compile your program until you make your code warning-free. This compiler forces a discipline of writing warning-free code.
To treat warnings as errors, you can use the GCC option '-Werror'.

Example:

```
kamal@ubuntu:~/srcdaw:~/tipsf gcc -Wall -Werror system.c -o system
system.c:4: error: implicit declaration of function 'printf'
system.c:4: error: return type of 'main' is not 'int'
system.c:5: in function 'main'
system.c:5: error: implicit declaration of function 'printf'
system.c:5: error: incompatible implicit declaration of built-in function 'printf'
system.c:5: error: implicit declaration of function 'system'
```

Follow us on twitter

Tags: developer, features, Linux, open source
Did you enjoy this article? Share it!

0 people like this.

Follow us on Twitter to find out about all the latest Linux news, reviews, previews, interviews, features and a whole more.

2 Comments

- Top-end
  Personally, I'd hesitate telling a beginning programmer to use the system() function call. But if I mentioned it, I'd at least explain that it could have grave system-security implications.

  For instance, the command that the system() function call executes may not at the same privileges as the program containing the system() function call. This could be bad news if your program somehow gets "root" privileges (via the Set User ID or Set Group ID settings, or otherwise).

  Next, the program containing the system() function call may not execute the command that you think it will. For instance, using your example ("system("ls -l");"), if a cracker can somehow get you to execute your program with a bogus executable file called "ls" in the current directory, then what's actually executed will be what the cracker intended, but not what you expected. Probably not the best example, but you get the idea. Using a full path name in the parameter for the system() function call may help mitigate this problem, at the possible expense of cross-compatibility, but not necessarily eliminate it.

  Admittedly, there's a time and place to use any of the hundreds of Linux function calls, but a good programmer will (or should) be aware of the implications of those that they choose.

  # 16 November 2010 at 12:02 am

Trackbacks

- a free open source programming for beginners - ubfuzzy

What's your opinion?

Add your comment below, or trackback from your own site. You can also subscribe to these comments via RSS.

Be nice. Keep it clean. Stay on topic. No spam.

Name (required)

Mail (will not be published) (required)

Website (optional)

Post Comment

Search for:

First Name: FIRST NAME...

Last Name: LAST NAME...

Email Address: EMAIL ADDRESS...

Subscribe

- **Recent Posts**
  - Issues in the real world – part 2
  - Some basic programming for beginners
  - Software freedom guaranteed by Judge Gray
  - Review: Ubuntu Linux 10.10 (Maverick Meerkat)
  - Review: Mandriva Linux 2010.1 (Henri Rousseau)
- **Categories**
  - Articles
  - Certifications
  - Features
  - Interviews
  - Opinion
  - Reviews
  - Tutorials
  - Uncategorized
  - Ubuntu Linux
  - Contacts
  - Forum
  - Subscribe
  - Advertising
  - Manuscripts
  - Contact Us
- **Archives**
- **Most recent articles**
  - Issues in the real world – part 2
  - Some basic programming for beginners
  - Software freedom guaranteed by Judge Gray
  - Review: Ubuntu Linux 10.10 (Maverick Meerkat)
  - Review: Mandriva Linux 2010.1 (Henri Rousseau)
- **Most popular articles**
  - Shortcut access to your favorite Ubuntu/Linux apps
  - Ubuntu 10.10 beta review
  - ATHEME vs KSM: which is ready for you?
  - Ubuntu 10.04 vs Fedora 12
  - Server management for your to power users

ADVERTISE | PRIVACY POLICY

IMAGINE PUBLISHING | IMAGINE NEWS | IMAGINE SUBS | MAGBOX
CALENDAR | IMAGINE BLOGS | IMAGINE NETWORK | IMAGINE PRINT SUBS

Copyright © 2005 - 2010 Imagine Publishing Ltd. All rights reserved

Imagine Publishing Ltd Bournemouth House, 33 Richmond Hill, Bournemouth, Dorset, BH2 6EZ

Registered company: 3174017 (England) VAT No 864 6042 18
Directors: Damian Bell, Steven Bryd, Mark Kendall, Alistair Ramsay, Harry Shand