

## Compiling for Linux

Matthew Newton, writing in PCWorld.

Let me start with a controversial statement: Installing new software is almost always easier on Linux than on Windows or the Mac OS.

I can already envision the angry e-mail. It'll come from the folks who write each month, certain that if they use enough capital letters and exclamation points, they'll convince me that Linux sucks.

But I'll say it again: Installing new software is, in most cases, easier under Linux than under other operating systems. I've touched on the simple reason why many times in Free Agent. On most Linux systems, an app called the package manager takes care of software installation and removal.

Package managers have become both more powerful and more friendly in recent years; under Ubuntu Linux, if I want to install an instant messenger, a DVD burning tool, or pretty much anything else, I don't start by searching the Web or a software downloads site, as there's no need. I open my package manager (Applications, Add/Remove... in the Ubuntu menu bar at the top of my screen) and search for the software I want. When I find the right application, the package manager downloads it.

The package manager also downloads any system components the new app requires. Remember, Free Software is an ecosystem where all the code is shared, so new apps are almost always built atop existing foundational layers. The package manager tracks all those layers and the complex relationships among them, grabs everything required, and installs the whole kit and caboodle. All the heavy lifting happens behind the scenes, and after the magical orange smoke disappears, your "new" app is ready to go.

### **Getting the Latest Software**

It all sounds great, and for the most part, it is. But Linux's way of doing things has one shortcoming. An example will illustrate it best. The trusty ThinkPad I'm writing this on is running Ubuntu 6.06, the "Dapper Drake" release:

["Meet the Dapper Drake"](#)

Rhythmbox version 0.9.3.1 had been the default app on this machine for managing and listening to my digital music. Though it wasn't a bad tool, the latest version, 0.9.6, came out recently with new features I desired. I wanted it.

Herein lay the problem: My package manager knew of only version 0.9.3.1, which had been tested and specially crafted (by a so-called package maintainer

with the Ubuntu project) to rock on Ubuntu 6.06. In other words, 0.9.3.1 is the only version of Rhythmbox that Ubuntu 6.06 officially supports, so it's the only version I could grab via the package manager with point-and-click ease.

## **Not a SETUP.EXE in Sight**

In the Windows world, if I want a newer version of an app I already have, I usually just download the new version's SETUP.EXE file and run it. But the Linux world has no counterpart to SETUP.EXE. If I wanted Rhythmbox 0.9.6 on my Dapper Drake machine, I had two choices. The first was to wait for Ubuntu's next release. The entire OS, along with all of its packages, is updated twice a year. So whenever Ubuntu's Edgy Eft release (aka Ubuntu 6.10) came out (it may already be out by the time you read this), I could tell its package manager to upgrade me to the latest, greatest Rhythmbox.

The second option was to do an end-run past the package manager and perform all the dirty work myself, compiling my own copy of Rhythmbox 0.9.6. This is a far more complicated task than running a Setup.exe. Compiling apps can drive a Linux newbie to madness."

Well, much as it pains me to admit it, the reader has a point: People don't have time for this, usually. But to be fair, in the long run Linux's approach saves me time. For the most part I stick with the apps my package manager provides, and everything always Just Works. Twice a year, I get fresh new versions of every app I use, hassle-free, courtesy of the package manager. I defy anyone to argue convincingly that this doesn't kick ass.

But enough cheerleading. I'm going to demonstrate that the second option I just mentioned--custom-compiling new versions of apps you love--really isn't all that difficult once you get the hang of it.

What makes compilation difficult to learn is the wide variety of problems you can encounter. So, in an attempt to teach by example, I'll spend the rest of this Free Agent showing how I compiled and installed the latest versions of Rhythmbox and Gaim (the venerable Free Software IM client) on an Ubuntu Dapper Drake machine. The steps I list in "Fear Not the Compiler" and "Once More Unto the Breach" are very similar on other distributions, and I'll point out important differences where I can. This content is archived online because it contains a lot of code that's difficult to duplicate in e-mail newsletter form.

First off, I downloaded the source code for Rhythmbox 0.9.6. Free Software source code is almost always distributed in .tar.gz or .tar.bz2 files (the Unix equivalents of .zip files). So I downloaded rhythmbox-0.9.6.tar.gz to my desktop.

Once it arrived, I right-clicked it and selected Extract Here. Then I opened a terminal window (Applications, Accessories, Terminal) and the advanced

package management interface, Synaptic (System, Administration, Synaptic Package Manager). (On most non-Ubuntu systems, you'll have a different package manager, which may or may not have the spiffy search abilities I describe below. You may need to experiment a bit to determine how to apply these instructions to your situation.)

In Synaptic, I began by right-clicking the package named 'build-essential' and selecting Mark for Installation. Then I clicked Apply to download and install the package. (Build-essential provides the tools required to compile apps from source code; I kept it installed for convenience in the future.) I left Synaptic open, because I wasn't done with it yet.

In my terminal window, I entered `cd Desktop/rhythmbox-0.9.6` to switch to the folder I had just created. And then I entered Magical Incantation 1 of 3:

```
./configure
```

This started the configure script in the current folder. The configure script is designed to check for all the system components that your app will need as it gets built by the compiler. If the script fails to find something it's looking for, it will quit and let you know. And that's what's happened here, as configure reported the following:

```
checking for XML::Parser... configure: error: XML::Parser perl module is required for intltool
```

In plain English, this meant that I needed to install a Perl module named XML::Parser before I could compile Rhythmbox. So, I switched back to my Synaptic window, where I clicked the Search button, selected Description and Name under 'Look in', and entered "XML::Parser" as my search term.

The results included a package called libxmlparser-perl. Based on the package's description in Synaptic, this seemed to be what I was looking for, so I installed the package by right-clicking it, selecting Mark for Installation, and then clicking Apply.

I then returned to my terminal and issued Magical Incantation 1 again. (Tip: To repeat the last command in a terminal window, press the Up Arrow key and then Enter.) This time configure spit out:

```
No package 'gnome-vfs-2.0' found
```

A search for "vfs" in Synaptic turned up a package called libgnomevfs2-dev. This is what's known as a "development library." It is not uncommon for an application to be dependent on dozens of these at compile time. On most distributions,

development library packages begin with "lib" and end with "-dev" or "-devel." I installed libgnomevfs2-dev and ran configure again. Now I saw:

```
No package 'gtk+-2.0' found
No package 'libgnomeui-2.0' found
No package 'libglade-2.0' found
```

Searching in Synaptic for "gtk", "gnomeui", and "glade" revealed the packages libgtk2.0-dev, libgnomeui-dev, and libglade2-dev. I installed them and ran configure again.

```
checking for TOTEM_PLPARSER... configure: error: totem playlist parsing library
not found or too old
```

I searched Synaptic for "totem" and found the libtotem-plparser-dev development library package. I installed it and ran configure again. My readers are sharp, so no doubt you're catching the pattern by now: Configure tells you what's missing. You go fetch it via the package manager. Rinse and repeat.

Now configure said:

```
checking for GSTREAMER_0_8... checking for GSTREAMER_0_10... configure:
error: GStreamer not found, or older than version 0.8.2/0.9.7
```

Searching Synaptic for "gstreamer", I saw that I had many gstreamer packages installed, but not the libgstreamer0.10-dev development library, so I installed that and ran configure again. It spit the exact same GSTREAMER message back at me again, so I returned to Synaptic and installed one more package, libgstreamer-plugins-base0.10-dev. Now configure reported something new:

```
checking for LIBNAUTILUS_BURN... configure: error: libnautilus-burn not found
or too old
```

I located libnautilus-burn-dev in Synaptic and installed it, only to be confounded when configure returned the same error once more. This was a tad frustrating, as the message specifically complained about libnautilus-burn, which I had just installed. But peeking in Synaptic again, I noticed a related development library that wasn't installed: libnautilus-extension-dev. I installed that, and when I ran configure it moved on:

```
configure: Rhythmbox was configured with the following options:
configure: ** Tree database is enabled
configure: ** Tag writing is enabled
configure: ** Track transfer is enabled
configure: ** iPod write support is enabled
configure: ** Multimedia keys support is enabled
```

```
configure: MusicBrainz support is disabled
configure: ** GStreamer 0.10 player is enabled
configure: iPod integration disabled
configure: ** CD burning support enabled
configure: DAAP (music sharing) support is disabled
configure: libnotify support is disabled
configure: ** HAL support enabled
configure: Python support disabled
configure: ** gnome-keyring support enabled
configure: Audioscrobbler support disabled
configure: ** Separate metadata helper process enabled
configure: using internal libsexy
configure: End options
```

Configure had run its battery of tests and reported no errors. My terminal was awaiting a new command. I could have compiled Rhythmbox right then, as I'd met the minimum dependencies to do so.

Instead I looked at the features that would be disabled, due to more missing dependencies, if I were to compile then. I certainly wanted iPod support--ditto for DAAP music sharing, which would let me share music with iTunes users on my local network. In fact, I wanted all of the missing features compiled in, so it was time for one more hunting trip through Synaptic.

Searching for "ipod" turned up libgpod-dev; when I ran configure after installing that package, it reported that iPod integration was enabled. I attacked the other disabled features similarly.

The only one that really hung me up was DAAP sharing. Synaptic didn't return anything useful when I searched its site for "DAAP." I was at a loss, so I returned to Rhythmbox's site to see what it had to say about dependencies.

Lo and behold (and I should have noticed it earlier), the site specified that something called "libsoup 2.2" is required for DAAP support. A Synaptic search for "libsoup" revealed libsoup2.2-dev. After installing that, configure reported that I was ready to compile Rhythmbox with all the features I wanted enabled. So, breathing a happy sigh of relief, I typed Magical Incantation 2 of 3:

```
make
```

The compiler launched into action, scrolling a dizzying array of commands through the terminal. When you experience this, don't be intimidated just because it's all Greek to you! The compiler's simply doing its thing. Compiling an app from source code takes some time, so while the PC is working, grab yourself a beverage, mow the lawn, scrub the tub, or give Mom a call. After a while, the

compiler will quit. If all has gone well, its last lines will be devoid of the word 'error' and will look something like this:

```
make[2]: Leaving directory '/home/mnewton/Desktop/rhythmbox-0.9.6/tests'
Making all in doc
make[2]: Entering directory '/home/mnewton/Desktop/rhythmbox-0.9.6/doc'
Making all in reference
make[3]: Entering directory '/home/mnewton/Desktop/rhythmbox-
0.9.6/doc/reference'
make[3]: Nothing to be done for 'all'.
make[3]: Leaving directory '/home/mnewton/Desktop/rhythmbox-
0.9.6/doc/reference'
make[3]: Entering directory '/home/mnewton/Desktop/rhythmbox-0.9.6/doc'
make[3]: Nothing to be done for 'all-am'.
make[3]: Leaving directory '/home/mnewton/Desktop/rhythmbox-0.9.6/doc'
make[2]: Leaving directory '/home/mnewton/Desktop/rhythmbox-0.9.6/doc'
make[2]: Entering directory '/home/mnewton/Desktop/rhythmbox-0.9.6'
make[2]: Nothing to be done for 'all-am'.
make[2]: Leaving directory '/home/mnewton/Desktop/rhythmbox-0.9.6'
make[1]: Leaving directory '/home/mnewton/Desktop/rhythmbox-0.9.6'
```

This indicated a successful compilation.

Now it was time for Magical Incantation 3 of 3:

```
sudo make install
```

Since sudo told the machine I was doing an administrative task (in this case, installing software by hand), it prompted me for my password. (On non-Ubuntu Linuxes, you may need to become the root user temporarily by entering the su and providing the root user's password when prompted. Then enter make install and then exit.)

Well looky here, I had just compiled and installed Rhythmbox 0.9.6. I verified this by running Rhythmbox (Applications, Sound & Video, Rhythmbox Music Player), selecting Help, About, and checking the version number. Hooray!

A couple housekeeping items remained. First, the .tar.gz file I had downloaded to my desktop could be deleted at that time.

I also could have deleted the extracted folder on my desktop, but if I had, I would have lost any easy means to remove the application I had just installed. Here's why: When I ran sudo make install above, Rhythmbox's assorted components installed across various directories on my system (this is true of most flavors of Unix). Tracking all those parts down individually and deleting them at some later date would not be at all practical. However, if I held on to the extracted folder

(perhaps storing it somewhere under my home directory for safekeeping) and returned to it later with a terminal window, I could issue `sudo make uninstall`--and all those various files would be deleted.

Next, I wanted to repeat the exercise by compiling and installing Gaim instant messaging software, as well as introducing an alternative to Magical Incantation 3 that would make for cleaner uninstallation of my custom-compiled app later on.

#### Once More Unto the Breach

To have an easier time with Gaim, I installed a whole bunch of common development libraries when I compiled Rhythmbox. Once again, step one was to download the source code to my desktop and extract it into a folder. Then I opened a terminal window and used the `cd` command to switch to the folder I had just created:

```
cd Desktop/gaim-2.0.0beta3.1
```

And then, Magical Incantation 1:

```
./configure
```

To my happy surprise, Gaim's `configure` didn't quit on me at all, instead running straight through to the end and reporting on what I was ready to compile. But I noticed this line:

```
Build with Audio support..... : no
```

Well, that wasn't good. I expected my IM client to make noise when a message arrived--that feature was nonnegotiable! So I searched through the rest of `configure`'s messages, looking for a clue as to why audio support was disabled. I noticed that `configure` was unable to find something called "ao"; when I searched for "ao" in Synaptic, I discovered `libao-dev`, which has the title "Cross Platform Audio Output Library Development." It sounded like a definite possibility, so I installed that package and ran `configure` again to find that audio support was now ready to go.

Similarly, I noticed this:

```
Build with GtkSpell support... : no
```

GtkSpell support is what allows Gaim to provide on-the-fly spelling checking in my message windows. (GtkSpell is a system component that provides this functionality to any app that requests it. Remember up above where I described Free Software as an ecosystem? Here's a perfect example.) I wanted that, so I

searched Synaptic for "gtkspell" and found libgtkspell-dev. I installed it and ran configure again. All looked good. So, it was time for Magical Incantation 2:

```
make
```

The compiler did its thing for a good many minutes; myself, I went to mash avocados for guacamole. When I returned, I saw the compiler had finished:

```
Generating and caching the translation database
```

```
WARNING: ./po/my_MM.po is not in UTF-8 but iso-8859-1, converting...
```

```
WARNING: ./po/tr.po is not in UTF-8 but ISO-8859-9, converting...
```

```
WARNING: ./po/nb.po is not in UTF-8 but ISO-8859-1, converting...
```

```
WARNING: ./po/en_GB.po is not in UTF-8 but iso-8859-1, converting...
```

```
WARNING: ./po/it.po is not in UTF-8 but iso-8859-1, converting...
```

```
WARNING: ./po/pl.po is not in UTF-8 but ISO-8859-2, converting...
```

```
Merging translations into gaim.desktop.
```

```
make[2]: Leaving directory '/home/mnewton/Desktop/gaim-2.0.0beta3.1'
```

```
make[1]: Leaving directory '/home/mnewton/Desktop/gaim-2.0.0beta3.1'
```

Don't be put off by all the warnings; they're often par for the course with a compiler. As long as you don't see 'error', you're probably okay.

It was time to install my custom-compiled Gaim. But instead of doing so manually with `sudo make install`, I used a utility that would make the package manager aware of what I was doing. The benefit of this will become apparent in a moment.

First, I needed the utility in question. It's called `checkinstall`, and it's available in Ubuntu's "Universe" software repositories. I enabled these repositories in Synaptic by selecting Settings, Repositories, checking the box beside all four entries marked 'Community maintained (Universe)', and clicking Close. Then I scrolled to the `checkinstall` listing and installed the tool. Once I was done, I closed Synaptic, because the command I was about to issue would fail every time if I left the package manager running.

Returning to my terminal window, I entered Alternative Magical Incantation 3:

```
sudo checkinstall
```

The `checkinstall` utility will ask you a yes/no question about technical documentation; go ahead and answer with an N. It will also ask you for a description of the package you're building; in this case, I entered Gaim version 2.0 beta.

Next, `checkinstall` builds a packaged version of your custom-compiled app, and feeds that to your package manager. When the job is done, you'll see a message in your terminal, including some instructions for removing the package from the

command line at a later date. You needn't remember those instructions, however, because the package you just built will show up in Synaptic.

At this point, I opened Synaptic to find that a search for "gaim" turned up the custom package that checkinstall had just built and installed. I can remove that package from the system cleanly at a later date simply by right-clicking it, selecting Mark for Removal, and then clicking Apply.

As far as immediate housekeeping goes, I can move the downloaded source code (both the .tar.gz and the extracted folder) to the trash. And if I'm feeling superconscious of my disk-space usage, I can also fire Synaptic back up and remove all the development libraries I installed to make the compilation work; these packages are not necessary for day-to-day execution of the apps I built.

But the more of them I leave hanging around, the easier my next compilation will be. So if you find yourself needing to stay on the bleeding edge with a couple of apps, leave their development libraries installed after your first successful builds. When your apps are updated, you can just download new source code, extract it, and perform the Three Magical Incantations:

```
./configure  
make  
sudo make install OR sudo checkinstall
```

and before long, you'll be over Linux's lack of SETUP.EXEs.