Linux Journal Issue #95/March 2002

Features

XSLT Powers a New Wave of Web Applications *by Cameron Laird*
Cameron explains the mysteries of XSLT and its multiple uses.
Client-Side Web Scripting *by Marco Fioretti*
Personalize your web experience with a little Perl.
Improving the Speed of PHP Web Scripts *by Bruno Pedro*
Discover what's holding back your PHP scripts and set them free.

Indepth

Ruby *by Thomas Østerlie*
The pluses of the scripting language taking Japan by storm.
Browser Comparison *by Ralph Krause*
A look at the strengths and weaknesses of seven web browsers.

Toolbox

Take Command Configuring pppd in Linux, Part II *by Tony Mobily*
Kernel Korner Inside the Linux Packet Filter, Part II *by Gianluca Insolvibile*
At the Forge Zope Products *by Reuven M. Lerner*
Cooking with Linux Scriptwriting for ze Web and Everywhere Else *by Marcel Gagné*
GFX Film GIMP at Rhythm & Hues *by Robin Rowe*
Linux in Education Putting Linux in Classrooms around the World *by John D. Biggs*

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

XSLT Powers a New Wave of Web Applications

Cameron Laird

Issue #95, March 2002

Cameron introduces XSLT and shows why it's such a hot topic in application development.

Extensible Stylesheet Language for Transformations (XSLT) is a computing language specialized for mapping XML documents into other XML documents.

Explanation of XSLT is no small ambition. The problem has to do with variety; there are many uses for XSLT, many instances of XSLT engines and many cooperating technologies involved in XSLT application, so it's important to focus on the essentials.

Universal XML

The first XSLT essential is its Extensible Markup Language (XML) base (also see the “Glossary of XSLT Terms” Sidebar). XML is the universal data format designed to encode everything: algorithmic data, programs and documents from purchase orders to biblical translations, in any human language, on any kind of computer and operating system. XML looks like HTML except it's a bit more complicated. In fact, one of XML's design goals is to generalize HTML in a way that preserves the comfort of HTML adepts. There's even a flavor of XML called XHTML that permits direct interpretation as HTML. *Linux Journal* frequently publishes articles on different aspects of XML.

A fully XML-ized world is a simpler one, in many ways. To analyze the operation of an accounts payable department, for example, you don't need to know who reports to whom, who is due for a three-week vacation and all those other messy human details. If you can draw a diagram that shows invoices coming in and payments going out, perhaps with authorization records spawned along the way, then you have abstracted what ought to be the essential information.

This is an intoxicating insight. It promises that a system that can transform one XML document (invoice) into one or more other XML documents (payment check, authorization records) and at least organizes, and possibly solves, all meaningful organization automations. That's why XSLT seems so important now.

Readers with a background in the XML world should generalize their project experience to get a notion of XSLT's true worth. Anyone with practical knowledge of XML knows that it's only the beginning of a solution, not the miracle cure marketing brochures often make it out to be. XSLT is exactly the same: a useful and even powerful way to organize the real work of engineering applications fit for production. The idea of transforming XML documents is an important one; to see whether it's the right idea requires close attention to the technical details.

An Engine of Your Own

To start you on your XSLT career and help you get the proper feel for the language, you'll need an engine, or language processor, of your own. The most widely used are based on Java and/or are proprietary. These often are integrated into larger server products: database servers, application servers and so on.

Rather than any of these, this article presents its examples in terms of the tDOM engine. tDOM has several advantages, among which the most important are that it's available under a liberal open-source license, it's exceptionally thrifty on memory and twice as fast as competing XSLT engines in our benchmarks, its installation is quick and compact and it exposes a scriptable command mode that's convenient for instruction. Moreover, tDOM fits well in the dual-level programming style explained below, and it's robust enough to be in production use at several demanding sites already.

To set up your own copy of tDOM, see the “How to Start XSLT Programming” Sidebar. That Sidebar concludes with a first example of XSLT use, invoked as

```
tclsh8.3 xslt.tcl example1.xml example1.xsl
example1.html
```

This command line says, “Use version 8.3 of the Tcl interpreter to launch the xslt.tcl program. The xslt.tcl utility applies the example1.xsl stylesheet to the example1.xml document and produces example1.html as its output.”

Look at this first as a machine that takes example1.xml as its input and produces example1.html, which has only a couple of lines:

```
<?xml version="1.0"?>
<datum>first message</datum>
```

Think of example1.html as an expansion of this into well-formatted HTML:

```
<html><body><h1>first message</h1></body></html>
```

XML as Data and Code

If all you need is a simple HTML document like example1.html, you can write it directly or use a lightweight macro language, rather than learn XSLT. The value of XSLT begins to appear when you look at more complex examples. You can set up the XSLT transformation to generate example1.html output in a particular style, perhaps with approved fonts or boilerplate site hyperlinks and disclaimers.

XSLT uses the language of stylesheets to specify these transformations. While stylesheets were in use before XSLT's invention, this article ignores other uses and consistently abbreviates XSLT stylesheet as just stylesheet.

On one level, a stylesheet is a program. Just as

```
int main()
{
    puts("Hello.");
}
```

is the source for a C program, a stylesheet is the source for an XSLT program. A peculiarity of stylesheets, though, is that they are themselves XML documents. Rather than looking like normal computer programs (in the way C, Java and ksh do, say), XSLT source is a kind of markup text (see Listing 1).

Listing 1. example1.xsl

With a verboseness typical of XML, this says, roughly, “act as a program that pulls out <datum> elements and puts their contents in <h1> headings of well-formatted HTML.” That's how example1.html is generated.

The application that implements this XSLT interpretation is itself a Tcl program. There's little you need to learn about Tcl at this point. tDOM exposes its XSLT engine with Tcl bindings, and the xslt.tcl script simply treats command-line variables as the filenames of XML documents and passes them on to the engine.

Let's review the example invocation. In

```
tclsh8.3 xslt.tcl example1.xml example1.xsl
example1.html
```

tclsh8.3 is the name of the executable program we're launching, and xslt.tcl is a minimal Tcl script that wraps the tDOM XSLT engine. If we wanted to improve the error handling of this utility, refinement of xslt.tcl would be the natural place to start.

Running xslt.tcl creates an XSLT processor that receives three filenames. The example1.xml file is a sample XML source document. This file names the stylesheet we apply to example1.xml. The process writes the resulting output document to example1.html. Select different logical contents for example1.html by naming a different XML source, perhaps example2.xml. To change the style of the output, rewrite example1.xsl.

You've now successfully run an XSLT program. All that's left to learn are the details of XSLT as a language and how it's applied to real-world problems. Before more on the syntax and semantics of XSLT, let's look at its uses.

One Language, Many Applications

Suppose you're responsible for a web site of tens of thousands of pages. You maintain those pages in an organization-specific XML vocabulary that strips out formatting information and HTML blemishes; your documents hold only the logical content specific to each page. Visitors need HTML, of course, but you generate that automatically, along with standard headers, frames, navigation bars, footnotes and all the other decorations we've come to expect on the Web. XSLT gives you the ability to update site style instantaneously for all the thousands of managed documents. Moreover, it partitions responsibility nicely between XML content files and XSLT stylesheets, so that different specialists can collaborate effectively.

That executive-level description masks quite a bit of implementation variability. Where and when does the XSLT transformation take place? You might have a back end of XML documents, which you periodically process with a command-line XSLT interpreter to generate static HTML documents served up by a conventional web server. You might keep the XML sources in a database, from which they're retrieved either as XML, as transformed HTML or even as full-blown HTTP sessions. Various application servers, content managers and even XML databases provide each of these interfaces. Another variation is this: you might keep only sources on your server and, with the right combination of HTML extensions and browser, direct the browser itself to interpret the XSLT you pass. You can make each of these steps as dynamic as you like, with caching to improve performance, customization to match browser or reader characteristics and so on.

This multiplicity of applications makes vendor literature a challenge to read. We all adopt different styles of Java programming depending on whether we're

working on applets, servlets, beans and so on, even though all these fit the label of Java web software. Similarly, it's important to understand clearly what kind of XSLT processing different products offer.

Complex Site Development

Neil Madden, an undergraduate at the University of Nottingham, has an XSLT system tuned for especially rapid deployment and maintenance. His scheme is organized around multisection sites, authored by teams of administrators, editors and users. He uses TclKit, an innovative open-source tool that combines database and HTTP functionality in a particularly lightweight, low-maintenance package. TclKit also knows how to interpret Tcl programs, so he wraps up tDOM with standard templates into a scriptable module. With this, he begins site-specific development:

1. Design an XML document structure that captures the content of the site's data.
2. Compose XSL stylesheets to transform data to meet each client's needs.
3. Repeat steps one and two for each section that needs special requirements.
4. Add users, sections and pages.

Scripted documents encapsulate these bundles of different kinds of data (site structure, XML sources, stylesheets) and make it easy to update and deploy a working site onto a new server or partition. Madden has plans to offer not just web-based editing but also a richer, quicker GUI interface. Tcl's uniformity and scriptability make this dual porting through either web service or local GUI practical.

Well-defined module boundaries are essential to the system. Designers maintain stylesheets, administrators manage privileges and editors assign sections without collision. With all of the functionality implemented as tiny scripts that glue together reliable components, it's easy to layer on new functionality. Madden's medium-term ambitions include a Wiki collaborative bulletin board, and XSP and FOP modules for generation of high-quality presentation output. Madden proudly compares his system to Cocoon, the well-known, Apache-based, Java-coded XML publishing framework. At a fraction of the cost in lines of code, his system bests Cocoon's performance by a wide margin.

Even further along in production use of tDOM XSLT is George J. Schlitz of MediaOne. He prepares financial documents with XSLT in a mission-critical web environment. While he originally began publication with Xalan, performance requirements drove him to switch to tDOM.

The fundamental point in all this is to be on the lookout for XML-coded or XML-codable data. Chat logs, legal transcripts, printer jobs, news photographs, screen layouts, genealogical records, game states, application designs, parcel shipments, medical files and much, much more are all candidates for XML-ization. Once in that format, XSLT processing is generally the most reliable and scalable way to render the data for specific uses.

Learning XSLT

We still have a lot to learn about XSLT as a profession. As fast as its use is expanding, there remain fewer programmers competent in XSLT than, say, Object Pascal.

Another hurdle in XSLT's diffusion, along with its unconventional XML-based syntax and confusing deployment, is its functional or applicative semantics. Most computing languages that appear in *Linux Journal* are more or less procedural: Java and C programs instruct a processor to perform one operation, then another, then another. Proceduralism is wrapped up in the how of computation.

XSLT is related to Lisp in its functionality. Good XSLT programs express the “what” of a desired result. Instead of a focus on sequence-in-time, XSLT operates on whole XML documents or well-formed fragments to yield results. This is called functional (to evoke the model of mathematics), where functions turn inputs into outputs without side effects or variation in time. Moreover, mathematical functions can be composed (stacked) in combination. Typical XSLT semantics express several different transformations without specification of their sequence-in-time. Stylesheets are applied simultaneously.

XSLT has variables, but they are immutable. They can receive only one value and cannot, in particular, be looped, as in

```
for (i = 0; i < 10; i++);
```

Parametrized or repetitive operations are done through explicit recursion and iteration. The syntax of XSLT variable use is rather ugly, as it must fit within XML's constraints. In C or Java we might write

```
if (level > 20)
    code = 3;
else
    code = 5;
```

To approximate that in XSLT, we need

```
<xsl:variable name = "code">
     <xsl:choose>
         <xsl:when test = "$level &gt; 20">
             <xsl:text>3</xsl:text>
```

```
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>5</xsl:text>
            </xsl:otherwise>
        </xsl:choose
    </xsl:variable>
```

Such exercises illustrate that, while XSLT has enough abstract power to handle general problems by itself, it's often best used in a dual-programming mode. XSLT's strengths generally lie in template processing, pattern matching and sorting and grouping XML elements. Steve Ball, a principal for consultancy Zveno Pty. Ltd., does as much as practical with XSLT, then embeds it in an application with another language to handle interfaces to external systems, including filesystems and user view.

Most popular among the developers I've encountered during the last six months are Java and Tcl, although Python, C, Perl and other partner languages also manage XSLT engines more or less adequately. Moreover, XSLT also defines extensibility mechanisms that allow developers to provide new semantics within XSLT: “extension elements”, “extension functions” and “fallback processing”.

XSLT's ultimate destiny remains unclear. In this, also, it's a bit like Java. Five years ago, Java's purpose appeared to be to construct cute visual applets. As we've discovered since then, heavy-duty enterprise servers actually make better hosts for the best Java programming. We're still at an early stage in deciding when to use XSLT. ReportLab, Inc., for example, is an enterprise vendor that delivers products and services having to do with high-quality report generation to some of the largest organizations in the world, including Fidelity Investments and American Insurance Group. ReportLab founder Andy Robinson explained to me the deep experience his development team has in projects that transform XML. Each project his company has fulfilled has involved coding in a lighter-weight scripting language rather than relying on XSLT. Even though XSLT is specialized for XML transformation, his consulting teams have found it easier to use Python as a more general-purpose, but powerful language.

Acknowledgements

My special thanks to Rolf Ade, who contributes both to tDOM software and especially to my understanding of it.

How To Start XSLT Programming

XSLT Study

Resources

A Glossary of XSLT Terms

email: claird@starbase.neosoft.com

Cameron Laird is a full-time developer and vice president of Phaseit, Inc. He also writes frequently on programming topics and has published several articles during the last year on XSLT. He's currently preparing a training course on the language.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Client-Side Web Scripting

Marco Fioretti

Issue #95, March 2002

Marco shows you how to read or download only the parts that interest you from a web page.

There are many web browsers and FTP clients for Linux, all rich in features and able to satisfy all users, from command-line fanatics to 3-D multiscreen desktop addicts. They all share one common defect, however: you have to be at the keyboard to drive them. Of course, fine tools like wget can mirror a whole site while you sleep, but you still have to find the right URL first, and when it's finished you must read through every bit that was downloaded anyway.

With small, static sites, it's no big deal, but what if every day you want to download a page that is given a random URL? Or what if you don't want to read 100K of stuff just to scroll a few headlines?

Enter client-side web scripting, i.e., all the techniques that allow you to spend time only looking at web pages (or parts of them) that interest you, and only after your computer found them for you. With such scripts you could read only the traffic or weather information related to your area, download only certain pictures from a web page or automatically find the single link you need.

Mandatory Warning about Copyright and Bandwidth Issues

Besides saving time, client-side web scripting lets you learn about some important issues and teaches you some self-discipline. For one thing, doing indiscriminately what is explained here may be considered copyright infringement in some cases or may consume so much bandwidth as to cause the shutdown of your internet account or worse. On the other hand, this freedom to surf is possible only as long as web pages remain in nonproprietary languages (HTML/XML), written in nonproprietary ASCII.

Finally, many fine sites can survive and remain available at no cost only if they send out enough banners, so all this really should be applied with moderation.

What Is Available

As usual, before doing something from scratch, one should check what has already been done and reuse it, right? A quick search on Freshmeat.net for "news ticker" returns 18 projects, from Kticker to K.R.S.S to GKrellM Newsticker.

These are all very valid tools, but they only fetch news, so they won't work without changes in different cases. Furthermore, they are almost all graphical tools, not something you can run as a cron entry, maybe piping the output to some other program.

In this field, in order to scratch only your very own itch, it is almost mandatory to write something for yourself. This is also the reason why we don't present any complete solution here, but rather discuss the general methodology.

What Is Needed

The only prerequisites to take advantage of this article are to know enough Perl to put together some regular expressions and the following Perl modules: LWP::UserAgent, LWP::Simple, HTML::Parse, HTML::Element, URI::URL and Image::Grab. You can fetch these from CPAN (www.cpan.org). Remember that, even if you do not have the root password of your system (typically on your office computer), you still can install them in the directory of your choice, as explained in the Perl documentation and the relevant README files.

Everything in this article has been tested under Red Hat Linux 7.2, but after changing all absolute paths present in the code, should work on every UNIX system supporting Perl and the several external applications used.

Collecting the Basic Information

All the tasks described below, and web-client scripting in general, require that you can download and store internally for further analysis the whole content of some initial web page, its last modification date, a list of all the URLs it contains or any combination of the above. All this information can be collected with a few lines of code at the beginning of each web-client script, as shown in Listing 1.

Listing 1. Collecting the Basic Information

The code starts with the almost mandatory "use strict" directive and then loads all the required Perl modules. Once that is done, we proceed to save the whole

content of the web page in the $HTML_FILE variable via the get() method. With the instruction that follows, we save each line of the HTTP header in one element of the @HEADER array. Finally, we define an array (@ALL_URLS), and with a for() cycle, we extract and save inside it all the links contained in the original web page, making them absolute if necessary (with the abs() method). At the end of the cycle, the @ALL_URLS array will contain all the URLs found in the initial document.

A complete description of the Perl methods used in this code, and much more, can be found in the book *Web Client Programming* (see Resources).

Download Web Pages from the Command Line

After having collected all this material, we can start to use it. If you simply want to save the content of a web page on your disk for later reading, you have to add a print instruction to the original script:

```
print $HTML_FILE;
```

And then run it from your shell prompt:

```
./webscript.pl http://www.fsf.org > fsf.html
```

This will allow you to save the whole page in the local file fsf.html. Keep in mind, however, that if this is all you want, wget is a better choice (see Resources, “Downloading without a Browser”).

Save the Images Contained in a Web Page to Disk

If all the absolute URLs are already inside the @ALL_URLS array, we can download all the images with the following for() cycle:

```
foreach my $GRAPHIC_URL (grep /(gif|jpg|png)$/,
@ALL_URLS) {
  $GRAPHIC_URL =~ m/([^\/]+)$/;
  my $BASENAME = $1;
  print STDERR "SAVING  $GRAPHIC_URL
  in $BASENAME....\n";
  my $IMG = get ($GRAPHIC_URL);
  open (IMG_FILE, "> $BASENAME") ||
  die "Failed opening $BASENAME\n";
  print IMG_FILE $IMG;
  close IMG;
  }
```

The loop operates on all the URLs contained in the document ending with the .gif, .jpg or .png extension (extracted from the original array with the grep instruction). First, the regular expression finds the actual filename, defined as everything in the URL from the rightmost slash sign to the end; this should be generalized to deal with URLs hosted on those systems so twisted that even the directory separator is backward.

The result of the match is loaded in the $BASENAME variable, and the image itself is saved with the already known get() method inside $IMG. After that, we open a file with the proper name and print the whole thing inside it.

Of course, many times you will not be interested in all the images (especially because many of them usually will be advertising banners, the site logo or other uninteresting stuff). In situations like this, a simple look at the HTML source will help you figure out what sets the image you need apart from the rest. For example, you may find out that the interesting picture has a random name but is always the third one in the list. If this is the case, modify the previous loop as follows:

```
my $IMG_COUNT  = 0;
my $WANTED_IMG = 3;
foreach my $GRAPHIC_URL (grep /(gif|jpg|png)$/,
@ALL_URLS) {
        $IMG_COUNT++;
        next unless ($IMG_COUNT == $WANTED_IMG);
        # rest of loop as before.....
        last if ($IMG_COUNT == $WANTED_IMG);
        }
print "FILE NOT FOUND TODAY\n" if
($IMG_COUNT != $WANTED_IMG);
```

The first instruction in the loop increments the image counter; the second jumps to the next iteration until we reach the third picture. The “last” instruction avoids unnecessary iterations, and the one after the loop informs that the script could not perform the copy because it found less than $WANTED_IMG pictures in the source code.

If the image name is not completely random, it's even easier because you can filter directly on it in the grep instruction at the beginning:

```
foreach my $GRAPHIC_URL
(grep /(^daily(\d+).jpg)$/, @ALL_URLS) {
```

This will loop only on files whose names start with the “daily” string, followed by any number of digits (\d+) and a .jpg extension.

The two techniques can be combined at will, and much more sophisticated things are possible. If you know that the picture name is equal to the page title plus the current date expressed in the YYYYMMDD format, first extract the title:

```
$HTML_FILE =~ m/<TITLE>([^<]+)<\/TITLE>/;
my $TITLE = $1;
```

Then calculate the date:

```
my ($sec, $min, $hour, $day, $month, $year, @dummy)
= localtime(time);
$month++;      # months start at 0
$year += 1900; # Y2K-compliant, of course ;-)))
$TODAY = $year.$month.$day;
```

And finally, filter on this:

```
foreach my $GRAPHIC_URL
(grep /(^$TITLE$TODAY.jpg)$/, <@>ALL_URLS) {
```

Extract and Display Only One Specific Section of Text

Now it starts to get really interesting. Customizing your script to fetch only a certain section of the web page's text usually requires more time and effort than any other operation described here because it must be done almost from scratch on each page and repeated if the page structure changes. If you have a slow internet connection, or even a fast one but cannot slow down your MP3 downloads or net games, you rapidly will recover the time spent to prepare the script. You also will save quite a bit of money, if you (like me) still pay per minute.

You have to open and study the HTML source of the original web page to figure out which Perl regular expression filters out all and only the text you need. The Perl LWP library already provides methods to extract all the text out of the HTML code. If you only want a plain ASCII version of the whole content, go for them.

You may be tempted to let the LWP library extract the whole text from the source, and then work on it, even when you only want to extract some lines from the web page. I have found this method to be much more difficult to manage in real cases, however. Of course, the ASCII formatting makes the text immediately readable to a human, but it also throws out all the HTML markup that is so useful to tell the script which parts you want to save. The easiest example of this false start is if you want to save or display all and only the news titles, and they are marked in the source with the <H1></H1> tags. Those markers are trivial to use in a Perl regular expression, but once they are gone, it becomes much harder to make the script recognize headlines.

To demonstrate the method on a real web page, let's try to print inside our terminal all the press-release titles from the FSF page at www.fsf.org/press/press.html. Pointing our script at this URL will save all its content inside the $HTML_FILE variable. Now, let's apply to it the following sequence of regular expressions (I suggest that you also look at that page and at its source code with your browser to understand everything going on):

```
$HTML_FILE =~ s/.*>Press Releases<//gsmi;
$HTML_FILE =~ s/.*<DL>//gsmi;
$HTML_FILE =~ s/<\/DL>.*$//gsmi;
$HTML_FILE =~ s/<dt>([^<]*)<\/dt>/-> $1: /gi;
$HTML_FILE =~ s/<dd><a href=[^>]*>([^<]*)<\/a>/
$1 /gsmi;
$HTML_FILE =~
s/\.\s+\([^\)]*\.\)<\/dd>/<DD>/gsmi;
$HTML_FILE =~ s/\s+/ /gsmi;
$HTML_FILE =~ s/<DD>/\n/gsmi;
```

The first three lines cut off everything before and after the actual press-release list. The fourth one finds the date and strips the HTML tags out of it. Regexes number five and six do the same thing to the press-release subject. The last two eliminate redundant white spaces and put new lines where needed. As of December 14, 2001, the output at the shell prompt looks like this (titles have been manually cut by me for better formatting):

```
-> 3 December 2001: Stallman Receives Prestigious...
-> 22 October 2001: FSF Announces Version 21 of the...
-> 12 October 2001: Free Software Foundation
   Announces...
-> 24 September 2001: Richard Stallman and
   Eben Moglen...
-> 18 September 2001: FSF and FSMLabs come
   to agreement...
```

The set of regular expressions above is not complete; for one thing, it doesn't manage news with update sections. One also should make it as independent as possible from extra spaces inside HTML tags or changes in the color or size of some fonts. This regular expression strips out all the font markup:

```
$HTML_FILES =~ s/<font face="Verdana" size="3">
([^<]+)<\/font>/$1/g;
```

This performs the same task but works on any font type and (positive) font size:

```
$HTML_FILES =~ s/<font face="[^"]+"
size="\d+">
([^<]+)<\/font>/$1/g;
```

The example shown here, however, still is detailed enough to show the principle, and again the one-time effort to write a custom set for any given page really can save a lot of time.

Make News Appear on Your Screen

Once you have managed to extract the text you want and to format it to your taste, there is no reason to limit yourself to a manual use of the script, or to use it only at the console for that matter. If you want to do something else and be informed by the computer only when a new headline about Stallman appears, only three more steps are needed.

First, put the script among your cron entries (**man cron** will tell you everything about this). After that, add the following check to your Perl script:

```
if ($HTML_FILE =~ m/Stallman/) {
       # INFORM ME!!!
}
```

This will do what you want only if the remaining text does contain the Stallman string (or whatever else you want to know about, of course).

Next, fill the block with something like this:

```
open (XMSG, "|/usr/bin/X11/xmessage -title \"NEWS!\"
-file -") || die;
print XMSG $HTML_FILE;
close XMSG;
```

This will open a UNIX pipe to the xmessage program, which pops up a window with the title given with the corresponding switch and containing the text of the file following the -file option. In our case, "-" tells xmessage to get the text from the standard input. As it is, the Perl script will wait to exit, so that you close the xmessage window. This may or may not be what you want. In the case of a cron script, it's much better to let it start xmessage in the background on a temporary file and exit, like this:

```
open (XMSG, "> /tmp/gee") || die;
print XMSG $HTML_FILE;
close XMSG;
exec "/usr/bin/X11/xmessage -title \"NEWS!\"
-file /tmp/gee&";
```

Check to See If a Page Was Changed after a Particular Date

If you want to process the page only if the content was changed since the last visit, or in the last two hours, you need the Last-Modified HTTP header. It is already available, expressed in seconds since January 1, 1970, in the third element of our @HEADER array. Hence, if you want to do something only on pages modified in the last two hours, start calculating what the time was in that moment (always in the "seconds since..." unit):

```
$NOW = time;
$TWO_HOURS_AGO = $NOW - (3600*2);
```

Then compare that time with the modification date of the web page:

```
if ($HEADER[2] > $TWO_HOURS_AGO) {
       # do whatever is needed
}
```

Add Dynamic Bookmarks to Your Window Manager Menu

This is one of the rare exceptions to the do-it-yourself rule stated at the beginning: download WMHeadlines (see Resources), install it, and then configure and modify to suit your taste. Out of the box, it can fetch headlines from more than 120 sites and place them in the root menu of Blackbox, WindowMaker, Enlightenment and GNOME in such a way that you start your browser on the dynamic menu voice you click on.

Driving Your Browser from within a Script

Netscape can be given several commands from the prompt or any script. Such commands will cause Netscape to start if it wasn't already running or will load the requested URL in the current window, or even in a new one. However, the

commands to run change depending on whether Netscape is already running. Look at the nslaunch.pl script in the WMHeadlines distribution to figure out how to check if Netscape is already running.

You also can drive Netscape to perform other actions from a script: to print a page just as Netscape would do if driven manually, make it load the page first:

```
exec($NETSCAPE, '-noraise', '-remote',
"openURL($URL,new-window)");
```

Then save it as PostScript:

```
exec($NETSCAPE, '-noraise', '-remote',
"saveAs(/tmp/netscape.ps,
PostScript)");
```

And finally, print it:

```
exec("mpage -PYOURPRINTER -1 /tmp/netscape.ps");
```

Or, even add it to the bookmarks:

```
exec($NETSCAPE, '-noraise', '-remote',
"addBookmark($SOME_URL, $ITS_TITLE)");
```

Konqueror, the KDE web browser, can be started simply by invoking it in this way:

```
system("/usr/bin/konqueror $URL");
```

Konqueror can be driven by scripts for many nonweb-related tasks, such as copying files, starting applications and mounting devices. Type **kfmclient --commands** for more details.

Galeon can be started in an almost equal way:

```
system("/usr/bin/galeon $URL");
```

As explained in *A User's Guide to Galeon* (see Resources), you also can decide whether Galeon (if already running) should open the URL in a new tab:

```
system("/usr/bin/galeon -n $URL");
```

in a new window:

```
system("/usr/bin/galeon -w $URL");
```

or temporarily bookmark the $URL:

```
system("/usr/bin/galeon -t $URL");
```

Smart Browsing

The opposite approach, i.e., starting a generic mirroring or image-fetching script from your browser, is possible in Konqueror (or even KMail) during normal browsing. If you right click on a link and select the “Open with..” option, it will let you enter the path of the script to be used and add it to the choices next time. This means you can prepare a mirror or fetch_images script following the instructions given here and start it in the background on any URL you wish with a couple of clicks.

Smart Mirroring and FTP

The URL list contained in the @ALL_URLS array also can be used to start mirroring or (parallel) FTP sessions. This can be done entirely in Perl, using the many FTP and mirroring modules available, or simply by collecting the URLs to be mirrored or fetched by FTP, and leaving the actual work to wget or curl, as explained in A. J. Chung's article, “Downloading without a Browser” (see Resources).

If your favorite web portal chooses a different cool site every day, and you want your PC to mirror it for you, just fetch the URL as you would do for images, and then say in your script:

```
exec "wget -m -L -t 5 $COMPLETE_URL";
```

All the commands for parallel FTP and mirroring explained in Chung's article can be started in this way from a Perl script, having as arguments the URLs found by this one.

Build Your Custom Web Portal

Many of us have more than one favorite site and would like to have them all in the same window. A general solution for this is to extract the complete HTML body of each page in this way:

```
$HTML_FILE = s/^.*<body[^>]*>//i; # strips everything
                                  # before
$HTML_FILE = s/<\/body[^>]*>.*$//i; # strips everything
                                    # after
```

and then print out an HTML table with each original page in each box:

```
print<<END_TABLE;
....All HTML <HEAD> and <BODY> stuff here
<TABLE>
<TR><TD>$HTML_FILE_1</TD></TR>
<TR><TD>$HTML_FILE_2</TD></TR>
.........
</TABLE></BODY></HTML>
END_TABLE
```

Save the script output in $HOME/.myportal.html, set that file as your starting page in your browser and enjoy! The complete script will probably require quite some tweaking to clean up different CSSes, fonts and so on, but you know how to do it by now, right?

Conclusion

We have barely scratched the surface of client-side web scripting. Much more sophisticated tasks are possible, such as dealing with cookies and password-protected sites, automatic form submission, web searches with all the criteria you can think about, scanning a whole web site and displaying the ten most-pointed-to URLs in a histogram, and web-mail checking.

You only need some patience, Perl practice and a good knowledge of the relevant modules to succeed. Good browsing!

Resources

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Improving the Speed of PHP Web Scripts

Bruno Pedro

Issue #95, March 2002

Bruno shows some causes of slow web scripts and gives possible solutions.

PHP web scripts lose performance for a lot of reasons. The bottleneck can be in database queries, web page access or even slow algorithms. When performance drops, the user gets frustrated waiting for results. Less users mean less business, and your web site becomes unpopular.

The main reason for performance loss is bad software analysis and engineering. Web sites often are created and launched without thorough performance testing. Databases often are designed to accommodate less data than they actually do. Algorithms often are designed poorly and usually are not optimized for speed.

When you cannot redesign the entire web design so that it runs more quickly, you must improve its performance by serving static pages instead of interpreting PHP whenever there's a hit. Let's look at the ways to achieve this goal.

The Traditional Solution

The first thing that you can do is preprocess those PHP scripts, or script parts, that take more time to execute. You can do it with the help of a PHP shell. Suppose you have a web script called index.php, and you want to preprocess it. Assuming that the PHP shell is called phpsh, the command line is:

```
phpsh -q /some_dir/index.php > /some_dir/index.html
```

The file index.html is now plain HTML and doesn't need any PHP processing. You can serve it right away to the web client. But what if the PHP script results change over time? You'd have to preprocess the script every time the results are different. The solution is to preprocess the PHP script periodically.

Periodic Preprocessing

In Linux, the easiest way to execute a given process periodically is called crontab. The following crontab entry illustrates a preprocessing that would execute every 15 minutes:

```
*/15 * * * * root phpsh -q /some_dir/index.php
> /some_dir/index.html
```

However, the chosen timing might not be enough to keep the information updated. Furthermore, some scripts are not accessed over long periods of time, while others are constantly accessed, making the use of this technique pointless. In this case, a script-based mechanism is needed.

Just-in-Time Preprocessing

This technique preprocesses scripts periodically, but only if they are accessed. It works much like a web proxy caching system. I will show two ways of implementing this functionality: output buffering and after-time processing.

Output buffering checks the cache file date and time and only processes the script if needed. The processing is done by buffering the output and saving it in the cache file before it is sent to the client.

In PHP you do this with the help of the ob_start() function. This function will turn output buffering on and send it to a callback function. There is another function to send the buffer back to the browser: ob_end_flush(). Let's take a look at an example:

```
<?php
// include header file
include("header.php");
?>
<?php
// sleep for 10 seconds
sleep(10);
?>
```

Test:

```
<?php
// include footer file
include("footer.php");
?>
```

Inside header.php, you'll find all the cache processing. It begins by checking whether the script needs caching by calling the needscache() function. This function can check the need for cache based on a time out or based on anything you like. For the purpose of this article, the checking is based on cache time out.

If the script needs caching, the ob cycle is started, and the script output is written into the cache file. If it doesn't need caching, the script output is read from the cache file and sent to the client's browser (see Listing 1).

Listing 1. The script output is read from the cache file and sent to the client's browser.

The footer.php script simply closes the ob processing:

```
<?php
ob_end_flush();
?>
```

You can test this technique by calling the script many times before the cache times out. You'll notice that in the first call you'll have to wait ten seconds (this is because the script sleeps for ten seconds, for testing purposes), and in the following calls the output is immediate.

However, when the cache times out, you'll have to wait for the script to finish processing. Let's see how you can prevent this and give the user the illusion that the script is always fast.

After-view processing also checks the cache file date and time, but the processing is done after the file is served, solving the cache time out processing burden. This is done by caching the file after the script ends execution.

In PHP, you can do this by associating an arbitrary function with the script termination event through the register_shutdown_function(). The only file you'll have to change is header.php (see Listing 2).

Listing 2. Associating an Arbitrary Function with the Script Termination Event

I simply added the doaftercache() function that is called only after the script finishes. It then calls the script like a normal browser does and caches it. The only time you'll have to wait is when the script has never been cached before. Test it and you'll get the feeling that the script is very fast.

Conclusion

This article shows you how PHP cache mechanisms work and provides a do-it-yourself solution. If you test the examples and like them, please feel free to implement your own solutions. However, there other ways to improve performance like function caching, or PHP script precompilation. Some off-the-shelf solutions can offer you these functionalities. You should always look for the best solution for your own needs and do your own testing.

email: bpedro@eth.pt

Bruno Pedro, cofounder and manager of ethernet lda., is a systems engineer with ten years' experience in database-related applications. He was an early adopter of Linux and has been using open-source technology since then. Since 1995 he has been developing applications for the Internet.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Ruby

Thomas Østerlie

Issue #95, March 2002

Thomas demonstrates the power and flexibility of Ruby.

Ruby is a full-fledged, modern, pure, object-oriented programming language. Its syntax is terse and consistent, making Ruby both easy to read and learn, and it's flexible and expressive as well. If you're coming from a background in an API-bloated language, you will be surprised by Ruby's small but powerful core API. That Ruby is tightly integrated with the underlying operating system, and that it is ridiculously simple to extend, makes it both a powerful and versatile programming language.

Bold assertions? Let's uncover the truths behind these claims. For demonstration, I have included a simple Ruby script that purges a temp directory of files older than a given number of days. The application lets me demonstrate both basic Ruby syntax and some of the language's more important features. The entire script is included in Listing 1 [available at ftp.linuxjournal.com/pub/lj/listings/issue95/4834.tgz]. It is invoked by

```
./purge.rb [tmp_dir] [max_file_age_in_days]
```

where age determines how old a file needs to be before it is purged from the temp directory. You can add a call to this script in your crontab.

Ruby Basics

Ruby, an object-oriented language, offers encapsulation of data and methods within objects, allows inheritance from one class to another and supports polymorphism. Everything, including primitive data types like strings and integers, is represented as an object. Even constants and classes are represented as objects. This makes Ruby a pure object-oriented language. The only exception here is the control structure, a handful of expressions such as for, if, while, etc. These are not objects.

As shown in Listing 2 [available at ftp.linuxjournal.com/pub/lj/listings/issue95/4834.tgz], the delete_older method contains the top-level program logic: traverse a given directory to check for files to delete.

To those used to typed languages like Java or C++, the method parameters' missing type declarations may seem strange. But Ruby is dynamically typed. That is, a variable has no type, but the object it holds a reference to does, hence the lack of types in the declaration. Dynamic typing favors object composition over class inheritance. There is no controlling the type of objects passed as parameters in method calls, alleviating the need to worry about complex inheritance hierarchies, as we no longer depend on polymorphism to pass objects into methods. This leads to simpler, more reusable code.

Ruby's method declaration should look familiar to Python programmers. The two languages declare methods in practically the same way, including the use of optional parameters. An optional parameter can be left out when calling a method. Leaving out the parameter is the same as invoking the method with the optional parameter's default value.

Ruby's method declaration also lacks a return value. Since the language is dynamically typed, there is no need to declare a return type. Unless a return object is explicitly specified with the return statement, the last expression evaluated will be returned, as in Lisp.

A method is invoked by sending the target object a message. This is the Smalltalk way. The target.message(parameterlist) message-passing notation should be familiar to all object-oriented programmers. Sending an object a message invokes the corresponding method on the target object. All inter-object communication is handled by message passing.

Ruby operates with the notion of two kinds of methods: class methods and what is simply called methods, or instance methods. Instance methods are invoked on instantiated classes, more commonly known as objects. Class methods are called on uninstantiated classes and are like static methods in Java and C++. As a class method is called on an uninstantiated class, it may be considered a library method. It does not operate on the object's member variables.

Containers

Consider the following code the script is invoked with, which processes the command-line parameters:

```
path = ARGV.shift or raise "Missing path to delete"
age  = ARGV.shift or raise "Missing age in days"
```

ARGV is an array object containing the command-line options from the invocation of the script. Calling “shift” returns and removes array's first element. Ruby has an advanced array class. The array is dynamic; it resizes itself. It is an object, so you need not worry about memory issues and walking off its end either. Methods allowing you to process the array by index, by element and as if it were a stack, a set or a queue, are also included with the class. Arrays may be reversed and they may be sorted. For table lookups, use the Hash class.

The following line from Listing 1 shows how elegant Ruby's array is:

```
Dir.entries(full_name) - ['.', '..']).empty?
```

Dir.entries(full_name) returns an array containing all files in the directory. The array **['.', '..']** is then subtracted from the directory listing by using with the **-** operator. We can then see if the directory is empty by calling isEmpty? on the directory listing. If the array is empty, i.e., isEmpty? returns true, no other files are left in the directory.

Error Handling

Once the invocation parameters have been processed, it is time to call delete_older:

```
delete_older(path, age_in_seconds)
rescue puts "Error: #$!"
```

Errors may occur during execution. If the script is invoked with the path to a nonexistent directory, for instance, an error will occur the first time Ruby invokes a method on the Dir class. The code above not only invokes delete_older, it also handles possible errors that occur during execution. The key here is the rescue expression. When an error occurs, the Ruby interpreter packages the error in an exception object. This object propagates back up the call stack until it reaches some code that explicitly declares it knows how to handle this particular type of exception. Exceptions that are never caught propagate through the call stack, ending up with an abnormal program termination; the stack trace is printed to stderr. This is opposed to returning error codes like shell scripts and C do, leading to less-nested statements, less spaghetti logic and simply better error handling.

Including an ensure statement in connection with the rescue expression ensures that a code block is run no matter what else happens. Combine this with the possibility of writing your own exceptions, making your own code throw exceptions (with the raise expression as shown in the program listing in Containers), and the ability to actually recover from an exception by running

some code and retrying the code that caused the failure, and you have one of the neatest error-handling mechanisms I've ever used.

Advanced Features

Let's return to delete_older to look at some of Ruby's more advanced features (see Listing 2 [available at ftp.linuxjournal.com/pub/lj/listings/issue95/4834.tgz]). Line two sees “foreach” being invoked on the Dir class; foreach is an implementation of the iterator design pattern. If you are doing object-oriented programming, but have not read the Gang of Four's groundbreaking book *Design Patterns*, you'd better run out and buy a copy. Iterator is not the only pattern implemented in core Ruby. Singleton, publisher/subscriber, visitor and delegation patterns also are implemented. Other patterns also can be implemented simply if required, but the listed patterns are shipped with Ruby.

foreach iterates over the files in a directory. Following the call to Dir's foreach is a block of code with a start and end very much resembling that of a regular Java or C++ code block. The code contained within the curly braces is called a block, which is like a method within a method. A block is never executed at the time it is encountered. Whenever the foreach method has read a single file from the directory, it yields control to the block. The code within the block is executed, and control returns to foreach, which reads a new file from the directory repeating the procedure over again until no more files are left to iterate over.

Instead of having to write helper classes to make iterators work, as you have to in Java or C++, Ruby includes the yield expression that makes it possible to implement iterators as methods. This is a prime example of the language's expressiveness and flexibility. Instead of writing the scaffolding to make it happen, Ruby lets you concentrate on doing the job.

As mentioned earlier, a method is invoked by sending a message to the target object on the target.message form. Only methods local to the class may be called without specifying a target. My script calls “puts”, which is a method belonging to the kernel, without specifying any target. How does the interpreter know which method I'm calling when puts is not local to the object and no target is specified?

It is the magic of mix-ins. Mix-ins basically allow you to mix methods implemented elsewhere into a class without the use of inheritance (for more on mix-ins, please refer to the article “Using Mix-ins with Python”, *Linux Journal*, April 2001). Mix-ins aren't a new idea, nor is Ruby the only language to support it. But it is most definitely one of the features that gives Ruby that nice and clean syntax.

I could never hope to deal with all of Ruby's features in this article. Instead I'll refer you to David Thomas and Andrew Hunt's book *Programming Ruby* for more details on issues like modules, aliasing, namespaces, reflection, dynamical method calls, system hooks, program distribution and networking. It is worth mentioning that Ruby also supports regular expressions that are just as good as Perl's and supports CGI, in addition to having its own Apache module, mod_ruby.

Conclusion

Is Ruby yet another scripting language? No, it is not. It is something more, something new and exciting coming out of the Japanese open-source scene. Ruby is the programmer's best friend. While Ruby is presented as a scripting language, it has proven equally suited for large projects. It includes some exciting features that other alternative languages are only beginning to implement. Ruby is therefore well worth checking out.

Acknowledgements

A special thanks goes out to my technical reviewers: Armin Roehrl, of approximity.com, for reviewing the draft manuscript and guidance in editing the final version. David Thomas, of pragmaticprogrammer.com, for massively improving the original sample script and reviewing the draft manuscript. Kent Dahl and Sean Chittenden for reviewing the draft manuscript. Last, but not least, Magnus Lie Hetland, Python guru, for invaluable assistance.

Resources

Thomas Østerlie is a consultant with Norwegian-based consulting company ConsultIT A/S, where he works with server-side systems development for UNIX platforms and with computer security. He has been an avid Linux user since 1995, after having been forced to install Windows 95 on his office computer. He can be reached at thomas.osterlie@consultit.no.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Browser Comparison

Ralph Krause

Issue #95, March 2002

A lot of web browsers are available for Linux, and quite a few are pre-1.0 versions. Here's what they can do—and what they can't do.

Since the open-source Mozilla Project began a few years ago, the promise of a lightweight, reliable, standards-compliant browser for Linux has loomed on the horizon. This article looks at seven Linux web browsers currently under development and documents how well they performed several browsing tasks.

All of the browsers worked well and were stable, but there were some disappointing test results, such as the inability to print a range of pages. As all of these browsers are still under development, there is hope that these types of problems will be fixed soon.

The browser versions covered in this article are: Beonex-Communicator 0.7-dev-2, BrowseX-1.5.0, Galeon-1.0.1, Konqueror-2.2.1, Mozilla-0.9.6, Opera-5.0-static and SkipStone-0.7.7. SkipStone and Konqueror were compiled from source, the others were installed as RPM packages. Due to the rapid development of these browsers and the production schedule inherent in a monthly publication, these versions probably will not be the latest by the time you read this.

The system running the browsers was a Red Hat 6.2 installation on a Pentium 133MHz computer with 80MB of RAM. GNOME 1.4 and KDE 2.2.1 also were installed, along with the CUPS-1.1.5 printing system.

The Tests

The tests that the browsers were put through were meant to determine how well they performed tasks such as browsing, downloading files and printing. The tests and results are summarized in Table 1, and the actual tests are explained below.

Table 1. Browser Test Results

Table Key and Notes

- Web Banking: this test determined if a browser could log in to my bank's web banking system and view account data.
- PayPal: each browser had to sign in to PayPal (www.paypal.com), view account balances and transfer money from the PayPal account.
- Encryption: the SSL Check page at Fortify.net (www.fortify.net/sslcheck.html) was used to determine the strength of each browser's encryption.
- My eBay: passing this test involved signing in to eBay (www.ebay.com) using the Sign In option and viewing multiple pages within My eBay without having to sign in again.
- Create an eBay Auction: each browser had to successfully create a new eBay auction that consisted of the following steps: clicking on the Sell button, selecting a category, allowing the sell-item form to be filled out and processing the form when the Continue button was clicked. Early Mozilla-based browsers failed this test because of the way they handled JavaScript errors (Mozilla bug 91018), but this appears to have been fixed.
- iPrint: the iPrint site (www.iprint.com) allows the creation of business stationery with a web browser. To successfully pass this test, the browser had to be able to select business checks and edit their layout.
- Printing: printing capabilities of the browsers were tested by seeing if the browsers could print a range of pages, print in first-page-first and first-page-last order, print in color and grayscale, print in portrait and landscape orientations and print a page to disk.
- Save Site to Disk: this test simply involved saving web pages to disk and then being able to view the files.
- Downloading: the downloading capabilities of each browser were tested by clicking on download links and by logging in to FTP sites. The ability to specify external downloading applications was also noted.
- Usability Features: this category notes some features that increase the ease of use of the browser. The following values were possible: disable animations (it is possible to disable GIF animations), drag-and-drop (URLs can be dragged to the browser or to other applications from the browser), ID (the browser's user-agent string can be changed), mouse wheel (the browser responded to the mouse scroll wheel), one-click clear location (the browser provided a method to clear the location text box with one click of the mouse) and zoom (the browser had the ability to increase and decrease the size of the text on the displayed page).

- Mail: this test indicates how each browser handled mailto: links. It also indicates which browsers offer the ability to launch user-defined mail programs.
- Java: this test determined if the browser could run Java applets, such as the demo programs from the Sun web site. The Java2 package used was downloaded from Netscape's site.
- Plugins: to determine the ability of the browser to recognize Netscape plugins, specifically the Macromedia Flash plugin, and successfully display a site that required it was the purpose of this test.
- Transparent PNG: to pass this test the browser had to properly display a web page containing a PNG picture with a transparent background that was created with The GIMP. The browser that failed the test showed the picture with a black background.

Browser Details

The Beonex Communicator is virtually identical to Mozilla in both appearance and functionality, but it boasts security and searching improvements. Its default Search tab, located in the sidebar, provides more default search engines than Mozilla's, and it allows searches to be run through several search engines at the same time, displaying the results from all the engines. Additional search engines can be added to the browser by going to Mozilla's Sherlock page at sherlock.mozdev.org.

Beonex-Communicator Screenshot

According to Beonex's Ben Bucksch, Beonex is targeting their efforts toward making their browser appealing to users. The browser is easy to install, and software such as Java can be installed from the Beonex web site with a couple of clicks. They also have changed the default settings, things like forced session

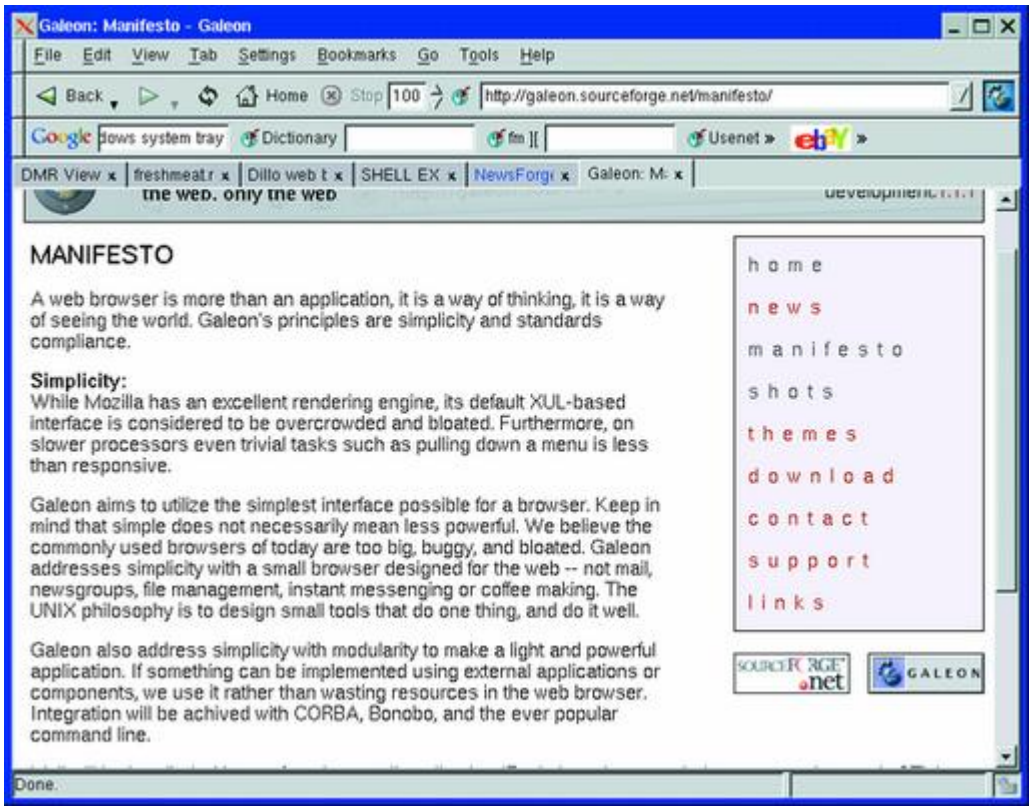
cookies and HTTP referrers, to more secure values. The company also tries to keep users informed of browser exploits and problems via their web site.

BrowseX-1.5.0 is one of the most unique browsers tested; it is written in a combination of Tcl and C, making it resource-light. It contains its own mail and talk/chat clients, supports HTML 3.2 and handles graphics, secure web sites and JavaScript. Another unique feature of this browser is that it implements the TML extension, which provides an embedded web-scripting interface to Tcl, Perl and Python. BrowseX also can import existing Netscape bookmarks by selecting Import Netscape Bookmarks from the File®Util menu.

BrowseX Screenshot

The browser did have some problems with printing and with eBay. Printing to the printer produced nothing, but printing to disk worked. The browser displayed extra characters when rendering eBay pages and had trouble selecting a category when creating an auction. Drop-down lists in BrowseX don't scroll; instead they have a More selection at the bottom of the list that displays another panel containing more list items, much like Netscape. It was impossible to navigate extremely long category lists in this way.

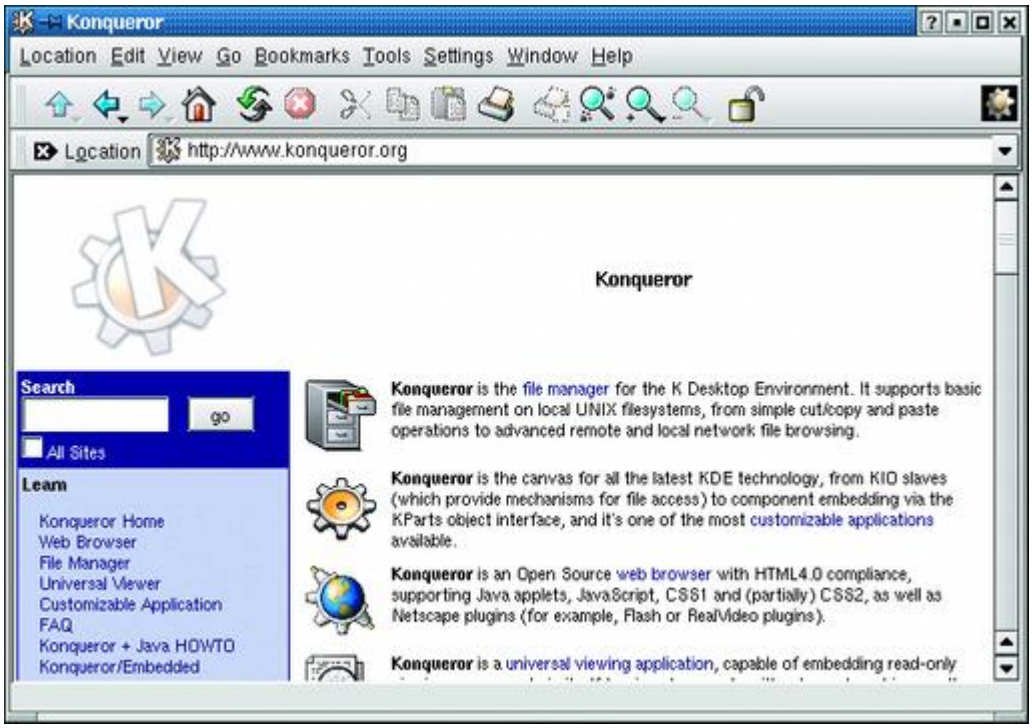
Galeon's motto is “The web, only the web”, and Galeon is meant to be only a web browser, not an all-in-one web tool. It is based on Mozilla code and requires Mozilla to be installed before it can be used. This was one of the first Mozilla-based browsers, and its maturity shows in the features provided. One such feature is the Smart Bookmarks, which have their own toolbar and allow the user to do things such as search Freshmeat II, Google and Google's news archive. Another nice feature is the Settings menu on the top menubar that allows easy access to such settings as proxies, animation control and JavaScript.



Galeon Screenshot

Galeon has a small annoyance when you are using its default download handler. If the program is set to allow the user to pick the download destination, the download filename is cleared when a directory is picked. The filename has to be typed back in after navigating to the desired directory before the file can be saved.

Konqueror is KDE's filesystem/web browser application and is not based upon Mozilla. The browser has unique features, including the ability to split the window into panes, with each pane displaying different web sites or even a web site and the contents of a local directory. Konqueror allows Java, JavaScript and browser identification to be controlled on a site-by-site basis. Finally, a toolbar button (a black button with a white X) is provided to conveniently clear the location text box.

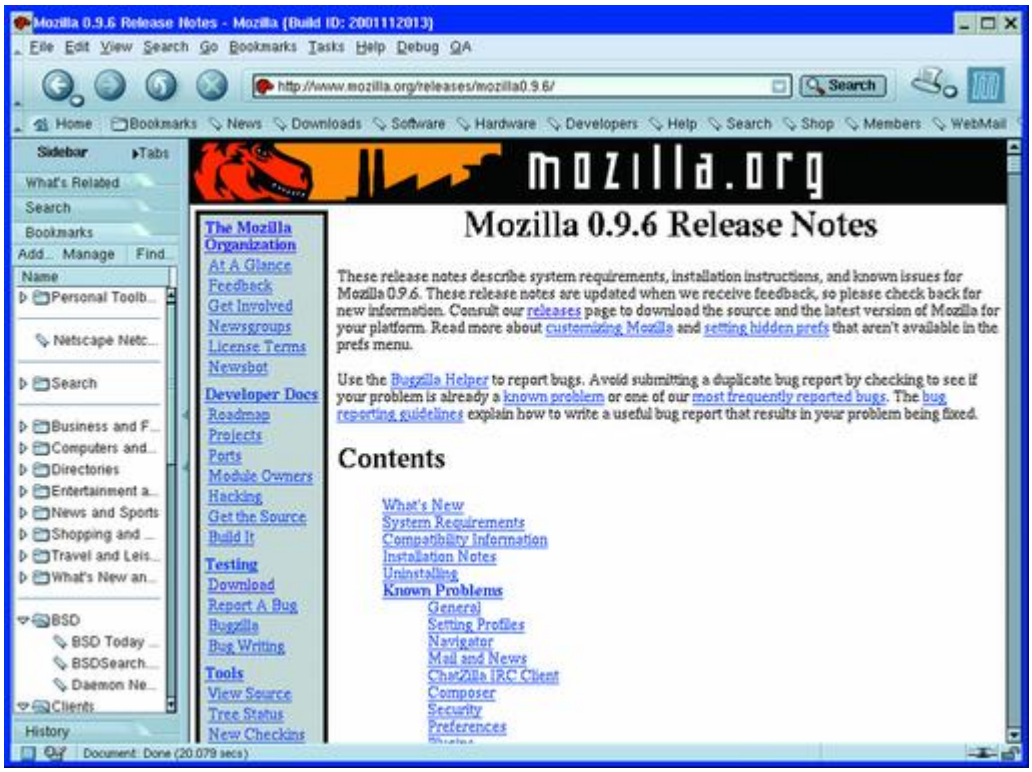


Konqueror Screenshot

Konqueror had a couple of problems with eBay. First, clicking on the My eBay link caused Konqueror to say it was downloading a .DLL file, and it then displayed HTML source instead of the correct page. When creating an auction, clicking on the Books category always brought up the Antiques category. Clicking on other categories worked correctly.

There is a Konqueror + Java HOWTO on www.konqueror.org that provides information for using Java in Konqueror. Also, when compiling KDE from source, use **config shared** when building OpenSSL so that the shared libraries needed by Konqueror are created.

Mozilla is the browser that was open sourced by Netscape back in 1998. It does not have lightweight as a design goal, and the browser's sluggishness can be quite noticeable. According to its web site, work is being done to speed up the code, and each new version does seem faster than the previous one. It does support some notable features, such as a sidebar that contains bookmarks and search results and the ability to switch themes. It was also one of the few browsers that could successfully print a range of pages, but it failed at printing the last page first.



Mozilla Screenshot

Version 0.9.6 introduces a print preview function that contains some oddities. First, the print preview is displayed in the main browser window, and there doesn't seem to be any way to switch back to the page display short of reloading the page. Also, the print preview shows headers, like the page title and URL, which do not appear when the pages are printed. Mozilla also has added the ability to show multiple web pages in one window, with tabs instead of a new window for each site.

If Mozilla is installed from RPMs, the Personal Security Manager (psm) package needs to be installed so that Mozilla and Mozilla-based browsers (e.g., Galeon and SkipStone) can handle encrypted web pages.

Long available on Windows, Opera bills itself as the fastest browser available. The program starts quickly and is very responsive, but it doesn't seem to render pages significantly faster than the other browsers. Opera for Linux is built using the Qt toolkit, and both statically and dynamically linked versions of the browser may be downloaded from the web site.

Opera Screenshot

Opera shows maturity in its interface and capabilities, but there are some things to watch out for. One large bug that cropped up was it occasionally wouldn't erase the old page and display a new web page, even though it insisted that it had finished loading it. This happened frequently with Freshmeat (freshmeat.com), and the only fix was to load the requested page again. There were also problems with printing; printing in landscape orientation produced no output, and the browser prints in color even if grayscale is selected. Opera 5.0 doesn't support Java or plugins, but the Opera 6 Technology Preview notes indicate that these items are available in version 6.

Also built on Mozilla, SkipStone is younger than Galeon and sports a sparser, though functional, interface. This browser still has some quirks, such as there is no way to browse local directories using the File®Open menu and there are no location histories for the Forward and Back arrows.

SkipStone Screenshot

While other Mozilla-based browsers were able to create an auction on eBay, SkipStone crashed as soon as the Continue button was pressed.

Conclusion

Quite a few browsers are available for Linux, and the competition seems to bring out useful features in all of them. For the most part, all of these browsers are good enough to be used for day-to-day browsing, but your choices may be limited depending upon the sites you visit. Several of the programs examined are pre-1.0 versions, so there is hope that any quirks and bugs will be worked out as development continues.

Resources

Ralph Krause lives in Michigan's lower peninsula and works as a writer, web designer and programmer. He has been using Linux for over three years and can be reached at rkrause@netperson.net

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Configuring pppd in Linux, Part II

Tony Mobily

Issue #95, March 2002

After showing you how to configure your modem in Part I, Tony moves on to show you how to connect to the Internet.

In Part I of this article (see the February 2002 issue of *LJ*), I explained how to configure the modem. At the end of the article you ended up with a symbolic link called /dev/modem that pointed to the right device file for your modem. You were sure that everything worked fine, as you connected to your internet service provider (even though you didn't establish a PPP connection). In this article I go further, explaining how to connect to the Internet. You are advised to read the previous article first. If you can't, just make sure that your modem is configured correctly, and that you do have a symbolic link called /dev/modem that points to the right device file in /dev.

In order to make any use of this article you should have all the login information about your provider, including the phone number you should dial to connect, your login and password, and a valid DNS server address (this is optional, as it can be assigned automatically by your provider).

This article assumes that your provider accepts PAP authentication. PAP is a way of sending your login and password information to the provider through the PPP protocol; it saves the users from the more complicated (and often manual) login procedures required by some internet service providers in the past. The vast majority of ISPs today will require you to use PAP. This article also assumes that you have a standard modem and not a Winmodem. Configuring a Winmodem is possible but can be tedious and is outside the scope of this article.

Establishing the Connection: the Basic Tools

First of all, you must be logged in as root to configure your internet connection. A connection to the Internet via the modem is established using PPP (point-to-

point protocol), which is used to encapsulate common TCP/IP packets so that they can be sent through a serial line (in fact, TCP/IP packets are meant to be transmitted over a network medium and wouldn't fit, as they are on a serial connection without being serialized first).

This article assumes that you have the following software installed: the kernel module that handles the PPP connection (every distribution I know of comes with a kernel that includes the PPP module, so you probably don't need to worry about it); the program pppd, which initializes the kernel modules after establishing the connection; the program chat, which is responsible for establishing the connection; and the program minicom, a very simple terminal program that lets you talk to the modem. To check if you have all of these programs, you can use the **which** command. This command tells you if the programs are available in one of the directories listed in the $PATH environment variable:

```
which pppd
/usr/bin/pppd
which chat
/usr/bin/chat
which minicom
/usr/bin/minicom
```

If you don't have some of these programs, you should grab the right package and install it. You also might want to disable the call waiting for your telephone line. If it is active, an incoming call could cause the line to drop while you are connected.

Overview of a Connection

Once we configure the modem, how do we connect to the Internet? We have to use (and configure) a program called pppd (point-to-point protocol dæmon). The following is what happens when you run pppd (assuming that chat is used as the dialer program and that pppd is correctly configured).

The dæmon starts. It sets the serial port parameters (speed, etc.). Then it runs an external program (chat) to establish the connection, which sends the connection command to the modem (ATDT followed by the provider's number). Then it waits for the string CONNECT from the serial port. At that point the connection has been established, and it's as if there were a serial cable running from your computer to the provider's computer. Once chat has finished its job, the program pppd takes over again. If the connection could not be established, pppd will exit and return an error. Otherwise, it will talk to the PPP dæmon on the other side of the line (the PPP handshake that you saw earlier as a bunch of indecipherable symbols) and will be assigned an IP address. A login and a password normally are required to complete this stage successfully (login information is sent during the PPP handshake). The program pppd makes sure

that a kernel network interface is created and that the network traffic is directed to it.

A Few Words about Logging

The two programs that you need, pppd and chat, are not interactive. They are run and then send any messages to the system log dæmon, syslogd. The syslogd will then write the received messages on the hard drive. There are several classes of messages, and the different classes usually are stored in separate files. The exact place they are stored depends on your syslogd configuration.

Now, you should configure syslogd so that you are 100% sure that the debugging information from the dæmons pppd and chat actually are stored on disk—and that you know where they are. The configuration file for syslogd is /etc/syslog.conf. All you have to do is enter one extra line to it. To do that, just type the command:

```
vi /etc/syslog.conf
```

Of course, you may use any editor you like (vi, Emacs, joe, pico, etc.). Now, insert the following line:

```
daemon.debug;*.info           /var/log/ppp_article
```

Remember that there should be a tab between info and /var/log/ppp_article.

Now, you have to make sure that the dæmon syslogd knows about the change in its configuration file. To achieve this, run the command:

```
killall -HUP syslogd
```

The file /var/log/ppp_article should have been created and should contain one line that tells you that syslogd has been restarted. To check that this is true, you can type the following command:

```
cat /var/log/ppp_article
Aug  4 19:28:46 merc_linux syslogd 1.3-3: restart.
```

Instead of the cat command, which just reads a file, you can use the command tail with the option -f. This will keep on reading a file and will print on the screen any new information added to it. This means that as soon as syslogd writes anything on the file ppp_article, tail will show it on the screen:

```
tail -f /var/log/ppp_article
Aug  4 19:28:46 merc_linux syslogd 1.3-3: restart.
```

From now on, any logging information recorded by pppd or chat will appear on the screen automatically. You really should keep this console open always, and check for messages whenever you need to.

Understanding chat

As you probably saw in Part I of this article, in order to establish the serial connection, you have to send the string ATDT12345678 (with your provider's phone number, of course) to your modem and wait for the string CONNECT to come back from the modem itself (that would happen once the connection has established). Some messages other than CONNECT might be returned: BUSY, NO CARRIER, NO ANSWER, etc. In the previous article, you tried this practically, using minicom.

Even though you could do all of this by hand using minicom, you might want to use a program that does it all for you. The program should be able to talk to the modem, sending information and expecting a particular string as a response. Of course, such a program does exist, and it's called chat. For example, try to run the following command:

```
chat ABORT "BUSY" "OK" "TRY" "THIS"
"TESTING" "COMMAND"
```

Be careful, because from now on the keyboard will be locked and you won't be able to quit the program, not even by pressing Ctrl-C. Type **ok**. The word TRY will pop out. Now type **this**; the word TESTING will appear on the screen. Finally, type **command**; the program chat will exit successfully. Try to run the command again: type **ok**, and again you will see the string TRY come out. At this point, type **busy**: the program will exit immediately. As you can guess, the chat program is designed to wait for a string and print something as a response. The first two words, ABORT BUSY, are special and instruct chat to exit if the word BUSY is received at any point during its execution. If something goes wrong, you can run the same chat command adding the switch -v:

```
chat -v ABORT "BUSY" "OK" "TRY" "THIS"
"TESTING" "COMMAND"
```

The -v option stands for verbose, meaning that chat will tell you exactly what is going on, what it is expecting and so on. Of course, all the debug information will be recorded in /var/log/ppp_article if you followed the instructions I gave you earlier about syslogd. Let's analyze a different chat command:

```
chat ABORT "BUSY" "" "AT" "OK"
"ATDT93355100" "CONNECT"
```

As you can probably guess, you will have to behave like a modem in order to get chat to exit successfully. It will send you an AT string, and you have to type **ok**. Then, it will send you the string ATDT93355100 and wait until you type

connect. Then, it will exit. This probably sounds familiar to most readers; this is exactly what you need to connect to your ISP, if you could get chat to talk to the modem and not the keyboard. The command I use for my provider is:

```
chat ABORT BUSY ABORT "NO CARRIER"
TIMEOUT 120 "" AT OK ATDT94310999 CONNECT
```

It's very simplistic, and as a matter of fact, it could be done a lot better. But in my case, it does the job and I am perfectly happy with it. You should have a look at the man page for chat (just type **man chat**) and look at the options it offers; later, you might want to change your connection script so that it uses all of the fancy options offered by chat. The next step is to write a shell script that encapsulates the chat command you wrote. The file will be placed in /etc/ppp and will be called chat-connect. To create it, just type the command:

```
vi /etc/ppp/chat-connect
```

(of course, you can use any editor you like if you don't like vi). The script should look like this:

```
#!/bin/sh
chat ABORT BUSY ABORT "NO CARRIER" TIMEOUT 120 ""
AT OK ATDT94310999 CONNECT
```

You should substitute 94310999 with your ISP's dial-up number. Now, save and exit the editor. You need to make the script executable, with the chmod command:

```
chmod +x /etc/ppp/chat-connect
```

See if the script works by running

```
/etc/ppp/chat-connect
```

If it works, you moved one step toward your working internet connection. Effectively, you are very close to the goal. All you have to do is run pppd with the right parameters.

Understanding pppd

At this point you can start to work on the actual pppd configuration. The files involved are /etc/ppp/options, /etc/ppp/chap-secrets, /etc/ppp/pap-secrets and /etc/ppp/peers.

The options file is used to give pppd a list of default options. For now you should make sure that the options file, stored in /etc/ppp, is totally empty; just edit it with your favorite editor and delete everything in it. If you don't feel comfortable deleting the content, you can comment all the lines out putting a # symbol in front of each line. It is important to have the options file empty to make sure you have a fresh start. The first thing now is to test if the chat script we wrote works in a real situation. In order to do that, you can run pppd with the minimum number of parameters:

```
pppd /dev/modem 38400 modem lock connect
/etc/ppp/chat-connect
```

The parameters can be given to pppd in any order. The /dev/modem option represents the serial port that the modem is connected to (as you know, it is a symbolic link that points to the real ttyS device). The parameter modem instructs the pppd dæmon that it will be dealing with a modem connection, and not a straight serial cable between you and your provider. The word lock tells pppd to lock the modem while using it (if you don't know what that means, don't panic; basically it's a way of guaranteeing that no other program will be accessing the modem while your connection is up). The last option, connect, comes with the parameter /etc/ppp/chat-connect and tells pppd what program it should run to dial the number and connect to the internet service provider; in your case, it's the chat script you wrote in the previous section of the article.

If nothing seems to work, you should add the option -v to the chat script, try again and look at the logs—at this point, it's normally quite easy to fix problems. If everything goes well, you should be able to see your modem connecting and hear it going through the usual whistling noises. Now you should be able to connect to the Internet with only one extra step. Edit the file /etc/pap-secrets and add your password to that file, adding a line that looks like this:

```
your_username_here  *     your_password_here
```

Remember that there should be a tab between each word. Now you are ready for the big test, an actual connection. Try the following command:

```
pppd /dev/modem 38400 modem lock
connect /etc/ppp/chat-connect
user your_username_here defaultroute
```

The only extra parameters are user (followed by your user name as it comes in /etc/ppp/pap-secrets) and the option defaultroute. This last option makes sure that your connection will be used by default by the packets that are supposed to reach the Internet. With this option, pppd will set up the correct routing table once the connection is established. You should see, in the log, a message like this:

```
Aug  4 16:12:23 merc_linux pppd[4430]:
local  IP address 94.232.195.174
Aug  4 16:12:23 merc_linux pppd[4430]:
remote IP address 194.232.195.4
```

If it didn't happen, you might have to run pppd with the debug option and read the log file (that is /var/log/ppp_article) to see what happened:

```
pppd /dev/modem 38400 modem lock
connect /etc/ppp/chat-connect
user your_username_here defaultroute debug
```

If everything worked, congratulations; you are now connected to the Internet. Remember that when you want to disconnect, you simply can type:

```
killall pppd
```

Testing the Connection

The next step is to test whether the connection actually works. The best way to see if the link is up is to run ifconfig (see Listing 1). This command shows you the active kernel network interfaces. In my case, I have lo, the standard loopback interface I will use if I want to connect to myself, and ppp0, which is the modem PPP interface.

Listing 1. The Result of My ifconfig Command

To see if you actually are routing to the Internet, you can run the traceroute command, followed by any IP address. For now you should use the -n option in order to disable the DNS name resolution (that hasn't been configured yet). For example:

```
traceroute -n 198.182.196.56
traceroute to 198.182.196.56 (198.182.196.56),
30 hops max, 38 byte packets
 1  194.232.195.4 (194.232.195.4) 181.518 ms
    139.473 ms  149.822 ms
 2  194.232.195.1 (194.232.195.1) 129.540 ms
    139.739 ms  139.821 ms
```

...

```
19  207.245.34.122 (207.245.34.122)  479.696 ms
    479.653 ms *
20  198.182.196.56 (198.182.196.56) 489.711 ms
    479.644 ms  479.874 ms
```

The IP 198.182.196.56 is the server for www.linux.org. The program traceroute will tell you about the path followed by the packets you send to the Internet. Now, you should make sure that you tell your system the IP of your DNS, through the file /etc/resolv.conf. My resolv.conf file looks like this:

```
nameserver 203.14.168.3
nameserver 202.0.185.226
```

Some ISPs don't provide a DNS server address, as your computer is given one once the PPP handshake is completed. If that is the case, you simply can disconnect and reconnect using the usepeerdns option when you run pppd:

```
pppd /dev/modem 38400 modem lock
connect /etc/ppp/chat-connect
user your_username_here defaultroute usepeerdns
```

Now, you can try to see if your DNS is working, using, for example, the Telnet program. The Telnet program is only an excuse to see if the system was able to translate the name www.linux.org into an IP address.

```
telnet www.linux.org 80
Trying 198.182.196.56...
Connected to www.linux.org.
Escape character is '^]'.
```

It worked! Now, you can start your browser (Netscape, Mozilla, Opera, Galeon, Lynx, etc.) and browse the Net as you like.

A Bit of Housekeeping

By now, everything should work well; the internet connection is up, and you can connect to the Internet whenever you want. There is, of course, room for improvement. The first thing to do would be to increase the speed of the serial port and see if everything still works. To do that, just substitute 38400 with 115200 in the pppd command line.

Also, after a couple of weeks you probably will start noticing that there is a high number of parameters that have to be typed for the command pppd. In fact, every time you want to connect you have to type:

```
pppd /dev/modem 115200 modem lock
connect /etc/ppp/chat-connect
user your_username_here defaultroute
```

The good news is that you can, of course, put all those parameters in a configuration file, /etc/ppp/options. So, in your case, the options file would look like this:

```
/dev/modem
115200
modem
lock
connect /etc/ppp/chat-connect
user
defaultroute
```

In this file the order of the parameters really doesn't matter. From this point on, you will be able to connect to the Internet simply by typing the command **pppd**. What happens if you have several providers you might want to call? In this case, you can create several options files and then place them in /etc/ppp/peers. The output below shows what my peers directory looks like:

```
ls -l /etc/ppp/peers
total 4
-rw-r--r--  1 root   root    197 Aug  4 15:41 main_net
-rw-r--r--  1 root   root    189 Mar 11  2000 primus
```

My file /etc/ppp/options is empty; when I run pppd, I always run:

```
pppd call main_net
```

This way, the file /etc/ppp/peers/main_net will be used as well as my /etc/ppp/options file (which happens to be empty). If my main provider (Main Net) is

down for some reason, I still can use some of my time-limited account with Primus.

Now, the best thing you can do is to read the man page for pppd (just type **man pppd**) and see if any of the esoteric options can somehow improve your connection. In Listing 2 you will find a very rich options file written by my friend and Linux guru Pancrazio De Mauro. Can you do better than that?

Listing 2. Pancrazio's Options File

Conclusion

This process certainly can look quite scary; the amount of knowledge you must have to connect to the Internet using Linux seems ludicrous, especially if you compare it to the simplicity of the Windows Remote access interface; the comparison makes you wonder whether it was worthwhile doing everything by hand.

In my opinion, there are two main advantages in configuring everything by hand. The first one is that you can (and should) go through the many options of pppd to optimize your connection. The second is that from now on when you use a graphical interface to configure your internet access, you know exactly what is going on, and you can fix problems if the automatic process doesn't seem to work properly.

Before I finish, I would like to point out that there is a command-line program (no GUI) that automatically does everything I have explained in this article (find the modem, connect to the provider with the right parameters, etc.). The program is called wvdial (www.worldvisions.ca/wvdial/index.html). When I discovered it a few years ago, I found it rather amazing. I would suggest it to impatient people who want to connect to the Internet quickly without going through the hassle of knowing everything about pppd, chat, etc.

Resources

Tony Mobily (merc@mobily.com) is the technical editor of Login, an Italian computer magazine. He is an LCI (Linux Certification Instructor, www.linuxcertification.com) and knows how to use English, Italian, C, Perl and a

few other languages. He works as a trainer and system administrator and is training as a dancer.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Inside the Linux Packet Filter, Part II

Gianluca Insolvibile

Issue #95, March 2002

Gianluca concludes the packet's journey through the kernel, picking up with TCP processing.

In the last issue we started following a packet's journey from the wire up to the higher levels of network stack processing. We left the packet at the end of layer 3 processing, where IP has completely finished its work and is going to pass the packet to either TCP or UDP. In this article, we complete our analysis by considering layer 4, the PF_PACKET protocol implementation and the socket filter hooks.

Apart from the case of IGMP and ICMP processing, which is dealt with in the kernel, the packet's journey toward the application proceeds by passing through either tcp_v4_rcv() or udp_rcv(). TCP processing is a bit intricate, since this protocol's FSM (finite state machine) has to deal with a lot of intermediate states (just think of the various states a TCP socket can assume: listening, established, closed, waiting and so on). To simplify our description, we can reduce it to the following steps:

- Inside tcp_v4_rcv() (net/ipv4/tcp_ipv4.c), perform TCP header integrity checks.
- Look for a socket willing to receive this packet (using __tcp_v4_lookup()).
- If it is not found, take appropriate actions (among which, cause IP to generate an ICMP error).
- Otherwise call tcp_v4_do_rcv(), passing to it both the packet (an sk_buff) and the socket (a sock structure).
- Inside this latter function, perform different receive actions based on the socket's current state.

The most interesting aspect in TCP processing from the LSF point of view, comes in the last function we mentioned; at its very beginning, tcp_v4_do_rcv()

calls sk_filter(), which as we will see, performs all the filter-related magic. How does the kernel know that it should invoke the filter for packets received on a given socket? This piece of information is stored inside the sock structure associated with the socket. If a filter has been attached to it with a setsockopt() system call, the appropriate field inside the structure is not NULL, and the TCP receive function knows that it has to call sk_filter().

For those of you who are fluent with sockets programming and recall that listening TCP sockets are forked upon reception of a connect message, it must be said that the filter is first attached to the listening socket. Then, when a connection is set up and a new socket is returned to the user, the filter is copied into the new socket. Have a look at tcp_create_openreq_child() in net/ipv4/tcp_minisocks.c for details.

Back to packet reception. After the filter has been run, the fate of the packet depends on the filter outcome; if the packet matches the filter rules, processing proceeds as usual. Otherwise, the packet is discarded. Furthermore, the filter may specify a maximum packet length that will be kept for further processing (the exceeding part is cut via skb_trim()).

The packet's trip proceeds on different paths depending on the socket's current state; if the connection is already established, the packet will be passed to the tcp_rcv_established() function. This one has the important task of dealing with the complex TCP acknowledgment mechanisms and header processing, which of course are not very relevant here. The only interesting line is the call to the data_ready() function belonging to the current sock (sk), commonly pointing to sock_def_readable(), which awakens the receiving process (the one that was receiving on the socket) with wake_up_interruptible().

Luckily, UDP processing is much simpler; udp_rcv() just performs some integrity checks before selecting the receiving sock (udp_v4_lookup()) and calling udp_queue_rcv_skb(). If no sock is found, the packet is discarded.

The latter function calls sock_queue_rcv_skb() (in sock.h), which queues the UDP packet on the socket's receive buffer. If no more space is left on the buffer, the packet is discarded. Filtering also is performed by this function, which calls sk_filter() just like TCP did. Finally, data_ready() is called, and UDP packet reception is completed.

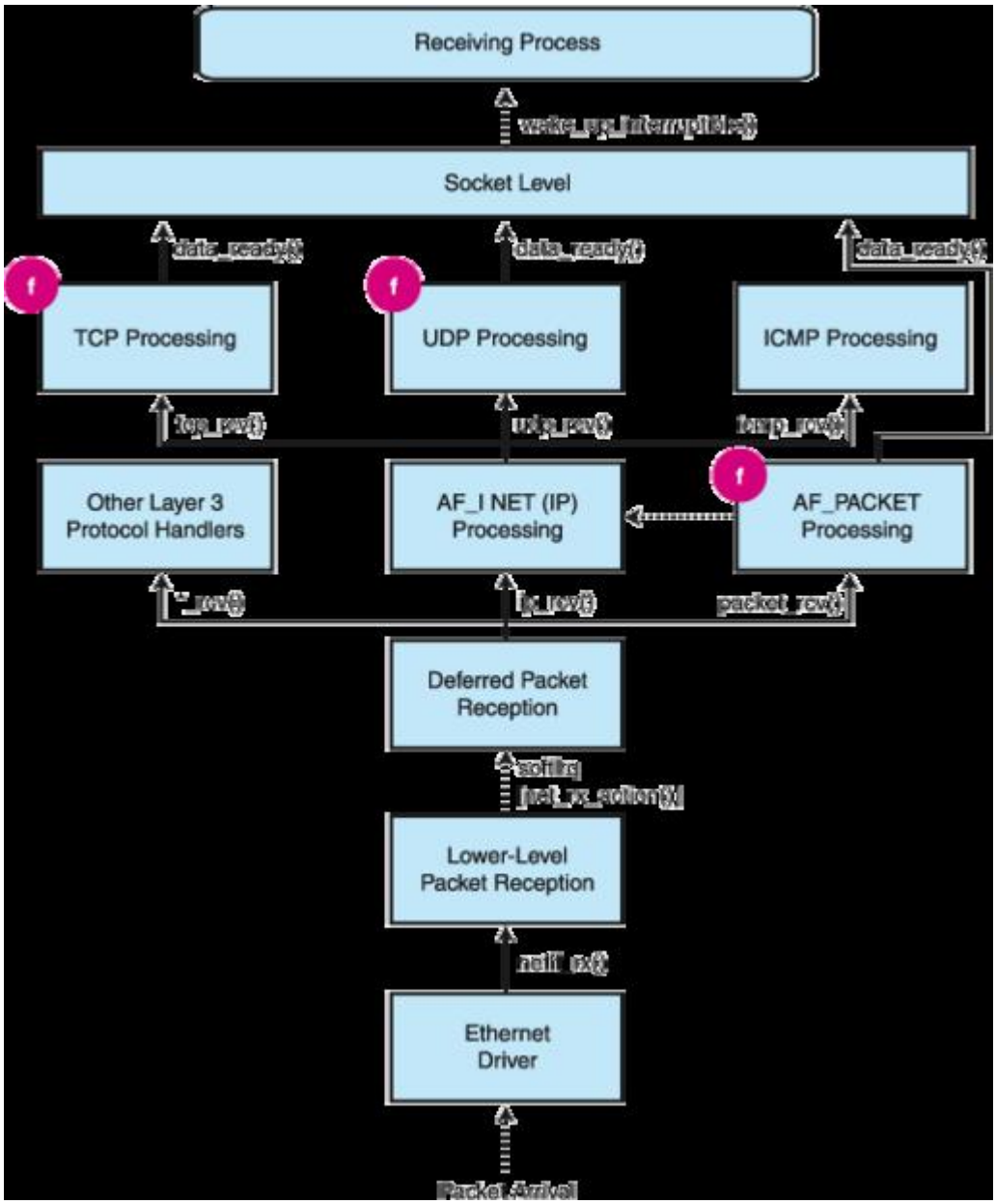


Figure 1. The Whole Reception Path, with Functional Blocks and Relevant Function Calls

What Happens to PF_PACKET Packets?

The PF_PACKET family deserves a special handling. Packets must be sent directly to the application's socket without being processed by the network stack. Given the packet processing mechanisms we have outlined in the previous sections, this is actually not a difficult task.

When a PF_PACKET socket is created (see packet_create() in net/packet/af_packet.c), a new protocol block is added to the list used by the NET_RX softirq. The packet type related to this protocol family is put either in the generic list (ptype_all) or in the protocol-specific one (ptype_base) and has packet_rcv() as a receive function. For reasons that will be clear in a while, the newly created sock's address is written inside the packet type data structure. This address would not logically belong to this part of the kernel, since usually

the socket information is dealt with by layer 4 code. Hence, in this case, it is written as private opaque information in the data field of the protocol block being registered—a field reserved inside the structure for protocol-specific mechanisms.

From that moment on, each packet entering the machine and going through the usual receive procedure will be intercepted during net_rx_action() execution and passed to the PF_PACKET receive function.

The first thing this function does is to try to restore the link layer header, if the socket type is SOCK_RAW (recall from my article, "The Linux Socket Filter: Sniffing Bytes over the Network", *LJ*, June 2001, that SOCK_DGRAM sockets will not see the link layer header). This header has been removed either by the network card (or any other generic link layer device that received the packet) or by its driver (interrupt handler). When dealing with Ethernet cards, the latter is almost always the case. Restoring the link layer header is not possible if removal has taken place at the hardware level, since that information never gets to the system main memory and is invisible outside the network device. The computational cost of header restoration is quite low, thanks to the smart handling of skbuffs inside the kernel.

The following step is to check whether a filter has been attached to the receiving socket. This part is a bit tricky because filter information is stored inside the sock structure, which is not known yet since we are at the bottom of the protocol stack. But for PF_PACKET sockets, which must be able to skip the protocol stack, the receiving sock structure address has to be known *a priori*. This explains why, during the socket creation phase, the address of the sock structure had been written opaquely into the protocol block's private data field; this provides a relatively clean way to retrieve that information during packet reception.

With the sock structure in hand, the kernel is able to determine whether a filter is present and to invoke it (via the sk_run_filter() call). As usual, the filter will decide whether the fate of the packet is to be discarded (kfree_skb()), be trimmed to a given length (pskb_trim()) or be accepted as it is.

If the packet is not discarded, the next step consists in cloning the sk_buff containing the packet. This operation is necessary because one copy will be consumed by the PF_PACKET protocol, and the other must be made available for possible legitimate protocols that will be executed later. For example, imagine running a program that opens a PF_PACKET socket on a machine that is browsing the Web at the same time. For each packet belonging to the web connection, the net_rx_action() function would call the PF_PACKET processing routines before calling the normal IP ones. In this case, two copies of the packet

would be needed: one for the legitimate receiving socket, which would go to the web browser, and the other for the PF_PACKET, which would go to the sniffer. Note that the packet is duplicated only after being processed by the filter. This way, only packets that actually match the filter rules engage the CPU. Also note that packet filtering performed at application level (i.e., using a PF_PACKET with no socket filter) would require cloning of all the packets received by the kernel, resulting in poor performance. Luckily, packet cloning simply involves copying the fields of the sk_buff, but not the packet data itself (which is referenced by the same pointer in the clone and in the original sk_buff).

The PF_PACKET receive function finally completes its job by invoking the data_ready() function on the receiving socket, just like the TCP and UDP processing functions did. At this point the application sleeping on a recv() or recvfrom() system call is awakened and packet reception is complete.

Sleeping Processes

As a side note, you may be wondering, how does a user process come to sleep on a given socket when it invokes a recv(), recvfrom() or recvmesg() system call? The mechanism is actually pretty easy: all the recv functions are implemented inside the kernel by calling, more or less directly, sock_recvmsg() (in net/socket.c). This function, in turn, calls the recvmsg() function that is registered inside the protocol-specific operations within the sock structure. For example, this function is packet_recvmsg() in the case of PF_PACKET protocol.

The protocol-specific recvmsg function, among other things, sooner or later will call skb_recv_datagram(), which is a generic function handling datagram reception for all the protocols. The latter function obtains process blocking by calling wait_for_packet() (in net/core/datagram.c), which sets process status to TASK_INTERRUPTIBLE (i.e., sleeping task) and queues it into the socket's sleep queue. The process rests there until a call to wake_up_interruptible() is triggered by the arrival of a new packet, as we saw in the previous paragraphs.

What about the Filter Itself?

The main filter implementation resides in core/filter.c, whereas the SO_ATTACH/DETACH_FILTER ioctls are dealt with in net/core/sock.c. The filter initially is attached to a socket via the sk_attach_filter() function, that copies it from user space to kernel space and runs an integrity check on it (sk_chk_filter()). The check is aimed at ensuring that no incongruent code is executed by the filter interpreter. Finally, the filter base address is copied into the filter field of the sock structure, where it will be used for filter invocation as we saw before.

The packet filter proper is implemented in the sk_run_filter() function, which is given an skb (the current packet) and a filter program. The latter is simply an array of BPF instructions (see Resources) that is a sequence of numeric opcodes and operands. The sk_run_filter() function does nothing more than implement a BPF code interpreter (or a virtual CPU, if you prefer) in a pretty straightforward way; a long switch/case statement discriminates the opcode and takes actions on emulated registers and memory accordingly. The emulated memory space, where the filter code is run, is of course the packet's payload (sk->data). The filter execution flow terminates, leading toward exiting the function, when a BPF RET instruction is encountered.

Note that the sk_run_filter() function is called directly only from PF_PACKET processing routines. Socket-level receive routines (i.e., TCP, UDP and raw IP ones) go through the wrapper function sk_filter() (in sock.h), which in addition to calling sk_run_filter() internally, trims the packet to the length returned by the filter.

Hooks to Packet Filter

Our tour of the kernel packet handling functions is now completed. It is interesting to draw some conclusions regarding the packet filter invocation points. As we have seen, there are three distinct call points inside the kernel where the filter may get invoked: the TCP and UDP (layer 4) receive functions, and the PF_PACKET (layer 2.5) receive function. Raw IP packets are filtered also because they pass through the same path followed by UDP packets (namely, sock_queue_rcv_skb()), which is used for datagram-oriented reception).

It is important to notice that, at each layer, the filter is applied to that layer's payload. That is, as the packet travels upward the filter can see less and less information. For PF_PACKET sockets, the filter is applied to layer 2 information, which includes either the whole link layer data frame for SOCK_RAW sockets or the whole IP packet; for TCP/UDP sockets, the filter is applied to layer 4 information (basically, port numbers and little other useful data). For this reason, layer 4 socket filtering is likely to be useless. Of course, in any case the application level payload (user data) is always available for the filter, even if it is often of little or no use at all.

Listing 2. udpsnd

A bright example of layer 4 uselessness is given in Listing 1 [available at ftp.linuxjournal.com/pub/lj/listings/issue95/5617.tgz and Listing 2, which presents a simple UDP server with an attached socket filter and an associated simple UDP data sender. The filter will accept only packets whose payload starts with “lj” (hex 0x6c6a). To test the program, compile and run Listing 1, called udprcv. Then compile Listing 2 (udpsnd), and launch it like this:

```
./udpsnd 127.0.0.1 "hello world"
```

Nothing will be printed by udprcv. Now, try writing a string starting with “lj”, as in

```
./udpsnd 127.0.0.1 "lj rules"
```

This time the string is printed correctly by udprcv since the packet payload matches the filter.

Another important issue that filter writers should be aware of is that the filter must be written depending on the type of socket (PF_PACKET, raw IP or TCP/UDP) that the filter will be attached to. In fact, filter memory accesses use offsets that are relative to the first byte in the packet payload as seen at a specific level. Filter memory base addresses corresponding to the most common families are reported in Table 1.

Table 1. Filter Memory Base Addresses

Moreover, the method described in the June 2001 article to obtain the filter code (i.e., using **tcpdump -dd**) does not apply anymore if non-PF_PACKET sockets are used, as it produces a filter working only for layer 2 (since it assumes that address 0 is the start of the link layer frame).

Conclusion

Following a packet's journey through the kernel can be an interesting experience. In our trip we encountered typical kernel data structures (such as skbuffs), discovered idiomatic programming techniques (such as the use of structures with function pointers as an efficient alternative to C++ objects) and met some new 2.4 mechanisms (softirqs).

If you are eager to learn more on the subject, arm yourself with kernel sources and a comfortable editor, swallow a good cup of coffee and start peeking here and there. The price is cheap, and fun is guaranteed!

Creation of PF_PACKET Sockets

Resources

Gianluca Insolvibile has been a Linux enthusiast since kernel 0.99pl4. He currently deals with networking and digital video research and development. He can be reached at g.insolvibile@cpr.it.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

Zope Products

Reuven M. Lerner

Issue #95, March 2002

Reuven continues his exploration of Zope, and this month shows how Zope products provide reusable functionality.

Last month we took an initial look at the open-source Zope application server. In particular, we saw how you can use Zope's DTML (dynamic template markup language) tags to create simple dynamic sites, as well as how you can manage a web site using nothing more than a web browser.

But anyone who has worked with DTML knows that it ceases to be wonderful when you want to create something relatively complex. DTML is best when it is used sparingly or when its functionality is obvious; writing pages of DTML that contain a half-dozen nested conditional (<dtml-if>) tags quickly becomes unreadable and difficult to maintain, not to mention very nonmodular.

Another problem is that DTML exists inside of individual documents, rather than in a central location. If we want to reuse functionality in multiple places, then we must copy our DTML methods and documents. This means that when we want to add or change some functionality, we must go through each of the copies and modify them as well.

The solution to this problem is the Zope product. Each Zope product is actually an object class (or a set of classes) that can be instantiated any number of times in our web site.

This month, we look at Zope products, which form the core of Zope's flexibility. After installing and working with some existing products, we will write our own simple product in Python.

What Can a Product Do?

A Zope product is a package of code, graphics and DTML that provides a piece of reusable functionality. For example, if we were interested in creating a simple page that displays the current time, we could create a DTML document:

```
<p>It is now <dtml-var name="ZopeTime" fmt="fCommon">.</p>
```

But what if we want to expand our page, displaying a weather forecast retrieved via HTTP from another server? DTML is not the answer here; even if we could use it to create our custom functionality, the result would be difficult to manage, as well as ugly to write. Because Zope products are written in Python, they can use any Python module they like, displaying their output in HTML or any other compatible format.

Because each product is treated as a single entity, we can install and remove them as a single unit even if it defines and uses a number of classes. However, this doesn't mean that each product stands on its own; on the contrary, it is possible for one product to use functionality provided by another product.

In addition to products and DTML documents, Zope provides two other means for creating dynamic content: Python scripts (implemented by a product, no less) allow us to write and use small Python programs within Zope. We also can create, edit and use new products using a system known as ZClasses. ZClasses allow you to create new products (and their associated classes) using nothing more than your web browser and DTML.

While these four options provide a great deal of flexibility, deciding which one to use can sometimes be difficult for beginning Perl programmers. Beehive's *The Book of Zope*, which I review in this issue of *Linux Journal*, suggests using ZClasses at the beginning of a project, migrating the code to a full-fledged product after everyone has agreed upon a design. The more complex your functionality is, the more likely it is that you will want to use or write a product rather than rely on DTML and Python scripts.

Managing Products

You can run almost every aspect of Zope products via the Zope management screen, which you can reach via the /manage URL of your Zope server. Click on the control panel link in the left-hand frame to bring up the Zope control panel and on the Product Management link in the main frame to bring up the product management screen.

You should see a list of Zope products, along with a button marked “Add product” at the top of the screen. Products that you can modify through the

Web (including ZClasses, which we briefly mentioned above) are identified with an open box, whereas standard Zope products have a closed-box icon. A closed box simply means that you cannot modify the product itself via the Web. However, most products will let you customize them by setting one or more properties via a web-based interface. But the product itself remains unchanged, unless you modify the source code.

Each product is actually a directory under your Zope installation directory in lib/python/Products. The Sessions product is under lib/python/Products/Sessions, while the Transience product is in lib/python/Products/Transience. (I installed Zope under /usr/local/zope/ on my system, so Sessions is actually in /usr/local/zope/lib/python/Products/Sessions/.) A product directory contains Python code, text files and directories, including:

- __init__.py: this is what Zope scans and executes when it loads your module. Among other things, the initialize method in __init__.py invokes context.registerClass, which (as its name implies) tells Zope that your product exists, what text to display in the Add menu on the /manage screen (with the meta_type parameter) and how to create a new instance of your product when the Add button has been pressed (with the constructors parameter).
- README.txt: as its name implies, this is the README file for a particular product. Clicking on a product name from within the control panel will display a README tab, among others. This tab allows you to look at README.txt without having to look at the filesystem. If the product directory contains no file named README.txt, then no README tab will appear at the top of the screen.
- version.txt: this file contains the name and current version number of your product, separated by minus signs (-). Version 1.2.3 of the product Foo thus will have a version.txt with the following contents: Foo-1-2-3. This version information is displayed in the control panel.
- Help files: a product may contain a help directory, which contains the text displayed by Zope when you click on the help link. Help files are often written using structured text, a minimalist formatting system similar in spirit to Perl's POD documentation system. Structured text is easy to write with a simple text editor and equally easy to read with a standard Linux tool like less.

Zope only looks at the current list of products when it starts up. This means that if you install a new product, you will need to restart your Zope server. This is done most easily from within the control panel.

Installing Products

Now that we have seen what a typical product may contain, we will install a product by downloading it from the Zope web site, unpack it within lib/python/Products and restart Zope. If all goes well, our newly installed product should then appear in our control panel screen. Moreover, we will be able to create new instances of this product anywhere we want in our web hierarchy.

For example, let's create a Slashdot clone using the Squishdot product for Zope. Our first task is to retrieve a copy of Squishdot from www.zope.org/Products. Squishdot is listed under the Feedback category, among others, and probably will be one of the first products listed. Click on the links that lead to a downloadable version of Squishdot; the latest version as of this writing is 1.3.0. Notice how even a product of moderate complexity is relatively small; the Squishdot version that I downloaded was a little more than 256KB.

To install Squishdot, we must unpack it into lib/python/Products. Assuming that we place newly downloaded files in /downloads, this means that we can unpack Squishdot in the following way:

```
# Set this to your Zope home
export ZOPE=/usr/local/zope
# Switch into the products directory
cd $ZOPE/lib/python/Products
# Unpack Squishdot into the current directory
tar -zxvf /downloads/Squishdot-1-3-0.tar.gz
```

Older Zope products expect to be unpacked from the Zope root directory, rather than from within lib/python/Products. Unfortunately, there does not seem to be any obvious way to know how a product was packaged without looking at it:

```
tar -ztvf /downloads/ProductName.tar.gz
```

If each filename begins with the lib/python/Products pathname, then you will want to switch into $ZOPE, rather than $ZOPE/lib/python/Products, before unpacking the product.

Unpacking the archive is all we need to do in order to install Squishdot. However, Zope only looks for products when it starts up; we must restart the server before we can create instances of Squishdot on our system. The best way to do that is to click on the Restart button from within the control panel. Don't panic if your browser complains that the server is no longer running after you click on Restart, or if you see an obscure-looking Python exception backtrace after clicking on the Restart button. Rather, wait several seconds before clicking again on the control panel link in the left-hand frame, and it should work.

You can check to see if your product has been added by returning to the Product Management page within the control panel. If the newly installed product (Squishdot, in this case) does not appear on the list, double-check that it was unpacked correctly and that the permissions allow the Zope user access to the product's files.

Using a Product

At this point, we should be able to create a new Squishdot site by moving to the root (/) directory of the Zope server, selecting Squishdot site from the selection list and clicking on Add. This invokes the methods named in the constructors parameter to context.registerClass, invoked by the initialize function in Squishdot's __init__.py.

And indeed, we could move ahead and create our Squishdot site at this point. But Squishdot uses the Zope MailHost object (which represents an SMTP server) to send e-mail notifications. If you have not yet created and defined a MailHost, the Squishdot configuration screen will remind you to do so.

When Squishdot looks for a MailHost, it begins its search in the current directory. If it does not find a MailHost object, the search continues up the directory tree, stopping when Squishdot reaches / or when it finds a MailHost object. While this might appear to be a simple issue, it demonstrates the concept of acquisition, which is central to Zope. Moreover, it means that different Squishdot sites can send e-mail via different SMTP servers, simply by creating more than one MailHost object. Indeed, we can define a global default MailHost in /, overriding it as necessary by placing additional MailHost objects in subdirectories. The concept of acquisition permeates Zope and means that we can define or redefine nearly anything—MailHosts, users, headers and stylesheets—at a local level.

In this particular case, we will create an instance of MailHost in the / directory by choosing MailHost from the new product list and clicking on Add. Because a MailHost object represents an SMTP server, the configuration of this object is pretty straightforward, requiring that we enter the name of our Zope server's SMTP server. Most Linux machines run their own mail servers, so “localhost” is probably a reasonable value.

The mandatory ID field is used to identify this MailHost uniquely within the current directory, which is why Zope uses IDs to identify objects in URLs uniquely. Just as a filename is a unique identifier within a directory, a Zope object ID is a unique identifier within a folder or other object. The optional Title field is meant for humans, rather than for the underlying Zope server; if it is defined, an object's title is displayed from within the Zope server interface.

After you have created your MailHost object, you will be returned to the main Zope management screen for /. You should see your new MailHost object (represented with a small envelope icon), along with any title that you defined, in the list of objects.

We are now ready to create our Squishdot site. Add a new Squishdot site object using the selection list and Add button in the upper right-hand corner, choose an ID (i.e., URL pathname), optional title and mailhost, and then select some other basic parameters for your Squishdot site. For example, I chose an ID of atf and otherwise left the configuration options with their default values.

To enter my Squishdot site, I now tell my web browser to display http://localhost:8080/atf/. Zope receives this request for /atf and sees that we are referring to a Squishdot object. Zope then asks this object to display itself. Sure enough, we see an introductory screen that looks something like Slashdot but is powered by Zope.

We can create as many Squishdot sites as we might like, keeping in mind that every new site must have its own unique ID. In this way, we can set up one moderated site, one unmoderated site and another internal site for our organization's own uses—each with its own URL, potentially protected with its own set of users and groups.

To modify the Squishdot site, simply append /manage to the name of the object you want to modify, as in http://localhost:8080/atf/manage. This invokes Zope's management system on our Squishdot site. Using tabs at the top of the screen, you can modify nearly any parameter having to do with Squishdot, from moderation rules to the color of the text in which the site name is displayed.

Conclusion

This month we discussed Zope products and saw how to download, install and configure products on our system. While products are inherently more complex than simple DTML pages, their centralized code and additional flexibility make them more suitable for serious tasks than DTML.

Next month we will look at how we can write our own Zope products using a combination of Python and DTML.

Resources

email: reuven@lerner.co.il

Reuven M. Lerner is a consultant specializing in web/database technologies. His book, Core Perl, was published by Prentice Hall in January 2002. He lives with his wife and daughter in Modi'in, Israel.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Scriptwriting for ze Web and Everywhere Else

Marcel Gagné

Issue #95, March 2002

Marcel introduces some easy ways to organize and outline your thoughts for your next screenplay.

François, *mon ami*, what are you doing? Ah, I see your creative juices are flowing, *non*? Dragons, spaceships, world-spanning Linux systems...that is a marvelous story. And, I see that you have included the restaurant as part of the setting. This is wonderful! But why this flurry of cranial activity, François?

What is this? Ah, the March 2002 *Linux Journal* theme is web scripting and you are writing a script for an upcoming webcast. You want to write scripts for the Web? Why limit yourself there? Why not write for television, radio or the stage? With the right tools and your Linux system, you are ready for anything. As it turns out, it is a good thing that we are always ready, because our guests are here.

Vite, François, to the cellar. Bring up the Australian Margaret River Chardonnay. A wonderful wine, thoughtful and aromatic. *Vite*!

While François is getting the wine, I will let you know what was happening here, *mes amis*. This month's theme was, alas, misinterpreted by my faithful waiter. When you came in, he was working on a screenplay for what he hopes will be a successful webcast. Rather than discourage him, we'll concentrate today's menu on that very topic, bringing stories to life. Ah, François, excellent. Please pour for our friends.

Laying out a scene and creating the perfect dramatic presentation requires some finesse with outlines and organizing your thoughts. That is exactly what Peter Teichman was, er, thinking about when he wrote Think, a Gtk/GNOME application that lets you create outlines based on a hierarchical structure. The result is then saved as an XML document. For a peek at Peter's Think, go to

primates.ximian.com/~peter/think and pick up the latest source. After extracting the code, you can then build it using the classic ./configure three-step:

```
tar -xzvf thnk-0.2.1.tar.gz
cd think-0.2.1
./configure
make
make install
```

To run Think, just type **think &** at the command prompt. Figure 1 shows Think in action.

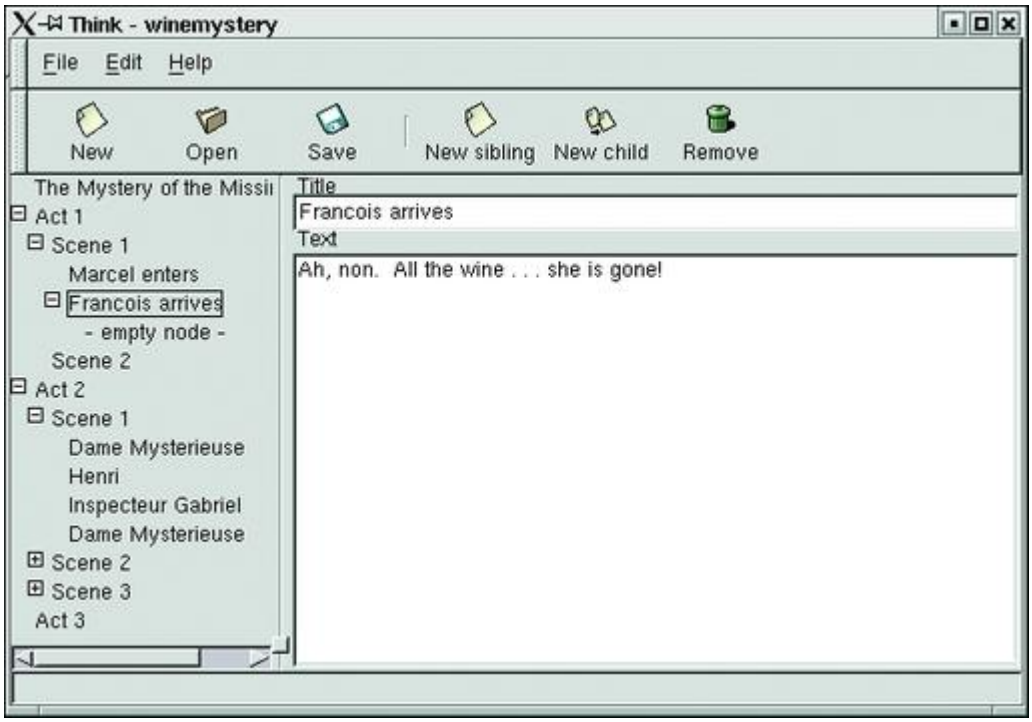


Figure 1. GNOME Think

When using this simple little thought organizer, you create nodes and subnodes, renaming them as you go to something that makes more sense to you than "- empty node -". Each one of these nodes can have associated text. For instance, dialogue text describes scene two, a subnode of Act 1. Each of these nodes can be dragged and dropped in different locations on the left-hand-side listing. To simplify the structure, each major node can be collapsed to a single line item.

Of course, you do not have to download a thing if you do not want to. Odds are pretty good that you already have that fabled editor, Emacs, loaded on your system. You don't even need to be running a graphical desktop because Emacs will run in text mode as well. Why not try out its outline mode? It is all very simple really. Here is what you do.

Start up Emacs by typing **emacs *some_filename*** at the command line. If the file does not already exist, it will be created for you. This is the basic Emacs editing mode. To use the outlining feature, press Esc-X, followed by the words **outline-mode** (see Figure 2).

Figure 2. Entering Emacs Outline Mode

All you have to do is start typing. For each level (or node, in Think-speak), type an asterisk (*****). For a sublevel, type two asterisks (******) and so on. You can type whatever you want below those headings, just as you would in any editor. To collapse a level, sublevel or tree, Emacs employs Ctrl-C sequences. Table 1 shows some of those sequences.

Table 1. Emacs Control Sequences

Of course, if you are running Emacs in X or graphical mode, you can just click on the Show and Hide menu items in the menubar. When a level is collapsed, its first line or title will appear with the trailing ellipses (**. . .**) and the information that follows it will be hidden. If you look at Figure 3, you'll see me spinning the “Mystery of the Missing Wine” using Emacs' outline mode.

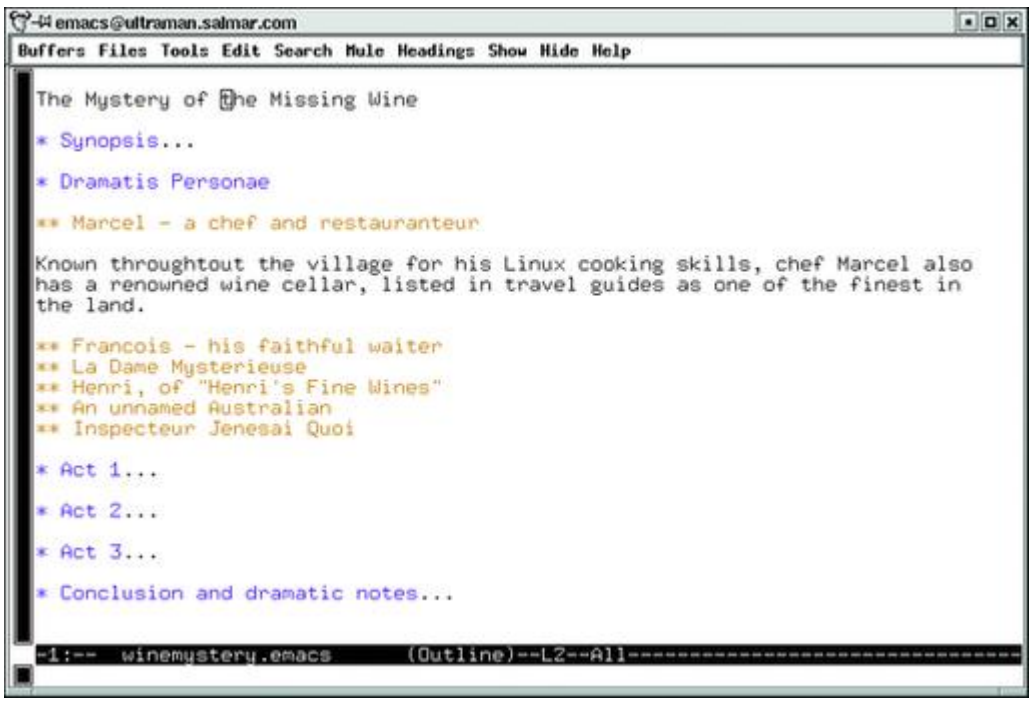


Figure 3. The mystery unfolds with an Emacs outline.

When Christopher Tomaras Lansdowne wanted to write some comedy sketches, he turned his hand to a little Perl and XML and put together

GScriptWriter, a Gtk tool that lets you define the basics of a story, enter characters, then pull them from a drop-down list as you create dialogue and action for them. This simple program also does a nice job of exporting to HTML.

You can pick up GScriptWriter by visiting the site at cs.alfred.edu/~lansdoct/linux/gscriptwriter. Once you have the source, install it like this:

```
tar -xzvf gscriptwriter-0.1.2.tar.gz
cd gscriptwriter-0.1.2
./gscriptwriter.pl
```

As you have already noticed, there is no compilation involved here. Simply run the script and you'll get a nice Gtk interface wherein you can define and work with all aspects of writing a comedic script. Right now, François is hard at work on a mystery taking place in this very restaurant.

You can organize the flow of your script by selecting lines and then moving them up or down in your script. Save your work, then use the HTML export feature. You'll wind up with a nice, clean finished product (once the mystery has been unraveled, of course). GScriptWriter is a work-in-progress, but the Perl code means you can modify it easily and use it as the base for your own needs.

For some, stories are those things you read on Slashdot.org that just don't come quickly enough. To rectify this problem, point your browser to bbspot.com/toys/slashtitle/index.html, a rather amusing site that generates pseudo-random Slashdot stories. Pay them a visit, *mes amis*, and you will understand what I mean. But we digress, *non*?

As you can see, the word story carries a lot of different meanings. Message boards are a collaborative effort that allow users to work together on ideas in a central location via the World Wide Web. A story develops with one individual posting the beginnings of an idea. Others respond, continue the discussion, and suddenly you either have chaos or the beginnings of the next great drama. In the right environment, a flexible message board can be a wonderful tool.

The next item on the menu is called Phorum. Phorum is free, open-source software and is distributed with a simple, Apache-like license. It requires a PHP-enabled, Apache server and one of the supported database types. This database can be PostgreSQL, MySQL, Sybase and others. Phorum is designed to be pretty database-independent, and the install will do an auto-detect for currently running database servers.

Speaking of installation, you should find this one very simple. Start by visiting the Phorum web site at phorum.org and grab the latest source distribution. Start by extracting the source into your web server's hierarchy:

```
tar -xzvf phorum-3.3.1a.tar.gz
mv phorum-3.3.1a.tar.gz
/usr/local/apache/htdocs/phorum
cd /usr/local/apache/htdocs/phorum
```

From here, you may want to take a moment to run the **secure** script to define locations and permissions relating to Phorum's security. You will find it in the scripts directory:

```
bash scripts/secure
```

If you are running a virtual web server, this may be particularly important. Since the layout of Phorum may be well known, you may want to change the name of the admin directory. That is the first question you will be asked. I would suggest that you also answer yes to the question regarding protecting this directory with an .htaccess file. Finally, the Phorum install will ask you what user name your Apache server runs as. In my case, it was www.

The remainder of the installation is done on-line. You do this by pointing your browser to the admin directory. If you changed it when you ran the secure script, you'll want to use that pathname:

```
http://yourwebserver/phorum/admin
```

Phorum automatically will try to detect your installed database. If you have multiple database packages (as I did), you will need to choose one at this time. For my install, I chose PostgreSQL. After clicking on Submit, I was then asked for my database server name (localhost), the database name, and a user name and password. Before you go ahead and fill in all those fields, let me give you a word of warning. That database user will have to be one that actually exists, unless your database installation requires no username and password (which wouldn't be a good thing). In the PostgreSQL world, adding a user looks like this:

```
createuser user_name
```

That user needs to be able to create and update databases. Then, you'll need to create an empty database. I called mine (perhaps unimaginatively) phorum. We do not have time to cover all the databases, but if you should happen to be running PostgreSQL, make sure that the “postmaster” is running with the -i flag to allow for TCP and web connections to the database:

```
/usr/bin/postmaster -D /var/lib/pgsql/data -i
```

Meanwhile, back at our browser installation, I fill in the remainder of my fields, click Submit, and then I am presented with a nice message about table creation and database updates being successful. A toast, *mes amis*! Ah, but wait. As they say on *la télévision*, there is more. I need to specify an admin user name and

password on the next screen. Finally, on the last screen, a URL to your Phorum and an e-mail address for the administrator. Click Submit one final time and you are done.

You then will be directed automatically to the admin login screen. Enter the newly created Phorum admin name and password, and it is now time to beautify your collaborative world. You can change the color scheme as well as define the number of messages per page, whether your collaborators are allowed to post with attachments and what size those attachments can be. Figure 5 shows Phorum's admin screen.

Figure 5. Phorum's Admin Screen

Since discussions aren't much use without topics, you may want to spend a few moments to seed your new Phorum with discussion topics. The associated dialogues are simple and easy to use, allowing you to specify who may post, whether the discussion is moderated, as well as display options like color and layout.

You've configured things, created forums and discussion groups, and chosen your colors. It's time to turn Phorum over to the users. Because of its simple nature, you can easily plug it into an existing web site by loading it as a frame, for instance. Many of the sites that use Phorum do so in exactly this manner. You also can have them point their browsers to the phorum hierarchy on your web site:

```
http://your_website_address/phorum
```

Now users can create their own profiles, read, post and reply to other posts. Whether they actually require a login profile will depend on the requirements you set up when you created the discussion areas through the Phorum admin screen. Figure 6 shows a Phorum discussion in process.

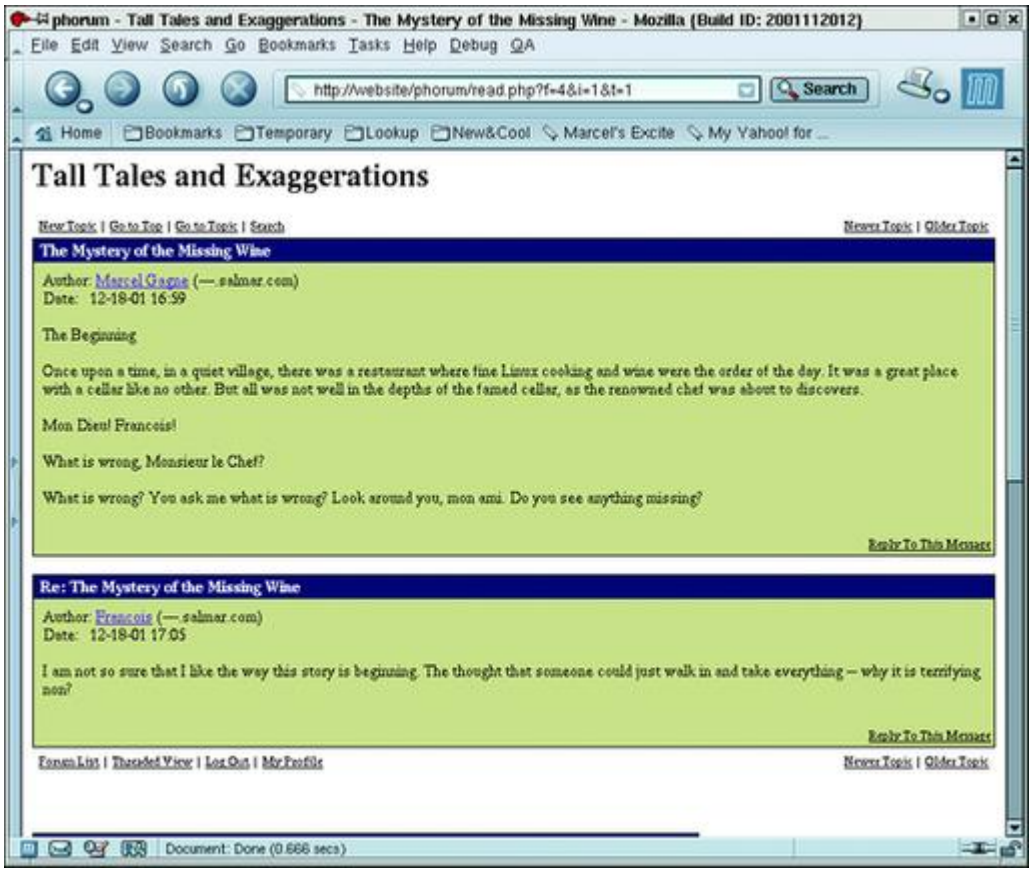


Figure 6. Funny Things Going on at the Phorum

Once again, *mes amis*, it is just about closing time here at *Chez Marcel*. The clock, she is unforgiving, *non*? François, our guests' glasses are running dry. Please, refill them a final time before we close for the evening. Until next time, *mes amis*, thank you once again for visiting *Chez Marcel*. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné (mggagne@salmar.com) is president of Salmar Consulting Inc., a systems integration and network consulting firm, and the author of Linux System Administration: A User's Guide, published by Addison-Wesley.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Film GIMP at Rhythm & Hues

Robin Rowe

Issue #95, March 2002

Robin introduces Film GIMP, an open-source Linux tool being used in major motion pictures.

The GIMP probably leads any list of killer applications for Linux. This Photoshop-like graphics package is very popular for retouching still images. However, fewer people are aware of its motion picture cousin called Film GIMP, intended for working on a series of images.

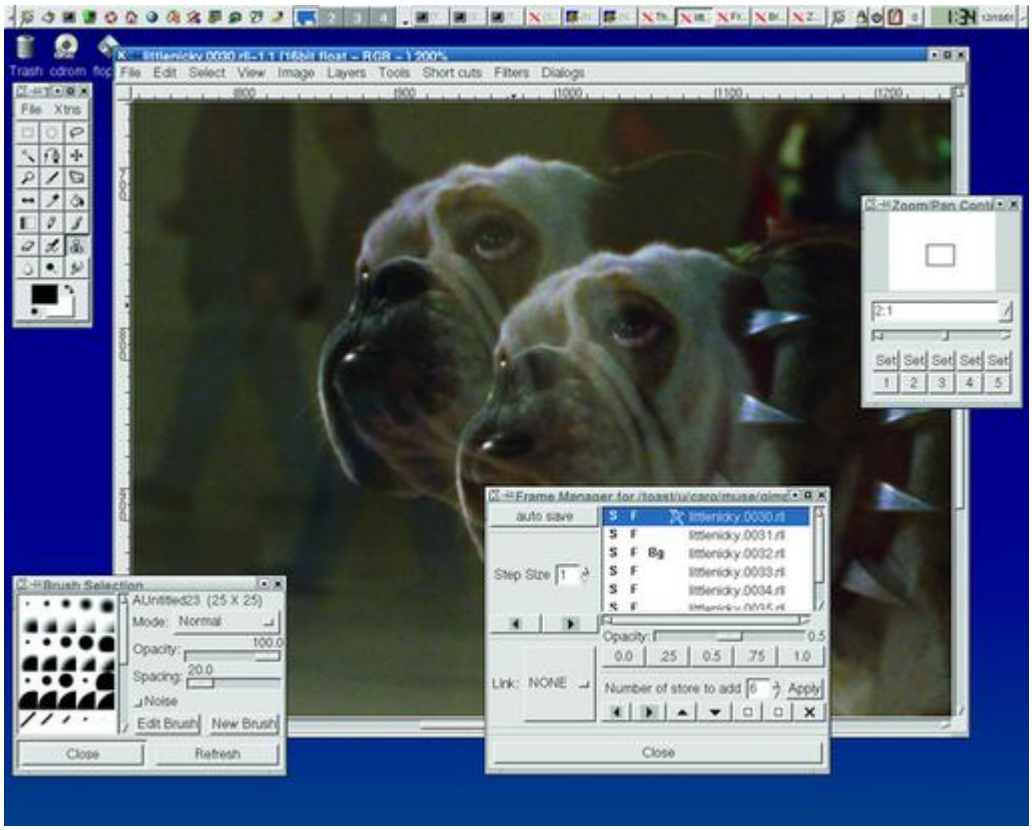
Linux is now being adopted in motion picture production at DreamWorks, Pixar, ILM and many other major studios. Most of this Linux effort involves commercial tools such as the popular 3-D animation package Maya (see the article “Alias|Wavefront Maya 4” in the October 2001 issue of *Linux Journal*) or obscure internal tools representing an investment of millions of lines of code created by the studios themselves (see “DreamWorks Features Linux and Animation”, August 2001 *Linux Journal*). Today there is just one significant open-source Linux tool being used in major motion pictures. Let's take a look at Film GIMP and its use at Los Angeles film and television commercial postproduction studio Rhythm & Hues.

R&H programmer Caroline Dahllöf is a lead developer and maintainer of Film GIMP. “We use Film GIMP on all talking animal jobs”, says Dahllöf. “Film GIMP is in use in production by various studios but probably is used most by R&H. Other studios say that they think GIMP is a great idea, but we seem to be the only production house currently developing and supporting it.” Dahllöf would like to see other studios become more involved in development. At R&H, Film GIMP has been used in *Harry Potter*, *Cats & Dogs*, *Dr. Dolittle 2*, *Little Nicky*, *How the Grinch Stole Christmas*, *The 6th Day*, *Stuart Little* and *Planet of the Apes*. R&H also creates commercials, such as the familiar Coca-Cola bear commercial. Dahllöf says:

> Our big thing is talking animals. We'll create those in 3-D because we like the look much better than 2-D morphing. We'll make a 3-D model and track it to the plate, matching the movement of the live animal with a CG animal head. Then the lighting department projects the frame on to 3-D model. The 2-D department fixes missing background parts as the animal talks.

Stretching has to be fixed with textures. The mouth interior is all CG. Some projects, such as the Coke commercial, are all CG. That's a different technique than making live-action animals talk.

As is typical with production studios, R&H uses not just one tool but a pipeline of tools for 3-D animation and live-action special effects. Before looking at Film GIMP, let's examine some of the proprietary tools in the production pipeline that Film GIMP must seamlessly interact with.



Film GIMP Screenshot—Working on

Film GIMP Screenshot—Visa Commercial

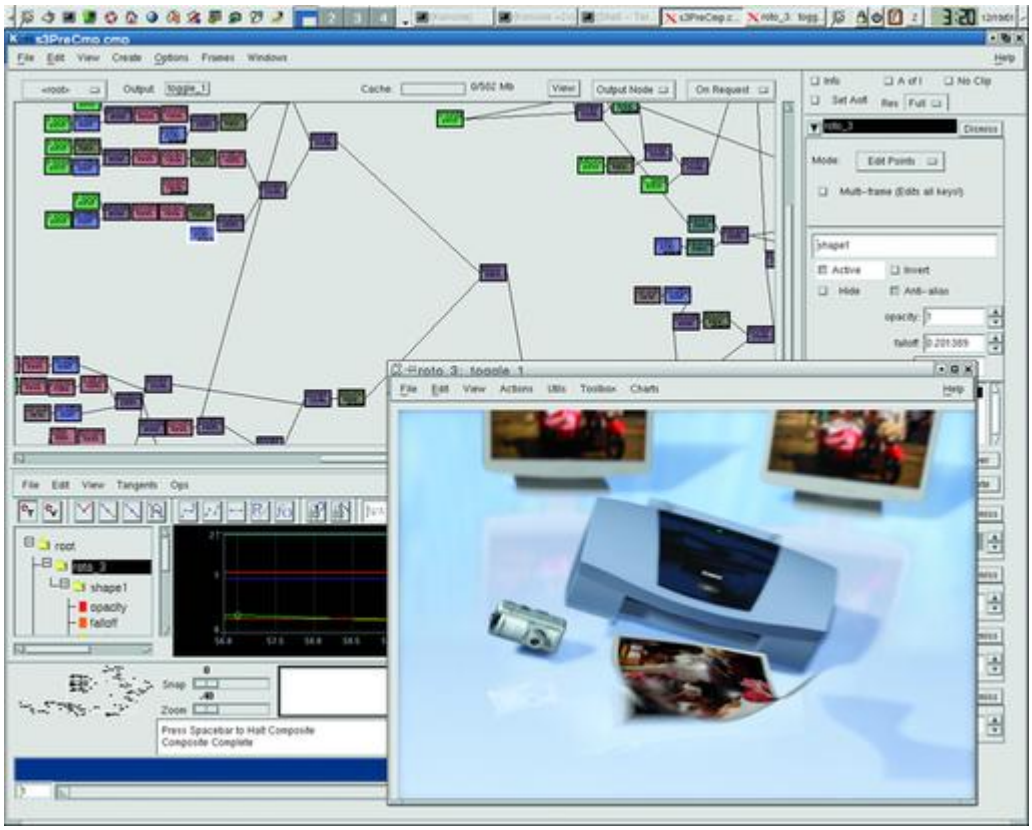
3-D Modeling with And

For 3-D modeling, R&H uses an in-house modeling tool called And and Maya. Modeler/TD Yeen-shi Chen explains:

> For a TV commercial I am creating a model of a cat wearing a wet suit. The cat model was retrieved from our model library and modified to match the cat in the commercial. The wet suit and diving gear are added later. The entire model was built in And. I create the model in a neutral pose because that makes it easier for the setup people to put a skeleton in it.

Compositing with ICY

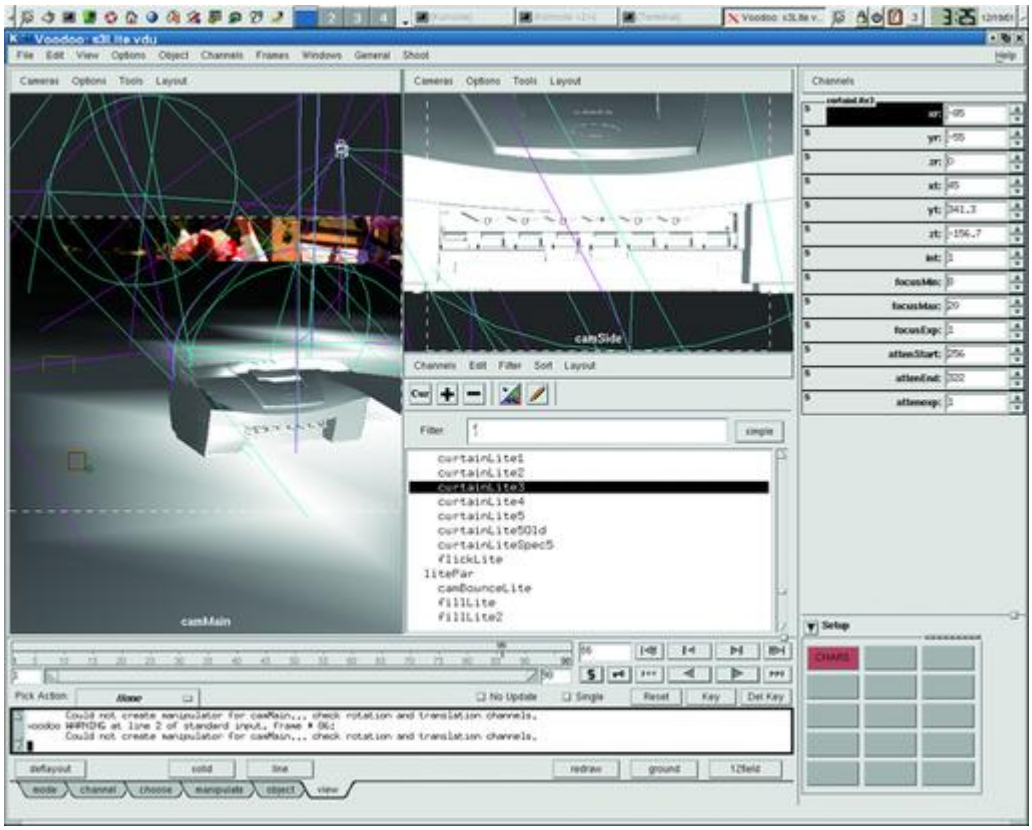
Technical Director Jeff McLean explains how R&H uses IC (Interactive Compositing), their internally developed compositor: “A typical task is removing the arms of a skateboarder from a scene in *Scooby Doo* to be replaced by Scooby in a barrel.” (*Scooby Doo* is due for release June, 2002.) When McLean replaces a live actor with an animated figure like Scooby, he must not just cover up the actor but handle situations where the actor's movements extend beyond what the animated figure will cover. “I have to do a background replace and repair the missing pieces of the background when I remove an object from a scene”, says McLean.



ICY Screenshot—Canon Commercial

Lighting with Voodoo

Lighting TD Greg Yepes uses internally created lighting tool Voodoo on projects such as *Harry Potter* or Canon printer commercials. “When setting up lighting for 3-D objects composited into a scene I'll initially set my virtual lights to an extreme value so I easily can see what I'm doing”, says Yepes. “Later it will go into Wren with more subdued values.” Similar to RenderMan, the Wren software is their internal rendering engine ported to Linux a year ago. “All the apps have gotten ported faster than we had thought”, says Yepes. “I started with DEC Alphas on *Cats & Dogs*, then the Intel PCs came on-line and really saved us.”



Voodoo Screenshot—Canon Commercial

Voodoo Screenshot—Canon Commercial

Using Film GIMP

Film GIMP replaced an internally developed paint tool called Inc. “We like to use Film GIMP to dust-bust”, says Dahllöf. “There might be dust or a piece of hair on the plate either from scanning or from the negative. Normally one will clone/merge from the previous frame or from the same frame to remove the dust.” Film GIMP also is used for rig and wire removal, for example on *How the Grinch Stole Christmas*. Lighters may use Film GIMP to edit fur control files. “These control files are used by our internal fur program, Fur. Lighters also use GIMP to paint textures and make paint fixes to frames”, says Dahllöf.

“As the studio moves to Linux, more and more people will use Film GIMP on Linux”, says Dahllöf. Because she is evaluating SGI-based commercial paint programs, Matador and Illusion, Dahllöf usually works with Film GIMP running on SGI, not Linux. “I'm looking to see what features I need to add to Film GIMP to make it an equal tool. We need a good paint tool for Linux that our 2-D department can use. We have not found a satisfactory commercial solution.”

“GIMP didn't support any sequence work, which is important for a 2-D artist”, says Dahllöf. “A lot of their work is cloning from one frame to the next in a sequence. So, we added a frame manager.” R&H uses its own proprietary RLL

file format. “A major feature of Film GIMP is 16-bit per channel color, and it is compatible with our file format. We do not get any of the color loss when using Film GIMP that we can get with other 16-bit paint packages.”

Downloading and Building Film GIMP

To get Film GIMP we had to check it out of anonymous CVS. There is no tarball, RPM or deb. You must build it from source. The Film GIMP branch is named HOLLYWOOD:

```
cvs -z3 -d:pserver:anonymous@gap:/cvs/gnome checkout
-r HOLLYWOOD gimp
```

We had configured our firewall PC named gap to have its port 2401 point to the server anoncvs.gimp.org. If we weren't behind a firewall, we would have specified that server directly in the cvs checkout command rather than our proxy.

After the 18MB download from CVS completed we had to make some minor corrections in order to build. In gimp/plug-ins/Makefile.am we had to delete rll, pts, fm_pts and parsley from the list of directories included in the SUBDIRS variable there. These plugins will not build in Film GIMP and would cause the build to fail.

```
cd gimp
libtoolize --force
aclocal
automake
autoconf
./configure -prefix=/usr/local
make
./app/gimp
```

The Linux Conversion at R&H

Technology VP Mark Brown says, “We have 50 Linux machines as desktops now and will have 250 Linux desktops by the end of 2002. We also have a 200-node Linux renderfarm, but that expands or contracts with demand.” R&H wrote their our own virtualized filesystem to support PTS, their production tracking system. R&H uses the ext2 filesystem.

“We decided against actually building our own boxes and have ordered 100 Dual processor 1.5GHz AMD machines with Angstrom Microsystems”, says Brown. “This was the most cost-effective move at this time. All our current desktops are dual PIII. We will support a completely heterogeneous environment. Video cards and CPUs are two of the things we know we can't keep completely consistent at the desktop.”

History of the Development of Film GIMP

Film GIMP came about thanks to the patronage of R&H and software development company Silicon Grail. Each hired an OSS GIMP programmer for a year: GEGL designer Calvin Williamson at R&H and GIMP maintainer Manish Singh at Silicon Grail. Silicon Grail founder Ray Feeney explains, “We had done some other open-source projects, such as film recorder drivers, and saw enhancing GIMP as an opportunity to do something with the Open Source community.”

Silicon Grail RAYZ product manager Craig Zerouni says, “We did a little work integrating GIMP into our compositor Chalice as a plugin. But in the end we decided a nonprocedural paint program didn't fit well into a procedural program like Chalice.” Silicon Grail was working with GIMP script-fu to create a series of Film GIMP commands that could be saved in Chalice. However, that work was abandoned when Silicon Grail developers switched to begin development on their new compositor product, RAYZ.

Zerouni feels a true procedural language is needed in the paint program, something like the language in RenderMan. Silicon Grail has lately acquired the Cineon source code from Kodak, including the program Retoucher. “Film GIMP was a useful thing for us to do”, says Zerouni. “We learned a lot about what paint should be.”

GEGL and the Future of Film GIMP

GEGL, the GIMP E Graphical Library, is an image-processing library based on GObjects. GEGL developer Calvin Williamson helped develop Film GIMP originally while at R&H, together with Ray Lehtiniemi from Silicon Grail. The next version of GIMP will be 1.4, but Film GIMP continues in development on a branch of 1.0.4. GEGL, whose design supports 16-bit channels, is due to integrate with GIMP 2.0, perhaps two years away. GIMP 2.0 is anticipated to bring the pro features of Film GIMP into mainstream GIMP.

Williamson says his current plan is to write a baby compositor for GEGL to test memory management for large images, multithreading, large composite trees and other heavy-duty professional requirements:

> The classes that do image and memory management have been split from the actual image-processing classes. This allows one to write image managers that traverse the graph of ops in custom ways, or write custom caching or memory managers for handling memory management. The classes that hold information about ops as part of graphs, with inputs, outputs, regions of interest (all extrinsic op info), have been separated from image-processing classes

> (intrinsic op info) as well. This makes graph traversals cleaner for things like multithreading.

“GEGL is still in a very early phase and many classes are under construction”, says Williamson. “There is no official release yet, but you can download it from anonymous CVS. I have made quite a few architecture changes recently.” GEGL is a fully 16-bit image engine for future GIMP and other projects.

Conclusion

Both GEGL and Film GIMP seek volunteers to help with programming. Williamson says PDI, ILM, ICT and Sony studios have expressed interest, but so far have not provided programmers. Williamson welcomes programmers interested in writing image operators, memory management code and working on multithreading to join the GEGL Project. “Filling out the library and writing image operators takes time”, notes Williamson. “GEGL is an important part of the future for GIMP.” For programmers who find the GEGL timeline too far away, Williamson suggests helping with enhancing Film GIMP for the next year or so.

Film GIMP maintainer Dahllöf says, “We want to enhance Film GIMP by adding more tools for artists working on a sequence of frames. We want to make it easier for them to paint and clone from one frame to another one.” R&H uses a proprietary flipbook player called Flicks. A feature missing from Film GIMP is a flipbook playback mode so users can detect flickering. “Artists do use filters, but some of the filters in the main branch of Film GIMP are not useful in motion pictures. Artists want more control over filters.”

On the topic of open-source software, R&H principal software engineer Green says:

> It's a big debate about releasing more of our proprietary software as open source, beyond Film GIMP. Most of our software would be difficult to use outside this building. It is tied to our production tracking system, PTS. Nothing will run without that.

Green says the biggest pro argument is to save on the high cost of training. New hires at R&H spend their first month just doing training. Having a pool of talent emerging from the universities already trained in R&H tools would help.

Resources

Trademark Information

email: Robin.Rowe@MovieEditor.com

Robin Rowe (robin.rowe@movieeditor.com) is a partner in MovieEditor.com, a technology company that creates internet and broadcast video applications. He has written for *Dr. Dobb's Journal*, the *C++ Report*, the *C/C++ Users Journal* and *Data Based Advisor*.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Putting Linux in Classrooms around the World

John D. Biggs

Issue #95, March 2002

An early start with Linux will allow the next generation of students—everywhere—more and better job opportunities.

Forty-three top students at the Shree Bachhauli Secondary School in Bachhauli, Nepal are learning computer programming, a skill that could keep them out of the child-labor market and rocket them into higher education and a real job after graduation. Their school has 14 teachers and over 600 students, but the computer classes are kept small and staffed by German and Swiss volunteers who work for a group called Ganesha's Project. They make do with donated machines and focus on open-source software like Linux, a move that cuts the cost of acquiring software licenses for an already impoverished school system.

“The main goal of Ganesha's Project is to try to create a humane alternative to child labor in Nepal”, said Kirstin Boettcher, a German graphic designer who is working with the group to raise funds. She added,

> In Nepal, as in other third-world countries, most people don't profit from the spread of progress. The point of our Project is to narrow the digital divide that keeps computing resources away from the masses and to show people how to bridge the chasm of poverty with education.

Ganesha's Project is only one startling example of under-funded and under-staffed schools around the world that are turning to Linux as a way to create an inexpensive and intensive computer curriculum. Like Apple's early efforts (in the 1980s) at giving free Apple II machines to schools, Linux software distributors and volunteer organizations are gaining life-long users by delivering open-source technology at little or no cost to elementary and high schools.

The key to this program's success is the freedom that administrators and teachers gain in using open-source software. Peter Farina teaches computer science using open-source software at Montini Catholic High School in Lombard, Illinois, a suburb of Chicago. “Once the students get past the hurdle of learning the Linux commands and the Linux directory structure, they get pretty excited about it”, he said. “Then they find that there are thousands of programs out there that are free for them to grab. It's not like they have to get some bootleg copy from a friend.”

Farina's school is part of SuSE's Free Linux for US High Schools Program. As part of the program, SuSE donated over 2,000 copies of their version of Linux to high schools across the nation.

“Having it in high schools around the US demonstrates the open-source philosophy and shows students, teachers, administrators and IT specialists that they do not have to be tied to expensive operating systems and ongoing costly upgrades”, said Dirk Hohndel, president of SuSE's US operations.

Like the volunteers at Ganesha's Project, Farina also is faced with a cash-strapped department, and he is turning to Linux to reduce costs.

“We're at a point now where we're trying to increase the size of the network”, he said. “The cable and infrastructure is in place, but licensing is so expensive. It's crippling us. I'm trying to find a way to increase our offerings without paying through the nose for each PC that's on the network.”

Farina uses Linux as a teaching tool and has gone as far as to instruct students how to build their own small computer networks. He said the biggest stumbling block was trying to convince teachers to learn the new operating system. He explained that most of the teachers were just beginning to feel comfortable with Microsoft Windows products, and that he “would get a lot of grief” if he sprung Linux on them too suddenly.

Students, parents and teachers at New York's Beacon School, on the other hand, use Linux daily but hardly know it's there. Shantanu Saha, deputy director of technology for the New York Board of Education, says Linux is the backbone of the Beacon School's network.

The Beacon School's web site, which runs Red Hat, provides an outlet for news and announcements. It also includes a parent/teacher interaction system that makes the many messages broadcast on-line by the administration hard to miss.

“This site is largely developed and maintained by the students, and it improves on every iteration”, said Saha. “I've been doing my Linux initiative purely as an opt-in program, with interested schools participating in a workshop that I ran this year”, he said.

So far, Saha plans to install networked servers running Linux to handle most e-mail and web-related tasks for each of the 70 schools in the area. He explains:

> My current paradigm is to install Linux servers at selected schools as the core of their networks, to train the teachers and technicians in basic administration and maintenance, and to help them out by managing the servers remotely, without traveling to the school or sending a technician.

“I want to replicate success where I find it”, he added, citing the Beacon School's reputation as one of the most high-tech schools in the New York school system.

Volunteer organizations around the world also are trying to replicate the success of many open-source educational projects in their own areas. One volunteer organization, headed by Paul Nelson and Eric Harrison of the Multnomah County Education Service District, located near Portland, Oregon, developed the K-12 Linux Project, a system designed to allow schools to use old and outdated hardware to their fullest advantage through Linux networking.

“Schools get old hardware”, said Nelson, who spent the last 20 years with the Riverdale School District as an educator and, later, as a system administrator:

> These computers come with the hard drives wiped and no operating system, and you have to pay a one-hundred-dollar license fee to Microsoft to get it running again. With Linux, you don't have to have a fast or new computer to make it useful.

The K-12 Linux Project uses a central server to transmit GNOME to computers around the school. Currently, Nelson administers hundreds of computers in the two school districts where he works. He says the Project has taken off. “Kids require no training at all. They just start clicking. In a matter of days, they're experts”, he said. “That's how kids learn. We want the operating system to be an on-ramp, not a roadblock.”

“The K-12 Linux software is spreading”, said Nelson. Schools in London and Belize are running the program already, and he has had requests for the software from as far away as Malaysia and the Philippines. “It's a snowball at the top-of-the-hill stage. There's a lot of potential here”, he said.

Nelson believes that the open-source paradigm is the best way for schools to remain competitive in an international marketplace. The software is free, he says, the need is great, and “schools have no money.” He adds,

> With this Project, we have one computer for every three students. We're able to administer those remotely at locations around the building, something we could never do before.

The companies and volunteers that are offering services and software to schools are doing more than marketing their products, although a healthy capitalist instinct obviously prevails. MandrakeSoft is working with hardware vendors to supply schools in under-funded and low-income districts of Los Angeles, as well as poorer schools in Canada and Mexico.

“In the next two to three years, most Mexican schools will be running Linux”, said Daniel Morales, vice president for MandrakeSoft in the Americas. “We are donating software and some of the servers, just to put installations all together”, he said. “We, as a company, are emphasizing education and creating new talent to handle open source.”

Ultimately, Linux is sneaking into schools around the world and becoming as ubiquitous as the Apple icon to America's educators. The real bottom line, of course, is money. Saha believes that the kind of computer literacy provided by open-source systems, coupled with the traditional three Rs, is key to future success.

“If you want to learn about computers, you need to know the operating system that basically drives the universe—UNIX.” Saha believes that students who know Linux, and by extension UNIX, have a “license to print money” in any job market. Even the one in Nepal.

email: jdb252@nyu.edu

John D. Biggs is a writer and consultant in Brooklyn, New York.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Natural Forces

Doc Searls

Issue #95, March 2002

Doc muses on the curiosity of kids and the power of Linux as a building material.

It was almost three years ago that Dr. Sugara Mitra, head of NIIT's Centre for Research on Cognitive Systems, began quietly to put internet kiosks where kids hang out in the poorest parts of New Delhi. The effort, titled “Hole in the Wall”, was an experiment in “minimally invasive education”, a concept that offers this affront to the base assumptions of formal education: “that in the absence of any directed input, any learning environment that provides an adequate level of curiosity can cause learning.”

Here is what happened, the Hole in the Wall web site (www.niitholeinthewall.com) reports:

> The objective of this experiment was to check if people would be interested in using an unmanned internet-based kiosk out in the open, without any instructions. It also aimed at ascertaining if an unmanned kiosk can be operational without any supervision in an outdoor location.
>
> The boundary wall of the NIIT office where the computer was placed is adjacent to a slum, which has a lot of children from 0-18 years of age. Some of these children do not go to school, and a few who do, go to government schools that lack resources, good teachers and student motivation. These children are not particularly familiar with the English language.
>
> The results of the experiment have been quite exciting. Within three months of opening up of the internet kiosk, it was found that the children, mostly from the slum, had achieved a certain level of computer skills without any planned instructional intervention. They were able to browse the Internet, download songs, go to cartoon sites and work on MS

> Paint. They even invented their own vocabulary to define terms on the computer, for example, *sui* (needle) for the cursor, “channels” for web sites and *damru* (Shiva's drum) for the hourglass (busy) symbol. By the fourth month, the children were able to discover and accomplish tasks like creating folders, cutting and pasting, creating shortcuts, moving/ resizing windows and using MS Word to create short messages in the absence of keyboard. When the issue of whether the kiosk should be removed from the boundary wall arose, the children strongly opposed the idea. The parents also felt that the computer was good for their children. The kiosk continues to be operational today with approximately 80 children using it per day.

Over the next two and a half years, Dr. Mitra and his team mounted internet-connected PCs in 29 different walls in four different Indian cities. The experiment continues, but at this point a number of additional findings are clear:

1. On the whole, users do not damage or abuse the equipment.
2. Learning is social. Children learn faster in groups because members are eager to share what they know and learn.
3. Shy children are not left out. Girls especially assume organizing roles, throwing screen hogs off the computer to let quieter children have a turn. Kids even organize classes for each other.
4. Adults don't participate, although they believe the kiosks are good things. Kids are the users.
5. Users figure out how to improve the computers. One found a way to improve the quality of the music files that played on the systems' little speakers.

Perhaps the most telling discovery was that Dr. Mitra's subjects do not like being subjects. One of the messages he received from one group said in Hindi, “We have found and closed the thing you watch us with.” He was gratified: “It made me so happy! I don't think [as a teacher] you can have a greater reward than to have a child beat you at your own game.”

Now let's pause to raise the obvious irony. These kiosks ran (and run) various forms of Microsoft Windows OSes, which cost money. The hands-down winner OS for the self-teaching crowd is Linux, which is free. The only OS with a legitimate claim to all-world relevance is Linux. You can recite the rest of the virtues list. But rather than do that, let's visit one quote from another corner of the world:

> We believe LINUX can play a very important role in Latin American and Caribbean modernisation,

> constructing networks to permit a great number of universities, colleges, schools and educational centers to connect to the Internet in order to use this fabulous tool to improve their scientific and cultural levels. In a few words, LINUX is the tool that permits reducing the "technological gap" between the countries. LINUX permits access to "the most advanced informatics" implemented according to the reduced economic capacities in our region. LINUX is a new way to make informatics, where the most important thing is "the technical quality and personal solidarity".

That one comes from the United Nations by way of ctrlaltesc.org. In that same region, our own Phil Hughes (*Linux Journal*'s founder and publisher) has been working on Linux adoption in Costa Rica.

So why not do more here? Since NIIT has taken the lead on this altruistic venture, how about working with that company to get some Linux boxes out there on the streets? I see that NIIT has a Linux training partnership with Red Hat. Let's get a hardware company to step up and take it to the next stage.

The only impediments to progress here are conceptual. In the same way it's hard for the educational establishment to admit the transcendent power of kids' natural curiosity, it's hard for the business establishment to admit the raw usefulness of Linux and other free and open-source software.

Education-as-usual assumes that kids are empty vessels who need to be sat down in a room and filled with curricular content. Dr. Mitra's experiments prove that wrong. Software-as-usual assumes that its business is only about selling bits in manufactured packages. Linux proves that wrong, simply because it is being put to use everywhere. People are making money. Commercial software remains unharmed and contributes plenty of good on its own. It's a big world and a big business with room enough for everybody. But it won't include the world's untapped billions of curious souls if all we try to do is sell them packaged bits. We have to think bigger than that, and more practically.

Almost two years ago I observed that the software business eventually would turn into something much more like the construction business: concerned fundamentally with architecture, design and building—even using much of the same vocabulary. Prefab products like Microsoft OSes wouldn't go away but would thrive in a much larger context where countless professionals and amateurs simply chose the best tools and materials for the work that needed to be done. Some of those materials would grow, as it were, on trees. (I explained this once to a Microsoft guy, and he immediately got it: "You mean, Linux is trees. Endless lumber for anything." Yes indeedy.)

The world needs to deploy a lot more technology if it's going to find a way to clothe, feed, house, employ and involve everybody in the prosperity enjoyed today by a relative few. There's plenty of business to be had in making that happen. It's a matter of exploiting natural materials that renew themselves and only improve the more you exploit them. And, there isn't a naturally abundant building material that fills that description better than Linux—unless, of course, we're talking about the natural curiosity of kids.

email: doc@searls.com

Doc Searls is senior editor of *Linux Journal*. His monthly column is Linux for Suits. He is also a coauthor of *The Cluetrain Manifesto*.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Seven Kernels on Five Systems

David A. Bandel

Issue #95, March 2002

David looks at compatibility problems with distros, kernels and software.

I know I'm not exactly your average Linux user. I guess I have to ask, who is? But, I hope most users don't have some of the problems I see regularly. At home, I usually have five systems connected, and my business has a fair number of servers and access points on the Internet. What I've been finding, and with greater frequency, is that between software not being available on some distros and software that only builds on some kernels, I have four different distros running seven different kernels just on the five systems I have at home. As of this writing (latest kernel version 2.4.17-pre6), the latest (CVS) Internet PhoneJACK software wouldn't build on any of the four most recent kernels I installed; other software required kernels 2.4.4-2.4.8 to build. Non-overlap of kernel versions needed to build and run some software is why I have seven kernels on five systems. FreeS/WAN compiles on some, but not all kernels. Then some distros have their own problems. Mandrake 8.1, which has a lot of bells and whistles, didn't include wireless utilities or support for CardBus or my ORiNOCO card—easily remedied, but not necessarily so for newbies. Caldera has a nasty habit of providing upgrades, such as those to the kernel, but not upgrading the kernel version itself. They patch it up with security patches, then leave the source as 2.4.2. Needless to say, some software refuses to build when 2.4.4 or better is required. Most readers already will be aware of the numerous problems with Red Hat. Solution? I'm afraid I don't have one. But I do know that what I consider only niggling annoyances can become show-stoppers for newbies. And, no amount of support will push Linux into the Microsoft strongholds if these problems aren't resolved.

Vipul's Razor razor.sourceforge.net

Last month I railed against spammers. This month I have a cure. If you're running an MTA (I run several), you can stop spam via a simple global /etc/procmailrc recipe (or you can do the same just for your mail with your own

personal ~/.procmailrc). Mine shunts all spam received to a spam box. I've checked the messages sent to this spam folder for over a month, and not one legitimate mail has been misdirected. Now I'm averaging one spam every three days, which I report to the razor servers, and with spams reportedly up 650% over last Christmas season, I'm happy. Requires: Perl, Perl modules Net::Ping, Net::DNS, Time::HiRes, Digest::SHA1, Mail::Internet and a strong desire to be spam-free.

XNetworkStrength gabriel.bigdam.net/home/xnetstrength

Running a wireless card? Want to know how good or bad your signal strength is without running **iwspy** every few seconds? Well, XNetworkStrength will do it for you if you're on an X screen. You can keep an eye on your connection while you're working. Requires: libX11, glibc.

CGIpaf stafwag.f2g.net/cgipaf

Need a simple, safe way for users to change their password? These CGI tools are compiled C programs that provide security, but you'll want them accessible only via https (not much sense changing a password over an insecure link). They also allow for users to forward mail and return a mail message (*à la* vacation). Unfortunately, this program won't take mailing lists into account (as vacation will). Requires: libdb1, libpam (optional), libdl, glibc, web server with PHP.

NorthStar www.brownkid.net/NorthStar

NorthStar will help you keep track of your IP allocations, equipment and locations of same. In fact, one of the nicest things about this is the way you can view your networks, devices and locations. If you have more than a few IPs or systems or locations, you'll want to look this program over. Requires: web server, Perl, PostgreSQL.

Simplyfied CD Backup scdbackup.webframe.org/main_eng.html

This backup to CD utility is actually a number of small programs to permit specific actions. There's scdbackup_home that permits users to back up just their home directory. There's scdbackup_sys that permits a backup of the system. There's also just an scdbackup that can be fed arguments about which directories to back up and which to exclude from backup. Some backups (home directories, for example) are backed up as filesystems. Others, such as the system backup, are done as afio archives. Simplyfied CD Backup handles multivolume backups as well as single CD backups. Requires: cdrecord, mkisofs, bash, afio.

mail-bounce www.spots.ab.ca/~gary/mail-bounce

This small Perl program will allow you to take mail and bounce it back where it came from. It also permits you to include a custom message. While a message can be bounced at any time, it doesn't make much sense to bounce it back hours later, thus procmail is suggested as an easy way to bounce the mail, although any program, even a command line, can be used. Requires: Perl, procmail (suggested).

tknotepad ftp.mindspring.com/users/joeja

I had a difficult time selecting between two excellent packages I continue to use. And while I think E*Reminders merits mention, I chose tknotepad. I'm not sure why this hasn't been more widely adopted, but it provides Windows refugees a familiar haven. This tool looks, acts and works just like the Windows notepad. In fact, it's what I use to write this column, and it could be used to write web pages, edit configuration files and much more. Easier for most than my favorite editor, vi. Requires: Tcl/Tk.

Until next month.

David A. Bandel (david@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Bully in the (Embedded) Playground

Rick Lehrbaum

Issue #95, March 2002

Microsoft aims its guns at its next victim: embedded Linux.

Bully in the (Embedded) Playground

Microsoft aims its guns at its next victim: embedded Linux. But, the embedded Linux community quickly responds to the challenge.

by Rick Lehrbaum

Monopolist Microsoft was up to its old tricks in the fourth quarter of 2001, publishing a lengthy and one-sided comparison of their newly introduced Windows XP Embedded with embedded Linux. The document, titled “Why Microsoft Windows XP Embedded and Not Embedded Linux?”, compares XP Embedded to embedded Linux:

> Selecting an operating system (OS) platform is one of the first decisions an embedded developer must make for any given device design. Whether you are considering migrating from a proprietary to commercial platform, or from one commercial platform to another, the objectives are the same: accelerated time to market; a solid, extensible OS core that can be used across all projects; superior technologies that support differentiation and address emerging opportunities; and predictability over the device lifecycle. In addition, you want it at the lowest possible total cost from a reputable vendor who will support you throughout the process.

The document then proceeds to compare XP Embedded to embedded Linux, claiming, as you might expect, that XP is superior in each case: Integrated, Comprehensive, Unmatched, Interoperable, Proven, Global, Linux Is Not Free and OEM Licensing.

The Embedded Linux Community Responds

After learning of the Microsoft anti-embedded-Linux document, LinuxDevices.com issued a call to action, noting that “embedded Linux is not the product of a single dominant vendor, but rather is the result of the collaborative (and competitive) efforts of an entire market consisting of dozens of large and small companies plus thousands of individual developers”, and urging the embedded Linux community to respond *en masse* to Microsoft's attack.

The embedded Linux community quickly rose to the challenge, responding with sharply worded talk-backs, as well as in the form of lengthy, detailed and sometimes entertaining rebuttals from Lineo and LynuxWorks. Below are excerpts from a few of the many talk-back posts received.

- This is not the first time that MS tries to discredit Linux by the means of a self-made comparison of features Windows supposedly has and Linux lacks. For someone living in Germany (like me), this type of PR is completely strange, as comparative advertisement is not allowed here. With Linux, we started seeing this kind of ad in newspapers, as Linux is not a company but a free product that can't sue another company for any sort of commercials. I always wondered why they are doing this. If we put away our natural preference for everything that represents freedom and individuality, we could thank MS for telling us where Linux can be improved and where we find good arguments to sell Linux to potential customers. Every article of this kind generates the same reaction—and at the end, MS seems to be the loser.
- Microsoft is doing what it does best: going above the developers' heads to the managers and CxOs who have the power to Hand Down Decisions From Above. Embedded guys like to roll their own custom solutions and with good reason: horses for courses. Different devices have different requirements. PhBs like to purchase the one-size-fits-all solution and make it the company-wide standard. PhBs have enormous clout, otherwise MS would likely not be a monopoly. I don't know how much they have in the embedded space, but I figure with devices like cell phones, PDAs, kiosks, etc., gaining prominence, the PhB-to-clueful-manager ratio is increasing.
- Microsoft has very valid reasons to be worried about its long-term importance and future fiscal profits with regard to its diminishing relationship and relevance to the IS “Backend Iron” and embedded products markets. Microsoft dominates the x86-32 desktop market and that is where it will die.
- Well, speaking of PR or political support for Linux depends on where you look at it. Within the European Union Linux is getting more and more

heavyweight political support. The governments of France, Germany, Spain, Portugal and Finland have started a lot of open-source projects that are evaluating the possibility of getting rid of MS.

- Microsoft's idea of “embedded” is...something heavier than a thin PC client. The Microsoft Toaster would be a quad P4, two for each side of the toast, and would actually do the toasting by dropping to command-line mode for the required amount of time. The idea of fitting a whole OS, web browser, et. al., into 4MB of NVRAM and 8MB of real RAM is anathema to them. Embedded, to them, means maybe be able to squeeze into 32MB of RAM if pared back to the bone. They spec enough resources for their minimal system to run a decent e-commerce site on if you used Linux instead. If you don't wrap your mind around that perspective, you'll never understand where they're coming from.
- Microsoft decries the fact that Linux provides developers with a choice: “For example, there are at least five different window managers and at least four competing browsers....” In Microsoft's mind, it would appear that for a developer to have a choice as to which window manager or browser best suits their needs is a bad thing. For some applications, the massive—and often unused—features of something like Microsoft's Internet Explorer may be overkill, and its multi-megabyte footprint would be prohibitive. For Microsoft, these issues seem to be secondary to the mindset that having choice is a bad thing.

From LynuxWorks:

> We did some investigating of our own and are adding some commentary on the new [Windows XP Embedded] release as a contender in the embedded market. In general, we found that the operating system has limited applicability in embedded markets and doesn't have the clout to really take on embedded Linux in head-on comparisons. XP...has shortcomings as an embedded offering. There are some places it can go, but those are limited for reasons you will see below. The bulk of Microsoft's issues continue to be in size and performance.
>
> [This is followed by a long discussion of specific features and issues.]
>
> In summary, Windows XP is not as good an embedded solution as embedded Linux for the following reasons:
>
> Memory footprint—Windows XP has a memory footprint between 5 and 15MB, where Embedded Linux has a memory footprint of 259KB.
>
> Performance—the market has proven that Linux offers performance superior to or equal to Windows for servers. Given the additional factors against Windows

XP for embedded (size and complexity), this comparison will be more in favor of Linux for embedded applications.

Maturity—Linux subscribes to an OS model proven through 40 years of innovations. Interoperability issues, performance and general design have had an extremely long time to be tried and improved. Moreover, all these improvements have been done in a very diverse set of platforms.

Configuration—Linux is highly configurable, having been developed and deployed in memory-limited environments, in comparison to Windows XP, which has operated in memory-hungry monolithic environments.

Innovation—because of the open nature of Linux source code, it has become a nexus of activity in regards to innovative computing to a far greater degree than any Windows product.

Third-party support—thousands of applications, drivers and kernel extensions are available from open source as well as commercial vendors for Linux. The number is certainly comparable with Windows XP.

Networking—all major networking protocols, security features and extensions are available for Linux. In fact, many are implemented on Linux before other platforms.

Security—open-source nature of Linux allows the “many eyes” approach to be used to incredible effect. Security protocols, in particular, benefit greatly from this approach because their design and implementation are well documented and understood. A very stunning example of this is the NSA's recent release of a secure Linux.

Interoperability—Linux servers are among the key participants in the evolution of the Internet and, as such, offer a state-of-the-art interoperability solution. With Java providing all the benefits of Windows XP's .NET framework, Linux has a much greater degree of interoperability than Windows XP.

Cost effectiveness—development in any environment is the greatest expense. Having a diverse community for testing and deployment figures greatly in the success of Linux. Also, because of the highly custom nature of many embedded solutions, the highly configurable nature of Linux makes it particularly cost effective.

Support—Microsoft provides a single source of support for their product, limiting competitive offerings. In the Linux world, however, there are many

> choices among vendors who will provide support, solutions and software.
>
> Development tools—while Windows XP is primarily constrained to development under an IDE environment, Linux provides the powerful UNIX development environment in addition to IDE environments.
>
> Reliability—the deployments speak for themselves. Windows is rarely considered for mission-critical applications, where Linux is routinely considered for them.

From Lineo:

> Historically, such comparisons from Microsoft have consistently attempted to show negative elements about a competitor's solution while ignoring Microsoft's own shortcomings altogether. In this paper, the Microsoft authors seemingly forgot to address the relevant community of intelligent and capable software engineers, device manufacturers and media editors who would care about embedded system software. The content and spin of this Microsoft paper seems to assume an audience unfamiliar with competitive offerings....
>
> [This is followed by a long discussion of specific features and issues.]
>
> Microsoft has tied the web browser and windowing environment to the underlying operating system and defines these items as major OS components. Apparently they do not believe in product differentiation or choice. This philosophy runs strong throughout this Microsoft document.
>
> On the other hand, Lineo does not attempt to dictate to its customers what must be included in the final configuration. For example, many embedded products do not require a GUI or a web browser; therefore it would be presumptuous for any embedded operating system company to conclude that these are major components required in every system. Instead, the Lineo Embedix embedded operating system offers a “core” set of features necessary to provide a fully functional operating system, allowing the developer to pick and choose elements that distinguish a product from the competition. The truth is that this flexibility allows developers to innovate far beyond what is possible under Microsoft's closed-source model.
>
> Lineo provides fully open-source Linux with no per-unit costs. The developer is free to modify and use thousands of existing applications for their specific device. [Optional] license bearing components are

> negotiated in a way that best suits both the customer and Lineo. Windows XP Embedded uses a one-size-fits-all technique with a commensurate royalty structure, restricting the ability of the development team to limit costs.
>
> Support for Linux is also competitive. The open-source nature of Linux means that effective support can be provided from a wide range of resources including internal, contractor, public access (internet-based resources) and commercial Linux corporations. What do these things mean to the developer? They mean that Linux vendors will be vying for your business, giving developers more choice to match their unique time-to-market, cost and feature requirements.

Why Attack Embedded Linux?

As the dust begins to settle, it's interesting to consider this question: Why has Microsoft's Embedded group aimed their big guns at embedded Linux at this time? Here are some clues:

Clue number one: Microsoft is losing to Linux in the general embedded market. A number of market studies, such Evans Data Corporation's "2001 Embedded Systems Developer Survey", have consistently begun to report that tremendous strides have been made by embedded Linux over the past one to two years.

Specifically, Evans Data Corporation's latest data says that embedded Linux was the third-most popular OS choice for new embedded system designs among 500 developers polled in 2001--behind Wind River's VxWorks and Microsoft's DOS, and ahead of Microsoft's WinCE (see Figure 1).

Of greater significance, though, is that the results of the study suggest that embedded Linux is poised to jump into first place, ahead of both Wind River's and Microsoft's offerings, within the next 12 months.

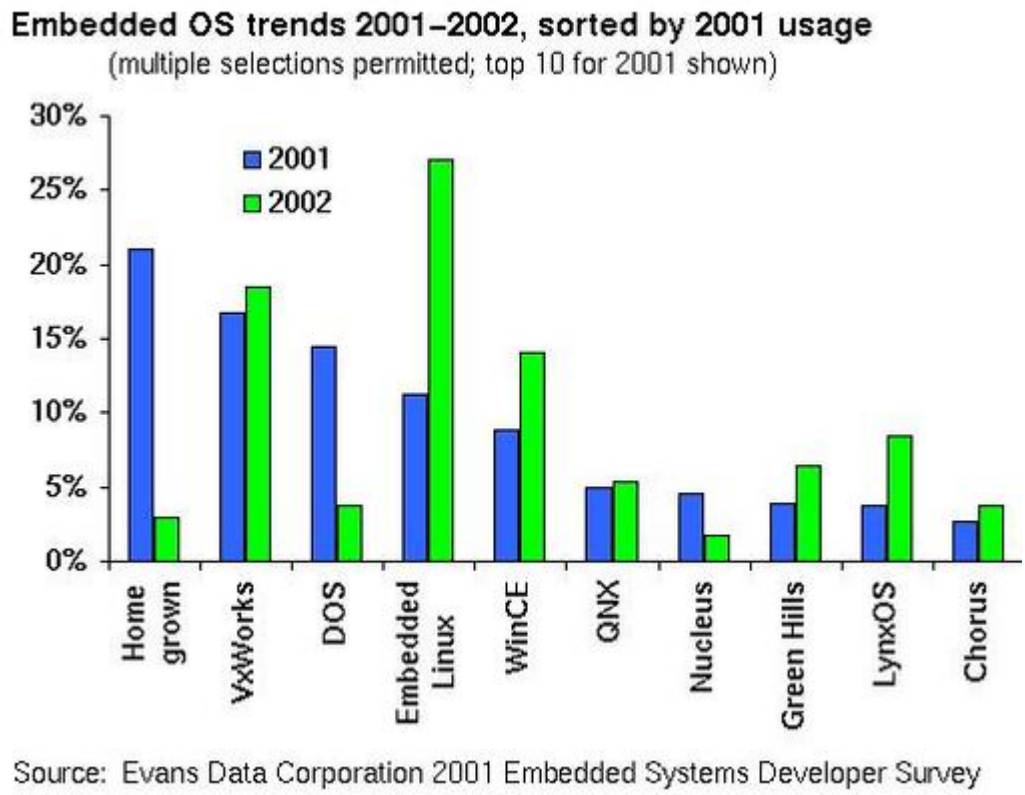


Figure 1. Evans Data Corporation 2001 Embedded Systems Developer Survey

Clue number two: the stakes are extremely high in emerging “post-PC devices” markets. Another likely reason for Microsoft's growing concern with embedded Linux is that major manufacturers like Hewlett-Packard, Sharp and Motorola recently have begun delivering new consumer devices that contain embedded Linux. These include handheld computers and TV set-top boxes—emerging markets with extremely high-volume potential, which Microsoft undoubtedly wants to dominate.

In the PDA space, where Microsoft has steadily gained ground on market-leader Palm, embedded Linux may well be perceived as a “dark horse” disruptive technology. This is especially of concern given the growing popularity of embedded Linux in the Far East, where most high-volume consumer products are manufactured.

In contrast to the handhelds market, there is no established leader in the emerging markets for set-top entertainment systems and auto-PCs. These markets clearly have the potential to absorb more OS royalty stickers than desktop PCs, so it is not surprising that Microsoft would want to nip the early embedded Linux lead in the bud.

More to Come

Watch for the action to heat up further in these and similar high-volume “post-PC” markets in the coming months. According to rumors from embedded Linux

vendors such as Lineo, MontaVista and Red Hat, there are literally hundreds of embedded Linux-based consumer devices in the pipeline—products that can't be discussed publicly until they're about to be shipped by their manufacturers.

All in all, 2002 promises to be another exciting year for embedded Linux!

For Further Reading

Rick Lehrbaum (rick@linuxdevices.com) created the LinuxDevices.com and DesktopLinux.com web sites. Rick has worked in the field of embedded systems since 1979. He cofounded Ampro Computers, founded the PC/104 Consortium and was instrumental in creating and launching the Embedded Linux Consortium.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Unbiased License FUD

Lawrence Rosen

Issue #95, March 2002

This month Lawrence explains “GPL infection”.

A few months ago I wrote about the dangers of the Microsoft shared-source license (*Linux Journal*, December 2001, /article/5496), calling the shared-source license a Trojan horse. By merely looking at Microsoft's code, you could potentially “infect” your own software, leaving yourself vulnerable to a copyright infringement lawsuit by Microsoft. A reader responded that I was applying a double standard—that the exact same problem exists with the GPL. “For companies developing traditional proprietary software”, he wrote, “the act of merely looking at GPL code can put them in exactly the same position that they would be in if they looked at Microsoft shared-source licensed code.” Even worse, he claimed, the “infection” clauses of the GPL would require that the entire proprietary work that uses GPL code be licensed under the GPL, or the GPL portion removed. His analysis of the GPL is only partly correct. He is confusing three different scenarios.

1. Suppose a proprietary software company has licensed code under the GPL and then includes the GPL code in its derivative work software. The company has agreed to the GPL license and must honor its terms. A court might impose a “specific performance” remedy requiring the company to distribute the proprietary derivative work, including publishing the source code, under the GPL—typically is referred to as “GPL infection”, but here it was a risk intentionally accepted by the company.

2. Suppose the proprietary software company does not agree to the GPL license but uses the GPL code anyway in its proprietary derivative work. Under the copyright law, the company may be forced to pay damages and to stop using the code in its derivative work, but the remedy of specific performance (e.g., publication of the source code) probably would not be available. This is not GPL infection; it is simply copyright infringement.

3. Suppose an employee of the proprietary software company, without authorization from or knowledge of his or her company, intentionally or otherwise incorporates GPL code into a proprietary derivative work. (In law, if the act is intentional the employee is said to have engaged in a “frolic and detour”.) In this scenario, the company probably will not be liable for willful infringement, although it must stop using the infringing software. Again, there is no GPL infection, merely an infringement.

The creators of proprietary software should indeed exercise caution. Incorporation of someone else's copyrighted code into a software product (even when unintended) can have undesired consequences, including the potential for expensive copyright infringement lawsuits, large damage awards and injunctions against further distribution or sale of the infringing software.

Every company that produces software must engage in safe development practices. That includes making sure that the development staff understands how important it is not to copy someone else's software before reviewing with an appropriately skilled attorney the terms under which that software is obtained. If a company wants to protect the proprietary nature of its software, it must be careful to avoid infection from other proprietary software as well as from free and open-source software. The burden of implementing proper safeguards, including management time spent training staff and securing the workplace—as well as the attorney time to review licenses—are costs of doing business, which must be factored into the price of the software.

These cautions also apply to the creators of open-source and free software. Simply because software is going to be distributed for free doesn't mean that it can't be stopped cold by an infringement lawsuit.

Here are a few safeguards I recommend to my clients:

Obtain a signed copyright assignment or an explicit license for every third-party contribution to your project with language such as the following: “The undersigned author(s) hereby represents and warrants that the software is original and that he/she is the author of the software.”

If contributed software was written by an employee of another company, the express permission of that company to use the software should be obtained. The Free Software Foundation recommends an Employer Disclaimer of Rights that authorizes the employee to assign the software “for distribution and sharing under its free software policies”.

If employees have been exposed to third-party software that is proprietary and whose source code is not available for copying, it may be appropriate to assign

the employees to other projects rather than risk an infringement (or theft of trade secrets) lawsuit.

As a lawyer, it is my duty to be cautious and to warn of risks. But as an advocate of free and open-source software, I also want to back off from sowing fear, uncertainty and doubt (FUD) more widely than is reasonable.

The goal of open-source development is to encourage the sharing of code and to avoid secrecy. Developers of proprietary software may need to be cautious about being exposed to other companies' source code, but the developers of free and open-source software should copy freely from other free and open-source software—within the constraints of the contributors' licenses. The result will be better software for all.

Legal advice must be provided in the course of an attorney-client relationship specifically with reference to all the facts of a particular situation and the law of your jurisdiction. Even though an attorney wrote this article, the information in this article must not be relied upon as a substitute for obtaining specific legal advice from a licensed attorney.

email: lrosen@rosenlaw.com

Lawrence Rosen is an attorney in private practice in Redwood City, California (www.rosenlaw.com). He is also executive director and general counsel for Open Source Initiative, which manages and promotes the Open Source Definition (www.opensource.org).

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

LINUX JOURNAL

The Book of Zope

Reuven M. Lerner

Issue #95, March 2002

The Book of Zope covers most of what a beginning Zope developer will need to know.

Zope, an open-source application server, has become an increasingly popular choice for web development in the last few years. Zope Corporation, formerly known as Digital Creations, has gone to great lengths to prove their commitment to the Open Source community and has encouraged Python and Zope developers around the world to spread the Zope gospel. The fact that Zope is written mainly in Python has given a boost to the worldwide Python community, providing what may indeed be the “killer app” that brings new people into the Python fold.

While Zope is an extremely powerful system for creating web applications, it also can be daunting to new users. Its paradigms are quite different from other web development systems, in no small part because of its heavy reliance on objects. If you are not comfortable with classes, instances, instance variables, class methods and instance methods, the learning curve for Zope is even steeper than otherwise would be the case.

While the Zope documentation has improved considerably over the last few years, and while many zope.org members have contributed their own documentation, tips, code and tutorials, there is still a need for solid introductory texts for learning Zope.

The Book of Zope aims to fill this niche. It was written by a number of programmers at Beehive, a web development company with offices in Berlin and Washington, DC. The book is an English translation of the original German version and reads better than I expected for a translation.

The Book of Zope covers most of what a beginning Zope developer will need to know. (While some of the chapters may be useful for designers and other

nontechnical people, nonprogrammers probably will have a difficult time understanding many of the items in it.) Initial chapters describe how to navigate Zope's management screens, DTML (the server-side programming language that can be used to implement functionality without having to write Python programs) and permissions with users and roles.

The book then begins to cover more complex ground, describing ZClasses, SQL connectivity and Python scripts. There is even a chapter on Zope products, introducing the notion of a product and how to write one of your own in Python.

The Book of Zope covers everything that you might expect in a book of this type and does so thoroughly. But as I was reading it, I felt that something was missing: a sense of perspective, helping the fledgling Zope programmer to get "Zope Zen", an intuitive sense for how Zope works. The book's text was informative, and its numerous examples were clear, but I wish that there had been more pauses to explain where each technology fits into the scheme of things, rather than simply introducing them.

The Book of Zope is a good complement to on-line Zope documentation and probably will be most useful to programmers who want more direction after experimenting with Zope on their own. Someone who is completely new to Zope might benefit from this book, but they may find themselves confused and frustrated.

Publisher Information/The Good/The Bad

Reuven M. Lerner owns a small consulting firm specializing in web and internet technologies. He lives with his wife Shira and daughter Atara Margalit in Modi'in, Israel. You can reach him at reuven@lerner.co.il or on the ATF home page, www.lerner.co.il/atf.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Letters

Various

Issue #95, March 2002

Readers sound off.

Letters

Webmin Joy

I have been using Webmin for a couple years now to administer my servers. It is a wonderful and powerful tool. I was very pleased to read about it in your December 2001 issue (“Webmin: Good for Guru and Newbie Alike”). Mr. Elmendorf did a great job. I hope to hear more on Webmin in the future in the pages of *LJ*. Like, how to add modules and such. Great work!

—Jody “JoLinux” Harvey

Thanks David Bandel

Thanks so very much for your article “Taming the Wild Netfilter” in *Linux Journal* (September 2001) regarding iptables. I've been banging my head on a wall to come up with a clean solution for protecting my home network from internet kiddies. Your script (with some modification) is a pleasure to work with. All I had to do is add some accepts for a couple of ports on the Linux box and *voilà*! Your tutorial is excellent!

—Don Lafontaine

Empirical Knowledge

This comment is a little frivolous, but still worth an e-mail. At the end of his article “Mainstream Linux”, December 2001 issue of *LJ*, Robin Rowe quotes Linus as saying, “Software is like sex: it's better when it's free.” I read the quote to my wife and her immediate reaction went something like this: “How does he

know? If what he says is true, then he *must have* paid for it sometime in his past!" Whoops! Choose your words carefully, Linus!

—Paul Barry

Confronting the Frontier

While Editor in Chief Richard Vernon has every right to support the Electronic Frontier Foundation and to encourage his readers to do the same, he should be more forthcoming about the positions taken by the Foundation [see "EEF Wants You", December 2001 issue of *LJ*]. The Foundation is about more than protecting open-source programming. It is also about preventing public libraries from filtering web content for children and allowing crackers to freely distribute the means to steal proprietary programming in the name of free speech. Before joining or contributing, I would urge readers to examine the positions of the Foundation and the writings of its cofounder, John Perry Barlow, by visiting its web site. Love the magazine, otherwise.

—Bill Moylan

What exactly does "the means to steal proprietary programming" mean? A tool that lets you view DVDs on Linux? A debugger? If anything that can be used to infringe copyright should be banned, then we have no Linux left. We too urge readers to examine the Foundation's positions by visiting its web site (www.eff.org)--the more they read about the EFF, the better.

—Editor

Love the Beans

After reading your last two articles in *Linux Journal* on using JBoss [see Reuven Lerner's December 2001 and January 2002 At the Forge column], I just have to say, "excellent job!" Your articles are really a pleasure to read, and I've passed them along to other developers here as well as our director, and they agree. We're starting to consider how to get JBoss accepted here as an approved platform for Kaiser Permanente Hospitals (Kaiser has about 110,000 employees here in the United States). I'm greatly looking forward to your treatment of Zope. Keep up the superb work!

—Cole Thompson

Flying LTO

I was pleased to see the review of the HP SureStore Ultrium 230 tape drive in the December 2001 issue. The company that I work for currently is considering

Qualstar TLS tape libraries for use on our data collection platforms. After reading the article, it appears that this might not be such a good idea. I was wondering if anyone has had any experience using robotic tape libraries on moving platforms such as ships or aircraft?

We have successfully deployed HP SureStore Ultrium 230 tape drives on our ships and are extremely happy with their performance. The "Open" part of LTO is what originally drew us to this drive. With multiple vendors for both the drives and the media, the prices should prove to be competitive. The price for the media has already come down significantly from launch. We tested a RAIDZONE RS-15 1TB NAS with a directly attached HP SureStore Ultrium 230 tape drive. The actual throughput we got was 13MB/sec using GNU tar. This works out to about 46GB/hour (pretty close to the advertised rate). The NAS had no problem supplying the data to the drive. In fact, it was coasting most of the time.

—Jan "Evil Twin" Depner

Criticism Not So Harsh

With regards to the query "128-Bit Precision with GCC" in the Best of Tech column of the December 2001 issue, the reply was neither correct nor helpful—GMP is not the equivalent and using GMP means rewriting code. Some compilers, AIX amongst them, will carry out calculations using 64 bits (or 128 bits, depending on the processor) with the appropriate option. (Using this, means that the product of two 32-bit numbers will always fit within an int. No code modification is needed.) GMP, however, is a multi-precision package that defines certain data structures wherein the multi-precision numbers are put. To use it, you must extensively rewrite your code. Some compilers, including gcc, accept the "long long" extension and use 64 bits for calculation but that still requires modifying your code and raises portability questions. My answer would be that, sadly, there is no equivalent without some sort of code modification. Such an option to gcc would be nice, though.

—John

Upon rereading my letter [above], I realized that it reads far harsher than I intended. Both the column and the individual answering the query have done your readers good service in the past. I did not mean to slight either, and I apologise if anyone took amiss. I merely wished to indicate that the solution is by no means simple.

—John

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

UpFront

Various

Issue #95, March 2002

All Your Blank

All Your Blank

Few social powers exceed that of permutation. Once a catchy phrase enters common parlance, endless variants soon permute into use. This happened with “Kilroy was here” in World War II, and it's happening now with “All your base are belong to us”. Here's what, other than base, that are now belong to us (or whomever):

- Bass: www.dearauntnettie.com/museum/museum-bass.htm
- Data (and biz plans): www.theregister.co.uk/content/4/18002.html
- Device: www.flashenabled.com/mobile
- Pulse: www.pulse.nl
- Data model: discuss.2020hindsight.org/manila/datamodel
- Al Qaeda: www.rushmagazine.com
- Java: community.borland.com/article/0,1410,27322,00.html
- Logs: www.irc-junkie.org/content/s-content.php
- Toad: www.hypnotoad.org
- Burners: medlem.tripodnet.nu/connymute/changelog.html
- Vault: www.volny.cz/lord_vader/vault/cz
- Surrealism: www.execpc.com/~bogartte
- Links: www.farces.com/farces
- Heaven: www.ashleypomeroy.com/fillums.html

And that's just in the first two pages of a Google search: 1-20 out of 31,600 results.

—Doc Searls (with thanks for the discovery to Don Marti)

Savannah and Free Software Development

What began as a simple move to reduce the workload of the GNU CVS maintainers has turned into an ambitious project to create a complete development hosting facility. In October 2001, the GNU Project announced a plan to rewrite completely the SourceForge software. This rewrite will address several key technical and practical issues.

SourceForge is an integrated collaborative development environment. It presents a web interface as a portal to CVS, FTP and e-mail services. The original SourceForge server, SourceForge.net, currently hosts more than 30,000 projects and 300,000 users.

The GNU Project has been running a modified version of the SourceForge software at savannah.gnu.org since late in the year 2000. Savannah was set up by GNU volunteers to automate and ease the process of GNU project management. Developers of the GNU Project want a service specifically for free software projects, and one independent from the VA Software Corporation.

Concern has been expressed over the centralized nature of the current SourceForge system. Where do the hosted projects go when and if VA Software loses the capital to support SourceForge.net? Where do those 30,000+ projects go if some SSSCA-like bill becomes a reality?

The development team has come up with an obvious answer: decentralization. Projects will be hosted on various sites across a network. All projects will be browseable from any node of the network.

Each machine running the new Savannah system will host any number of read/write and read-only projects. A read/write project will exist locally on that machine. Read-only projects are mirrors of a project hosted elsewhere. In case one of the host machines goes down, locally hosted developers will be able to move to one of the mirrors of their project and set that to be the read/write server for the project. Project definitions are exchanged between distinct servers via an XML-based format. The Savannah service is fault-tolerant. It allows for machines going out of service without loss of data.

The Savannah developers are basing all of the content of the new system on templates. Sections of pages can be pulled from GNU gettext files, based on the language of the reader. **gettext** is a package for developers, translators and users for creating multilingual applications. This provides for internationalization, a feature sorely lacking in the current SourceForge system.

The developers of the new system have determined to create a system where there is a clearly defined upgrade path between versions of the software. The

software will be packaged using Debian's .deb packages, and upgrades will be automated through use of the package system.

The system is based on the GNU phpGroupWare code base. phpGroupWare implements templates needed for internationalization, authentication, database access, an XML-RPC interface and session management. The Savannah team is working closely with the phpGroupWare team to exchange improvements.

Bradley Kuhn, vice president of the Free Software Foundation, wrote:

> A collaborative site providing a unified interface for project management is key for free software development. To truly help the cause of software freedom, such sites must be implemented completely with free software. Savannah does this for the GNU Project and will soon do the same for all GPL-compatible free software projects.

Savannah will provide important services to free software developers. It will provide the services of SourceForge.net on a world-spanning network of servers that each speak the individual developer's language. Savannah will have fault tolerance and data recovery. Best of all, the only support that the system needs is for volunteers to provide hosting services and support to their ability. Look for more information on Savannah at savannah.gnu.org.

The GNU Project can be found at www.gnu.org. For a definition of free software, see www.gnu.org/philosophy/free-sw.html.

—Nicholas E. Walker

LJ Index—March 2002

1. Year at which the over-65 population of Germany will approach 50%: 2030
2. Multiple of over-65 growth rate by which the under-35 population of Germany will shrink if the birth rate remains constant: 2
3. Millions of immigrants Germany will need to acquire per year to sustain its current workforce: 1
4. Number of times Tove Torvalds has won the Finnish karate championship: 6
5. Cost of a Windows network solution tested at the CRN test center: $4,688
6. Cost of an equivalent Linux-based network solution at the same CRN test center: $317
7. Percentage of cost savings of Linux vs. Windows at the CRN test center: 93
8. Percentage of humanity that lives on less than $2 per day: 50

9. Billions of people who live on less than $1 per day: 1
10. Number of women who die per minute in childbirth: 1
11. Days of paperwork processing it would take to legalize a bakery in Cairo: 500
12. Percentage by which use of “web bug” surveillance (via 1 × 1 pixel surveillance GIFs) has grown over three years ending August 2001: 500
13. Percentage of top 100 web destinations that use web page “spawning” (opening of unwanted windows) of some kind: 30
14. Percentage of the top 100 European domains that employ spawning: 20
15. Percentage of sites on the Internet that use “mouse trapping” to prevent the user from closing a page or using its back button: 5.7
16. Average number of sexual partners among persons aged 16-55 in the US: 14.3
17. Average frequency of sex per year among the same population in the US: 124
18. Position of both the above in relation to all 27 countries surveyed: 1

Sources

1-3: *Economist*

4: Open Source Initiative

5-7: *Computer Reseller News*

8-11: Bill Clinton

12: CNET, sourcing Cyveillance

13-15: Cyveillance

Quotes without Comment

The Open Source movement, and Linux in particular, are massive volunteer nonprofit projects that share the spirit of community media. It's a radical alternative movement creating successful mainstream software. In fact, it's the same movement that produced the software that the internet revolution depends on. Now the movement has produced a cutting-edge technology that suits the CBAA's needs far better than the commercial competition. The technology is Linux. A Linux server is one the CBAA could be proud of.

—From the Community Broadcasting Association of Australia

Big Blue Offers Big Tux Test Drives

One of the most successful commercial Linux initiatives last year was IBM's offering of Linux eServer partitions on its zSeries mainframes, which can mount up to thousands of simultaneous Linux servers. Winnebago Industries and Korean Air are two high-profile customers that already serve a lot of Linux out of zSeries mainframes.

Now IBM is moving downscale toward smaller businesses, noting IDC's estimate that small businesses represent 48% of all Linux installations, and that small and medium-sized businesses will account for over 50% of the worldwide server market in another two years. The company claims that over 200,000 customers around the world already run their business on IBM eServer iSeries, but they want to raise that number with a new offering: the Linux for iSeries Test Drive.

You can test drive Linux on an iSeries system with a choice of Turbolinux or SuSE distributions and 170MB of user space, for up to 14 days. There are fee-based offerings available as well. Visit www.iseries.ibm.com/developer/factory/testdrive for details.

—Doc Searls

Stop the Presses: Linus Torvalds Wins World Technology Award

Linus Torvalds has won the 2001 World Technology Award for Commerce. These awards are given by the World Technology Network "to honour those individual leaders or, at times, co-equal teams from across the globe who most contribute to the advance of emerging technologies of all sorts for the benefit of business and society." It honors

> ...those innovators who have done work recently which has the greatest likely future significance and impact over the long-term...and who will likely become or remain key players in the technological drama unfolding in coming years.

The group adds that the awards "are about those individuals whose work today will, in our opinion, create the greatest ripple effects in the future...in both expected and unexpected ways."

"Linus Torvalds was selected for his work on Linux and the Open Source Software Paradigm", the Awards site says.

> Linus Torvalds wrote the kernel of Linux and established the Open Source software model, which is a revolutionary way of creating software. In doing so,

> he not only designed one of the most important pieces of software ever, but he also created a new paradigm for software engineering.

It continues:

> Linux is one of the most important operating systems, at least as important as UNIX and MSDOS. It is crucial for mobile communication devices, for webservers, for the development of the Internet and for many other areas in computing, networking and information technology.
>
> Linus Torvalds is not only an outstanding software engineer, but also a global community leader (of the open source software community).

The winners are announced at the end of each year, so the 2001 Awards were announced at the beginning of 2002. There were awards in twenty-three categories, with five finalists in each category. Other winners included:

- Lawrence Lessig, Professor of Law at Stanford University and author, for Law.
- Robert Metcalfe, inventor of Ethernet and founder of 3Com, for Communications Technology.
- Gordon Moore, cofounder and Chairman Emeritus of Intel, for Information Technology—Hardware.
- Shawn Fanning, author of Napster, for both Entertainment and Entrepreneurship.

The full list of award recipients is at Nature: www.nature.com/nature/wta.

—Doc Searls

They Said It

The world doesn't fear a new idea. It can pigeon-hole any idea. But it can't pigeon-hole a real new experience. It can only dodge.

—D.H. Lawrence

The world is full of abandoned meanings.

—Don DeLillo

While the flightless bird may have been booted off Wall Street, it is being welcomed on Main Street as a dependable substitute for more expensive software sold by competitors such as Microsoft and Sun Microsystems. From auto dealers in Florida to grocery stores in the Arctic Circle, companies are using Linux to run web sites, power databases, track inventory and balance the books.

—Elise Ackerman, in the *San Jose Mercury News*

If you stop and look at the broader picture, in many cases Linux has gone from a novelty to something that people are starting to deploy certain types of software solutions on. It's the deployment that's quiet, but ultimately more important than the noise.

—Dan Kusnetsky, IDC

It has occurred to me that if people really knew how software got written, I'm not sure if they'd give their money to a bank or get on an airplane ever again.

—Ellenn Ullman

Diogenes was run out of town for counterfeiting coins. Conscience is the small voice that says “someone might catch me”. Integrity can only exist in a vacuum. Cynicism should be taught in kindergarten. Have a profitable day.

—Opus Dark

When they say, “Gee it's an information explosion!”, no, it's not an explosion, it's a disgorgement of the bowels is what it is. Every idiotic thing that anybody could possibly write or say or think can get into the body politic now, where before things would have to have some merit to go through the publishing routine, now, ANYTHING.

—Harlan Ellison

Education is a state-controlled manufactory of echoes.

—Norman Douglas

Redefining the role of the United States from enablers to keep the peace to enablers to keep the peace from peacekeepers is going to be an assignment.

—George W. Bush

War is God's way of teaching Americans geography.

—Ambrose Bierce

Since we cannot know all that there is to be known about anything, we ought to know a little about everything.

—Blaise Pascal

So where are these imaginary earthshaking geek outlaws who laugh in derision at mere government? Well, they do exist, and they're in Redmond. The big time in modern outlaw geekdom is definitely Microsoft. The Justice Department can round up all the Al Qaeda guys they can wiretap, but when they went to round up Redmond, they went home limping and sobbing, and without a job. That is a geek *fait accompli*, it's a true geek lock-in. In 2001, Microsoft has got its semi-legal code in every box that matters. They make those brown-shoe IBM monopolists of the 1950s look like model public citizens.

—Bruce Sterling

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

SPAM, Not Spam, Is the Stuff of Memories

Richard Vernon

Issue #95, March 2002

From the Editor

SPAM, Not Spam

Why the need for filters? It's just pork shoulder.

Yesterday in a company meeting we were discussing the various types of spam we receive and creative ways of dealing with it. Somehow the conversation degenerated to tossing around the idea of a sculpting contest of Linux celebrities out of SPAM lunch meat.

This got me wondering about SPAM's position on the world's use of the term *spam* to describe unsolicited e-mail. Like almost everything else in the computer world, the term comes from a Monty Python sketch in which singing Vikings sing “SPAM, SPAM, SPAM” with increasing volume, preventing any other communication.

A visit to www.spam.com revealed a company with enough confidence (it feeds America's soldiers and is a delicacy in Korea after all) to have a sense of humor about themselves. From their FAQ:

> Q: A lot of people—comedians, especially—poke fun at SPAM. Does it hurt your feelings?
>
> A: SPAM doesn't live in glass houses. It comes in cans.

You even can download SPAM desktop patterns and icons (Mac and Windows only). They say they don't mind people using the term as it doesn't harm their trademark, but prefer that when used to describe electronic junk mail, it be spelled in lowercase, rather than their trademark uppercase spelling.

I'll say it hasn't hurt their trademark. Every time I hear the term, or even see some in my mailbox, my mouth waters. I must confess that despite my respect for pig the animal, I really love pig the meat—and I'm not scared by meat in a can.

Growing up in the '70s with a mother always on some kind of health kick we never had SPAM in the house, and my first experience with it was during a camping trip with my 7th-grade friend Carl Gerlock and his parents. Carl's dad was the kind of old man who came home from work, drank a big glass of vodka and would pretty much leave you alone if you didn't get between him and his television—oh, and he insisted on putting out his campfires with his urine. These people could put away the SPAM, and they did at every meal—and I was glad to help them (despite the nasty reputation, it really is mostly pork shoulder, ham and spices). I suspect there are a lot of other closet SPAM lovers out there since there have been 239,025,706 cans consumed since July 28, 1998.

I haven't eaten SPAM since that trip (my wife is at least as health conscious as my mother), but someday I hope to. In any case, I'll always retain those fond SPAM memories, and they'll always be invoked by discussions of spam. Incidently David Bandel discusses a nice cure for spam in this month's Focus on Software—if that's your thing.

Resources

Richard Vernon is editor in chief of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Best of Technical Support

Various

Issue #95, March 2002

Our experts answer your technical questions.

Best of Technical Support

Modem Works with Old Kernel, Not New One

After an upgrade from Red Hat 7.1 to 7.2, my modem no longer works when I boot into the SMP kernel. It works fine when booting the non-SMP kernel. I have downloaded and compiled the latest stable kernel, and the problem still exists. Under 7.1 the same modem worked fine with the SMP kernel. I have a US Robotics 56K FaxModem (model 3CP5610A) with a Tyan motherboard and dual-Intel Pentium 133.

—Nathan Myers, myersn@voyager.net

Check to see if the modem is detected by the kernel. You can see this with a **grep ttyS /var/log/messages***. You will see a list of serial devices; note that some will be built-in on your motherboard.

—Christopher Wingert, cwingert@qualcomm.com

Thank you for including the model number of your modem; it is most useful. A quick search on Google shows that you have a PCI modem, which works slightly differently from old-style ISA modems. The good news is that it is a real modem and not a Winmodem. The best link I found for you is this one: www.idir.net/~gromitkc/3cp5610.txt. USR also seems to provide an example script for Red Hat here: www.usr.com/support/drivers-template.asp?prod=s-modem.

—Marc Merlin, marc_bts@valinux.com

NFS Living in the Past

I have one machine (Red Hat 7.2) that serves the user home directories, which is now an ext3 fs, to several other peripheral machines (all Red Hat 6.2). I've noticed that files updated on the peripheral machines don't get updated on the server, and the changes don't get reflected on the other peripheral machines.

The server exports options (rw, no_root_squash). The client invokes with defaults, nodev, rw. This seems to indicate that the clients are caching, but it never seems to flush (I have a file that was changed a day ago that is still unchanged on the server). I've searched the Web and have found nothing that would help.

—R. K. Owen, rk@owen.sj.ca.us

Nothing in NFS should cache a file for a day. I would check the obvious and make extra sure that the clients are indeed writing in an NFS-mounted directory and accessing the NFS server. You also can check that when you modify a file on the server, the clients are seeing the new copy.

—Marc Merlin, marc_bts@valinux.com

Can StarOffice Import EPS?

My system consists of Red Hat 7.1 with a 933MHz Pentium III. The problem, which occurs with both StarOffice 5.2 and 6.0 beta, is that encapsulated PostScript graphics files (.ps or .eps files) are read as text, not as graphics. This happens with any selection (e.g., Text Document, Presentation, Drawing or Chart). **gv** shows the correct graphics. So the question is: does StarOffice have the capability to read graphics files in PostScript format and display the graphics rather than the PostScript text? If so, how do you do it?

—John C. Burgess, burgess@wiliki.eng.hawaii.edu

The menu item Insert-->Graphics-->From File does understand .eps files.

—Scott Maxwell, maxwell@ScottMaxwell.org

PCI Modem Not Recognized

I recently purchased a US Robotics 56K PCI modem card. It is not a Winmodem, which is why I purchased it. I planned on using it on my dual booting (Windows 98/Linux) system. Windows 98 listed it as device COM5, not the usual COM2.

I went into the Linux side and created a /dev/ttyS4, using setserial to set the port and irq. I made a symbolic link to /dev/modem. When I run minicom, it does not complain that the device isn't there, it just does not seem to do anything. How do I get Linux to recognize my modem? I tried an **echo ATH1>/dev/ttyS4**, and there is no dial tone in the modem speaker, so I am pretty sure the command is not making it to the modem.

—Tony Preston, apreston@k2nesoft.com

This sounds like the IRQ is not set correctly. You should check the IRQ with **lspci -vv**. Look for your modem in the list and use setserial to set the IRQ.

—Christopher Wingert, cwingert@qualcomm.com

See www.idir.net/~gromitkc/3cp5610.txt for an example of how to set the IRQ with setserial.

—Marc Merlin, marc_bts@valinux.com

SCSI Error Message

I recently upgraded from a SCSI TR4 Travan tape drive to a 24GB DAT drive on a Linux server, and every morning I have the following message on the console after an overnight backup:

```
st0: Error with sense data:
[valid=0] Info fld=0x0, Current st09:00:
sense key U
nit Attention
Additional sense indicates Not ready to ready
transition (medium may have changed)
```

Can anyone say why this is so?

—Clark, CLARKKclr@cs.com

Check to make sure your SCSI bus is correctly terminated.

—Christopher Wingert, cwingert@qualcomm.com

You do not say whether the backup actually occurs before you get this message, or whether you are able to write anything on tape. If you haven't yet been able to write anything on tape, make sure that you do not have SCSI connection or termination issues. If your backup does occur, you may have some mt command after the backup that does something that isn't supported by the tape drive, or the tape drive may require some attention, like a cleaning.

—Marc Merlin, marc_bts@valinux.com

Where's /dev/fd0?

I try to mount my floppy disk (1.44MB) on Mandrake 8.1 with the command **mount /dev/fd0 /mnt/floppy**, and it says “unknown device”. It is connected correctly and it works perfectly, but Mandrake doesn't recognize it.

—Luis, godoman1@hotmail.com

Check to make sure that /dev/fd0 exists. Also, check /var/log/messages after running the mount to see if there is any clue to the problem.

—Christopher Wingert, cwingert@qualcomm.com

Make sure the kernel sees your floppy. You should have something like this in /var/log/dmesg:

```
Floppy drive(s): fd0 is 1.44M
FDC 0 is a National Semiconductor PC87306
```

Then see what dmesg says after you try your mount command.

—Marc Merlin, marc_bts@valinux.com

When you say that “it is connected correctly and it works perfectly”, I assume you dual boot your system and the floppy drive works in the other OS environment. Here are a few things you can check: make sure that the /dev/fd0 file exists, that it is indeed a device-special file and that it has the correct permissions on it. The floppy you are trying to mount has a filesystem on it, and the filesystem support is present in the kernel.

—Usman Ansari, uansari@yahoo.com

I Have No /dev/printer and I Must Print

I am using Red Hat 7.1 and can't seem to find /dev/printer. I need this socket for a Perl script. I can change the name in the script, but can someone tell me what to change it to? I do not know the name of the socket that the lpd dæmon uses in Red Hat 7.1.

Scott Statland, scott@nycgiftbaskets.com

Not to be obstructionist, but are you sure you need direct access to the device file? If you're just trying to print, you can do that in Perl with

```
open(FH, '| lpr')  ||  die $!;
```

and print your desired text to the FH filehandle. If your script is intended for broad distribution, bear in mind that UNIX printing is very flexible; a given print queue may be serviced by a printer attached to some other machine on the network, for example, so the local machine may not have any file under /dev that represents that printer. You might want to find a Perl module that helps you parse /etc/printcap and look for a print queue's lp resource to learn what device file, if any, is associated with that printer. (Type **man printcap** for a thorough description of that file.)

—Scott Maxwell, maxwell@ScottMaxwell.org

The device nodes /dev/lp[0-2] give you direct access to your parallel printer (bypassing lpd altogether).

—Marc Merlin, marc_bts@valinux.com

If you want to print using lpd, there's a Perl Net::Printer module downloadable from CPAN that lets you print to lpd and check the status of your print jobs from a Perl script. Read the man page on-line here: search.cpan.org/doc/CFUHRMAN/Net-Printer-0.20/Printer.pm.

—Don Marti, info@linuxjournal.com

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

New Products

Heather Mead

Issue #95, March 2002

BRU-Pro 2.0 and BRU 17.0, Volution Manager 1.1, PowerUpdate 2.0 and more.

New Products

BRU-Pro 2.0 and BRU 17.0

New updates of BRU-Pro and BRU Workstation are now available from The TOLIS Group, Inc. The BRU-Pro 2.0 Linux server includes network traffic data encryption for secure communication between clients and the server, as well as network traffic data compression for better bandwidth usage. BRU Workstation 17.0 supports small to medium-sized commercial network systems; BRU Desktop 17.0 supports SOHO systems with locally attached archive devices; and BRU Personal Edition 17.0 provides data protection for noncommercial use. BRU-Pro 2.0 and BRU 17.0 have new GUI features and support 64-bit filesystems.

Contact: The TOLIS Group, Inc., 10225 East Via Linda, Suite 300, Scottsdale, Arizona 85258, 480-346-2008, sales@tolisgroup.com, www.tolisgroup.com.

Volution Manager 1.1

Caldera, Inc. has released Volution Manager 1.1, a web-based systems management solution for securing, remotely managing and updating multiple systems through a browser. New features in version 1.1 include extended platform support that unifies management of multiple platforms into one interface, simplified installation options and revised status and diagnostic features. In addition, Volution Manager 1.1 supports the latest versions of all major Linux distributions and Caldera UNIX products.

Contact: Caldera, Inc., 240 West Center Street, Orem, Utah 84057, 1-888-GO-LINUX (toll-free), www.caldera.com.

PowerUpdate 2.0

PowerUpdate 2.0 is a multiplatform, Java-based software updating and delivery solution. Comprised of a web browser, database, reporting modules and custom management logic, PowerUpdate is designed to update any kind of software onto any client or server platform. Developers control the updating process, including what will be updated and when and how that will occur. Version 2.0 adds file synchronization, MSI support, support for Mac OS X and the ability to extract and execute archive files. PowerUpdate 2.0 runs on Linux, Solaris, HP-UX and AIX.

Contact: Zero G Software, 514 Bryant Street, San Francisco, California 94107, info@ZeroG.com, www.ZeroG.com.

X4 NAS

NetEngine, Inc. released X4 NAS, a workgroup network attached storage (NAS) for small to mid-sized businesses, workgroups, branch offices and service providers. X4 NAS supports simultaneous users and can be used in applications such as file sharing and on-line and off-line backup storage. Storage space of 160GB to 480GB is available in a 1U system that is set as a rackmount or a standard rack. X4 NAS supports dual 10/100TX Ethernet with a failover feature and a mirrored OS for OS failover. Other features include built-in RAID, automatic data checks and automatic rebuilds.

Contact: NetEngine, Inc., 4116 Clipper Court, Fremont, California 94538, 510-668-2112, solutions@netengine1.com, www.netengine1.com.

cPCIS-2103 Chassis

The cPCIS-2103 Chassis, the newest addition to ADLINK Technology's 3U CompactPCI system line, is a 19" rackmountable or table-top enclosure compact chassis design that is fully PICMG 2.0-compliant. The chassis can be configured with six slots on both the primary and the secondary side for 32-bit user-defined peripheral cards. The chassis has space for the system CPU and three redundant, hot-swappable power supplies, as well as a PCI-to-PCI bridge for expansion. The cPCIS-2103 is available on its own or as part of an integrated system.

Contact: ADLINK Technology, 15279 Alton Parkway, Suite 400, Irvine, California 92618, 866-423-5465 (toll-free), www.adlinktechnology.com.

GFS 5.0

Version 5.0 of Sistina's Global File System (GFS) is now available and allows multiple servers on a SAN to have concurrent read/write access to a shared data pool. New for GFS 5.0 are advanced installation and cluster configuration tools, dynamic multipath support in the pool volume manager to tolerate single path failures, a shared root filesystem, additional lock managers and improved support for third-party snapshot capabilities.

Contact: Sistina Software, 1313 Fifth Street Southeast, Suite 111, Minneapolis, Minnesota 55414, 612-638-0500, www.sistina.com.

Optimizeit Suite

The Optimizeit Suite is a toolkit that allows developers to pinpoint performance and reliability issues throughout the development process of any Java program. Providing scalability and accurate tuning data for all size J2EE applications, the Optimizeit Suite includes full application-server integration and remote process connectivity. The three parts of the suite are the Profiler, which allows developers to find buggy code or faulty algorithms and correct memory leaks; the Thread Debugger, which allows a real-time display of the status of threads and monitors; and Code Coverage, which displays in real time how frequently methods and code lines execute.

Contact: VMGEAR, 1479 Saratoga Avenue, Suite 200, San Jose, California 95129, 888-655-0055 (toll-free), www.vmgear.com.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.