

Advanced search

Linux Journal Issue #131/March 2005



Features

Legacy Database Replacement with LAMP by *Richard Hulse*

Chalk up another victory for Do-It-Yourself IT, as one in-house project replaced three incompatible proprietary applications.

Managing Projects with WebCollab by *Mike Cohen*

Keep your project status info and the key files in one place with this easy-to-use tool.

A Database-Driven Web Application in 18 Lines of Code by *Paul Barry*

Want to see all the code for a soccer team tracking application?
Want to see it again?

Indepth

Introducing Ardour by *Dave Phillips*

A *Linux Journal* first, this article features a new song recorded just for this issue. Get started with hard-disk recording and have a listen.

Centralized Authorization Using a Directory Service, Part II by *Alf Wachsmann*

Single sign-on is one step closer as we replace /etc/passwd entries with a centralized directory of users and groups.

Event-Driven Programming with Twisted and Python by *Ken Kinder*

Develop scalable software quickly with this project that gets a handle on a high-performance programming technique.

GNU Motion: Your Eye in the Sky for Computer Room Surveillance by *Phil Hollenback*

Make your security Webcam show you all the crimes, none of the empty rooms.

The Perl Debugger by Daniel Allen

`print("hello? Is this thing on?\n ");`—or is there a better way?

The Oddmuse Wiki Engine by Brian Tanaka

Get your company or project information organized with a system that lets everyone contribute fixes.

LaTeX Equations and Graphics in PHP by Titus Barik

Put the math you want on your Web site, right inside the pages you're already building.

Embedded

Optimization in GCC by M. Tim Jones

Want to shrink your program's memory requirements, run time or both?

Toolbox

At the Forge Bloglines Web Services, Continued by Reuven M. Lerner

Kernel Korner Analysis of the HTB Queuing Discipline by Yaron Benita

Paranoid Penguin Book Review: Islands in the Clickstream by Mick Bauer

Columns

Linux for Suits Migration Stories by Doc Searls

EOF Data Center Linux at OSDL by Ibrahim Haddad

Departments

From the Editor

Letters

upFRONT

New Products

Archive Index

Advanced search

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

[Advanced search](#)

Legacy Database Replacement with LAMP

Richard Hulse

Issue #131, March 2005

Some legacy database applications are prime candidates for replacement with LAMP-based Web applications. Richard Hulse explains how Radio New Zealand completed such a project.

Radio New Zealand is a public radio broadcaster, and as with other broadcasters, we have a huge library of music and programmes about music. In 1987, a new computerized library cataloging system, called BRS, was commissioned to assist broadcasters in the storage of library data.

Specifically, BRS was used to store data about LPs (and later CDs), tapes, live concert recordings, interviews and the collection of classical music scores. The system was accessed by way of dumb terminals and, later, terminal emulators on PCs. It also was used by radio staff to schedule and track the music broadcast on Concert FM—the company's classical music network.

BRS was a proprietary cataloging application sold by Maxwell Online, Inc. It ran on UNIX and had a long life. It had a couple of hardware upgrades during its 16-year life; on the software side, a few extra database tables were added for other types of data. BRS survived Y2K without a glitch, in spite of claims to the contrary, and in 2003 talks began in earnest to replace it.

DIY-IT

In the past, a replacement project of this nature probably would have been outsourced. Experience has shown, however, that in some cases we'd end up with a closed-source custom application and be locked in to one company for ongoing upgrades and modifications. Sometimes when these companies cease trading and people move on, the application we depend on becomes an orphan, and the data is difficult or impossible to move to a new application.

DIY projects are not always appropriate, and we carefully weighed all the issues. Because of the critical nature of both the data and the application, plus the availability of in-house skills, we felt it was appropriate in this case to undertake the project ourselves.

Hello BRAD

Bruce Intemann from our IT department was the project leader and put together a quick proof of concept on a desktop PC running Red Hat Linux 8, an Apache Web server, MySQL and PHP (LAMP). Bruce was able to work out how to extract the data from BRS in plain-text format, and he constructed a simple search interface based on the Full-text Index of MySQL, with a small sample of the data converted by hand. Access was granted by way of a standard Web browser.

Around this time, I was completing a PHP Web project for another part of the company and offered my skills to this new project. When it came to name the system, I thought it would be nice to retain the B and R since they are the first initials of the system's "parents". My wife came up with the name BRAD, and one of our staff decided the acronym stood for Bruce and Richard's Audio Database. The name stuck.

After the proof of concept was accepted, I wrote a short Perl script to parse all the data—about 200,000 records—and insert it into the MySQL database. This was complicated because several of the smaller databases had been merged into the main database, Works, to aid global searching. Fortunately, one field was used to indicate the location (source) of the original data. See Listing 1 for a sample of BRS data.

Listing 1. One Record from the BRS in Original Text Format

```
*** BRS DOCUMENT BOUNDARY ***
..Document-Number:
    000080019
..TI:
    Ode for the centenary of Trinity College Dublin,
    Great parent, hail to thee (Z327)
..MA:
    Hyperion
..CA:
    CDA 66476
..ME:
    cd
..RA:
    The King's Consort
..CF:
    vocal - ode
..CP:
    PURCELL
..CD:
    Robert King
..SO:
    Gillian Fisher, Evelyn Tubb (sopranos), James Bowman, Nigel
    Short (counter-tenors), Rogers Covey-Crump (high tenor), John Mark
    Ainsley (tenor), Michael George, Charles Pott (basses)
```

```
..ST:
    T12-21
..AT:
    Purcell - Complete odes and welcome songs vol 5
..DU:
    002419
..PT:
..RD:
    2-4 Jan 1991
..RE:
..AD:
..SR:
..RO:
..CY:
    1694
..LI:
    Nahum Tate
..OR:
..LN:
..PU:
..RI:
..ED:
..LP:
..LQ:
    cp
..LQ:
..NO:
..IS:
..LD:
    921012
..LU:
    leander
*** BRS DOCUMENT BOUNDARY ***
```

Once a complete snapshot of the data was transferred, I rewrote Bruce's code using object-oriented PHP. I also utilised a search class I wrote for another project, modifying it to display music data instead of news stories.

The rough-and-ready demo was deployed on a development server, and staff members were asked for comments. Based on their responses, we decided the best way to proceed was to improve the system continually based on staff feedback, alongside the operation of the existing system. Dual operation during development ensured that staff still had access to a working system and also allowed comparisons between the search results obtained from both systems. It also allowed staff to gain confidence in using the new system and the results it presented.

To separate out the data into its original sets, a more complex script was written to parse the data files, un-merging all the original data sources from BRS. These sets were inserted into the separate databases and tables shown in Figure 1. Each division of the company is considered a zone in BRAD, and each data source is known as a section. Any zone can contain aliases to sections in other zones or options to search across any list of tables, regardless of where they are in the system.

Zones (databases)

Sections (tables)

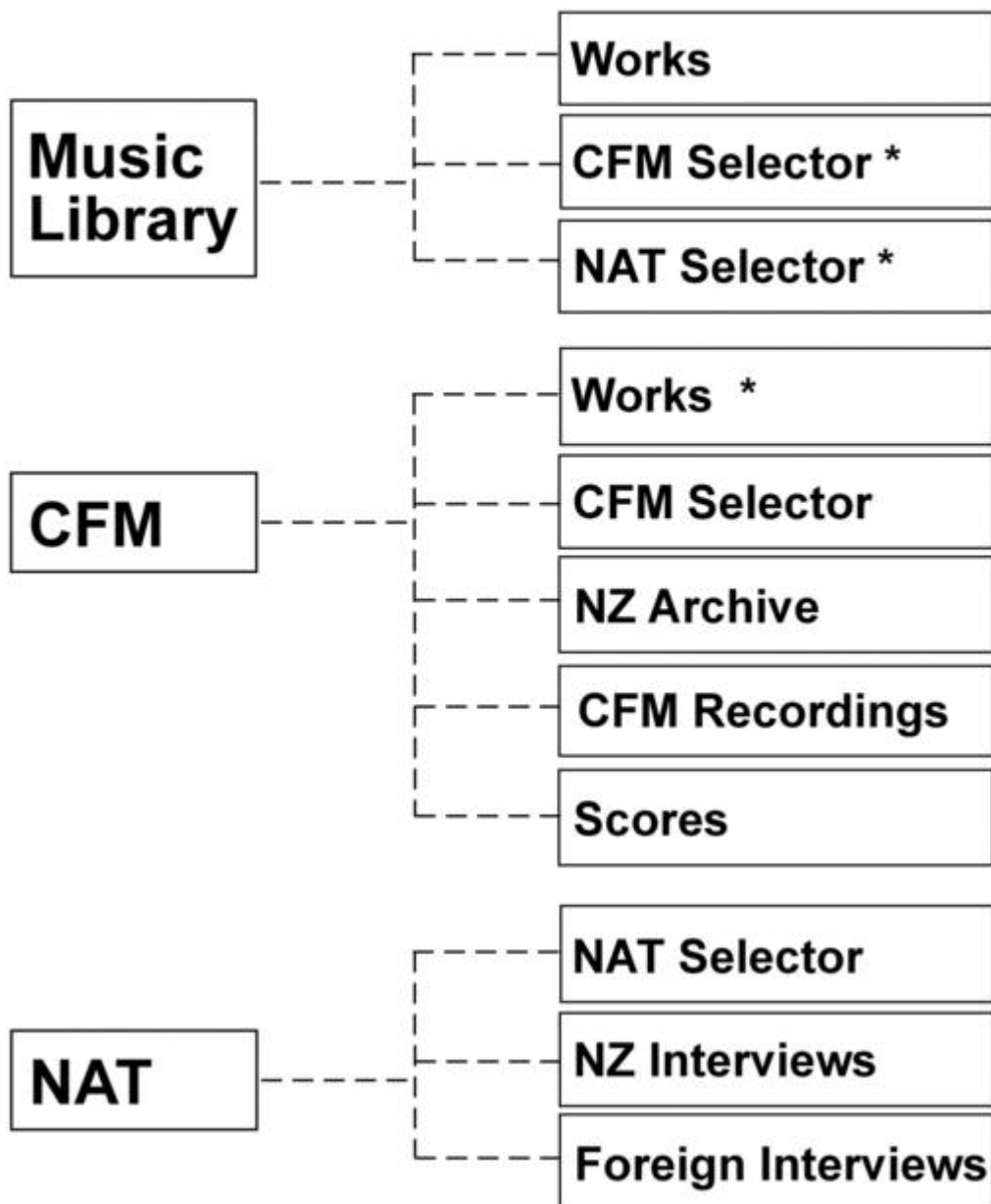


Figure 1. The arrangement of BRAD's data sources. The table marked * represents an alias to another table outside the current zone.

The BRS database was flat (nonrelational), and data had been entered by different people in different formats over many years. As I viewed the results of each snapshot going into the new system, I adjusted the Perl script to clean up some of the data anomalies—particularly in date fields. For example, the original text date field for the last update to a record was edited manually in the past—in BRAD this is a datetime field maintained by the system. Fields also were added to track the creation date.

BRAD Meets Open Source

BRAD was built on a server running LAMP, and it seemed obvious that we should use open-source PHP classes in its development. PHP Extension and Application Repository (PEAR) modules were used for database access, form generation and processing and basic error handling. An existing error class was modified to warn of an error but hide the full message from the user.

Whenever I needed a particular function, I went looking for an open-source module before writing my own. Doing so dramatically sped up the development cycle (see Table 1 for a list of modules used in BRAD).

Table 1. Open-Source Modules Used in BRAD

PEAR::DBM	Database access.
PEAR::HTML:QUICKFORM	Forms on the editing interface.
PATUSER	User management and control of editing access.
Error Reporter Class	Heavily modified to allow swapping of error messages with the main content of page.
Paginator	Pagination of results. Modified to allow parsing of URL into the class.

Meeting Expectations

Because the BRS system had been around for so long, staff had refined their use of the system to a high degree. BRS did have a powerful and fast search facility. It was able to search for particular words in all or any fields specified by the user. Some quirks had to be overcome, however, such as stop words, words not indexed. These included complete names of some musical groups, The Who being one example. In this particular case, to find items by The Who you have to know something else about the group, such as one of the members (Pete Townsend) or something they wrote (*Tommy*). Neither approach always was reliable.

The sometimes unexpected behavior and the difficulty of using a command-line interface meant that most staff used the music librarian to find items, simply presenting a handwritten list of requests. Among the expectations for the new system were an equal or better search capability and a simplified interface that could be used by anyone with minimal training.

One of the most powerful features of BRS was that you could limit search terms to certain fields, for example:

```
Mozart.cp. piano
```

would return anything with Mozart in the composer (cp) field and piano in any field. We decided to retain this syntax in BRAD so that power users still could do the kinds of searches they were used to doing. We had planned to add an advanced search page for BRAD; however, this syntax has turned out to be so flexible that we haven't needed it.

Overcoming Limitations

We faced several challenges with the project. Most of them had to do with modifying MySQL's default behaviors to suit our requirements. The first challenge was to remove all stop words—the list of words not indexed by MySQL due to their presumed commonality in the data. In our situation, every word is considered important.

In MySQL, removing stop words is achieved simply by adding the following line to the MySQL configuration file before adding anything to the database:

```
ft_stopword_file = ""
```

The second challenge was to allow searches for words smaller than the four-character limit typically used by MySQL. The BRS system indexed every word regardless of size, apart from those listed as stop words, and removing all stop words would make any search results more in-line with the terms entered.

This problem was solved by doing two things. First, we reduced the index word size to three characters by adding the following to the config file:

```
set-variable = ft_min_word_len=3
```

Because of the amount of data, these settings were considered to be acceptable performance trade-offs.

The second thing we did was implement a smart query engine that adapted the query, depending on the shortest word in the search terms, before sending it MySQL. This allows full-text searching regardless of the length of any search term.

The last challenge was to make all searches AND by default. MySQL's boolean full-text mode is an OR search when no modifiers are used. You normally would

add a + before each term to make it an AND search. The query engine was built to add the + automatically when no other modifier is present.

The Query Compiler

At the core of BRAD is a term parser and a query compiler. The term parser takes a query, breaks it down and places the components into an array. The array contains a MySQL modifier, +, -, <, >, ~; an atom, a part of the query string—either a word or a phrase; and an optional field name.

The term parser automatically adds a + to each atom when no modifier is present, making all searches AND by default. This is a good thing because users expect that this is how a search engine will work—the more terms you add, the more refined the search.

The optional field is used to support advanced searches when particular words are required in a specific field. In BRAD we retained the . field search operator.

When a normal search is undertaken, the query compiler interrogates each table within the scope of the search and returns a list of full-text fields for each. These are used to compile a query that spans all the full-text fields.

The query compiler can manage a mix of full-text general terms and non-full-text, field-specific terms. The query compiler allows BRAD's data sources to be extended almost without limit and the generated queries to adapt dynamically—removing the need for static query boilerplates for each object class that represents real objects in the database.

A standard Web page form allows the user to control all aspects of the search (Figure 2). Users can select as few or as many zones or tables as they want. These can be customised to meet company and user requirements.

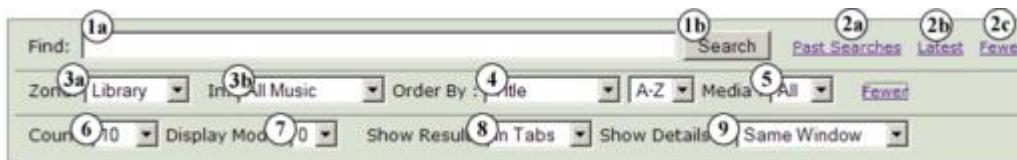


Figure 2. An Explanation of the BRAD Search Interface (See Table 2)

Table 2. Fields in the Search Interface

1a: Search Area	This is where the search terms are entered.
-----------------	---

1b: Search Button	Press to search.
2a: Past Searches	Opens an area that shows searches made in the past. Items in this list can be clicked to do the search again.
2b: Latest	Shows the latest entries in Works, CFMS and NATS data sources. Limited to 250 from each, and sorted with the latest at the top.
2c: Fewer	This link (and the one on the next row) toggles between More and Fewer and reduces or increases the number of BRAD search options that show.
3a: Zone	BRAD divides its data into zones that relate to different parts of the company. Each zone has a number of different data sources. The zone selector allows you to choose the area of the company in which you'd like to search.
3b: In	This selector allows you to determine which data in the zone will be searched. You typically can search across all data in a zone or only one type of data.
4: Order by	
5: Media	You can ask BRAD to search only for records that are stored on certain types of media.
6: Count	The number of items displayed per page.
7: Display Mode	Display modes relate back to the old BRS system and allow the user to choose different summaries for the search results. Display modes can be customised, so if users need a special format it can be added.
8: Show Results	In tabs or as a list. BRAD's normal mode displays the results of each search under a tab-style interface. In list mode, it prints out a list under individual headings. List mode can be used to make lists for printing or pasting into e-mails.
9: Show Details in	This selector allows users to choose between viewing full-record data, by clicking on the link for an item, in a new window or the current browser window.

Extending BRAD

BRAD was written to be extensible in all respects. The data searching can be extended to any kind of data, and specific kinds of searches can be applied to that data.

One of the problems that existed in our company was the use of different database applications for different tasks, with different data being spread across several applications. Our two radio networks use an application called Selector to schedule music items for air. One database, with around 10,000 music tracks, is used for National Radio, while 100,000 music tracks spread across five databases are used for Concert FM. Five databases are used for Concert FM due to limitations in the size of data that Selector can handle.

If a staff member wanted to search for a piece of music, he or she would have to go to each of three applications—BRS, National Radio Selector and Concert FM Selector. There was no way to search all of these at one time.

Fortunately, Selector has a utility to export data in XML format. Although there is no documentation for this and none could be obtained, Bruce was able to determine how to run the export utility and FTP the data to the BRAD server from a Windows workstation. This is done each morning, and a Perl script is run on BRAD to import all the data. The five Concert FM databases are merged into one table, as all the data is unique.

The original search module was extended to search more than one table and return the results, regardless of the number or type of fields. Results are displayed in a tab-style manner (Figure 3).

Find: mozart piano concerto

Search

[Past Searches](#)[Latest](#)[More](#)Searched in: Works, CFM Selector, NR Selector
701 results were found. Query took 1.3350 sec.

Works (497)

CFM Selector (204)

Displaying items 1 to 10 of 497 items found.

0 1 2 3 4 5 6

1 2nd movement from Piano concerto No 21 in C (K467)[Add to Cart](#)

Decca 430 498 cd d

Title	2nd movement from Piano concerto No 21 in C (K467)
Recording Artist	Vladimir Ashkenazy (piano), Philharmonia Orchestra
Cat Form	orchestral - concerto movement
Composer	MOZART
Conductor	Vladimir Ashkenazy
Side & Track	T02
Album Title	The world of Mozart
Duration	7:55
Record Date	1977
Composed Year	1785
First Added	9 Apr 2004
Last Update	9 Apr 2004
Last Updated By	0000-00-00 00:00:00

2 2nd movement from Piano Concerto No 21 in C major, K467[Add to Cart](#)

Decca 425 852 cd cp, d

Tracks:	2nd movement from Piano Concerto No 21 in C major, K467
Recording Artist	English Chamber Orchestra; Radu LUPU, piano
Cat Form	orchestral - concerto movement
Composer	MOZART
Conductor	Uri SEGAL
Side & Track	trk 3
Album Title	Your Hundred Best Tunes, volume 6
Duration	6:24
First Added	1 Jun 1993
Last Update	1 Jun 1993
Last Updated By	0000-00-00 00:00:00

9 Adagio from Violin concerto No 3 in G (K216)[Add to Cart](#)

Biddulph LAB 031 cd cp

Title	Adagio from Violin concerto No 3 in G (K216)
Recording Artist	Yehudi Menuhin (violin), Louis Persinger (piano)
Cat Form	chamber - miscellaneous
Composer	MOZART
Side & Track	T07
Album Title	The young Yehudi Menuhin - The early Victor recordings
Duration	4:23
Record Date	12 Feb 1929
Composed Year	1775
First Added	7 Nov 1991
Last Update	7 Nov 1991
Last Updated By	0000-00-00 00:00:00

10 Allegro assai (Piano Concerto in A (K488))[Add to Cart](#)

PHILIPS 422 269 cd cp

Title	Allegro assai (Piano Concerto in A (K488))
Recording Artist	Alfred Brendel (piano), Academy of St Martin-in-the-Fields
Cat Form	orchestral - miscellaneous
Composer	MOZART
Conductor	Sir Neville Marriner
Side & Track	T05
Duration	8:00
Record Date	June 1973
First Added	9 Aug 1993
Last Update	9 Aug 1993
Last Updated By	0000-00-00 00:00:00

1 2 3 4 Next

Figure 3. A Truncated Page of Results

You can see the first results from the Works table. The other inactive tab shows the number of results in the CFM Selector table. Depending on the scope of the search and the results, any number of tabs might be showing. Producers now can search any of the music data from one simple interface.

Since the first Alpha version was released, many other new features have been added at the request of the staff. Among these are a search history and a shopping cart. The cart can hold items from any table. Carts can be saved and restored, and once created, a cart number also can be e-mailed to the librarian. This saves staff having to print or e-mail whole lists of material—they simply e-mail the cart number.

Finding NZ Content and Duration Search

The most recent feature that was added was the alias search. An alias replaces a more complex set of terms that might be used a lot. An example of this is a search for New Zealand content—music that contains NZ artists or was composed by a New Zealander. This is useful as we have self-imposed NZ music quotas for both networks.

Over many years of data entry and staff changes, different fields and identifiers were used to indicate NZ status in the main Works database. The NZ Music alias automatically adds the required terms and fields to the query as an OR search. This was achieved by building a new class on top of the term parser and using it to extract any aliases from the query. The parser then adds the required parameters to the query stack maintained by the Query Compiler. Here are some BRAD Alias examples. The query:

```
Mozart @nza
```

gives us Mozart and any NZ content field true. The actual query looks like this:

```
SELECT * FROM brs.works WHERE (cf REGEXP '[[[:<:]]local[[[:>:]]]' OR cf  
REGEXP '[[[:<:]]nz[[[:>:]]]' OR lq REGEXP '[[[:<:]]nz[[[:>:]]]') AND MATCH  
ti,ra,cf,cd,cp,so,at,notes,lq AGAINST ('+Mozart' IN BOOLEAN MODE) ORDER BY ti asc LIMIT 1000
```

A duration search also was added so that producers can find material within certain ranges—it is quite common to need music by a certain composer of approximately a known duration. In BRAD, numbers in square brackets are treated as a duration query. BRAD can do approximate searches or searches within a range of durations. See the sidebar for some examples.

BRAD Duration Examples

Less than:

```
brahms [<20]
```

Between (you also can specify a range of times). The following looks for anything with mozart in it that is between 20 minutes, 30 seconds and 30 minutes, 15 seconds:

```
mozart [20:30-30:15]
```

Approximate matches—this looks for the time you specify plus or minus 10%; c is short for circa:

```
mozart [ c 24 ]
```

You also can add a time range. The following input retrieves items 24 minutes in length, plus or minus 1 minute:

```
mozart [ c 24 r 1 ]
```

Complex duration searches— the following searches for pieces with Beethoven as the composer that last between 20 and 22 minutes:

```
beethoven.cp [20-22]
```

The query compiled for the last search:

```
SELECT * FROM cfm.cfms WHERE (du <= 1320)
AND (du >= 1200)
AND MATCH ti,ca,ma, ra,cd,cp,so,at,notes AGAINST
('+beethoven' IN BOOLEAN MODE)
AND MATCH cp AGAINST ('+beethoven' IN BOOLEAN MODE)
ORDER BY ti asc LIMIT 1000
```

The Concert FM Selector data mentioned earlier has NZ artist and duration fields set correctly for all data, so these aliases can be used reliably on the whole data set. Because there is a mix of item types in the Works data, only those with a valid duration are searched. In the past, it was not possible to do any duration search at all within Works, so this is an improvement.

The Future

At the time of writing, I was asked about putting the company phone directory into BRAD, and a proof-of-concept pronunciation guide was added for our News department.

Conclusion

This project has enabled us to replace a key Radio NZ cataloging system and provide enhanced functionality to staff at a low TCO. It also has provided a storage platform for new and legacy data.

In the future, it may be possible for programme producers to do a single search on a composer or artist and get back a whole set of results that includes music tracks, interviews and archival material. It even could indicate the correct pronunciation of the person's name and provide his or her phone number.

BRAD probably will continue to be a work in progress as we find more uses for it. This is one of the benefits of DIY-IT—the system is ours to extend or modify as we see fit, whenever we need to do so.

Resources for this article: www.linuxjournal.com/article/7968.

Richard Hulse is a Senior Recording Engineer for Radio New Zealand and currently is working on a number of IT projects, including improving the Radio NZ Web site (www.radionz.co.nz).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Managing Projects with WebCollab

Mike Cohen

Issue #131, March 2005

This undiscovered gem of an open-source project brings project status information and important project files together with one streamlined Web interface. Try it and see if it fits your work style.

As a network consultant I frequently work with small teams on any IT-related project. For any given job, I have to coordinate with other consultants, clients, data-line vendors and office staff. I am constantly in need of a way to update coworkers on various aspects of a project, and oftentimes this includes sharing files, notes and comments. My corporate days have taught me that this can be done with a combination of the Microsoft Exchange public foldering system and Microsoft Project.

Although the Microsoft products are able to give me most of what I want, they have their limitations. First, Exchange/Project is a proprietary setup geared exclusively toward Microsoft Windows users and, more specifically, Microsoft users within the same organization. Second, Microsoft's Exchange package is not cost effective for most small companies. Third, although MS Project and other Gantt charting applications are the de facto standard for project management applications, they tend to take a macro-management approach to jobs and are a better fit for larger-scale operations like construction. Finally, I wanted integration between the file repository and the project status information, so that a project could be managed by a group of people with the proper collaboration between parties and across organizations.

I stumbled upon WebCollab on SourceForge and was pleasantly surprised. According to its creators, WebCollab is "a collaborative Web-based system for projects and project management; WebCollab is easy to use and encourages users to work together. The software is functionally elegant and secure without being cumbersome for users or graphically intensive."

I found all of the above to be true. The authors of the software designed with function over form in mind, and the interface is extremely plain and simple with speed and security as its primary goals.

WebCollab is ideal for projects involving small groups of users who have a fairly constant stream of communication. More than simply a multiuser to-do list, each created project carries with it a task list, due dates, color-coded completion status meters, priority settings, message boards and a file upload section. When any task has a change in status, there is an option to notify any involved users or groups by e-mail. Between the continuous message board banter, the file exchange and status e-mails, WebCollab creates an interactive environment for project management. A manager easily could use this as a tool to delegate tasks and keep tabs on exactly what is going on by having a constant dialog with employees, all through this software.

WebCollab has quite a few great features, and coupled with its simple install and nonexistent learning curve, it's a great fit for a small office environment or for projects involving people from different organizations. For example, I use this as a tool to keep my clients updated on the status of my work for them, as well as a tool to communicate with the other engineers and technicians I may be working with.

System Requirements/Architecture

The software consists of an Apache-hosted PHP front end to a database back end. Once the PHP pages are made available with the Web server, any computer with connectivity, a Web browser and user credentials can access WebCollab.

I found the optimal configuration to be a Linux, Apache, MySQL, PHP box, but any operating system capable of running Apache, PHP and either MySQL or Postgres can be used. The database can be hosted on a separate server if necessary. I am currently using a Pentium III 500MHz workstation with 256MB of RAM and a 20GB hard drive without issue, and I probably am overdoing things for my load of about 15 users. This software is perfect for the old system you have been meaning to use for something productive or as a lightweight service on an existing server. All personal biases aside, I'd recommend using Linux over Windows, or even Mac OS, because of the ease and security of Linux remote administration and its lower cost. To set up WebCollab, you need the ability to create a database and change a few permissions within the WebCollab directory. Depending what your file upload traffic is like, you shouldn't need more than a few hundred megabytes of space in your Web directory, because the individual file upload size is limited to 2MB.

With Apache and MySQL under the hood, I have had no stability issues, and the quality of the PHP code seems, in general, very solid. That being said, this is not a software package I would recommend for an enterprise-level organization. With a small user base, WebCollab is unproven under heavy load. Most corporate firms also put an emphasis on support when choosing software packages. With WebCollab's current status as a small open-source project, there is not a programmer standing by 24/7 to help with any data catastrophes.

Installation

Installation is extremely simple and took me less than ten minutes. I used my distro's vanilla installations of Apache, PHP and MySQL, which worked perfectly. Linux beginners will find the most difficult part of the install to be creating a new user in MySQL that has the appropriate access to the database. Aside from that, this is definitely something that a person who is just beginning to experiment with Linux can install without complication.

There are two methods of installation, one using the command line and the other through a Web-based setup routine. I chose the latter. For the sake of an example, I use collab.example.com here.

Download and unzip the tarball into your Web directory:

```
# tar -zxvf WebCollab-1.62.tar.gz
```

Change the permissions on the main config file:

```
# cd WebCollab-1.62/config  
# chmod 666 config.php
```

Point a Web browser to collab.example.com/WebCollab-1.62/setup.php. This guides you through the automated portion of the setup, which includes creating an SQL database and running a table creation script, as well as setting four environment variables in the relevant config.php file.

Restore the permissions on the config file:


```
# chmod 664 config.php
```

User/Groups

A user name and password are required to access any part of the software and determine which projects can and cannot be viewed and/or edited. As with most other systems, only administrative users have the ability to add or remove accounts. A user also can be designated as a project or task owner, giving that

account the ability to perform administrative tasks on that particular project or task.

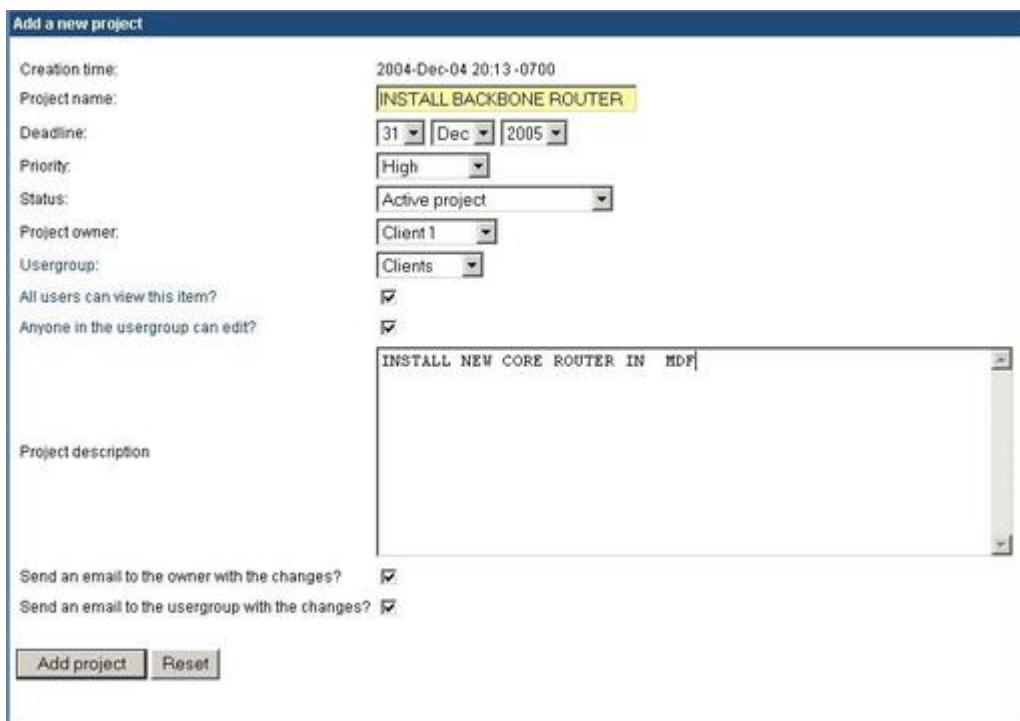
Groups also play an important role in task designation. Projects can be assigned by the owner to a user or to an entire group. Subgroups or task groups can then be used to further delegate tasks within a given project.



The screenshot shows a web interface for editing a user. On the left is a 'Main menu' with links to Home page, Summary page, ToDo list, Calendar, and Log out. Below that is a 'Users' section with links for Who is online?, Edit user details, and Show user details. The main content area is titled 'Edit a user' and contains a form with the following fields: Login name (Client), Full name (Sample User One), Password (with a note to leave blank for current password), and E-mail (client!@nodomain.com). At the bottom of the form are 'Submit changes' and 'Reset' buttons. The footer of the page reads 'Powered by WebCollab 0202-2004'.

Figure 1. Administrators can create new user accounts from this page. The e-mail address entered here is used for the e-mail notifications for status updates.

The access control is fairly comprehensive and provides a lot of flexibility for administrators and project owners. For example, I frequently have projects with other engineers, which required the ability of all parties to edit tasks, mark jobs complete and change due dates. In these cases, I assign everyone in my group edit rights to the project. I also have projects that I allow clients to view, and in these cases, I want to restrict them to read-only. I also do not want clients to be able to view any project other than their own. Both of these details easily are achieved by checking or unchecking the All users can view or Anyone in the user group can edit buttons.



The screenshot shows a web interface for adding a new project. The form is titled 'Add a new project' and contains the following fields: Creation time (2004-Dec-04 20:13-0700), Project name (INSTALL BACKBONE ROUTER), Deadline (31 Dec 2005), Priority (High), Status (Active project), Project owner (Client 1), and Usergroup (Clients). There are two checkboxes: 'All users can view this item?' (checked) and 'Anyone in the usergroup can edit?' (checked). A large text area for 'Project description' contains the text 'INSTALL NEW CORE ROUTER IN MDF'. At the bottom of the form are 'Add project' and 'Reset' buttons.

Figure 2. The creation form for a new project gives you the opportunity to set up access control and user notification, simply by checking the appropriate boxes.

Project/Task Creation

WebCollab is ideal for projects that can be broken down into a series of tasks with brief, one or two sentence, descriptions. Each project and task has a description field, start date, end date, priority and assigned group. When a task is created for a given project, it is treated as a subproject, with the same information fields and editable data as its parent job. This portion of the software is similar to most versions of popular to-do lists, but it adds the flexibility to be used as a quick checklist or a fairly in-depth breakdown of a task with running commentary.

The project and task views are where the software really shines. As mentioned before, each has its own file upload section and message board. This is where the collaborative aspects of WebCollab really set it apart from a traditional to-do list. Users can ask each other questions, make comments, upload relevant documents and much more. This is what separate WebCollab from task management applications and makes it more of a project-oriented groupware. I find the message board element of WebCollab to make things more interactive and thus more interesting for everyone involved.

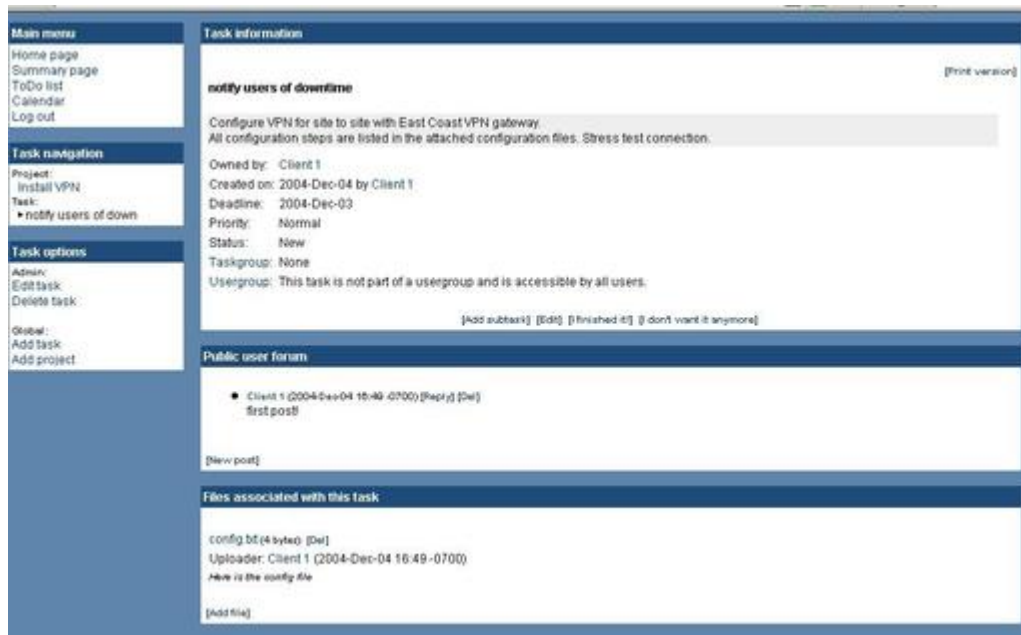


Figure 3. Task views include links to relevant files. This VPN task information page makes it easy to find the necessary configuration file.

Views and Navigation

The ability to navigate between views is intuitive and easy. Almost everything is a hyperlink, which, when clicked, takes you to a more in-depth view of that particular piece of data. For example, in the main view for a project, each task is

represented by its title, and clicking on that title takes you to the main view for that task, complete with summary, due dates, related files and more.

Every user or group name displayed within the context of the project or task descriptions is also a link to more information about that particular user. Everything initially is represented in a shorthand view, with more information available when clicked. Although this may not seem like much, it becomes handy and makes toggling between various views a breeze. A navigation bar also is present along the left side of the screen at all times.

As is typical with project management packages, there are a variety of different views, each of which gives a slightly different perspective. When a user logs in, he or she is initially at Home Page view. This shows a listing of projects and tasks that the user is involved in with completion status and due dates. Two other key views are the To-Do List view and the Calendar view; both are self-explanatory. Again, interpreting each view is far from complex, which is not always the case with project software. All views can be filtered by both user and group and have a print view button, which displays the current screen in a more paper-friendly format.



Figure 4. The Calendar view is basic, but it provides a nice visual breakdown of what has to be done when.

Security

WebCollab is best fitted for a small or home office environment. Being a noncommercial open-source project, the only real support available is through the message board on the Web site. Although that may be excellent, it usually is something that corporate higher-ups will frown upon. Given its tiny user base, I doubt that software like this is a target for attacks, but Apache and MySQL are.

For those who are not intimately familiar with Apache, PHP and MySQL, I would advise looking into using some of the Linux distro-specific security update tools, like Debian's apt-get or SuSE's YaST, to keep these packages up to date and as secure as possible.

The user management and access-control tools within the software itself do a good job of limiting what people can and cannot see, and these controls are granular enough that they can be customized on a user, group or subgroup level. All accounts and passwords are stored in MySQL with the passwords obviously being an encrypted field. It's probably a good idea to configure your Web server to serve this with SSL, over port 443, to deter any potential snoopers.

I also would suggest that anyone putting mission-critical data on this do a bit of homework on the interaction of PHP and MySQL with respect to passing database user names and passwords. All in all, I am fairly satisfied with the developers' efforts to make this software as secure as possible given its target audience.

Final Thoughts

WebCollab's beauty is in its simplicity. It's easy to install, use and maintain. It provides a comprehensive and flexible take on small-scale project management. Some users may be deterred by its lack of aesthetic detail, but if you prefer streamlined interfaces and quick-to-render pages to other bells and whistles, I would definitely suggest giving WebCollab a try. It's built on proven technology, so it's fast and stable, and because it's Web-based, it's essentially clientless. Users can be added to the system, and creating tasks and projects takes minutes. I also recommend this for transient users looking for a centralized to-do list that they can access from anywhere, as it's a very useful single-user tool.

As someone new to the Open Source community, I've found browsing sites like Freshmeat and SourceForge to be a lot like watching independent films or listening to indie rock bands. Every once in a while, you come across an unbridled gem that no one seems to know about; I've found WebCollab to be one of these.

Resources for this article: www.linuxjournal.com/article/7965.

Mike Cohen is a cofounder of Antropy, Inc., a small-business IT-consulting firm in Southern California. He enjoys spending time with his family and tow-in surfing at Todos Santos Island in Mexico.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

A Database-Driven Web Application in 18 Lines of Code

Paul Barry

Issue #131, March 2005

From zero to Web-based database application in eight easy steps.

The LAMP combination of Linux, Apache, MySQL and a programming technology (typically Perl, Python or PHP) is a powerful one. Once you've built one Web-based database application, however, you've built them all. From a programmer's perspective, things become boring and repetitive pretty quickly.

I recently worked on my first Web application. I built it on Linux, of course, running through Apache and talking to MySQL. I used Perl as my glue language, with CGI figuring heavily. I created all the code to talk to a MySQL table, adding/editing/updating as need be. And it all worked, which was good. What was bad was, I was faced with repeating this activity (and effort) for each of the remaining tables in my database. In a time-honoured tradition among Perl programmers, I started to look for ways to be constructively lazy. There had to be a better way. After a few false starts and some searching, I found Maypole.

Initially created by Simon Cozens and maintained by Sebastian Riedel, Maypole is a rapid application development framework for Web applications. Maypole's home page promised a fully functioning application in about 20 lines of Perl code. This sounded too good not to try.

Having tried Maypole, I can confirm that Simon and Sebastian are not lying. Only a handful of lines of code is required to build a very functional application. Some setup is required, but—critically—this activity is not programming. Once the setup is complete, any number of applications can be created, each with a handful of lines of code. In the rest of this article, I step you through building an application with Maypole.

Step 1: Install Linux (If Needed)

This step used to require an entire article to describe. Today, a single sentence summarizes. Pick your favourite distribution, and install it.

Having recently taken delivery of a new PC, I grabbed Fedora Core 3 and custom installed everything. If you don't have this luxury, be sure to install the following packages from your chosen distribution: httpd, httpd-devel, mod_perl, mod_perl-devel, mysql (client and server) and Perl.

Step 2: Prepare Your Apache/mod_perl Environment

Increasingly, modern distributions are shipping with release 2 of Apache and version 1.99 of mod_perl, as opposed to the entrenched 1.3.x release of Apache. Thankfully, Maypole can work with either release of Apache and also can be configured to use CGI (if mod_perl is not available). My Fedora installation shipped with release 2.0.52 of Apache and 1.99_16-3 of mod_perl, so that's what I use here. Users on the Maypole mailing list have reported successful installations on the vast majority of Linux platforms, including SuSE, Debian and Red Hat. Maypole also can be installed on Apple's Mac OS X and, with some extra effort, Microsoft's Windows.

As root, I edited Fedora's Apache configuration file at /etc/httpd/conf/httpd.conf and commented out the ServerTokens directive. I then arranged to start Apache automatically at boot time, and fired up the Web server using these commands:

```
chkconfig httpd on
service httpd start
```

The lynx text-based browser can be used to check the status of the server with this command:

```
lynx -head -dump http://localhost/
```

The results confirm that Apache and mod_perl are up and running, as shown on the third line of this output:

```
HTTP/1.1 403 Forbidden
Date: Wed, 17 Nov 2004 23:30:01 GMT
Server: Apache/2.0.52 (Fedora)
      mod_perl/1.99_16 Perl/v5.8.5 DAV/2
Accept-Ranges: bytes
Content-Length: 3931
Connection: close
Content-Type: text/html; charset=UTF-8
```

Happy that all was okay, I re-edited httpd.conf and uncommented the ServerTokens directive, as it is best not to give away too much about the

internals of your Web server to potential attackers. While in `httpd.conf`, I changed the `ServerAdmin` directive to a more appropriate e-mail address, then set `ServerName` to the DNS name for my server. I also made a note of the value set for `DocumentRoot`, which was `/var/www/html` on my machine.

Step 3: Prepare MySQL

Depending on the distribution you are running, MySQL already may be installed. If MySQL is missing, download it from your distribution's download area, or go to the MySQL Web site. On my Fedora machine, I issued the usual commands to prepare MySQL for use, while logged in as root:

```
chkconfig mysqld on
service mysqld start
```

With MySQL running, I then set the MySQL administrator password:

```
mysqladmin -u root password 'passwordhere'
```

Step 4: Install Maypole

Maypole interacts directly with Apache through `mod_perl`. To work with Apache 2, a development library called `libapreq2` needs to be fetched from the CPAN repository and installed into Perl. I downloaded `libapreq2-2.04_03-dev.tar.gz` from CPAN. Prior to installing the library, I upgraded the `ExtUtils::XSBuilder` module that ships with Perl. A single command, issued as root, suffices:

```
perl -MCPAN -e "install ExtUtils::XSBuilder"
```

If this is the first time the CPAN shell has executed, you'll be prompted to configure the local CPAN module. Be sure to select follow when asked about fetching prerequisite modules. With the module upgraded, I installed the `libapreq2` library with the usual set of Perl module installation commands:

```
tar zxvf libapreq2-2.04_03-dev.tar.gz
cd libapreq2-2.04-dev/
perl Makefile.PL
make
make test
su
make install
<Ctrl-D>
```

The actual installation of Maypole starts by invoking the CPAN shell as root:

```
perl -MCPAN -e "shell"
```

As Maypole depends on a large collection of prerequisite CPAN modules, installation can take a while. Prior to actually asking the CPAN shell to install Maypole for you, issue the following commands to ensure that some of the more troublesome modules are dealt with:

```
cpan> install CGI::Untaint::date
cpan> force install Class::DBI::mysql
```

I had to force the installation of `Class::DBI::mysql` as a number of tests failed, effectively aborting the automatic installation. By forcing the install, the broken tests are ignored, allowing the install to proceed. With the prerequisites dealt with, install Maypole with this CPAN command:

```
cpan> install Maypole
```

A series of automated interactions with the CPAN repository begin after this step. Keep an eye on what's going on, because at certain points, you have to respond to some self-explanatory prompts. When all was done and dusted, the most recent release of Maypole—2.04 at the time of this writing—was installed on my machine.

Step 5: Create a Database and Some Tables

Returning to MySQL, I logged in as administrator and issued these commands to remove any default accounts:

```
mysql -u root -p
mysql> use mysql;
mysql> delete from user where User = '';
mysql> flush privileges;
```

I then created a new database, together with a user to act as owner of the data:

```
mysql> create database CLUB;
mysql> grant all on CLUB.* to manager identified
      by 'passwordhere';
mysql> quit
```

These commands create the database, called CLUB, and add a user, called manager, to the database system. For the purposes of this article, this simple application manages data about an under-age soccer club. In addition to storing personal details about each player, the system maintains data on which players are in which squads, as well as any medical conditions players may have.

Here are the SQL files that I used to define the tables used within the CLUB database. The first file, `create_player.sql`, creates the player table:

```

create table player
(
  id          int not null auto_increment
             primary key,
  name       varchar (64) not null,
  date_of_birth date,
  address    varchar (255),
  contact_tel_no varchar (64),
  squad     int,
  medical_condition int
);

```

The second file, `create_squad.sql`, creates the initial list of squads:

```

create table squad
(
  id    int not null auto_increment primary key,
  name  varchar (32) not null
);

insert into squad (name) values ('--');
insert into squad (name) values ('Under 8');
insert into squad (name) values ('Under 9');
insert into squad (name) values ('Under 10');
insert into squad (name) values ('Under 11');
insert into squad (name) values ('Under 12');

```

The squad table is initialized to a reasonable set of default values. The third and final file, `create_condition.sql`, creates a list of possible medical conditions:

```

create table condition
(
  id    int not null auto_increment primary key,
  name  varchar (64) not null
);

insert into condition (name) values ('--');
insert into condition (name) values ('Asthma');
insert into condition (name) values ('Epilepsy');

```

As with the squad table, the condition table is initialized with some default data. The data item in the squad and condition tables is called `name`. The significance of this point will be returned to later in this article.

Use the SQL files to create the tables within the database:

```

mysql -u manager -p CLUB < create_player.sql
mysql -u manager -p CLUB < create_squad.sql
mysql -u manager -p CLUB < create_condition.sql

```

As can be guessed, the CLUB database maintains data on players. Players belong to a squad and may have a medical condition.

Step 6: Set Up Your Application

With the database ready and Maypole installed, it's time to configure the application. A directive needs to be added into the Apache `httpd.conf`

configuration file to set up a `mod_perl` handler for the Maypole application. I added the following to the end of the configuration file:

```
<Location /Club>
    SetHandler perl-script
    PerlHandler ClubDB
</Location>
```

These lines tell Apache that when a request is made for the `/Club` URL, it is to be handed off to the `ClubDB` Perl script, which we write in the next step. Use the following commands, as root, to set up the URL location:

```
mkdir /var/www/html/Club
cd /var/www/html/Club
cp -r ~/.cpan/build/Maypole-2.04/templates/* .
cp maypole.css ../club.css
```

Having first created a directory to contain my application's URL underneath Apache's root directory, I then copied the default templates that ship with Maypole into this location. I also copied Maypole's CSS file into my Web server's DocumentRoot, giving it a name that corresponds to my application.

One final setup activity involves creating a configuration file within Apache's `/etc/httpd/conf` directory to hold the application's MySQL user ID and password. Called `ClubDB.conf`, this file contains these lines:

```
[client]
user=manager
password=passwordhere
```

Step 7: Write Your 18 Lines of Code

The code for the Soccer Club Database resides in the `ClubDB.pm` file. Every Maypole application starts with a package statement declaring a Perl namespace. Strictness is turned on, then the base Maypole module, called `Apache::MVC`, is used:

```
package ClubDB;

use strict;

use base 'Apache::MVC';
```

The code then establishes a connection to the database, using the user ID and password from the named configuration file:

```
ClubDB->setup( "dbi:mysql:CLUB;
    mysql_read_default_file=
    /etc/httpd/conf/ClubDB.conf" );
```

A few more lines of code inform Maypole of the base Web address for the application, as well as a list of tables in the database to which to provide access. For this simple application, it makes sense to provide access to all the tables:

```
ClubDB->config->{uri_base} =  
    "http://webmason.itcarlow.ie/Club/";  
  
ClubDB->config->{display_tables} =  
    [ qw[ player squad condition ] ];
```

When it comes to squads, my application allows the user to view, edit or delete squad names. Specifying this takes a couple of lines of code, one of which sets up another namespace:

```
package ClubDB::Squad;  
  
sub display_columns{ "name" };  
  
ClubDB::Player->untaint_columns(  
    printable => [ "name" ] );
```

The `untaint_columns` method identifies the type of data expected in the column, as well as indicates to Maypole that the column can be edited using the Web interface. Medical conditions are handled in the same way:

```
package ClubDB::Condition;  
  
sub display_columns{ "name" };  
  
ClubDB::Condition->untaint_columns(  
    printable => [ "name" ] );
```

The code for the player table is more complex but not by much. After declaring another namespace, two calls to the `has_a` method establish the links between the player table and the others. The link is specified in terms of only the declared namespaces:

```
package ClubDB::Player;  
  
ClubDB::Player->has_a(  
    squad => "ClubDB::Squad" );  
  
ClubDB::Player->has_a(  
    medical_condition => "ClubDB::Condition" );
```

For players, we list the columns to display using the `display_columns` method. Doing so allows the programmer to control the order in which the columns appear within the Web interface. If `display_columns` is not used, Maypole displays the columns in alphabetical order, which may not always suit your needs. The invocation of `untaint_columns` identifies the types of data that can be edited within each of the columns. The code concludes with Perl's familiar `1;`, which is required of all Perl modules:

```

sub display_columns{ qw( name address
    date_of_birth contact_tel_no
        squad medical_condition ) };

ClubDB::Player->untaint_columns(
    integer =>
        [ "squad", "medical_condition" ],
    printable =>
        [ "name", "address", "contact_tel_no" ],
    date =>
        [ "date_of_birth" ] );

1;

```

Count the semicolons. Bearing in mind that the presented code has been formatted to fit the printed page, there are only 18 lines of code in all. All that's left to do is copy the Perl module into a location where Apache and mod_perl can find it:

```

mkdir -p /etc/httpd/lib/perl/
cp ClubDB.pm /etc/httpd/lib/perl/

```

Step 8: Give It a Go!

Restart Apache before accessing the Maypole application:

```

service httpd restart

```

I entered `http://webmason.itcarlow.ie/Club/` into the Firefox location bar, and up popped Figure 1, which, although something, was not quite what I was expecting.

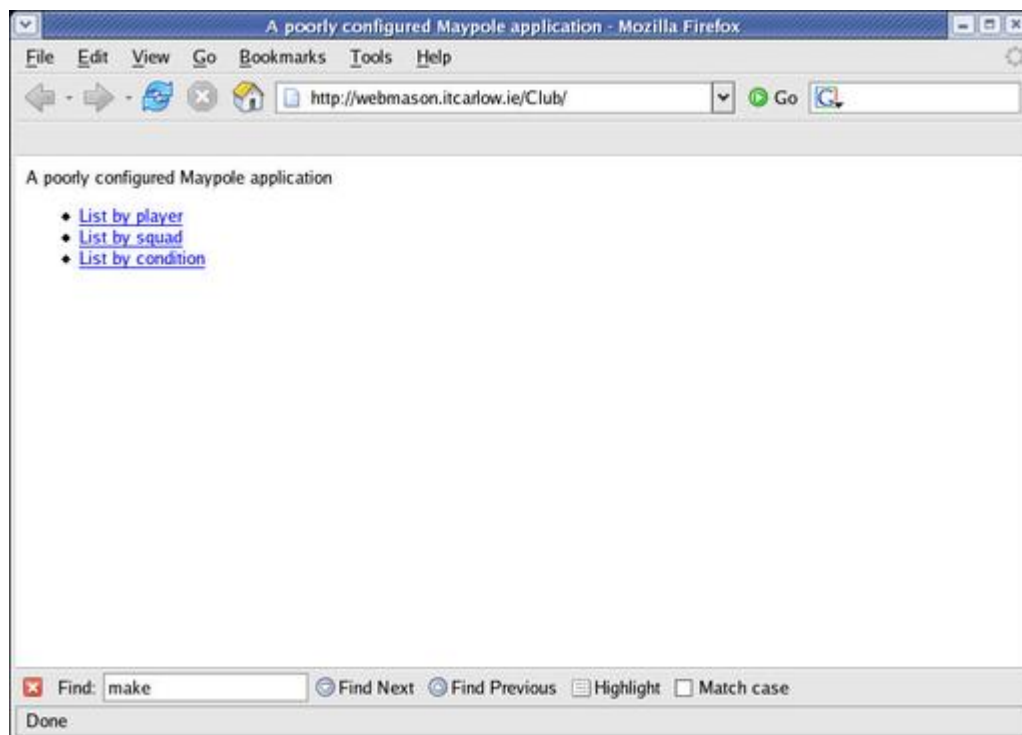


Figure 1. The Default Maypole Opening Screen

For starters, I was expecting to see some nice CSS output, not the plain HTML I was seeing. To fix this problem, I explored the default template files copied into the Web server during Step 6. By changing these, it is possible to alter the appearance of the application, without changing the source code to the application. The significance of that last sentence cannot be overstated. In essence, the way the application looks is controlled by the CSS templates. The way the application behaves is controlled by the code. The data used by the application is controlled by MySQL. All of this separation of duties makes for a very productive development environment, as changing one part of the application shouldn't adversely affect either of the others.

The templates live within a subdirectory called factory, located beneath the URL of the application, which is Club/ in this case. The factory templates are the Maypole defaults and are used unless overriding templates are found within another directory, called custom.

After creating the custom directory underneath the Club/ URL, I copied the header file from factory to custom and edited it with vi. I changed /maypole.css to read /club.css, in addition to replacing the "A poorly configured" message with a more appropriate description of the application. I also copied the frontpage file from factory to custom and edited it to use a better application description. Then, I changed the anchor tag within custom/frontpage to read "Work with the player data" as opposed to the default "List by player" text. With these changes made, I clicked the Reload button within Firefox, resulting in Figure 2, which—I think you'll agree—looks a whole lot better.

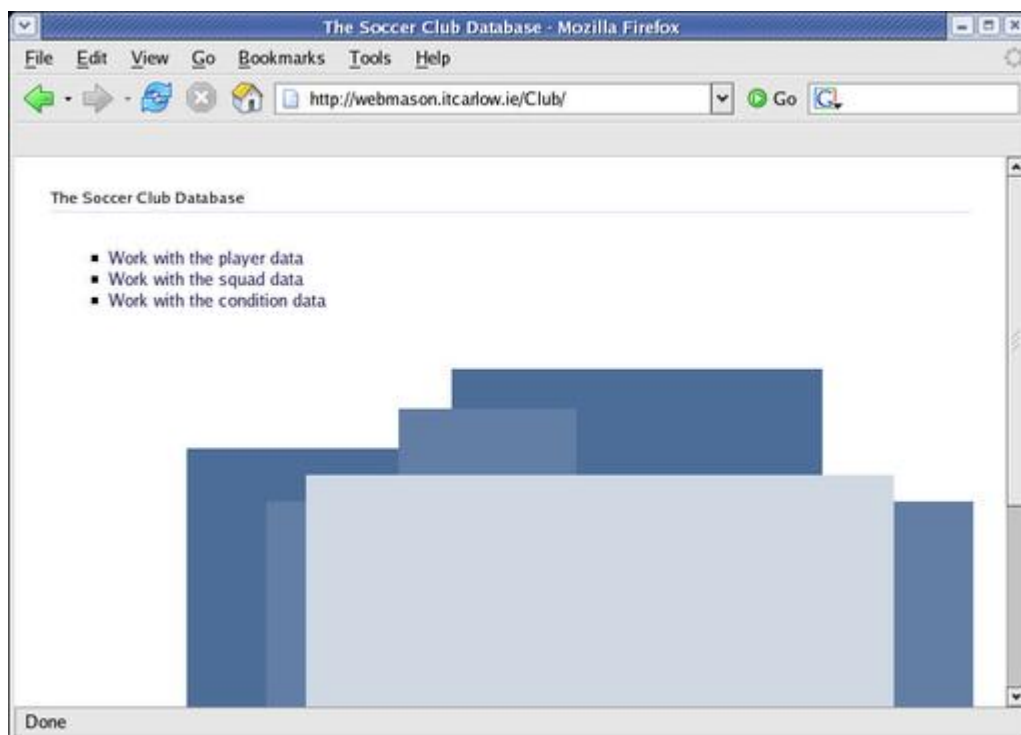


Figure 2. The Customized Soccer Club Opening Screen

Clicking on any of the menu options produces a beautifully formatted input screen, like those shown in Figures 3 and 4.

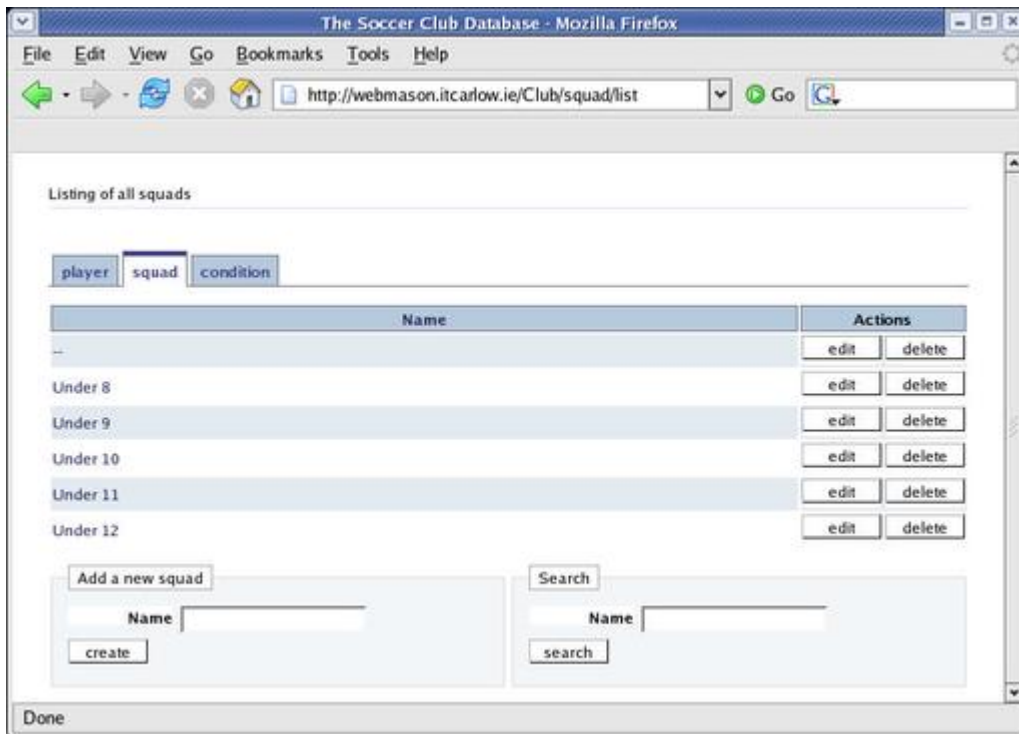


Figure 3. The Maypole Front End to the Squad Table

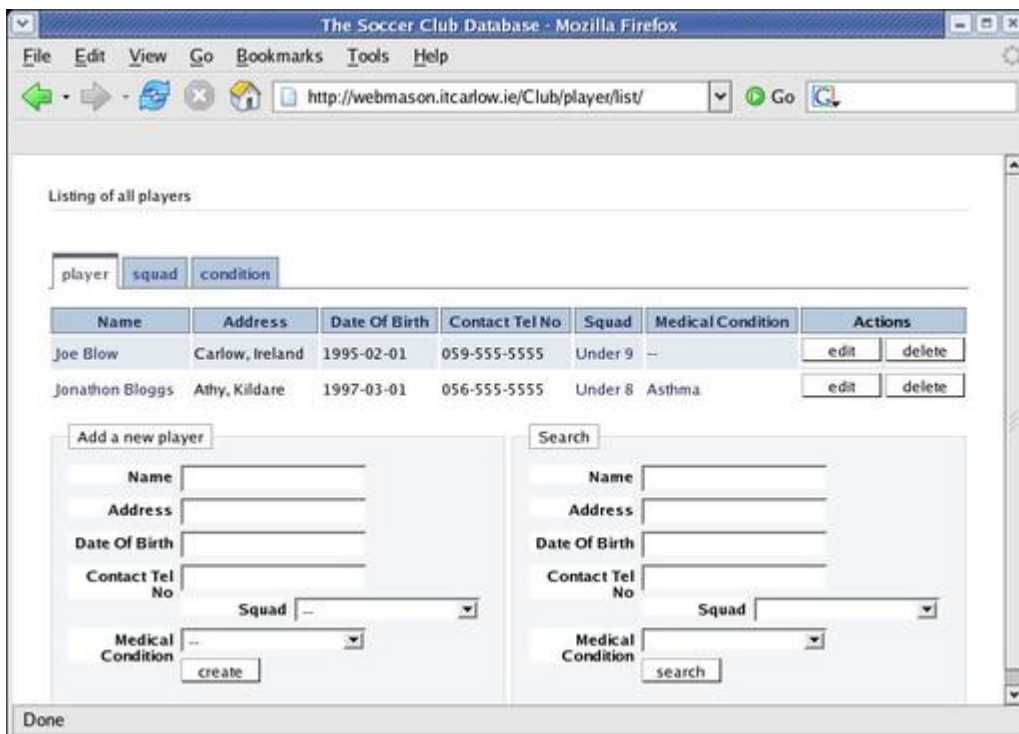


Figure 4. The Maypole Front End to the Player Table

Figure 4 shows the display after the entry of two fictitious players. Notice all the functionality provided for free. Tabs for each of the tables are located along the top of the display. Simply click on the tab to display that table's data. Each row of data has an associated edit and delete button. Click on any column heading

to sort the display on the data in that column. Perform a search using the provided search form. Add more players using the add form. Notice the drop-down menus for the player's squad and medical condition. Click on the field and a drop-down box appears with the choices available to you. This bit of magic occurs because Maypole has been told that each player "has a" squad and "has a" condition. By default, Maypole uses the name data column in the referred to table to provide the data to these drop-down boxes.

And, that's it—a fully functioning Web interface to an underlying database, in eight easy steps.

Despite the fact that Maypole is quite new, an active community already has gathered around it. The mailing list recently split, one for developers and the other for users, and the Maypole Web site is now hosted by perl.org.

As I hope I've demonstrated, Maypole—once set up—is a breeze to use. Most of the guts of any Web application is provided for free. Adding additional functionality also is possible. Maypole is not stuck on MySQL either, as any SQL DBMS can be used. Refer to the articles and documentation referenced on the Maypole site for more details.

Resources for this article: </article/7964>.

Paul Barry (paul.barry@itcarlow.ie) lectures at the Institute of Technology, Carlow in Ireland. Information on the courses he teaches, in addition to the books and articles he has written, can be found on his Web site, glasnost.itcarlow.ie/~barryp.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Introducing Ardour

Dave Phillips

Issue #131, March 2005

The heart of your Linux recording studio is the hard-disk recorder. Get started with Ardour, which brings pro recording features to open source.

Contemporary musicians employ the computer as a digital audio workstation (DAW) to perform a wide variety of tasks dealing with sound. Typical uses include recording and editing soundfiles, adding effects and dynamics processing and preparing audio tracks for a CD master disc.

The centerpiece of the modern musician's computer-based studio is the hard-disk recorder (HDR). Musicians working on Apple or Microsoft Windows machines have an impressive selection of HDR systems to choose from, but until recently Linux users have had nothing truly comparable for professional work. A professional-quality HDR is a profoundly nontrivial programming endeavor, and proprietary HDR developers have provided little technical guidance for would-be designers of an open-source DAW.

Today, thanks to the talent and perseverance of chief designer/programmer Paul Davis and his talented crew, Linux musicians now have a native-born professional-quality HDR/DAW, named Ardour.

What It Is, What It Isn't

Ardour is a multitrack recording and editing system for high-quality digital audio. Ardour supports audio processing plugins (LADSPA and VST), parameter control automation, sophisticated panning control and many advanced editing procedures. Recognized synchronization protocols include MIDI time code (MTC), a means of encoding SMPTE time code to MIDI; MIDI machine control (MMC), a set of MIDI messages for controlling transport features of hardware mixers and recorders; and JACK, a low-latency audio server and application transport control interface for Linux and Mac OS X.



Figure 1. Ardour is a multitrack recording and editing system that supports consumer and professional audio hardware.

Professional audio recording hardware supports datatypes not commonly encountered in consumer-grade systems. A pro DAW can handle audio at greater bit depth, for deeper amplitude range and precision; at high sampling rates, for more accurate frequency resolution; and with greater flexibility in sound location and spatialization. It is not uncommon for professional recordists to work with 32-bit soundfiles recorded with a sampling rate of 96kHz, more than twice the resolution of compact disc audio.

Ardour is not a MIDI sequence recorder or editor. It knows nothing about music notation, and it is not designed to be a soundfile editor. Ardour has only a few built-in signal processing capabilities, and most of its processing power comes from its supported plugins. Finally, Ardour does not directly provide facilities for CD mastering and burning, but it is designed to work well with the excellent JAMin mastering suite.

Hardware Requirements

The extent of your use of Ardour is limited fundamentally by the capabilities of your hardware. If a sound card or audio board is supported by the ALSA sound system, it should work with Ardour; however, standard consumer-grade audio devices are not suitable for using Ardour to its full extent. You can do great things with Ardour and a SoundBlaster Live, but for professional work, Ardour is happiest with a multichannel digital audio interface, such as the RME Hammerfall and M-Audio Delta cards.

Do your homework before purchasing your audio interface. Check the ALSA site for up-to-date news regarding supported systems, and try to find others who can comment on the suitability of a particular card. Ardour can respond to MIDI parameter control, so study the MIDI implementation charts for any external equipment you plan to use. See if your mixer specifications support MMC or MTC. Ardour is designed to work with automated mixer control surfaces, but again, your equipment has to support the features.

The question of a sufficient base computer system often pops up on the Ardour users mailing list. Satisfactory results have been reported with a 500MHz CPU, but such a system is too limiting for professional use. A fast CPU ensures accurate synchronization while recording or playing multiple audio tracks, and it is absolutely necessary if you intend to use many effects. For example, a good reverberation effect can be intensely CPU-hungry. You also should have a large, fast hard disk, properly tuned with the `hdparm` utility for maximum performance. Multiple tracks of streaming audio data absolutely require a powerful CPU and a fast hard disk to ensure perfect synchronization. Also, 32-bit digital audio files can be huge, and a typical recording session can create nontrivial storage demands. For professional use, you are advised to equip your system with two disks—one for your system and application software and one dedicated only to session audio storage. The Ardour Web site offers suggestions for specific CPUs, hard-disk specifications and even recommended motherboards. For best results with Ardour, follow the designer's advice.

Aaron Trumm's excellent article "The Linux-Based Recording Studio" [*Linux Journal*, May 2004] describes room considerations and the setup and configuration of external equipment needed for serious recording. Rather than rehashing Aaron's recommendations, I simply refer readers to that article for advice on selecting microphones, mixers, monitor speakers and other outboard gear.

Software Requirements

The current public version of Ardour is available as source code. Packages are available for various Linux distributions, including Red Hat/Fedora, Mandrake, Debian and Slackware; see the on-line Resources. CVS access is selective at this time, but a nightly tarball is available from the Ardour Web site. Compiling Ardour from its source code is uncomplicated, and complete instructions are included with the tarball.

Check the Ardour Web site for the latest support software required to build and install the program. Ardour's dependencies include up-to-date installations of the ALSA kernel sound system, the JACK audio server, the LADSPA plugin API and collections and various other audio-related software components. You can

configure your existing Linux installation for optimal performance with the program, but if you're serious about using Ardour you are advised to install an audio-optimized system such as AGNULA/Demudi, a Debian-based distribution, or Planet CCRMA, Red Hat/Fedora packages. Slackware users should install Luke Yelavich's AudioSlack packages, and Mandrake users can find all necessary packages at Thac's site.

JAMin is a suite of post-production audio utilities designed for the preparation of tracks destined for burning to a master CD. This mastering process uses tools such as compressors, equalizers and limiters to reduce frequency and amplitude imbalances between tracks. If you plan to record a full CD with Ardour, you should plan on mastering it with JAMin.

Major Features

Any modern HDR performs three basic functions, corresponding to the typical work-flow stages of recording, editing and mixing digital audio. Each stage can be quite complex within itself, but Ardour's user interface sensibly organizes the program's complexities.

Ardour records to the available hardware limits. If your audio interface supports 32-channel I/O and there's an ALSA driver for it, you should be able to work with 32 simultaneous audio channels. Assignment of channels to tracks is completely flexible, and each track supports mono or stereo input. Ardour's synchronization capabilities currently work best with JACK-compliant applications, such as the Hydrogen drum machine and Rosegarden audio/MIDI sequencer. Much work, however, is going into improving support for MIDI time code.

As mentioned above, Ardour is not a soundfile editor, but it does include a variety of edit processes optimized for multitrack recording, such as typical and not-so-typical cut/copy/paste operations, grouped track editing and fine control over track and segment relocation. Ardour offers its own time stretching and amplitude normalization, but the bulk of its processing power comes from LADSPA plugins.

LADSPA

The Linux Audio Developers Simple Plugin Architecture (LAPSDA) is an easy-to-implement programming interface for hosting effects and other plugins in Linux audio applications. The API has been adopted widely by Linux audio developers to the point that users expect LADSPA in new sound and music software.

Ardour lets you work with your data as tracks, regions, ranges, chunks and groups. Data display is variable: you can enlarge or reduce a track's display, and grouping lets you view only the member tracks in a specified edit group. Figure 2 shows off Ardour's track display resizing feature, along with the region editor and the gain automation curve.



Figure 2. Resized Tracks, the Region Editor and a LADSPA Plugin

How Many Tracks?

As it's used by recordists, the term track properly applies to a concept derived from the possibilities of recording sound to magnetic tape. Digital audio is recorded to your hard disk in a different manner, but the old terminology persists in the graphic representation of multiple audio streams as linear tracks. The concept of a channel is perhaps best understood in the context of a mixer. Multichannel mixers provide a number of inputs (channels) whose signals can be combined and routed freely before being sent to the mixer outputs. Likewise, a multichannel digital audio interface provides a number of channels to be combined and routed to your HDR's tracks. Channel limits are determined by your interface hardware.

As mentioned earlier, consumer audio interfaces rarely venture beyond stereo I/O, while professional hardware can deliver 50 or more channels. Disk space is consumed rapidly by alternate takes, temporary tracks and backup copies of your session files. Your software may limit the number of available tracks, but your disk speed determines the number of available simultaneous tracks.

Remember, high-quality digital audio is a heavy hitter on your system resources.

Ardour's mixer panel presents a series of per-track fader strips and a master control strip. A track strip is divided into sections corresponding to a "from the top downward" data path. Input begins at the top of the strip and passes through controls such as signal polarity and track solo/mute status. It moves on through pre-fader plugins, the track fader itself and post-fader plugins before reaching its pan position and heading out to the output or outputs, typically but not always a master bus. Other notable features of Ardour's mixer include gain automation, record and playback fader movement; MIDI control to and from mixers with MIDI machine control; and plugin parameter control automation.



Figure 3. In the Mixer panel, think of audio data moving from the top down through each track strip.

Thanks to Ardour's use of Erik de Castro Lopo's sndfile library, you can import and export audio data (track, region and session) to and from Ardour in any

format supported by libsndfile. Currently, that's about 20 different audio file types, including the most popular consumer-grade and professional formats.

A Session with Ardour

My goal in the following example session is to demonstrate how Ardour functions as a multitrack recording system. I've tried to keep technical terminology to a minimum, but this article does not intend to be a primer for digital recording. Basic information on the subject can be found at www.homerecording.com, while more advanced topics are covered at www.prorec.com. Many other on-line and hard-copy resources can be found with relevant searches on Google and Amazon.

Session hardware included an M-Audio Delta 66 digital audio interface, a system that includes a PCI card and a breakout box that together provide 4x4 analog I/O and 2x2 digital I/O. The digital ports can be configured for either S/PDIF, a high-quality consumer-grade digital I/O or AES/EBU, a standard for the recording industry. Each input point is a stereo port, effectively giving me a possible total of 12 input channels, with far more flexible routing than is possible with a consumer-grade, stereo-only sound card.

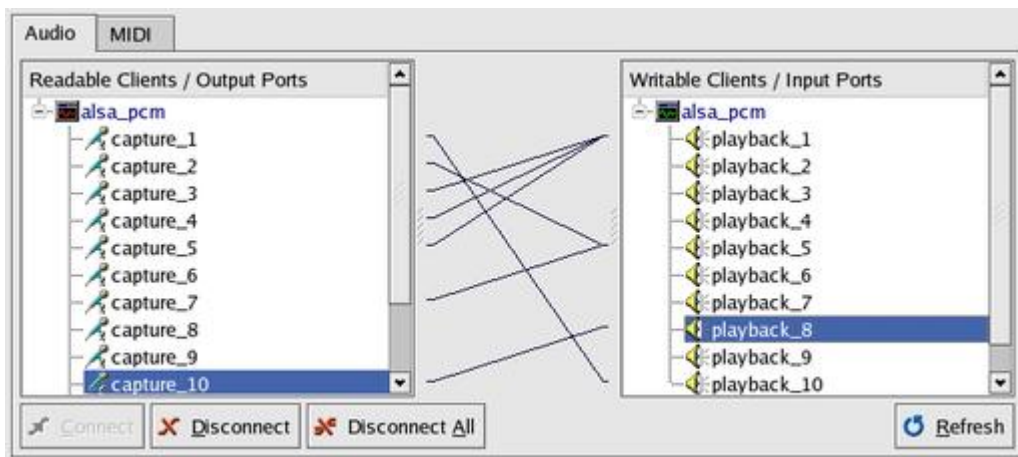


Figure 4. Delta 66 I/O Channels Displayed in qjackctl

I employed two external mixers for the session. A Yamaha DMP11 was used as a submixer for external synthesizers, and I used a Tascam TM-D1000 for mixing vocal, guitar and harmonica performances before sending them to the Delta 66. The Tascam mixer provides S/PDIF digital output, so I routed its feed to the digital ports of the Delta card.

My plan was to use Ardour to record an original song, with multiple instrumental and vocal tracks, and mix it to recreate the sound of a small group playing live. However, in this session the small group is only me, with some imported WAV files and some multitracking in Ardour. I also planned to use a few LADSPA plugins to add effects to some of the tracks and to use Ardour's

pan controls to position my tracks across the stereo audio panorama, as players would be positioned on a stage.

I used a MIDI sequencer to create parts for piano, bass, guitar and drums. I saved each part as a MIDI file and converted them all to WAV audio files with the popular TiMidity MIDI utility. The drum track was converted to a stereo file, the others were converted to monaural files and all the files were created in 16-bit 44.1kHz WAV format. At that point they were ready for importing to Ardour.

When you open Ardour for the first time, you see an empty track display complete with a kindly reminder that you need to create an Ardour session by using the Session/New dialog. Session templates are available, but I knew my immediate track needs and created a custom track layout—one stereo and four mono tracks. I imported my WAV files into those tracks, and there was my backing band.

Next, I recorded three more tracks in Ardour itself, adding a rhythm guitar part, the vocal track and a harmonica solo. Recording in Ardour is simple: click on the R button in your selected track to arm it for recording and then set your inputs and levels in the track's mixer strip. Next, click on the main display's big red Record button, click on the transport play control and record at will. You can monitor some or all other tracks, muting and soloing tracks and groups of tracks in real time for testing different ensembles.

When I was happy with the recorded performances, I started working on the mix. Many aspects of the raw mix were in need of attention: the rhythm guitar track needed a volume fade-out at the end and equalization (EQ) throughout, the MIDI instruments weren't bright enough in the mix and everything needed to be normalized and balanced. Fortunately, Ardour handled these tasks with the greatest of ease. I used LADSPA plugins for equalization and amplification, and I employed Ardour's own internal normalization routine when it was needed. I also added reverb to the vocal and harmonica parts, again by using a LADSPA plugin.

Adding the fade-out to my rhythm guitar track proved to be an interesting task. First, I clicked on the track's automation button to set the automation curve display, and then with the mouse on the Gain mode button I could draw the amplitude curve. Then, I discovered that I could use Ardour's control automation in the mixer as well. I set the automation state to write, and then I played the section to be faded out and reduced the fader level. Ardour recorded my adjustments to the fader, I reset the automation status to play, and the fader moved downward automatically on playback. By the way, faders can be ganged together as a mix group for simultaneous operation, including the recording of simultaneous automation curves.

Normalization and Equalization

Normalization is a process that increases the peak amplitude of a signal to the maximum level before clipping. All other amplitudes are then raised in their original proportions to the new peak amplitude. Equalization boosts or cuts the strength of a specified frequency or range of frequencies. The graphic equalizer found in a car stereo system is a common example of one kind of EQ device called a shelving equalizer. It divides the audible spectrum into more or less narrow bands and provides a control for boosting or weakening the strength of the frequencies within each band.

So how did all this creative activity turn out? You can hear the results for yourself; see Resources for the URL.

Bear in mind that the recording has not yet been mastered, and I still can return to my original session and make other changes. Feel free to suggest improvements, but try to be kind about my modest efforts.

Synchronization

Paul Davis also is responsible for creating the JACK low-latency audio server and transport control interface, so you would expect Ardour's JACK synchronization capabilities to be well evolved. I tested the Hydrogen drum machine and the Rosegarden sequencer running in JACK master and slave modes and experienced mixed levels of success. Both programs were synced to and from Ardour by way of JACK without trouble. Hydrogen got along nicely with Ardour, but I had problems recording the output of softsynths driven by Rosegarden. The Rosegarden team is aware of this issue and intends to resolve it, but you should check the Rosegarden Web site for the latest news.

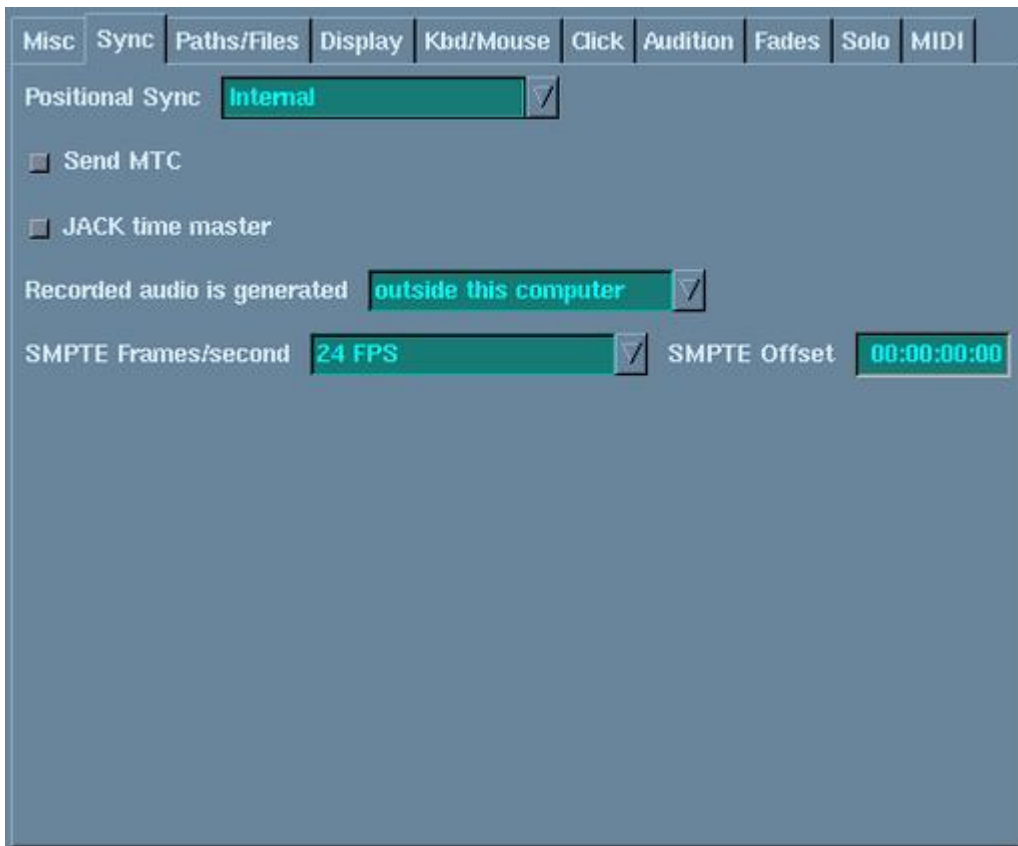


Figure 5. The Options Window

At the time of these tests, Ardour's support for MTC send/receive was under heavy revision, so I was unable to complete any meaningful tests. However, MTC support is a major item on many Ardour users' wishlists, and the remaining bugs should be worked out before version 1.0 is released. Other means of synchronization may be supported in future versions of Ardour. Direct SMPTE reading, MIDI clock and SPP (song position pointer) have been suggested as likely candidates, but work on those protocols will have to wait until after the Ardour 1.0 release.

Impressions

As I learn more about Ardour, there seems to be even more to learn. It's a credit to Ardour's designers that the initial interface view is clear and uncluttered, using pull-down and pop-up menus to reveal many underlying features. Also, many helpful keyboard bindings exist, but you need to run `ardour -b` at the command prompt in an xterm to discover them.

After gaining more confidence with Ardour's basic features, I started to explore more of its capabilities. The TM-D1000 sends MIDI messages and controller streams for most of its actions, and Ardour's controls can be bound to external MIDI control surfaces. Ctrl-middle-click on a button or fader and then activate the controller to set the binding. This feature is very cool, allowing any hardware device that sends MIDI controller streams to act as a control surface

for Ardour. Incidentally, group definitions still are in effect, so you can control multiple faders in Ardour from a single external fader.

The TM-D1000 also transmits and responds to MIDI machine control (MMC) commands, a valuable feature because Ardour can control and be controlled from such a device. MMC messages include typical transport control actions such as start/stop, fast-forward and rewind. Thus, a MIDI-sensible mixer such as the TM-D1000 can function as a hardware control surface for almost all of Ardour's operations.

In an article of this length I could test only the features most relevant to my goals. I have not worked yet with Ardour's looping mechanisms, MTC still was stabilizing as I completed this article, I didn't check out the timestretch capability and so on. As I said, Ardour is a deep application, and there are many useful and interesting features not touched on in this article's use of the program.

The Future

Development activity around Ardour is intense, especially as the program closes in on its 1.0 release. Many people are interested in a viable alternative to the proprietary lock-in solutions available for other operating systems, and Ardour appears to be moving along the right development path. Much work remains to be done, including improved MIDI capabilities, video track support, expanded synchronization possibilities and a GUI overhaul (GTK2 support is planned). However, Ardour currently is in a feature-freeze, with bug fixes and stability being the first order of the day before version 1.0 is released.

As might be expected in a pre-1.0 beta release of a large-scale complex project, it is not quite bug-free. Ardour's Mantis bug-tracking and feature-request system provides an excellent way to check on the status of known problems, report new ones and make suggestions for future releases.

VST plugin support currently is problematic, due mainly to continuing changes in WINE and Linux kernel development, but it is a high-priority item for the developers. Many users have expressed their willingness to switch platforms for their audio work if VST/VSTi support becomes seamless under Linux.

VST/VSTi Plugins

VST/VSTi audio plugins are a staple item in the sound software world. The VST API was created by the Steinberg company, makers of the popular Cubase audio/MIDI sequencer, and has been adopted by developers and embraced by users worldwide. Properly speaking, a VST plugin typically is an audio or MIDI processor, and a VSTi plugin is an instrument such as a synthesizer or drum

machine. Thousands of VST/VSTi plugins exist today, ranging from the homemade to the costly and commercial. Many free-of-charge VST plugins are quite good, and some VST authors have released their plugins as truly free open-source software licensed under the GPL.

Ardour's documentation is another lively issue, because currently there is no official users' manual. It is likely that Paul Davis will continue to make Ardour freely available while charging a fee for a high-quality manual. Meanwhile, users unfamiliar with the basic design concepts of a hard-disk recorder are advised to retrieve and study manuals for proprietary DAWs, such as Pro Tools or Cubase. Some Ardour-specific documentation can be found in the source package text files and in various on-line resources, such as the Quick Toots series (see Resources), and in the traffic on the ardour-users and ardour-dev mail lists. Developers and testers also communicate on the #ardour IRC channel, while normal users carry on considerable discussion of Ardour-related matters on the mail lists for AGNULA/Demudi, Planet CCRMA, ALSA and the Linux Audio Users group.

Is Ardour ready for the big time? Perhaps not quite yet, but its road map is clearly headed there, and the remaining trip won't take long. I believe that it will be only a short time before Ardour starts raising eyebrows in the mainstream commercial audio software world. Ardour already has been used to record and mix entire CD projects, and more users are reporting success with Ardour in their own recording projects. I expect to be making a lot more music with Ardour. Feel free to stop by my site and check the occasional results.

Acknowledgements

The author sends vast thanks and appreciation to Paul Davis, Taybin Rutkin, Jesse Chappell, Steve Harris and all the other members of the Ardour development crew. Their work on Ardour, and so many other valuable Linux audio projects, truly is innovative and indeed a labor of love. The free musicians of the world salute you!

Major thanks also go to the members of the Ardour users mail list, especially Jan Depner, Mark Knecht, Aaron Trumm and Josh Karnes. Developers and users all were helpful as I found my bearings in some of Ardour's trickier places, proving that good company does indeed lessen the difficulties.

Resources for this article: www.linuxjournal.com/article/7969.

Dave Phillips is a musician, teacher and writer living in Findlay, Ohio. He has been an active member of the Linux Audio community since his first contact with Linux in 1995. He is the author of *The Book of Linux Music & Sound*, as well as numerous articles in *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

Centralized Authorization Using a Directory Service, Part

II

Alf Wachsmann

Issue #131, March 2005

Get a handle on administering who can log in where, with a proven, reliable centralized directory.

Authorization is the process of deciding if entity X is allowed to have access to resource Y. Determining the identity of X is the job of the authentication process. One task of authorization in computer networks is to define and determine which user has access to which computers in the network. A simple example would be one line in a computer's `/etc/passwd` file, `joe:X:1234:56:/home/joe:/bin/bash`, to allow user joe access to this computer. If you want to give user joe access to several computers, you have to add this line to every computer's `/etc/passwd` file.

On Linux, the tendency exists to create a local account for each single user who should be allowed to log in to a computer. This typically is the case, because a user needs not only login privileges but also access to additional resources, such as a home directory to do some work. Creating a local account on every computer takes care of all this.

The problem with this local account approach is that these accounts can be inconsistent with one another. The same user name could have a different user ID and/or group ID on different computers. Even more problematic is when two different accounts share the same user ID and group ID on different computers. User joe on computer1 could have user ID 1234 and group ID 56, and user jane on computer2 could have the same user ID 1234 and group ID 56. This is a big security risk in cases where shared resources are used. These two different accounts are the same for an NFS server, so these users can wipe out each other's files.

The solution to this inconsistency problem is to have only one central, authoritative data source for this kind of information and a means of providing all your computers with access to this central source. This is what a directory service does. The two directory services most widely used for centralizing authorization data are the network information service (NIS, formerly known as yellow pages or YP) and lightweight directory access protocol (LDAP).

NIS vs. LDAP

A few things need to be considered when it comes to deciding which directory service to use, NIS or LDAP. If your company already maintains an LDAP server, it seems simple enough to add the authorization data to it. However, usually company LDAP servers are used for white pages and similar fairly lightweight tasks. Adding the authorization task puts a significant load on an LDAP server, because every single lookup for user name, UID, GID and so on done by programs needs to be answered by it. It usually makes sense to add an additional LDAP server dedicated solely to authorization. Also, due to the many different kinds of directory queries, it is rather hard to get the performance tuning right. You need to add all necessary LDAP index definitions in your slapd.conf file in order to speed up common lookups, but you don't want to add too many index definitions. Doing so makes the LDAP back-end database files large, and everything slows down again.

LDAP is the better choice in networks that have problems with many dropped UDP packets, because it uses TCP/IP where retransmits are built into the network protocol layer. NIS, on the other hand, uses remote procedure calls (RPCs) over UDP. Every dropped packet results in a non-answered NIS query, and the NIS client needs to repeat the query. Use the command `netstat -s -u` at different times on different machines on your network to see whether your network suffers from this problem. You should see very few errors reported by this command.

I concentrate on NIS in this article, because it is easier to start out with and there is a fairly simple migration path to LDAP in case you see problems. PADL Software Pty, Ltd., provides a set of open-source tools to help you convert all your NIS data files to LDAP (see the on-line Resources). You still have to do the performance-tuning part, however. You have to write migration tools yourself if you want to migrate from LDAP to NIS.

Configuring the NIS Servers

An NIS server does not require a lot of hardware resources. Any machine you have around should do the job. You might want to put this new functionality on a dedicated machine, though. At the Stanford Linear Accelerator Center (SLAC), we serve, without any problems, up to 500 Linux and Solaris clients with one

old Sun Netra T1 server. We have four of these NIS servers for about 700 Solaris and Linux desktop computers and another six NIS servers for about 2,500 Solaris and Linux compute servers. Our clients are spread out somewhat unevenly over the servers.

Master Server Configuration

Log on to the machine where you want to install your master NIS server, and make sure the latest portmap, ypserv and yp-tools RPMs are installed. If not, download and install them now. All following commands have to be issued as root. Start the portmapper daemon with:

```
# service portmap start
```

The next step is to define the name of your new NIS domain. This name can be anything you like, but it probably makes sense to pick one that represents your department inside your company; nis.example.com for an NIS domain for all of Example.Com or eng.example.com for the Engineering department inside of Example.Com would be good choices.

Set the NIS domain name on your master server with the command:

```
# domainname nis.example.com
```

You also have to add the line:

```
NISDOMAIN=nis.example.com
```

to the file /etc/sysconfig/network.

Restrict access to your new NIS server by creating a file /var/yp/securenets with the content:

```
# netmask      # network
255.255.255.0  192.168.0.0
```

This is a crucial security step. The world is able to query your NIS server if you don't have this file.

The next step is to define the things you would like to put into NIS. For the purpose of authorization, the /etc/group and /etc/passwd files as well as something called netgroup are sufficient. However, many more things are possible. To get an idea, have a look at the file /var/yp/Makefile on your NIS server.

Below, I show how the three files I've mentioned are configured to be distributed by way of NIS.

Adjust the Makefile generating the NIS map database files:

```
# cp /var/yp/Makefile /var/yp/Makefile.save
# vi /var/yp/Makefile
```

Change the following two entries from true to false to prevent the merging of passwd and shadow files as well as group and gshadow files:

```
MERGE_PASSWD=false
MERGE_GROUP=false
```

Change the directory name where NIS should look for its data sources:

```
YPSRCDIR = /etc/NIS
YPPWDDIR = /etc/NIS
```

Comment all files from which the NIS databases should not be built. I left only these three files:

```
GROUP      = $(YPPWDDIR)/group
PASSWD     = $(YPPWDDIR)/passwd
NETGROUP   = $(YPSRCDIR)/netgroup
```

Comment the line starting with a `ll`: that contains the list of all potential NIS maps. Add the new line:

```
all:      passwd group netgroup
```

Watch out for TAB characters. In a Makefile, you must use only TAB characters, not spaces, to indent commands.

Now, create the data source directory defined in the Makefile:

```
# mkdir /etc/NIS/
# chmod 700 /etc/NIS
```

and put a passwd file in there:

```
# grep -v '^root' /etc/passwd > /etc/NIS/passwd
```

You should remove not only the root account but all system accounts from this file and leave only the real user accounts.

If you still are using `/etc/passwd` with encrypted passwords, it now is time to convert them to Kerberos 5, as described in the previous article [L], February

2005]. If you don't do this, your encrypted passwords are exposed on the network when the passwd file is distributed to the slave NIS servers or to the NIS clients.

Now, collect the local /etc/passwd files from all the machines that are to be members of your new NIS domain. Remove all system accounts and then merge them together with:

```
% cat passwd_1 passwd_2 passwd_3 ... > passwd_merge
```

Remove all duplicate entries with this command:

```
% sort passwd_merge | uniq > passwd_uniq
```

Check the consistency of the remaining entries with:

```
% cut -d':' -f1 passwd_uniq | sort | uniq -c | \
egrep -v "\s*1"
```

If this produces any output, you have two different entries with the same account name. If the difference is not in the UID or GID field, simply decide on one of the entries and remove the other one. If the difference is the UID or GID field, you need to resolve this conflict, which can be rather complex.

Another consistency check is to see whether any two different accounts have the same UID, which is the case if this command:

```
% cut -d':' -f3 passwd_uniq | sort | uniq -c | \
egrep -v "\s*1"
```

produces any output; the second number in the output is the duplicate UID. Resolving this conflict again can be rather complex. Do the same kind of merging and checking for all your /etc/group files.

Copy the resulting files to /etc/NIS/passwd and /etc/NIS/group. I will return to the netgroup file later. Leave it out for now.

Now, start your master NIS server with:

```
# service ypserv start
```

Initialize the NIS maps with the command:

```
# /usr/lib/yp/ypinit -m
```

and follow the printed instructions.

In order to have all the NIS maps available to your NIS master server, you probably want to set up this machine as an NIS client as well. Make sure this NIS client can bind only to the NIS master as server in order to prevent circular dependencies when booting all your machines, as after a power outage.

Slave Server Configuration

NIS slave servers are NIS clients that redistribute the maps they receive from the NIS master server to other NIS clients. Make sure the newest portmap, ypserv, ypbind and yp-tools RPMs are installed on all your slave server machines. The first step in configuring an NIS slave server is to configure it as an NIS client. See the next section for how to do this.

Once the NIS client is configured, start it with:

```
# service ypbind start
```

On your NIS master server, add the name of the new NIS slave server to the file `/var/yp/ypservers` and run the following commands:

```
# cd /var/yp
# /usr/lib/yp/makedbm ypservers
/var/yp/nis.example.com/ypservers
```

You also need to change the definition of `NOPUSH` in the file `/etc/YP/Makefile` on your NIS master server from `true` to `false` in order to get updated NIS maps pushed from your master server to your slave server(s).

Back on your new NIS slave server, initialize the slave server with:

```
# /usr/lib/yp/ypinit -s nismaster
```

where `nismaster` is the name of your NIS master server. This needs to be the fully qualified domain name (FQDN) if your DNS returns the FQDN for a name lookup. Copy the file `/var/yp/securenets` from your NIS master server over to the new slave server, and start the new NIS slave server with:

```
# service ypserv start
```

Remember to update your disaster recovery plan to reflect the new dependency of your NIS slave server on your NIS master server.

Client Configuration

Install the latest `ypbind`, `yp-tools` and `portmap` RPMs on all your clients. Edit the file `/etc/yp.conf` to tell the client about your NIS server:

```
ypserver nismaster.example.com
```

Add a line for each of your slave servers as well, if you have some. Use a random order for these servers on your clients to get somewhat even load balancing over all available servers.

Add a line to `/etc/sysconfig/network` to define the NIS domain of the client:

```
NISDOMAIN=nis.example.com
```

and set the NIS domainname with the command:

```
# domainname nis.example.com
```

Start the portmapper with:

```
# service portmap start
```

and the NIS client with:

```
# service ypbind start
```

on each client.

The command `ypwhich` should now output the NIS server to which this client has bound.

Use the `ypcat` command to check the content of your NIS maps. For example:

```
% ypcat passwd
```

Next, you have to tell all lookups on your client to use NIS. This is done in the name service switch configuration file `/etc/nsswitch.conf(5)`. Change the `passwd`, `group` and `netgroup` entries to:

```
passwd:      compat
group:       files nis
netgroup:    nis
```

This defines the search order for group lookups: start with the local `/etc/group` file and then try an NIS lookup. Netgroups come only from NIS. I return to the `compat` entry for `passwd` later.

The name service caching daemon `nscd(8)` sometimes has problems updating its internal cache. The effect is that changes in an NIS map are not visible on a particular client. Restarting `nscd` on that machine is the only solution to this problem.

Typical Usages

Two commands you should be familiar with to query information from NIS are `ypcat(1)` and `ypmatch(1)`. `ypcat` prints values of all keys in an NIS map. The command `ypcat passwd` prints all entries in your NIS `passwd` map. `ypmatch` prints the values of one or more keys from an NIS map; `ypmatch jane passwd` outputs the `passwd` entry for account `jane`.

NIS Group Map

A typical use of the NIS group map is to allow file sharing between multiple users. This works with local files as well as with files in NFS. Here is how to set it up. Let's say you have two users (this technique works for any number of users) with the following `passwd` map entries:

```
jane:*:1234:42:Jane:/home/jane:/bin/bash
joe:*:5678:57:Joe:/home/joe:/bin/bash
```

This defines the *primary* group IDs for `jane` to be 42 and for `joe` to be 57.

With the NIS group map you can add additional, *secondary* group memberships for accounts. The group entry:

```
projectX:*:127:jane,joe
```

defines a new group `projectX` with no password (*), group ID 127 and two members. No comments are allowed in the group file.

If you now set up a directory with read/write/execute permissions for group `projectX`:

```
# mkdir /projects/X/
# chgrp projectX /projects/X/
# chmod g+wx /projects/X/
```

every member in the projectX group has permission to read/write/execute files inside that file space. The user might need to do a `newgrp projectX` first.

Whenever you need to add or remove accounts to or from the group map, do it on your NIS master server by editing the `/etc/NIS/group` file and executing the commands:

```
% cd /var/yp
% sudo make group
```

These generate a new group map that makes the changes visible instantaneously on all clients. There is no need to touch any client to make these changes. Everything now is centralized in one place on your NIS master server.

NIS Netgroups

Netgroups are very different from groups. Netgroups come in two flavors, user netgroups and host netgroups. Both types of netgroups can contain netgroups as members, so netgroup definitions can be hierarchical. Both types of netgroups are defined in the same netgroup file. Comments are allowed in the netgroup file.

Host netgroup definitions in `/etc/NIS/netgroup` look like this:

```
# Group of project groups:
projects \
    projectA \
    projectB \
    projectX

# Group of hosts for Project X
projectX \
    (host1.example.com, -, ) \
    (host2.example.com, -, ) \
    (host3.example.com, -, )
```

These host netgroup definitions now allow you to, for example, export NFS space only to subsets of your machines. In your NFS server's `/etc/exports` file, you can use constructs like these:

```
# export the /projects directory to all machines
# in the "projects" netgroup
/projects    @projects(rw,root_squash)

# export Project X' space only to machines
# in the "projectX" netgroup
/projects/X  @projectX(rw,root_squash)
```

Again, adding or removing hosts or adding/deleting netgroups is a simple edit of the `/etc/NIS/netgroup` file on your NIS master server. Execute `cd /var/`

yp; sudo make netgroup to update the NIS map, and the changes are visible everywhere instantly.

User Netgroups

User netgroups, which are netgroups with accounts as members, typically are used to restrict login to computers. User netgroup definitions look slightly different from host netgroup definitions:

```
# Group of project user groups
u-projects \
    u-projectA \
    u-projectB \
    u-projectX

# Group of users in Project X
u-projectX \
    (-,jane,) \
    (-,joe,) \
    (-,nick,)
```

The prefix u- in the names is a convention to distinguish user netgroups from host netgroups.

With these definitions in place, you now can grant or restrict login access to your computers with these kinds of entries in a machine's local /etc/passwd file. Remove a + at the very end of the passwd files if present:

- Allow access for all accounts in the u-projects netgroup and no one else:

```
+@u-projects
```

- Allow access for only the u-projectX netgroup members and no one else:

```
+@u-projectX
```

- Allow access to everybody in u-projects but not in u-projectX:

```
-@u-projectX
+@u-projects
```

Order here is important. The first match determines what happens.

- Allow everybody in u-projectA and also account nick

```
+@u-projectA
+nick
```

The information about nick (home directory, login shell and so on) comes out of the NIS passwd map. It is better to avoid putting explicit account names in here, because management of these entries is not centralized.

To make this +/- syntax work, your clients need to have the entry

```
passwd:          compat
```

in their `/etc/nsswitch.conf` files.

Conclusion

Once you are over the initial hurdle of installing an NIS server and making your authorization data consistent, you can start enjoying the centralization. Netgroups allow for complex and fine-grained access control from one central place.

Resources for this article: www.linuxjournal.com/article/7967.

Alf Wachsmann, PhD, has been at the Stanford Linear Accelerator Center (SLAC) since 1999. He is responsible for all areas of automated Linux installation, including farm nodes, servers and desktops. His work focuses on AFS support, migration to Kerberos 5, a user registry project and user consultants.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

Event-Driven Programming with Twisted and Python

Ken Kinder

Issue #131, March 2005

Before you turn your server app into a thundering herd of processes or a hairball of threads, consider this clean, logical event-driven way to do it. Download the 600-line proxy server example and follow along.

In the beginning, there were forking servers and then came threaded servers. Although they manage a few concurrent connections well, when network sessions reach into the hundreds or even thousands, forking and threading servers spawn too many separate, resource-consuming processes to be efficient. Today, there is a better way, asynchronous servers. A new breed of frameworks for third-generation languages is taming the once complex world of event-driven programming.

A rising star in the Python community has been Twisted, which makes asynchronous programming simple and elegant while providing a massive library of event-driven utility classes. In this article, I discuss asynchronous event-driven programming and how it's done in Twisted. Because reading about code only gets you so far, I cite examples from a real Twisted application developed for this article: a simple proxy server that blocks unwanted cookies, images and connections. Instructions on how to get the complete source code are in the on-line Resources.

What Is Twisted?

The Twisted Project has been gaining popularity as a powerful and increasingly stable way of implementing networked applications. At its core, Twisted is an asynchronous networking framework. But unlike other such frameworks, Twisted boasts a rich set of integrated libraries for handling common protocols and programming tasks, such as user authentication and even remote object brokering. One of the philosophies behind Twisted is breaking down traditional separations among toolkits, as the same server that serves Web content could resolve DNS lookups. Although the package itself is quite large, applications

need not import all the components of Twisted, so run-time overhead is kept to a minimum.

As with Python, Twisted's user base has been expanding from its academic roots to the commercial and government sectors. At Zoto, we're using Twisted in a distributed photo storage and management application, because it enables us to develop scalable network software quickly in a famously productive language, Python. Programming day to day, I appreciate Twisted for its impressive toolkit and supportive community. And as with all community-oriented open-source projects, Twisted is a safe business bet, because its existence doesn't hinge on the continued support of any single company or institution.

What Is Asynchronous Programming?

Have you ever been standing in the express lane of a grocery store, buying a single bottle of water, only to have the customer in front of you challenge the price of an item, causing you and everyone behind you to wait five minutes for the price to be verified? Plenty of explanations of asynchronous programming exist, but I think the best way to understand its benefits is to wait in line with an idle cashier. If the cashier were asynchronous, he or she would put the person in front of you on hold and conduct your transaction while waiting for the price check. Unfortunately, cashiers are seldom asynchronous. In the world of software, however, event-driven servers make the best use of available resources, because there are no threads holding up valuable memory waiting for traffic on a socket. Following the grocery store metaphor, a threaded server solves the problem of long lines by adding more cashiers, while an asynchronous model lets each cashier help more than one customer at a time.

This isn't to say there aren't benefits to a threaded model. For instance, with microthreads, the amount of resources used by any particular thread is reduced substantially. There's an inherent complexity in asynchronous programming, especially when you need to do many blocking operations in succession. In Python, however, the benefits of threading are diminished by Python's Global Interpreter Lock (GIL). Threaded programming in Python is refreshingly simple, because all internal Python operations are thread-safe. To add an item to a list or set a dictionary key, no locks are required, so as to avoid race conditions among threads. Unfortunately, this is implemented through an interpreter-wide lock that Python's interpreter uses liberally. So, although two threads safely can append to the same list at the same time, if they're appending to two different lists, the same lock is used. Because threaded Python applications suffer a resulting performance hit, asynchronous single-thread programming is all the more desirable for a language such as Python.

Accepting Connections and Sending Responses

Let's start with a simple example of a server that accepts connections on port 1100. For each connection, it sends the UNIX time and closes the socket.

Listing 1. This simple Twisted server sends the time and then closes the socket.

```
import time
from twisted.internet import protocol, reactor

class TimeProtocol(protocol.Protocol):
    def connectionMade(self):
        self.transport.write(
            'Hello. The time is %s' % time.time())
        self.transportloseConnection()

class TimeFactory(protocol.ServerFactory):
    protocol = TimeProtocol

reactor.listenTCP(1100, TimeFactory())
reactor.run()
```

Addressing the complexity of handling multiple sessions with one thread is at the core of a framework such as Twisted. Network sessions are represented by subclasses of the `twisted.internet.protocol.Protocol` class, such that each Protocol instance represents a network session. These objects are spawned by Factory objects, which inherit from `twisted.internet.protocol.Factory`. A singleton, `twisted.internet.reactor`, handles the dirty work of polling sockets and invoking events. Calling `reactor.run()` in Twisted simply starts the event loop, and `run()` exits when the application finishes, the same as an event loop in GTK or Qt.

The Proxy Server Example

Our proxy server has two kinds of networked chat sessions: incoming HTTP requests and their respective outgoing proxies. Because HTTP is a chat-like protocol, we can inherit our protocol class from Twisted's `LineReceiver`, which subclasses `Protocol` while providing extra functionality useful for chat sessions, such as HTTP. Twisted actually includes classes specifically for making and handling HTTP requests. We are writing our own in part because Twisted's prefab classes don't facilitate proxy serving and also because it's a good programming exercise for this article.

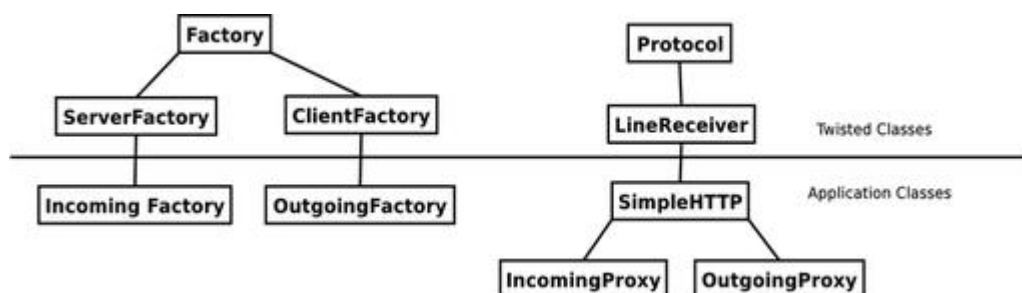


Figure 1. Class diagram for a proxy server. The Protocol classes handle individual connections while the Factory classes create them.

Refer to Figure 1 for the class structure we are going to use. Instances of the Factory classes are used by Twisted to spawn off Protocol instances for each connection made. We create one SimpleHTTP class and inherit from it classes for managing incoming and outgoing traffic. Because HTTP is mostly the same for client and server, we can manage most of the lexical processing in one superclass and let subclasses do the rest, which is exactly how Twisted's own HTTP classes work.

Handling Callbacks

Operations you'd otherwise do with one or two methods tend to require several callback methods in event-driven programming. The rule of thumb is, any time there's a blocking operation you need to wait on, it happens outside your code and, therefore, between two of your methods. In the case of our proxy server, we can break down into separate chunks each part of handling a request. Most of what a proxy server does amounts to reading in data from a browser, making a few changes to that data and sending the modified data to the remote Web server. As of HTTP/1.1, multiple Web hits can be handled over one network connection. In Figure 2, you can see what happens to each request, keeping in mind that multiple requests can be made per HTTP connection. Arrows connecting boxes show which events are spawned and in what order.

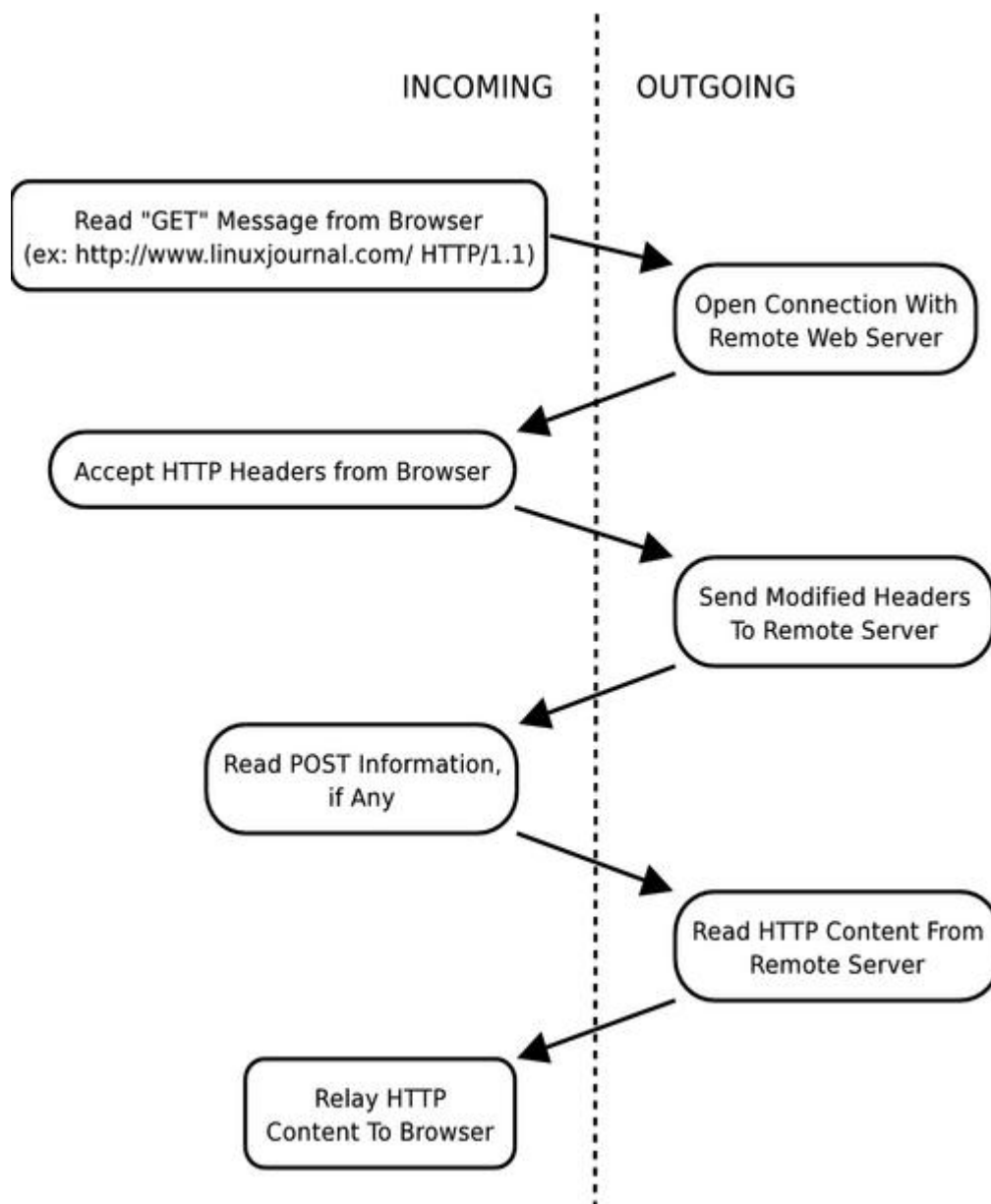


Figure 2. Overall Steps in Processing Proxy Hits

In a blocking program, one might expect to handle opening a remote connection and sending it a line of text like this:

```

connection = socket.open(remote_server, remote_port)
connection.write(get_string)
response = connection.readline()
  
```

We've all seen this kind of blocking code before, so what is different about the Twisted way? Because we don't want to wait around for the connection to be made in an event-driven program, we simply schedule some code to run when the remote server gets back to us. In Twisted, this kind of deferment is handled by using an instance of the `twisted.internet.defer.Deferred` class as a placeholder for the result you would expect from a blocking operation. For example, in our proxy server, we accept a `Deferred` object when we initiate a remote connection (Listing 2).

Listing 2. Deferring operations in Twisted are like putting them on hold until a blocking operation gets back to us.

```
d = self.outgoing_proxy_cache.getOutgoing(
    host, int(port))
d.addCallback(self.outgoingConnectionMade, uri)
d.addErrback(self.outgoingProxyError, uri)
```

The `self.outgoing_proxy_cache.getOutgoing` method initiates an outbound proxy connection. It doesn't wait, however, for the connection to be made to return to the caller; it returns immediately. The behavior of all methods to return as soon as possible is what makes a single-threaded server possible. Any and all CPU time taken by a method is spent processing, not waiting for external things to happen.

Notice how as a replacement for the connection object itself, a Deferred object is returned. By calling `addCallback` and `addErrback` on the Deferred object, we are scheduling future events to be fired, such that when an outbound connection is ready, the `self.outgoingConnectionMade` method is called. By passing `uri` as a second argument to `addCallback`, we are telling Twisted that `self.outgoingConnectionMade` also should be called, with `uri` as an additional argument.

Handling Errors

In the event of an error, `self.outgoingProxyError` is called with a Failure object, which brings us to error handling. Python's traditional error handling is done through exceptions, a concept familiar to other high-level languages, such as Java (Listing 3).

Listing 3. Traditional Error Handling in Python

```
try:
    (offending code)
except ValueError:
    (error handling code)
except MyError:
    (error handling code)
```

Although Python's model of exception handling works exceptionally well (pun intended) for synchronous designs, it does not take into account asynchronous design. For example, when we initiate an outbound HTTP connection, Twisted continues processing other events while the connection is made. But, we want to specify behavior to address any problems that may occur at the time we request the connection. Fortunately, the good people making Twisted took this into account. Just as code is scheduled to run when a blocking operation completes successfully, it also can be scheduled to run in case of an error.

Twisted also handles all exceptions raised within the event loop, with hooks for developers to manage and log exceptions. This has an added benefit too: although an exception might abort a specific event from completing, it does not bring down the server, even if you haven't put any exception-handling code in place.

Twisted Classes and Event Handling

When using some of the Twisted classes, such as the `LineReceiver` class we're using, you can handle many events simply by adding methods with the correct names to your classes. Each time the protocol receives a line, the `lineReceived` method is invoked with the text of the line as an argument. Our `SimpleHTTP` class, which is intended to do minimal processing of an HTTP session, has methods such as these:

- `startNewRequest`: invoked at the beginning of each request.
- `lineReceived`: designed to facilitate chat-oriented protocols. Each time a line of text comes over the socket, this method automatically is called.
- `rawDataReceived`: when sending a binary file or raw streams of data, it isn't reasonable to process information separated by newline characters. To account for this, `LineReceiver` lets us switch to raw mode transfer, in which case `rawDataReceived` is called instead of `lineReceived`.
- `handleFirstLine`: HTTP works by starting each request with a single line. Generally, the client is sending a GET or POST request with a URI, and the server responds with a status code. `handleFirstLine` is used to handle either of these cases.
- `handleHeadersFinished`: invoked when HTTP headers are sent fully.
- `handleRequestFinished`: invoked when the HTTP request itself has completed.

Writing separate methods for states or actions that occur in the processing of a protocol is how Twisted programmers queue up events. At the beginning of a request, we can specify events to occur at each stage of handling a request. In our earlier example, we decided to call `self.outgoingConnectionMade` once a connection has been made. Let's take a look at that method, as shown in Listing 4.

Listing 4. Scheduling Events in Twisted

```
def outgoingConnectionMade(self, outgoing_proxy,
                           uri):
    """
    This occurs when our outbound proxy has
    connected. It's a Twisted callback method.
    """
    assert(outgoing_proxy, OutgoingProxy)
```

```

self.outgoing_proxy = outgoing_proxy
outgoing_proxy.incoming_proxy = self

# Send HTTP command and echo back result
outgoing_proxy.write('%s %s %s' % \
    (self.http_command,
     uri,
     self.http_version) \
    + self.delimiter)

outgoing_proxy.firstline_sent_def.addCallback(
    self.outgoingFirstlineReceived)

# Send anything we have queued.
self.flushOutgoingBuffer()

# Add callbacks for when headers are ready
outgoing_proxy.headers_finished_def.addCallback(
    self.outgoingHeadersReceived)
outgoing_proxy.request_finished_def.addCallback(
    self.handleOutgoingRequestFinished)

```

Notice that `outgoing_proxy` represents the connection we are making to a remote server, on behalf of the Web browser we are serving. We're sending the HTTP request by calling `outgoing_proxy.write`. We're also scheduling the `self.outgoingFirstlineReceived` method to be called when a response is received from the remote server. The `self.outgoingHeadersReceived` method is called when the remote server has sent back all of its HTTP headers. Finally, `self.handleOutgoingRequestFinished` is called when the remote server has finished entirely responding to our outgoing HTTP request.

Although the `outgoingConnectionMade` method returns before any of this happens, we're lining up events to happen in the future. It well may be that while waiting for a response on one connection, ten other requests are opened and closed—all in the same thread. All information relevant to a connection is stored as instance data on protocol classes. Factories spawn protocol instances, protocol instances keep session states and deferred objects bind future data to event handlers. Completing the puzzle, the reactor manages the dirty work of polling sockets. This is the combination of tools upon which Twisted is built.

Wrap-Up

You can download for tinkering all 606 lines of the proxy server discussed in this article. Although I wouldn't put the company intranet behind it, I've been using it for a week now to filter out unwanted cookies and images and even to block access to a certain vendor from my desktop. When I started using Twisted, it was easy to wrap my head around the concept of asynchronous programming, a little harder to figure out how to map events to the flow I wanted and harder still to explain it to someone else. Do not be discouraged, however. Although we at Zoto started with almost no Twisted knowledge, we've built a fully functional and extremely scalable clustered application to store and manage on-line photos in less than a year, with only one person (me) working full-time on the server.

Of course, Twisted is not for everyone. Its vastness, although powerful, can be intimidating. For a simple asynchronous chat server in Python, take a look at Medusa. Like Twisted, Medusa organizes asynchronous programming into Factories (called Dispatchers) and chatting classes.

Resources for this article: www.linuxjournal.com/article/7963.

Ken Kinder is currently developing a clustered Twisted server for Zoto in Oklahoma City, Oklahoma. He enjoys hiking, skiing, photography and (of course) Linux. His hometown is Boulder, Colorado.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Motion: Your Eye in the Sky for Computer Room

Surveillance

Phil Hollenback

Issue #131, March 2005

Your door might be nice, but do you really need 23 hours of video of it standing closed? Use this software to process your security videos to include only the key events, so you can catch entries and exits.

Let's say you have a room full of thousands of dollars' worth of computer equipment. That's probably something you want to keep an eye on, right? With that in mind, you install a network-connected camera. Now, you can surf over to the camera's Web page and see what's going on in the server room at any time of day or night. That's an improvement, but you quickly realize some sort of recording facility is needed, in case you need to figure out who was in the room last Tuesday. So, you start saving the video to another system on the network for possible viewing at a later time. Maybe you write a few scripts to rotate the video after a week or so to keep from filling up your hard drive.

After wading through hours of video to find out who "borrowed" your favorite screwdriver, you realize further refinements are necessary. Wouldn't it be great if the computer could keep only the interesting video and throw out everything else? Enter Motion, a free motion-detection program. Process your video through it, and 24 hours of daily video becomes 15 minutes of video clips documenting every time something moved in that room—technology to the rescue.

The Hardware

Motion works with either standalone netcams, such as those offered by Axis (see the on-line Resources), or any camera connected to a video4linux-compatible video capture card. I concentrate here on using a standalone camera, the Axis 2100, because it's simpler to set up. In any case, you need a Linux system to save the video and to run Motion as well. Motion can require

quite a bit of processing power, but a system with a Pentium III CPU or higher should work okay if the machine is dedicated to running only Motion.

Installation and configuration of the Axis camera is straightforward. Pick a location for it in the room you want to monitor, and run power and Ethernet cables. In my experience, a camera mounted slightly above eye level, seven feet up or so, in a corner of the room provides the best coverage. Follow the camera install instructions to assign it an IP address on your network. Then, verify that the camera works by pointing your Web browser at the camera's Web page.

The computer system that is going to save the video and run Motion can be situated anywhere you like. It's probably best to keep it on the same logical and physical network as the camera, for simplicity's sake.

The Software

Any modern Linux distribution should work fine. I use Fedora Core 1 in my setup.

Obtain Motion from the Motion Web site (see Resources). The current version at the time of this writing is 3.1.16. You can use either the RPM supplied on the Motion Web site or build from source. I don't recommend using RPMs or Debian packages from elsewhere as they tend to be out of date and lacking features. Numerous important changes have occurred in Motion development in a few months' time.

The only other software dependency is the ffmpeg library, which Motion uses to generate MPEG videos. You must use the released version 0.4.8 of ffmpeg, as newer development versions do not work well with Motion. Download ffmpeg source (see Resources); you must have ffmpeg built and installed before building Motion. Otherwise, Motion attempts to use an older tool called mpegplayer to create videos. You probably don't have that installed either, so Motion won't work very well.

Building the Software

After you have downloaded both Motion and ffmpeg, untar them in a directory such as /tmp. Then, cd to the ffmpeg source directory and run:

```
$ ./configure
$ make
# make install
```

The last command must be run as root.

These commands install the ffmpeg libraries under /usr/local/lib. Then, cd to the Motion source directory and again run ./configure. This time, make sure to check the results. In particular, under Configure Status, FFMpeg Support must say Yes. If not, Motion didn't find the ffmpeg library on your system. This is the number one cause of problems and confusion when installing Motion. Don't continue until you resolve this problem. Figure out where on your system the file libavcodec-0.4.8.so is located, and rerun configure in the Motion directory as follows:

```
$ ./configure --with-ffmpeg=/some/random/path
```

Once you are able to run configure and see it report FFMpeg Support : Yes, you can build and install motion:

```
$ make  
# make install
```

Again, the final command must be run as root. After all of this completes, you will have a /usr/local/bin/motion executable on your system.

Refer to the Motion Guide (see Resources) if you encounter any problems building or installing Motion. Some of the guide is outdated, but it contains a useful explanation of how to install and operate Motion.

Configuring Motion

Motion runs as a daemon, constantly analyzing and storing video. It is controlled by a configuration file, per the standard UNIX paradigm. Copy the file motion-dist.conf from the source directory to /etc/motion.conf, and edit a few parameters. The first thing you need to change is the netcam_url setting. Motion retrieves JPEG images from the camera through this URL. For the Axis 2100 camera, this takes the form http://netcam.example.com/axis-cgi/jpg/image.cgi?resolution=640x480. When you set the netcam_url variable in motion.conf, all the settings pertaining to directly connected video cameras, such as video device, rotate, height and width, are ignored.

You should be aware of one limitation between netcams and standard video capture devices. Motion at this time knows how to request images from netcams only one JPEG snapshot at a time. The overhead of this limits your video to a maximum of 12–15 frames per second (fps). Some work has been done to pull the images from the cameras in motion-jpeg streams, but that effort is not yet complete. In practice, 10 or 12fps is perfectly adequate for surveilling a room.

You need to decide where to keep your Motion-generated videos. I generally use the directory `/var/log/vcr` on my Linux server. The location you use depends on your disk-space situation. Ideally, you should create a new filesystem exclusively for the Motion videos in order to avoid filling your root or `/var` filesystem with video files. This directory is set with the variable `target_dir` in `motion.conf`.

Next, decide on the type of video you want to create. Motion 3.1.16 supports MPEG1, MPEG4 and MS-MPEG4. MPEG1 has the advantage of being a simple and well supported format. However, MPEG4 offers better video and better compression. The final format, MS-MPEG4, is understood by Microsoft Windows Media Player without any special plugins.

One warning: MPEG4 and MS-MPEG4 support were introduced in Motion 3.1.16, so they have not been tested as extensively as MPEG1 video has been. I have found MS-MPEG4 to work fine, however, and it is much easier for Windows users to view. MPlayer or any other modern video player can be used to watch video in any format on Linux systems.

The video type is controlled by the `motion.conf` variable `ffmpeg_video_codec`.

This should be enough basic configuration for you to start using Motion. You should check that `output_normal` is off; otherwise, JPEG images of all the frames are stored in `target_dir`. This may be useful later on for debugging, but right now it is unnecessary clutter.

Starting Motion

Run Motion from the command line, as root, with the command `/usr/local/bin/motion`. Motion should start up and continue running. If it aborts immediately, there probably is an error in your config file. Follow the error messages to troubleshoot. Once you have it fixed so that Motion starts and continues to run, generate some input. Walk in front of the camera or, better yet, have an assistant do it. Remember to turn the lights on in your server room, or the camera might not pick up much action.

As the activity in front of the camera starts, Motion begins to generate an output file. After the activity stops, check your `target_dir` for the resulting output file. Examine the file with your video player. The video may appear jerky because of the limitations of pulling the still images from the netcam. Motion fills in the missing frames so that the video runs at normal speed, and it may have the stop-motion quality you see on convenience store cameras. If everything looks good, it's time to set up Motion to run on system startup.

To make Motion run on every system boot, set up an init script. On Red Hat-based systems, copy `motion.init` from the Motion source directory to `/etc/init.d/motion` and run, as root:

```
# /sbin/chkconfig --add motion
# /sbin/chkconfig motion on
```

Then, test that the initscript works by running it manually with `/etc/init.d/motion start`. Finally, if you are paranoid, reboot the system and verify that motion is up and running after system boot.

Tweaking Your Configuration

Like any good Linux program, Motion has many tuning variable. The best advice when you tune Motion is to change one variable, restart Motion and test. Some of the configuration variables can have non-obvious interactions with one another.

As a first step, you might want to turn on the `locate` and `text_changes` `motion.conf` variables. `locate` draws a box around the motion detected in each frame, and `text_changes` prints the number of changed pixels in each image in the corner of the image. These two settings allow you to determine where Motion thinks the motion in the image is, and how much motion there is—how many pixels have changed in the image.

Right off the bat, I recognized I probably placed my camera in the wrong spot in my server room. The room has a window that looks into another office space. It took me a while to figure out why I was getting so many tiny Motion movies when the only change would be a slight brightening and dimming of the room. I finally realized that occasionally a light-colored door in the other room would open and reflect light through the window into my server room. Then, that light would reflect off a shiny metal air-conditioning unit into the camera. So even though the camera couldn't see the window at all, light bouncing through it would produce occasional spurious results.

In retrospect, I should have mounted the camera to point away from possible external light sources and away from shiny metal surfaces. However, I decided to leave it where it was, because that really was the best view of what was going on in the room. Instead of moving the camera, I adjusted Motion to compensate.

The first thing I did was create a mask file. This simply is a black-and-white image the same size as the camera output images, 640×480 for the Axis camera. Any black areas are ignored by Motion. I created this file in The GIMP and blacked out the area corresponding to the metal surfaces of the A/C unit.

Unfortunately, Motion is picky about this file; you must save it as a raw, not ASCII, portable graymap (PGM) file.

Motion doesn't like PGM files because they are generated by The GIMP. If you use one, Motion starts but then exits a few moments later with the message:

```
This is not a ppm file, starts with 'P6'
```

A few minutes of source code digging revealed the fix. Motion expects the PGM file version number at the start of the file to be P5, not P6. Edit your mask file and change the magic number at the start from P6 to P5. You can edit this file safely in vi. After that change, Motion loads the mask file without incident.

This reduced, but did not eliminate, the empty motion capture videos. I then moved on to other adjustments. I tried turning the light switch variable, which the comments in motion.conf indicate might help filter out sudden light changes. I found this to be ineffective. I also experimented with lowering the threshold, the number of changed pixels required to trigger motion detection. The text_changes output is useful for this as it prints the number of changed pixels on each motion output frame. If too many bogus movies are output by Motion, you can try to raise the threshold to a number higher than what's printed by text_changes.

Ultimately, the best tweak I found was to increase motion_minimum_frames. This is the number of frames that must contain motion before Motion starts generating a movie. I set this variable to three and found that most of my spurious movies from the light changing disappeared. Most of those movies were only a few frames long, because the light level change happened quickly. Conversely, real motion-capture events tended to be many frames long. Thus, if you see many tiny movies with a duration of one second or so, my advice is to increase motion_minimum_frames to at least three and possibly more.

Future Improvements

One non-software tweak I have considered but not yet implemented is a motion sensor for the light in my server room. This neatly solves the problem of making sure there is enough light in the room when Motion records an event. Something moves in the room, the lights come on and Motion records. Motion-sensitive light switches can be found at hardware stores for around \$15 and require only basic wiring skills.

For now, I simply let my storage area /var/log/vcr fill with movies and delete them manually on occasion. It probably makes sense to set up an automated mechanism to handle this. My current thinking is that movies should be deleted after 30 days. Obviously, this depends on your particular needs.

Several experimental mjpeg support patches have appeared on the mailing list recently. As I mentioned earlier, mjpeg means that Motion pulls a continuous stream of images off the camera instead of requesting them one by one. This should provide much smoother resulting videos, although current Motion videos from netcams do have an enjoyable Keystone Kops feel to them.

Active development continues on Motion. The mailing list (see Resources) is an excellent place to ask questions and find out about current development. Most of what I've learned about Motion has come from reading the mailing-list archives.

Conclusion

Motion provides a solution for one of the most vexing problems we face in the computer industry, too much data. What good is information such as video imagery if there's more of it than you ever could watch? With a little bit of image analysis, Motion quickly eliminates the boring, unchanging video you don't care about. The results are more effective server room monitoring and more time for you to work on other projects.

Resources for this article: www.linuxjournal.com/article/7966.

Phil Hollenback is a Linux system administrator at Telemetry Investments in New York City. He spends his time skateboarding the streets of Manhattan when he's not writing Perl scripts. Visit him at his Web site, www.hollenback.net.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

The Perl Debugger

Daniel Allen

Issue #131, March 2005

Sticking in extra print statements is one way to debug your Perl code, but a full-featured debugger can give you more information.

Debugging is an annoying necessity in any language, whether it's debugging your own code or somebody else's that you've been given to make work on your system. Anything you can do to make debugging easier is a big win. Perl includes a command-line debugger that can make your debugging job considerably easier. This article covers the basics of the debugger and shows off a few tricks you may find useful.

Avoiding Bugs with Warnings and Strict

An astounding number of bugs can be caught by Perl automatically by turning on warnings and strict at the beginning of your program. If your program includes the line `use warnings;` you can catch dozens of common errors, including variables used only once, which often are typos; scalar variables used before they are set; and redefined subroutines.

These diagnostic messages can be explained further by including the line `use diagnostics;`, which prints an explanation for each warning. Or, you can look up the explanations using `man perldiag`.

If your Perl version is older than 5.6, instead of `use warnings;` you have to use the `-w` option on the first line of the script, like this: `#!/usr/bin/perl -w`.

Finally, you can catch additional common errors with `use strict;`, which in effect, forbids a few unsafe programming shortcuts. The rules that `use strict` turns on are as follows:

- Variables must be declared before use, with `my` or `our`, or imported with `use vars` or fully qualified with a package name.

- Bare words must be subroutines, not strings, such as `$string = blah;`.
- References cannot be symbolic; see sidebar.

As you can see, warnings and strict tighten up a few of Perl's features that can be used for good but also can be abused. These commands make debugging easier, because Perl catches these bugs for you.

Symbolic References

Symbolic references are different from regular hard references, where a variable refers to another variable. Symbolic references are created when the programmer uses a string as a reference. For example, this normally is valid Perl code:

```
$name = "username";  
$$name = "da";      # sets $username
```

This code easily can cause a case of the interpreter doing what you said, not what you meant. It is easy to put a symbolic reference where a hard reference was intended or to confuse the generated variable name because it never appears in the code. A much safer way to accomplish the same thing is to use a hash to store such variables and to turn on strict variable checking with `use strict`.

What's Wrong with Print Statements?

You might ask at this point, what's wrong with debugging by scattering print statements in your code? Nothing is wrong with this debugging technique, but you have more power with the interactive debugger. You can examine all aspects of the program and environment, not only those you thought of when you ran the program, and you can see more clearly what the program actually does. Hopefully, by the end of this article you will agree that investing a little effort in learning the debugger pays off in saved time.

Starting the Debugger

The debugger is run from the command line by passing Perl the `-d` option:

```
perl -d filename.pl
```

If you are debugging a CGI program written with `CGI.pm`, simply call it on the command line with the arguments you'd like to pass, along with `-d`:

```
perl -d filename.pl param=value param2=value
```

Instead of using the command line, you could use the Perl debugger as part of certain IDEs, such as GNU Emacs or Activestate Komodo, or from debugger GUI front ends, such as ddd or ptbdb. For space reasons, I discuss only the command line in this article, but the principles hold for a GUI debugger as well.

If you're using the command-line debugger, it is useful to have the `Term::ReadLine` module installed, which enables cursoring through the command history.

Here's an example program we use in this article. Copy the following to a file called `sample.pl`:

```
#!/usr/bin/perl

use warnings;
use strict;

my $name = "Pengu";

foreach (1..20) {
    &shout($name);
}

sub shout {
    my $name = shift;
    print "**** $name ****\n";
}
```

Essential Debugger Commands

The following seven commands are sufficient for basic debugging:

- `s`: single-step execute the next line, stepping into subroutines.
- `n`: single-step execute the next line, stepping over subroutines.
- `r`: nonstop execute until the return from the current subroutine.
- `c <line-number>`: nonstop execute until a particular line.
- `l <line-number, range or subroutine>`: list source code.
- `x <expression>`: evaluate and pretty-print `<expression>`.
- `q`: quit debugger.

To try these out, run the test program with the debugger:

```
perl -d sample.pl
```

You should see debugger startup information:

```
Default die handler restored.
Loading DB routines from perl5db.pl version 1.07
Editor support available.
Enter h or h h for help or
man perldebug for more help:
```

```
main::(sample.pl:6):   my $name = "Pengu";  
DB<1>
```

This is the state before the program starts running. The next-to-last line has useful information about the debugging status: you're in the main package, file sample.pl line 6, and it displays the line that is about to be run.

The last line is a prompt with the command number (incrementing as you enter more commands) and angle brackets, where the number of angle brackets signifies nested commands. You don't need to worry about those here.

Type `s` at the prompt and press Enter to single-step one line into the program:

```
DB<1> s  
main::(sample.pl:8):   foreach (1..20) {  
DB<1>
```

To repeat the command, press Enter; repeat this as long as you like to be convinced that the program is stepping through its paces. Every time you pass the print statement, it is echoed to the screen, interspaced with the debugging materials.

Now, try the command to step over subroutines (`n`), and press Enter a few times. You go through the loop and receive your subroutine results right away, without stepping through each command in the subroutine.

Next, try the command to return from the current subroutine (`r`). But wait—if you do it now, it will run until the program finishes, because you're “returning” from the main program. First, do a couple repetitions of `s` to step into the subroutine. Then, with an `r`, you should see something like:

```
DB<1> s  
main::(sample.pl:8):   foreach (1..20) {  
DB<1>  
main::(sample.pl:9):   &shout($name);  
DB<1>  
main::shout(sample.pl:13): my $name = shift;  
DB<1> r  
*** Pengu ***  
void context return from main::shout  
main::(sample.pl:8):   foreach (1..20) {  
DB<1>
```

Notice the `void context return from main::shout` line. If we had asked for a return value in the main loop, we would see it displayed here. In Perl, functions and subroutines can return different values based on the context of the caller (scalar, array or void). A nice feature of the Perl debugger is

the `r` command, which tells you what context was requested by the caller. It can find the bug if you ask your subroutine for a scalar, but you mistakenly have the subroutine return an array.

Next, we have the `l` command. Try it now:

```
DB<1> l
8==>  foreach (1..20) {
9:      &shout($name);
10     }
11
12     sub shout {
13:         my $name = shift;
14:         print "*** $name ***\n";
15     }
DB<1>
```

Alone, `l` lists a page of the source code, starting at the next line to be executed, with a text arrow pointing to the next line. You also can list a range by specifying the line numbers, such as `l 200-230`. Additionally, you can list a subroutine by naming it: `l shout`.

The `c` command continues execution until you hit a particular line number, so you can jump ahead to a particular piece of code that is interesting:

```
DB<1> c 14
main::shout(sample.pl:14): print "*** $name ***\n";
DB<1>
```

You can execute any Perl expression, including code that changes the running program, by typing it at the prompt. This can include setting variables in the program by hand.

The `x` command evaluates and pretty-prints any expression, prepending a numbered index on each line of output, dereferencing anything that can be dereferenced and indenting each new level of dereferencing. As an example, below we set an array, `@sample`, and then display it:

```
DB<1> @sample = (1..5)

DB<2> x @sample
0  1
1  2
2  3
3  4
4  5
DB<3>
```

Notice that hashes are displayed with keys and values, each one on a line. You can display hashes properly by preceding the hash with a \, which turns the hash into a hash reference, which is properly dereferenced. This looks like:

```
DB<4> %sample = (1 .. 8)
DB<5> x \%sample
0 HASH(0x83d53bc)
  1 => 2
  3 => 4
  5 => 6
  7 => 8
DB<6>
```

When you are satisfied with the results, quit the debugging exercise with q.

Four More Debugger Commands

Many people use the Perl debugger with no more than these commands. Once you are comfortable with those, however, an additional four commands can make your debugging more efficient, especially for programs that use object-oriented code:

- /<pattern>: lists source code at next regular expression match.
- ?<pattern>: lists source code at previous regular expression match.
- S: lists all subroutines and methods available to the program.
- m <object or package>: lists all methods available on the given object or package.

You can search and display code that matches a string or regular expression with / for forward searches and ? for backward searches. There should be no space before the string you're looking for:

```
DB<6> /name
6:      my $name = "Pengu";
```

The S and m commands are useful for exploring what subroutines or methods are available: S lists every subroutine and method available to the program. These are in reverse order of when they were loaded by use or require, and they include routines loaded from the debugger, such as Term::ReadLine. The m command lists every method available to an object or by way of a package. Here is a sample:

```
DB<7> use CGI
DB<8> $q = new CGI
DB<9> m $q
```



```
AUTOLOAD
DESTROY
XHTML_DTD
_compile
_make_tag_func
_reset_globals
_setup_symbols
add_parameter
all_parameters
[...]
```

Actions, Breakpoints and Watchpoints

Actions, breakpoints and watchpoints provide even more control over the debugger and the running program. You may prefer using them from a graphical Perl debugging front end, such as ddd, ptkdb or Activestate Komodo. The most common complaint about the Perl debugger is remembering the proper command-line shortcut for each command, and these commands add still more shortcuts to remember.

Additionally, in Perl 5.8 some of the keyboard commands have changed to make them more internally consistent. Often, though, people need to use both 5.8 and an earlier version, so it may be easier to use a GUI. I describe the commands from the command line below; the principles remain the same.

An action is used to wedge code into your program without modifying the source file. It can be useful when the code is in production and you want to test a change. It's also useful if you're in the middle of a debugging run and want to change code without restarting the debugging session from scratch.

You set an action like so, a `<line-number> <code>`. An example could be:

```
DB<10> a 9 $index = $_;
```

This adds a new command inside the foreach loop that stores the index count, which is incremented each time through. If you list the program, you see an a next to the line number that has the action. The action is executed before the line to which it is attached. You can list actions you've set with L and delete an action by specifying a with the line number without a command. The previous is for Perl 5.6 and earlier; in Perl 5.8, delete an action with A and the line number of the action.

Breakpoints and watchpoints return control to the debugger from continuous execution, such as from r and c described above. They are useful for jumping ahead to the particular iteration of a loop that is having problems, without repeatedly stepping through the loop by hand.

A breakpoint is set on a line number or subroutine, with an optional condition that must be met. A breakpoint is set with `b` as shown here:

```
b shout
```

If you list the program, you can see a `b` next to the line number at the first line of the subroutine `shout`. Press `C` to continue execution, and it stops inside the subroutine.

If you followed the previous example and set the action on line 9, you could set a breakpoint to stop on a particular iteration of this loop:

```
b shout ($index eq 8)
```

This should give you an idea of the power of actions and breakpoints, if you imagine debugging a longer program with complex conditional statements and external data sources.

You can list breakpoints with `L` and delete one with `d` in Perl 5.6 and earlier. In Perl 5.8, you delete a breakpoint with `B`.

A watchpoint probably is better known as a watch expression. It halts the program as soon as a specified expression changes. In Perl 5.6, it is set with `W` as shown here:

```
W $name
```

You can list watchpoints with `L` and delete all of them by specifying no parameter to `W`. In Perl 5.8, add a watchpoint with `w` and delete it with `W`.

Customizing the Perl Debugger

The first thing to know is that the debugger is simply a Perl library that takes advantage of hooks in the Perl interpreter. You could replace the debugger completely, if you like, by copying the file somewhere and requiring the file in your code in a `BEGIN` loop:

```
cp /usr/lib/perl5/5.6.1/perl5db.pl ~/myperl5db.pl
```

And, place this line in your program:

```
BEGIN { require "~/myperl5db.pl" }
```

You might do this, for example, if you preferred the syntax and operation of the 5.6 version debugger over the 5.8 version.

You also can specify an alternative debugger with the `-d` command switch. Perl versions 5.6 onward include DProf, a profiler that uses debugger hooks. You can use it like this:

```
perl -d:DProf mycode.pl
```

You also can use the debugger hooks in your own programs. You can set a breakpoint directly in your code by setting the variable `$DB::single = 1;`, which is useful if you need to debug code in a BEGIN block. Otherwise, they are executed before the debugger gives you a prompt. Or, you could use the hooks to run particular code whenever any subroutine is run. To find out more about these and other hooks, check the `perldebug` man page.

The debugger has a set of internal variables, also described in the `perldebug` man page. To change these variables you can use a configuration file, `.perldb` in the current directory or in your home directory. This configuration file has Perl code that is run when the debugger starts. For example, you can add new commands of your own, like this:

```
$DB::alias{'quit'} = 's/^quit(\s*)/q/';
```

This allows you to quit the debugger by typing `quit` at the prompt. The `perldebug` man page describes a few similar aliases that might be useful.

A number of debugger options can be set inside the debugger with the `O` command. The only one I have used changes the pager:

```
O pager=|less
```

This way, any command that would print more than a screen of output can be sent through your favorite pager by using a pipe character before the command: `|L`.

Resources for this article: www.linuxjournal.com/article/7962.

Daniel Allen (da@coder.com) discovered UNIX courtesy of a 1,200-baud modem, a free local dial-up and a guest account at MIT, back when those things existed. He has been an enthusiastic Linux user since 1995. He is President of Prescient Code Solutions, a software consulting company in Kitchener, Ontario, and Ithaca, New York.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

The Oddmuse Wiki Engine

Brian Tanaka

Issue #131, March 2005

Wikis are fun and useful. Try one on for size by installing Alex Schroeder's Oddmuse.

In 1995, Ward Cunningham established the first wiki Web site, the WikiWikiWeb, for the Portland Pattern Repository Project. In the years since, a number of wiki engines have been developed in a variety of programming languages. One such wiki engine is Oddmuse, written in Perl by Alex Schroeder.

What Is a Wiki?

Although wikis have been around for nearly a decade, they appear to be enjoying a rise in popularity, and many people only now are encountering the wiki concept. Therefore, a brief overview of wikis is in order. If you already are familiar with wikis, you might want to skip to the next section.

The underlying concept of a wiki is simple: a wiki is a Web site that allows any user to add and edit pages by using nothing more than a Web browser. This simple arrangement is quite powerful. It enables arbitrarily large numbers of volunteer editors to contribute to collaboratively created Web sites. It also enables smaller groups or even lone individuals to create and organize information with great ease. Put another way, as Cunningham once said, “[A wiki is] the simplest on-line database that could possibly work.”

In my experience, I've found wikis to be useful tools that make tasks such as organizing projects and creating documentation about almost anything surprisingly less painful and time consuming than they often can be. Perhaps the best way to grasp what wikis are, how they work and the scale and quality of documents that can grow from them, is to explore a mature, established wiki. An excellent example is Wikipedia.

Wikipedia is an encyclopedia, exactly like a multivolume, printed-and-bound encyclopedia. Unlike a bound encyclopedia, however, the Wikipedia has an enormous "editorial staff" composed of any Internet user with a Web browser. In addition, the text of each article is hyperlinked where appropriate to other Wikipedia articles. Any visitor to the site can create and edit articles, additions and changes that become immediately accessible to other users. A revision control system allows changes to be backed out easily, guarding against mistaken or malicious edits.

As an example of how easy it is to contribute content to a wiki, let's say you are reading, on some wiki Web site, an article about rodents. You notice that, although it's a fine article, it fails to mention the world's largest living rodent, the capybara. To correct this omission, you click the Edit text of this page link, after which you are presented with a standard Web form that allows you to edit the text. You do so by adding a sentence about the capybara, and when you save your changes, the new version of the article instantly is available. It's that easy.

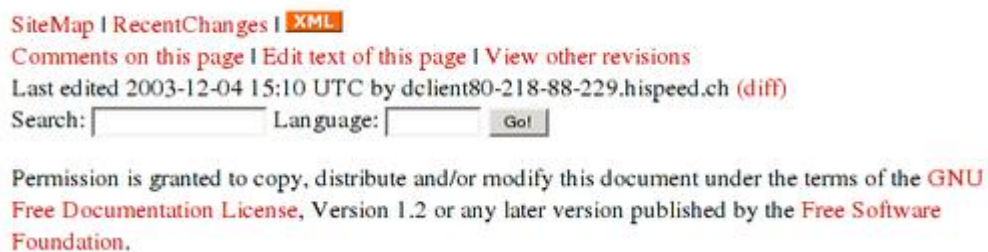


Figure 1. Typical Oddmuse page footer, with links to common wiki functions such as edit text of this page and search.



Editing What Is A Wiki

```
A wiki is a website that usually has the following two properties:

# Anybody can edit the pages of the wiki, and anybody can undo these edits
# It is easy to write new pages for the wiki, because it doesn't use HTML

(There are exception to both rules.)

References:

* WhatIsaWiki on Meatball Wiki [http://www.usemod.com/cgi-bin/mb.pl?WhatIsaWiki]
* "A wiki is the simplest online database that could possibly work." -- Ward
[http://wiki.org/wiki.cgi?WhatIsWiki]

== Why a Wiki? ==

With a wiki, creating and maintaing a website is trivial: You don't need to know HTML,
nor FTP, nor anything else. This is what people normally use for their websites.

A wiki is great if you want to enable other people to help and contribute.
The wiki just helps them to start contributing faster, since it is so easy to use.

== Wiki is also Groupware ==

A wiki is ideal for a small group of people: classes, friends, project teams, gaming
groups. It allows the group members to communicate with each other when you do not or
cannot meet each other face-to-face. A wiki also works for chat rooms. Often the logs
```

Summary:

This change is a minor edit.

Please make sure you contribute only your own work, or work licensed under the [GNU Free Documentation License](#).

Figure 2. Typical edit text of this page form. Not shown in this screenshot are the Preview and Save buttons.

Let's also say you made reference in the article to South America, because it's the habitat of the capybara, and you wanted the words South America to be a link to an article about South America. To create that link, you would use one of the simple wiki linking syntax rules—enclosing South America in double square brackets if the site uses Oddmuse—and the wiki engine creates the link to the South American article when it generates the page. If there is no South American article, the wiki engine then creates a link to a form that allows anyone to write one. In this way, article by article, the wiki grows and becomes increasingly useful.

Another example, and one that demonstrates the usefulness of wikis for small groups, is the wiki that my business partner and I use. We use it to record technical information, such as server configuration changes, project documentation, software installations, standard procedures for common tasks and so on. We also use it to track business data, such as client contact information.

The benefits have been abundantly obvious for us. Based on past experience, we've noticed that a much greater percentage of system administration and project documentation is recorded than if we were not using the wiki.

Furthermore, the information is searchable and easy to edit. And, because we have an RSS feed for article changes and additions, everyone involved can stay up to date easily.

Oddmuse Features

Oddmuse is a single Perl script. This makes basic installation straightforward and supports another of Schroeder's project goals: to keep Oddmuse simple and easy to use. The potential loss of flexibility is offset by Oddmuse's extensibility. As evidence, you can find a list of available modules on the project Web site.

Aside from the ordinary, requisite wiki features, such as automatic tracking of recent changes, powerful linking syntax, revision control and visual diffs of revisions, Oddmuse has a number of notable features and advantages. One of the first things that impressed me about Oddmuse was the quality of its documentation. Schroeder makes abundant documentation in an assortment of languages a priority, and it shows. Conveniently, he gets to use the tool he's documenting to write and maintain the documentation. A visit to the Oddmuse project Web site is both a demonstration of the engine in action and an encounter with ample information about how to set up, use and customize it.

[SiteMap](#) | [RecentChanges](#) | [XML](#)



Selber Hosten

Difference (from prior major revision)

Geändert: 12c12

< Keine Ahnung. Erfahrungsgemäss braucht das Abspeichern von längeren Oddmuse Seiten

stattdessen:

> * Keine Ahnung. Erfahrungsgemäss braucht das Abspeichern von längeren Oddmuse Seiten

Eingefügt: 13a14

> * Auf Deutsch? Gehts jetzt oder gehts nicht?

Figure 3. Oddmuse's visual diff makes understanding what has changed between revisions obvious and clear.



History of Comments on Migrating From UseMod

- Revision 9** 2004-05-11 22:52 UTC by RolfUnger from p213.54.133.246.tisdip.tiscali.de -- new comment
- Revision 8** 2004-05-09 02:04 UTC by Alex Schröder from 80-219-235-82.dclient.hispeed.ch -- Neuer Kommentar
- Revision 7** 2004-05-08 00:20 UTC by RolfUnger from p213.54.167.6.tisdip.tiscali.de -- new comment
- Revision 6** 2004-05-07 23:24 UTC by Alex Schröder from 80-219-234-239.dclient.hispeed.ch -- Looking at the source, it seems that the summary field should be extracted just fine.
- Revision 5** 2004-05-07 22:24 UTC by Alex Schröder from 80-219-234-239.dclient.hispeed.ch -- Neuer Kommentar
- Revision 4** 2004-05-07 20:28 UTC by RolfUnger from p508B3B97.dip.t-dialin.net -- new comment

Figure 4. Typical History page. Checkboxes allow the user to define which page revisions to diff.

As someone who strives to adhere to World Wide Web Consortium (W3C) standards, I was pleased to note that Oddmuse produces valid HTML 4.01 Transitional. On a related note, Oddmuse integrates nicely with Cascading Style Sheets (CSS). If you're interested in syndication by way of RSS, Oddmuse supports both an outbound RSS feed for your wiki and inbound RSS feed aggregation.

An obvious concern for any wiki is page vandalism. What, after all, is to prevent someone from editing a page and intentionally making a mess of it in one way or another? If you want to take full advantage of wikis' biggest strength—namely, that anyone can create and edit pages—simply leave the system open and revert defaced pages as they appear. An attentive group of editors can catch vandalism by monitoring the RecentChanges page.

[1 days](#) | [3 days](#) | [7 days](#) | [14 days](#) | [21 days](#) | [28 days](#)

[List all changes](#) | [Include minor changes](#)

[List later changes](#)

2004-05-11

- 22:52 UTC (diff) (history) [Comments on Migrating From UseMod](#) RolfUnger from p213.54.133.246.tisdip.tiscali.de -- new comment [en, de]
- 19:45 UTC (diff) (history) [Selber Hosten](#) pD9EAA0D0.dip.t-dialin.net [de]
- 10:15 UTC (diff) (history) [Migrare Da UseMod](#) host162-31.pool8172.interbusiness.it -- Migrating UseMod ITA

2004-05-10

- 22:12 UTC (diff) (history) [Portraits Support Extension](#) Alex Schröder from 80-219-227-150.dclient.hispeed.ch -- simplified the code a bit, and so now span.new is needed anymore. [en]
- 20:53 UTC (diff) (history) [Comments on URL Abbreviations](#) Alex Schröder from 80-219-227-150.dclient.hispeed.ch -- Neuer Kommentar [en]
- 14:06 UTC (diff) (history) [Script Multipli](#) host162-31.pool8172.interbusiness.it -- Multiple Script ITA

Figure 5. Typical RecentChanges Page

If, on the other hand, you prefer to forgo the advantages of open editing and would prefer to prevent vandalism in the first place, Oddmuse provides two methods for doing so. You can lock specific pages or the entire site and use passwords to grant page editing permission to certain individuals. Your other option is simply to ban problematic users.

Oddmuse uses flat files for data storage rather than a database such as MySQL or PostgreSQL. A good argument can be made for why this is a strength, and an equally good argument can be made for why this is a weakness. On the one hand, using flat files instead of a database means more simplicity and less administrative overhead. On the other hand, databases provide advantages when it comes to data storage and retrieval. If you feel that you must have a database behind your wiki, then Oddmuse is not for you. However, if your concern is speed, it's worth noting that searches are reasonably fast, for instance, on the Emacs Wiki, which uses Oddmuse and currently has 2,242 pages.

Installation

The installation section of the Oddmuse Web site has links to the latest version of the script and guides to installing and configuring it. Basic installation is absurdly simple. Once you've downloaded the script, simply place it in your CGI directory and make it executable. It should work immediately. If it doesn't, consult the troubleshooting guide on the Oddmuse Web site.

Even if you have no plans to customize your wiki, you should make one important change before you do anything else. Namely, you should change the

`$DataDir` variable in the script to the absolute path of the directory in which you would like Oddmuse to store its data. If you don't change this variable, Oddmuse uses `/tmp/oddmuse` as its data directory, and you could lose your data down the road.

Another step you should take, no matter how simple or complex you intend to make your wiki configuration, is setting the administrator and editor passwords. The variables to change are, respectively, `$AdminPass` and `$EditPass`. Because the passwords are stored in plain text, the security of your wiki is only as good as the security of your server in general. Nonetheless, you should pick strong passwords.

Although the basic installation provides a fully functional Oddmuse wiki and may be all you need, you might want to consider the more-advanced installation. I chose the more-advanced installation for the Oddmuse wikis I've set up recently, and I've found that the added flexibility is well worth the trivial amount of additional effort required during its setup.

First, I used the wrapper script strategy clearly explained on the Web site. This strategy eliminates the need to define the data directory pathname every time you upgrade the script. Second, I used an external configuration file, and in it I specified an external CSS file.

Configuration

Now that your Oddmuse wiki is up and running, you can customize it as much as you like. Some configuration is accomplished through the wiki itself, such as locking pages or the entire site if you don't want an open wiki.

In order to perform configuration changes and other administrative functions by way of the wiki, you need to authenticate yourself as an administrator. To do so, you must visit the password page. There's no link to the password page by default, so you must go to the following URL manually: `http://www.example.com/wiki/current.cgi?action=password`. You must substitute your own hostname and path, of course. Also, the Oddmuse CGI program in the above example URL is called `current.cgi` per the Oddmuse wrapper script installation instructions. Once you're authenticated, you can use the special menu items that appear in the footer of every page to make the configuration changes you desire.

Other changes are implemented by editing the script itself. A number of configuration variables can be changed to suit your preferences. If you elected to use an external configuration file, you can change configuration variables there instead.

The Web site provides detailed explanations of all configuration variables. Some examples of what they control include the name of the wiki, which stylesheet to use, the URL of your logo, what's displayed in the footer of every page and so on.

I'm glad Schroeder included CSS support in Oddmuse, because it makes altering the layout and appearance of a wiki much easier than it otherwise would have been. Once the wiki was up and running, I spent a few minutes experimenting with changes to the stylesheet until the wiki looked the way I wanted.

Beyond that, you can make significant additional changes to the wiki by using the available modules and extensions or even by writing your own. See the Oddmuse Web site for a full list of modules and extensions. If you want to, you even can make Oddmuse behave like a blogging system.

Bringing the Wiki to Life

Once you're satisfied with your wiki installation and configuration, you can begin creating pages. But before you do, I strongly recommend that you read the Text Formatting Rules section of the Oddmuse Web site. Once you know the rules, creating and editing pages is easy to do, but it's invaluable to know what you can accomplish by using those rules.

As founder of a new wiki, you need to know more than only the details of the technology. A successful wiki depends on the participation of contributors, and the social component needs to be cared for as much as any of the technical components. This complicates things significantly. After all, social communities don't come in tarballs with makefiles! You need to encourage and entice people to participate, and you need to nurture the community that grows around the wiki. This takes time and is hardly a precise science.

Fortunately, founders of other wikis have taken the time to write about their experiences. One excellent resource is the WikiLifeCycle page on the MeatballWiki (see the on-line Resources). In it you can learn about best practices for attracting contributors, choosing a name, establishing effective boundaries, defining the mission or goal of the wiki, shaping behavioral norms, preventing stagnation and so on. Now, armed with a well-configured wiki and a grasp of the nontechnical, social issues, the road to a thriving wiki lies open before you.

Resources for this article: www.linuxjournal.com/article/8010.

Brian Tanaka has been a UNIX system administrator since 1994 and has worked for companies such as The Well, SGI, Intuit and RealNetworks. He can be reached at btanaka@well.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

LaTeX Equations and Graphics in PHP

Titus Barik

Issue #131, March 2005

Make the difficulties of displaying mathematical equations on the Web a thing of the past by embedding LaTeX content in your pages.

It's safe to say that the world of Weblogs and wiki Web sites are here to stay. Although such systems are great for journals, general text posting and even photography, their limitations become apparent when working in environments that require the use of features more advanced than simple text entry and images. In particular, technical Weblogs need support for graphs, mathematical expressions, diagrams and more. Such functionality is difficult, if not impossible, to implement with HTML alone.

Using external applications such as dia, xfig and Microsoft Equation Editor is equally difficult, as the poster first must create the figure or mathematical equation and then upload an image representation to a Web site. Moreover, if other posters in a collaborative Weblog want to modify the figure, they also must possess the application as well as the original file that created the image. Obviously, this sort of system has its share of complications, and it fragments the overall quality of figures and equations for a site.

In this article, I demonstrate the use of LaTeX, a typesetting tool and language designed specifically for technical document preparation, from within PHP to address these demands. I call LaTeX from within PHP when HTML is not sufficient to address these complex needs and then render the result uniformly as a PNG image, a format all modern browsers support. Because the software is available entirely on the server, all posters and users have access to the same set of tools and packages for publication.

Why Not MathML?

According to the W3C, MathML is a low-level XML specification for describing mathematics. Although MathML is human-readable, in all but the simplest

cases, authors need to use equation editors or other utilities to generate XML code for them. Moreover, modern browsers support only a limited subset of the MathML language, and even then, many of these browsers require external plugins to support MathML. Although the future is quite promising for this language, as of now, it essentially is unsupported and unusable.

To complicate matters further, Leslie Lamport's LaTeX typesetting system has become the de facto standard for the production of technical and scientific documentation. Based on Donald Knuth's TeX document layout system from the early 1970s, LaTeX has been around since 1994 and is a mature and well-understood technical documentation preparation platform with a committed user base. That's not to say that learning LaTeX is a walk in the park. It certainly isn't, but as of now, MathML does not provide compelling evidence to warrant a transition from this already-established system.

Requirements

Following the UNIX philosophy to “write programs to work together”, I use a composition of common tools available for the Linux platform and chain them together to produce a PNG-equivalent rendering of the LaTeX source. Specifically, you need a recent version of LaTeX with dvips and the ImageMagick toolkit. You are going to use the convert utility from the ImageMagick tools to convert your result into a PNG image. Luckily, most hosting providers that provide shell access already have these utilities available.

Project Overview

The rendering system takes a string of text and extracts segments enclosed in [tex] and [/tex] pairs for future substitution. These extracted segments are called thunks. If a thunk previously has been processed, meaning an image representation of the thunk code already is available, the thunk is replaced with a URL to that image. If the thunk is new, it is passed to the LaTeX typesetter, which outputs its result as a DVI file. The DVI file then is converted to a PNG image with ImageMagick and placed into the cache directory. A URL of the newly created image is substituted for the thunk in the original text. When all thunks have been processed, the resulting text is returned to the caller. The process for converting a single thunk is illustrated in Figure 1.

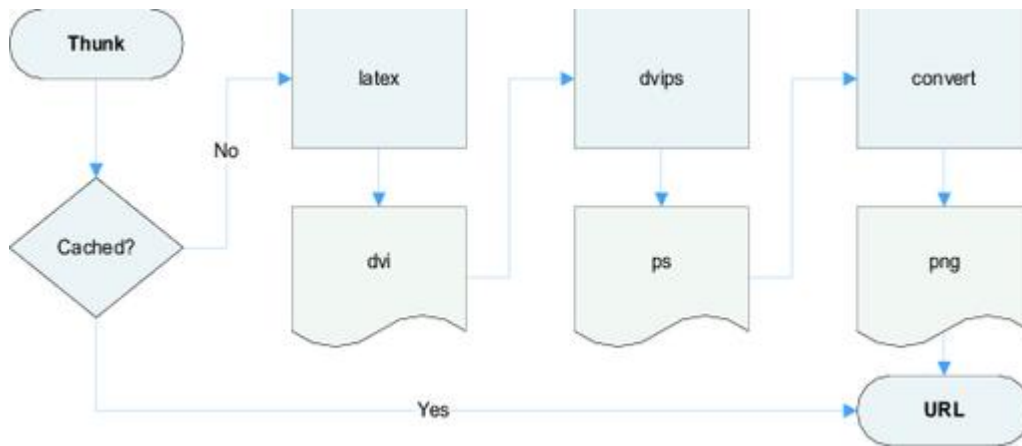


Figure 1. A Flowchart of the Rendering Process for a Single Thunk

Usage

I think it is best to start top-down and first look at how to invoke the rendering process, without discussing implementation specifications. The driver is simply an HTML front end that provides a mechanism for testing the LaTeX rendering system. It allows you to see how the render class should be invoked. To get you started, I've provided the basic template shown in Listing 1.

Listing 1. render_example.php

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html>
<head>
<title>LaTeX Equations and Graphics in PHP</title>
</head>

<body>

<!-- form to enter LaTeX code -->
<form action="render_example.php" method="post">
<textarea rows="20"
  cols="60"
  name="render_text"></textarea><br />
<input name="submit"
  type="submit"
  value="Render" />
</form>

<?php
if (isset($_POST['submit'])) {
  echo '<h1>Result</h1>';

  require('render.class.php');

  $text = $_POST['render_text'];

  if (get_magic_quotes_gpc())
    $text = stripslashes($text);

  $render = new render();
  echo $render->transform($text);
}
?>

```



```
</body>
</html>
```

This PHP page provides a form for entering LaTeX code and then replaces the thunks with URLs to rendered PNG images through the transform method. Everything else is done behind the scenes in the render class.

Minimal Configuration Options

The skeleton for the render class is shown in Listing 2.

Listing 2. render.php

```
class render {

    var $LATEX_PATH = "/usr/local/bin/latex";
    var $DVIPS_PATH = "/usr/local/bin/dvips";
    var $CONVERT_PATH = "/usr/local/bin/convert";

    var $TMP_DIR =
        "/usr/home/barik/public_html/gehennom/lj/tmp";
    var $CACHE_DIR =
        "/usr/home/barik/public_html/gehennom/lj/cache";

    var $URL_PATH = "http://www.barik.net/lj/cache";

    function wrap($text) { ... }
    function transform($text) { ... }
    function render_latex($text) { ... }

}
```

You need to let PHP know where your tools are located and provide a directory where PHP can write temporary files and store its cache. For convenience, a URL_PATH also is needed. This URL_PATH is used when generating the image tags in HTML.

Don't be fooled by the simplicity. A vast array of options is available that you can pass to LaTeX and ImageMagick to modify the output PNG image, and you should explore them all. Here, I've merely provided the framework.

wrap Method

The wrap method takes your LaTeX thunk and surrounds it with a prologue and epilogue to create a valid LaTeX source file. You can consider this to be the equivalent of adding additional includes to a C file or importing packages in Java to extend the functionality of the language (Listing 3).

Listing 3. wrap.php7870l3.qrk

```
function wrap($thunk) {
    return <<<E0S
        \documentclass[10pt]{article}
```

```

% add additional packages here
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{pst-plot}
\usepackage{color}

\pagestyle{empty}
\begin{document}


```

As you can see, I include the packages I routinely need in the LaTeX wrapper. Consequently, I've included the American Mathematical Society (AMS) package, which provides additional mathematical constructs, as well as the PSTricks package to render vector graphics. The pagestyle is set to empty so that page numbers do not appear on images. Also, the thunk is inserted between the document blocks.

Not all of these packages may be available on your system. If necessary, you can download additional packages from the Comprehensive TeX Archive Network (CTAN) Web site (see the on-line Resources) to extend the functionality of your base LaTeX system. For example, packages for bar charts, UML notation and even Karnaugh maps can be downloaded. Whatever your needs, the repository is worth a look.

render_latex Method

The render_latex method (Listing 4) extracts all thunks and processes them individually until the thunk pool is exhausted.

Listing 4. render_latex.php

```

function render_latex($thunk, $hash) {
    $thunk = $this->wrap($thunk);

    $current_dir = getcwd();
    chdir($this->TMP_DIR);

    // create temporary LaTeX file
    $fp = fopen($this->TMP_DIR . "/"$hash.tex", "w+");
    fputs($fp, $thunk);
    fclose($fp);

    // run LaTeX to create temporary DVI file
    $command = $this->LATEX_PATH .
        " --interaction=nonstopmode " .
        $hash . ".tex";
    exec($command);

    // run dvips to create temporary PS file
    $command = $this->DVIPS_PATH .
        " -E $hash" .
        ".dvi -o " . "$hash.ps";
    exec($command);

    // run PS file through ImageMagick to

```

```
// create PNG file
$command = $this->CONVERT_PATH .
    " -density 120 $hash.ps $hash.png";
exec($command);

// copy the file to the cache directory
copy("$hash.png", $this->CACHE_DIR .
    "$hash.png");

chdir($current_dir);
}
```

The `thunk` parameter is obvious: it's the block of LaTeX code we're currently examining. The `hash` parameter is a unified version of the `thunk`, essentially, an md5 of the filename base.

I change to the temporary directory and write the `thunk` to a temporary LaTeX file. LaTeX then creates a DVI file. The command-line parameter tells LaTeX to run non-interactively. The resulting DVI file is converted to PostScript with the use of `dvips`, and the `-E` option specifies a bounding box. I then run the resulting PostScript file through `convert`—that's the program name—to convert the file to a PNG image. The `convert` tool has a slew of options, and the settings that will work best for you depend on your site.

Finally, be aware that the `exec` command returns a failure status code. For brevity, I've left out the error checking and always assume that all steps succeed. LaTeX also has a few dangerous commands that could be an issue for multiuser Web sites. It therefore might be prudent to return an error if certain keywords are found in the `thunk`.

When Things Go Awry

If something goes wrong at the rendering stage, you can try to process a LaTeX file manually by using the shell with the following commands for diagnostics:

```
latex --interaction=nonstopmode my.tex
dvips -E my.dvi -o my.ps
convert -density 120 my.ps my.png
```

This allows you to isolate the specific step at which the LaTeX renderer fails.

cleanup Method

During the LaTeX rendering process, a large number of temporary files are created. This cleanup method deletes these extraneous files, and there's really not much to it, as shown in Listing 5.

Listing 5. cleanup.php

```

function cleanup($hash) {

    $current_dir = getcwd();
    chdir($this->TMP_DIR);

    unlink($this->TMP_DIR . "/"$hash.tex");
    unlink($this->TMP_DIR . "/"$hash.aux");
    unlink($this->TMP_DIR . "/"$hash.log");
    unlink($this->TMP_DIR . "/"$hash.dvi");
    unlink($this->TMP_DIR . "/"$hash.ps");
    unlink($this->TMP_DIR . "/"$hash.png");

    chdir($current_dir);
}

```

transform Method

The transform method, shown in Listing 6, drives the rendering class and provides a public access point for the programmer.

Listing 6. transform.php

```

function transform($text) {

    preg_match_all("/\[tex\](.*?)\[\/tex\]/si", $text, $matches);

    for ($i = 0; $i < count($matches[0]); $i++) {

        $position = strpos($text, $matches[0][$i]);
        $thunk = $matches[1][$i];

        $hash = md5($thunk);
        $full_name = $this->CACHE_DIR . "/" .
            $hash . ".png";
        $url = $this->URL_PATH . "/" .
            $hash . ".png";

        if (!is_file($full_name)) {
            $this->render_latex($thunk, $hash);
            $this->cleanup($hash);
        }

        $text = substr_replace($text,
            "<img src=\"$url\" alt=\"Formula: $i\" />",
            $position, strlen($matches[0][$i]));
    }

    return $text;
}

```

The `preg_match_all` function in PHP extracts the thunks as well as the positions of each thunk. Each thunk then is parsed individually through the loop. Next, a unique md5 of the thunk text is created. This tells us whether a thunk has been cached before. If the thunk has not been cached, I call the LaTeX renderer method and immediately clean up the resulting temporary files. In either case, the thunk is substituted with a URL. When all thunks are processed, the text is returned.

Equation Examples

Now, let's look at a few examples that illustrate the kinds of equations you can render with the help of LaTeX. Most of these equations are taken from *A Guide To LaTeX* by Helmut Kopka and Patrick W. Daly, considered by many to be one of the essential books on the LaTeX system.

$$\frac{a^2 - b^2}{a + b} = a - b$$

Figure 2. Example: Fractions

```
[tex]
\begin{displaymath}
\frac{a^2 - b^2}{a + b} = a - b
\end{displaymath}
[/tex]
```

$$\text{corr}(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2 \right]^{1/2}}$$

Figure 3. Example: Correlation of Two Variables, X and Y

```
[tex]
\begin{displaymath}
\mathop{\mathrm{corr}}(X, Y) =
\frac{\displaystyle
\sum_{i=1}^n (x_i - \overline{x})
(y_i - \overline{y})}
{\displaystyle \biggl[
\sum_{i=1}^n (x_i - \overline{x})^2
\sum_{i=1}^n (y_i - \overline{y})^2
\biggr]^{1/2}}
\end{displaymath}
[/tex]
```

$$I(z) = \sin\left(\frac{\pi}{2} z^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n}}{1 \cdot 3 \cdots (4n+1)} z^{4n+1} - \cos\left(\frac{\pi}{2} z^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n+1}}{1 \cdot 3 \cdots (4n+3)} z^{4n+3}$$

Figure 4. Example: A More Complex Equation

```
[tex]
\begin{displaymath}
I(z) = \sin\left(\frac{\pi}{2} z^2\right) \sum_{n=0}^{\infty}
\frac{(-1)^n \pi^{2n}}{1 \cdot 3 \cdots (4n+1)} z^{4n+1}
- \cos\left(\frac{\pi}{2} z^2\right) \sum_{n=0}^{\infty}
\frac{(-1)^n \pi^{2n+1}}{1 \cdot 3 \cdots (4n+3)} z^{4n+3}
\end{displaymath}
[/tex]
```

```

-\cos( \frac{\pi}{2} z^2 ) \sum_{n=0}^{\infty}
\frac{ (-1)^n \pi^{2n + 1} }{1 \cdot 3
\cdots (4n + 3) } z^{4n + 3}
\end{displaymath}
[/tex]

```

Plotting Examples

Though LaTeX is a mathematical typesetting powerhouse, it also is capable in other arenas with the help of packages such as PSTricks. These plots are provided courtesy of Herbert Voss. On his Web site (see Resources), you can find further examples of using PSTricks to test the LaTeX rendering system. Getting some of his more-advanced examples to display correctly, however, may require considerable effort.

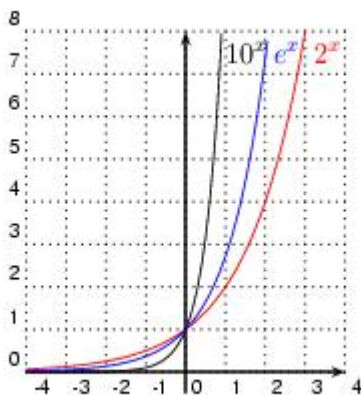


Figure 5. Example: Plot of 10^x , e^x , and 2^x

```

[tex]
\psset{unit=0.5cm}
\begin{pspicture}(-4,-0.5)(4,8)
\psgrid[subgriddiv=0,griddots=5,
gridlabels=7pt](-4,-0.5)(4,8)
\psline[linewidth=1pt]{->}(-4,0)(+4,0)
\psline[linewidth=1pt]{->}(0,-0.5)(0,8)
\psplot[plotstyle=curve,
linewidth=0.5pt]{-4}{0.9}{10 x exp}
\rput[l](1,7.5){$10^x$}
\psplot[plotstyle=curve,linecolor=red,
linewidth=0.5pt]{-4}{3}{2 x exp}
\rput[l](2.2,7.5){\color{blue}$e^x$}
\psplot[plotstyle=curve,linecolor=blue,
linewidth=0.5pt]{-4}{2.05}{2.7183 x exp}
\rput[l](3.2,7.5){\color{red}$2^x$}
\rput(4,8.5){\color{white}change\normalcolor}
\rput(-4,-1){\color{white}bounding box\normalcolor}
\end{pspicture}
[/tex]

```

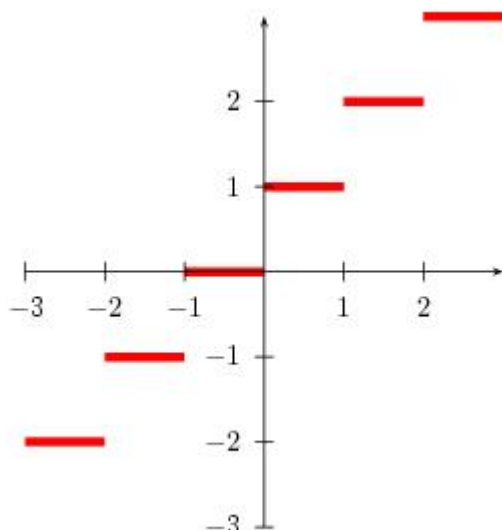


Figure 6. Example: Ceil Function

```
[tex]
\SpecialCoor
\begin{pspicture}(-3,-3)(3,3)
  \multido{\i=-2+1}{6}{%
    \psline[linewidth=3pt, linecolor=red]
      (\i,\i)(! \i\space 1 sub \i)%
    \psaxes[linewidth=0.2mm]{->}(0,0)(-3,-3)(3,3)
  }
\end{pspicture}
[/tex]
```

Available Implementations

Several implementations of LaTeX renderers are available on the Web today, some of which work better than others. Steve Mayer, for example, now maintains Benjamin Zeiss' original LaTeX renderer for PHP. Mayer also has written several plugins for common Weblog systems, including WordPress. If you want a pluggable solution for your site, this is the one I recommend.

Additionally, John Walker provides textogif, a Perl program that uses the LaTeX2HTML tools to render images in either GIF or PNG format by way of CGI. Finally, John Forkosh provides mimeTeX, written using C through CGI. Its advantage is that it does not require LaTeX or ImageMagick but does so at the expense of rendering quality.

Conclusion

Integrating LaTeX with your wiki or Weblog at first may seem like a daunting task. Once you get the hang of it, however, you'll wonder how you ever lived without it. Using this model, you also can see how other languages might be embedded within PHP in addition to LaTeX. Other ideas to consider include using Gnuplot to generate plots, Octave to evaluate complex expressions or POV-Ray to render 3-D scenes.

Today, the topics represented by the Weblog community largely are disproportionate. Indeed, many technical writers outside the field of programming have stayed away from Weblogs simply because the means to convey their ideas easily do not exist. I hope that the use of LaTeX rendering systems for the Web will bridge this critical gap.

Resources for this article: www.linuxjournal.com/article/8011.

Titus Barik is an IT consultant for small businesses. He's also an active Weblogger and technical bookworm. You can visit his Weblog at barik.net.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Optimization in GCC

M. Tim Jones

Issue #131, March 2005

Here's what the O options mean in GCC, why some optimizations aren't optimal after all and how you can make specialized optimization choices for your application.

In this article, we explore the optimization levels provided by the GCC compiler toolchain, including the specific optimizations provided in each. We also identify optimizations that require explicit specifications, including some with architecture dependencies. This discussion focuses on the 3.2.2 version of gcc (released February 2003), but it also applies to the current release, 3.3.2.

Levels of Optimization

Let's first look at how GCC categorizes optimizations and how a developer can control which are used and, sometimes more important, which are not. A large variety of optimizations are provided by GCC. Most are categorized into one of three levels, but some are provided at multiple levels. Some optimizations reduce the size of the resulting machine code, while others try to create code that is faster, potentially increasing its size. For completeness, the default optimization level is zero, which provides no optimization at all. This can be explicitly specified with option -O or -O0.

Level 1 (-O1)

The purpose of the first level of optimization is to produce an optimized image in a short amount of time. These optimizations typically don't require significant amounts of compile time to complete. Level 1 also has two sometimes conflicting goals. These goals are to reduce the size of the compiled code while increasing its performance. The set of optimizations provided in -O1 support these goals, in most cases. These are shown in Table 1 in the column labeled -O1. The first level of optimization is enabled as:

```
gcc -O1 -o test test.c
```

Optimization	Included in Level			
	-O1	-O2	-O3	-O4
defer-pop	●	●	●	●
thread-jumps	●	●	●	●
branch-probabilities	●	●	●	●
cprop-registers	●	●	●	●
guess-branch-probability	●	●	●	●
omit-frame-pointer	●	●	●	●
align-loops	○	●	○	●
align-jumps	○	●	○	●
align-labels	○	●	○	●
align-functions	○	●	○	●
optimize-sibling-calls	○	●	●	●
cse-follow-jumps	○	●	●	●
cse-skip-blocks	○	●	●	●
gcse	○	●	●	●
expensive-optimizations	○	●	●	●
strength-reduce	○	●	●	●
rerun-cse-after-loop	○	●	●	●
rerun-loop-opt	○	●	●	●
caller-saves	○	●	●	●
force-mem	○	●	●	●
peephole2	○	●	●	●
regmove	○	●	●	●
strict-aliasing	○	●	●	●
delete-null-pointer-checks	○	●	●	●
reorder-blocks	○	●	●	●
schedule-insns	○	●	●	●
schedule-insns2	○	●	●	●
inline-functions	○	○	○	●
rename-registers	○	○	○	●

Table 1. GCC optimizations and the levels at which they are enabled.

Any optimization can be enabled outside of any level simply by specifying its name with the `-f` prefix, as:

```
gcc -fdefer-pop -o test test.c
```

We also could enable level 1 optimization and then disable any particular optimization using the `-fno-` prefix, like this:

```
gcc -O1 -fno-defer-pop -o test test.c
```

This command would enable the first level of optimization and then specifically disable the `defer-pop` optimization.

Level 2 (-O2)

The second level of optimization performs all other supported optimizations within the given architecture that do not involve a space-speed trade-off, a balance between the two objectives. For example, loop unrolling and function inlining, which have the effect of increasing code size while also potentially making the code faster, are not performed. The second level is enabled as:

```
gcc -O2 -o test test.c
```

Table 1 shows the level `-O2` optimizations. The level `-O2` optimizations include all of the `-O1` optimizations, plus a large number of others.

Level 2.5 (-Os)

The special optimization level (`-Os` or `size`) enables all `-O2` optimizations that do not increase code size; it puts the emphasis on size over speed. This includes all second-level optimizations, except for the alignment optimizations. The alignment optimizations skip space to align functions, loops, jumps and labels to an address that is a multiple of a power of two, in an architecture-dependent manner. Skipping to these boundaries can increase performance as well as the size of the resulting code and data spaces; therefore, these particular optimizations are disabled. The size optimization level is enabled as:

```
gcc -Os -o test test.c
```

In gcc 3.2.2, `reorder-blocks` is enabled at `-Os`, but in gcc 3.3.2 `reorder-blocks` is disabled.

Level 3 (-O3)

The third and highest level enables even more optimizations (Table 1) by putting emphasis on speed over size. This includes optimizations enabled at `-O2` and `rename-register`. The optimization `inline-functions` also is enabled here, which can increase performance but also can drastically increase the size of the object, depending upon the functions that are inlined. The third level is enabled as:

```
gcc -O3 -o test test.c
```

Although -O3 can produce fast code, the increase in the size of the image can have adverse effects on its speed. For example, if the size of the image exceeds the size of the available instruction cache, severe performance penalties can be observed. Therefore, it may be better simply to compile at -O2 to increase the chances that the image fits in the instruction cache.

Architecture Specification

The optimizations discussed thus far can yield significant improvements in software performance and object size, but specifying the target architecture also can yield meaningful benefits. The -march option of gcc allows the CPU type to be specified (Table 2).

Table 2. x86 Architectures

Target CPU Types	-march= Type
i386 DX/SX/CX/EX/SL	i386
i486 DX/SX/DX2/SL/SX2/DX4	i486
487	i486
Pentium	pentium
Pentium MMX	pentium-mmx
Pentium Pro	pentiumpro
Pentium II	pentium2
Celeron	pentium2
Pentium III	pentium3
Pentium 4	pentium4
Via C3	c3
Winchip 2	winchip2
Winchip C6-2	winchip-c6
AMD K5	i586
AMD K6	k6

Target CPU Types	-march= Type
AMD K6 II	k6-2
AMD K6 III	k6-3
AMD Athlon	athlon
AMD Athlon 4	athlon
AMD Athlon XP/MP	athlon
AMD Duron	athlon
AMD Tbird	athlon-tbird

The default architecture is i386. GCC runs on all other i386/x86 architectures, but it can result in degraded performance on more recent processors. If you're concerned about portability of an image, you should compile it with the default. If you're more interested in performance, pick the architecture that matches your own.

Let's now look at an example of how performance can be improved by focusing on the actual target. Let's build a simple test application that performs a bubble sort over 10,000 elements. The elements in the array have been reversed to force the worst-case scenario. The build and timing process is shown in Listing 1.

Listing 1. Effects of Architecture Specification on a Simple Application

```
[mtj@camus]$ gcc -o sort sort.c -O2
[mtj@camus]$ time ./sort

real    0m1.036s
user    0m1.030s
sys     0m0.000s
[mtj@camus]$ gcc -o sort sort.c -O2 -march=pentium2
[mtj@camus]$ time ./sort

real    0m0.799s
user    0m0.790s
sys     0m0.010s
[mtj@camus]$
```

By specifying the architecture, in this case a 633MHz Celeron, the compiler can generate instructions for the particular target as well as enable other optimizations available only to that target. As shown in Listing 1, by specifying the architecture we see a time benefit of 237ms (23% improvement).

Although Listing 1 shows an improvement in speed, the drawback is that the image is slightly larger. Using the size command (Listing 2), we can identify the sizes of the various sections of the image.

Listing 2. Size Change of the Application from Listing 1

```
[mtj@camus]$ gcc -o sort sort.c -O2
[mtj@camus]$ size sort
   text  data  bss   dec    hex filename
   842   252    4   1098   44a sort
[mtj@camus]$ gcc -o sort sort.c -O2 -march=pentium2
[mtj@camus]$ size sort
   text  data  bss   dec    hex filename
   870   252    4   1126   466 sort
[mtj@camus]$
```

From Listing 2, we can see that the instruction size (text section) of the image increased by 28 bytes. But in this example, it's a small price to pay for the speed benefit.

Math Unit Optimizations

Some specialized optimizations require explicit definition by the developer. These optimizations are specific to the i386 and x86 architectures. A math unit, for one, can be specified, although in many cases it is automatically defined based on the specification of a target architecture. Possible units for the `-mfpmath=` option are shown in Table 3.

Table 3. Math Unit Optimizations

Option	Description
387	Standard 387 Floating Point Coprocessor
sse	Streaming SIMD Extensions (Pentium III, Athlon 4/XP/MP)
sse2	Streaming SIMD Extensions II (Pentium 4)

The default choice is `-mfpmath=387`. An experimental option is to specify both `sse` and `387` (`-mfpmath=sse,387`), which attempts to use both units.

Alignment Optimizations

In the second optimization level, we saw that a number of alignment optimizations were introduced that had the effect of increasing performance but also increasing the size of the resulting image. Three additional alignment optimizations specific to this architecture are available. The `-malign-int` option allows types to be aligned on 32-bit boundaries. If you're running on a 16-bit

aligned target, `-mno-align-int` can be used. The `-malign-double` controls whether doubles, long doubles and long-longs are aligned on two-word boundaries (disabled with `-mno-align-double`). Aligning doubles provides better performance on Pentium architectures at the expense of additional memory.

Stacks also can be aligned by using the option `-mpreferred-stack-boundary`. The developer specifies a power of two for alignment. For example, if the developer specified `-mpreferred-stack-boundary=4`, the stack would be aligned on a 16-byte boundary (the default). On the Pentium and Pentium Pro targets, stack doubles should be aligned on 8-byte boundaries, but the Pentium III performs better with 16-byte alignment.

Speed Optimizations

For applications that utilize standard functions, such as `memset`, `memcpy` or `strlen`, the `-minline-all-stringops` option can increase performance by inlining string operations. This has the side effect of increasing the size of the image.

Loop unrolling occurs in the process of minimizing the number of loops by doing more work per iteration. This process increases the size of the image, but it also can increase its performance. This option can be enabled using the `-funroll-loops` option. For cases in which it's difficult to understand the number of loop iterations, a prerequisite for `-funroll-loops`, all loops can be unrolled using the `-funroll-all-loops` optimization.

A useful option that has the disadvantage of making an image difficult to debug is `-fomit-leaf-frame-pointer`. This option keeps the frame pointer out of a register, which means less setup and restore of this value. In addition, it makes the register available for the code to use. The optimization `-fomit-frame-pointer` also can be useful.

When operating at level `-O3` or having `-finline-functions` specified, the size limit of the functions that may be inlined can be specified through a special parameter interface. The following command illustrates capping the size of the functions to inline at 40 instructions:

```
gcc -o sort sort.c --param max-inline-insns=40
```

This can be useful to control the size by which an image is increased using `-finline-functions`.

Code Size Optimizations

The default stack alignment is 4, or 16 words. For space-constrained systems, the default can be minimized to 8 bytes by using the option `-mpreferred-stack-`

boundary=2. When constants are defined, such as strings or floating-point values, these independent values commonly occupy unique locations in memory. Rather than allow each to be unique, identical constants can be merged together to reduce the space that's required to hold them. This particular optimization can be enabled with `-fmerge-constants`.

Graphics Hardware Optimizations

Depending on the specified target architecture, certain other extensions are enabled. These also can be enabled or disabled explicitly. Options such as `-mmmx` and `-m3dnow` are enabled automatically for architectures that support them.

Other Possibilities

We've discussed many optimizations and compiler options that can increase performance or decrease size. Let's now look at some fringe optimizations that may provide a benefit to your application.

The `-ffast-math` optimization provides transformations likely to result in correct code but it may not adhere strictly to the IEEE standard. Use it, but test carefully.

When global common sub-expression elimination is enabled (`-fgcse`, level `-O2` and above), two other options may be used to minimize load and store motions. Optimizations `-fgcse-lm` and `-fgcse-sm` can migrate loads and stores outside of loops to reduce the number of instructions executed within the loop, therefore increasing the performance of the loop. Both `-fgcse-lm` (load-motion) and `-fgcse-sm` (store-motion) should be specified together.

The `-fforce-addr` optimization forces the compiler to move addresses into registers before performing any arithmetic on them. This is similar to the `-fforce-mem` option, which is enabled automatically in optimization levels `-O2`, `-O3` and `-O4`.

A final fringe optimization is `-fsched-spec-load`, which works with the `-fschedule-insns` optimization, enabled at `-O2` and above. This optimization permits the speculative motion of some load instructions to minimize execution stalls due to data dependencies.

Testing for Improvements

Earlier we used the `time` command to identify how much time was spent in a given command. This can be useful, but when we're profiling our application, we need more insight into the image. The `gprof` utility provided by GNU and the

GCC compiler meets this need. Full coverage of gprof is outside the scope of this article, but Listing 3 illustrates its use.

Listing 3. Simple Example of gprof

```
[mtj@camus]$ gcc -o sort sort.c -pg -O2 -march=pentium2
[mtj@camus]$ ./sort
[mtj@camus]$ gprof --no-graph -b ./sort gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
 %   cumulative   self           self         total
time  seconds  seconds   calls  ms/call  ms/call  name
100.00    0.79    0.79         1    790.00    790.00  bubbleSort
  0.00    0.79    0.00         1     0.00     0.00  init_list
[mtj@camus]$
```

The image is compiled with the `-pg` option to include profiling instructions in the image. Upon execution of the image, a `gmon.out` file results that can be used with the `gprof` utility to produce human-readable profiling data. In this use of `gprof`, we specify the `-b` and `--no-graph` options. For brief output (excludes the verbose field explanations), we specify `-b`. The `--no-graph` option disables the emission of the function call-graph; it identifies which functions call which others and the time spent on each.

Reading the example from Listing 3, we can see that `bubbleSort` was called once and took 790ms. The `init_list` function also was called, but it took less than 10ms to complete (the resolution of the profile sampling), so its value was zero.

If we're more interested in changes in the size of the object than speed, we can use the `size` command. For more specific information, we can use the `objdump` utility. To see a list of the functions in our object, we can search for the `.text` sections, as in:

```
objdump -x sort | grep .text
```

From this short list, we can identify the particular function we're interested in understanding better.

Examining Optimizations

The GCC optimizer is essentially a black box. Options and optimization flags are specified, and the resulting code may or may not improve. When they do improve, what exactly happened within the resulting code? This question can be answered by looking at the resulting code.

To emit target instructions from the compiler, the `-S` option can be specified, such as:

```
gcc -c -S test.c
```

which tells gcc to compile the source only (-c) but also to emit assembly code for the source (-S). The resulting assembly output will be contained in the file test.s.

The disadvantage of the previous approach is you see only assembly code, no aspect of the size of the actual instructions is given. For this, we can use objdump to emit both assembly and native instructions, like so:

```
gcc -c -g test.c  
objdump -d test.o
```

For gcc, we specify compile with only -c, but we also want to include debug information in the object (-g). Using objdump, we specify the -d option to disassemble the instructions in the object. Finally, we can get assembly-interspersed source listings with:

```
gcc -c -g -Wa,-ahl,-L test.c
```

This command uses the GNU assembler to emit the listing. The -Wa option is used to pass the -ahl and -L options to the assembler to emit a listing to standard-out that contains the high-level source and assembly. The -L option retains the local symbols in the symbol table.

Conclusion

All applications are different, so there's no magic configuration of optimization and option switches that yield the best result. The simplest way to achieve good performance is to rely on the -O2 optimization level; if you're not interested in portability, specify the target architecture using -march=. For space-constrained applications, the -Os optimization level should be considered first. If you're interested in squeezing the most performance out of your application, your best bet is to try out the different levels and then use the various utilities to check the resulting code. Enabling and/or disabling certain optimizations also may help exploit the optimizer to receive the best performance.

Resources for this article: www.linuxjournal.com/article/7971.

M. Tim Jones (mtj@mtjones.com) is a senior principal engineer with Emulex Corp. in Longmont, Colorado. In addition to being an embedded firmware engineer, Tim recently finished writing the book *BSD Sockets Programming from a Multilanguage Perspective*. He has written kernels for communications and research satellites and now develops embedded firmware for networking products.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

At the Forge

Bloglines Web Services, Continued

Reuven M. Lerner

Issue #131, March 2005

Although some Web community sites get evil and lock in the users, Bloglines takes an open approach and lets you point your own scripts at its Web services API. Drop in and catch up with your favorite blogs.

I am writing this column a few days after the November 2, 2004, elections in the United States. As an admitted political junkie, I enjoy the modern era of computerized, always-on punditry. No longer must I switch TV stations or read several newspapers at the local library; now, I can follow the sound bites as they pass from the candidates to the press to the various partisan sites.

Keeping up with many different news and opinion sites can consume quite a bit of time. As we have seen over the last few months, everyone has benefited from the creation of news aggregators—programs that read the RSS and Atom syndication feeds produced by Weblogs, newspapers and other frequently updated sites. An aggregator, as its name suggests, takes these feeds and puts them into a single, easily accessible listing.

Bloglines.com is an Internet startup that provides a Web-based news aggregator. In and of itself, this should not surprise anyone; the combination of syndication, aggregation and the Web made this a natural idea. And, Bloglines isn't unique; there are other, perhaps lesser-known, Web-based news aggregators.

One unique service that Bloglines offers its subscribers, however, is the ability to use Bloglines' internal database to create their own news aggregators or their own applications built from the data Bloglines has collected. This information is available without charge, under a fairly unrestrictive license, to any programmer interested in harvesting the results of Bloglines' engine. The fact that Bloglines checks for updates on hundreds of thousands of blogs and

sites approximately every hour means that someone using the Web services API can be assured of getting the most recent Weblog content.

Last time [LJ, January 2005], we looked at the Notifier API, which provides access to a particular user's available-but-unread feeds. We also discussed the Blogroll API, which allows users to determine and use programmatically, if they wish, a list of people who are pointing to a feed. As we saw, these APIs made it easy for us to find out that new Weblog entries were available or to create our own custom aggregation page listing Weblogs of interest.

Something was missing in the functionality that we exposed in that article, however. It's nice to know that new Weblog entries are among my Bloglines subscriptions, but it would be even nicer to know which blogs had been updated. And, it's nice to get a list of my current subscriptions, but I would be much happier to find out which of them have been updated—and to find out when they were most recently updated, how many new entries are in each Weblog and what those entries contain. In other words, I want to be able to replace the current Bloglines interface with one of my own, displaying new Weblog entries in a format that isn't dictated by the Bloglines.com Web site.

Luckily, the Web services developers at Bloglines have made it possible to do exactly this by way of the sync API. This month, we continue our exploration of Bloglines Web services, looking in detail at the sync API it provides. We also are going to create a simple news aggregator of our own, providing some of the same features as the Bloglines interface.

Subscriptions and Items

At the end of the day, a news aggregator such as Bloglines simply is a list of URLs. Indeed, the Python-based news aggregator we created two months ago using the Universal Feed Parser was precisely such a program—it looked at a set of URLs in a file and retrieved the most recent items associated with those URLs. Each individual Weblog posting must be associated with one of the URLs on a list. Removing a URL from the subscription lists makes its associated postings irrelevant to that user and invisible to them.

The fact that Bloglines has multiple users rather than a single user means it must keep track of not only a set of different URLs, but also which URL is associated with each user. Although this obviously complicates things somewhat, modern high-level languages make the difference between these two data structures easily understood. Rather than simply storing a list of URLs, we must create a hash table, in which the key is a user ID and the value is the list associated with that particular user. Once we have the user's unique ID, we easily can keep track of that particular user's subscriptions.

Of course, Bloglines is keeping track of subscriptions not for a few thousand users, but for many tens or hundreds of thousands of users. Thus, it is safe to assume they are not using such a naive implementation, which would suffice for a small experiment or an aggregator designed for a small number of people. Things get a bit trickier when you approach Bloglines' user load. Each user's list of subscriptions can't be a simple URL; it is more likely to be an ID number (or primary key, in database jargon) associated with a URL. Such a system gives multiple participants the chance to subscribe to a site's syndication feed and allows Bloglines to suggest new Weblogs that they might enjoy, based on their current subscriptions.

It thus should come as no surprise to learn that retrieving new Weblog postings from Bloglines is a two-step process, with the first step requiring us to retrieve a list of subscriptions. That is, we first ask Bloglines for a list of subscription IDs associated with a user. We then ask Bloglines to send us all of the new items for this user and this subscription ID.

Implementations of the Bloglines Web services API are available in several different languages. Because Perl is my default language for creating new applications, I am going to use the `WebService::Bloglines` module that has been uploaded to CPAN, the Comprehensive Perl Archive Network, a worldwide collection of Web and FTP servers from which Perl and its modules can be retrieved. For example, Listing 1 contains a simple program (`bloglines-listsubs.pl`) that displays the title, subscription ID and URL for each of a user's subscriptions. A number of additional values are available for each of the subscriptions; the documentation for `WebService::Bloglines`, as well as the Bloglines API documentation, lists these in detail.

Listing 1. Display a User's Subscriptions

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use WebService::Bloglines;

my $username = 'reuven@lerner.co.il';
my $password = 'MYPASS';

my $bloglines =
    WebService::Bloglines->new(username => $username,
                               password => $password);

# Do we want to mark them as read?
my $mark_unread = 0;

# From what date do we want to download items?
# (This should be in Unix "time"

my $subscriptions = $bloglines->listsubs();

if ($subscriptions)
```

```

{
    # list all feeds
    my @feeds = $subscriptions->feeds();

    # Get each feed's title and URL
    foreach my $feed (@feeds) {
        my $title = $feed->{title};
        my $url   = $feed->{htmlUrl};
        my $subId = $feed->{BloglinesSubId};

        print "Subscribed to '$title', "
            . "subId '$subId' at '$url'\n";
    }
}
else
{
    print "No subscriptions.\n"
}
}

```

If you are interested in preserving the subscription hierarchy the Bloglines.com interface gives users, you might want to examine the folders function, rather than the feed function used in Listing 1. Although feed returns a flat list of subscriptions, folders keeps things organized as they exist on the Bloglines site.

Getting Items within a Subscription

Now that we know how to retrieve the subscription IDs associated with a particular Bloglines user, we can retrieve individual items associated with a particular subscription ID. For example, Listing 2 is a short program that retrieves all of a user's subscriptions and then displays all of the newly updated items for each. The output is in plain-text format, not in HTML, which means the displayed link is not clickable. But, it would not be particularly difficult to run such a program in a cron job and dump its output into an HTML file, thereby giving an up-to-the-minute personalized list of feeds. Of course, Bloglines provides such a service at no cost whenever you might want to check its Web site. So, although such a program is an interesting use of the Bloglines Web services, it doesn't have a compelling use outside of those services.

Listing 2. bloglines-getitems.pl

```

#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use Webservice::Bloglines;

my $username = 'reuven@lerner.co.il';
my $password = 'MYPASS';

my $bloglines =
    Webservice::Bloglines->new(username => $username,
                               password => $password);

# Do we want to mark them as read?
my $mark_unread = 0;

# From what date do we want to download items?

```

```

# (This should be in Unix "time"

my $subscriptions = $bloglines->listsubs();

if ($subscriptions)
{
    # list all feeds
    my @feeds = $subscriptions->feeds();

    foreach my $feed (@feeds) {
        my $title = $feed->{title};
        my $url = $feed->{htmlUrl};
        my $subId = $feed->{BloglinesSubId};

        print "Subscribed to '$title', "
            . "subId '$subId' at '$url'\n";

    }

    my $update;

    # Trap errors!
    eval {$update = $bloglines->getitems($subId)};

    # Keep track of errors, showing "no change"
    if ($?) {
        if ($? == /304 No Change/) {
            print "\t No change\n";
        }
        else {
            print "\t Error code '$?' "
                . "retrieving updates.\n";
        }
    }

    # No errors? Show some basics about the items.
    else
    {
        foreach my $item ($update->items)
        {
            my $title = $item->{title};
            my $creator = $item->{dc}->{creator};
            my $link = $item->{link};
            my $pubDate = $item->{pubDate};
            print "\t$title by $creator "
                . "on $pubDate ($link)\n";
        }
    }
}
else
{
    print "No subscriptions.\n"
}

```

One of the clever things that Bloglines has done in its Web services definition is to use HTTP return codes to indicate errors and unusual circumstances. For example, the 200 (OK) response code indicates that new items may be read and that `getitems($subId)` contains one or more such data structures. The 304 (unchanged) response code, which normally indicates a page of HTML has not changed since it last was requested, here has a slightly different function; it indicates that a particular subscriber already has seen all of the available items for this subscription. Other response codes (401, 403 and 410) indicate authentication errors and probably mean that the requesting user has made a mistake in typing the Bloglines user name, password or both.

Unfortunately, Perl's handling of such response codes is less than optimal. In order to handle them, we must invoke `$bloglines->getitems()` inside of an eval block and check for a non-empty value of `$@` immediately after the eval. If `$@` is empty, we can assume that we received a 200 (OK) HTTP response code and there are new items to read. But if it contains a value, we then can rewrite the output message, as we did in Listing 2 . If we fail to trap this method call within an eval block, however, our program will die with a fatal runtime error the first time we receive anything other than a 200 response code.

Finally, two optional parameters make the Bloglines functionality complete. The first, known as `n`, is a simple true-or-false (1 or 0) value that tells Bloglines if it should update the already-seen bit for the articles it is sending to you. Normally, when a user is viewing Weblog postings with the Bloglines.com Web interface, this is set to 1, meaning you do not see any already-seen articles a second time. Perhaps because they knew the Web services API currently supplements other news aggregation applications, Bloglines wisely changed the default to 0 in this API.

The second optional parameter, known as `d`, tells Bloglines the first date from which you would like to download a particular site's postings. The value is in UNIX time format, meaning that you send the number of seconds since January 1, 1970. This number is readily available with the `time` function in most major languages, and it allows you to indicate with great precision exactly how far back you want to delve into a particular site's history, as stored by Bloglines.

Conclusion

To be honest, I am an enthusiastic Bloglines user without being sure exactly where the site and company are headed. I cannot imagine that it will continue to be free of charge and of any advertising indefinitely, unless its investors are highly charitable or extremely naive. I enjoy its fine interface, the fact that I easily can access the Weblogs on which I have depended for political insight—or screaming, depending on how you interpret such punditry—and its speedy, robust functionality.

But as Amazon, eBay and Google have demonstrated over the last few years, providing a Web services interface to your core data opens the door to many new creative applications that a company's internal developers never think to create. Bloglines is only beginning to expose its functionality with Web services, and although it has taken only an initial and tentative step in this direction, what I have seen appears to be promising. I look forward to seeing applications that will be built on top of this API, as well as the additional APIs that Bloglines and its competitors will offer in an attempt to make Bloglines the central site for Weblogs, readers and developers alike.

Resources for this article: www.linuxjournal.com/article/7961.

Reuven M. Lerner, a longtime Web/database consultant and developer, now is a graduate student in the Learning Sciences program at Northwestern University. His Weblog is at altneuland.lerner.co.il, and you can reach him at reuven@lerner.co.il.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Kernel Korner

Analysis of the HTB Queuing Discipline

Yaron Benita

Issue #131, March 2005

Can Linux do Quality of Service in a way that both offers high throughput and does not exceed the defined bandwidth? Here's a thorough test.

The Hierarchical Token Buckets (HTB) queuing discipline, part of the Linux set of traffic control functions, is a mechanism that provides QoS capabilities and is useful for fine-tuning TCP traffic flow. This article offers a brief overview of queuing discipline components and describes the results of several preliminary performance tests. Several configuration scenarios were set up within a Linux environment, and an Ixia device was used to generate traffic. This testing demonstrated that throughput accuracy can be manipulated and that the bandwidth range is accurate within a 2Mbit/s range. The test results demonstrated the performance and accuracy of the HTB queuing algorithms and revealed methods for improving traffic management.

The traffic control mechanism comes into play after an IP packet is queued for transmit on an output interface but before the packet actually is transmitted by the driver. Figure 1 shows where traffic control decisions are made in relation to packet transmission on the physical Ethernet and transport-layer protocols, such as UDP and TCP.

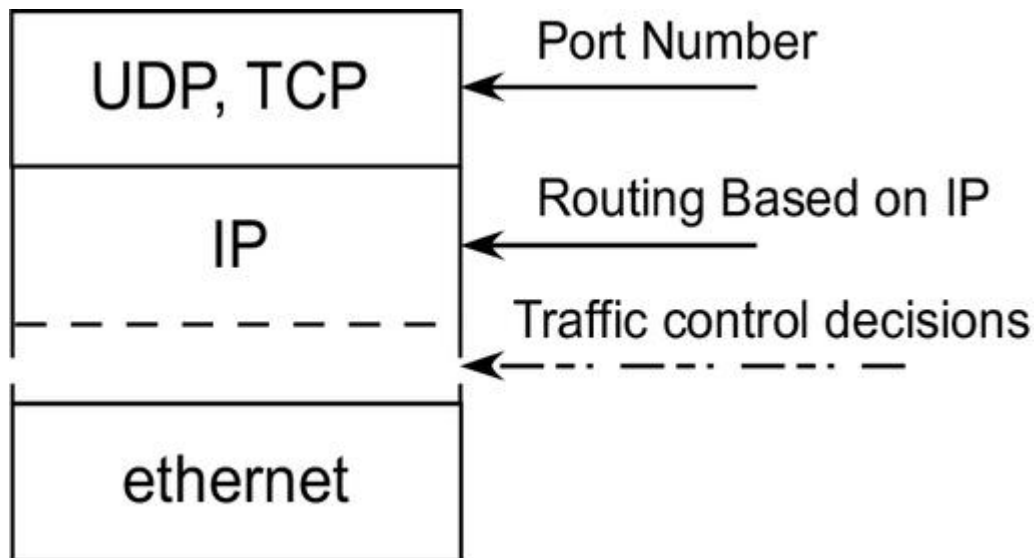


Figure 1. The kernel makes traffic control decisions after the packet is queued for transmit.

The traffic control kernel functionality, as implemented in Linux by Alexey Kuznetsov, includes four main components: queuing disciplines, classes of service, filters and policing.

Queuing disciplines are software mechanisms that define the algorithms used for treating queued IP packets. Each network device is associated with a queuing discipline, and a typical queuing discipline uses the FIFO algorithm to control the queued packets. The packets are stored in the order received and are queued as fast as the device associated with the queue can send them. Linux currently supports various queuing disciplines and provides ways to add new disciplines.

A detailed description of queuing algorithms can be found on the Internet at "Iproute2+tc Notes" (see the on-line Resources). The HTB discipline uses the TBF algorithm to control the packets queued for each defined class of service associated with it. The TBF algorithm provides traffic policing and traffic-shaping capabilities. A detailed description of the TBF algorithm can be found in the Cisco IOS Quality of Service Solutions Configuration Guide (see "Policing and Shaping Overview").

A class of service defines policing rules, such as maximum bandwidth or maximum burst, and it uses the queuing discipline to enforce those rules. A queuing discipline and a class are tied together. Rules defined by a class must be associated with a predefined queue. In most cases, every class owns one queue discipline, but it also is possible for several classes to share the same queue. In most cases when queuing packets, the policing components of a specific class discard packets that exceed a certain rate (see "Policing and Shaping Overview").

Filters define the rules used by the queuing discipline. The queuing discipline in turn uses those rules to decide to which class it needs to assign incoming packets. Every filter has an assigned priority. The filters are sorted in ascending order, based on their priorities. When a queue discipline has a packet for queuing, it tries to match the packet to one of the defined filters. The search for a match is done using each filter in the list, starting with the one assigned the highest priority. Each class or queuing discipline can have one or more filters associated with it.

Policing components make sure that traffic does not exceed the defined bandwidth. Policing decisions are made based on the filter and the class-defined rules. Figure 2 shows the relationship among all the components in the Linux traffic control mechanism.

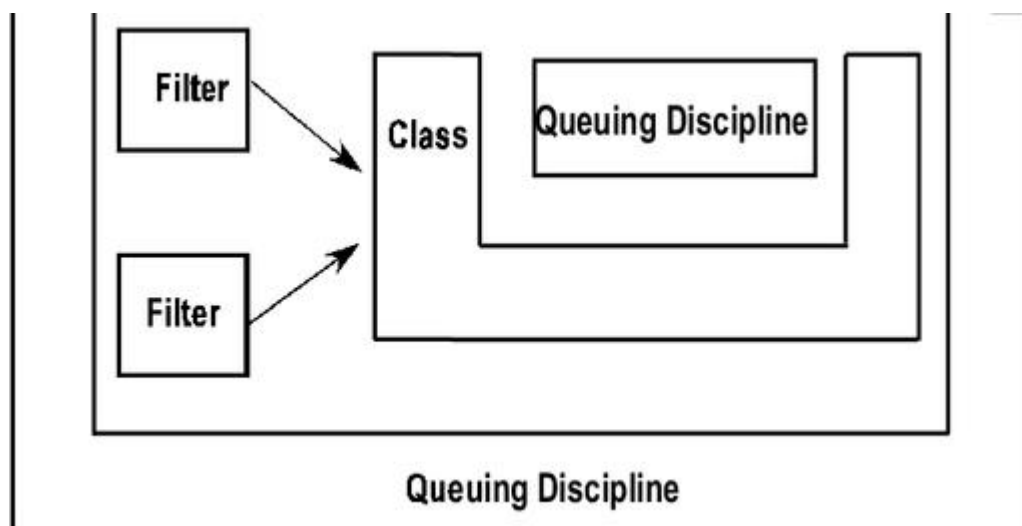


Figure 2. Linux traffic control mechanisms include queuing disciplines, classes, filters and policing.

TC Tool and HTB Definitions

TC is a user-level program that creates queues, classes and filters and associates them with an output interface (see “tc—Linux QoS control tool” in Resources). The filters can be set up based on the routing table, u32 classifiers and TOS classifiers. The program uses netlink sockets in order to communicate with the kernel's networking system functions. Table 1 lists the three main functions and their corresponding TC commands. See the HTB Linux Queuing Discipline User Guide for details regarding TC command options.

Table 1. TC Tool Functions and Commands

TC Function	Command
tc qdisc	Create a queuing discipline.

TC Function	Command
tc filter	Create a filter.
tc class	Create a class.

The HTB mechanism offers one way to control the use of the outbound bandwidth on a given link. To use the HTB facility, it should be defined as the class and the queuing discipline type. HTB shapes the traffic based on the TBF algorithm, which does not depend on the underlying bandwidth. Only the root queuing discipline should be defined as an HTB type; all the other class instances use the FIFO queue (default). The queuing process always starts at the root level and then, based on rules, decides which class should receive the data. The tree of classes is traversed until a matched leaf class is found (see “Hierarchical Token Bucket Theory”).

Testing

In order to test the accuracy and performance of the HTB, we used the following pieces of network equipment: one Ixia 400 traffic generator with a 10/100 Mbps Ethernet load module (LM100TX3) and one Pentium 4 PC (1GB RAM, 70GB hard drive) running a 2.6.5 Linux kernel. Two testing models were designed, one to test policing accuracy and one to test bandwidth sharing.

The first model (Figure 3) was used for testing the policing accuracy of a specific defined class. Port 1 in the Ixia machine generated traffic sent to IP 192.168.10.200 from one or more streams. The Linux machine routed the packets to interface eth0 (static route) and then sent them back to the Ixia machine on Port 2. All of the traffic control attributes were defined on the eth0 interface. All of the analysis was completed based on traffic results captured on Port 2 (the Ixia machine).

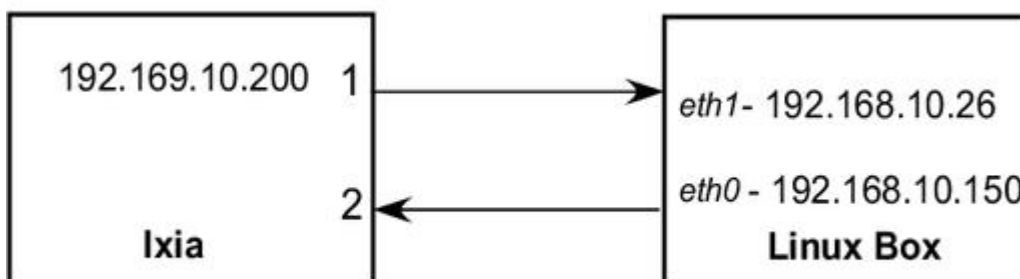


Figure 3. Test Model #1 Configuration

The second model (Figure 4) was used to test the way the bandwidth of two streams from the same class is shared. In this case, another two Ixia ports for transmitting data were used.

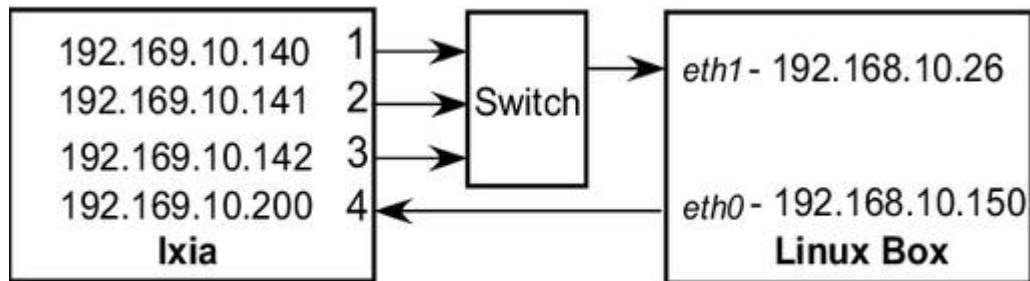


Figure 4. Test Model #2 Configuration

Port 1, Port 2 and Port 3 in the Ixia machine generated traffic sent to IP 192.168.10.200, each using one stream. The Linux machine routed those packets to interface eth0 based on a static route and then sent them back to the Ixia machine on Port 2. Traffic control attributes were defined on the eth0 interface. All analysis was done based on the traffic result captures on Port 2 (Ixia machine).

Ixia Configuration and Limitations

In all of the tests, the sending ports transmitted continuous bursts of packets on a specified bandwidth. The Ixia 10/100 Mbps Ethernet load module (model LM100TX3) has four separate ports, and each port can send up to 100Mbit/s. The Ixia load module provided support for generating multiple streams in one port but with one limitation: it couldn't mix the streams together and served only one stream at a time. This limitation exists because the scheduler works in a round-robin fashion. It sends a burst of bytes from stream X, moves to the next stream and then sends a burst of bytes from stream Y.

In order to generate a specific bandwidth from a stream, which is part of a group of streams defined in one port, specific attributes of the Ixia machine's configuration had to be fine-tuned. The attributes that required fine-tuning and their definitions are as follows:

- Burst: the number of packets sent by each stream, before moving to serve the next stream.
- Packet size: the size of a packet being sent by a stream.
- Total bandwidth: the total bandwidth used by all streams.

See Table 2 for Ixia configuration details.

Table 2. Ixia Configuration

Stream	Generated-Bandwidth	Packet Size	Burst Size
1	15Mbit/s	512B	150
2	10Mbit/s	512B	100
3	2Mbit/s	512B	20
Total	27Mbit/s	-	-

The goal was to determine the appropriate burst size that would achieve the requested generated bandwidth for each stream. Because all three streams used the same physical line, the way the data was sent on the line resembles the illustration in Figure 5.

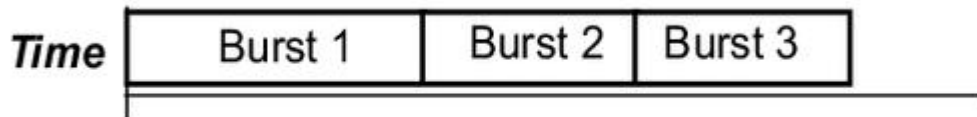


Figure 5. Data as sent on the line from the Ixia machine to the Linux system being tested.

The following equations define the relationship between the attributes:

$$T_c = \text{Sum}(B_{s-i} * 8 * P_{s-i}) / T_b$$

$$N_c = 1/T_c$$

$$B_{n-i} = N_c * P_{s-i} * 8 * B_{s-i}$$

Table 3 explains the variables used in the equation.

Table 3. Variables Used in the Attribute Relationship Equation

Attribute	Definition
T_c	The sum of the times (in seconds) it takes to send bursts 1-i ($T_{c1} + T_{c2} + T_{c3} + \dots$).
B_{s-i}	The number of packets in a burst of stream i.
P_{s-i}	The size of packet sent by stream i.
T_b	The total bandwidth sent by all streams (bits/sec).

Attribute	Definition
N_C	The number of T_C bursts in one second.
B_{n-i}	The requested bandwidth of stream i (bits/sec).

Assuming that the packet size is the same for all streams, as in the example, the remaining calculation is that of the burst size.

Because all the streams share the same bandwidth, the requested burst values can be found by examining the ratios between the requested bandwidths, using the equation $B^{S-i} = B^{n-i}$. This number could be unusually large, though, so it can be divided until a reasonable value is obtained. In order to have different packet sizes defined for each stream, the burst size values can be altered until the required bandwidth is obtained for each stream. A spreadsheet program simplifies the calculation of multiple bandwidths.

Test Cases and Test Results

When defining the HTB configuration, the following options of the tc class commands were used in order to achieve the required results:

- rate = the maximum bandwidth a class can use without borrowing from other classes.
- ceiling = the maximum bandwidth that a class can use, which limits how much bandwidth the class can borrow.
- burst = the amount of data that could be sent at the ceiling speed before moving to serve the next class.
- cburst = the amount of data that could be sent at the wire speed before moving to serve the next class.

Most of the traffic in the Internet world is generated by TCP, so packet sizes representative of datagram sizes, such as 64 (TCP Ack), 512 (FTP) and 1,500, were included for all test cases.

Testing Model 1



Figure 6. Test 1: One Stream In, One Stream Out

Table 4. Results for Testing Model 1

Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)
Default	Default	128	40	33.5
Default	Default	64	40	22 (Linux halt)
Default	Default	64	32 (Max)	29.2
15k	15k	64	32 (Max)	30
15k	15k	512	32 & 50 & 70	25.3
15k	15k	1,500	32 & 50 & 70	25.2
18k	18k	64	32 (Max)	29.2
18k	18k	512	32 & 50 & 70	30.26
18k	18k	1,500	32 & 50 & 70	29.29

From the results in Table 4, the following statements can be made:

- The maximum bandwidth a Linux machine can forward (receive on one interface and transmit on another interface) with continuous streams of 64-byte packets, is approximately 34Mbit/s.
- The burst/cburst values, which give the most average accuracy results, are 18k/18k.
- A linear relation exists between the burst value and the requested rate value. This relationship becomes apparent across tests.
- The amount of bandwidth pushed on the output interface doesn't affect the accuracy of the results.

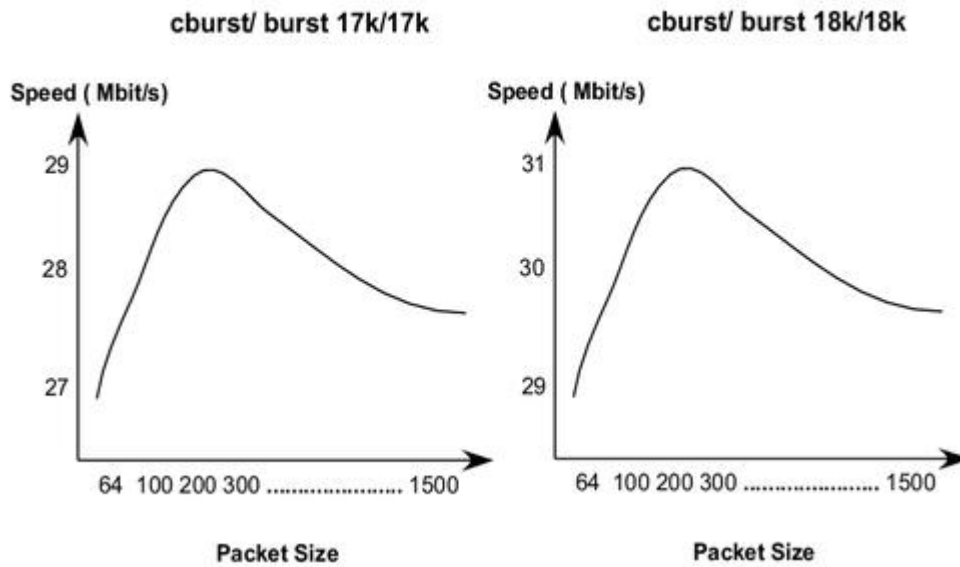


Figure 7. Graphic Analysis of Packet Size vs. Output Bandwidth

Figure 7 illustrates the relationship between packet size and output bandwidth after testing various packet sizes. From the results in Table 4 and Figure 7, we can conclude two things: throughput accuracy can be controlled by changing the cburst/burst values and the accuracy bandwidth range size is 2Mbit/s when using packets of sizes between 64 and 1,500 bytes. To verify that a linear relationship exists between the burst/cbursts values and the rate bandwidth, multiple burst and cburst values were tested. Table 5 shows the significant portion of the sampled data from the test case.

Table 5. Relationship between Burst/Cburst and Rate Values

Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)	Assigned-Rate (Mbit/s)
9k	9k	64	32	17.5	15
9k	9k	512	32	15.12	15
9k	9k	1,500	32	15.28	15
4.8k	4.8k	64	32	8.96	8
4.8k	4.8k	512	32	8.176	8
4.8k	4.8k	1,500	32	8	8
3k	3k	64	32	17.5	15
3k	3k	512	32	15.12	15

Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)	Assigned-Rate (Mbit/s)
3k	3k	1,500	32	15.28	15

18k/18k values were used as a starting point. The burst/cburst values were obtained by using the formula $\text{cburst/burst (Kbytes)} = 18/(30\text{M}/\text{Assign rate})$. From the results in Table 5, cburst/burst values can be defined dynamically for the rate value, such as when assuming a linear relationship.

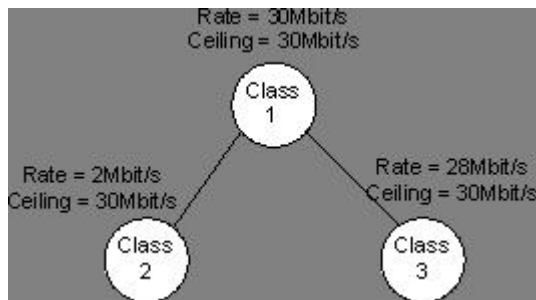


Figure 8. Test 2: Three Streams In, One Stream Out

Table 6. Test 2 Results

Stream	Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)	Class
1	17k	17k	64	15	12.7	3
2	17k	17k	512	20	17.1	3
3	1k	1k	512	4	2.01	2
Total	18k	18k	-	39	31.8	-

Table 6 shows an example of one level of inheritance. Class 2 and class 3 inherit the rate limit specification from class 1 (30Mbit/sec). In this test, the rate ceiling of the child classes is equal to the parent's rate limit, so class 2 and class 3 can borrow up to 30Mbit/sec. The linear relation assumption was used to calculate the cburst/burst values of all classes, based on their desired bandwidth.

Table 6 describes how the linear relationship works in the case of one level of inheritance. In this test, the input stream transmits continuous traffic of 39Mbit/s, and the accumulated output bandwidth is 31.8Mbit/s.

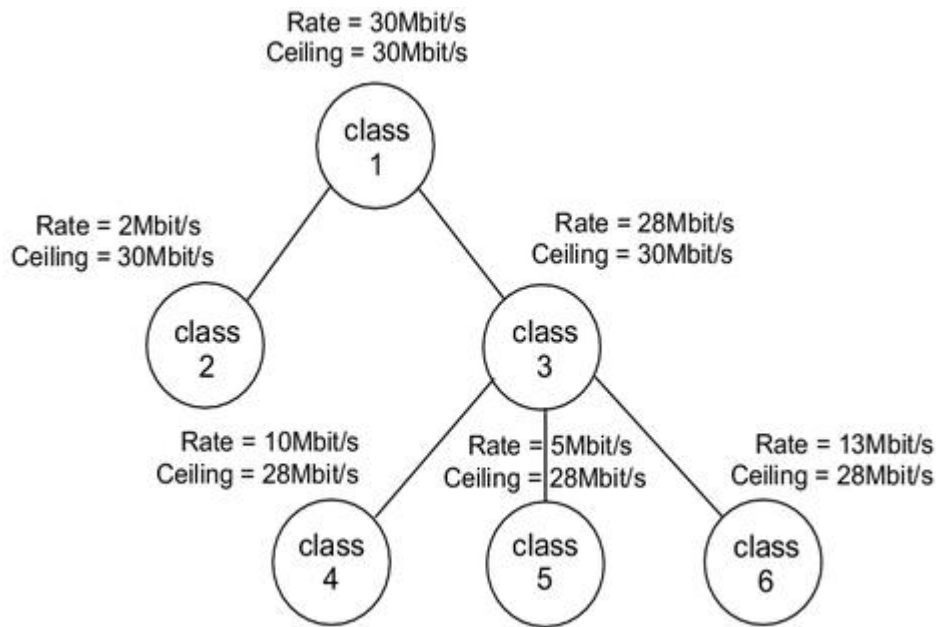


Figure 9. Test 3: Four Streams in, One Stream out

Table 7. Test 3 Results

Stream	Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)	Class
1	1k	1k	512	5	2.04	2
2	6k	6k	6	15	11.326	4
3	3k	3k	64	10	5.67	5
4	7.8k	7.8k	512	20	13.02	6
Total	18k	18k	-	50	32.05	-

Table 7 shows the case of two levels of inheritance. Class 2 and class 3 inherit the rate limit specification from class 1 (30Mbit/s). Classes 4, 5 and 6 inherit the rate limit of class 3 (28Mbit/s) and share it based on their own rate limit specifications. In this test, the rate ceiling of the child classes is equal to the parent's rate limit, so classes 4, 5 and 6 can borrow up to 28Mbit/s. The linear relation assumption was used for calculating the cburst/burst values of all classes, based on their desired bandwidth.

From the results of Table 7, it can be observed that the linear relationship works in the case of two levels of inheritance. In this test the input port transmits continuous traffic of 50Mbit/s, and the accumulated output bandwidth is 32.05Mbit/s.

Testing Model 2

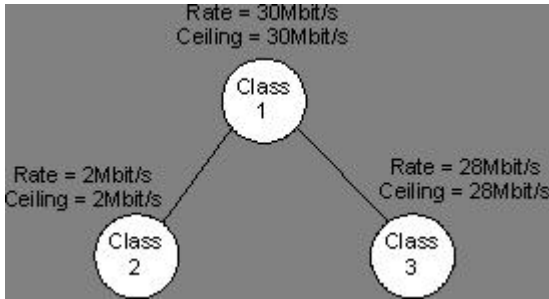


Figure 10. Test 1: Three streams In, one stream Out

Table 8. Test 1 Results

Stream	Burst (Bytes)	Cburst (Bytes)	Packet-Size (Bytes)	In-Bandwidth (Mbit/s)	Out-Bandwidth (Mbit/s)	Class
1	1k	1k	512	5	0.650	2
2	1k	1k	512	5	0.600	2
3	1k	1k	512	5	0.568	2
Total	18k	18k	-	15	1.818	-

As shown in Table 8, the bandwidth is distributed evenly when several streams are transmitting the same number of bytes and belong to the same class. Another test, in which the input bandwidth in stream 1 was higher than that of streams 2 and 3, showed that the output bandwidth of stream 1 also was higher than streams 2 and 3. From these results, it can be concluded that if more data is sent on a specific stream, the stream is able to forward more packets than other streams within the same class.

Conclusions

The test cases presented here demonstrate one way to evaluate HTB accuracy and performance. Although continuous packet bursts at a specific rate don't necessarily simulate real-world traffic, it does provide basic guidelines for defining the HTB classes and their associated attributes.

The following statements summarize the test case results:

- The maximum bandwidth that a Linux machine can forward (receive on one interface and transmit on another interface) with continuous streams of 64-byte packets is approximately 34Mbit/s. This upper limit occurs because every packet that the Ethernet driver receives or transmits

generates an interrupt. Interrupt handling occupies CPU time, and thus prevents other processes in the system from operating properly.

- When setting the traffic rate to 30Mbit/s, the cburst/burst values, which give the most average accuracy results, are 18k/18k.
- There is a linear relationship between the burst value and the requested rate. The cburst/burst values of a 30Mbit/s rate can be used as a starting point for calculating the burst values for other rates.
- It is possible to control the throughput accuracy by changing the cburst/burst values. The accuracy bandwidth range size is approximately 2Mbit/s for 64–1,500 byte packet sizes.
- Bandwidth is distributed evenly when several streams are transmitting the same number of bytes and belong to the same class.

Resources for this article: www.linuxjournal.com/article/7970.

Yaron Benita is originally from Jerusalem, Israel, and currently lives in San Francisco, California. He is the CMTS software manager at Prediwave. He works mostly in the networking and embedded fields. He is married and has a lovely six-month-old daughter. He can be reached at aronb@prediwave.com or ybenita@yahoo.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Paranoid Penguin

Book Review: *Islands in the Clickstream*

Mick Bauer

Issue #131, March 2005

When questions about computer security turn into questions about society in general, Richard Thieme takes on the hard problems.

Why do we hack? Of all the things a person can spend time on, why obsess over the myriad details involved in getting a C program to compile cleanly, a network application to communicate properly or a Web application to withstand SQL-injection attacks? Why do these things matter?

Richard Thieme, hacker sage, journalist, business and government consultant, humanist and former Episcopalian priest, has some ideas why, and he provides 336 pages worth of credible responses to this and many other important questions in his unique book *Islands in the Clickstream* (Syngress Publishing, 2004).

This is the sort of book you may not realize you want or need until you know it exists. That's true of a lot of things, many of them trivial (shrimp-flavored crackers and brass collar-stays come to mind), but there's nothing trivial about *Islands in the Clickstream*. Based on Thieme's on-line column of the same name, *Islands in the Clickstream* is about "the impact of computer technology on organizations, society, and one's own self."

If you think that sounds like a big topic, you're right. This book is composed of columns spanning the better part of a decade, and Richard still has a long way to go before exhausting his chosen subject.

Actually, though, that's a little like saying "that chap Stephen Hawking is on a roll; I hope he doesn't get bored." Because the picture Richard Thieme is trying to paint is that of the human condition itself, which nowadays happens to be inextricably entwined with technology. Each new "Islands in the Clickstream"

essay he writes adds clarity here or expands onto fresh canvas there, but the painting never can be completed, even if it were possible for one person to paint the whole thing.

Disclosure: Mick Was Already a Fan

Because I'm all about full disclosure, and because I think saying so might make me look cool anyhow, I need to tell you I go *way* back with Richard Thieme. Before I became a network security enforcer, I was a musician, and when still in music school, I had the privilege of serving as a paid singer for the Reverend Richard Thieme at St. Paul's Episcopalian Church in Milwaukee, Wisconsin. Yes, I was a professional choirboy. Think that's funny, *punk?*

Anyhow, one of the best parts of that job was hearing Richard's sermons. Each Sunday, Reverend Thieme would deliver a deep, frequently quirky and always mind-bending monologue on *what it all means*. This was seldom the expected dose of "this is what the Bible *tells* us life means." No, even back then, Richard was addressing questions like "we're all connected spiritually, but technology and the media also connect us, in ever more tangible ways; how does that change the nature of our spiritual connectedness?" That's not a quote—I wish my memory were that good—but it's the type of thing he'd talk about.

Well, imagine my surprise, almost a decade later, when I learned that one of my fellow presenters at the Root Fest 2 hackers' convention was that very same Richard Thieme. And, aside from the radical change in venue, he really hadn't changed that much. He was still working on life's more subtle yet compelling questions; the miner had merely switched mountains. I've been a big fan of his columns and speeches ever since, and I am proud to be acquainted with him personally.

Rather than simply being arranged chronologically, the essays in *Islands in the Clickstream* are distributed across the following chapters:

1. Introduction: This Is the Way the Internet Works
2. Computer-Mediated Living: The Digital Filter
3. Doing Business Digitally
4. Hacking and the Passion for Knowledge
5. Digital Spirituality
6. Mostly True Predictions
7. The Psychology of Digital Life: Identity and Destiny
8. Political Implications
9. The Dark Side of the Moon and Beyond

10. Technology Gets Personal

This is as good a way to organize Thieme's essays as any. Following Richard's suggestion, however, I've been reading them more or less at random. Although certainly there are common themes and even common observations between essays, there's no narrative per se demanding sequential reading, even within a given chapter, let alone between chapters. A single essay, whether five pages in length or one and one-half, is a completely self-contained Richard Thieme reading experience.

In this respect, and I hope Richard doesn't clobber me for saying so, *Islands in the Clickstream* is ideal bathroom reading.

So that's what the book is about and how it's organized. But what does it contain?

As readers of my Paranoid Penguin column might predict, some of my favorite bits are in Chapter 4, Hacking and the Passion for Knowledge. Here's a wonderfully representative snippet from the essay "Knowledge, Obsession, Daring" (December 26, 1998):

At a recent hacker con, I was struck—again—by the fact that hacker culture is the space in which everyone will live in the next century. Hacking is not about breaking into locked rooms. Hacking is about mapping, then exploring; or perhaps exploring, then mapping. Hacking is a mandate from evolving technologies to enter a play space characterized by limitless vistas. Properly understood, hacking in its essence is a kind of spiritual quest.

Hacking is not just hard work. It is playfulness at its very best.

Personally, I find it extremely refreshing to read such a concise and insightful redux of the hacker ethos. I get a very good feeling knowing that this sort of insight is being preached not only to the choir, so to speak, but to audiences that include business people, law enforcement officers and representatives of the federal government. Now, more than ever, the world needs hackers to continue exploring and creating, and it furthermore needs to know that that's what hackers do.

Thieme's insight goes far beyond understanding hacker culture, however. He also understands our day jobs, and better still, how they relate to larger societal issues. Consider this passage from "Lest We Forget" (November 23, 2001):

Computer security is a good metaphor for societal security because computer networks are holographic images of societies, a piece of the whole that contains the whole in symbolic form. Perimeter defense of electronic networks, we have learned, only goes so far. It's the nature of networks to subvert boundaries because networks interpenetrate one another in indeterminate ways. Nodes can belong to any of several networks the way a subway station can be a stop on any of several lines. One consequence of this is that insiders cause the great majority of security incidents, which is also a way of saying that "insiders" and "outsiders" are difficult to distinguish in a networked world. Through the use of keystroke loggers, telephone recorders, and surveillance cameras, "insiders" in electronic networks are constantly watched. Now that the United States has been attacked from the inside with its own infrastructure, there is pressure to do the same on a societal level.

I find it remarkable that Thieme sees not only subtle truths and non-instinctive fine points (and if you think his point about "inside" vs. "outside" is totally obvious, think about how rare it still is for people to take internal security very seriously), he finds common threads between these truths and fine points, and then relates them to larger truths. It's a little like having a friend who not only finds your lost car keys, but also accurately describes correlations between short-term memory loss and work-related stress, and ends up suggesting ways in which the national rates of heart disease and career dissatisfaction might both be lowered.

I don't mean to sound facetious (though, I am having a little fun here). My point is that although Richard Thieme is fascinated by hackers, network security, espionage, UFO encounters, intellectual property and many other geeky topics, what he's really trying to figure out is how all that relates to the larger human truths about who we are, why we do what we do and how our perceived and constructed realities relate to "real" reality.

As I stated earlier, a book about the hacker ethos, reality constructs and humankind's place in the universe may not be the first thing that comes to mind when you think of computer books worth purchasing. But on the other hand, that very well may be because it's never occurred to you that there was such a book. *Islands in the Clickstream* is one, probably the only one, and it may provide some crucial things you haven't been looking for but should have been.

***Islands in the Clickstream: Reflections on Life in a Virtual World* by Richard Thieme**

Publisher: Syngress Publishing, Inc., 2004

Length: 336 pages

Price: \$29.95 US, \$34.95 CAN

ISBN: 1931836221

Mick Bauer, CISSP, is *Linux Journal's* security editor and an IS security consultant in Minneapolis, Minnesota. He's the author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Linux for Suits

Migration Stories

Doc Searls

Issue #131, March 2005

Telling Doc about your IT department's Linux plans means risking your job. That's not stopping these two. What's the Big Secret that Management wants to keep quiet?

Marc Andreessen says, "All the significant trends start with technologists." If you want to know about migration from Microsoft Windows to Linux, your best stories aren't going to come from big vendors on either side, or even from CIOs. They'll come from the technologists clearing paths to their own favorite mousetraps.

Take Tyler and Dash, both from the IT department of a credit union in the Bay area of California. Together they comprise two-fifths of the the full-time IT staff. Both also are brave members of a rare breed: IT staffers whose bosses let them talk to the press. For every IT staffer with the right to talk, there are 50 or more whose jobs depend on staying mum. In fact, one IT guy I quoted on these pages last year recently told me he came within a hair's breadth of being fired for letting the world in on the Big Secret that his company was doing what approximately every other member of the Fortune 500 was also doing with Linux and open source.

I met Tyler and Dash at Apachecon in fall 2004, over breakfast with a bunch of other technologists, all with stories to tell. The major thread was migration. Because I'm always interested in stories about how Linux helps make smart companies smarter, I pressed for more. To my surprise, they were glad to give it.

About his work, Tyler said, "I'm a tech specialist. I do Perl, system administration, desktop support, whatever it takes. I also work with outside consultants on dirty work we don't want to do." Dash described his job as

“developer”, adding “I work on our in-house document management server, which is core infrastructure at a savings institution like ours.”

Dash brought the first Linux box—a public Web server—to the company. “Windows DNS and DHCP servers went out the door. Now we run ISC BIND to manage that infrastructure. Next step will be directory servers. We're looking at all the LDAP directories, commercial and noncommercial.”

“Generally we're inclined to roll our own solutions”, Tyler added.

When asked what precipitated their migration, Tyler said:

We had a number of good reasons for moving off Windows and other proprietary systems. We didn't want to sign NDAs or pay for round after round of licensing costs. We wanted open development and deployment environments. We were looking for more freedom and independence. So we made a commitment to convert to Linux and open source everywhere it made sense.

The big issue for us is compatibility everywhere. We have Windows desktops. Mac desktops. What's stopping us now are applications from vendors that use things like Microsoft Access Engine. WINE doesn't support that very well, yet. We're looking at Graphon. That way we can run a Windows server with Graphon loaded, serving applications out to a Linux terminal, thin client style. Right now we're running full clients on the teller stations. In the long run we'll have thin Linux clients everywhere that's customer-facing. We'll have the fat clients in the back office.

As with many other enterprises, “Microsoft Exchange is a big hang-up.” But, unlike many other enterprises, these guys are eager to find alternatives. Two candidates are Groupwise and Scalix, but they say they're “open to anything”.

When I asked for specifics about their server and client platforms, Tyler said, “Our Linux servers run Gentoo. We like Debian but it's releasing too slow. Gentoo is conservative, but it's current. Our personal machines are G5s running OS X, because it's UNIX. We live in a bash shell.”

One surprising statement: “There's a knowledge gap in Windows. Management is easier now, because everybody knows how to manage UNIX machines.”

A core issue from the start of Dash and Tyler's migration project has been document management. They wanted to migrate off their proprietary document management server, Dash said, “by whatever means”. The means chosen were pure Do-It-Yourself IT (DIY-IT):

We have quite a range of docs to manage: plain-text files. Scanned documents. Customer identification. We used to run something on NT Server and SQL Server. Not much flexibility there. We had to buy licenses and the client was a pain in the butt to administer. We were totally limited to what our reseller said we could work with. Disaster recovery was very difficult, especially from a cost point. You could spend the equivalent of a hundred thousand dollars just trying to fix the mess.

So we decided to start off small, with a few key features in our set. We wanted an open, Web-based app, and to operate in a browser. All HTML, almost no JavaScript, XHTML 1-compliant, with CSS (Cascading Style Sheets). As simple and cross-platform as possible, with PHP and MySQL on the back end. We started off with Linux and Mozilla as a test bed. Then we tested with Windows, using Mozilla, then with Mac and Safari. Then with Firefox on everything. The result works pretty well.

At Apachecon, I also met with Jon Walker, cofounder and CTO of Versora (versora.com), a small company from Santa Barbara, California. Versora went into business about a year ago to meet what Jon called “the demand for Windows-to-Linux migration”.

When I asked him what was driving that market, he said, “It's hard to beat zero as a licensing cost, just for starters.” He's optimistic about the company's prospects. “Given the economic appeal alone, we figure the market will be huge.”

Versora currently offers two migration products. The first is ProgressionWeb, for migrating from IIS to Apache. Among other things, it allows customers to continue running ASP (Active Server Pages, or .asp) on Apache servers. The second is ProgressionDB, announced at LinuxWorld in February 2004. ProgressionDB helps customers migrate from Microsoft SQL Server to open-source databases such as MySQL, PostgreSQL and Ingres.

In fact, Jon said Versora is hoping to win the Ingress Challenge, which will award \$1 million to “the members of the Open Source community that develop applications to convert, transform and migrate data and applications from the selected databases [Oracle, Microsoft SQL Server, et al.] to the Ingres database”. (The contest closes in February 2005, with winners announced in April 2005.) Jon, a 6' 9" former varsity college basketball player, is a competitive guy with a quiet confidence about his company's abilities. I'm not sure I'd want to bet against him.

On the other hand, I'd be willing to bet against any market for Linux-to-Windows migration.

Doc Searls is senior editor of *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

EOF

Data Center Linux at OSDL

Ibrahim Haddad

Issue #131, March 2005

If you want to get Linux into big enterprise customers' data centers, you need a working group that speaks their language. Here's how OSDL is matching data center needs to development work.

A lot is happening at the Open Source Development Labs (OSDL). The Carrier Grade Linux initiative helps Linux gain new grounds in the telecom world, and the Data Center Linux (DCL) initiative is working to accelerate the adoption of Linux in enterprise data centers. The Desktop Linux initiative aims to accelerate the adoption of Linux desktops. Plus a lot of activities, such as the higher educational forum, kernel testing, the legal defense fund, Linux advocacy to the world and much more are keeping OSDL busy. In this short article, I introduce the Data Center Linux initiative and report on its goals and ongoing work.

The DCL community is made up of vendors, users and open-source developers that interact and define a Linux road map for data centers. It's a place where people's interests meet and create a collective will to help advance Linux in this sector.

The goal is to accelerate Linux adoption in enterprise-class data centers. DCL functions as a center of gravity for developers, users, vendors and the Open Source community to work together toward a common goal: improve Linux capabilities and feature requirements to accelerate the development and adoption of Linux in the data center.

The working group has many responsibilities. It captures, discusses, publishes, develops, validates and monitors Linux capabilities needed for its adoption in enterprise data centers. The DCL Technical Capabilities v1.0 document is the work of current OSDL members. As for technical contributions, for the hackers among us, DCL has been working to identify existing open-source projects that

meet the requirements identified in the technical capabilities document and contribute or initiate open-source projects to meet the identified needs.

The working group follows open working methods collaborating with industry companies and end users to identify a list of the capabilities needed in Linux and prioritize them, very similar to the goals of other OSDL working groups. This list, then, is used as guidance for member companies and the Open Source community to help them start or refocus development efforts centered around data center capabilities to Linux.

The Technical Capabilities v1.0 document describes many capabilities categories, such as scalability, RAS (reliability, availability and serviceability), performance, manageability, clustering, standards, security and usability. Some of these categories are common to other OSDL initiatives. For this reason, OSDL has identified those common areas among its working groups and created Special Interest Groups (SIGs) where these issues are discussed independently of a specific initiative. Current SIGs include the Storage Networking SIG, which covers direct-attached storage; the Security SIG covers aspects of security; the Clustering SIG examines clustering capabilities; and the Hot Plug SIG focuses on CPU, memory, I/O bus and node hot-plug capabilities.

Several recent contributions have been made available from the DCL working group, for example, the work on hardening crash dump utilities that enable first-time failure data capture and analysis. Another example is that of persistent device naming, in particular persistent storage device naming. These contributions are significant—especially when systems (or nodes) grow and scale and interconnect in large networked environments, such capability becomes essential.

Based on the goal of the DCL initiative, the success of DCL can be realized when there is an increase in Linux deployment in the data center. At this point, the work is going at full speed. Only time will tell what the future holds for DCL.

Participation is open to anyone who wants to contribute to the DCL initiative. For more information, please visit www.osdl.org/lab_activities/data_center_linux. You will find a lot of information in addition to various documents that are available for download.

Ibrahim Haddad, contributing editor to *LJ*, is a Researcher at the Ericsson Research & Innovation Department in Montréal, Canada.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

From the Editor: March 2005 - View Source

Don Marti

Issue #131, March 2005

LAMP is great, but the first Web development trick you learned is still one of the most important.

This issue is going to give you a bunch of Web development knowledge. Please use it to help understand the next site or Web application you build. Don't just download a bunch of open-source Web code and mangle it. Clean, valid HTML makes the difference between a Web application that works like an application and one that works like a slow-loading Web site. Make your software do HTML right, and you can do more with the Web.

When the Web was new, we all learned HTML, and it was good. Then the browser wars, WYSIWYG HTML editors and the Internet boom came along. When a Web designer gets squashed between short deadlines, pixel-for-pixel "make it look like that" requirements and buggy inconsistent browsers, the result is bloated pages full of invalid HTML that tricks broken browsers into doing the right thing.

Meanwhile, on the development side, we learned the bad habit of treating HTML like PostScript—something we don't write ourselves. We gave the designers a template directory and faithfully threw their incomprehensible HTML out to the users.

But now it's time to "View Source" again and care about HTML. The Mozilla Project trashed the old Netscape 4 code and released standards-compliant versions. Konqueror got good. Even the proprietary browser vendors got up to speed with Cascading Style Sheets. All of a sudden, a Web application that was used to spewing out tons of `<td valign="top" bgcolor="#ffffaa">` could simply apply a class to the element and let the stylesheet fill in the details. Life is good again.

At least, it would be, if we would just clean up our sites and Web-based applications. One Linux site I just visited served up a page with a nice stylesheet imported in the page head, then ruined it with a bunch of unnecessary spacer images and tag attributes that override the CSS. Unfortunately, that's typical.

We can't blame the flaky browsers or the pixel-pushing designers anymore. And, it's not like we don't know HTML—we just don't care about it enough. The cause of awful Linux Web sites these days is steaming piles of code that we download, install and don't understand—then clot up with garbage tags until the site looks kind of like how we want. In order to make the Web work, you must understand the software you deploy. If you can't figure out what template or script is responsible for an element you want to change, `rm -rf` the steaming pile and start over.

When you're testing a Web application locally, or on a fast network next door to the server, it looks like it doesn't matter, and a lean, mean 12kB page seems as fast as a bloated 32kB monster. When you demo the new Web app to Management, all you get is “great job”. When the application goes live, that changes. Even if it's “internal”, other employees will be using it over a VPN from home or a café. Then it all falls apart.

You don't have to be an HTML guru, though you could do a lot worse than to read Jeffrey Zeldman's *Designing with Web Standards*. But if you want to be a Webmaster, you have to justify the “master” part by keeping site bloat under control. Save the bytes and enjoy the issue.

Don Marti is editor in chief of *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

Letters

Readers sound off.

Captive Audience and CAD

Your magazine is my absolute favorite! Over the last few years (while in prison), I have become a radical fanatic of the Linux/Open Source Movement, and since I have a captive audience, I preach the virtues and advantages of open-source apps and the power of the Linux OS to anyone that will listen. Needless to say, I have had many spirited discussions concerning that other OS while in the shower. Regardless, I'm sending out "Fresh Meat" to the streets with the Linux message.

Seriously though, I just wanted to relay my profound respect to all the hackers and followers out there whose genius and devotion has created and sustained this computing revolution.

Finally, I am interested in Computer Aided Drafting and Design programs and the formatting standards used by manufacturers. Thus, I am wondering if there are any open-source CADD programs used by industry or projects in the works, because I don't recall reading anything regarding this subject.

PS: Because I will be released very soon (3/2/05) and I want to get involved (seriously involved) with the Linux/Open Source Movement, I hope to someday meet and become friends with all of you at the *Linux Journal*.

—

Mark Allen Laliberte
(AKA Mr Linux)

Thanks for writing. There is a GPL program called QCad that looks promising. We'll look for an article on open-source CAD for a future issue. When you get out, check the Industry Events section of our Web site for conferences and tradeshows where you can meet us in person. —Ed.

What to Call the Users?

I would like to propose a temporary moratorium on the use of the terms Linux and user-friendly in the same sentence. More and more users are adopting Linux as their OS of choice because what they consider friendly is being redefined. A friend might attempt to educate you, or expect you to raise your level of awareness to realize the full potential of your friendship. But a friend would never answer your phone without permission, give personal information to strangers that come to your door or announce to the world where you hide your spare key!

Therefore, I hereby petition the editors of this, my favorite publication, to replace the term user-friendly with abuser-friendly where appropriate. As in: "A major consideration of most home PC users when considering a REAL OS, such as Linux or, on that rare occasion, some lesser alternative, is the level of abuser-friendliness...", or, "Many industry insiders are of the opinion that Linux will never be what you might call an abuser-friendly OS."

—

R. B.

Open Up Old Articles, Please

I was quite surprised at being greeted with a subscription-only viewing of articles printed in your highly esteemed magazine. Is there some way that you can open up articles in kernel land only, which are a month or two old, for general viewing, something similar to what *Linux Magazine* does?

—

anupam

Bus Reading

I've been an *LJ* subscriber for three years or so, and sometimes I take a first look at the magazine on the bus while going to work. Today, a funny thing happened to me. A person came to ask if this is a Linux-dedicated journal and where could he get it. Linux has little expression in Portugal and Linux-related publications are hard to find, so I gave him the *LJ* subscription form that comes with the magazine, hoping that one more person will learn the advantages of using a open-source OS.

—

nb

Starting Young on the Kernel Books

I thought you might get a kick out of my son Gus examining my *IA64 Linux Kernel* book. He also enjoys chewing on it, though that does make it a bit more difficult, and soggy, for me to use.



—
Dave Lloyd

Pumpkin Project

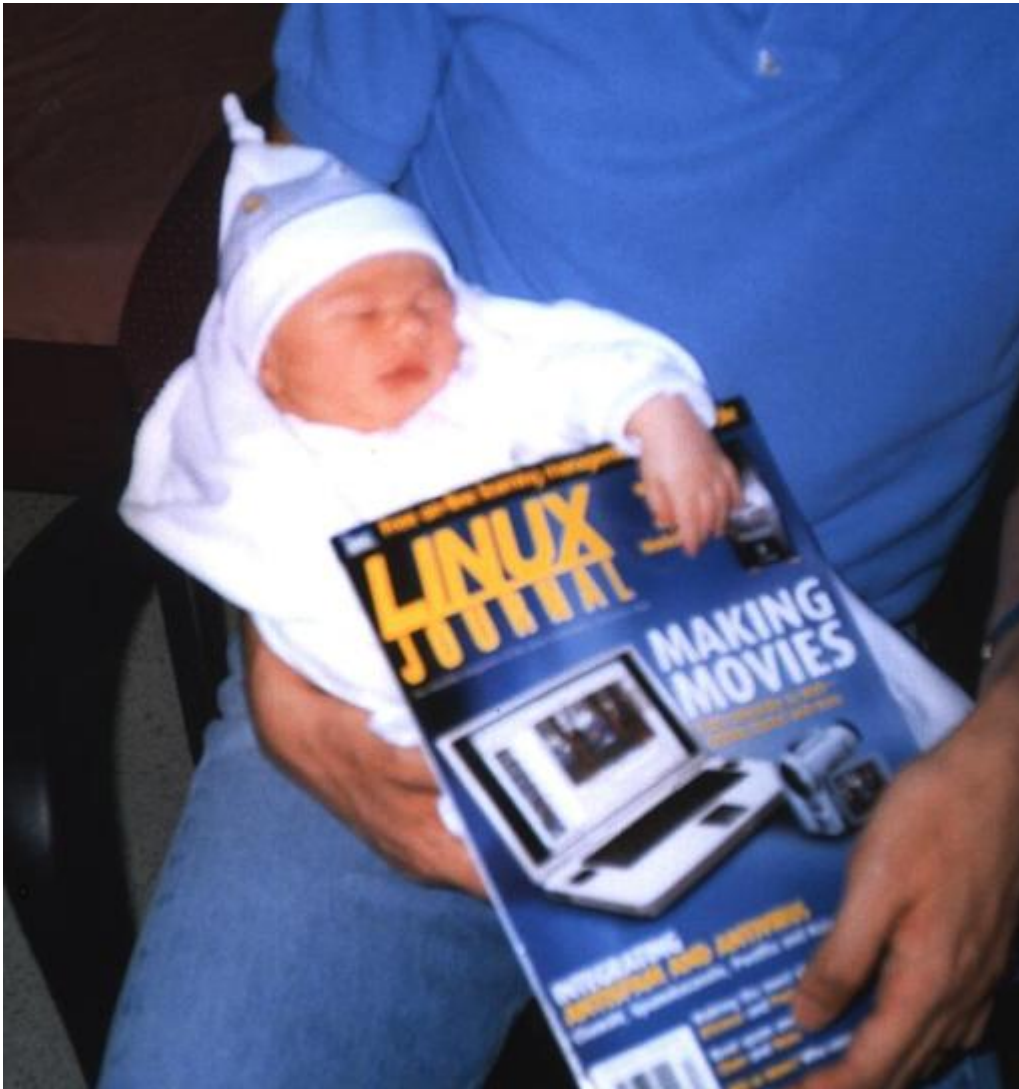
Here is my family's Tux-O-Lantern from Halloween 2004. The Tux-O-Lantern repelled bad spirits while attracting the attention of kids and Linux fans alike—like there is much of a difference between kids and Linux fans!



—
Paul

Luca's Birthday Issue

I know, I know, too much babies and Linux, but don't blame me, it's the best day of my life. This is my recently born son, named Luca, with an issue of *Linux Journal*, the day after he was born. He felt tired and decided to take a nap after browsing all the *LJ* material.



Pablo

Take-Anywhere Desktop?

The basic objective for me is a common desktop or desktop continuity; XFCE calls it persistent desktop. I want the same desktop at home and work—and there are plenty of options for solving this problem.

The most recent suggestion is a bootable mini-distro on a thumbdrive; however, that means that the hardware has to be USB-bootable. And then there are the discussions about the lifespan of the Flash thumbdrives.

Richard

If you carry a live CD for each hardware architecture you use plus a copy of your home directory on a thumbdrive, you can work even on hardware that won't boot from USB. —Ed.

Geekcorps Needs Linux Radio Gurus

In the past, you've written good articles about Geekcorps' work (www.linuxjournal.com/article/6017) and we thank you. Now, we're in the enviable position of offering volunteering opportunities to Linux professionals again, and we're wondering if you can help us spread the word.

Geekcorps (www.geekcorps.org) is looking for a few volunteers to travel to Mali, West Africa to help teach Malian radio stations and community centers how to work with audio tools and software on Linux-based systems, connect the systems by Wi-Fi over several kilometers and maintain both the systems and connections with great ingenuity and minimal expense. These volunteers are needed immediately (January–February 2005) and would stay in Mali about four months, teaching small groups in a hands-on setting. More volunteers will be needed starting in March or April to continue the experience.

Fluent French and at least 3–5 years of professional experience with advanced audio tools and/or Wi-Fi on Linux is a must. Radio broadcasting and/or Wi-Fi antenna and mast construction experience preferred. Willingness to work with people and innovate with minimal technical equipment in a developing country also is a must. Airfare, lodging, a small living stipend and dedicated in-country staff are provided.

—

Wayan Vota

Missing Code?

I'd have to agree with an earlier Letter to the Editor, please put the code samples back in the magazine. What if I'm on a plane and have no connection to the Net? I'd gladly pay an extra \$5 for my subscription.

—

Jeff Macdonald

We never took any code examples out. We merely consolidated big lists of URLs into jump pages on our Web site. The code examples are still there, and we're keeping it that way. Besides the examples that appear in print, we sometimes put a whole application on the FTP site to save you some typing. —Ed.

HP Laptop Support

I am surprised that HP will support a laptop with Linux. Every time I call in or send an e-mail for support I am told that it is not supported on any HP laptops.

—

Henry Gleason

Try it now. We got Linux phone support for the HP nx5000 on our first call from the second person we talked to. —Ed.

Photo of the Month: Antarctic Dawn

We were prepping for dive operations near the Bismark/Gerlache Straits at about 1:30 AM when my coworker Fred Stuart strolled out on deck carrying my November copy of *LJ*. I told him "Fred, only you would walk out into a beautiful Antarctic dawn reading *Linux Journal*."



—
Brent Evers

—
Fred Stuart

Photo of the month gets you a one-year subscription or a one-year extension. Photos to info@linuxjournal.com.

Send Your *What on a Postcard?*

I was astounded when I saw what was on the cover of the *Linux Journal* that I received today—a form to fill out to order the archive CD and a space to fill in credit card info and it's all conveniently mailable on a POSTCARD????

Anyone dumb enough to fill out that info on a POSTCARD and send it in would, I would think, barely have the capability to read and, unfortunately, some of them would likely blame *LJ* if and when they found out how someone had obtained their credit card number, expiration date and even their signature.

I would advise you to mention this oversight to the marketing department or whoever is responsible for this. No offense to you or any other individuals there, but it seems like an incredibly obvious oversight for no one to have thought of this before it was sent.

—

Lisa

We left out the blanks for your root password and bike-lock combination. —Ed.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

[Advanced search](#)

UpFront

- [diff -u: What's New in Kernel Development](#)
- [Kingdom of Loathing:](#)
- [On the Web](#)
- [They Said It](#)
- [First look: Ubuntu](#)
- [Web Developer Extension:](#)

diff -u: What's New in Kernel Development

Zack Brown

Issue #131, March 2005

The revision control wars have not cooled down, but they have gone fairly quiet. For the most part, this is because **BitKeeper** works much better than any free alternative, and feature requests from kernel folks are given top priority for development. In November 2004, only twice did the possibility of an alternative come up on the linux-kernel mailing list. The first time, **Andrea Arcangeli** once again presented **tla**, also known as arch, and although it did seem that various other developers also had been keeping an eye on that tool, the consensus was that tla was not yet scalable enough for the kernel.

Later on, in a completely different discussion, the **darcs** revision control tool came up. **David Roundy**, its maintainer, announced a darcs mirror of the kernel repository. This is a holy grail among revision control systems, because even a mirror of the kernel history is a very large and difficult hurdle. For now, BitKeeper is still in the far lead.

It's always great to see documentation. **Alexander Viro** recently wrote a HOWTO for **cross-compilation**. According to him, it's neither difficult nor time consuming to compile a kernel on one architecture, for use on another, and he typically tests all his changes for compile errors on builds for six different architectures: i386, x86_64, sparc32, sparc64, alpha and ppc. **Geert Uytterhoeven** had some patches to help extend this to the m68k architecture

as well, but this may be put off for a while, because currently the m68k architecture is not so well supported by the official kernel sources. Several hard-core problems confront the developers, for which no solutions have yet been found.

The 2.6 kernel development model continues to undergo transformations and clarifications under the **Linus Torvalds/Andrew Morton** maintainership duo. The new **Signed-off-by** tag, for instance, develops new wrinkles over time. Linus recently clarified that if patches are passed back and forth between developers, it is better for developers to move their Signed-off-by tag to the top, rather than have multiple copies reflecting each contribution they made. He also suggests including a CC list within each changelog entry, although he says this is unofficial, for informational purposes only, and has no technical protocol to follow. We'll see how long that lasts.

Meanwhile, a whole bunch of developers is unhappy that, as **Adrian Bunk** has said, "2.6 is currently more a development kernel than a stable kernel." These folks feel that a 2.7 development kernel should be forked off as soon as possible, and 2.6 should be allowed to stabilize. Although development is definitely more fun than maintenance, **Alan Cox** has made multiple offers to take over 2.6 maintainership, and he is known for giving a lot of weight to stability in his patches. His 2.6-ac patch series is currently the preferred source tree of a significant number of kernel developers. So far, neither Linus nor Andrew has shown any sign of handing 2.6 off to Alan or slowing down their pace of development. If anything, 2.6 development speed is only increasing at this time.

Meanwhile, Linus continues to refine his version numbering habits in response to recent criticism. His current thinking is that, for the 2.6 series, all pre-releases will be tagged rc, the latest being 2.6.10-rc3. This has the virtue of simplicity, which also motivated his decision in the 2.5 time frame to drop the -pre and -rc tags entirely and release full 2.5.x versions each time. Given the relaxation of the stable/development alternation, we'll have to see whether the 2.5 standard persists into 2.7. For the moment, the only point of clarity with regard to the kernel development model is that there is no clarity. The Linux development model is being explored, revised, carved up and mutilated—however it ends up, it clearly will be different from what it was before.

The 2.4 kernel continues to struggle toward stability. **Marcelo Tosatti** has revised his goals many times, ranging from total lock-down, to accepting certain important new features while rejecting everything else, to having a more relaxed acceptance policy in general. Ever since 2.6 came out, he has been trying to put 2.4 into deep-freeze, but with no 2.7 anywhere on the horizon, it becomes harder to reject features from 2.4. There are always folks who need

2.4's stability, with more up-to-date features; and so more and more features are backported from 2.6, and these continue to try to find their way into the official 2.4 source tree.

Recently, the **Device Mapper** subsystem was thoroughly rejected, as being too invasive for inclusion in 2.4 under any circumstances, and **iswraid**, on the other hand (with some bumps along the way), made it in under the wire. After 2.4.28, Marcelo made another valiant attempt to batten down the hatches, although, as he put it, "New drivers are okay, as long as they don't break existing setups and if a substantial amount of users will benefit from it." In particular, new drivers should be reviewed by someone knowledgeable in a specific area, he said.

Kingdom of Loathing: www.kingdomofloathing.com

Don Marti

Issue #131, March 2005

On-line games need bleeding-edge client-side software with 3-D graphics, an enormous back-end server farm and stiff subscription charges, right?

More than 100,000 people beg to differ. For the LAMP issue, here's a surprise hit on the on-line gaming scene, *Kingdom of Loathing*. The *Kingdom* is a Web-based adventure game that combines the problem solving of classic text adventures with the emergent economics and politics of Internet gaming and stick-figure graphics. It's hosted on Linux.

Don't expect your standard sword-and-sorcery or space-war plot though. Start off slaying cans of tomatoes and asparagus in the Haunted Pantry and work your way up to beating down Orcish Frat Boys, hippie chefs and other monsters. Sell your extra loot in the Flea Market or the Mall. There's even an Internet radio station where you can win in-game prizes. It's free of charge, but we highly recommend that you donate \$10 to get a powerful magic item.

On the Web

Check out the *Linux Journal* Web site this month for follow-ups on articles from this month's issue by two of our most popular regular contributors, Dave Phillips and Mick Bauer.

- After you finish Dave Phillips' article on how to use Ardour to create your own professional-grade digital audio files, read his follow-up article on our Web site. "[Further Notes on Recoding 'Talkin bout the Weather'](#)" () offers

more details about the process of creating the song Dave wrote for the Ardour article. He shares his choice of equipment for the recording process, plus the editors, sequencers and plugins he used to complete the song. If you want to hear the for-now final result, go to Dave's Web site, linux-sound.org/ardour-music.html.

- In this issue's Paranoid Penguin column, Mick Bauer reviews *Islands in the Clickstream: Reflections on Life in a Virtual World*, a book by Richard Thieme that explores some of the philosophical questions and issues that have arisen as our daily lives become more and more entwined with modern technology. Mick had the opportunity to interview Richard about some of the themes in his book; the transcript of their conversation is available [here](#). They discuss language, video games and Richard's religious background as a backdrop for considering some of the questions about the intersection of humanity and technology.

They Said It

Selling me a different brand of washing machine will not instantly fix my stupidity—even one with fewer buttons to press.

—Peter Galbavi (source: Tyler Hardison's .signature)

I think, fundamentally, open source does tend to be more stable software. It's the right way to do things. I compare it to science vs. witchcraft. In science, the whole system builds on people looking at other people's results and building on top of them. In witchcraft, somebody had a small secret and guarded it—but never allowed others to really understand it and build on it.

Traditional software is like witchcraft. In history, witchcraft just died out. The same will happen in software. When problems get serious enough, you can't have one person or one company guarding their secrets. You have to have everybody share in knowledge.

—Linus Torvalds, in *BusinessWeek*, www.businessweek.com/technology/content/aug2004/tc20040818_1593.htm

You don't build reliable bridges by refusing to let anyone see the plans.

—Alan Cox, www.itwales.com/999721.htm

First look: Ubuntu www.ubuntulinux.org

Don Marti

Issue #131, March 2005

If the idea of an up-to-date GNOME desktop environment on top of an administrator-friendly Debian base sounds good to you, then burn a Ubuntu install CD and give it a try.

Ubuntu's main strong point is a thundering herd of well-informed, helpful early adopters. As I write this in December 2004, the Linux Web sites and mailing lists are buzzing with Ubuntu questions and answers.

In the current Warty Warthog release, hardware autodetection is a notch behind the current champ, SuSE Linux Professional. Ubuntu didn't get the USB Wacom tablet or set up both heads of the dual-head video card on one test system. It did fine on a more generic PC.

Ubuntu won't try to wow you with a flashy install, so be patient through its text-based, competent first impression. When the desktop comes up, you'll get no surprise, just the same Mozilla Firefox/Novell Evolution/OpenOffice.org setup that's become the desktop Linux standard. Fans of KDE applications, such as the Konqueror file manager/Web browser and the k3b CD burner, will have to do some extra tweaks, though.

The Synaptic package manager, from Debian, is easier to use than other distributions' tools for updating software. You can get anybody started using the standard GUI apps on Linux, but Ubuntu's thoughtful choices make it easier to teach new users how to install new software.

The simplicity makes this a great distribution to carry around for lack-of-Linux emergencies you may encounter at other people's homes and businesses. When the local coffeehouse had a problem with spyware on a legacy OS running on the computer for customers, I put Ubuntu on there and gave the manager a quick tutorial. After the install, the one issue requiring a Web search to resolve was setting up printing to a printer connected to a Microsoft Windows machine.

Because you probably know what Firefox, Evolution and OpenOffice.org look like by now, here's Anna Goldberg, age 5, learning TuxPaint.

Web Developer Extension: www.chrispederick.com/work/firefox/webdeveloper

Don Marti

Issue #131, March 2005

Mozilla Firefox supports easy-to-install extensions, and one of the most useful is Chris Pederick's Web Developer Extension, which brings together many Webmasters' ideas for viewing and testing a site's look and functionality. For example, you can display all classes and IDs, as shown here, to make it easy to work on your stylesheet without viewing source on the HTML. You also can clear out cookies and HTTP authentication for your site to start a new session easily or run the W3C validator on the current page. You even can sanity-check tables with a temporary border without changing the HTML or the CSS.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

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

Advanced search

New Products

BladeRunner Cluster-in-a-box, Outblaze-SME, Xandros Desktop and more.

BladeRunner Cluster-in-a-Box

Penguin Computing announced the BladeRunner Cluster-in-a-Box server system, which integrates blade servers, Ethernet switches, storage subsystems, management software and cluster OS software in a single 4U chassis. The BladeRunner cluster comes installed with Scyld Beowulf, a distribution designed for cluster management that provides a single point of installation, login and administration. The single master node blade has dual 2.4GHz Xeon LV processors, a 2GB PC2100 DDR RAM drive and a 60GB fixed 2.5" IDE drive. The 11 slave blades also have dual Xeon LV processors and PC2100 DDR RAM drives and are PXE boot-enabled diskless nodes. BladeRunner configurations can be scaled by adding additional 4U chassis and connecting the integrated Ethernet switches, up to a 42U rack with 240 processors.

Penguin Computing, 300 California Street, Suite 600, San Francisco, California 94104, 888-736-4846, www.penguincomputing.com.



Outblaze-SME

Outblaze-SME is an e-mail platform designed for VARs targeting the small- to medium-sized enterprise (SME) market. Outblaze-SME features administration and collaboration tools that enable SMEs to purchase and allocate storage, as well as administer e-mail, calendar and file-cabinet services through a Web interface. Its collaboration tools allow employees to share calendars, contacts and files, and SME administrators can self-manage user accounts, storage,

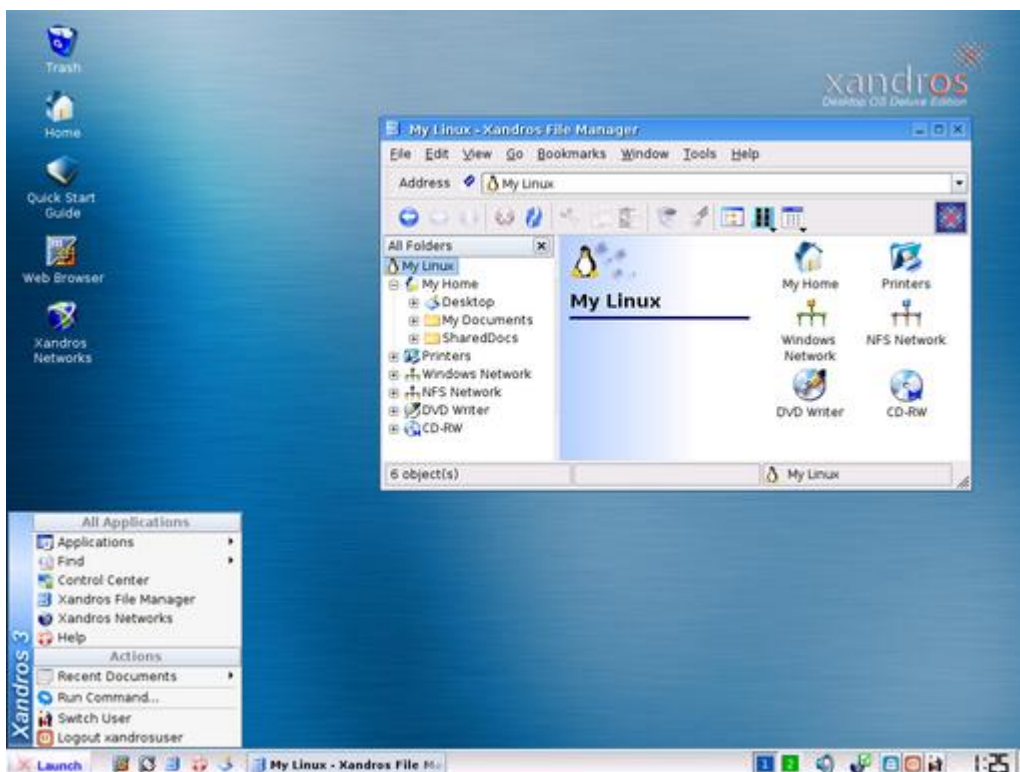
group lists and global address books. Outblaze-SME also includes POP3, IMAP4 and SMTP protocols for access to e-mail through the Web and mail clients. Outblaze-SME also comes with Outblaze's Sentry antivirus and antispam services.

Outblaze, 10 Marshall Street, Old Greenwich, Connecticut 06870, 203-286-1424, www.outblaze.com.

Xandros Desktop 3

Version 3 of the Xandros Desktop OS now is available for desktop and laptop systems. Version 3 is built on the 2.6.9 Linux kernel and includes a customized version of KDE 3.3. New features in version 3 include drag-and-drop DVD burning in Xandros File Manager, Xandros Personal Firewall, Intel Centrino and wireless card support, automatic encryption for user files, secure access PPTP VPNs, CrossOver Office 4.1 and automatic alerts to Xandros Networks updates. Xandros Desktop Version 3 enables users to drag and drop files from anywhere, including Windows network shares and FTP sites. Users also benefit from automatic spam filtering and virus protection.

Xandros Corporation, 301 Moodie Drive, Suite 200, Ottawa, Ontario K2H 9C4, Canada, 613-842-3494, www.xandros.com.



Kiwi T1x0

EmperorLinux announced a new workstation, the Kiwi T1x0, based on the Sony VAIO, models T140, T150, T160 or T170. This three-pound laptop has a 1280 ×

768 wide-aspect LCD (10.6"), which X runs in native mode. The Kiwi T150 has been certified for Fedora, Red Hat Enterprise, Debian, Slackware and SuSE. The Kiwis have 1.1GHz Pentium-M 733 CPUs with 2MB cache, 512–1,024MB of RAM, 40GB hard drives and CDRW-DVD or DVD-RW drives. The Kiwis also offer full support for X at 1280 × 768, 24bpp, i855gm; internal 10/100 land-line Ethernet; internal 802.11 a/b/g Wi-Fi Ethernet at 11–54Mbps; USB 2.0; IEEE 1394 FireWire; CardBus cards; and ACPI Hibernate. All versions of the Kiwi come with the EmperorLinux care package and one year of toll-free phone and e-mail tech support.

EmperorLinux, Inc., 900 Circle 75 Parkway, Suite 1380, Atlanta, Georgia 30339, 770-612-1205, www.emperorlinux.com.

DiskOnChip H1

M-Systems introduced a new line of DiskOnChip devices featuring up to 8GB of storage capacity, designed for use in music and video handsets. The 4GB DiskOnChip H1, the first product released, offers 90 nanometer process MLC NAND Flash, x2 technology and M-Systems' TrueFFS Flash filesystem, making it capable of managing MP3 and other multimedia files at high capacities in a single chip. The DiskOnChip H series features a legacy NOR-compatible interface, allowing it to be used with any mobile chipset. The H1 offers support for major mobile operating systems, including Symbian OS, Windows Mobile, Palm OS, Nucleus and Linux, and it is compatible with all major CPUs and multimedia processors.

M-Systems, Inc., 555 North Mathilda Avenue, Suite 220, Sunnyvale, California 94085, 408-470-4440, m-systems.com.



[Advanced search](#)

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