Advanced search

Linux Journal Issue #107/March 2003

eVote Adds Elections to Mailing Lists

Marilyn Davis

Issue #107, March 2003

Mailing lists are great for discussion, and now you can take a vote with your mailing list too.

A specialized database server for keeping votes, combined with a conferencing system, can create a medium for a true nonhierarchical democracy (see the Ideal Democracy Sidebar). The work of administering the voting system would be shared by the users and the software, not the system administrator. When the correct architecture for the problem came to me, I could no longer simply advocate Electronic Democracy; I built it.

Ideal Democracy Includes Both Discussion and Voting

The specialized vote server, the Clerk, is written in C++ and provides many features for voters. Anyone can initiate a poll, and users can change their votes as long as the poll is open. Polls also can be public, allowing all the participants to view each other's votes, private, or if_voted, allowing us to know who voted but not how they voted. Several poll types are supported: yes/no, numeric and grouped. The software is easily enhanced to add new features and poll types by extending the existing classes. The Clerk maintains data on the fly, requiring no help from the administrator. True to the promise of object-oriented architecture, the addition of each new feature has made the code more robust.

The Ballots

The ballots are dynamic. When the user closes her poll and then drops it from the conference, and when the software is otherwise idle, the ballots are collapsed, rolling the storage bytes for other items toward the top and making space for new polls. The item objects recalculate their new places in the ballots, and everything is set to continue without any intervention from the administrator.

The Clerk's Communication with the Voter

The Clerk's main() function is an infinite loop that watches for incoming messages on the interprocess communication messaging facility from eVote clients, i.e., user-interface processes with live voting users. The Clerk has one permanent message queue for incoming requests from users, and it is managed by the single instance of the InQ class. Each eVote process has a temporary message queue of its own for receiving messages from the Clerk. These are instantiations of the OutQ class.

Shared Memory

Although the statistics about polls and the personal information requested by a voter are communicated via the OutQ objects, another interprocess communication facility, shared memory, is used for slow-moving data. The list of poll items and their properties are stored in shared memory so that all the eVote clients currently active in the same conference can see them. Properties of the poll items may include public, private or if_voted, where users can see if another voter has voted, but not how he voted; yes/no or numeric; visible or hidden, where the statistics are available only after the poll is closed; single or grouped; and the title.

The conference's ItemList object is responsible for maintaining the shared memory. When a new poll is created for a conference, if the growing list of poll items requires a new patch of shared memory, a message is sent to all active eVote clients. This dynamic notification feature enables the voters to conduct their meeting in real time.

Data Files

The Clerk keeps three data files for each conference or e-mail list and one overall data file that lists all the e-mail addresses and a corresponding numerical ID. For the sample e-mail list, a sample.dat with the current ballots, keyed by the voter's numerical ID, is kept by the BallotBox object. The BallotBox also keeps sample.bnf, which contains a hash into sample.dat and some handy statistics. A sample.inf file storing the current map of items onto the ballots is maintained by the ItemList object.

eVote, the user interface, keeps a petition's signatures and comments in a flat file, one file per petition.

User Interfaces

At this time, the Clerk has two user interfaces and invites more. The Clerk's first user interface is a simple text-based Telnet interface, designed with conferencing systems and BBSes in mind.

The explosion of the Internet dampened enthusiasm for conferencing systems. E-mail arose as the dominant form of communication, and e-mail lists became the community discussion medium. So the e-mail user interface was built to allow e-mail communities to make formal decisions, while still using the popular Mailman mailing-list software.

The E-mail Interface

eVote's e-mail list interface provides three levels of participation:

1. Voter: users can vote and change their votes; they also have the same power to query the data, as does the administrator of the poll and the list's owner.
2. User/administrator: any user can initiate a poll. Under ordinary circumstances, only the user who starts a poll can close or drop it from the data.
3. List owner: some commands are password-protected. These provide overriding powers so the owner can close/drop any poll, change the voting privileges or move a participant's ballot to a new e-mail address. The list's owner also retains the same responsibilities and powers as owners of lists without eVote.

Five Executables

The overall architecture of the e-mail facility is shown in Figure 1. eVote is five programs that work together: eVote_Clerk, the Clerk; eVote_insert, the e-mail list user interface; eVote_mail, the mail administrator's utility interface; eVote_petition, the interface for signers of petitions; and eVote, the command center for controlling the Clerk.

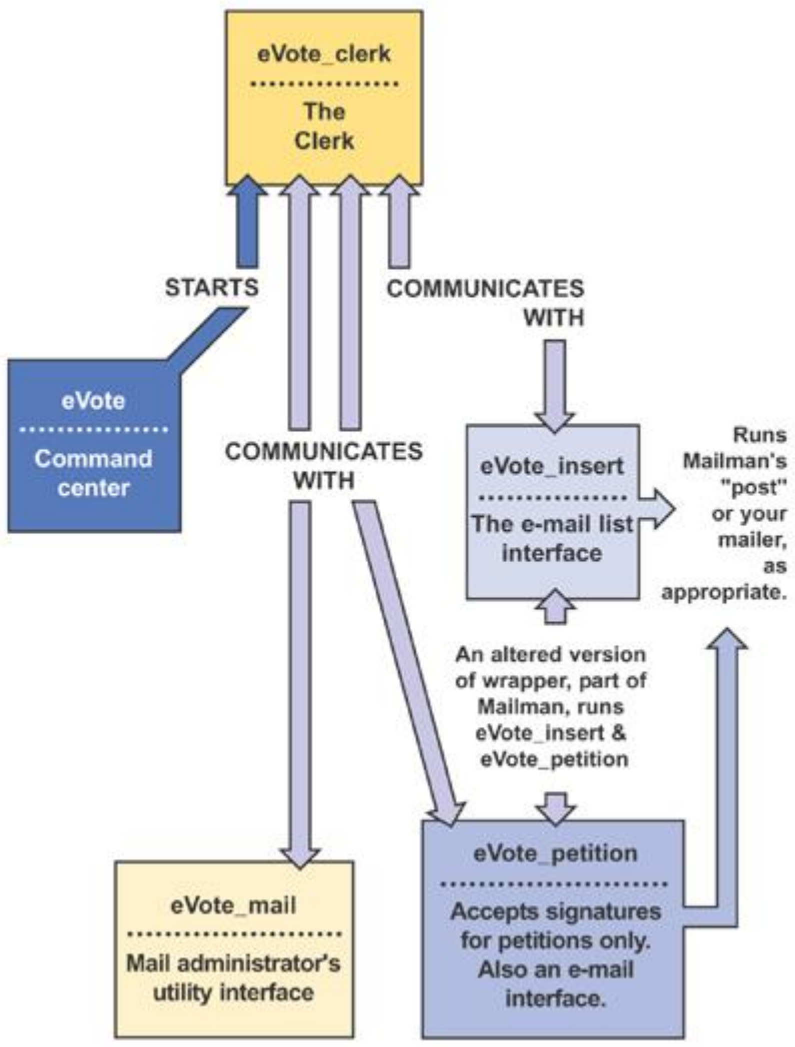

eVote_clerk
The Clerk
STARTS
COMMUNICATES WITH
eVote
Command center
COMMUNICATES WITH
eVote_insert
The e-mail list interface
Runs Mailman's "post" or your mailer, as appropriate.
An altered version of wrapper, part of Mailman, runs eVote_insert & eVote_petition
eVote_mail
Mail administrator's utility interface
eVote_petition
Accepts signatures for petitions only. Also an e-mail interface.

Figure 1. eVote's Mail Interface

eVote_Clerk runs continuously in the computer's background, establishing new polls, dropping old items, accepting, tallying, storing and reporting votes and statistics. eVote_Clerk has no direct user interface. It is started, controlled and stopped by the eVote executable.

Many Possibilities: eVote's Poll Types

The eVote_insert executable is the e-mail interface that coordinates with Mailman, the popular open-source e-mail list server. Mailman provides the discussion medium; eVote provides the voting facility. This cooperation is configured in the alias file of the mail transfer agent (MTA). Exception: if the MTA is Exim and the listserver is Mailman, Exim's configuration file handles lists and the cooperation with eVote.

eVote_mail allows the site administrator to synchronize the Clerk's list of subscribers to Mailman's list. The site administrator can use this program to block voting from a specific address or to drop an address from all lists.

Similarly, this program can delete stale messages that have been awaiting confirmation.

Two facilities are present in eVote's e-mail interface: polling in e-mail lists and petition support. The petition facility allows anyone to participate, while the e-mail list facility allows only addresses on the e-mail list to participate. Petitions are administered collaboratively by members of a petition list, which is any list whose name starts with the word petition, say, petitiona, petitionb and so on. Polls initiated in petition lists have the option of being open to nonmembers.

The eVote executable is the command center for eVote and can be called with various arguments. Depending on the argument, eVote will start, stop or check the Clerk, check and synchronize data, or flush or restart the log.

How eVote/Clerk Integrates with Mailman

Mailman can be invoked by any MTA such as sendmail, Exim or Postfix. Normally, mail directed to the e-mail list address is piped to Mailman's wrapper program to control permissions on the process and to limit the programs executed through the pipe. The wrapper then calls Mailman's post script to broadcast the mail to the list's addresses.

The alias entry for the regular Mailman list called sample might look like:

```
sample:          "|/home/mailman/bin/wrapper post
sample"
sample-admin:    "|/home/mailman/bin/wrapper mailowner
sample"
sample-request: "|/home/mailman/bin/wrapper mailcmd
sample"
sample-owner:    sample-admin
```

The mailcmd program needs a few new lines of code to tell it to send e-mail notification to eVote whenever someone successfully subscribes or unsubscribes from the list.

Mail to be broadcast to list members is piped to Mailman's post program by the sample: alias.

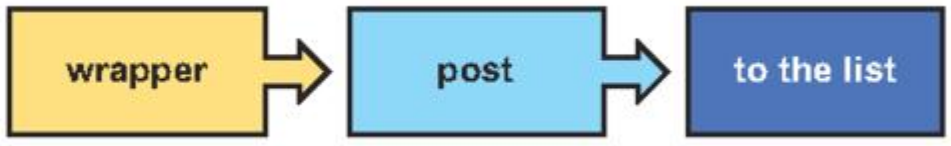


Figure 2. Mailman without eVote

eVoting is turned on by inserting eVote_insert in the pipe:

```
sample: "|/home/mailman/bin/wrapper
eVote_insert post sample"
```

Wrapper's C source code gets a few modifications so it will allow eVote_insert to be run. Now eVote gets a first look at all the mail coming into the list's broadcasting address. If the first word in the incoming message is eVote, eVote_insert intercepts the message for vote processing. Otherwise, it sends the message on to post (Figure 3).

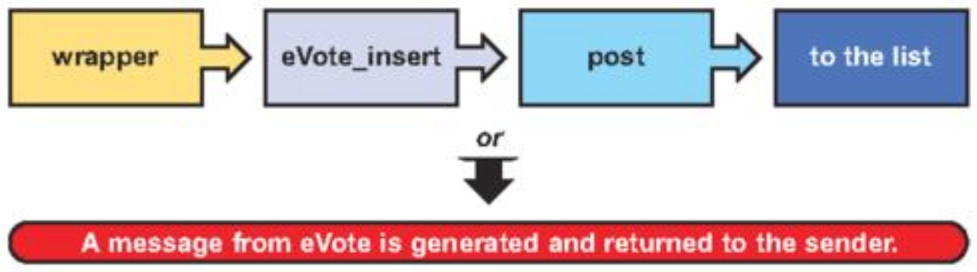

wrapper
eVote_insert
post
to the list
or
A message from eVote is generated and returned to the sender.

Figure 3. eVote Checks Mail

Petition lists are set up exactly as other eVote lists. As previously mentioned, eVote recognizes them as special because their names start with “petition”. These are intended to be used for collaborating on the administration of a petition. Members of a petition list can discuss and poll themselves, and they also have the power to set up a petition for the whole world to sign. These petitions can include any of eVote's vote types, and they always invite a comment from the signers.

The petition facility has an additional alias for receiving signatures:

```
eVote: "|/home/mailman/mail/wrapper eVote_petition"
```

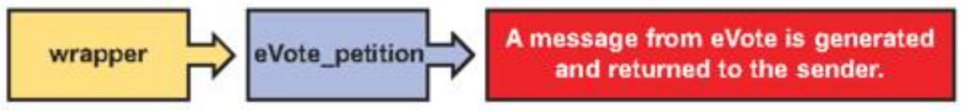

wrapper
eVote_petition
A message from eVote is generated and returned to the sender.

Figure 4. Petition Facility

The one eVote_petition alias processes signatures for all petition lists at the facility.

Two Enhancements for Elections

As it is, the eVote/Clerk system is not designed for and is not suitable for presidential elections. With two major enhancements, however, it could be the most secure and accurate solution. The required enhancements are:

1. A networking layer so networked Clerks manage the distributed data. The network would also facilitate a check on other Clerks' data and calculations. For example, if a voter votes with one Clerk and is sent a receipt from that Clerk, and later, a second receipt is sent to the voter from a different, randomly chosen Clerk, this process would ensure the integrity of all the Clerks.
2. A secure encryption layer so only the software and the voter can see that voter's ballot.

With these enhancements, the Clerk could provide more security than generalized database servers because Clerks can redundantly and geographically distribute the votes to many small computers running a GNU system. In addition, the individual administrators have minimal responsibility and minimal power, and each administrator is watched by the network of Clerks. Finally, the Clerk involves voters in facilitating not only recounts and redundant checks but revotes as well. With a system such as this one, we can go confidently into our electro-democratic age.

> **Marilyn Davis** (marilyn@deliberate.com) earned her PhD in Theoretical Radio Astronomy in the ancient past. Now she waits tables and teaches Python, C and the GNU development tools at UCSC Extension in Sunnyvale, California.

email: marilyn@deliberate.com

An Introduction to the Spambayes Project

Richie Hindle

Issue #107, March 2003

A trainable system that works with your current e-mail system to catch and filter junk mail.

The Spambayes Project is one of many projects inspired by Paul Graham's “A Plan for Spam” (www.paulgraham.com/spam.html). This famous article talks about using a statistical technique called Bayesian Analysis to identify whether an e-mail message is spam. For the full story of how the mathematics behind Spambayes works and how it has evolved, see Gary Robinson's accompanying article on page 58.

In a nutshell, the system is trained by a set of known spam messages and set of known non-spam, or “ham”, messages. It breaks the messages into tokens (words, loosely speaking) and gives each token a score according to how frequently it appears in each type of message. These scores are stored in a database. A new message is tokenized and the tokens are compared with those in the score database in order to classify the message. The tokens together give an overall score—a probability that the message is spam.

The fact that you train Spambayes by using your own messages is one of its strengths. It learns about the kinds of messages, both ham and spam, that you receive. Other spam-filtering tools that use blacklists, generic spam-identification rules or databases of known spams don't have this advantage.

The Spambayes software classifies e-mail by adding an X-Spambayes-Classification header to each message. This header has a value of spam, ham or unsure. You then use your existing e-mail software to filter based on the value of that header. We use a scale of spamminess going from 0 (ham) to 1 (spam). By default, < 0.2 means ham and > 0.9 means spam. Any e-mail between those figures is marked as unsure. You can tune these thresholds yourself; see below for information on how to configure the software.

Why Spambayes Is Different

Spambayes is different from other spam classifiers in three ways: its test-based design philosophy, its tokenizer and its classifier.

We can all think of obvious ways to identify spam: it has SHOUTING subject lines; it tells you how to Make Money Fast!!!; it purports to be from the vice president of Nigeria or his wife. It's tempting to tune any spam-classification software according to obvious rules. For instance, it should obviously be case-sensitive, because FREE is a much better spam clue than free. But the Spambayes team refused from the outset to take anything at face value. One of the earliest components of the software was a solid testing framework, which would compare new ideas against the previous version. Any idea that didn't improve the results was ditched. The results were often surprising; for instance, case sensitivity made no significant difference. This prove-it-or-lose-it approach has helped develop an incredibly accurate system, with little wasted effort.

The tokenizer does the job of splitting messages into tokens. It has evolved from simple split-on-whitespace into something that knows about the structure of messages, for instance, tagging words in the Subject line so that they are separately identified from words in the body. It also knows about their content, for instance, tokenizing embedded URLs differently from plain text. All the special rules in the tokenizer have been rigorously tested and proven to improve accuracy. This includes deliberately hiding certain tokens—for example, we strip HTML decorations and ignore most headers by default. Surprising decisions, but they're backed up by testing.

The classifier is the statistical core of Spambayes, the number cruncher. This has evolved a great deal since its beginnings in Paul Graham's article, again through test-based development. Gary's article, “A Statistical Approach to the Spam Problem” (page 58), covers the classifier in detail.

Requirements and Installation

The Spambayes software is available for download from sf.net/projects/spambayes. It requires Python 2.2 or above and version 2.4.3 or above of the Python e-mail package. If you're running Python 2.2.2 or above, you should already have this. If not, you can download it from mimelib.sf.net and install it: unpack the archive, **cd** to the email-2.4.3 directory and type **setup.py install**. This will install it in your Python site-packages directory. You'll also need to move aside the standard e-mail library; go to your Python Lib directory, and rename the file email as email_old.

Keeping up to Date

Because the project is in constant development, things are sure to change between my writing this article and the magazine hitting the newsstand. I'll publish a summary of any major changes on an Update page at www.entrian.com/spambayes.

Some of the things we're working on as I write this article include more flexible command-line training; enabling integration with more e-mail clients, such as Mutt; web-based configuration; security features for the web interface; and easier installation. I'll provide full details of these items on the Update page.

Components

Three classifier programs are in the Spambayes software: a procmail filter, a POP3 proxy and a plugin for Microsoft Outlook 2000. I cover the procmail filter and the POP3 proxy in this article. A web interface (covered below) and various command-line utilities, test harnesses and so on are also part of Spambayes; see the documentation that comes with the software for full details.

Procmail-Based Setup

If you use a procmail-based e-mail system, this is how the Spambayes procmail system works:

- All your existing mail has a new X-Spambayes-Trained header. The software uses this to keep track of which messages it has already learned about.
- The software looks at all your incoming mail. Messages it thinks are spam are put in a “spam” mail folder. Everything else is delivered normally.
- Every morning, it goes through your mail folders and trains itself on any new messages. It also picks up mail that's been refiled—something it thought was ham but was actually spam and vice versa. Be sure to keep spam in your spam folder for at least a day or two before deleting it. We suggest keeping a few hundred messages, in case you need to retrain the software.

You'll need a working crond to set up the daily training job. Optionally, you can have a mailbox of spam and a mailbox of ham to do some initial training.

To set up Spambayes on your procmail system, begin by installing the software. I'll assume you've put it in $HOME/src/spambayes. Then, create a new database:

```
$HOME/src/spambayes/hammiefilter.py -n
```

If you exercise the option to train Spambayes on your existing mail, type:

```
    $HOME/src/spambayes/mboxtrain.py \
    -d $HOME/.hammiedb -g $HOME/Mail/inbox \
    -s $HOME/Mail/spam
```

You can add additional folder names if you like, using -g for good mail folders and -s for spam folders. Next, you need to add the following two recipes to the top of your .procmailrc file:

```
    :0fw
    | $HOME/src/spambayes/hammiefilter.py
    :0
    * ^X-Spambayes-Classification: spam
    $HOME/Maildir/.spam/
```

The previous recipe is for the Maildir message format. If you need mbox (the default on many systems) or MH, the second recipe should look something like this:

```
    :0:
    * ^X-Spambayes-Classification: spam
    $HOME/Mail/spam
```

If you're not sure what format you should use, ask your system administrator. If you are the system administrator, check the documentation of your mail program. Most modern mail programs can handle both Maildir and mbox.

Using **crontab -e**, add the following cron job to train Spambayes on new or refiled messages every morning at 2:21 **AM**:

```
    21 2 * * * $HOME/src/spambayes/mboxtrain.py -d
    $HOME/.hammiedb -g $HOME/Mail/inbox
    -s $HOME/Mail/spam
```

You also can add additional folder names here. It's important to do this if you regularly file mail in different folders; otherwise Spambayes never learns anything about those messages.

Spambayes should now be filtering all your mail and training itself on your mailboxes. But occasionally a message is misfiled. Simply move that message to the correct folder, and Spambayes learns from its mistake the next morning.

Many thanks to Neale Pickett for the information in this section.

Setting Up the POP3 Proxy and the Web Interface

If you don't use Procmail or don't want to mess with it, or if you want to set up the software on a non-UNIX machine, you can use the POP3 proxy. This is a middleman that sits between your POP3 server and your e-mail program, and it adds an X-Spambayes-Classification header to e-mails as you retrieve them.

You also can use the POP3 proxy with Fetchmail; simply reconfigure Fetchmail to talk to the POP proxy rather than your real POP3 server.

The web interface lets you pretrain the system, classify messages and train on messages received via the POP3 proxy, all through your web browser. The software is configured through a file called bayescustomize.ini. This is true of the Procmail filter as well. There's no need to change any of the defaults to use it out-of-the-box, but the POP3 proxy needs to be set up with the details of your POP3 server. All the available options and their defaults live in a file called Options.py, but you need to look at that only if you're terminally curious or want to do advanced tuning. The minimum you need to do is create a bayescustomize.ini file like this:

```
[pop3proxy]
pop3proxy_servers: pop3.example.com
```

where *pop3.example.com* is wherever you currently have your e-mail client configured to collect mail. The proxy runs on port 110 by default. This is fine on non-UNIX platforms, but on UNIX you'll want to use a different one by adding this line:

```
pop3proxy_ports: 1110
```

to the [pop3proxy] section of bayescustomize.ini. If you collect mail from more than one POP3 server, you can provide a list of comma-separated addresses in pop3proxy_servers and a corresponding list of comma-separated port numbers in pop3proxy_ports. Each port proxies to the corresponding POP3 server.

You can now run pop3proxy.py. This prints some status messages, which should include something like:

```
Listener on port 1110 is
    proxying pop3.example.com:110
User interface url is http://localhost:8880
```

This means the proxy is ready for your e-mail client to connect to it on port 1110, and the web interface is ready for you to point your browser at the given URL. To access the web interface from a different machine, replace localhost with the name of the machine running pop3proxy.py.

Classifying Your E-mail Using the POP3 Proxy

You now need to configure your e-mail client to collect mail from the proxy rather than from your POP3 server. Where you currently have pop3.example.com, port 110, set up as your POP3 server, you need to set it to localhost, port 1110. If you're running the proxy on a different machine from your e-mail client, use *machinename*, port 1110.

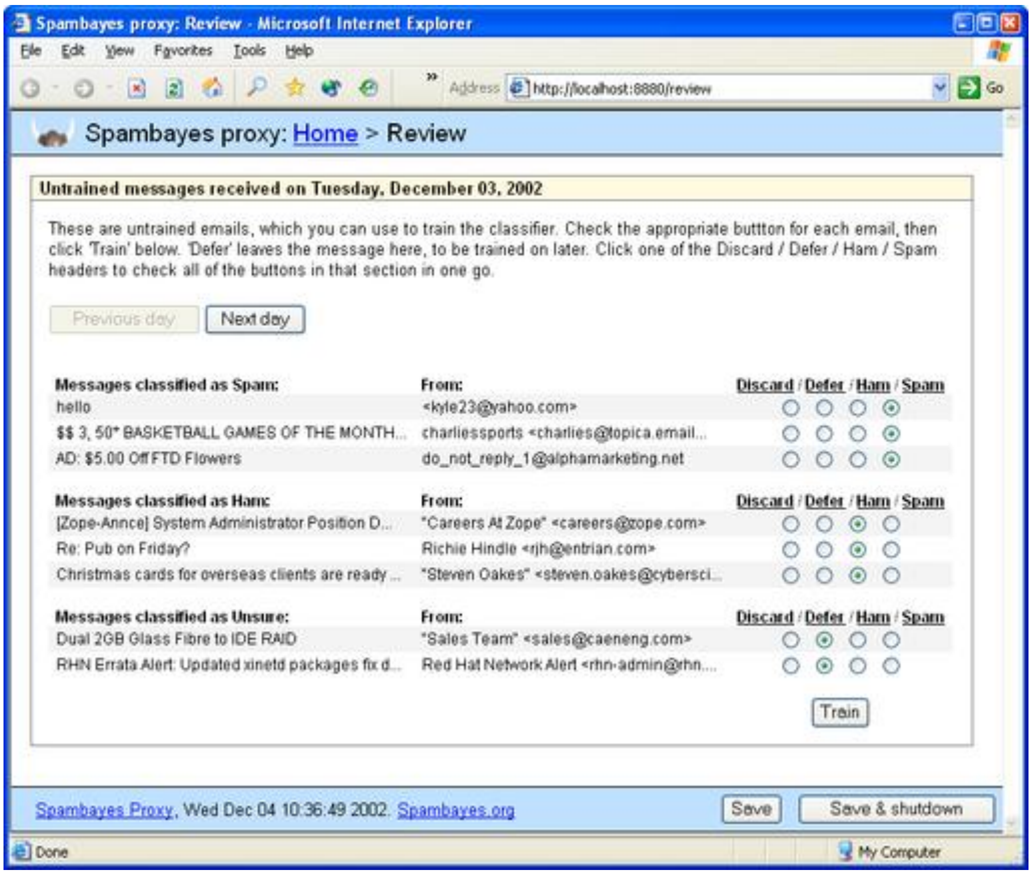
Classifying your mail is now as easy as clicking “Get new mail”. The proxy adds an X-Spambayes-Classification header to each message, and you can set up a filter in your mail program to file away suspected spam in its own folder. Until you do some training, however, all your messages are classified as unsure.

Once you're up and running, you should check your suspected spam folder periodically to see whether any real messages slip through, so-called false positives. As you train the system, this will happen less and less often.

Training through the Web Interface

Initial training isn't an absolute requirement, but you'll get better results from the outset if you do it. You can use the upload a message or mbox file form to train via the web interface, either on individual messages or UNIX mbox files.

Once you're up and running, you can use the web interface to train the system on the messages the POP3 proxy has seen. The Review messages page lists your messages, classified according to whether the software thought they were spam, ham or unsure. You can correct any mistakes by checking the boxes and then clicking Train. After a couple of days (depending on how much e-mail you get), there'll be very few mistakes.



Spambayes proxy: Review - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8880/review Go

Spambayes proxy: Home > Review

Untrained messages received on Tuesday, December 03, 2002

These are untrained emails, which you can use to train the classifier. Check the appropriate buttton for each email, then click 'Train' below. 'Defer' leaves the message here, to be trained on later. Click one of the Discard / Defer / Ham / Spam headers to check all of the buttons in that section in one go.

Previous day | Next day

Messages classified as Spam:	From:	Discard / Defer / Ham / Spam
hello	<kyle23@yahoo.com>	○ ○ ○ ●
$$ 3, 50* BASKETBALL GAMES OF THE MONTH...	charliessports <charlies@topica.email...	○ ○ ○ ●
AD: $5.00 Off FTD Flowers	do_not_reply_1@alphamarketing.net	○ ○ ○ ●
Messages classified as Ham:	**From:**	**Discard / Defer / Ham / Spam**
[Zope-Annce] System Administrator Position D...	"Careers At Zope" <careers@zope.com>	○ ○ ● ○
Re: Pub on Friday?	Richie Hindle <rjh@entrian.com>	○ ○ ● ○
Christmas cards for overseas clients are ready ...	"Steven Oakes" <steven.oakes@cybersci...	○ ○ ● ○
Messages classified as Unsure:	**From:**	**Discard / Defer / Ham / Spam**
Dual 2GB Glass Fibre to IDE RAID	"Sales Team" <sales@caeneng.com>	○ ● ○ ○
RHN Errata Alert: Updated xinetd packages fix d...	Red Hat Network Alert <rhn-admin@rhn....	○ ● ○ ○

Train

Spambayes Proxy, Wed Dec 04 10:36:49 2002. Spambayes.org | Save | Save & shutdown

Figure 1. Spambayes Proxy Web Training Page

Training Tips

Spambayes does an excellent job of classifying your mail, but it's only as good as the data on which you train it. Here are some tips to help you get the best results:

- Don't train on old mail. The characteristics of your e-mail change over time—sometimes subtly, sometimes dramatically—so it's best to use recent mail.
- Take care when training. If you mistakenly train a spam message as ham, or vice versa, it will throw off the classifier.
- Try to train on roughly as much spam as ham. This isn't critical, but you'll get better results with a fair balance.

Possible Future Directions

The Spambayes software is in constant development. Many people are involved, and we have many ideas about what to do next. Here's a taste of where the project might go:

- Improving the tokenizer and classifier as new research reveals more accurate ways to classify spam.
- Intelligent autotraining: once the system is up and running, it should be possible for it to keep itself up-to-date by training itself, with users correcting only the odd mistake. We're already doing something along these lines with the Procmail system, but we're looking at ways of making it more automated and compatible with all platforms.
- SMTP proxy: to train the system from any e-mail client on any platform, you could send a message to a special ham or spam address. This could be a simple way to correct classification mistakes, and it would combine well with intelligent auto-training techniques.
- Database reduction: the more you train the system, the larger its database gets. We're looking at ways to keep the database size down.
- Integration with spam-reporting tools: the web interface and the e-mail plugins could let you report spams to systems like Vipul's Razor and Pyzor.
- More e-mail client integration: we already have the Outlook plugin, and we'd like to integrate with more e-mail clients. The POP3 proxy and the web interface work well with any e-mail client, but having a Delete as Spam button right there in your e-mail client is much more convenient than switching to your web browser.
- Better documentation: we aim to publish documentation on how to set up Spambayes on all the popular platforms and e-mail clients.

By the time this article is in print, some of these things already may be happening; see my Update page at www.entrian.com/spambayes for details.

email: richie@entrian.com

Richie Hindle is a professional software engineer in the UK. He works full-time writing business intelligence software, and in his spare time he works on Spambayes and his own Python projects at www.entrian.com. He only occasionally wears a silly hat.

Archive Index Issue Table of Contents

Advanced search

A Statistical Approach to the Spam Problem

Gary Robinson

Issue #107, March 2003

Using Bayesian statistics to detect an e-mail's spamminess.

Most people are spending significant time daily on the task of distinguishing spam from useful e-mail. I don't think I'm alone in feeling that we have better things to do. Spam-filtering software can help.

This article discusses one of many possible mathematical foundations for a key aspect of spam filtering—generating an indicator of “spamminess” from a collection of tokens representing the content of an e-mail.

The approach described here truly has been a distributed effort in the best open-source tradition. Paul Graham, an author of books on Lisp, suggested an approach to filtering spam in his on-line article, “A Plan for Spam”. I took his approach for generating probabilities associated with words, altered it slightly and proposed a Bayesian calculation for dealing with words that hadn't appeared very often. Then I suggested an approach based on the chi-square distribution for combining the individual word probabilities into a combined probability (actually a pair of probabilities—see below) representing an e-mail. Finally, Tim Peters of the Spambayes Project proposed a way of generating a particularly useful spamminess indicator based on the combined probabilities. All along the way the work was guided by ongoing testing of embodiments written in Python by Tim Peters for Spambayes and in C by Greg Louis of the Bogofilter Project. The testing was done by a number of people involved with those projects.

Generating Word Probabilities

We will assume the existence of a body of e-mails (the corpus) for training, together with software capable of parsing each e-mail into its constituent words. We will further assume that each training e-mail has been classified manually as either “ham” (the e-mail you want to read) or “spam” (the e-mail you don't). We will use this data and software to train our system by generating a probability for each word that represents its spamminess.

For each word that appears in the corpus, we calculate:

- $b(w)$ = (the number of spam e-mails containing the word w) / (the total number of spam e-mails).
- $g(w)$ = (the number of ham e-mails containing the word w) / (the total number of ham e-mails).
- $p(w) = b(w) / (b(w) + g(w))$

$p(w)$ can be roughly interpreted as the probability that a randomly chosen e-mail containing word *w* will be a spam. Spam-filtering programs can compute $p(w)$ for every word in an e-mail and use that information as the basis for further calculations to determine whether the e-mail is ham or spam.

However, there is one notable wrinkle. In the real world, a particular person's inbox may contain 10% spam or 90% spam. Obviously, this will have an impact on the probability that, for that individual, an e-mail containing a given word will be spam or ham.

That impact is ignored in the calculation above. Rather, that calculation, in effect, approximates the probabilitily that a randomly chosen e-mail containing *w* would be spam in a world where half the e-mails were spams and half were hams. The merit of that approach is based on the assumption that we don't want it to be harder or easier for an e-mail to be classified as spam just because of the relative proportion of spams and hams we happen to receive. Rather, we want e-mails to be judged purely based on their own characteristics. In practice, this assumption has worked out quite well.

Dealing with Rare Words

There is a problem with probabilities calculated as above when words are very rare. For instance, if a word appears in exactly one e-mail, and it is a spam, the calculated $p(w)$ is 1.0. But clearly it is not absolutely certain that all future e-mail containing that word will be spam; in fact, we simply don't have enough data to know the real probability.

Bayesian statistics gives us a powerful technique for dealing with such cases. This branch of statistics is based on the idea that we are interested in a person's degree of belief in a particular event; that is the Bayesian probability of the event.

When exactly one e-mail contains a particular word and that e-mail is spam, our degree of belief that the next time we see that word it will be in a spam is not 100%. That's because we also have our own background information that guides us. We know from experience that virtually any word can appear in either a spam or non-spam context, and that one or a handful of data points is not enough to be completely certain we know the real probability. The Bayesian approach lets us combine our general background information with the data we have collected for a word in such a way that both aspects are given their proper importance. In this way, we determine an appropriate degree of belief about whether, when we see the word again, it will be in a spam.

We calculate this degree of belief, *f*(*w*), as follows:

$$f(w) = \frac{(s * x) + (n * p(w))}{s + n}$$

Equation 1

where:

- *s* is the strength we want to give to our background information.
- *x* is our assumed probability, based on our general backround information, that a word we don't have any other experience of will first appear in a spam.
- *n* is the number of e-mails we have received that contain word *w*.

This gives us the convenient use of *x* to represent our assumed probability from background information and *s* as the strength we will give that assumption. In practice, the values for *s* and *x* are found through testing to optimize performance. Reasonable starting points are 1 and .5 for *s* and *x*, respectively.

We will use *f*(*w*) rather than *p*(*w*) in our calculations so that we are working with reasonable probabilities rather than the unrealistically extreme values that can often occur when we don't have enough data. This formula gives us a simple way of handling the case of no data at all; in that case, *f*(*w*) is exactly our assumed probability based on background information.

Those who are interested in the derivation of the above formula, read on; others may want to skip down to the next section.

If you are already familiar with the principles of Bayesian statistics, you will probably have no trouble understanding the derivation. Otherwise, I suggest that you read sections 1 and 2 of David Heckerman's “A Tutorial on Learning with Bayesian Networks” (see Resources) before continuing.

The formula above is based on assuming that spam/ham classification for e-mails containing word *w* is a binomial random variable with a beta distribution prior. We calculate the posterior expectation after incorporating the observed data. We are going to do a test that is the equivalent of the “experiment” of tossing a coin multiple times to see whether it appears to be biased. There are *n* trials. If we were tossing a coin, each coin toss would be a trial, and we'd be counting the number of heads. But in our case, the trial is looking at the next e-mail in the training corpus that contains the word “porn” and seeing whether the e-mail it contains is a spam. If it is, the experiment is considered to have been successful. Clearly, it's a binomial experiment: there are two values, yes or no. And they are independent: the fact that one e-mail contains “porn” isn't correlated to the question of whether the next one will. Now, it happens that if you have a binomial experiment and assume a beta distribution for the prior, you can express the expectation that the *n*th + 1 trial will be successful as shown in Equation 2.

$$E = \frac{u+q}{u+v+n}$$

Equation 2

- *q* is the number of successes.
- *n* is the number of trials.
- *u* and *v* are the parameters of the beta distribution.

We want to show that Equation 1 is equivalent to Equation 2. First perform the following substitutions:

$s = u + v$ $s * x = u$

The next step is to replace *q* with *n* * *p*(*w*). We have already discussed the fact that *p*(*w*) is an approximation of the probability that a randomly chosen e-mail containing *w* is spam in an imaginary world where there are the same number of spams as hams. So *n* * *p*(*w*) approximates the count of spams containing *w* in that world. This approximates the count of successes in our experiment and is therefore the equivalent of *q*. This completes our demonstration of the equivalence of Equations 1 and 2.

In testing, replacing *p*(*w*) with *f*(*w*) in all calculations where *p*(*w*) would otherwise be used, has uniformly resulted in more reliable spam/ham classification.

Combining the Probabilities

At this point, we can compute a probability, *f*(*w*), for each word that may appear in a new e-mail. This probability reflects our degree of belief, based on our background knowledge and on data in our training corpus, that an e-mail chosen randomly from those that contain *w* will be a spam.

So each e-mail is represented by a set of of probabilities. We want to combine these individual probabilities into an overall combined indicator of spamminess or hamminess for the e-mail as a whole.

In the field of statistics known as meta-analysis, probably the most common way of combining probabilities is due to R. A. Fisher. If we have a set probabilities, *p*1, *p*2, ..., *p*n, we can do the following. First, calculate -2ln *p*1 * *p*2 * ... * *p*n. Then, consider the result to have a chi-square distribution with 2*n* degrees of freedom, and use a chi-square table to compute the probability of getting a result as extreme, or more extreme, than the one calculated. This “combined” probability meaningfully summarizes all the individual probabilities.

The initial set of probabilities are considered to be with respect to a null hypothesis. For instance, if we make the null hypothesis assumption that a coin is unbiased, and then flip it ten times and get ten heads in a row, there would be a resultant probability of (1/2)10 = 1/1024. It would be an unlikely event with respect to the null hypothesis, but of course, if the coin actually were to be biased, it wouldn't necessarily be quite so unlikely to have an extreme outcome. Therefore, we might reject the null hypothesis and instead choose to accept the alternate hypothesis that the coin is biased.

We can use the same calculation to combine our *f*(*w*)s. The *f*(*w*)s are not real physical probabilities. Rather, they can be thought of as our best guess about those probabilities. But in traditional meta-anaytical uses of the Fisher calculation they are not known to be the real probabilities either. Instead, they are assumed to be, as part of the null hypothesis.

Let our null hypothesis be “The *f*(*w*)s are accurate, and the present e-mail is a random collection of words, each independent of the others, such that the *f*(*w*)s are not in a uniform distribution.” Now suppose we have a word, “Python”, for which *f*(*w*) is .01. We believe it occurs in spams only 1% of the time. Then to the extent that our belief is correct, an unlikely event occurred; one with a probability of .01. Similarly, every word in the e-mail has a probability associated with it. Very spammy words like porn might have probabilities of .99 or greater.

Then we use the Fisher calculation to compute an overall probability for the whole set of words. If the e-mail is a ham, it is likely that it will have a number of very low probabilities and relatively few very high probabilities to balance them, with the result that the Fisher calculation will give a very low combined probability. This will allow us to reject the null hypothesis and assume instead the alternative hypothesis that the e-mail is a ham.

Let us call this combined probability *H*:

$$H = C^{-1}(-2 \ln \prod_{w} f(w), 2n)$$

Equation 3

where *C*-1() is the inverse chi-square function, used to derive a p-value from a chi-square-distributed random variable.

Of course, we know from the outset the null hypothesis is false. Virtually no e-mail is actually a random collection of words unbiased with regard to hamminess or spamminess; an e-mail usually has a telling number of words of one type or the other. And certainly words are not independent. E-mails containing “sex” are more likely to contain “porn”, and e-mails containing the name of a friend who programs Python are more likely also to contain “Python”. And, the *f*(*w*)s are not in a uniform distribution. But for purposes of spam detection, those departures from reality usually work in our favor. They cause the probabilities to have a nonrandom tendency to be either high or low in a given e-mail, giving us a strong statistical basis to reject the null hypothesis in favor of one alternative hypothesis or the other. Either the e-mail is a ham or it is a spam.

So the null hypothesis is a kind of straw dog. We set it up only in order for it to be knocked down in one direction or the other.

It may be worthwhile to mention here a key difference between this approach and many other combining approaches that assume the probabilities are independent. This difference applies, for example, to such approaches as the “Bayesian chain rule” and “Naïve Bayesian classification”. Those approaches have been tested in head-to-head combat against the approach described here using large numbers of human-classified e-mails and didn't fare as well (i.e., didn't agree as consistently with human judgment).

The calculations for those approaches are technically invalid due to requiring an independence of the data points that is not actually present. That problem does not occur when using the Fisher technique because the validity of the calculations doesn't depend on the data being independent. The Fisher calculation is structured such that we are rejecting a null hypothesis that includes independence in favor of one of the two alternative hypotheses that are of interest to us. These alternative hypotheses each involve non-independence, such as the correlation between “sex” and “porn”, as part of the phenomenon that creates the extreme combined probabilities.

These combined probabilities are only probabilities under the assumption that the null hypothesis—known to be almost certainly false—is true. In the real world, the Fisher combined probability is not a probability at all, but rather an abstract indicator of how much (or little) we should be inclined to accept the null hypothesis. It is not meant to be a true probability when the null hypothesis is false, so the fact that it isn't doesn't cause computational problems for us. The naïve Bayesian classifier is known to be resistant to distortions due to a lack of independence, but it doesn't avoid them entirely.

Whether these differing responses to a lack of independence in the data are responsible for the superior performance of Fisher in spam/ham classification testing to date is not known, but it is something to consider as one possible factor.

The individual *f*(*w*)s are only approximations to real probabilities (i.e., when there is very little data about a word, our best guess about its spam probability as given by *f*(*w*) may not reflect its actual reality). But if you consider how *f*(*w*) is calculated, you will see that this uncertainty diminishes asymptotically as *f*(*w*) approaches 0 or 1, because such extreme values can be achieved only by words that have occurred quite frequently in the training data and either almost always appear in spams or almost always appear in hams. And, conveniently, it's the numbers near 0 that have by far the greatest impact on the calculations. To see this, consider the influence on the product .01 * .5 if the first term is changed to .001, vs. the influence on the product .51 *.5 if the first term is changed to .501, and recall that the Fisher technique is based on multiplying the probabilities. So our null hypothesis is violated by the fact that the *f*(*w*)s are not completely reliable, but in a way that matters vanishingly little for the words of the most interest in the search for evidence of hamminess: the words with *f*(*w*) near 0.

Alert readers will wonder at this point: “Okay, I understand that these Fisher calculations seem to make sense for hammy words with correspondingly near-0 probabilities, but what about spammy words with probabilities near 1?” Good question! Read on for the answer that will complete our discussion.

The Indicator of Hamminess or Spamminess

The calculation described above is sensitive to evidence of hamminess, particularly when it's in the form of words that show up in far more hams than spams. This is because probabilities near 0 have a great influence on the product of probabilities, which is at the heart of Fisher's calculation. In fact, there is a 1971 theorem that says the Fisher technique is, under certain circumstances, as powerful as any technique can possibly be for revealing underlying trends in a product of possibilities (see Resources).

However, very spam-oriented words have $f(w)$s near 1, and therefore have a much less significant effect on the calculations. Now, it might be assumed that this is a good thing. After all, for many people, misclassifying a good e-mail as spam seems a lot worse than misclassifying a bad e-mail as a ham, because no great harm is done if a single spam gets through but significant harm might result from a single good e-mail being wrongly classified as spam and therefore ignored by the recipient. So it may seem good to be sensitive to indications of hamminess and less sensitive to indications of spamminess.

However, there are ways to deal with this problem that in real-world testing do not add a noticeable tendency to wrongly classify good e-mail as spam, but do significantly reduce the tendency to misclassify spam as ham.

The most effective technique that has been identified in recent testing efforts follows.

First, “reverse” all the probabilities by subtracting them from 1 (that is, for each word, calculate 1 - $f(w)$). Because $f(w)$ represents the probability that a randomly chosen e-mail from the set of e-mails containing *w* is a spam, 1 - $f(w)$ represents the probability that such a randomly chosen e-mail will be a ham.

Now do the same Fisher calculation as before, but on the (1 - $f(w)$)s rather than on the $f(w)$s. This will result in near-0 combined probabilities, in rejection of the null hypothesis, when a lot of very spammy words are present. Call this combined probability *S*.

Now calculate:

$$I = \frac{1 + H - S}{2}$$

Equation 4

I is an indicator that is near 1 when the preponderance of the evidence is in favor of the conclusion that the e-mail is spam and near 0 when the evidence points to the conclusion that it's ham. This indicator has a couple of interesting characteristics.

Suppose an e-mail has a number of very spammy words and also a number of very hammy words. Because the Fisher technique is sensitive to values near 0 and less sensitive to values near 1, the result might be that both *S* and *H* are very near 0. For instance, *S* might be on the order of .00001 and *H* might be on the order of .000000001. In fact, those kinds of results are not as infrequent as one might assume in real-world e-mails. One example is when a friend forwards a spam to another friend as part of an e-mail conversation about spam. In such a case, there will be strong evidence in favor of both possible conclusions.

In many approaches, such as those based on the Bayesian chain rule, the fact that there may be more spammy words than hammy words in an example will tend to make the classifier absolutely certain that the e-mail is spam. But in fact, it's not so clear; for instance, the forwarded e-mail example is not spam.

So it a useful characteristic of *I* that it is near .5 in such cases, just as it is near .5 when there is no particular evidence in one direction or the other. When there is significant evidence in favor of both conclusions, *I* takes the cautious approach. In real-world testing, human examination of these mid-valued e-mails tends to support the conclusion that they really should be classified somewhere in the middle rather than being subject to the black-or-white approach of most classifiers.

The Spambayes Project, described in Richie Hindle's article on page 52, takes advantage of this by marking e-mails with *I* near .5 as uncertain. This allows the e-mail recipient to give a bit more attention to e-mails that can't be classified with confidence. This lessens the chance of a good e-mail being ignored due to incorrect classification.

Future Directions

To date, the software using this approach is based on one word per token. Other approaches are possible, such as building a hash table of phrases. It is expected that the math described here can be employed in those contexts as well, and there is reason to believe that phrase-based systems will have performance advantages, although there is controversy about that idea. Future *Linux Journal* articles can be expected to cover any developments in such directions. CRM114 (see Resources) is an example of a phrase-based system that has performed very well, but at the time of this writing it hasn't been directly tested against other approaches on the same corpus. (At the time of this writing, CRM114 is using the Bayesian chain rule to combine *p*(*w*)s.)

Conclusion

The techniques described here have been used in projects such as Spambayes and Bogofilter to improve performance of the spam-filtering task significantly. Future developments, which may include integrating these calculations with a phrase-based approach, can be expected to achieve even better performance.

A Python Implementation of the Inverse Chi-Square Function

Resources

Gary Robinson is CEO of Transpose, LLC (www.transpose.com), a company specializing in internet trust and reputation solutions. He has worked in the field of collaborative filtering since 1985. His personal weblog, which frequently covers spam-related developments, is radio.weblogs.com/0101454, and he can be contacted at grobinson@transpose.com.

Building with Blogs

Doc Searls

David Sifry

Issue #107, March 2003

The category is hot and huge but still new. Here's a look at your choices.

Name a topic with a community of interest around it. Now go to Google and look it up. There's a good chance one or more of the top results will include somebody's weblog (aka blog). Let's take three examples:

- 802.11b: the top listing out of 687,000 results is WiFi news, a weblog by a pair of journalists, Glenn Fleishman and Adam Engst. The IEEE working group's site is in the #2 position. The WiFi Alliance's portal is #3.
- Segway: the top listing out of 81,500 results is the Segway company's site, followed by the Berkeley Segway Portal and then followed by Segway News—a weblog by Paul Nakada.
- Weblog: out of 2,620,000 results, the top listing is Aaron Swartz's Google Weblog, followed by the Guardian Limited's weblog. Following them is the blog of a certain *Linux Journal* editor who also happens to be writing what you're reading right now.

Blogs succeed largely because they are extremely native to the Web as Tim Berners-Lee conceived it in the first place. Here's how weblog software pioneer Dave Winer explains it:

> The first weblog was the first web site, info.cern.ch, the site built by Tim Berners-Lee at CERN. From this page TBL pointed to all the new sites as they came on-line. Luckily, the content of this site has been archived at the World Wide Web Consortium. (Thanks to Karl Dubost for the link.)

Linking to sources and crediting them (as Dave does in that last line) has always been a native ethical and journalistic practice on the Web. While big-time broadcasters, publishers and VC-funded wannabes continue to see the Net as nothing more than a plumbing system for distributing “content” to “consumers”, blogging software developers have quietly added enormous value to the linking and crediting functionality of the Web. Dave and other independent developers have created standards like XML-RPC, SOAP and RSS, plus open APIs that together turn the Web into a writing and publishing medium like nothing we've ever seen before.

Blogging is not about “architecting”, “building”, “designing” or “authoring” anything, because blogs aren't “sites” in the usual sense. Blogs are journals. With blogs you write directly on the Web. Most posts are short, though they don't have to be. All are topical and current, or they disappear from the aggregation sites and services and eventually from the “blogrolls” of listed favorite links on other blogs.

Each blog is like a fireplace, and each post is like a log heaved on top to keep the fire burning. Every post has its own “permalink”, so others can point directly to it. As long as a blog puts out heat and light, others who care about the author's subject are drawn to it. So are Google and other search engines, which sift constantly through the ashes.

At their best, blogs are link magnets as well as sources of links, which is why Google likes them so much. Google equates inbound links with authority and ranks the results accordingly. More links from more highly linked-to pages result in higher page ranks. That's why so many blogs rise to the top of so many subject searches. The whole system—which includes blogs, aggregators, web services and Google itself—feeds, builds and grows on itself. It also attracts and feeds on the RSS streams offered up by discussion and news sites ranging from Slashdot and *Linux Journal* to the *New York Times*. RSS is one among a growing number of free and open technologies created and/or improved by weblog developers.

As weblogs account for more and more of the traffic in knowledge about a given subject, they become powerful instruments for hacking common wisdom. In many categories, they are moving ahead of mainstream journals and portals and building useful community services where over-funded dot-com efforts failed spectacularly. One example from that last category is John Hiler's Cityblogs, which appeared in December 2002. Hiler explains:

> Local sites are caught between a rock and a hard place: either they hire expensive full-time writers to create content they can't afford or they fire their writers and turn to automated content: weather reports, local news and movie listings.

> It's a Gordian knot—hire expensive writers or you have no content—that blogs are uniquely positioned to cut in two. Now that I've set up the site, I can cover these three categories in an hour or two a day. As Glenn “Instapundit” Reynolds put it at a recent conference on weblogs, “blogging is cheap”.

So blogging is becoming an option in all kinds of places, which means there's a good chance that you, as a *Linux Journal* reader, fall into one or both of two groups:

1. Users who want to set up a blog and start writing on the Web.
2. System administrators and others with scripting and programming skills, who are either looking to set up a blogging system or to manage or change a system that's already in place.

To get a handle on both, let's go back to Google.

For better or worse, Google is a commercial company whose services have become de facto web infrastructure. This is especially true for blogs, which make liberal use not only of Google's search engine but also of its APIs, which allow automated queries from programs.

Google's APIs are part of a growing raft of sites and services, mostly hacked together by enterprising independent developers. Technorati (see Sidebar), for example, pays attention to fresh links between blogs (leapfrogging referrer logs) and organizes the information into “watchlists” and other useful listings. If you want Technorati to tell you who's currently linking to your blog, you can ask for this information on the site or pay $5 per year for a watchlist sent out each day by e-mail.

The Technorati Story: How a New Web-Services Product Review Grew Out of a Research Assignment

Technorati is the creation of this article's coauthor, David Sifry, who is a cofounder of Linuxcare and Sputnik. It's a good example of a LAMP program—one based on the de facto platform of Linux, Apache, MySQL and Python, PHP or Perl. Another LAMP creation is Phillip Pearson's Blogging Ecosystem, which keeps two Top 300 lists: one for the most-linked-to blogs and one for the blogs that do the most linking.

Both Technorati and the Blogging Ecosystem are made possible in large part by RSS, the XML dialect whose acronym means really simple syndication. Thanks to RSS, every story you read on the *Linux Journal* web site is syndicated automatically to anyone who wants to read and point to it or aggregate it with other sources.

The Blogger API is another enabling infrastructure hack. It's used not only by Blogger (the most popular weblog system, in terms of sign-ups) but by Radio Userland and Movable Type, the other two leading weblog systems. This API is what made it possible for one of your authors to hack methods for posting to various breeds of blogs through e-mail and Jabber. There are many other hacks just as there are many other blogging systems. SourceForge alone lists dozens of weblog systems in various states of completion. In the LAMP vein, Geeklog and CafeLog are PHP-based and use MySQL, as does Drupal. In fact, PHP-Nuke, PostNuke, Drupal and Slashcode all are flexible enough to serve as weblog systems. So is Rusty Foster's Scoop, which is written in Perl, as is Movable Type. Roller is written in Java for J2EE environments; in fact, there is a whole community of Java bloggers. The list goes on, and it's a long one. So let's break the list down a bit into four family trees.

Built-for-Blogging Systems

Packages designed from the ground up specifically for blogging include Radio Userland, Blogger, Movable Type, Greymatter, Roller and CafeLog. Most include advanced features like the Blogger API, MetaWeblog API, RSS feeds and subscriptions and XML-RPC pings to aggregator sites such as www.weblogs.com. Without RSS feeds and XML-RPC pings, blogs don't get included in blog-based web services like DayPop, Blogdex and Technorati. You hear a lot about the theory of web services, but in the blog world they're easy to put into practice.

Also falling into this category is LiveJournal, an open-source project with a large following that puts a high emphasis on community ties and participation. It's designed as a centralized system open to many clients on different platforms, all of which are also open source. While LiveJournal does RSS feeds, it doesn't make XML-RPC pings to aggregator sites. It also lacks some of the formatting characteristics one associates with blogs, such as blogrolls in the margins, all of which makes it less blog-like than the others in this family.

Discussion Sites

After the wild success of discussion sites like Slashdot and Kuro5hin, a number of software packages emerged with Slashdot-like functionality. Slash is Slashdot's own code base, Scoop is Kuro5hin's, and mod_virgule is the trust system on which Advogato is based. These first-generation packages allow you to set up story posting, site membership, comment moderation, topics or categories (with icons), polls, post archives and so on. These cover the main requirements of blogging tools, although many lack more-advanced functions, such as RSS feeds and XML-RPC pings to aggregators. Given the essentially personal nature of blogging, even on blogs with more than one author, features like karma and moderation are essentially unimportant.

The growing popularity of PHP invited the second generation of discussion-oriented sites. PHP-Nuke launched the generation; but when it went unmaintained, a number of programmers forked the code base and created PostNuke, Geeklog and PostTEP, among others. This generation also allows easy theming and better plugin management, so you can easily write code that might go, say, in a sidebox on the site to perform a specific task. PHP-Nuke is maintained again, by the way. This family also includes hybrids like PHP-Slash, which is Jay Bloodworth's port of Slash code from Perl and mod_perl to PHP.

Content Management Systems

Some blogging tools come out of the traditional web site content management space, including support for multiple authors and permissions and work-flow enforcement. This is the Vignette StoryServer tradition, which includes Zope, which is more than just a CMS, Nucleus and Drupal. Content management sites provide for a more formalized model-view-controller approach, with clear separations between content, markup and work flow. These tools often are ideal for creating dynamic web sites with a lot of authors and editors, but they tend to lack some of the advanced features found in blogging tools, such as RSS feeds and XML-RPC pings. They also lack Blogger and MetaWeblog APIs for posting and editing content from other non-web browser-based applications.

Wikis

Wikis are another way to create dynamic web sites. They allow anyone to edit and mark up a page easily, while still maintaining version control. Some blogging tools are based on Wiki code or on Wiki ideas. These include SnipSnap, TWiki, Tiki, WikiWiki, MoinMoin and ZWiki. They offer some content management but focus mostly on ease of editing and posting and speed of updates, neglecting such blogging functions as dated entries and RSS feeds. Other families worth noting are the Java-based blogging tools, such as Roller and WebForum; and Python- and Ruby-based tools, such as Pyblosxom and tDiary, which is hot in Japan.

We list all these families because your particular needs may not be restricted to blogging. But if blogs are what you want to do, or what you want to support, you need to pay close attention to what works best for blogging purposes.

Simply put, your blogging system doesn't qualify for the label if it can't answer yes to the following questions:

1. Can a user dynamically post to a site?
2. Are posts easy to create, review and edit again after they're posted?
3. Can an administrator limit who posts to the front page?

4. Can a user edit in a browser (at a minimum) or another tool of his or her choice?
5. Does its page format allow blogrolls and other sections outside the daily posting area?
6. Does it produce RSS feeds?
7. Does every post have a permanent URL (or permalink)?
8. Do current posts have unique URLs?
9. Can search engines crawl the archives?
10. Are the archives stable and safe from rot?

The big three—Blogger, Movable Type and Radio Userland—qualify on all those grounds, because they were built from the ground up as pure blogging systems. This is also why most of the blogs listed in the Blogging Ecosystem's Top 300 lists are produced by big three tools.

We've been experimenting at *Linux Journal* with various weblog systems, hosted on a server kindly provided by Penguin Computing. Most of our efforts have been focused on Movable Type, which is the only one of the big three that hosts on Linux and the one that appears to have the most momentum in the Linux development community. The source code is available to end users but not under an open-source license.

Movable Type offers two licenses of its own: a free (as in beer) one for noncommercial use and a $150 US one for commercial use. While this disqualifies Movable Type as an option for writers and publishers, including *Linux Journal*, who prefer to use free software, many blogging organizations that support the Open Source and Free Software movements, such as the Electronic Frontier Foundation and Creative Commons, are using Movable Type.

So what are your choices here? If you want to put up a bare-bones blog and don't mind if it doesn't say yes to all the questions listed above, LiveJournal is a handy and popular choice. You'll be left outside the Blogging Ecosystem (as roughly defined by Phil Pearson's aggregation site by that name), but you'll be using mostly GPLed open-source software and be involved with a lively community.

If you want your blog to thrive in the Ecosystem and don't care too much about what's happening on the back end (just as you might not care what's behind a Hotmail-type web e-mail system), you might consider Blogger or Radio Userland. Both allow you to blog from any server to which you can FTP data and serve HTTP. For example, to do a Blogger blog, go to www.blogger.com, set up an account, make it point to your FTP server with your user name, password and web server HTML directory. In Debian the default is /var/www. On Red Hat, it's /home/httpd/html. This is a simple, easy-to-set-up blog system and a popular choice—even for a hacker who doesn't want to be bothered setting up everything from scratch.

If you want a full-featured Linux-based blogging system for yourself or your organization, and you don't have a problem with its licensing scheme, Movable Type is your best choice. Installing it is almost easy enough for novices. It's also extremely flexible, capable and easy to maintain.

If platform and licensing issues keep you away from the big three, you need to look among the discussion site and content management systems. If you're already using a PHP-based system such as PHP-Nuke, you might consider adapting your current system or going to another one in the same general family, such as Geeklog or Drupal. None are as easy to install as Movable Type, but none are hard to maintain once you master their methods.

Command-line tools for blog editing and posting exist on Linux as well. One example is Philip Myelin's Bzero. Philip also is the author of the Python Community Server and phpStorageSystem, both of which are clones of the Userland Radio Community Server. Running one of these applications on your web server allows you to host Radio-based blogs on your Linux box.

If none of the alternatives suits your fancy, you might consider creating or improving an open-source built-for-blogging system. Greymatter, which is written in Perl and GPLed, had some good momentum going until 2001, when Noah Grey decided he had better things to do. You could pick up where he left off, or you could build a new blog system from scratch.

Countless options are available. URLDIR can generate a variety of feature comparison tables that cover all the systems listed here and then some. If you're at the tire-kicking stage, it's a good place to start.

But don't make a decision before looking at the blogs produced by these different systems. Follow David Ogilvy's classic advice to companies looking for an advertising agency: “Look for work you envy, and find out who does it.”

Blosxom: Think of It as cat(1) with Stylesheets

Doc Searls is senior editor of *Linux Journal*.

David Sifry is cofounder of Linuxcare and Sputnik, where he is also CTO.

Archive Index Issue Table of Contents

Advanced search

Getting Started with Emacs

Charles Curley

Issue #107, March 2003

Emacs is the text editor with everything. Learn the basics—maybe you'll even want to keep your calendar on it.

This article is a whirlwind introduction to Emacs that assumes you have Emacs installed and running (easy enough on most Linux distributions). It also assumes that you have used Emacs' built-in tutorial. Program development in Emacs is not the topic, as that was covered in my June 2002 *LJ* article "Emacs: the Free Software IDE" [available at www.linuxjournal.com/article/5765].

To launch Emacs from an xterm, enter **emacs &**. The ampersand puts Emacs into the background. As X provides the display for Emacs, this setup is fine.

You also can run Emacs in a console by entering **emacs** without the ampersand. To run Emacs in an xterm without opening a new window, launch it with **emacs -nw**. These console and xterm modes are great for situations where you don't have X, such as an SSH connection to a remote server. But if you have SSH set up to forward X, you can run Emacs (and other X applications) remotely.

If you haven't taken the Emacs tutorial, now is the time to do it. Pressing Ctrl-H then T gets you to it. The tutorial is ancient as computers go (1985), so it ignores cursor keys and other modern conveniences. Emacs supports these features but the tutorial doesn't take them into account. It's a good idea to be aware of, if not learn, some of the Emacs keystrokes, though. You can set bash and many other GNU programs to use them. So, for example, Ctrl-B and Ctrl-N can do exactly the same things in Emacs as they do in bash. In fact, Emacs-style key bindings are the default in bash.

The tutorial should teach you basic cursor movement, how to abort an Emacs command, Emacs windowing, the relationship between buffers and files and so on. Probably the most important thing to remember from the tutorial is the movement keys are usually a given key (F for forward) with different modifiers for different ranges. For example, Ctrl-F moves one character forward, and M-F moves a word forward (M- is Emacs notation for Meta, which on most keyboards means the Alt key).

Emacs existed long before web browsers, so it uses the term *frame* for what X calls a window, and *window* for a section within a frame. As this is an Emacs article, this article uses Emacs terminology. To display a new window with a horizontal split, use Ctrl-X 2. For a new window with a vertical split, use Ctrl-X 3. Whereas Ctrl-X 5 1 gives you a whole new frame to play in, Ctrl-X 0 and Ctrl-X 5 0 kill off the current window and frame, respectively.

Another highlight of the tutorial is an introduction to Emacs' incremental search commands. They make life much easier, so learn and remember them.

Ctrl-H is the gateway to Emacs' help system. Pressing Ctrl-H ? gives you a menu for different parts of the help system. The Info system (Ctrl-H I) gives you access to FSF documentation in FSF's Info format. A form of hypertext that predates the World Wide Web, it is arranged in a tree structure. You also can go to an Info node for an Emacs function with Ctrl-H Ctrl-F. This section provides information on the current major and minor modes (more on those in a moment), the warranty and license under which Emacs is provided and other information. Because Emacs' help system is displayed by Emacs, the cursor movement keystrokes you learned in the tutorial apply to the help system.

Getting in the Mode

According to the top Emacs info page, Emacs is the extensible, customizable, self-documenting, real-time display editor. It is extensible because it is written in Emacs Lisp, or elisp, a dialect of Lisp especially customized for Emacs and text processing. You therefore can extend Emacs by writing code in elisp. Furthermore, you can customize it by changing the values of existing elisp variables. Self-documenting might be a slight exaggeration, but elisp does encourage programmers to document. And as we've seen, there is extensive help available.

Users also can customize Emacs by adapting it to specific applications. Do this by switching to what is called a major mode. Only one major mode can be active in a buffer at a time, but you can switch major modes on the fly. For example, when writing CGI scripts it is useful to toggle between Perl mode and HTML Helper mode.

To identify the current modes active in a buffer, see the mode line. In parentheses you will find one or more modes, with the current major mode listed first. Not all minor modes identify themselves in the mode parentheses, but their action is obvious, such as Column Number mode.

Major Modes

Major modes are generally associated with file extensions. A Lisp variable, auto-mode-alist, does this association, and we'll show you how to add to it. Emacs also recognizes associations with shebang entries in the first line of scripts, like this one for Perl:

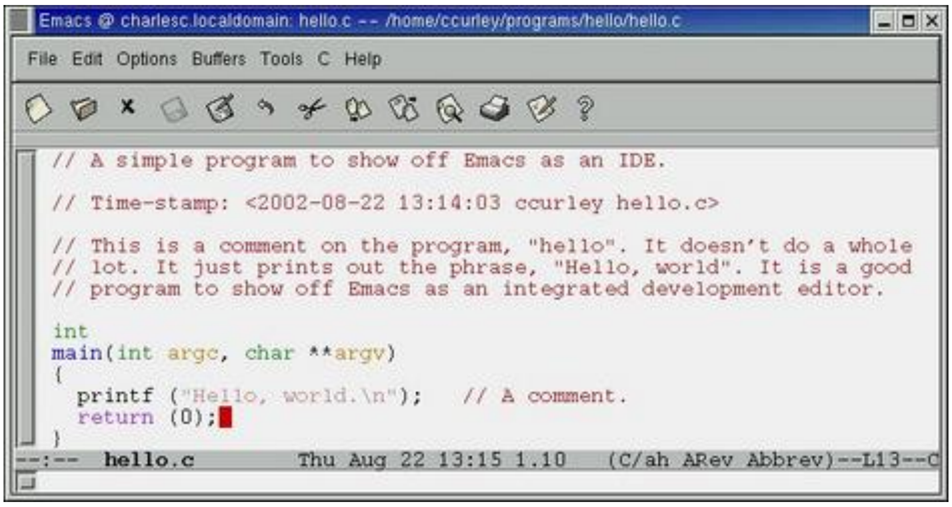
```
#! /usr/bin/perl
```

And you can always force the mode in the first line of a document by surrounding it with **-*-**, like this:

```
# -*- shell-script -*-
```

To switch manually from one major mode to another, use M-X ***mode-name***. For example, M-X **perl-mode** puts Emacs into the major mode for editing Perl.

Major modes provide a number of useful facilities. They usually provide custom indentation and pretty printing options suitable to the subject at hand. There is often some way to insert a comment with a short key sequence. A region of text can be commented out with M-X **comment-region**. One advantage of using Emacs for all your editing is the functions (and their keystrokes and menu entries) available in one major mode tend to be available in another, so if you know how to edit C in Emacs, you probably can edit SQL in Emacs as well (Figure 1).


Emacs @ charlesc.localdomain: hello.c -- /home/ccurley/programs/hello/hello.c
File Edit Options Buffers Tools C Help

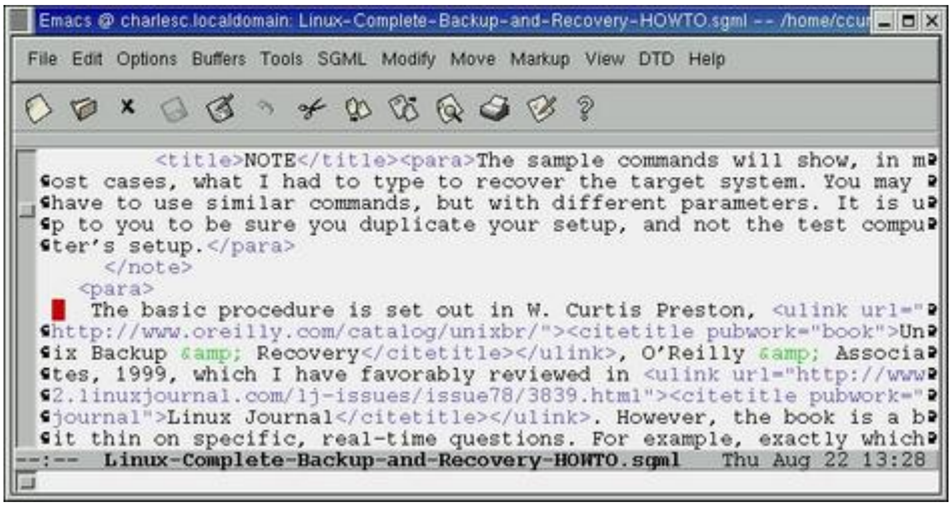
```
// A simple program to show off Emacs as an IDE.

// Time-stamp: <2002-08-22 13:14:03 ccurley hello.c>

// This is a comment on the program, "hello". It doesn't do a whole
// lot. It just prints out the phrase, "Hello, world". It is a good
// program to show off Emacs as an integrated development editor.

int
main(int argc, char **argv)
{
  printf ("Hello, world.\n");   // A comment.
  return (0);
}
--:--  hello.c        Thu Aug 22 13:15 1.10   (C/ah ARev Abbrev)--L13--C
```

Figure 1. C mode in Emacs, showing font locking (color syntax highlighting). The indenting is courtesy of C mode.

Major modes typically provide what Emacs calls font locking, a feature everyone else calls syntax coloring. It automatically associates syntax with colors. For example, comments show up in red, data types in green and strings in a light red. Another advantage of editing with Emacs is that color associations operate across modes, so comments are red regardless of whether you are working in assembler or XML.

Major modes redefine how keystrokes operate, usually the Tab and Delete keys. Also, major modes have mode-specific commands accessed with the prefix Ctrl-C. For example, to validate an SGML document in PSGML mode, use Ctrl-C Ctrl-V.

One of the most powerful major modes around is Lennart Staflin's PSGML mode (see Resources). It facilitates inserting SGML or XML tags and provides automatic pretty printing comparable to C mode. PSGML mode has font locking and other goodies, but it also reads the DTD and uses it to enforce proper tag nesting. For example, in DocBook, it won't let you insert a <sect4> directly into a <sect1>. It also is a front end for a validator (Figure 2).

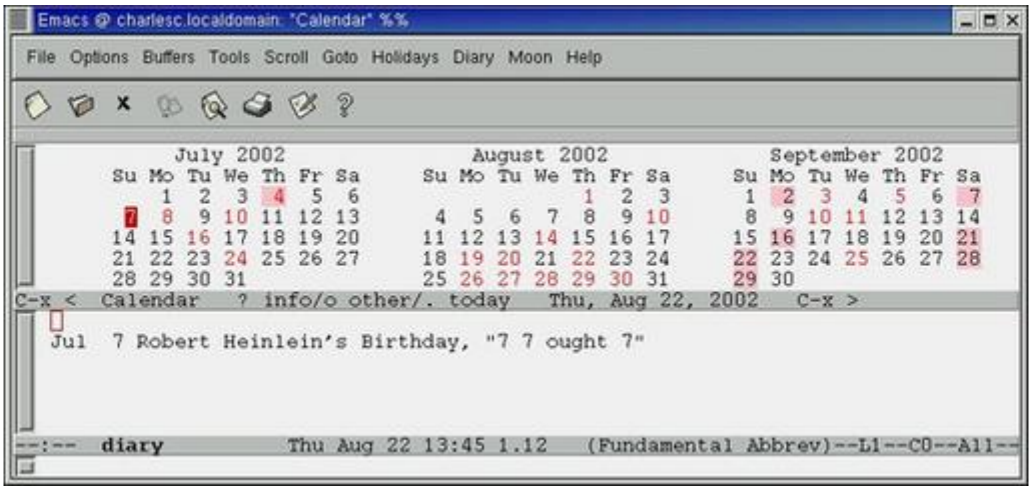

Emacs @ charlesc.localdomain: Linux-Complete-Backup-and-Recovery-HOWTO.sgml -- /home/ccur
File Edit Options Buffers Tools SGML Modify Move Markup View DTD Help

```
        <title>NOTE</title><para>The sample commands will show, in m
ost cases, what I had to type to recover the target system. You may 
have to use similar commands, but with different parameters. It is u
p to you to be sure you duplicate your setup, and not the test compu
ter's setup.</para>
      </note>
    <para>
  The basic procedure is set out in W. Curtis Preston, <ulink url="
http://www.oreilly.com/catalog/unixbr/"><citetitle pubwork="book">Un
ix Backup &amp; Recovery</citetitle></ulink>, O'Reilly &amp; Associa
tes, 1999, which I have favorably reviewed in <ulink url="http://www
2.linuxjournal.com/lj-issues/issue78/3839.html"><citetitle pubwork="
journal">Linux Journal</citetitle></ulink>. However, the book is a b
it thin on specific, real-time questions. For example, exactly which
--:--  Linux-Complete-Backup-and-Recovery-HOWTO.sgml    Thu Aug 22 13:28
```

Figure 2. Editing a Linux Documentation Project document in the DocBook SGML DTD in Emacs. The font locking highlights tags and entities for you.

Other major modes that almost anyone will find useful are Dired mode, Ediff mode, W3 and the calendar and diary. Dired mode is for editing directories. You can navigate from directory to directory, visit files and edit file metadata, such as permissions and ownership. One of the more powerful features of Dired mode is the ability to grep a number of files and have Dired mark the hits. You can then visit each hit in sequence and edit it. This allows you to manipulate files *en masse*, including renaming them or deleting them.

One mode that has proven to be quite useful is the calendar/diary. Not just any calendar tool, Calendar mode allows you to do date manipulations in and conversions between Gregorian and Julian, Copt, Hebrew and Islamic calendars. And, for something completely different, date your next intra-office memo in the Persian or Mayan calendar. Or, send your next bug report to the Free Software Foundation in the French Revolutionary Calendar (Figure 3).

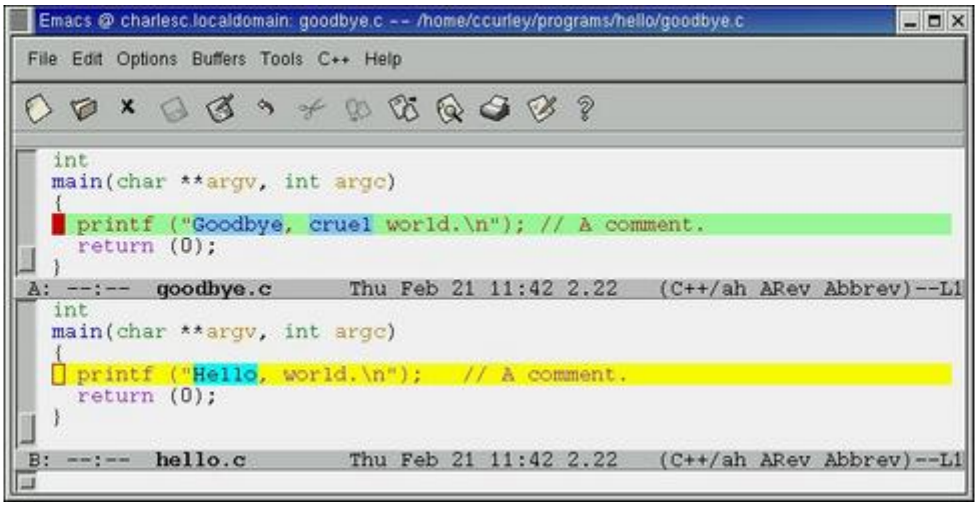

Emacs @ charlesc.localdomain: "Calendar" %%
File Options Buffers Tools Scroll Goto Holidays Diary Moon Help

```
        July 2002              August 2002            September 2002
  Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
      1  2  3  4  5  6                 1  2  3     1  2  3  4  5  6  7
   7  8  9 10 11 12 13     4  5  6  7  8  9 10     8  9 10 11 12 13 14
  14 15 16 17 18 19 20    11 12 13 14 15 16 17    15 16 17 18 19 20 21
  21 22 23 24 25 26 27    18 19 20 21 22 23 24    22 23 24 25 26 27 28
  28 29 30 31             25 26 27 28 29 30 31    29 30
C-x <  Calendar   ? info/o other/. today    Thu, Aug 22, 2002   C-x >

Jul  7 Robert Heinlein's Birthday, "7 7 ought 7"

--:--  diary           Thu Aug 22 13:45 1.12   (Fundamental Abbrev)--L1--C0--All--
```

Figure 3. Emacs' Calendar and Diary

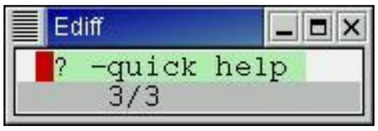
Somewhat more useful than obscure calendars is the diary. With the diary, you can set appointments, anniversary reminders, cyclical events (such as “every third Thursday of the month”) and other types of events. If you specify the event time, Emacs will remind you as the time approaches. Not only is this diary system useful, but it runs in Emacs, so it runs on any computer on which Emacs runs—and that's most of them. The diary file is portable as well.

Ediff mode is useful for selectively applying patches. You also can use it to update files on several computers, such as my .emacs and diary files. Because it is selective, ediff lets you propagate changes in both directions. This can be important if you set appointments on your laptop and your secretary sets them on your desktop (Figures 4 and 5).


Emacs @ charlesc.localdomain: goodbye.c -- /home/ccurley/programs/hello/goodbye.c
File Edit Options Buffers Tools C++ Help

```
int
main(char **argv, int argc)
{
  printf ("Goodbye, cruel world.\n"); // A comment.
  return (0);
}
A: --:--  goodbye.c       Thu Feb 21 11:42 2.22   (C++/ah ARev Abbrev)--L1
int
main(char **argv, int argc)
{
  printf ("Hello, world.\n");    // A comment.
  return (0);
}
B: --:--  hello.c         Thu Feb 21 11:42 2.22   (C++/ah ARev Abbrev)--L1
```

Figure 4. Diffing two files. Emacs shows not only which lines are changed, but what the changes are.



```
? -quick help
   3/3
```

Figure 5. The control window for Ediff mode.

If browsing the World Wide Web is your thing, take a close look at William M. Perry's W3 mode. It is a web browser written in Emacs Lisp.

Minor Modes

Add-ons, called minor modes, supplement major modes. Most minor modes operate regardless of the major mode, so they can operate in different documents. For example, Show Paren mode matches parentheses for you. It is useful for the C programmer even in Text mode, and it's a godsend to the Lisp programmer.

Minor modes can be turned on and off as you wish. For example, when programming, Auto Fill mode (for filling, or line wrapping, paragraphs) is useful in comments, but a nuisance outside of them.

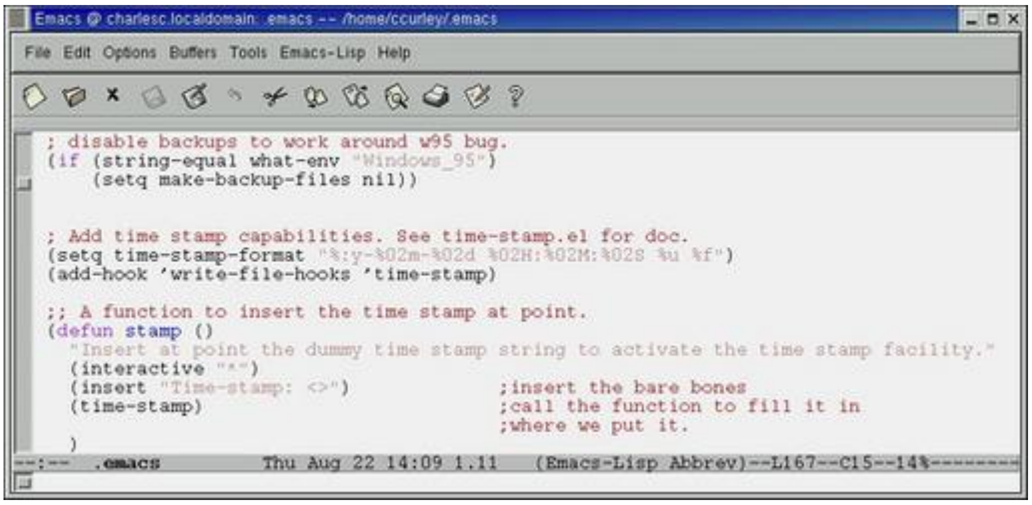
Some minor modes are global; they extend across all buffers when they are active. Others are local to a buffer. To activate a given mode, append **-mode** to its name and execute that command. So to activate Parentheses mode, press M-X then type **show-paren-mode**. To deactivate it, run the command again.

Several useful buffer-local minor modes are Abbrev mode (autocorrection on the fly), Auto Save mode, Font-Lock mode (color highlighting), Flyspell mode (spell checking on the fly) and Overwrite mode. Two useful minor modes that apply to all buffers are Line Number mode and Column Number mode. These print the current position of point in the mode line, usually over to the right.

Another useful mode is Ispell, which lets you spell check your buffer. It has special submodes for checking e-mail messages, programming language comments and strings, and other special uses.

Your .emacs File

Key to customizing Emacs is the initialization file, ~/.emacs. Administrators usually provide a global init file. If you don't like it you can tell Emacs to ignore it in your own init file. And, you can start Emacs with no init file with **emacs -q**, useful for debugging. The init file is nothing but some elisp used to set up Emacs the way you (or your administrator) like it (Figure 6).


Emacs @ charlesc.localdomain: .emacs -- /home/ccurley/.emacs
File Edit Options Buffers Tools Emacs-Lisp Help

```
; disable backups to work around w95 bug.
(if (string-equal what-env "Windows_95")
    (setq make-backup-files nil))


; Add time stamp capabilities. See time-stamp.el for doc.
(setq time-stamp-format "%:y-%02m-%02d %02H:%02M:%02S %u %f")
(add-hook 'write-file-hooks 'time-stamp)

;; A function to insert the time stamp at point.
(defun stamp ()
  "Insert at point the dummy time stamp string to activate the time stamp facility."
  (interactive "*")
  (insert "Time-stamp: <>")                ;insert the bare bones
  (time-stamp)                             ;call the function to fill it in
                                           ;where we put it.
  )
--:--  .emacs        Thu Aug 22 14:09 1.11   (Emacs-Lisp Abbrev)--L167--C15--14%--------
```

Figure 6. Editing the author's .emacs, an example of Emacs Lisp, in Emacs.

You also can set variables in the init file. I customize HTML Helper mode by setting some mode variables:

```
(setq html-helper-do-write-file-hooks t)
(setq html-helper-build-new-buffer t)
(setq html-helper-address-string "<a href=
\"mailto:ccurley@trib.com\">Charles
Curley</a>")
```

Short useful functions, or macros, also go into .emacs. For example, the following function inserts today's date at point:

```
(defun insert-date ()
  "Insert the current date according to the variable
\"insert-date-format\"."
  (interactive "*")
  (insert (format-time-string insert-date-format
                              (current-time))))
```

Keystrokes and key sequences also can be bound to functions. This allows you to use the key sequence to activate the function. For example, having written the function insert-date, I can bind it to the F3 function key with this line:

```
(global-set-key [f3] 'insert-date)
```

You also can use this capability to remap your keyboard. If you don't like some of the long key sequences in Emacs, you can rebind them.

The other way to customize Emacs is with the Customize menu, accessed with M-X **customize** or from the Options pull-down menu. This extensive menu system allows users to change variables and store the changes in your init file.

Emacs as a Server

A number of programs, such as crontab and mutt, invoke an external program as their editor. To let them run Emacs, set Emacs up to run as a server by putting this line into your .emacs file:

```
(server-start)
```

Next, set the environment variable EDITOR or VISUAL to emacsclient. In Bash, add this to your /etc/bashrc or your ~/x.bash_profile:

```
export VISUAL=emacsclient
```

Now, when you execute **crontab -e** or edit a message in mutt, you edit in your existing Emacs session instead of waiting for a new Emacs to start up. To finish editing and make emacslient exit, end your session in that buffer with Ctrl-C # instead of Ctrl-X K.

For emacsclient to work, Emacs must be running when the external program invokes it. This is consistent with the preferred way of using Emacs, which is to start Emacs when you log in and leave it running until you log out. One result of using emacsclient is you only have one instance of Emacs running at any one time. While memory is cheap today, it wasn't always so. And even today, if you want to run Linux on your laptop or elderly computers, conserving memory is always a good idea.

You might want to have Emacs edit your mail in Mail mode. If you use mutt, add this to your .emacs file:

```
;; Automatically go into mail-mode if
filename starts with /tmp/mutt
(setq auto-mode-alist (append (list (cons
"^\/tmp\/mutt" 'mail-mode))
                                  auto-mode-alist))
```

Of course, to comply with the RFCs on netiquette, you will want Auto Fill mode active when you edit mail. Most major modes have a hook they execute on entering the mode and another they execute on leaving. Here is how to get Mail mode to invoke Auto Fill mode:

```
(defun my-mail-mode-hook ()
  (auto-fill-mode 1)
  )
(add-hook 'mail-mode-hook 'my-mail-mode-hook)
```

When you are done writing your e-mail, if you want to annoy the NSA, use Spook. To protest the Communications Decency Act (a decent thing to do) and annoy a lot of American politicians, see Bruce.

Finally, before we take our leave of this wild and woolly editor, let me bring the etc directory (in the Emacs directory tree) to your attention. It contains a number of useful documents, such as an Emacs English language reference card, in source (refcard.tex) and postscript (refcard.ps) form. Translations of the reference card into other languages are available. There is also some background material on Emacs and the GNU Project and a copy of the GPL.

Something you rarely find in proprietary software (at least, not deliberately) is present in Emacs: humor. Check out the bug report from the year 2199, the word list for Spook mode, some explanations of what Emacs stands for and more. And if you really want to exercise your font server, visit the file “HELLO”.

Resources

email: ccurley@trib.com

Charles Curley (w3.trib.com/~ccurley) writes about and teaches Linux. Naturally, he wrote this article using Emacs on Linux.

Archive Index Issue Table of Contents

Advanced search

Linux Signals for the Application Programmer

Dr. B. Thangaraju

Issue #107, March 2003

Signals are a fundamental method for interprocess communication ad are used in everything from network servers to media players. Here's how you can use them in your applications.

A good understanding of signals is important for an application programmer working in the Linux environment. Knowledge of the signaling mechanism and familiarity with signal-related functions help one write programs more efficiently.

An application program executes sequentially if every instruction runs properly. In case of an error or any anomaly during the execution of a program, the kernel can use signals to notify the process. Signals also have been used to communicate and synchronize processes and to simplify interprocess communications (IPCs). Although we now have advanced synchronization tools and many IPC mechanisms, signals play a vital role in Linux for handling exceptions and interrupts. Signals have been used for approximately 30 years without any major modifications.

The first 31 signals are standard signals, some of which date back to 1970s UNIX from Bell Labs. The POSIX (Portable Operating Systems and Interface for UNIX) standard introduced a new class of signals designated as real-time signals, with numbers ranging from 32 to 63.

A signal is generated when an event occurs, and then the kernel passes the event to a receiving process. Sometimes a process can send a signal to other processes. Besides process-to-process signaling, there are many situations when the kernel originates a signal, such as when file size exceeds limits, when an I/O device is ready, when encountering an illegal instruction or when the user sends a terminal interrupt like Ctrl-C or Ctrl-Z.

Every signal has a name starting with SIG and is defined as a positive unique integer number. In a shell prompt, the **kill -l** command will display all signals with signal number and corresponding signal name. Signal numbers are defined in the /usr/include/bits/signum.h file, and the source file is /usr/src/linux/kernel/signal.c.

A process will receive a signal when it is running in user mode. If the receiving process is running in kernel mode, the execution of the signal will start only after the process returns to user mode.

Signals sent to a non-running process must be saved by the kernel until the process resumes execution. Sleeping processes can be interruptible or uninterruptible. If a process receives a signal when it is in an interruptible sleep state, for example, waiting for terminal I/O, the kernel will awaken the process to handle the signal. If a process receives a signal when it is in uninterruptible sleep, such as waiting for disk I/O, the kernel defers the signal until the event completes.

When a process receives a signal, one of three things could happen. First, the process could ignore the signal. Second, it could catch the signal and execute a special function called a signal handler. Third, it could execute the default action for that signal; for example, the default action for signal 15, SIGTERM, is to terminate the process. Some signals cannot be ignored, and others do not have default actions, so they are ignored by default. See the signal(7) man page for a reference list of signal names, numbers, default actions and whether they can be caught.

When a process executes a signal handler, if some other signal arrives the new signal is blocked until the handler returns. This article explains the fundamentals of the signaling mechanism and elaborates on signal-related functions with syntax and working procedures.

Signals inside the Kernel

Where is the information about a signal stored in the process? The kernel has a fixed-size array of proc structures called the process table. The u or user area of the proc structure maintains control information about a process. The major fields in the u area include signal handlers and related information. The signal handler is an array with each element for each type of signal being defined in the system, indicating the action of the process on the receipt of the signal. The proc structure maintains signal-handling information, such as masks of signals that are ignored, blocked, posted and handled.

Once a signal is generated, the kernel sets a bit in the signal field of the process table entry. If the signal is being ignored, the kernel returns without taking any action. Because the signal field is one bit per signal, multiple occurrences of the same signal are not maintained.

When the signal is delivered, the receiving process should act depending on the signal. The action may be terminating the process, terminating the process after creating a core dump, ignoring the signal, executing the user-defined signal handler (if the signal is caught by the process) or resuming the process if it is temporarily suspended.

The core dump is a file called core, which has an image of the terminated process. It contains the process' variables and stack details at the time of failure. From a core file, the programmer can investigate the reason for termination using a debugger. The word core appears here for a historical reason: main memory used to be made from doughnut-shaped magnets called inductor cores.

Catching a signal means instructing the kernel that if a given signal has occurred, the program's own signal handler should be executed, instead of the default. Two exceptions are SIGKILL and SIGSTOP, which cannot be caught or ignored.

sigset_t is a basic data structure used to store the signals. The structure sent to a process is a sigset_t array of bits, one for each signal type:

```
  typedef struct {
                   unsigned long sig[2];
                }  sigset_t;
```

Because each unsigned long number consists of 32 bits, the maximum number of signals that may be declared in Linux is 64 (according to POSIX compliance). No signal has the number 0, so the other 31 bits in the first element of sigset_t are the standard first 31 signals, and the bits in the second element are the real-time signal numbers 32-64. The size of sigset_t is 128 bytes.

Handling Signals

There are many system calls and signal-supported library functions, which provide an easy and efficient way of handling the signals in a process. We start with the standard old signal system call, then we discuss some useful functions like sigaction, sigaddset, sigemptyset, sigdelset, sigismember and kill.

The Signal System Call

The signal system call is used to catch, ignore or set the default action of a specified signal. It takes two arguments: a signal number and a pointer to a user-defined signal handler. Two reserved predefined signal handlers are available in Linux: SIG_IGN and SIG_DFL. SIG_IGN will ignore a specified signal, and SIG_DFL will set the signal handler to the default action for that signal (see **man 2 signal**).

On success, the system call returns the previous value of the signal handler for the specified signal. If the signal call fails, it returns SIG_ERR. Listing 1 explains how to catch, ignore and set the default action of SIGINT. Try pressing Ctrl-C, which sends SIGINT, during each part.

Listing 1. Catching and Ignoring a Signal

sigaction

The sigaction system call can be used instead of signal because it has lot of control over a given signal. The syntax of sigaction is:

```
int sigaction ( int signum,
                const struct sigaction *act,
                struct sigaction *oldact);
```

The first argument, signum, is a specified signal; the second argument, sigaction, is used to set the new action of the signal signum; and the third argument is used to store the previous action, usually NULL.

The sigaction structure is defined as:

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
}
```

The members of the sigaction structure are described as follows.

sa_hander: a pointer to a user-defined signal handler or predefined signal handler (SIG_IGN or SIG_DFL).

sa_mask: specifies a mask of signals when the signal is handled. To avoid the blocking of signals, the SA_NODEFER or SA_NOMASK flags can be used.

sa_flags: specifies the action of signal. Sets of flags are available for controlling the signal in a different manner. More than one flag can be used by ORing:

- SA_NOCLDSTOP: if we specify the SIGCHLD signal, when the child has stopped its execution it does not receive notification.

- SA_ONESHOT or SA_RESETHAND: restores the default action of the signal after the user-defined signal handler is executed. To avoid setting the default action, SA_RESTART can be used.
- SA_NOMASK or SA_NODEFER prevents masking the signal. SA_SIGINFO is used to receive signal-related information.

sa_sigaction: if the SA_SIGINFO flag is used in sa_flags, instead of specifying the signal handler in sa_handler, sa_sigaction should be used.

sa_sigaction is a pointer to a function that takes three arguments, not one as sa_handler does, for example:

```
void my_handler (int signo, siginfo_t *info,
                     void *context)
```

Here, signo is the signal number, and info is a pointer to the structure of type siginfo_t, which specifies the signal-related information; and context is a pointer to an object of type ucontext_t, which refers to the receiving process context that was interrupted with the delivered signal.

Listing 2 is similar to Listing 1 but uses the sigaction system call instead of the signal system call. Listing 3 explains signal-related information using the SIG_INFO flag.

Listing 2. Same as Listing 1, but with Sigaction

Listing 3. Using SA_SIGINFO and sa_sigaction to Extract Information from a Signal

Sending Signals

Until now, we've been pressing Ctrl-C to send SIGINT from the shell. To do it from a program, use the kill system call, which accepts two arguments, process ID and signal number:

```
int kill ( pid_t process_id, int signal_number );
```

If the pid is positive, the signal is sent to a particular process. If the pid is negative, the signal is sent to the process whose group ID matches the absolute value of pid.

As you might expect, the kill command, which exists as a standalone program (/bin/kill) and is also built into bash (try **help kill**) uses the kill system call to send a signal.

Not all processes can send signals to each other. In order for one process to send a signal to another, either the sender must be running as root, or the sender's real or effective user ID must be the same as the real or saved ID of the receiver. This means your shell, running as you, can signal a setuid program that you started, but that is now running as root, for example:

```
cp /bin/sleep ~/rootsleep
sudo chmod u+s ~/rootsleep
./rootsleep 40
killall rootsleep
rm ~/rootsleep
```

A normal user can't send signals to system processes such as swapper and init.

You also can use kill to find out if a process exists. Specify a signal number of 0, and if the process exists, the kill returns zero; if it doesn't, kill returns -1.

Listing 4. Programs to Send and Receive SIGINT

Listings 4 and 4a explain how to use the kill system call. First, execute the 4a program in one window and get its process ID. Now, run the Listing 4 program in another window and give the 4a example's pid as the input.

This article should help you understand the fundamental concept of a signal and some of its importance. Try the sample programs, and see the man pages for the system calls and the references in Resources for more information.

Resources

email: balasubramanian.thangaraju@wipro.com

Dr B. Thangaraju received a PhD in Physics and worked as a research associate for five years at the Indian Institute of Science, India. He is presently working as a manager at Talent Transformation, Wipro Technologies, India. He has published many research papers in renowned international journals. His current areas of research, study and knowledge dissemination are the Linux kernel, device drivers and real-time Linux.

Using the Input Subsystem, Part II

Brad Hards

Issue #107, March 2003

No matter how many buttons an input device has or how many kinds of events it can generate, you can now work with it from user space.

In last month's article, we saw how the Linux input subsystem worked inside the kernel, ending with a quick mention of the event handlers. Each handler essentially provides a different user-space API, converting input events into the particular format that makes up that API.

One of the key aspects of the input subsystem integration into Linux is the availability of the event interface. This basically exposes the raw events to userspace through a collection of character device nodes—one character device node per logical input device. The event interface is a really powerful technique, because it allows the manipulation of the events in userspace without information loss. For example, legacy mouse interfaces support only two relative axes and up to five buttons. These are normally mapped to the two real axes and three real buttons, with the fourth and fifth buttons logically being mapped to the scroll wheel up and scroll wheel down events.

However, this mapping becomes a problem when trying to use a mouse with a scroll wheel and more than three buttons, because any additional buttons can be mapped only to an existing button. The legacy APIs also impede use of advanced input devices, such as space balls and other devices' with many axes. By contrast, the event API provides full access to the devices capabilities, and it even includes a per-device description of those capabilities and other device characteristics.

This month's article focuses on the various ioctl capabilities of the event interface, in addition to the normal read and write calls.

Finding the Version of the Event Interface

The event interface supports determining the version of the event device code, using the EVIOCGVERSION ioctl function. The argument is an int (32 bits) and is meant to be interpreted as a major version (two high bytes), a minor version (third byte) and a patch level (low byte). The same value is returned from each event device on a machine.

An example of the EVIOCGVERSION is shown in Listing 1. The first argument to the ioctl function is an open file descriptor for the event device node (for example, /dev/input/event0). Notice that you have to pass a pointer to the integer variable, not the variable itself, as the third argument to the ioctl call.

Listing 1. Sample EVIOCGVERSION Function

Finding Information about the Device's Identity

The event interface supports retrieving information associated with the underlying device using the EVIOCGID ioctl. The argument is a pointer to an input_id structure; the input_id structure is defined as shown in Listing 2. The __u16 data type is a Linux-specific, unsigned 16-bit integer. You can safely cast it to a standard uint16_t in your code.

Listing 2. iput_id Structure Definitions

The bus type is the only field that contains accurate data. You should probably consider it to be an opaque, enumerated type, and compare it with the various BUS_x type definitions provided in <linux/input.h>. The vendor, product and version fields are bus type-specific information relating to the identity of the device. Modern devices (typically using PCI or USB) do have information that can be used, but legacy devices (such as serial mice, PS/2 keyboards and game ports on ISA sound cards) do not. These numbers therefore are not meaningful for some values of bus type.

An example of the EVIOCGID ioctl is shown in Listing 3. This example calls the ioctl and then prints out the results. The case logic shows all current bus types. Here is an example of running that code: **vendor 045e product 001d version 0111 is on a Universal Serial Bus**.

Listing 3. Sample EVIOCGID ioctl

In addition to the type of bus and the vendor, product and version information, some devices can provide strings that make up meaningful names. This can be obtained from the event interface using the EVIOCGNAME ioctl. This ioctl provides a string and returns the length of the string (or a negative error value).

If the string is too long to fit into the argument, it will be truncated. An example is provided in Listing 4. If it seems strange that the argument is not &name, remember the name of an array is the same as a pointer to the first element. Therefore, &name would be a pointer to a pointer to the first element, which is not what we want. If you really want to use a dereference, use &(name[0]).

Listing 4. Example Trunctated String

Here is an example of running that event code:

```
The device on /dev/input/event0 says its name
    is Logitech USB-PS/2 Optical Mouse
```

Not all devices contain meaningful names, however, so kernel input drivers try to provide something meaningful. For example, USB devices without manufacturer or product strings concatenate the vendor and product ID information.

Although device identity and name information is often useful, it may not be sufficient information to tell which device you have. For example, if you have two joysticks that are the same, you may need to identify them based on which port they use. This is usually known as topology information, and you can get this from the event interface using the EVIOCGPHYS ioctl. Like EVIOCGNAME, this provides a string and returns the length of the string (or a negative error number). An example is shown in Listing 5; running that example will produce something like:

```
The device on /dev/input/event0 says its path
    is usb-00:01.2-2.1/input0
```

Listing 5. Using EVIOCGPHYS for Topology Information

To understand what this string is showing, you need to break it down into parts. The usb part means this is a physical topology from the USB system. The 00:01.2 is the PCI bus information for the USB host controller (in this case, bus 0, slot 1, function 2). The 2.1 shows the path from the root hub to the device. In this case, the upstream hub is plugged in to the second port on the root hub, and that device is plugged in to the first port on the upstream hub. input0 means this is the first event device on the device. Most devices have only one, but multimedia keyboards may present the normal keyboard on one interface and the multimedia function keys on a second interface. This topology example is shown in Figure 1.

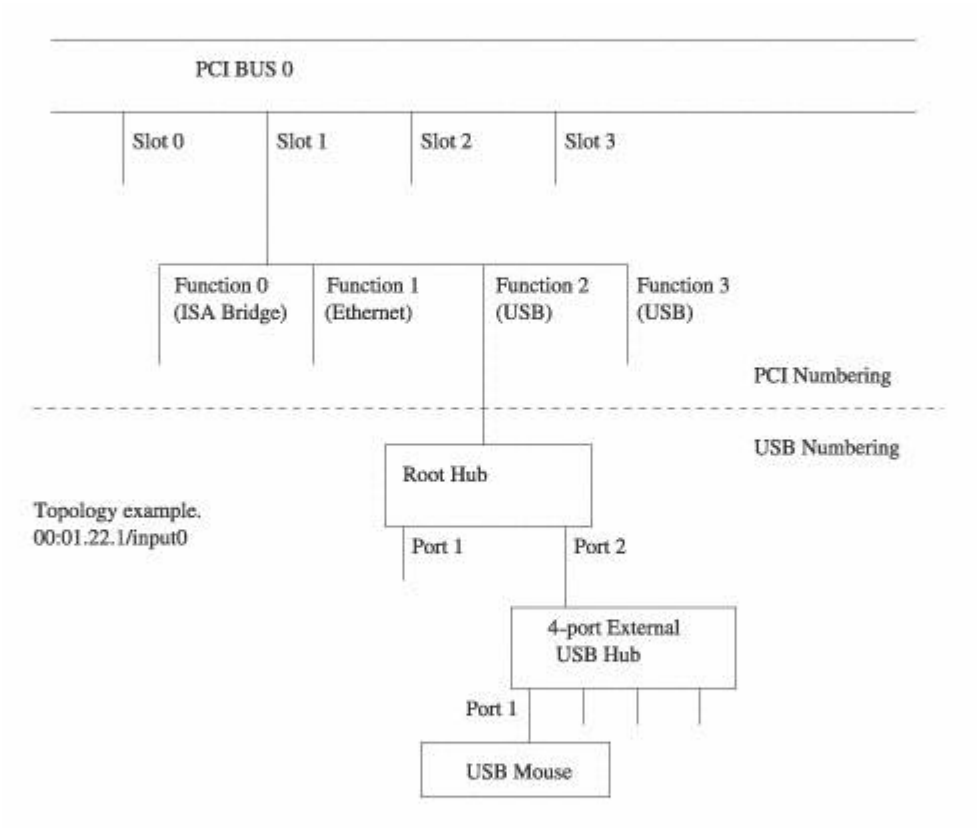


Figure 1. Keyboard Topology

This setup doesn't help if you swap the cables on two identical devices. The only thing that can help in this case is if the device has some form of unique identifier, such as a serial number. You can get this information using the EVIOCGUNIQ ioctl. An example is shown in Listing 6. Most devices don't have such an identifier, and you will get an empty string from this ioctl.

Listing 6. Finding a Unique Identifier

Determining the Device Capabilities and Features

For some applications, it might be enough to know the device identity, because this would allow you to handle any special cases depending on what device is being used. However, it doesn't scale well; consider a case where you want to enable scroll wheel handling only if the device has a scroll wheel. You really don't want to have to list the vendor and product information for every mouse with a scroll wheel in your code.

To avoid this problem, the event interface allows you to determine which features and capabilities are available for a particular device. The types of features supported by the event interface are:

- EV_KEY: absolute binary results, such as keys and buttons.
- EV_REL: relative results, such as the axes on a mouse.

- EV_ABS: absolute integer results, such as the axes on a joystick or for a tablet.
- EV_MSC: miscellaneous uses that didn't fit anywhere else.
- EV_LED: LEDs and similar indications.
- EV_SND: sound output, such as buzzers.
- EV_REP: enables autorepeat of keys in the input core.
- EV_FF: sends force-feedback effects to a device.
- EV_FF_STATUS: device reporting of force-feedback effects back to the host.
- EV_PWR: power management events.

These are only the *types* of features; a wide range of individual features can be found within each type. For example, the EV_REL feature type distinguishes between X, Y and Z axes and horizontal and vertical wheels. Similarly, the EV_KEY feature type includes literally hundreds of different keys and buttons.

The capabilities or features of each device can be determined through the event interface, using the EVIOCGBIT ioctl. This function allows you to determine the types of features supported by any particular device, for example, whether it has keys, buttons or neither. It further allows you to determine the specific features that are supported, for example, which keys or buttons are present.

The EVIOCGBIT ioctl takes four arguments. If we consider it as ioctl(fd, EVIOCGBIT(ev_type, max_bytes), bitfield), then the fd argument is an open file descriptor; ev_type is the type of features to return (with 0 as a special case, indicating the list of all feature types supported should be returned, rather than the list of particular features for that type); max_bytes shows the upper limit on how many bytes should be returned; and bitfield is a pointer to the memory area where the result should be copied. The return value is the number of bytes actually copied on success or a negative error code on failure.

Let's look at a couple of examples of the EVIOCGBIT ioctl call. The first example, Listing 7, shows how to determine the types of features present. It determines how much memory is required for the bit array using evtype_bitmask, based on the EV_MAX definition in <linux/input.h>. The ioctl is then submitted, and the event layer fills in the bit array. We then test each bit in the array and show where the bit was set, which indicates the device does have at least one of this type of feature. All devices support the EV_SYN feature type in 2.5; the input core sets this bit.

Listing 7. Determining Features with EVIOCGBIT

When run, with a keyboard as the target, the example in Listing 7 produces:

```
Supported event types:
  Event type 0x00  (Synchronization Events)
  Event type 0x01  (Keys or Buttons)
  Event type 0x11  (LEDs)
  Event type 0x14  (Repeat)
```

With a mouse as the target, the example produces:

```
Supported event types:
  Event type 0x00  (Synchronization Events)
  Event type 0x01  (Keys or Buttons)
  Event type 0x02  (Relative Axes)
```

Retrieving Input from (and to) the Device

Having determined what capabilities a particular device has, you know what types of events it will produce and what types of events you can send.

Retrieving events from a device requires a standard character device “read” function. Each time you read from the event device (e.g., /dev/input/event0), you will get a whole number of events, each consisting of a struct input_event.

Listing 8. Checking for Busy Spots

The example shown in Listing 8 does a busy loop on the open file descriptor, trying to read any events. It filters out any events that don't correspond to keys and then prints out the various fields in the input_event structure. Running this while typing on my keyboard produced:

```
Event: time 1033621164.003838, type 1, code 37, value 1
Event: time 1033621164.027829, type 1, code 38, value 0
Event: time 1033621164.139813, type 1, code 38, value 1
Event: time 1033621164.147807, type 1, code 37, value 0
Event: time 1033621164.259790, type 1, code 38, value 0
Event: time 1033621164.283772, type 1, code 36, value 1
Event: time 1033621164.419761, type 1, code 36, value 0
Event: time 1033621164.691710, type 1, code 14, value 1
Event: time 1033621164.795691, type 1, code 14, value 0
```

You get one event per key press and another per key release.

This read interface has all the normal characteristics of a character device, meaning you don't need to use a busy loop. You can simply wait until your program needs some input from the device and then perform the read call. In addition, if you are interested in the input from a number of devices, you can use the poll and select functions to wait on a number of open devices at the same time.

Sending information to the device is a process similar to receiving it, except you use the standard write function instead of read. It is important to remember that the data used in the write call has to be a struct input_event.

A simple example of writing data is shown in Listing 9. This example turns the Caps Lock LED on, waits 200 milliseconds and then turns the Caps Lock LED off. It then turns the Num Lock LED on, waits 200 milliseconds, and then turns the Num Lock LED off. The cycle then repeats (in an infinite busy loop), so you see alternate flashing of the two keyboard LEDs.

Listing 9. Sample Data Write Function

By now it should be fairly clear that you receive events only when something changes—a key is pressed or released, the mouse is moved and so on. For some applications, you need to be able to determine what the global state of the device is. For example, a program that manages keyboards may need to determine which LEDs are currently lit and which keys are currently depressed on the keyboard, even though some of the keys may have been depressed before the application started.

The EVIOCGKEY ioctl is used to determine the global key and button state for a device. An example is shown in Listing 10. This ioctl is similar to the EVIOCGBIT(...,EV_KEY,...) function in some ways; instead of setting a bit in the bit array for each key or button that the device can send, EVIOCGKEY sets a bit in the bit array for each key or button that is depressed.

Listing 10. Determining a Device's Global Key and Button State

The EVIOCGLED and EVIOCGSND functions are analogous to EVIOCGKEY, except that they return which LEDs are currently lit and sounds that are currently turned on, respectively. An example of how you would use EVIOCGLED is shown in Listing 11. Again, each bit is interpreted in the same way as the bits in the bit array are filled in by EVIOCGBIT.

Listing 11. Using EVIOCGLED

You can determine the repeat rate settings for a keyboard using the EVIOCGREP ioctl. An example is shown in Listing 12, with two elements to the array. The first element specifies the delay before the keyboard starts repeating, and the second element specifies the delay between subsequent repeats. So if you hold down a key, you'll get one character straight away, a second character rep[0] milliseconds later, a third character rep[1] milliseconds after the second character and another character every rep[1] milliseconds thereafter, until you release the key.

Listing 12. Checking the Repeat Rate Settings

You also can set the key repeat rate using EVIOCSREP. This uses the same two-element array that you'd use to get the settings, as shown in Listing 13; it sets the initial delay to 2.5 seconds and the repeat rate to 1 per second.

Listing 13. Setting the Repeat Rates

Some input drivers support variable mappings between the keys held down (which are interpreted by the keyboard scan and reported as *scancodes*) and the events sent to the input layer. You can determine what key is associated with each scancode using the EVIOCGKEYCODE ioctl. An example is shown in Listing 14, which loops over the first 100 scancodes. The value of the scancode (input to the function) is the first element in the integer array, and the resulting input event key number (keycode) is the second element in the array. You can also modify the mapping, using the EVIOCSKEYCODE ioctl. An example is shown in Listing 15; this ioctl maps my M key to always produce a letter N. Be aware that keycode ioctl functions may not work on every keyboard—that USB keyboard is an example of a driver that does not support variable mappings.

Listing 14. Looping over Scancodes

Listing 15. Mapping Keys

The EVIOCGABS function also provides state information. Instead of filling in a bit array that represents the global state of boolean values, however, it provides a struct input_absinfo (see Listing 16) for one absolute axis. If you want the global state for a device, you have to call the function for each axis present on the device. An example is shown in Listing 17. The elements in the array are signed 32-bit quantities, and you can safely treat them as equivalent to int32_t. The first element shows the current value of the axis, the second and third elements show the current limits of the axis, the fourth element shows the size of the “flat” section (if any) of the response and the last element shows the size of the error that may be present.

Listing 16. input_absinfo for an Absolute Axis

Listing 17. Checking Globabl State by Axis

Force Feedback

Three additional ioctl functions may be used for controlling force-feedback devices: EVIOCSFF, EVIOCRMFF and EVIOCGEFFECT. These functions currently send a force-feedback effect, remove a force-feedback effect and determine how many simultaneous effects can be used, respectively. Because the force-feedback support is still emerging and substantial work remains to be done, it is a little premature to fully document the API. The web sites listed in the

Resources section of this article may have updated information by the time you read this.

Resources

email: bhards@bigpond.net.au

Brad Hards is the technical director for Sigma Bravo, a small professional services company in Canberra, Australia. In addition to Linux, his technical focus includes aircraft system integration and certification, GPS and electronic warfare. Comments on this article may be sent to bradh@frogmouth.net.

Archive Index Issue Table of Contents

Advanced search

Unicode

Reuven M. Lerner

Issue #107, March 2003

Unicode is necessary for international web development but poses a few pitfalls.

Growing up in the northeastern United States, I never had to use a language other than English. I read in English, spoke in English, wrote in English and conducted business in English. This was also true of the engineers who created ASCII back in 1968, who made sure the 128 ASCII characters would suffice for English-language documents. So long as you stuck with the standard set of ASCII characters, you were guaranteed the ability to move files from one computer to another without having to worry about them getting garbled.

ASCII was fine in its day, but people who spoke French, Spanish and other Western European languages quickly discovered that it was insufficient for their needs. After all, people who write in these languages on a computer want to use the correct accent marks. So over the course of time, the 7-bit ASCII code became the 8-bit extended ASCII code, including a number of special letters and symbols necessary for displaying Western European text.

But because extended ASCII was never declared a standard, a number of different, incompatible extensions to the base ASCII code became widespread. Windows had its own extensions, as did the Macintosh and NeXTSTEP operating systems. So although you could write a document in French using Windows, you would need to translate it when moving it to the Macintosh. Otherwise, the bytes would be interpreted on the receiving machine incorrectly, turning your otherwise superb French screenplay into something more akin to French toast.

International standards finally prevailed, at least somewhat, with a standard known formally as ISO-8859-1 and informally as Latin-1. Computer manufacturers could then exchange Western European documents without having to worry about things becoming garbled. Of course, this meant we were using all eight bits of each character's byte, doubling the number of available characters from 128 to 256.

However, this didn't solve all of the problems. For example, Hebrew speakers have their own standard, ISO-8859-8, which is identical to Latin-1 for characters 0-127 and quite different from 128-256. A document written in Hebrew but displayed on a computer using Latin-1 will look like a letter substitution puzzle using letters from the wrong alphabet.

Practically speaking, this means you cannot write a document that contains English, Hebrew and French using the ISO-8859 series of standards. And indeed, this makes sense given the fact that we have only 256 characters to play with in a single 8-bit byte. But it raises some serious questions and issues for those of us who work with more than two languages.

Things get especially hairy if you want to display a page in English, French, Hebrew and Chinese. After all, there are tens of thousands of ideographs in Chinese, not to mention Japanese and other languages.

Enter Unicode, the ASCII table for the next century. Like ASCII, Unicode assigns a number to each letter, number and symbol. Unlike ASCII, Unicode contains enough space for every written symbol ever created by humans. This means that a Unicode document can contain any number of characters from any number of languages, without having to worry about clashes between them. Unicode also handles a number of issues that ASCII never dreamed about, including combining characters (for accents and other diacritical marks) and directional issues (for languages that do not read from left to right).

Unicode has been around for about a decade, but it is only now becoming popular and supported for web applications. This month, we take a look at Unicode as it affects web developers. What should you consider? What do you need to worry about? And, how can you get around the problems associated with Unicode?

Introduction to Unicode

Unicode, like ASCII, assigns a unique number to each letter, number, symbol and control character. As indicated above, though, Unicode extends through each of the symbols and character sets ever created. So using Unicode, you can create a document that uses English, Russian, Japanese and Arabic, in which each character is clearly distinct from the others.

How do we turn these unique numbers—known as code points in the Unicode universe—into bits and bytes? The encoding for ASCII is very straightforward; with only 127 characters (or 256, if you include the various extensions), each

ASCII character will fit into a single byte. And indeed, C programmers know that the char data type is an 8-bit integer.

The most obvious solution is to assign a fixed multibyte encoding for our Unicode characters. And indeed, UCS-2 is such an encoding, using two bytes to describe all of the basic 65,536 Unicode characters. (There are some extended characters that require additional bytes, but we won't go into that.) UCS-2 assigns a single 2-byte code to each of these characters. Documents are thus equally long regardless of the language in which they are written, and programs can easily calculate the number of bytes they need by doubling the number of characters. Microsoft's modern operating systems use UCS-2, as you might have noticed if you exchange any documents with users of those systems.

But there is a basic problem with UCS-2, namely its incompatibility with ASCII. If you have 100,000 documents written in ASCII, you will have to translate them into UCS-2 in order to read them accurately. Given that most modern programs work with ASCII, this lack of backward compatibility is quite a problem.

Enter UTF-8, which is a variable-length Unicode encoding. Just as Roman and Arabic numerals represent the same numbers differently, UTF-8 and UCS-2 are simply different encodings for the same underlying Unicode character set. But whereas every UCS-2 character requires two bytes, a UTF-8 character might require anywhere from one to four bytes. One-byte UTF-8 characters are the same as in ASCII, which means that a legal ASCII document is also a legal UTF-8 document. However, Latin-1 and other 8-bit character sets are incompatible with UTF-8; existing Latin-1 documents will not only need to be transformed but could potentially double in size.

UTF-8 is the preferred encoding on UNIX and Linux systems, as well as in most of the standards and open-source software that I tend to use. Perl, Python, Tcl and Java all encode strings in UTF-8. PostgreSQL has supported UTF-8 for years, and Unicode support has apparently been added to MySQL 4.1, which will be released in alpha in the coming months.

Adding Unicode support to an existing system is a Herculean task for which the various developers should be given great praise. Not only do developers need to add support for multibyte characters, but databases and languages also need to support regular expressions and sorting operators, neither of which is easy to do.

Unicode and HTTP

Now that we have gotten the basics out of the way, let's consider how Unicode documents are transferred across the Web. The basic problem is this: when your browser receives a document, how does it know if it should interpret the bytes as Latin-1, Big-5 Chinese or UTF-8?

The answer lies in the Content-type HTTP header. Every time an HTTP server sends a document to a browser, it identifies the type of content it is sending using a MIME-style designation, such as text/html, image/png or application/msword. If you receive a JPEG image (image/jpeg), there is only one way to represent the image. But if you receive an HTML document (text/html), the Content-type header must indicate the character set and/or encoding that is being used. We do this by adding a charset= designation to the end of the header, separating the type from the charset. For example:

```
Content-type: text/html; charset=utf-8
```

Purists rightly say that UTF-8 is an encoding and not a character set. Unfortunately, it's too late to do anything about this. This is similar to the fact that the word “referrer” is misspelled in the HTTP specification as “referer”; everyone knows that it's wrong but is afraid to break existing software.

If no Content-type is specified, it is assumed to be Latin-1. Moreover, if no Content-type is specified, individual documents can set (or override) the value within a metatag. Metatags cannot override an explicit setting of the character set, however.

As you begin to work with different encodings, you will undoubtedly discover an HTTP server that has not been configured correctly and that is announcing the wrong character set in the Content-type header. An easy way to check this is to use Perl's LWP (library for web programming), which includes a number of useful command-line programs for web developers, for example:

```
$ HEAD http://yad2yad.huji.ac.il/
```

Typing the above on my Linux box returns the HTTP response headers from the named site:

```
200 OK
Cache-Control: max-age=0
Connection: close
Date: Tue, 10 Dec 2002 08:38:37 GMT
Server: AOLserver/3.3.1+ad13
Content-Type: text/html; charset=utf-8
```

As you can see, the Content-type header is declaring the document to be in UTF-8.

Mozilla and other modern browsers allow the user to override the explicitly stated encoding. Although this should not normally be necessary for end users, I often find this functionality to be useful when developing a site.

Unicode and HTML

Although it's nice to know we can transfer UTF-8 documents via HTTP, we first need some UTF-8 documents to send. Given that ASCII documents are all UTF-8 documents as well, it's easy to create valid UTF-8 documents, so long as they contain only ASCII characters. But what happens if you want to create HTML pages that contain Hebrew or Greek? Then things start to get interesting and difficult.

There are basically two ways to include Unicode characters in an HTML document. The first is to type the characters themselves using an editor that can work with UTF-8. For example, GNU Emacs allows me to enter text using a variety of keyboard options and then save my document in the encoding of my choice, including UTF-8. If I try to save a Chinese document in the Latin-1 encoding, Emacs will refuse to comply, warning me that the document contains characters that do not exist in Latin-1. Unfortunately, for people like me who want to use Hebrew, Emacs doesn't yet handle right-to-left input.

A better option, and one which is increasingly impressive all of the time, is Yudit, an open-source UTF-8-compliant editor that handles many different languages and directions. It can take a while to learn to use Yudit, but it does work. Yudit, like Emacs, allows you to enter any character you want, even if your operating system or keyboard does not directly support all of the desired languages.

Both Emacs and Yudit are good options if you are working on Linux, if you are willing to tinker a bit, and if you don't mind writing your HTML by hand. But nearly all of the graphic designers I know work on other platforms, and getting them to work with HTML editors that use UTF-8 has been rather difficult.

Luckily, Mozilla comes with not only a web browser but a full-fledged HTML editor as well. As you might expect, Mozilla's composer module is a bit rough around the edges but handles most tasks just fine.

Another option is to use HTML entities. The best-known entities are &lt;, &gt; and &amp; which make it possible to insert the <, > and & symbols into an HTML document without having to worry that they will be interpreted as tags.

Modern browsers not only understand entities such as &copy; (the copyright symbol) but also include the full list of Unicode characters. Thus, you can refer to Unicode characters by inserting &#*XXXX*; in your document, entering the character's decimal code instead of the *XXXX*. For example, the following HTML document displays my name in Hebrew, using Unicode entities:

```
<html>
    <head><title>Reuven's name</title></head>
    <body><p>&#1512;&#1488;&#1493;&#1489;&#1503;</p>
    </body>
</html>
```

Creating the above document does not require a Unicode-compliant editor, and it will render fine in any modern browser, regardless of the Content-type that was declared in the HTTP response headers. However, editing a file that uses entities in this way is tedious and difficult at best. Unfortunately, the save-as-HTML feature in the international editions of Microsoft Word uses this extensively, which makes it easy for Word users to create Unicode-compliant documents but difficult for people to edit them later.

Pitfalls

As I indicated earlier, Unicode is a complex standard, and it has taken some time for different languages and technologies to support it. For example, Perl 5.6.x used Unicode internally, but input and output operations couldn't easily use it, which made such support basically useless. Perl 5.8 by contrast has excellent Unicode support, allowing developers to write regular expressions that depend on Unicode properties.

There are still some problems, however. A major problem that developers have to deal with is the issue of input encoding vs. storage encoding, such as when your terminal might use Latin-1 but the back end might use UTF-8. This sort of arrangement means you can continue to use your old (non-Unicode) terminal program and fonts but connect to and use your Unicode-compliant back-end program.

Various implementations also have some holes, which might not be obvious when you first start to work on a project. For example, I recently worked on a J2EE project that used PostgreSQL on its back end and stored all of the characters in Unicode. Everything was fine until we decided to compare the user's input string with text in the database in a case-insensitive fashion. Unfortunately, the PostgreSQL function we used doesn't handle case insensitivity correctly for Unicode strings. We found a workaround in the end, but it was both embarrassing and frustrating to encounter this.

Collating, or sorting, is also a difficult issue—one that has bitten me on a number of occasions. Unicode defines a character set, but it does not indicate in which order the characters in that set should be sorted. Until recently, for example, “ch” was sorted as its own separate letter in Spanish-speaking countries; this was not true for speakers of English, German and French. The sort order thus depends not only on the character set, but on the locale in which the character set is being applied. You may need to experiment with the

LANG and LC_ALL environment variables (among others) to get things to work the way you expect.

Conclusion

Unicode is clearly the way of the future; most operating systems now support it to a certain degree, and it is becoming an entrenched standard in the computer world. Unfortunately, Unicode requires unlearning the old practice of equating characters and bytes and handling a great deal of new complexities and problems.

If you only need to use a single language on your web site, then consider yourself lucky. But if you want to use even a single non-ASCII character, you will soon find yourself swimming in the world of Unicode. It's worth learning about this technology sooner rather than later, given that it is slowly but surely making its way into nearly every open-source system and standard.

Resources

Reuven M. Lerner (reuven@lerner.co.il) is a consultant specializing in web/ database technologies. His first book, Core Perl, was published by Prentice Hall in January 2002. His next book, about open-source web/development environments, will be published by Apress in late 2003. Reuven lives with his wife and daughters in Modi'in, Israel.

Archive Index Issue Table of Contents

Advanced search

Chatting Up the Chef

Marcel Gagné

Issue #107, March 2003

Introducing two chat servers, geektalkd and Shadowlands Forum, and a fine Jabber instant-messaging client.

Everything looks wonderful, François. Candles on every table, wineglasses, workstations, DSL feeds—we are ready. Tonight's menu is a toast, if you will, to this issue's theme of On-line Fora. Every item is designed to bring people closer together, even if in a virtual fashion, *non*? Perhaps we should serve a somewhat romantic wine; the 1997 Alsatian Gewurztraminer sounds right for this evening. You don't agree, *mon ami*? Well, perhaps *romantic* is a bit much, but consider this, human interaction, even on-line, is all about communication, and communication implies a certain closeness—even if you don't like the person, *non*?

Quoi? Well, it is all very simple, François. The whole idea of people communicating with other people is at the heart of the Internet's existence—whether it is a web page trying to sell a product or a colleague sending an e-mail. When we cook with Linux, we can take advantage of those communication possibilities in many different ways.

Ah, but our guests have arrived! *Bonsoir, mes amis*. Welcome to *Chez Marcel*, home of fine Linux fare and most excellent wine, *non*? Your tables are ready. Please, be seated. My faithful waiter will serve the wine.

On-line fora exist in a number of incarnations, *mes amis*, from web portals, to intranets, to chat rooms and even instant messaging. The tools at our disposal provide us with the means of joining in existing discussions or hosting them. Your Linux system is the perfect platform for hosting a server, and doing so need not be complicated. Witness this simple little package called geektalkd, written by Michael Plump who credits David Elkins with a lot of the bug fixes. geektalkd is a simple chat server written entirely in Perl. Its simple, open-source nature, makes it easy to extend, modify or simply appreciate. It supports channels, private discussions and more.

To install and run geektalkd, you will need to have the Net::Ident Perl module installed. The easiest way to do this is by using Perl's CPAN mode. You can do the entire installation on the command line by typing:

```
   perl -MCPAN -e "install Net::Ident"
```

If this is the first time you have installed something using the Perl CPAN shell, you'll have to go through a little question-and answer-session. For the most part, you can accept everything the system suggests with a couple of exceptions, notably those having to do with selecting a mirror. Depending on whether this was your first time or you whether you have been around the CPAN block before, this might take a minute to several minutes.

When this is done, extract and build geektalkd:

```
   tar -xzvf geektalkd-1.23.tar.gz
   cd geektalkd-1.23
   perl install
```

That's it. Running the server is as simple as typing the command itself, which will then fork into the background. You might want to consider adding the -l option and specifying a log file, or adding the -p to specify another port number. By default, geektalkd runs on port 41000:

```
   /usr/local/sbin/geektalkd -l /var/log/gtd_log -p 41001
```

With the server running, all a client has to do is telnet to the port.

```
   telnet your_site.dom 41000
```

The server then responds with a connect message. Type **/help** for a list of commands.

Another candidate for an extremely easy chat server is the Shadowlands Forum. Written by Jeremy Monin, Dan Chin and Matt Lochansky, this software is a highly functional chat room with some excellent (and occasionally just plain fun) features. For instance, users can set options such as away messages, idle timeouts, create macros, enable mail checks (for local users) and much more. Administrators can set defaults for the whole room, decide who participates and who doesn't, set topics and so on. To try out Shadowlands Forum, pick up your copy at www.shadowlands.org/forum, then extract the software into a local directory:

```
   tar -xzvf slforum-1.9.6.1.tar.gz
   cd slforum-1.9.6
```

Next, you'll want to edit slfprefs.h before continuing. This contains site-specific configurations, such as slforum's home directory and where its files live. If your system uses shadow passwords, you will need to uncomment the #define USE_SHADOW line (the default uses PAM). You also can edit system messages here, or you could, of course, accept the defaults. When you are happy, compile the program like this:

```
make -f Makefile-linux
```

There is no **make install**. When the compile is done, simply copy the slforum executable to an appropriate directory. If you are using PAM, you also need to copy slforum.pam to the PAM directory (**cp slforum.pam /etc/pam.d/slforum**). It is now time to run your new chat room:

```
./slforum -ag marcel +aa /path_to/chat.log
```

The -ag option identifies the group ID of the admin group to the server. Meanwhile, the +aa option turns on the guest auto-accept feature. As for the log file at the end of the command, this can be anything you like. Use -h to see all the options. To use the forum, use any telnet client and connect on port 7777:

```
telnet your_site.dom 7777
```

From there, you can either log in as guest or as a member of the admin group. Guests will be asked for their handle (or name) and admins will have to enter a password. Once you are logged in, you can get a list of available commands by typing **!help**. Shadowlands Forum is loaded with features. It even comes with a magic 8 ball to help answer all of life's complex questions.

```
marcel> !8 ball Should I serve Chenin Blanc with
    on-line fora?
FORUM: The 8 ball has advised marcel:
FORUM: To Answer Would Dishonor the 8 ball
```

Well, *mes amis*, perhaps this 8 ball is not very useful after all.

The final item on tonight's menu is an instant-messaging client. What, you might ask, is the difference between the chat servers already featured and instant messaging? The two seem similar indeed. The difference is that every other chat tool we have worked with requires that you connect to a central location. The only way to find out whether another person is on, is to connect.

Instant-messaging protocols like Jabber (www.jabber.org) make it possible to chat in real time with a friend or relative (colloquially known in IM parlance as a *buddy*) in the same way as our previous examples. The difference is that IM clients communicate with each other, letting you know when your buddy comes on-line. Jabber, by the way, is an open, nonproprietary, XML-based instant-messaging protocol you should consider using as an alternative to the other services. In fact, Jabber's open nature means that you can run your own server, an ideal situation for private intranets.

It is entirely likely that you already have a fine Jabber IM client loaded on your system. It is called Gaim. It was originally written by Mark Spencer and is now maintained by Rob Flynn (with the help of the usual band of dedicated coders and patch writers). We will be using Gaim to set up a Jabber account, but Gaim also can talk to AOL's instant-messaging client, and Yahoo's client, and Microsoft's MSN client and others as well. This multiprotocol support is achieved through the use of plugins, making Gaim one of the most flexible instant-messaging clients around.

If you don't already have Gaim on your system, pick up the latest source at gaim.sourceforge.net. Building the package is a matter of executing the old extract and build five-step:

```
tar -xzvf gaim-0.59.6.tar.gz
cd gaim-0.59.6
./configure
make
su -c "make install"
```

Start Gaim, either from your menu or by typing **gaim &** at the command line, and the main Gaim window will appear (Figure 1).

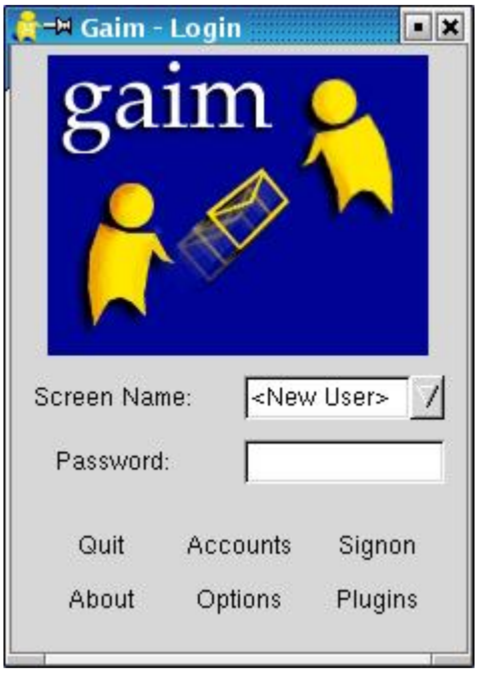


Figure 1. Starting Gaim for the First Time

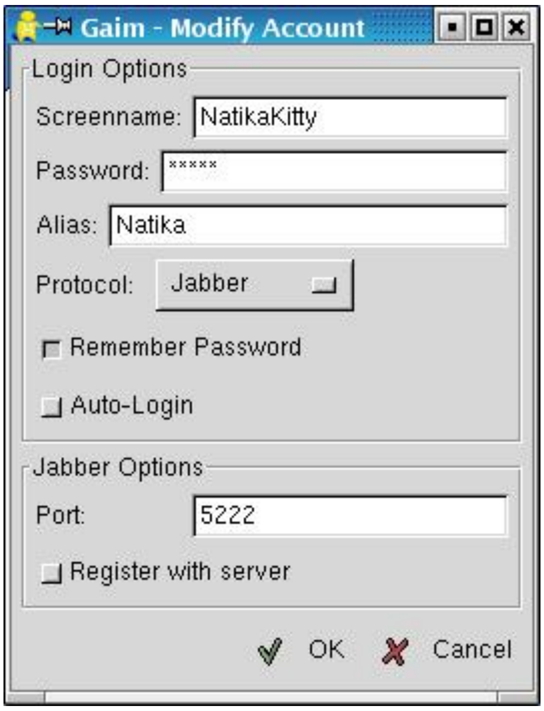
The first time you start up Gaim, you'll get a simple window with text fields for Screen Name and Password. The Screen Name field will have the words <New User> entered letting you know you don't have any accounts set up. Obviously, *mes amis*, before we start using instant messaging through Gaim, we will need at least one account, *non*?

Look below the text fields and you'll see six buttons labeled Quit, Accounts, Signon, About, Options and Plugins. By default, Gaim is ready to accept an ICQ account (if you already have one). In order to use Jabber as one of your protocols, you may need to load its plugin, which isn't activated by default. This is very easy to do. Click the Plugins button.

The Gaim Plugins window appears. Click on Load, then select a protocol from the pop-up list. In the case of Jabber, you would click on libjabber.so and click OK. You would follow the same steps to load the Yahoo protocol (libyahoo.so) or the MSN chat protocol (libmsn.so). Once you have loaded the Jabber plugin (or any other that took your fancy), click Close.

When the plugin window disappears, click the Accounts button. The Account Editor window will appear with nothing in it.

Click Add and the Modify Account window will appear. Halfway down that window, you should see a drop-down list labeled Protocol. By default, it says AIM/ICQ. Click on it, select Jabber, and watch as the window changes to reflect the requirements of setting up a Jabber account.


Gaim - Modify Account

Login Options

Screenname: NatikaKitty

Password: *****

Alias: Natika

Protocol: Jabber

Remember Password

Auto-Login

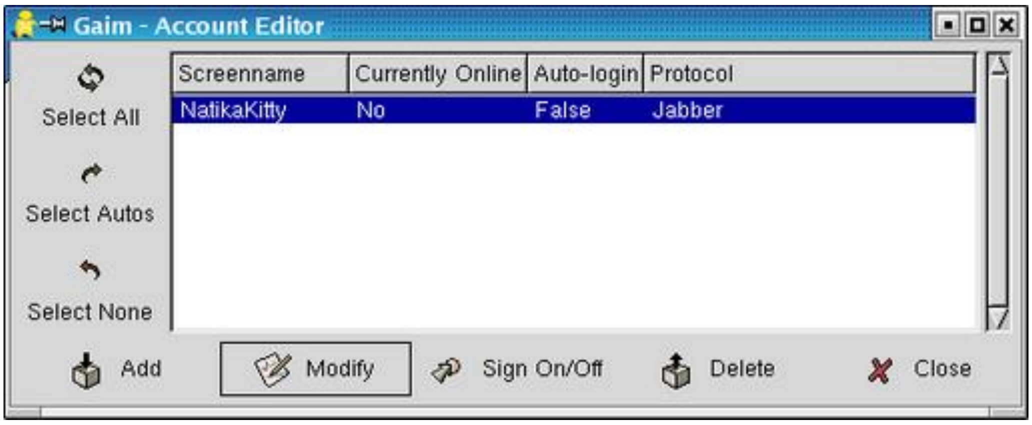
Jabber Options

Port: 5222

Register with server

Figure 2. Creating or Modifying a Jabber Account

Enter your Screenname, Password and Alias, then click on the Register with server radio button. If you would like your Gaim client to log in to Jabber automatically every time you start up the client, click on the Auto-Login radio button as well. When you are happy with your changes, click OK. Your Account Editor window will show your new account (Figure 3).



Screenname	Currently Online	Auto-login	Protocol
NatikaKitty	No	False	Jabber

Figure 3. Adding or editing your accounts is easy.

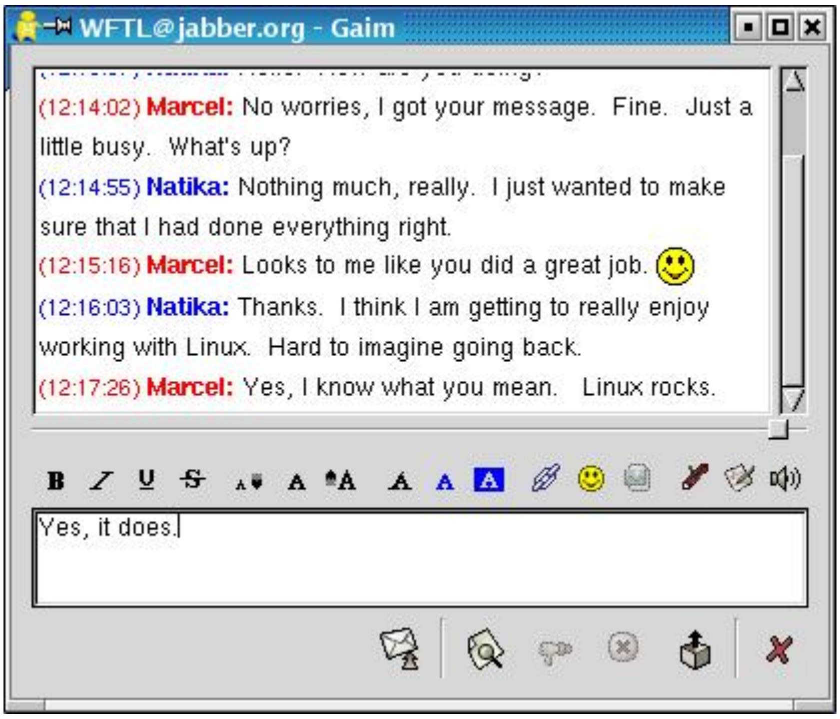
You can sign on here either by clicking the Sign On/Off button or by clicking Close and signing on from the main Gaim window. With your first time in, you'll get a welcome message from the Jabber.org server. You can close this window or visit the site (as indicated in the message) for additional information.

Now that you have your very own Jabber instant-messaging account, you need people to talk to. There are on-line chats you can join by clicking File on the menubar and selecting Join A Chat. You also can use the keyboard shortcut Ctrl-C instead. Add friends to your Buddy List by selecting Add A Buddy from the menu.

After you have added buddies to the list, they will get messages letting them know that you want to add them. When they see the pop up, they will click Accept, at which point you can begin conversing.

This accepting of buddies has to happen at both sides of the connection. They accept you, after which you accept them. Think of it as saying “I do”, but to a more casual, dare I say, virtual, relationship. François, all this closeness is calling for Champagne, would you not agree? Perhaps we should share a bottle with our guests before they leave, *non*? *Vite*, François! To the wine cellar. The 1988 Brut Champagne Rare, please.

While François is off to get the Champagne, we'll finish up our examination of Gaim. Once all this accepting has taken place, your buddy will appear on your buddy list. An icon beside the name on your buddy list will indicate whether your buddy is on-line. If so, double-click on the name and start chatting. It is that easy.



(12:14:02) Marcel: No worries, I got your message. Fine. Just a little busy. What's up?
(12:14:55) Natika: Nothing much, really. I just wanted to make sure that I had done everything right.
(12:15:16) Marcel: Looks to me like you did a great job.
(12:16:03) Natika: Thanks. I think I am getting to really enjoy working with Linux. Hard to imagine going back.
(12:17:26) Marcel: Yes, I know what you mean. Linux rocks.

Yes, it does.

Figure 4. Talk all you want, instantly.

Gaim is highly configurable as well. Click Tools and Preferences for a list of options. One I particularly like (under Conversations) is the option to have all conversations tabbed in one chat window. If you find yourself talking to a number of people, this is one you will definitely want to set.

Ah, François, welcome back, *mon ami*. Please pour a glass for our friends. I fear that closing time is upon us again. Until next month, *mes amis*, let us all drink to one another's health. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné lives in Mississauga, Ontario. He is the author of Linux System Administration: A User's Guide (ISBN 0-201-71934-7), published by Addison-Wesley (and is currently at work on his next book). He can be reached via e-mail at mggagne@salmar.com.

rsync, Part I

Mick Bauer

Issue #107, March 2003

rsync makes efficient use of the network by only transferring the parts of files that are different from one host to the next. Here's how to use it securely.

Andrew Tridgell's rsync is a useful file-transfer tool, one that has no encryption support of its own but is easily “wrapped” (tunneled) by encryption tools such as SSH and Stunnel. What differentiates rsync (which, like scp, is based on rcp) is that it has the ability to perform differential downloads and uploads of files.

For example, if you wish to update your local copy of a 10MB file, and the newer version on the remote server differs in only three places totaling 150KB, rsync will automatically download only the differing 150KB (give or take a few KB) rather than the entire file. This functionality is provided by the rsync algorithm, invented by Andrew Tridgell and Paul Mackerras, which rapidly creates and compares rolling checksums of both files and thus determines which parts of the new file to download and add/replace on the old one.

Because this is a much more efficient use of the network, rsync is especially useful over slow network connections. It does not, however, have any performance advantage over rcp in copying files that are completely new to one side or the other of the transaction. By definition, differential copying requires that there be two files to compare.

In summary, rsync is by far the most intelligent file-transfer utility in common use, one that is both amenable to encrypted sessions and worth taking the trouble to figure out how. Using rsync securely is the focus of the remainder of this article.

rsync supports a long list of options, most of them relevant to specific aspects of maintaining software archives, mirrors and backups. Only those options directly relevant to security will be covered in depth here, but the rsync(8) man page will tell you anything you need to know about these other features.

Getting, Compiling and Installing rsync

Because Andrew Tridgell, rsync's original lead developer, is also one of the prime figures in the Samba Project, rsync's home page is part of the Samba web site, rsync.samba.org. That, of course, is the definitive source of all things rsync. The resources page, rsync.samba.org/resources.html, has links to some excellent off-site rsync documentation.

The latest rsync source code is available at rsync.samba.org/ftp/rsync, with binary packages for Debian, LinuxPPC and Red Hat Linux at rsync.samba.org/ftp/rsync/binaries. rsync is already considered a standard Linux tool and therefore is included in all popular Linux distributions; you probably needn't look further than the Linux installation CD-ROMs to find an rsync package for your system.

However, there are security bugs in the zlib implementation included in rsync prior to rsync v.2.5.4. These bugs are applicable regardless of the version of your system's shared zlib libraries. There is also an annoying bug in v2.5.4 itself, which causes rsync sometimes to copy whole files needlessly. I therefore recommend you run no version earlier than rsync v.2.5.5.

Happily, compiling rsync from source is fast and easy. Simply unzip and untar the archive, change your working directory to the top-level directory of the source code, type **./configure**, and if this script finishes without errors, type **make && make install**.

Running rsync over SSH

Once rsync is installed, you can use it several ways. The first and most basic is to use rcp as the transport, which requires any host to which you connect to have the shell service enabled (i.e., in.rshd) in inetd.conf. Don't do this! The reason the Secure Shell was invented was because of a complete lack of support for strong authentication in the “r” services (rcp, rsh and rlogin), which led to their being used as entry points by many successful intruders over the years.

Therefore, I won't describe how to use rsync with rcp as its transport. However, you may wish to use this method between hosts on a trusted network; if so, ample information is available in both rsync's and in.rshd's respective man pages.

rsync Works Two Ways

A much better way to use rsync than the rcp method is by specifying the Secure Shell as the transport. This requires that the remote host be running sshd and that the rsync command is present (and in the default paths) of both hosts. If you haven't set up sshd yet, do that first.

Suppose you have two hosts, near and far, and you wish to copy the local file thegoods.tgz to far's /home/near.backup directory, which you think may already contain an older version of thegoods.tgz. Assuming your user name, yodeldiva, exists on both systems, the transaction might look like this:

```
yodeldiva@near:~> rsync -vv -e ssh \
./thegoods.tgz far:~
```

Let's dissect the command line. rsync has only one binary executable, rsync, which is used both as the client command and, optionally, as a dæmon. In this example, it's present on both near and far, but it runs on a dæmon on neither; sshd is acting as the listening dæmon on far.

The first rsync option in the above example is -vv, which is the nearly universal UNIX shorthand for “very verbose”. It's optional, but instructive. The second option is -e, with which you can specify an alternative to rsync's default remote-copy program rcp. Because rcp is the default, and because rcp and SSH are the only supported options, -e is used to specify SSH in practice.

Perhaps surprisingly, **-e scp** will not work, because prior to copying any data, rsync needs to pass a remote rsync command via SSH to generate and return rolling checksums on the remote file. In other words, rsync needs the full functionality of the **ssh** command to do its thing, so specify this rather than scp if you use the -e option.

After the options come rsync's actionable arguments, the local and remote files. The syntax for these is very similar to rcp's and scp's. If you immediately precede either filename with a colon, rsync will interpret the string preceding the colon as a remote host's name. If the user name you wish to use on the remote system is different from your local user name, you can specify it by immediately preceding the hostname with an @ sign and preceding that with your remote user name. In other words, the full rsync syntax for filenames is the following:

```
[[username@]hostname:]/path/to/filename
```

There must be at least two filenames. The right-most must be the destination file or path, and the others must be source files. Only one of these two may be remote, but both may be local (colon-less), which lets you perform local differential file copying—this is useful if, for example, you need to back up files from one local disk or partition to another.

The source file specified is ./thegoods.tgz, an ordinary local file path, and the destination is far:~, which translates to “my home directory on the server far”. If your user name on far is different from your local user name, say yodelerwannabe rather than yodeldiva, use the destination yodelerwannabe@far:~.

Setting Up an rsync Server

Using rsync with SSH is the easiest way to use rsync securely with authenticated users, in a way that both requires and protects the use of real users' accounts. But as I mentioned earlier in the “SFTP and SSH” section [of the book, see Sidebar], SSH doesn't lend itself easily to anonymous access. What if you want to set up a public file server that supports rsync-optimized file transfers?

This is quite easy to do. Create a simple /etc/rsyncd.conf file, and run rsync with the --daemon flag (i.e., **rsync --daemon**). The devil, however, is in the details. You should configure /etc/rsyncd.conf very carefully if your server will be connected to the Internet or any other untrusted network. Let's discuss how.

rsyncd.conf has a simple syntax; global options are listed at the beginning without indentation. Modules, which are groups of options specific to a particular filesystem path, are indicated by a square-bracketed module name followed by indented options.

Option lines each consist of the name of the option, an equal sign and one or more values. If the option is boolean, allowable values are yes or no (don't be misled by the rsyncd.conf(5) man page, which in some cases refers to true and false). If the option accepts multiple values, these should be comma-space delimited, for example, **option1, option2**.

Listing 1 is part of a sample rsyncd.conf file that illustrates some options particularly useful for tightening security. Although I created it for this purpose, it's a real configuration file and syntactically complete. Let's dissect it.

A Sample rsyncd.conf File

As advertised, the global options are listed at the top. The first option set also happens to be the only global-only option: syslog facility, motd file, log file, pid file and socket options may be used only as global settings, not in module settings. Of these, only syslog facility has direct security ramifications. Like the ProFTPD directive SyslogFacility, rsync's syslog facility can be used to specify which syslog facility rsync should log to if you don't want it to use **daemon**, its default.

For detailed descriptions of the other global-only options, see the rsyncd.conf(5) man page; I won't cover them here, as they don't directly affect system security, and their default settings are fine for most situations, anyhow.

All other allowable rsyncd.conf options can be used as global options, in modules or both. If an option appears in both the global section and in a module, the module setting overrides the global setting for transactions involving that module. In general, global options replace default values, and module-specific options override both default and global options.

The second group of options falls into the category of module-specific options.

use chroot = *`yes`*: if use chroot is set to yes, rsync will chroot itself to the module's path prior to any file transfer, preventing or at least hindering certain types of abuses and attacks. This has the trade-off of requiring that rsync --daemon be started by root, but by also setting the uid and gid options, you can minimize the amount of the time rsync uses its root privileges. The default setting is yes.

uid = *`nobody`*: the uid option lets you specify with which user's privileges rsync should operate during file transfers, and it therefore affects which permissions will be applicable when rsync attempts to read or write a file on a client's behalf. You may specify either a user name or a numeric user ID. The default is -2, which is nobody on many systems, but not on mine, which is why uid is defined explicitly.

gid = *`nobody`*: the gid option lets you specify with which group's privileges rsync should operate during file transfers, and it therefore affects (along with uid) which permissions apply when rsync attempts to read or write a file on a client's behalf. You may specify either a user name or a numeric user ID; the default is -2 (nobody on many systems).

max connections = *`20`*: this limits the number of concurrent connections to a given module (*not* the total for all modules, even if set globally). If specified globally, this value will be applied to each module that doesn't contain its own max connections setting. The default value is zero, which places no limit on concurrent connections. I do not recommend leaving it at zero, as this makes Denial-of-Service (DoS) attacks easier.

timeout = *`600`*: the timeout also defaults to zero, which in this case also means “no limit”. Since timeout controls how long (in seconds) rsync will wait for idle transactions to become active again, this also represents a DoS exposure and should likewise be set globally (and per module, when a given module needs a different value for some reason).

read only = ***`yes`***: the last option defined globally is read-only, which specifies that the module in question is read-only, i.e., that no files or directories may be uploaded to the specified directory, only downloaded. The default value is yes.

The third group of options defines the module [public]. These, as you can see, are indented. When rsync parses rsyncd.conf downward, it considers each option below a module name to belong to that module until it reaches either another square-bracketed module name or the end of the file. Let's examine each of the module [public]'s options, one at a time.

[*`public`*]: this is the name of the module. No arguments or other modifiers belong here: just the name you wish to call this module, in this case public.

path = ***`/home/public_rsync`***: the path option is mandatory for each module, as it defines which directory the module will allow files to be read from or written to. If you set the global option use_chroot to yes, this is the directory rsync will chroot to prior to any file transfer.

comment = ***`Nobody home but us tarballs`***: this string will be displayed whenever a client requests a list of available modules. By default there is no comment.

hosts allow = ***`near.echo-echo-echo.org, 10.18.3.12`*** and **hosts deny =** ***`*.echo-echo-echo.org, 10.16.3.0/24`***: you may, if you wish, use the hosts allow and hosts deny options to define Access Control Lists (ACLs). Each accepts a comma-delimited list of FQDNs or IP addresses from which you wish to explicitly allow or deny connections. By default, neither option is set, which is equivalent to “allow all”. If you specify an FQDN, which may contain the wildcard *, rsync will attempt to reverse-resolve all connecting clients' IP addresses to names prior to matching them against the ACL.

rsync's precise interpretation of each of these options depends on whether the other is present. If only hosts allow is specified, then any client whose IP or name matches will be allowed to connect, and all others will be denied. If only hosts deny is specified, then any client whose IP or name matches will be denied, and all others will be allowed to connect.

If, however, both hosts allow and hosts deny are present, hosts allow will be parsed first, and if the client's IP or name matches, the transaction will be passed.

If the IP or name in question didn't match hosts allow, then hosts deny will be parsed, and if the client matches there, the transaction will be dropped.

If the client's IP or name matches neither, it will be allowed.

In this example, both options are set. They would be interpreted as follows:

- Requests from 10.18.3.12 will be allowed, but requests from any other IP in the range 10.16.3.1 through 10.16.3.254 will be denied.
- Requests from the host near.echo-echo-echo.org will be allowed, but everything else from the echo-echo-echo.org domain will be rejected. Everything else will be allowed.

ignore nonreadable = *`yes`*: any remote file for which the client's rsync process does not have read permissions (see the uid and gid options) will not be compared against the client's local copy thereof. This probably enhances performance more significantly than security; as a means of access control, the underlying file permissions are more important.

refuse options = *`checksum`*: the refuse options option tells the server-side rsync process to ignore the specified options if specified by the client. Of rsync's command-line options, only checksum has an obvious security ramification. It tells rsync to calculate CPU-intensive MD5 checksums in addition to its normal rolling checksums, so blocking this option reduces certain DoS opportunities. Although the compress option has a similar exposure, you can use the dont compress option to refuse it rather than the refuse options option.

dont compress = *`*`*: you can specify certain files and directories that should not be compressed via the dont compress option. If you wish to reduce the chances of compression being used in a DoS attempt, you also can specify that nothing be compressed by using an asterisk (*), as in our example.

This simple example should get you started offering files for download by rsync. Next month, we'll cover setting up rsync modules (directories) at the filesystem level to accept anonymous uploads and authenticate users.

Building Secure Servers with Linux

Mick Bauer (mick@visi.com) is a network security consultant for Upstream Solutions, Inc., based in Minneapolis, Minnesota. He is the author of the O'Reilly book Building Secure Servers with Linux, composer of the “Network Engineering Polka” and a proud parent (of children).

Archive Index Issue Table of Contents

Advanced search

Original and Ultimate Communities

Doc Searls

Issue #107, March 2003

Will the Web ever support on-line communities like the best of what we had in the old days?

I am a pack rat; I save all kinds of stuff. But moving four times in two years has cheapened the growing tons of archival printed matter I began putting in boxes and storing away in my twenties. So lately I've been yielding to the urge to purge my life of crap.

In my latest purge I emptied 40 boxes; about half of them were filled with books going back to college days. Want a manual for QuickBooks 1.0? Netware 2.0? How about ones that compare WangNet, DECnet and OmniNet? All of those went into recycling, along with about half a ton of paper in loose-leaf binders.

But I kept one binder. It was full of printouts from on-line discussions. Some were from the Compuserve Broadcast Professionals Forum (BPF), and some were from a forum called The Buzz, which was run by my old friend, Denise Caruso, on AOL. The Buzz and Denise were the only reasons I had AOL.

Both The BPF and the Buzz were communities in the deepest sense that word can apply to a virtual space. The BPF was where disc jockeys, engineers, program directors and music obsessives would get together to ask and answer tough questions, help each other find better jobs and comment wisely on the gradual decline of a business they all loved, however corporatized and heartless it was becoming. The Buzz was a mix of techie and intellectual types that hit its peak during the Gulf War.

The BPF was a collateral casualty of Compuserve's gradual suicide, completed by its sale to AOL. The Buzz died faster than the BPF, mostly because nobody could stand staying on AOL. Both, however, were doomed by the same design flaw: everything posted scrolled to oblivion.

The main business model for both AOL and Compuserve back then was metered use. Compuserve also charged customers to download files. No value at all was placed on archiving what people said; that was up to the users. That was why I printed out so many of those old postings.

The Net and the Web are natural and liberating environments for communities. Nobody needs to depend on clueless and uncaring corporate entities. In fact, clueful corporate entities can get together with free-range hackers to improve everybody's environment. That's what's been happening with weblogging, which has produced RSS, XML-RPC, SOAP and other handy standards.

Weblogs form communities in much the same way that people do. Follow the links from Glenn Reynolds' Instapundit, and you'll find most are in agreement with Glenn's libertarian/conservative political philosophy. Glenn is widely considered the leading “warblogger”. Dave Winer, the prime mover behind the acronyms in the last paragraph, is widely considered the leading “techblogger”. There are blogs focused on photography, music, raising kids, women's issues, you name it. What makes them radically different from any other kind of on-line forum is they aren't contained by their categories.

Eric Olsen, whose main blog, TresProducers, is more or less in the warblogger camp, also is a music producer who has organized a bunch of fellow bloggers at Blogcritics.org. Group blogs and hot topics gather people the way cuisines cause restaurants to gather certain kinds of customers, who are also customers of other restaurants and fond of other cuisines.

Still, I think we're missing something we had in the best of those old on-line forums, especially The Well. One of my life's regrets was that I never participated in The Well, even though I went to the trouble of belonging to it. I have similar feelings about Woodstock: I drove people up there, then turned around and went home in the rain.

With all due respect to Slashdot, Kuro5hin and Advogato, I don't think there are any Wells on the Web yet—including The Well itself, which still exists. I'm on half a dozen e-mail lists that are excellent (I'd name them but I don't want to burden them with more participants than they already have), and most of them are also exposed on the Web. But I still don't think any of them meets The Well standard.

However, I think it will happen because I think we're still early in the Net's evolution. How will we know that Well-grade communities are happening big-time on the Web? Here's my guess: they'll matter politically. They'll mobilize to elect some people and prevent the election of others. They'll also bring down bad companies and industries and raise others up.

Why politics? Why muscular market action? Because the Web is a public place; it's the commons; it's where public communities gather; it's utterly uncontained. Ultimately, our communities are going to keep it that way.

Doc Searls is senior editor of *Linux Journal*.

“Roadcast Flag” Is MPAA's Latest Attack on Linux

Seth David Schoen

Issue #107, March 2003

Will the US government ban free device drivers for TV-capable hardware?

The Digital Millennium Copyright Act (DMCA) gave copyright holders powerful new rights to control implementations of proprietary protocols and media formats. Yet, it took a hands-off position toward non-proprietary formats. You still are free under the DMCA to implement a system based on open standards with the features of your choice.

But now, even this “no mandate” rule is under attack from proposals to regulate the use of open standards. The best-known is the Consumer Broadband and Digital Television Promotion Act (CBDTPA), introduced by Sen. Ernest Hollings. CBDTPA goes beyond the DMCA and requires that any “digital media device” must include “standard security technologies”.

The CBDTPA has been criticized from many directions. But entertainment industries have effectively split it into a series of more palatable pieces. The Motion Picture Association of America (MPAA) has spent the past year pursuing three new regulatory measures:

1. A “broadcast flag” mandate for all devices that can receive digital television broadcasts.
2. An “analog hole” watermark-detection mandate for “all devices that perform analog-to-digital conversions”.
3. An unspecified technology mandate for peer-to-peer file-sharing software.

None of these measures, individually, is as drastic as the CBDTPA. Mandates can be imposed gradually; technology industries may not launch an all-out campaign against any particular measure. Tech vendors may be divided and choose not to oppose measures that don't affect them directly.

MPAA's lobbying for a “broadcast flag” mandate is already underway. This proposal is specific to US terrestrial digital television broadcasts. But it's the first part of a larger regulatory agenda that needs to be stopped now—movie studios should have less, not more, control over our technology.

Current-generation US television broadcasts use the 50-year-old analog NTSC transmission standard. In 1996, the US adopted a plan to replace NTSC gradually with a next-generation digital standard, ATSC, capable of carrying high-definition TV pictures. This transition was meant to finish around 2006 with the complete discontinuation of analog TV broadcast. Meanwhile, stations are broadcasting both analog and digital TV signals (using additional channels lent to them by the government). ATSC-capable home theater equipment and PC adapters are now available. The transition is underway.

ATSC is an open standard, and terrestrial broadcasts using it are always unencrypted. Therefore, the DMCA does not restrict the kinds of features that can be included in an ATSC receiver. Hollywood studios have begun to complain that this means ATSC broadcasts are too insecure for transmission of Hollywood movies—if anybody can build a receiver, the movies could be used in ways the studios might wish to prevent.

The studios have argued that the “insecurity” of ATSC receivers is deterring them from licensing their movies for ATSC broadcast. They regularly license their movies for unencrypted NTSC broadcast, but the superior quality of ATSC purportedly makes it a more attractive target for copyright infringement. Instead of encrypting the broadcasts, however, the studios have proposed new legal restrictions that limit how a TV signal can be used on receiving devices. According to the studios' proposal, the TV receiver, whether hardware or software, would have to be “compliant” and “robust” against modification by an end user.

Instead of encryption, the proposed measure is merely a label or header akin to an e-mail header defined to signal “technological control of consumer redistribution”. Such a measure is no more effective than an “X-No-Forward:” e-mail header, but a law or regulation could deem it illegal to fail to honor such a header.

This regulation leads us away from a world where people are free to implement open standards as they choose. ATSC has been such a standard, but the benefits of its openness, including support by free software, could be lost.

“Compliance” requirements limit permissible outputs to a short list of restricted formats, implementations of which are subject, in turn, to DMCA restrictions. Open standards need not apply. This creates a tremendous incentive for consumer electronics manufacturers to implement proprietary technologies. Open interfaces, which one studio representative referred to as “legacy interfaces” because they lack copy controls, would suffer a substantial competitive disadvantage.

“Robustness” requirements prevent programmers from developing free drivers for ATSC, such as the free drivers already being developed for ATSC tuner cards. They also would ban free software implementations of ATSC, such as FSF's GNU Radio Project, which interprets radio signals in software. If software can be modified by an end user, it can't be trusted to control how users use it—the same argument the MPAA uses against the development of free DVD players. The broadcast flag rules could result in a ban on certain kinds of free software specifically because that software can be modified by end users.

The Federal Communications Commission (FCC) is now considering whether to mandate that TV-receiving equipment comply with these rules.

EFF is fighting this proposal and recently filed comments with the FCC against it. We attended all the meetings of the Broadcast Protection Discussion Group where the proposal was developed. With our “Consensus At Lawyerpoint” web site, we've tried to fill the information vacuum surrounding the broadcast flag issue. (Reporters were barred from meetings, which had a $100-per-meeting admission fee.)

email: schoen@loyalty.org

Seth David Schoen created the position of EFF staff technologist, helping other technologists understand the civil liberties implications of their work, EFF staff better understand the underlying technology related to EFF's legal work, and the public understands what the technology products they use really do.

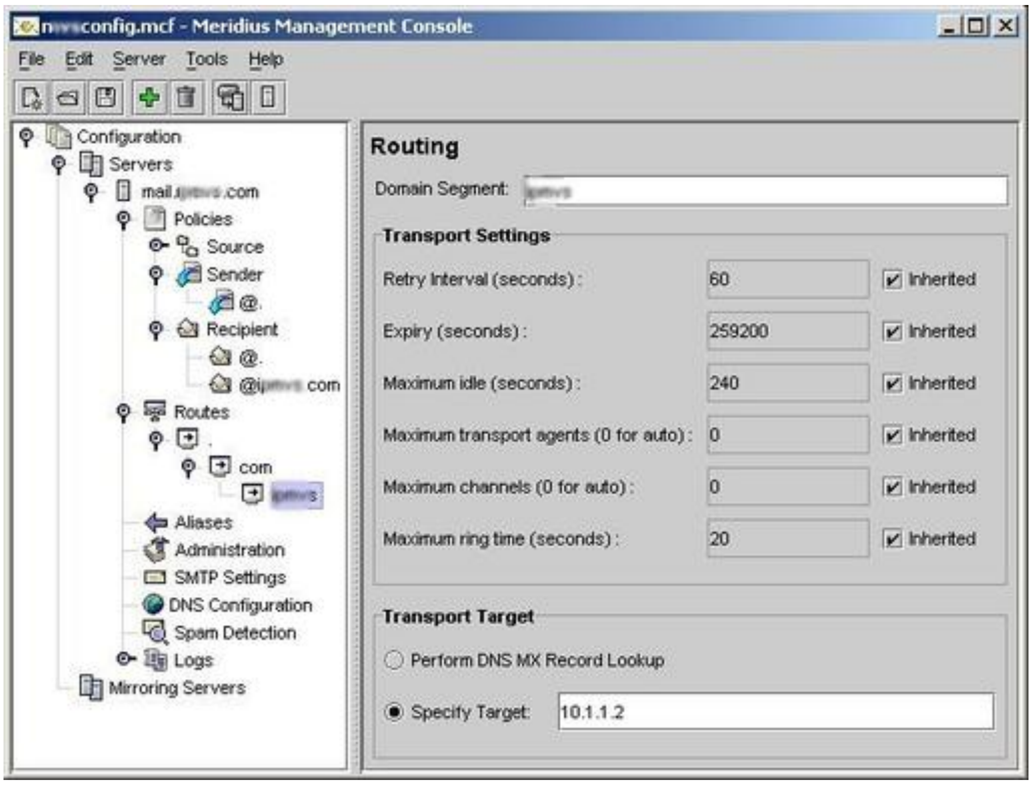
BlueCat Networks' Meridius Mail Relay

Nathan Smith

Issue #107, March 2003

The Meridius Mail Relay is designed to relay mail in order to protect your internal mail server and filter out spam.

The Meridius Mail Relay is a rackmountable (1U) Linux-based mail appliance. It is designed to relay mail in order to protect your internal mail server and filter out spam. The Meridius is placed on the network between the Internet and your e-mail server in a DMZ on the Internet or behind the firewall.



Routing

Domain Segment: [redacted]

Transport Settings

Retry Interval (seconds): 60 [✓ Inherited]

Expiry (seconds): 259200 [✓ Inherited]

Maximum idle (seconds): 240 [✓ Inherited]

Maximum transport agents (0 for auto): 0 [✓ Inherited]

Maximum channels (0 for auto): 0 [✓ Inherited]

Maximum ring time (seconds): 20 [✓ Inherited]

Transport Target

Perform DNS MX Record Lookup [✓ Specify Target:] 10.1.1.2

Figure 1. The Java-Based Management Console

The hardware specifications are likely to change with fluctuations in pricing and availability, but the published specification is an Intel Pentium III running at 800MHz, with a 20GB hard drive and 256MB of RAM with two network ports.

The Meridius is based on several software packages, including Zmailer, SpamAssassin, Apache and Tomcat. It is managed by a custom Java-based management console.

Network configuration, including the IP address, netmask and gateway, is done with small buttons and an LCD panel on the front of the unit. The Meridius is then accessed using a browser, where a web page offers three options: open the management console, install Java Web Start or read the system documentation (a PDF file).

The first-time configuration of mail policies is performed using a wizard-like tool. The wizard was enough to get a small organization like mine up and running. The console is used for advanced configuration, such as routing mail within an organization or within an ISP. Policies also can be configured to refuse mail based on IP address, domain or e-mail address. E-mail aliases for individual users also can be created and maintained in the console. All of these settings, familiar to a mail administrator as part of the configuration files, are easily set and modified in the Meridius console.

I placed the demonstration unit behind a firewall running NAT, which caused a few problems that were not immediately apparent. When I had trouble getting the antispam features to work, BlueCat's able technical support helped me analyze the logs. There we determined that all inbound e-mail was coming from the same address—the address of our firewall. The Meridius was relaying mail for our domain properly before we uncovered this problem, but the antispam features did not work. After we enabled reverse-NAT on the firewall, the Meridius could see the true IP addresses and domains of the inbound mail, and the antispam features began to work.

We use Microsoft Exchange 5.5 as our e-mail system, so we set the Exchange server to route all mail to the Meridius and set the Meridius to route all inbound mail to Exchange. I tested the Meridius in my organization for over a month in a production environment. There were no noticeable changes in the speed of our internet e-mail after the Meridius was added. We receive around 800 messages a day, and the Meridius had no trouble keeping pace.

In my early experimentation with the Meridius, I somehow managed to download the 1.2 system update before I should have, which caused the Meridius to stop working correctly. The mistake required me to restore the Meridius from disk, and the system restore process was straightforward and relatively simple. A monitor and a keyboard are connected to the Meridius, which is then powered on with a boot disk in the drive, automatically restoring the system. It was easy to re-install the settings I had previously configured, as they are saved from the Java console to a location the administrator chooses.

An update has not been issued since 1.2, the first update, but I am told updates should go much more smoothly in the future.

I had questions about the functions of the device several times, and the people at BlueCat Networks were always responsive and helpful. From e-mailing and talking to them, I got the feeling that they are a small company that cares about releasing quality products.

Spamazon.com?

I was most excited about getting the power of SpamAssassin with Meridius, without going through the learning curve of installing and configuring. The users, however, were not entirely delighted when messages with a subject starting with “** SPAM ” began arriving in their inboxes, with blocks of technospeak at the top and no human-readable HTML messages. BlueCat Networks was in the process of releasing a fix (1.2) to restore HTML e-mail. User acceptance of Meridius' antispam component was probably the hardest part of installing the appliance. Some users were offended that advertisements they got from Amazon.com were labeled as spam. I wrote up some instructions on how to create a folder for spam and pass anything with the “X-Spam-Flag: yes” setting into it. I was happy with the anti-spam performance in my own mailbox. After the Meridius installation, the signal-to-noise ratio in my inbox improved significantly. I now have an average of 45 messages dropping nightly into the folder I created for spam. Meridius made going through morning messages a breeze.

I am pleased how simple the Meridius is to administer. When a user forwards me a message with an address or domain they want placed on the whitelist, I simply copy the original sender's e-mail address and paste either the address or the domain into the console. After running the update process, the new changes take immediate effect.

BlueCat Networks is hinting at a lot of improvements and more functionality to come in new versions of the software, which can be downloaded. I would like to see context-sensitive help in the administration console and user-submitted whitelists.

Does the functionality and usability of the system offset its $7,000 US price tag? If Meridius is used only for relaying messages, I think the answer may be no for many organizations. A lot of administrators could put similar functionality together for less money. With the Meridius you are paying for good hardware, support and the administration tool assembled and tested as a polished product. I believe an organization like ours (about 50 users) is probably on the small end of company sizes that would make such an investment. Used correctly, over the life of the device, the anti-spam feature could easily save our professionals $7,000 US in billing hours. For perspective customers, BlueCat Networks offers a web demonstration and an evaluation unit of the Meridius.

My experience with the Meridius has been a pleasant one. If you would rather spend your time doing interesting things instead of sorting through spam messages, worrying about open relays and administering electronic mail systems, you may find the Meridius is a good fit for you too.

Product Information

email: smith@ipmvs.com

Nathan Smith (nathan.smith@rockinghamridge.com) is a systems administrator for an intellectual property law firm. A Windows user but a Linux and *BSD aficionado, he is always looking for ways to move toward a more open computing environment.

Archive Index Issue Table of Contents

Advanced search

Mathematica 4.2

Dragan Stancevic

Issue #107, March 2003

Mathematica is the Swiss Army Knife of technical computing.

It is hard to describe what Mathematica is and what it is used for because it can be used for so many diverse things. If I were asked “What is Mathematica?” by someone who has never heard of it, I would probably say it is the “Swiss Army Knife of technical computing”. That might not exactly answer the question, but once they got to know Mathematica they would understand the meaning of the reference.

To elaborate on my Swiss Army Knife characterization, here are a few fields in which Mathematica is in use today: engineering, physics, computer science, publishing, finance/economics, mathematics, social sciences and life sciences.

Besides Mathematica's wide field of application, the support for various hardware platforms and operating systems is just as varied. Mathematica 4.2 runs on the following operating systems: Linux, Mac OS X, Mac OS 8.1 or later, Windows, Solaris, Tru64, HP-UX, AIX and IRIX.

If we concentrate only on the Linux release of Mathematica, we'll see that to Wolfram Research, Linux is not just another operating system to which to port. The Mathematica kernel is entirely developed on Linux and then ported to other platforms. No wonder the Linux release of Mathematica actually runs on four different architectures: IA-32 (x86), PowerPC, Alpha and IA-64 (Itanium). There is also a version of Mathematica for Linux clusters called gridMathematica.

There is no doubt that Linux is an important market for Wolfram Research both in professional and student applications. The commercial license might be a bit too pricey for a casual home user, but it's well worth it for an enterprise or a research environment. On the other hand, the student version of Mathematica is a bargain and has exactly the same features as the commercial version.

The Mathematica 4.2 package contains: “Installing Mathematica 4.2 for Unix and Linux”, “Getting Started with Mathematica”, *The Mathematica Book* (Fourth Edition), *Standard Add-on Packages* and Mathematica 4.2 (installation CD).

“Installing Mathematica 4.2 for Unix and Linux” includes information on UNIX and Linux installation and describes the upgrade process from earlier versions.

The “Getting Started with Mathematica” guide is a 50-page booklet describing the basics of Mathematica. It is a good reference for beginners as it is a fast crash course on usage and interaction with Mathematica. The guide also lists various on-line resources, which are a must-see for new users.

The Mathematica Book (Fourth Edition) is an approximately 1,500-page book on Mathematica functions and capabilities. This has got to be the most definitive resource on the subject. It contains more information than I could ever possibly describe for this review.

Standard Add-on Packages is a 500-page book dedicated to more than 1,000 additional functions that are in the Standard Add-on packages. Standard Add-on packages are a part of the full version of Mathematica and include, but are not limited to graphics, geometry, audio, statistics, units, algebra, calculus, discrete mathematics and numerical mathematics. The subject of this book is somewhat complementary to *The Mathematica Book*, and I felt that it should really be a part of it, especially considering the fact that the Standard Add-on packages are distributed with all releases of Mathematica.

Mathematica 4.2 for Linux runs on three different platforms: x86, Alpha and PowerPC (an Itanium version has just been released, so it wasn't on the CD). The media also contains both *The Mathematica Book* and the *Standard Add-on Packages* book. The books are in the Mathematica “Notebook” format, which is browseable and searchable. Having the books in such a format is a great reference while working on your own creations in Mathematica, whatever they might be.

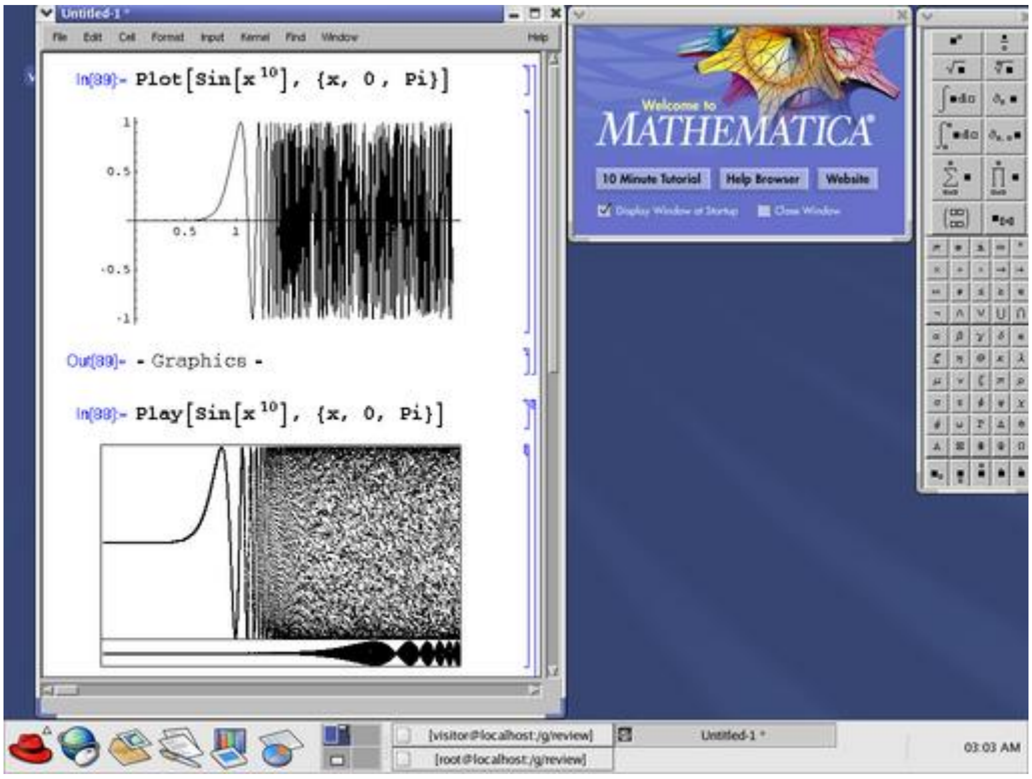
Mathematica installation is pretty straightforward. Pop the CD into your CD-ROM; go into the appropriate platform subdirectory under the Installers directory and run the MathInstaller script. At the end of the installation, the installer will ask for your license number and password. It is okay if you don't have all the required information at this time. You will be prompted again later when you run Mathematica.

Your license number always remains the same, even when you upgrade. Your password is generated by Wolfram Research when you register. In order to give you a password, they need your license number and your MathID, a value that is generated dynamically when Mathematica starts up.

From my own experience I wouldn't recommend installing Mathematica on a system where the configuration changes frequently. If the system configuration changes, it's very likely that the MathID will change too. This also holds true for upgrading to a new kernel. What this really means is that you have to contact Wolfram Research and get another password. I am sure that this helps them prevent some opportunistic unauthorized copying of the program. Having said that, I found it a nuisance.

The fundamental idea behind Mathematica is what makes it such a versatile and powerful computing platform. Everything in Mathematica is represented as a symbolic expression typically looking like “Sin[x]”. And that is the key because any function in Mathematica can be an output to or take input from any other function. This opens up so many possibilities I was literally overwhelmed by the sheer number of things I could try for the review.

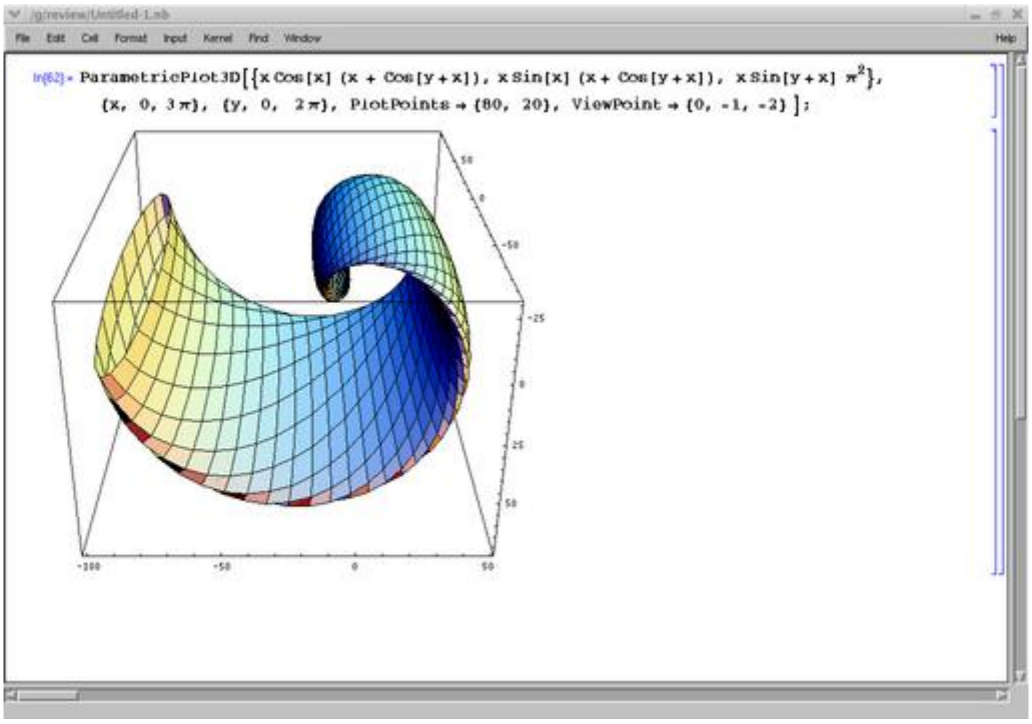
To help you better visualize the “any to any” concept of Mathematica let's take a look at the Sin function in Figure 1. The function Sin takes the variable x10 as input and outputs to the function Plot. This makes a nice plot of the values that the function Sin produces. If we decide we want something other than a regular plot we can simply represent the Sin function output with a different function. Play for example, produces an actual sound that is heard when you click on the picture.


Untitled-1 *
File Edit Cell Format Input Kernel Find Window Help
Welcome to MATHEMATICA
10 Minute Tutorial
Help Browser
Website
Display Window at Startup
Close Window
[visitor@localhost:/g/review]
[root@localhost:/g/review]
Untitled-1 *
03:03 AM

```
In[89]:= Plot[Sin[x^10], {x, 0 , Pi}]
Out[89]= - Graphics -
In[88]:= Play[Sin[x^10], {x, 0, Pi}]
```

Figure 1. Some 2-D Graphs

I also wanted to explore the 3-D capabilities of Mathematica. As you can see in Figure 2, I am using the Sine and Cosine functions as input for the "ParametricPlot3D" function. As I finish typing the expression, I press Shift-Enter and Mathematica produces a 3-D plot of the data.



```
In[62]:= ParametricPlot3D[{x Cos[x] (x + Cos[y + x]), x Sin[x] (x + Cos[y + x]), x Sin[y + x] π^2},
    {x, 0, 3 π}, {y, 0, 2 π}, PlotPoints → {80, 20}, ViewPoint → {0, -1, -2} ];
```

Figure 2. Exploring the 3-D Capabilities of Mathematica

Those are just a few out of the 1,100 built in and 1,000 additional functions that come with the 4.2 Mathematica package. In the limited amount of time I had for this review, I felt that I had not even scratched the surface of the computing system that is Mathematica.

Not only that, Dr Michael Trott of Wolfram Research, the author of the book *Graphica*—www.graphica.com, generously provided me with the full notebooks (2GB) of his upcoming book series called *The Mathematica Guidebook Series*. It was a real treat to browse, experiment and evaluate the expressions that produce some of the stunning images that appear in Wolfram Research publications—images such as the "Mathematica 4 Dodecahedron" from the cover of *The Mathematica Book* and many, many more. I can't wait until the book series is finished.

Evaluation of Mathematica was done on two different architectures and two different operating systems. I personally reviewed the Linux version on x86 while a mathematician, Dr Helen Moore, worked on the Mac OS X release. Helen helped with the evaluation, although this review is solely my own opinion.

Using Mathematica on the two platforms was basically analogous, and both Helen and I liked the product. However, we were a bit vexed with the registration and password creation process. Despite these issues, we would still recommend Mathematica for Linux or Mac OS X to anyone.

The people at Wolfram Research were really willing to listen and work on all the issues that I brought up (great service). Mathematica for Linux is a great product. It's probably the most beautiful piece of Linux software I have ever used. During my review, I couldn't help but spend hours and hours just experimenting with the capabilities and exploring the options.

Helen said, “Mathematica is great. It's pretty much the best commercial package available, both for symbolic and numerical computations.”

Product Information

Acknowledgements

Special thanks to Dr Helen Moore for her help with the evaluation, Dr Michael Trott for letting me use his notebooks, and Ben Wilson and PJ Hinton of Wolfram Research for their help and quick answers to my queries.

email: visitor@xalien.org

Dragan Stancevic is a kernel and hardware bring-up engineer in his late twenties. Although Dragan is a software engineer by profession, he has deep interest in applied physics and has been known to play with extremely high voltages in his free time.

Letters

Various

Issue #107, March 2003

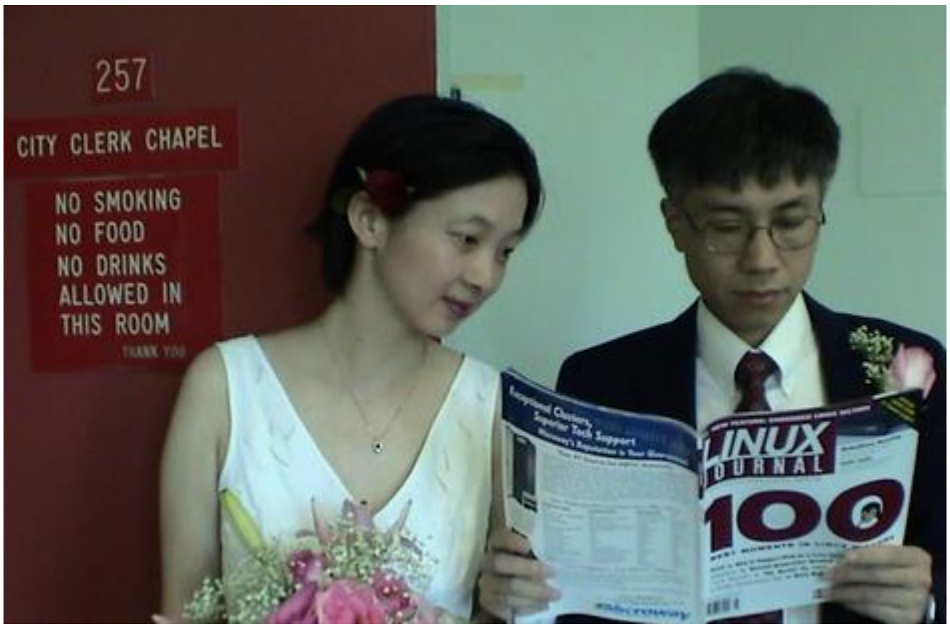
Readers sound off.

Letters

Marry Me, Marry My *LJ*

This photo was taken when my wife and I were waiting to get married in the chapel on the second floor of New York City Hall. I just received the 100th issue of *LJ* and brought it along. You can tell from the photo that my wife was a little jealous. So the title for the photo is “Beauty or *LJ*”. Hope you like it.

—Ning Qian



Beauty or *LJ*: Bride-to-be Qing Xiang peeks at fiancé Ning Qian's *Linux Journal* (photo: Nicole Xiang).

Two FTP Proxies?

I enjoyed reading your article about setting up an FTP proxy [*LJ*, December 2002]. I was just curious about one point. You make a comment about not being able to configure the acceptable commands list differently for internal and external users. Is it possible to set up two proxies then route the incoming requests at the firewall for internal addresses to one proxy and those from external addresses to the other?

—John

Mick Bauer replies: That's an excellent idea! You could set up a proxy on the firewall for external users, with read-only permissions, and set up a proxy on some host on the inside for outbound transactions, with looser permissions. You could then configure your firewall to permit outbound FTP only if it originates from the designated internal proxy.

Acronym Recycling Department

It's great to see those huge old machines coming back! I saw an advert in *Linux Journal* for a CDC 6400, the baby brother of the immense CDC 6600. Of course today we could not use Freon to cool it. And software is coming back! SOAP, for instance, the Symbolic Optimal Assembly Program for the IBM 650. Magnetic drum memory will never die! I have even seen articles about ASP, the Attached Support Processor for the IBM 360/65. Who says I'm a dinosaur?

—Peter Chase, Alpine, Texas

Is ptrace Secure?

The articles on ptrace in the November and December 2002 issues were very informative. So what's to stop someone from using ptrace to insert some malevolent code in a running program? Forgive me for looking on the dark side.

—Walter S. Heath, Concord, Massachusetts

Keep thinking evil thoughts. You can't keep a system secure without studying possible attacks. Fortunately, you can only ptrace a process if it's your own process, or if you're root.

—Editor

Another OpenLDAP Win and a Fix

I have found an error in my article, “OpenLDAP Everywhere” [*LJ*, December 2002]. In the auto.home section on page 54, the gomerp entry has the line:

```
cn: super3
```

The line should read:

```
cn: gomerp
```

A reader contacted me after copying the entry exactly from the article. He has fixed his configuration and is up and running with unified logon. Matt Lung and I are very pleased with the article as published. Our mothers are very proud!

—Craig Swanson

Where's zNav?

We were a little disappointed by your article on the applications for the Zaurus. You listed many commercial applications and yet when it came to mapping/navigation software you listed only the free one. zNav and zNav Lite are the only Zaurus mapping/navigation products that use commercial navigation charts.

—Patrick Cannon, Barco Software, LLC

Installfest at 11,000 Feet

Here the world's highest Linux install event took place Saturday the 12th of December 2002, 11,000 feet above the sea. It was the first of its kind in this part of Bolivia—and maybe a world record when it comes to altitudes and Linux install events.

The organizers task force: Janus Sandsgaard, Edgar Ruiz, Juan Conorel, Devin Conde, Hardy Beltran, Christian Mollo and Jorge Castro.

—Janus Sandsgaard

Zaurus Articles Monthly, Please

Yesterday I was very happy to see that the Zaurus was on the cover of the January 2003 issue of *LJ*. I am involved in the OpenZaurus Project, and I would really appreciate a monthly section on the Zaurus on *LJ*.

—Paolo

tkcGallery vs. MooView

When you mention the Zaurus applications tkcGallery and MooView [*LJ*, January 2003], you completely leave out the point that MooView cannot handle even reasonably sized files, just like the included image viewer, while tkcGallery can handle virtually any size image. I've tested up to ten megapixels so far. The general feeling here is that if it is a commercial app you must make some negative mention, and if it is free then there must be nothing wrong with it.

—Shawn Gordon, President, theKompany.com

Guylhem P. Aznar responds: I just could not recommend every single application. Yes, I did mention the problems I had and the existing free software equivalent if there were any, but I also gave strong purchase advice for some unique software such as tkcJabber.

Meet Linux Professionals at Borders

Go to a Borders bookstore some Saturday as I do and pick up a book on Linux, or *Linux Journal*, and read it. Stay there for two hours and you will have a minimum of two people come up to you and tell you of their companies' recent conversion from Windows to Linux servers.

—Charles Ebert

Hey! This isn't a library! Are you going to buy that?

—Editor

Thanks for the Clues, *LJ*

Somehow I managed to forget to bring reading material for my Thanksgiving flight, so I found myself looking through the junk that the airports had to offer. My brother said I should get a magazine but I refused to pay five bucks for something without content. I returned home to my latest *Linux Journal*. In every issue you folks manage to cram lots of good stuff onto those pages. So many computer magazines are more like bridal zines, to be purchased only when one is in the market for something new. *Linux Journal*, on the other hand, puts together great articles for people who actually use the machines they own. So I just wanted to send out a thanks. Some days I think your organization is the only place that has a clue as to what people want.

—Tim

Thumbs-up on Bluecurve

I have to disagree with the comments made by Tom Amon in your January 2003 Letters section (entitled “Red Hat 8.0: Love the License, Hate the Look”). Red Hat has made an attempt to bring forward a “best of breed” desktop. Having KDE (or GNOME if you wish) pre-setup with Mozilla, Evolution and OpenOffice seemed, to me, a vast improvement over the 7.3 release. I'll concur with Tom that the result is not perfect (where did Xine go, Red Hat?), but this is a road I would like Red Hat to stay on.

—Timothy J Halloran, Carnegie Mellon University

Hooray for Zaurus and *LJ* Ads

An excellent article on how to use your Zaurus and what various hardware and software options are available [*LJ*, January 2003]. The Zaurus is a system that can stand on its own, and more articles on it would be welcome. I can also say that I find the advertising in your magazine an excellent resource of what's available in the Linux world.

—Julian Macassey

Why Java without the VM?

The author of “Compiling Java with GCJ” [*LJ*, January 2003] misses the point of what makes Java a more viable solution than any natively compiled language, namely, platform neutrality á la the Virtual Machine. Simply put, natively compiling Java defeats the primary purpose of why Java was invented in the first place. I compile GIS Java code once on Linux and run that same code on Mac OS X, AIX, Solaris and more, with great success and eye-popping performance. A key Java goal is to limit the long development cycle time of software by avoiding native porting overhead.

—Bryan McKinley

Rackspace vs. Spam Blacklists

On page 15 of the January 2003 *Linux Journal* you have a full-page ad from Rackspace. I would advise anybody looking for hosting NOT to go to Rackspace. They are well known for hosting spammers. The “100% network uptime” they boast about in the ad does you no good if your class C is blackholed due to the spammers you share it with. I would advise anybody seeking hosting to closely check the anti-spamming resources before contracting for hosting.

—David D. Hagood

Likewise, you should check out any spam-filtering service you plan to use to see how much legitimate mail gets blocked because of who the sender's network neighbors are. The www.linuxjournal.com web site is at Rackspace, and it works fine for us.

—Editor

Go on, Take the Microsoft Ad

I enjoyed both Renato Carrara's letter in the December 2002 *Linux Journal*, and Gary Nickerson's reply in the January 2003 issue. I don't think that it's necessarily dangerous for a (relatively) small-circulation publication like *Linux Journal* to sell advertising to Microsoft; Bill's money is just as green as anyone else's. However, as a publication grows, it needs to beware of developing a dependency on large, powerful advertisers such as Microsoft. High overhead can make them susceptible to the threat (explicit or unstated) of one or more large advertisers pulling their ads. I hope that *LJ* can grow at a measured pace and thus maintain its independent spirit and its outstanding content. In the meantime, I appreciate your great work in building a very useful and readable publication. Please keep it up. And, hey, if you're able to siphon off a few bucks from The Beast along the way, more power to ya.

—Ken

Thanks for Not Taking the Microsoft Ad

Regarding the letter “An Ad Is an Ad” in the January 2003 issue, I think the decision not to run ads from Microsoft is a great one. If I need information from them I know exactly where to find it. The ads in *Linux Journal* nowadays are really a welcomed break from other “infested” publications. I think it's really important that you guys keep your focus on the values of where you came from, since it really makes a big difference for us readers. Truly, it's a joy to read your magazine, all the fresh people and talk and the lack of all the corporate bull.

—Boris Debic, San Mateo, California

Archive Index Issue Table of Contents

Advanced search

Upfront

Various

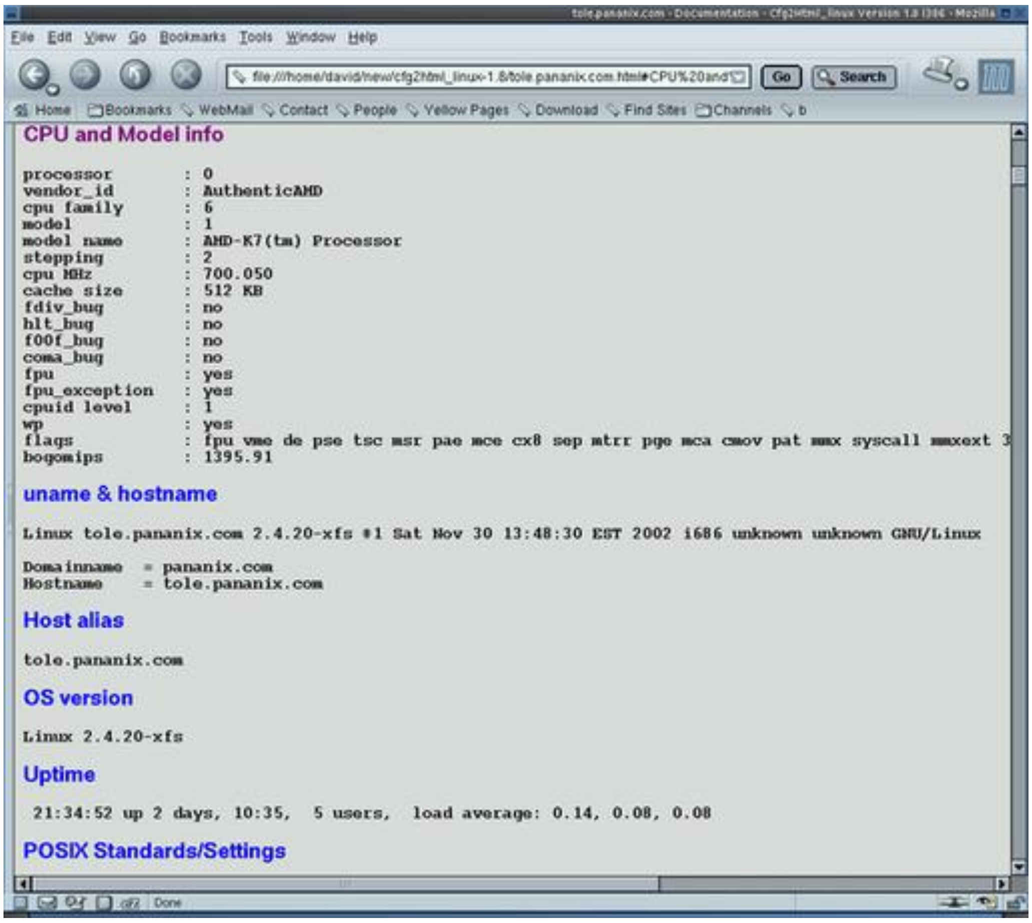
Issue #107, March 2003

LJ index, diff -u and more.

upFRONT

cfg2html: members.tripod.com/rose_swe/cfg/cfg.html

This utility will go over your system and extract a lot of information, enough information that you should be able to rebuild the system almost exactly as it was. That's probably a little more detail than I'd be comfortable with putting on a web site, but great to print out and put in your system's notebook (your systems do have notebooks, right?). Requires: BASH and standard UNIX tools.


tole.pananix.com - Documentation - Cfg2Html_linux Version 1.8 i386 - Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///home/david/new/cfg2html_linux-1.8/tole.pananix.com.html#CPU%20and
Go
Search
Home
Bookmarks
WebMail
Contact
People
Yellow Pages
Download
Find Sites
Channels
Done

CPU and Model info

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 1
model name      : AMD-K7(tm) Processor
stepping        : 2
cpu MHz         : 700.050
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat mmx syscall mmxext 3
bogomips        : 1395.91
```

uname & hostname

```
Linux tole.pananix.com 2.4.20-xfs #1 Sat Nov 30 13:48:30 EST 2002 i686 unknown unknown GNU/Linux

Domainname   = pananix.com
Hostname     = tole.pananix.com
```

Host alias

```
tole.pananix.com
```

OS version

```
Linux 2.4.20-xfs
```

Uptime

```
 21:34:52 up 2 days, 10:35,  5 users,  load average: 0.14, 0.08, 0.08
```

POSIX Standards/Settings

—David A. Bandel

diff -u: What's New in Kernel Development

Shortly after the kernel's Halloween feature-freeze, **Guillaume Boissiere** decided to put together some statistics on the incorporation of features into the 2.5 tree. He examined almost the entire history of the 2.5 development cycle, starting in early 2002. He created seven possible status categories for any given feature: planning, started, alpha, beta, ready for inclusion, pending inclusion and fully merged. His first progress chart (Figure 1) shows the percentage of features in each category. The second progress chart (Figure 2) shows the actual number of features in each category, changing over time. Without making any claim to complete accuracy, the graphs are interesting, if for no other reason than to observe how seriously most developers took the drive toward feature-freeze. Note also the hump of work done over the summer, followed by a complete end to new feature planning. That hump of activity corresponds roughly to the time when the decision was made to freeze by November.

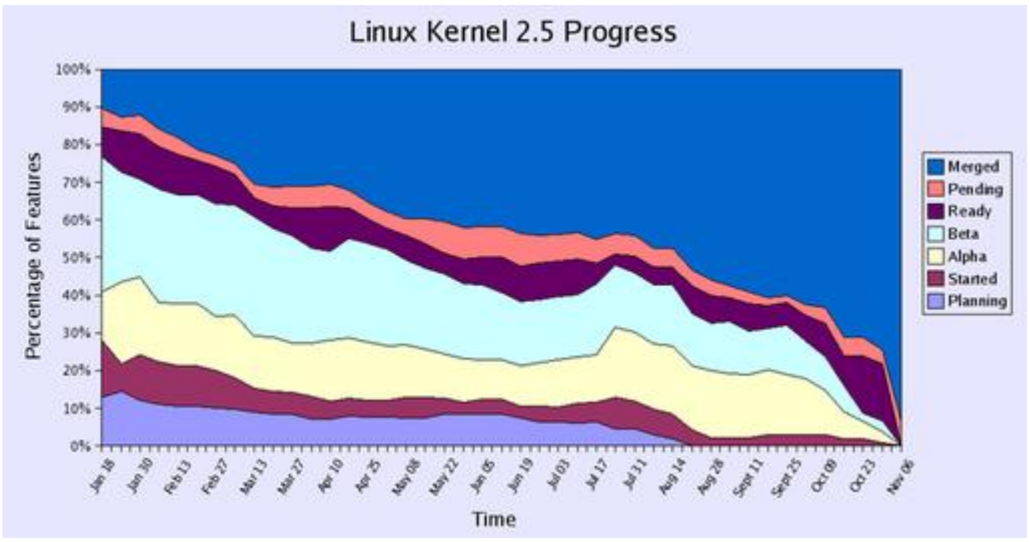


Figure 1. The Percentage of Features in Each Category

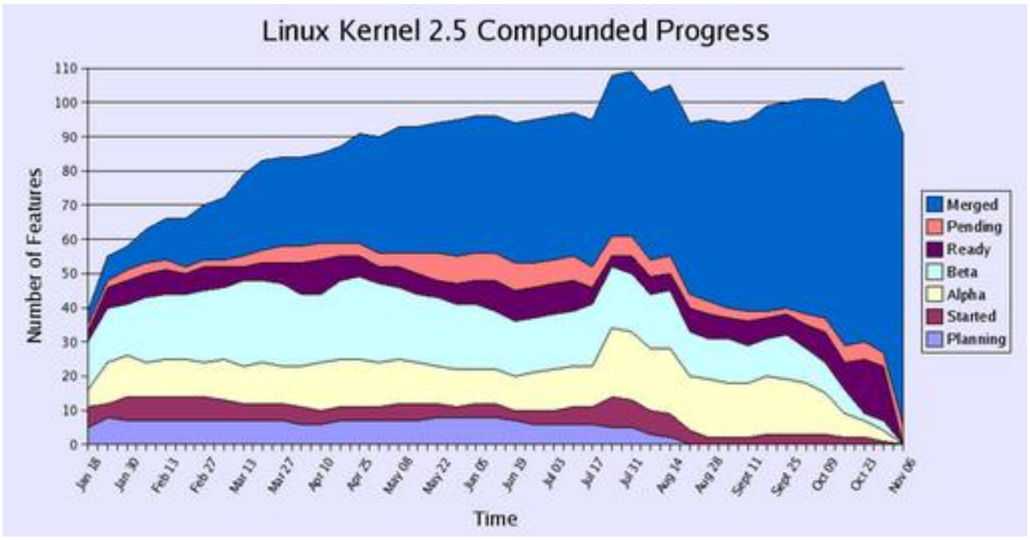


Figure 2. The Actual Number of Features in Each Category

To facilitate the movement from feature-freeze to actual 2.6 (or 3.0) release, the **Open Source Development Lab** (OSDL) donated labor and equipment to maintain a Bugzilla bug-tracking database for the Linux kernel at bugzilla.kernel.org. Support for this was initially strong among developers, but it tapered off a bit when big guns, like **David S. Miller**, found duplicate entries and frivolous reports made the system, at least in its original form, more trouble than it was worth. No one wanted to give up on it, however, and a concerted effort seems to be underway to bring the bug database to a usable state.

In more debugging news, **Linus Torvalds** indicated for the first time he might be willing to accept patches into the kernel to support a **kernel-based debugger**. Traditionally, Linus' stance has been that real programmers debug from source files. While not actually explaining the reason for his change of policy, he now seems to think that a kernel debugger running across a network would be a good feature to let into the kernel. Don't look for it in the next stable series, however, as he was careful to make this statement after the feature-freeze had passed.

A new read-only compressed filesystem, along the lines of cramFS, emerged in late October and targets the 2.4 kernel. **SquashFS** claims to be faster and to produce tighter compression than either zisoFS or cramFS. The author, **Phillip Lougher**, wanted to address some of the limitations of other compressed filesystems, particularly in the areas of maximum file size, maximum filesystem size and maximum block size.

And speaking of filesystems, does anyone remember **xiaFS**? In 1993 it was regarded, along with ext2fs, as a serious contender for world domination. In fact, the two filesystems leapt into public use within a few weeks of one another. For a while it even looked as though xiaFS had taken the lead. By 1994, however, it had essentially dropped off the map, and a few years later it was actually dropped from the official kernel tree. In 2000, Linus remarked that it would be fun to have it back. Finally, just after the Halloween freeze **Carl-Daniel Hailfinger** asked if this offer was still good. Linus said sure, and even offered to make an exception to the feature-freeze, if Carl could deliver the patches.

—Zack Brown

LJ Index—March 2003

1. Percentage of movies released between 1927 and 1946 that are currently unavailable: 93
2. Number of government desktops converted to Linux in Spain's Extremadura region by November 2002: 10,000
3. Number of government desktops expected to be converted to Linux in Spain's Extremadura region by November 2003: 100,000
4. Downloads of Extremadura's own Linux distro, Linex, from outside the district: 55,000
5. Dozens of countries with laws encouraging free software: 2
6. Number of free software laws or policies pending in those countries: 70
7. Number of Linux management tools in IBM's Tivoli in 2001: 2
8. Number of Linux management tools in IBM's Tivoli in 2002: 20
9. Percentage of IT managers employing Linux “in some capacity”: 39
10. Number of different Linux-based PDAs: 23
11. Number of servers in a Linux cluster installed at the University of Buffalo in September 2002: 2,000
12. Number of servers in another Linux cluster installed at the University of Buffalo in November 2002: 300

Sources

- 1: Jason Schultz, in a letter to Lawrence Lessig
- 2-6: *Washington Post*
- 7, 8: *Information Week*
- 9: Goldman Sachs Research, IDGnet
- 10: LinuxDevices.com
- 11, 12: *Boston Globe*

Listening Post: On-line Fora as Art

Setting the mood for this month's issue is our cover photograph of Mark Hansen and Ben Rubin's *Listening Post*, currently featured at the Whitney Museum of American Art in New York (www.whitney.org). This remarkable installation runs on four computers and as many operating systems (including Linux, of course) and expresses the collective voice of the Internet, transforming on-line communication into a multimedia installation.

According to the *Listening Post* web site, "statistical analysis organizes the messages into topic clusters based on their content, tracking the ebb and flow of communication on the Web. A tonal soundscape underlies the spoken text, its pitches and timbres responding to changes in the flow and content of the messages."

I was lucky enough to "view" the *Listening Post* when it was in Seattle in November 2002, and my first impression was an almost eerie sense of humanness the piece unveils—a poetry not typically associated with computer technology. In a dark room complete with pillows on the floor, a wall of tiny screens depict glowing green text gathered in real time from thousands of public on-line communication channels. These bits of text are accompanied by a computer-generated voice with a British accent, randomly speaking different messages as they flash by. I was particularly struck by the "I am" series; real-time messages beginning with the string "I am" spoken into the darkness: "I am tired." "I am happy." "I am Norwegian." Hundreds of people communicating the most basic aspects of themselves at that precise moment from who knows where to who knows who, while my imagination worked double time wanting to fill in the rest of their stories.

Capturing the ephemeral nature of the *Listening Post* is difficult; however, Ben Rubin, one of the creators of the *Listening Post*, best describes the piece in his artist's statement:

> Anyone who types a message in a chat room and hits "send" is calling out for a response. *Listening Post* is

> our response—a series of soundtracks and visual arrangements of text that respond to the scale, the immediacy and the meaning of this torrent of communication.
>
> Every word that enters our system was typed only seconds before by someone, somewhere. The irregular staccato of these arriving messages form the visual and audible rhythms of the work. The sound-generating systems are constructed almost as wind chimes, where the wind in this case is not meteorological but human, and the particles that move are not air molecules but words. At some level, *Listening Post* is about harnessing the human energy that is carried by all of these words and channeling that energy through the mechanisms of the piece.
>
> *Listening Post* represents the most significant outcome so far of my collaboration with Mark Hansen, the only artist I know whose medium of expression is statistics. Since we began working together, my conceptual vocabulary has grown to include notions like clustering, smoothing, outliers, high-dimensional spaces, probability distributions, and other terms that are a routine part of Mark's day-to-day work. Having glimpsed the world through Mark's eyes, I now hear sounds I would never have thought to listen for.

Visit the *Listening Post* web site (www.earstudio.com/projects/ListeningPost.html) for exhibition dates and further information.

—Jill Franklin

Monster Cluster

The Linux NetworX Evolocity super cluster built for Lawrence Livermore National Laboratory is the number five supercomputer in the world, according to the Top500 Supercomputing List, and it's tops among Linux-based supercomputers.

Here are some facts about it:

- 2,304 processors.
- Can process 5.694 trillion calculations per second (teraFLOPs) running the Linpak benchmark and up to 11 trillion calculations per second using other measures.
- Is the size of a full tennis court.
- Cooling requires 109 tons of air conditioning, enough to cool 22 homes.
- Uses nearly nine miles of cable.
- Weighs 35 tons.
- Is 8.6× more powerful than Deep Blue, the IBM computer that beat Garry Kasparov in 1997.
- Has the same amount of processing power as 11,200 PCs.
- Can do in one day what would take an average PC 25 years.
- Could assemble the human genome in 18 days, compared to the 150 days it took Celera.
- Has 5× the memory required to hold the entire Library of Congress.
- Is 5.6× more powerful than the computer used by Pixar to create the movie *Monsters Inc.*

(Source: Linux NetworX)

—Doc Searls

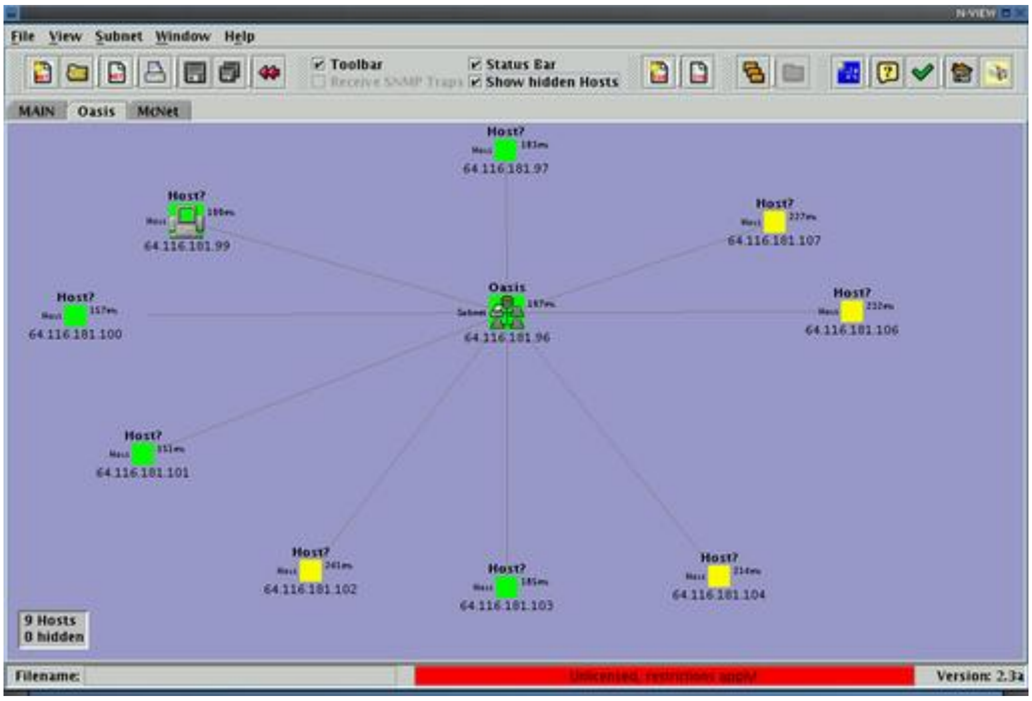
Note: www.0x49.org

I had a tough time choosing between a couple of extremely good utilities I reviewed three years ago. These included Downloader for X, which I use regularly, gnotepad+ and xglobe. But I had to go with Note. There's nothing incredibly special about Note other than it does exactly what it says—keeps notes for you. It works by keeping your notes in a binary, DBM or about any Perl-supported SQL back end. Notes can be edited, deleted or moved into subfolders (topics). Your notes database also can be either in plain text or encrypted form. This makes it ideal for storing passwords or other sensitive information. Requires: Perl, Perl modules: IO::Seekable, DBI::mysql (or other DBI-specfic module, optional), DB_File (optional), MD5 (optional), Crypt::IDEA (optional), Crypt::DES (optional), Crypt::CBC (optional).

—David A. Bandel

N-View: www.n-view.de/index_en.html

For those of you who've used tkined, the network monitor N-View will not be a stranger. However, N-View is quite a bit faster and easier to use. You can have multiple tabs with different subnets. Its biggest drawback, requiring JRE, can be overcome by simply using the package with JRE included. N-View will also mail you if a particular system is suddenly unreachable, though that's not much good if that system is your e-mail server. Requires: Java.



—David A. Bandel

They Said It

I've visited a whole lot of government organizations. Virtually every government agency I've visited has Linux somewhere in the enterprise....The question is: Does anyone know about it?...I suspect at least half of those who say they don't use it have it in their enterprise but don't know about it.

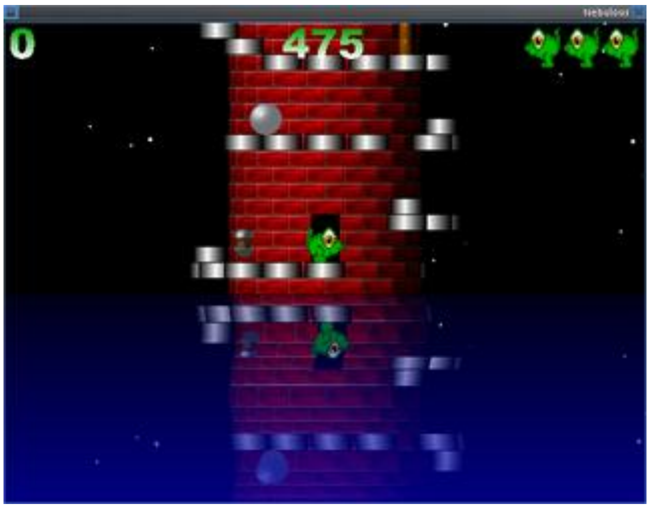
—Robert Hibbard of Red Hat

We haven't developed the vocabulary to credit the open-source dynamic for what it is rather than a puzzling aberration of hackerdom. Once we have the vocabulary—a way of measuring quality vs. cost—we'll elevate open source to the pinnacle it deserves: the most productive process in an economy obsessed with productivity.

—Britt Blaser

Tower Toppler: toppler.sourceforge.net

Here's a great little game, especially for your preteens. The object: get to the top of the tower. The problem: all kinds of things are trying to knock you down. Some of the things you can destroy, and some you simply have to avoid. The game has excellent graphics, and play is extremely smooth. A number of levels and towers are available, but at the rate I get knocked off, I'll be on the beginner tower for a while. Requires: libSDL, libpthread, libz, libstdc++, libm, libX11, libXext, libdl, glibc.



—David A. Bandel

email: zbrown@tumblerings.org

On-line Communities, Hold the Spam

Don Marti

Issue #107, March 2003

You can start a blog without getting buried in junk mail.

People who are new to on-line communities are naturally concerned about all the spam they might get. What? Post your address publicly, for all the spammers to put on their CDs? It's true, you do start to get a lot of spam if you're active on publicly archived lists or run a web diary or blog. But don't be too afraid of spam. A world without spam would be a world without a lot of other things, too.

You could stop spam with one global uniform internet law, strictly enforced worldwide. But before any such law made any impact on spam, it would shut down a lot of stuff you like that your least favorite country convinced the global spam conference to ban.

You could change your address constantly and throw away anything sent to an old address. But there goes your chance to get mail from new people.

You could do what some companies and members of the US Congress are doing and turn the process of mailing you into a little dance with an automatic challenge that's impossible for spamware. But apply the Golden Rule for a minute and imagine if everyone you wanted to write to did that to you.

A total end to spam would be worse than spam. If you want the benefits of mail, you're going to have to put up with a little spam. But how little can you get away with? Gary Robinson is pushing the limits of automatic spam detection with some mathematical research on page 58.

Then, on page 52, Richie Hindle takes a look at how to set up Spambayes to start marking and filtering your spam right now. Spam-filtering developers are coming up with new math and new functionality all the time.

Sometimes I think Google works as well as it does because Doc Searls and his friends thoughtfully link to the good stuff on the Web, thereby pumping up the “Google Juice” of interesting pages. How do you become a commentator, tastemaker or diarist? Pick a blog package and start typing. Naturally Doc and I disagree on which software is best, but he's the man where blogs are concerned. See for yourself on page 66.

On page 72, Marilyn Davis explains that true democracy needs both elections and deliberation. Most systems offer one or the other, but not both. However, Marilyn's eVote system makes it possible for any user of a Mailman-based mailing list to start a wide variety of single-choice or multiple-choice polls. After the initial setup, there's no need for an administrator.

Every once in a while, face-to-face contact is a good idea, too. Fortunately for us Linux users, there are many ways to meet off-line and have been since the beginning. Check out the “Groups of Linux Users Everywhere” (GLUE) link on the *Linux Journal* home page, give your carpal tunnels a break, and meet some Linux people face-to-face too.

Don Marti is editor in chief of *Linux Journal* and usually finds the answer right after he asks a dumb question on a mailing list.

E-mail Everywhere from Anywhere

Heather Mead

Issue #107, March 2003

Whether you want to connect your home system to the Web while you're at work or need to view e-mail attachments on a remote system, the *LJ* web site has an article about it.

I am writing this article on my last day of work before I leave; later tonight, I'll be on a red-eye flight to Pennsylvania to spend Christmas and New Year's with my family. Besides the chance to wake up to a snow-covered lawn, one of the things I am most looking forward to is not having e-mail access for two whole weeks. Oddly enough, my friends find this weird—both my lack of access and my excitement over it. Even my friend who barely knew what e-mail was a mere six months ago expressed shock about my willingly going without it.

For those of you who can sooner imagine parting with a limb than parting with e-mail, the *Linux Journal* web site has some articles that should prove both interesting and useful. First off is Nick Moffitt's advice for “Busting Spam with Bogofilter, Procmail and Mutt” (www.linuxjournal.com/article/6439). Nick presents the macros he added to his system-wide configuration file to make training a Bayesian spam-filtering system as easy as deleting spam and saving or replying to interesting mail—things you do anyway. His macros are also good for personal Mutt configurations, so you don't need root access to start catching spam now.

If you've ever had a need or desire to access your home system remotely but can't SSH in because of a firewall, “Using E-mail as a System Console” can help solve your dilemma. This three-part series (www.linuxjournal.com/article/6453, with links to Parts II and III) is the work of Michael Schwarz, Jeremy Anderson, Peter Curtis and Steven Murphy from their book *Multitool Linux*. They explain how you can use Fetchmail, Procmail and a few scripts to access information, execute commands and even connect to the Internet strictly from e-mail.

Writing an article rejection letter led to “Mutt Over SSH, but What about Attachments?” (www.linuxjournal.com/article/6511) by *Linux Journal* Editor in Chief Don Marti. Usually our rejection letters are not this extensive, but in explaining why the article did not go far enough, Don ended up offering some methods for viewing e-mail attachments remotely. Reader-posted comments have continued the conversation, debating the merits of IMAP vs. its slow speed. If you've got a better way of dealing with viewing e-mail attachments remotely, please write about it in the article comments section. Don really wants to know a better way.

If you want to share the details of a unique e-mail configuration you've designed that can, say, command your house to clean itself or enable you to launch bottlerockets from work, drop us a line at info@linuxjournal.com. Remember to check the *Linux Journal* web site often; new articles are posted daily.

Epilogue: So the holidays are over, and it's my first day back at work. Over dinner last night, I told a friend that I'd be happy if less than 500 e-mails were waiting for me. The final count? 846.

Heather Mead is senior editor of *Linux Journal*.

Best of Technical Support

Various

Issue #107, March 2003

Our experts answer your technical questions.

SSH Won't Let Me In

I've installed Red Hat 7.3 and Red Hat 8.0, and when I use SSH I get an error that says “Connection Refused”. I can SSH on either machine to itself, but not from another machine. I've shut off the iptables service, and I've made sure that in /etc/xinetd.d/telnet the line **disable=no** is present. I've done a **netstat -t | grep telnet** and **netstat -t | grep ssh**, and they tell me that these services are running.

—Robert Haack, haack@nclack.k12.or.us

Check to be sure you can ping from one box to the other. If so, use a tool such as Nmap to verify that the port appears to be open end-to-end from one system to the other.

—Chad Robinson, crobinson@rfgonline.com

Look at /var/log/messages or /var/log/auth.log, which should show you why SSH is dropping the connection. Odds are your machine checks the reverse DNS mapping for your IP addresses and fails. One way to fix that is to populate /etc/hosts with the IP and hostnames of your machines.

—Marc Merlin, marc_bts@google.com

Check the /etc/hosts.allow and add **sshd: ALL** (or the IP address of the remote machine), because this is probably the reason you can connect locally but not from another machine.

—Mario Bittencourt, mneto@argo.com.br

SSH does everything Telnet does and more, and uses encryption, so you should leave the obsolete Telnet service off. Especially now that wireless networks are everywhere, you can't afford to reveal your password on the Net. You're right to use netstat to check for a listening SSH dæmon; however, you need to add the -a option. Do this:

```
$ netstat -at | grep ssh
```

And look for a line that looks like:

```
tcp  0 0 *:ssh  *:*   LISTEN
```

to see that sshd is listening for incoming connections.

—Don Marti, info@linuxjournal.com

If the SSH dæmon isn't running, start it with **service sshd start** or set it to come up automatically with:

```
chkconfig --level 2345 sshd on
```

—Felipe E. Barousse Boué, fbarousse@piensa.com

Modem Lights Won't Let Me Log Off

I can use the Modem Lights applet in Red Hat 8.0 to establish a PPP connection, but I can't use it for disconnecting. When I press the button a second time in order to disconnect, I'm once again confronted with the question “Do you want to connect?” If I answer No, nothing happens. If I answer Yes, I get disconnected and then connected again. How can I configure Modem Lights to do what it's actually supposed to do?

—Martin A. Boegelund, goblin@linuxmail.org

The default setup is the problem here. In the Preferences window for the Modem Lights applet, you will find an Advanced tab. Click it, then set the modem lock file appropriately. Try setting it to **/var/lock/LCK..modem**.

—Ben Ford, ben@kalifornia.com

Dual-Boot Hangs on Install

I just bought a Compaq Presario 1516US, and it came with Microsoft Windows XP. I partitioned the hard drive using Partition Magic, and when I put in the Red Hat 8.0 install CD it goes through the initial checking screen. When it gets to the following it hangs:

```
Partition Check:
 hda:
```

The cursor only blinks and nothing happens.

—Avran, idontlikemail@earthlink.net

This page has a useful tutorial on how to set up the Linux GRUB boot loader to handle a dual-boot system on machines that have Windows installed before Linux: www.geocities.com/epark/linux/grub-w2k-HOWTO.html.

—Felipe E. Barousse Boué, fbarousse@piensa.com

No repartitioning tool is 100% foolproof. All of them warn you to do a good backup first. Dual-boot is an inefficient way to work, because the application you need always seems to be on the other OS. But if you are going to dual-boot, make sure you have a good backup of your original OS and can restore it.

—Don Marti, info@linuxjournal.com

SMP System Won't Power Off

I am running Red Hat 8 with SMP. I have noticed that when I am running the kernel for a single processor the system powers off normally. If I select the SMP kernel during boot and then shut down, however, the system will shut down all processes and then produce the prompt to shut power off. My question is, why doesn't the system power off automatically when using SMP?

—Ron Oliva, rmoliva@citlink.net

Power off uses an APM call on Linux, but APM is unsafe in SMP mode, so Linux disables it. There is one command you give to the kernel to enable just enough APM to allow for power off. With newer kernels, add this to your append= line in lilo or grub:

```
apm=power-off
```

—Marc Merlin, marc_bts@google.com

PowerPC Motherboard?

My brother and I would like to build a PowerPC Linux computer. Do you have ideas on where we can get a motherboard to build a PPC machine? As far as PPC Linux distros, we have found Yellow Dog, Mandrake, SuSE, Rock, Holon, Debian, Vine and Gentoo. So there are a few Linux distros out there for the PPC.

—Rick Killingsworth, iamrlk@yahoo.com

Linux Journal just got a chance to play with a PowerPC ATX motherboard www.terrasoftsolutions.com/products/boxer. Check our next issue for our first look.

—Don Marti, info@linuxjournal.com

These links may be of interest to you and your brother; they contain info about Linux on PPC-based machines and compatible hardware: lppcfom.sourceforge.net and linuxppc64.org. This link is related to IBM 64-bit PPC hardware www.openppc.org, and this one talks about the open PPC architecture and includes some board diagrams/plans.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Management without Monitor and Keyboard

I have been looking for x86-based servers (rackmount) with “lights-out management” abilities. I currently have several Sun V100 systems with this capability, but I'm forced to use Solaris. I've been able to get headless systems set up effectively using port redirection to a serial interface, but that doesn't fix the problem of an OS crash or a server that is hung and needs a power reset or to refresh the OS remotely.

—Ron Culler, ron@firelan.net

You need a hardware management card. Many of the major Intel architecture server vendors, including Dell, HP and IBM offer these cards as options on their servers. Essentially, these cards are similar to console redirection to the serial port, except they also allow you to do other things, such as remotely rebooting or completely turning off the server. Some of these cards even have network ports to do away with serial-based communications.

—Chad Robinson, crobinson@rfgonline.com

Some motherboards like the Intel 440GX have a second serial port that can be used for out-of-band management as you mention (hardware monitoring, reset and power cycle), and of course, you can also get BIOS redirection on the other port. To control the EMP (emergency management port), you can use VACM on Linux, vacm.sf.net.

—Marc Merlin, marc_bts@google.com

Middle Digital, Inc. makes ISA and PCI cards that give headless management functionality to most PC-architecture systems. Telnet to demo.realweasel.com for a live demo, or search the *Linux Journal* web site for “weasel”.

—Don Marti info@linuxjournal.com

Which GUI Toolkit?

I am making a program for running lab experiments and examining data, using arbitrarily long command strings with RPN-style syntax. Command recall and editing features, such as those provided by readline, are essential, and the program has to run on a graphics screen. I am leaning toward using svgalib because that will ease the transition from my DOS version and give greater efficiency in graph drawing, but I will consider other implementations. Can you give me some general ideas about how to get my command-line function?

—Bill McConnaughey, mcconnau@biochem.wustl.edu

Depending on which look and language you're most comfortable with, you're probably going to be best off with a modern GUI toolkit such as GTK+ or Qt. There is more code you're going to be able to borrow there. Also, it's going to be easier in the long run, as displays get bigger, if you ever want to be able to run your program and another program on the same screen at the same time. For example, Ricardo Fernández Pascual has written interesting-looking autocompletion functionality for the GtkEntry widget. His project is called EggEntry and looks general enough to support entering complex commands. See www.geocrawler.com/mail/msg.php3?msg_id=9808742&list=521.

—Don Marti info@linuxjournal.com

New Products

Heather Mead

Issue #107, March 2003

Powerwall 3, CrossOver Office Server Edition, PureMessage 3.0 and more.

Powerwall 3

The AdminForce Powerwall 3 Network Security System (3NSS) performs security monitoring and intrusion detection for small- to mid-sized businesses. Powerwall 3NSS provides a remote installation element, remote monitoring, weekly activity reports and alarm and early warning features. Proxy services are built in to the Powerwall, including DNS, Web, e-mail, CUSeeMe, RealAudio and RealVideo. VPN service authorization allows or denies access to any TCP- or UDP-based application. Powerwall 3 comes in a 19" rackmountable case and features 128MB of RAM, up to four Ethernet interfaces, Token Ring and FDDI support, a command-line interface and address masquerading.

Contact AdminForce Remote LLC, 240 Copley Road, Upper Darby, Pennsylvania 19082, 610-734-1900, sales@adminforce.net, www.adminforce.net.

CrossOver Office Server Edition

CodeWeavers released CrossOver Office Server Edition, software that allows enterprise users to operate MS Windows software in a distributed thin-client environment for both Linux and Solaris, without the presence of a Microsoft OS and the accompanying licenses. CrossOver Office supports core office-automation packages, such as MS Office, Outlook, Internet Explorer and various other business applications. Server Edition offers licensing based on concurrent server usage, and unlimited enterprise site licenses also are available.

Contact CodeWeavers, Inc., 2550 University Avenue West, Suite 439S, St. Paul, Minnesota 55114, sales@codeweavers.com, www.codeweavers.com.

PureMessage 3.0

PureMessage (formerly PerlMx) announced version 3.0 of its antispam, antivirus and policy-compliance product for stopping unsolicited e-mail and protecting end users' mailboxes. PureMessage is built on open-source technologies, such as SpamAssassin and WebAdmin, is extensible in Perl and supports most UNIX platforms. Version 3.0 includes a web-based administration GUI for policy management, improved spam identification and management flexibility, optional end-user quarantine management and the McAfee antivirus engine. PureMessage also uses periodically updated antispam heuristics for evolving spam methodology.

Contact ActiveState Corporation, #400 - 580 Granville Street, Vancouver, BC V6C 1W6, Canada, 604-484-6400, support@activestate.com, www.activestate.com.

Xilinx Virtex-II Pro ML300 Evaluation Platform

The Virtex-II Pro ML300 Evaluation Platform from Xilinx allows designers to experiment with features of the Virtex-II Pro FPGA, which contains an embedded PowerPC processor [see *Linux Journal*, August 2002]. It is a development platform for designs using the PowerPC, RocketIO transceivers and other Virtex-II Pro FPGA features. The product includes GNU tools and reference designs, the ChipScope Pro 5.1i hardware debug tools, over 40 parameterizable IP cores, an evaluation version of the Xilinx ISE 5.1i FPGA implementation tools, plus cables. The ML300 is supported under MontaVista Linux Professional Edition.

Board functions include: four ports of Gigabit Ethernet, two serial ATA connectors, two HSSDC2 connectors, SystemACE CF Interface with 1GB IBM MicroDrive, 128MB DDR SDRAM, 6.4" VGA TFT LCD with integrated touchscreen, IEEE-1394 support, 32/33 PCI Mezzanine Card slot, two CardBus slots, AC97 Audio, IIC EEPROM, temperature sensors, digital potentiometers; SPI EEPROM; PS2, serial and parallel ports and a 9.6-square-inch prototyping area with 90 free FPGA I/Os.

Contact Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124, 408-559-7778, www.xilinx.com/ml300.

IBM eServer p630

IBM announced the eServer p630, the company's first pSeries system to run Linux natively. Designed to lower costs of users deploying Linux on 64-bit processor technology, the p630 is equipped with POWER4 microprocessors. The p630 also supports AIX5L, IBM's UNIX OS and a combination of Linux and AIX5L in logical partitions. The p630 Express Configuration is available with one, two or four POWER4 processors and up to 8GB of memory in a rack or tower configuration. The p630 comes with self-diagnosing and self-healing server features, including optional hot-plug power supplies and cooling fans, dynamic deallocation of processors and PCI bus slots and first failure data capture (FFDC).

Contact IBM Corporation, 1133 Westchester Avenue, White Plains, New York 10604, 888-746-7426, www-1.ibm.com/linux.

Fonix DECtalk RT

DECtalk text-to-speech technology is a formant-based synthesizer that transforms ordinary text into natural-sounding, intelligible speech. DECtalk technology offers personalized voices, and its extensive user controls ensure reliable performance in real-world applications. Fonix DECtalk RT for Linux provides multilanguage text-to-speech synthesis; nine defined voice personalities; the ability to say letters, words or phrases; pause, continue and stop speaking controls; the ability to insert voice control commands into any text file for use by DECtalk software applications; and much more. Available as a download, DECtalk works as a runtime component for any application written to support it.

Contact Fonix, 180 W Election Road, Draper, Utah 84020, 801-553-6600, www.fonix.com.

Archive Index Issue Table of Contents

Advanced search