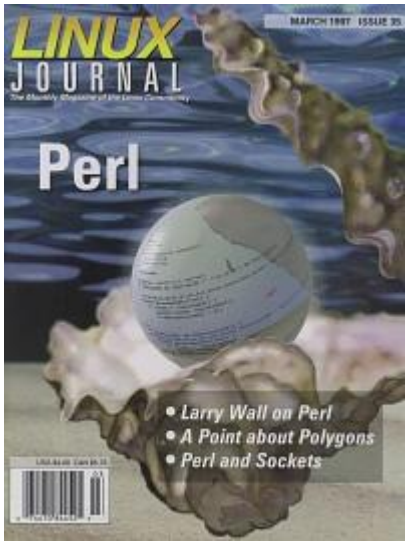
Linux Journal Issue #35/March 1997



Features

Creating and Using a Database with Perl *by Randy Scott*
There are a variety of different database formats in a UNIX environment available to the Perl programmer.

Perl and Sockets *by Mike Mull*
Learn about Perl's ability to use sockets, a mechanism of interprocess communication and the basis for most Internet clients and servers.

Wherefore Art, Thou? *by Larry Wall*
The discourse on the art of Perl programming, originally printed in The Perl Journal .

News & Articles

AMD—AutoMount Daemon *by Matthew Crosby*
Using the 12C Bus with Linux *by Simon G. Vogl*
The Death of Xenix *by Evan Leibovitch*
The Guide to Virtual Services *by Chad Robinson*
NEdit *by Dan Wilder*
Setting Up UUCP *by James L Hill*

Reviews

Product Review Metro-X and Accelerated-X *by Jonathan Gross*
Book Review Programming Perl *by Phil Hughes*
Book Review Perl 5 by Example *by Sid Wentworth*

Creating and Using a Database with Perl

Randy Scott

Issue #35, March 1997

Perl 5 includes packages enabling your Perl scripts to create and modify databases in standard Unix formats. One of these databases can be a more efficient alternative than a flat text file (which Perl handles marvelously), and it will be compatible with other languages, like C.

Perl programmers, like programmers of any other language, typically need to store large amounts of data. For this data to be manageable, it needs to be stored in a conveniently accessible format. It never hurts to make the stored data easy to write, as well.

Even though Perl is an exceptional language for text processing, in many circumstances, a more structured database-like format offers quicker access. In addition, it may also be necessary for a Perl script to read or write a database that is also accessed through a C program.

To accomplish this, the Perl distribution includes packages that give a Perl programmer access to a variety of different database formats available in a Unix environment. These formats include: the Berkeley DB format, the Free Software Foundation's GDBM format and the NDBM format.

Associative Arrays

The associative array (or “hash”) is one of the most powerful data structures available to a Perl programmer. To those familiar with traditional arrays (in C, Pascal or Perl), an associative array can be thought of as an array indexed using an arbitrary string instead of an integer subscript. Basically, an associative array is a data structure that allows the programmer to associate one string—a key—with another—its value.

Here is an example of an associative array that can be used to convert the abbreviated name of a day of the week to its full name.

```
%days = (
        "Sun", "Sunday",
        "Mon", "Monday",
        "Tue", "Tuesday",
        "Wed", "Wednesday",
        "Thu", "Thursday",
        "Fri", "Friday",
        "Sat", "Saturday"
);
```

The **%** in front of the variable name **days** is used to tell Perl the variable is an associative array. As shown, associative arrays are initialized by using pairs of values that relate to each other.

To access the data stored in an associative array, you can use a syntax similar to the following:

```
$long_name = $days{"Sun"};
```

This expression will set the scalar variable **long_name** to the value associated with the key “Sun”, the string “Sunday” in this example.

You can see already that associative arrays can be a powerful tool for organizing data used inside of a Perl script. This technique can easily be extended to something more useful by creating *values* made up of more than one field. Take, for instance, this simple address book database where multiple fields in the associative array's value are separated by colons:

```
$phone_db = (
        "Bill Jones", "123 West Avenue:New York, NY:12345",
        "Jane Smith", "6789 1st Street:Chicago, IL:56789"
);
```

New entries in this database can be added with an expression like:

```
$phone_db{"Bill Smith"} = join(":", $street, $city, $zip_code);
```

Data can be extracted from this simple database with an expression like:

```
($street, $city, $zip_code) = split(/:/, $phone_db{"Bill Smith"});
```

As you can see, these arrays come in very handy for manipulating data inside a Perl script. However, how can we export this data easily to a file so our scripts or other programs can access the data? One simple method would be to use a text file with the fields of our database separated by colons. This method would make writing out the database from our Perl script very simple. It could be done using a piece of code like the following:

```
while (($name, $record) = each %phone_db) {
        print "$name:$record\n";
}
```

This method does not lend itself well to performing a search through the file, as we would need to read, on average, half the lines in the file in order to find the

record we are seeking. In addition, writing code to search such a file in other languages (C, for instance) may not be as simple as the Perl script we have written.

To solve this problem, Perl supports “binding” associative arrays to the various types of database formats mentioned above. This allows a Perl programmer to create, access and update databases in the popular Unix database formats as easily as performing operations on an associative array.

Database Support in Perl

Perl version 5 includes a set of “packages” that manipulate the various database formats. These packages are:

- DB_File—for Berkeley DB databases
- GDBM_File—for the Free Software Foundation's GDBM databases
- NDBM_File
- ODBM_File
- SDBM_File

To use any of these database packages, a Perl programmer must *include* the package at the beginning of the script using the following statement:

```
use DB_File;
```

In addition, the Fcntl package also needs to be included. This is accomplished by including the following at the beginning of the script:

```
use Fcntl;
```

Man pages are included in the Perl distribution for each of these packages. For simplicity's sake, only the DB_File package and its associated Berkeley DB databases are discussed in this article.

Opening a Database

Databases are opened in Perl using the **tie()** function. This function is responsible for “joining” an associative array with a database package. Operations performed on the associative array are then translated by the database package into function calls that operate on the database file itself.

Here is an example of opening a database named “phone.db” using the DB_File database package:

```
tie (%phone_db, DB_File, "phone.db") ||
        die ("Cannot open phone.db");
```

This command binds the associative array named **phone_db** to the Berkeley DB database file named “phone.db”. In this example, the file must exist and must be readable by the Perl script.

Creating a Database

Creating a database is nearly as simple as opening one. The following command will create a database named “phone.db” in the current directory with the file's permissions set to read-write for the owner and read-only for everyone else. The file will be created only if it does not already exist. If the database file exists in the current directory, the database file will simply be opened for read-write access by the Perl script.

```
tie (%phone_db, DB_File, "phone.db", O_CREAT|O_RDWR, 0644) ||
        die ("Cannot create or open phone.db");
```

The **O_CREAT** and **O_RDWR** flags are the same flags used as parameters to the Unix **open()** system call. They specify that the file should be created if it does not exist and opened with read-write access.

Reading from the Database

Reading from the database works exactly like reading data from an associative array. If the key is known, specific records can be read from the file with an expression like:

```
$record = $phone_db{"Bill Smith"};
```

All the records in the database file can be scanned (in a seemingly random order) with something like:

```
while (($name, $record) = each %phone_db) {
        [ commands to process data here ]
}
```

During each pass through the **while** loop, the **$name** scalar variable will be set to the key value from the database, and the **$record** variable will be set to the data associated with the key.

Writing to the Database

New data can be written into the database by creating a new key in the associative array and setting the key's value. This is done with a command similar to:

```
$phone_db{"Bill Smith"} = $data;
```

where **$data** is the information to be associated with the key “Bill Smith”. Any changes made to the associative array will be written into its corresponding database file.

Deleting Items from the Database

Keys can be removed from the database in exactly the same way items are removed from an associative array in Perl—by using the **delete()** function. The following code removes the record in the database that refers to “Bill Smith”.

```
delete $phone_db{"Bill Smith"};
```

Closing the Database File

Changes to an associative array may not be immediately written out to the database file. To insure that changes are successfully written to the database file, the file must be closed.

Closing the database file involves *un-binding* the associative array from the database package. This is done with the **untie()** function in the following manner:

```
untie(%phone_db);
```

This closes the database file, making updates to the file if necessary. The associative array **%phone_db** can now no longer be used to access the records in the database.

Other Types of Databases

All of the examples provided here use the default type of Berkeley DB database, the **DB_HASH** type. This form of database uses a hash table (like Perl does) to store the keys and their values in the database file. Two other types of databases are provided with the Berkeley DB package: **DB_BTREE** and **DB_RECNO**.

The **DB_BTREE** format uses a sorted, balanced binary tree to store the key and value pairs. This format allows data to be stored and read in a sorted order as opposed to the seemingly random order the **DB_HASH** format produces. The default comparison routine sorts the keys in the database file in lexical order (alphabetically). The DB_File man page discusses this format in more detail and shows how to replace the default comparison routine with one of your own.

The **DB_RECNO** format is designed to operate on flat text files. It is bound (with **tie()**) to normal Perl arrays, not associative arrays. Indexing this array with a

number provides the text found on that line of the database file. This format is also discussed in more detail in the DB_File man page.

The desired format of database file is specified with an additional parameter for the **tie**() function.

```
tie (%phone_db, DB_File, "phone.db", O_RDONLY, 0644, $DB_BTREE) ||
                die ("Cannot open phone.db");
```

This command will open the **DB_BTREE** database named “phone.db” in read-only access mode. If the file does not exist, the command fails.

Other Fun Stuff with Associative Arrays

Sometimes it is necessary to sort an associative array within a Perl script. Sorting by the key values of an associative array is done like this:

```
for (sort keys %phone_db) {
        print "$_ = $phone_db{$_}\n";
}
```

Each iteration of this loop will set the **$_** scalar to a key value from the associative array provided in alphabetical order. This method works very nicely for sorting associative arrays by their keys. Sorting by an associative array's values is slightly more difficult:

```
sub sort_by_value {
        ( $phone_db{$a} cmp $phone_db{$b} ) || \
( $a cmp $b );
}
for (sort sort_by_value keys %phone_db) {
        print "$_ = $phone_db{$_}\n";
}
```

This piece of code substitutes the default routine that **sort()** uses to order the elements it is given with a special routine. This routine, **sort_by_value**, sorts the associative array first by the values, and secondly by the keys (i.e., when the two values are identical, compare their respective keys to determine which should appear first).

Keep in mind that these two methods for sorting an associative array do not actually rearrange the array in any fashion. They simply provide a way to pull every key and value pair from an associative array in a particular sorted order.

Putting It All Together

An example of how databases in Perl can be used is provided in Listing 1, a short script designed to keep a database of hits on a World Wide Web site. The script reads the NCSA HTTPD *access* log file, stores the information in the database and creates an HTML page that displays all the statistics for the site.

Listing 1. Example Web Site Hits Database Script

This implementation is not complete—it keeps track only of which documents were accessed and their sizes. A more complete implementation could also store information about the hosts that accessed the web server, for instance. Some method for "expiring" entries in the database after a particular time interval would be a handy feature as well.

The script begins by reading the existing database file and placing all the data into associative arrays indexed by the document file name. Next, the script reads the access log file from standard input and places the data into the associative arrays that store the statistics. Finally, the script creates an HTML page using tables to display the statistics.

Conclusion

The topics provided in this article are by no means a definitive reference guide for using the built-in database support included with Perl, but they can be used as a starting point for further experimentation and exploration.

Randy Scott is a senior Computer Engineering student at the Milwaukee School of Engineering. He been programming with Unix and C for nearly three years and has become an avid Perl fan in the last six months. Any questions or comments regarding this article can be sent to scottr@bork.com.

Archive Index Issue Table of Contents

Advanced search

Perl and Sockets

Mike Mull

Issue #35, March 1997

One of the many features of Perl is its ability to use sockets, which are a mechanism of interprocess communication and the basis for most Internet clients and servers. This article introduces using sockets and Perl with a simple client program.

Perl works well for writing prototypes or full-fledged applications because it is so complete. The language seldom needs to be extended to do the sorts of things you'd expect to do with C or C++ on a Linux system. One notable example is the Berkeley socket functions, which Perl included even back when the Internet was just a cool bit of technology rather than a global cultural phenomenon.

Sockets are a general-purpose inter-process communication (IPC) mechanism. When processes run on separate machines, they can employ sockets to communicate using Internet protocols. This is the basis for most Internet clients and servers. Many Internet protocols are based on exchanging simple text; so is much of the interesting content. Since Perl excels at processing text, it's ideal for writing applications like web servers or any type of client which parses or searches for text. In this article, we develop a very simple client that searches for regular expressions on specified web sites—a not-so-intelligent agent, you might say.

I assume the reader has no prior knowledge of sockets, but if you have used the socket functions in C, they'll look quite familiar in Perl. The basic functions include **socket, connect, bind, listen,** and **accept**. Perl also has versions of functions like **gethostbyname** and **getprotobyname**, which make socket communication much easier. These Perl functions, of course, eventually invoke the C versions, so the argument lists are quite similar. The only differences arise because Perl file handles aren't the same as C file descriptors (which are just integers) and the Perl versions don't need the additional lengthy arguments for strings or structures.

We'll discuss the details of the socket functions needed for an Internet client later, but let's first look briefly at the normal sequence of operations for Internet communication. The server first establishes a socket with the **socket** function, which returns a socket descriptor much like a file descriptor. The server next assigns the socket an address with **bind**, then tells the system that it is willing to receive connections with the **listen** function. The **accept** function can block until a client connects to the server. The client program also calls **socket** and gets a socket descriptor. The client connects to the address specified by the server's **bind** call using the **connect** function. If all goes well, the client can read and write to the socket descriptor just as if it were a file descriptor. Refer to Listing 2 to see how the **socket** and **connect** functions are used in a typical program.

As mentioned above, a client program must first call **socket** to get a socket descriptor or, in the case of Perl, a file handle. This function specifies a particular communications protocol to use and sets up an endpoint for communication—that is, a place to plug in a connection—a “socket”, for lack of a better term. The syntax of this function is:

```
socket SOCKET, DOMAIN, TYPE, PROTOCOL
```

SOCKET is the file handle. **DOMAIN** and **TYPE** are integers that specify the address domain (or family) and the socket type. In Perl 4, you had to set these numbers explicitly, but Perl 5 defines them in the Socket module. To access the Socket module, add the following line to the top of your program:

```
use Socket;
```

For Internet applications, set **DOMAIN** to **AF_INET** (usually 2) and **TYPE** to **SOCK_STREAM** (usually 1). This basically means the address of the server will have the familiar Internet form (e.g., 192.42.55.55) and you'll read from and write to the socket like any I/O stream. You can set the **PROTOCOL** argument to 0 for most applications, but it's easy to get the correct value with the **getprotobyname** function.

Next, you need to connect to the server with the **connect** function. This can get a bit tricky in Perl if you don't have the most recent versions of the Socket module, primarily because it's hard to specify the server's address. The syntax of the **connect** function is:

```
connect SOCKET, NAME
```

SOCKET is the file handle created by the **socket** function, so that's easy. The **NAME** argument, however, is described as a “packed network address of the proper type for the socket”, which might leave you scratching your head if you're not already familiar with sockets. For Internet applications, the proper

type of network address for the C version of the **connect** function is given by structures something like those in Listing 1 (from either **<netinet/in.h>** or **<linux/in.h>**).

Listing 1

With a bit of scrutiny, you can see you need to pack three pieces of information into a binary structure 16 bytes long. First you need the address family, which is **AF_INET**, the same as the **DOMAIN** argument to **socket**. The second piece is the port number of the server socket. Most common servers have what's called a "well-known" port number (in the case of HTTP servers, this is 80), but an application should have some method of indicating alternate port numbers. Finally, you need to know the Internet address of the server. From the structures above, you can tell this is a 32-bit value. Fortunately, if you know the Internet name of the server (e.g., www.linux.com) you can get the address with the **gethostbyname** function. Once you've assembled this information, you can create the **NAME** argument with the Perl **pack** function. The code might look something like this:

```
$sockaddr_in = 'S n a4 x8';
$in_addr = (gethostbyname("www.linux.com"))[4];
$server_addr = pack( $sockaddr_in, AF_INET, 80, $in_addr );
```

Recent versions of Perl (5.002 and later) greatly simplify this whole process with the **sockaddr_in** function from the Socket module. This function takes the port number and the Internet address of the server and returns the appropriate packed structure. I use this technique in our mini-client in Listing 2. If you need portability, or simply want readability, I strongly recommend using Perl version 5.002 or later.

Listing 2

So we've finally set up our socket and made a connection to the server. Now things get considerably easier because we can treat the socket like any other file handle. The only wrinkle is we want to make sure anything we write to the socket is not buffered, because it needs to get to the server before we can read the server's response. For this we use the Perl **select** function, which sets the file handle to use for standard output. Note in Listing 2 that the socket file handle is selected; then the special variable **$|** is set to 1 to force a buffer flush after every write; then **STDOUT** is re-selected.

Now our client can send a request to the server. This application just sends a GET command to the HTTP server so that it will return the page specified by the URL. Once the command is sent, we read anything arriving at the socket line-by-line and look for the patterns we've specified. You could do anything you

wanted with the HTML returned from the server, even parsing it or looking for other hypertext links to follow.

It will come as no shock that there are many aspects of sockets we haven't covered. In particular, I haven't discussed writing servers (mainly to keep this article to a manageable length). If you want to learn more about writing Internet servers in Perl, I recommend reading *Programming Perl* by Wall, Christiansen, and Schwarz (commonly called “the Camel book”). Perl also contains several socket functions I haven't mentioned, including **send** and **recv**, that can be used like write and read calls, and **sendto** and **recvfrom**, which are used for so-called “connectionless” communications. Again, see the Camel book for details on these functions, and for network communication in general, I recommend *Unix Network Programming* by W. Richard Stevens. Also, don't forget that many Perl Internet applications live out there on the Internet already, so look to these for examples. I particularly recommend tinyhttpd, a very compact HTTP server, as a good way to learn how to construct servers (see http://www.inka.de/~bigred/sw/tinyhttpd.html).

Mike Mull writes software to simulate sub-microscopic objects. Stranger still, people pay him to do this. Mike thinks Linux is nifty. His favorite programming project is his 2-year-old son, Nathan. Mike can be reached at mwm@cts.com.

Archive Index Issue Table of Contents

Advanced search

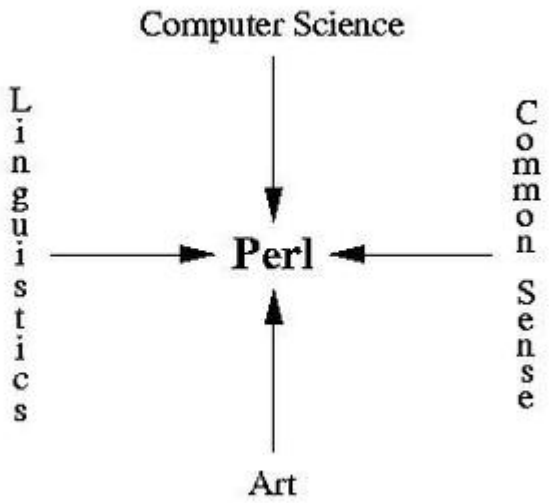
Wherefore Art, Thou?

Larry Wall

Issue #35, March 1997

Larry Wall entertains while expounding upon Art and the Art of Programming. This treatise originally appeared in The Perl Journal, Volume 1, Issue 1. (Check out TPJ at http://tpj.com/tpj/.)

I don't know whether a picture is really worth a thousand words (most pictures seem to be considerably larger these days), but when I give talks about Perl, I often put up a picture showing where some of the ideas in Perl come from.



I usually make a joke about Linguistics not really being the opposite of Common Sense, and then proceed to talk a lot about both of them, with some Computer Science thrown in for good measure. But last December as I was giving a talk in Stockholm, someone asked me how Perl got its inspiration from

Art. I was stumped. I mumbled something semi-irrational (always appropriate when discussing Art) and went on to the rest of my talk.

But the question continued to bother me; or more specifically, it continued to bother my left brain. My right brain continued to be perfectly content with the purported connection. Unfortunately, it's also not at all forthcoming with the verbiage necessary to explain itself. Right brains tend to be like that. So let me see if my left brain can make something of it all.

Art is first of all based on the notion that there exist amoral decisions, that is, choices you can make either way, without feeling like you're being naughty or nice. So let's presume that the Artist has free will of some sort or another, and can therefore behave as your ordinary, everyday Creator.

Now, it's more or less immaterial whether your Artist creates because of a liking for Deluxe Designer Universes or merely because of a liking for caffeine. The simple fact is, we have Artists, and they do Art. We just have to deal with it. We really do. You can make life miserable for the Artist, but the Artist has ways of getting revenge. (Of course, if you don't make an Artist miserable, they'll make them selves miserable, but that comes into a different story.)

We can further subdivide the Artists into those who enjoy getting their revenge by being **more** than properly miserable, and those who prefer to get their revenge by being **less** than properly miserable. Artists of the first sort will prefer to work in a more formal medium, one that inflicts extra pain on the Artist, such as composing sonnets, dancing ballet, or programming C++. Artists of the second sort tend to be much more fun-loving, free-wheeling and undisciplined, whether the verb in question is composing, dancing, programming, or slinging. (Especially slinging. There's nobody quite so joyful as a B.S. artist. I should know...)

There is, of course, a third category of Artist, the one who oscillates between the two extremes.

Perl was written first of all to let the Artist make amoral decisions. That's why the Perl slogan is “There's More Than One Way To Do It!” Perl doesn't really care whether you use cobalt blue or burnt umber in a particular spot in your painting. It's your choice—you're the Artist. You're responsible for the overall effect. Indeed, your boss will hold you responsible for the overall effect, so why should Perl?

But more than that, Perl is intended to be a medium for those who are tired of composing in a formal computer language and want to write some “free verse” without arbitrary restrictions. Sure, from a motivational point of view, arbitrary

restrictions are challenging to work with, but when's the last time you saw a gleeful COBOL programmer?

On the other hand, with Perl 5, we've made strides in making life wonderful for those Artists who oscillate. You can have your cake and eat it, too. When you're in a manic mood, you can pour forth unstructured, unreadable (but expressive) code to your heart's content. Later on, when you are in a dour mood, you can put a *-w* and a *use strict* at the top of your script and greatly increase your level of discipline (read “pain”). Next you can prototype your function definitions. While still in your somber state, you can go back and put white space in all your regular expressions and comment every last little bit as penance for your past indiscretions. You can restructure all your code into modules and unit test it in a jiffy because the Perl interpreter is so handy to invoke. Then as you swing back into a more carefree frame of mind, you can cheat by tweaking all those carefully encapsulated variables in all those painstakingly restructured modules. Ain't it the life.

Now, Linguistics may not be the opposite of Common Sense, but it's certainly the case that over the last twenty years or so, many Computer Scientists have come out in opposition to the Art of Programming. In trying to make programming predictable, they've mostly succeeded in making it boring. And in so doing, they've lost sight of the idea that programming is a human pursuit. They've designed languages intended more to keep the computer happy than the programmer. Was any SQL programmer ever happy about having to declare a value to be varchar(255)? Oops, now it's a key, and can't be longer than 60. Who comes up with these numbers?

They've also lost sight of the idea known to any Artist, that form and meaning are deeply interdependent. One of the ideas I keep stressing in the design of Perl is that things that **are** different should **look** different. The reason many people hate programming in Lisp is because every thing looks the same. I've said it before, and I'll say it again: Lisp has all the visual appeal of oatmeal with fingernail clippings mixed in. (Other than that, it's quite a nice language.)

A large part of the design of Perl is driven by the dictates of visual psychology. That's why Perl lets you structure your code with the condition on the left or on the right, depending on which part you want to look important. That's why the large nested structures like while loops require an explicit beginning and end, while the small ones like list operators don't. That's why scalars start with $, arrays with @, and hashes with %. That's why filetest operators look like “-M”, while numeric tests look like “==”, and string tests look like “eq”. Perl is very much a What-You-See-Is-What-It-Does language. You can talk about readability all you like, but readability depends first and foremost on recognizability.

Music To My Ears

Like many computer geeks, much of my artistic training has been in music. Of all the arts, it most clearly makes a programmer/interpreter distinction, so perhaps it's natural for a musician to think about how interpreters work. But the interpreters for a computer language are located both in the computer and in the human brain. I don't always know what makes a computer sad (or happy), but I do have a pretty good idea what makes a person mad (or sappy). Er, sorry.

Anyway, when I was young, I was taught that music has progressed through four major periods: Baroque, Classical, Romantic and Modern. (The other so-called fine arts have also gone through these periods, though not necessarily at the same rate.) I always thought it rather curious that we called the current period Modern, since definitionally the idea of modernity seems to be a permanently latched-on state, bound to the cursor of time, so to speak. But that was because the word “modern” still meant something back then. This was, after all, the 1960s. Who could have guessed that Modern period would be followed by the Post-Modern?

If you're willing to concede by now that the design of computer languages is an artistic medium of sorts (and searches), then it's reasonable for us to ask ourselves whether programming languages have been progressing through the same sequence of artistic development. Certainly, people have occasionally claimed that Perl is “baroque”, to which my usual retort is, “Thanks, I like Bach, too.” But this is merest rhetoric (on both sides).

So what do we really mean when we talk about these periods? Let's start at the beginning, which is the Baroque period. Of course, it's not really the beginning. People were producing music long before they ever invented the bucket in which to carry the tune. But before and during the Baroque period, there was tremendous technological progress in both the production and publication of music. Composers and performers could make a name for themselves. Innovators were rewarded, but the forms of expression were heavily influenced both by cultural expectations and by available hardware. People were expected to improvise. What we got was more or less the Cambrian explosion of music.

Similarly, at the dawn of the computer era, there were new opportunities to innovate. The geniuses of that period improvised many forms of assembly language. To them, these languages all looked very different. But nowadays we tend to see all assembly language as the same, just as a lot of Baroque music seems the same to us, because the music tends to follow particular forms and sequences. Baroque music is structured like a weaving on a loom, and it's no accident that punch cards were invented to run looms before they were used to run computers.

It's easy to take a superior attitude toward these innovators, but this is unfair. We owe a great debt to these people. They invented the algorithms we use, even if the music does seem a bit limited at times. (Except for Bach, and Backus, of course.)

The Classical period was a time of standardization. Most of our modern instruments took their current form during this period, and this continued the trend of turning virtuosity into a marketable and portable commodity. Being able to program in FORTRAN was like being able to play the piano forte. It was a skill you could use on someone else's machinery. Mozart could now go on tour.

The Romantic era was a time of seeing how far the classical forms could be stretched. And considerably stretched they were, in Beethoven and Mahler, as well as PL/1 and COBOL. The word “excessive” has been applied to all of them, as it will always be applied to anyone or anything that attempts to sling the entire universe around by any of its handles. But this is difficult at the best of times.

Finally, the typical overreaction took place, and we arrived in the Modern era, in which subtlety and minimalism were mandated, and antiquated cultural expectations were thrown over and thrown out. Reductionism and deconstructionism were the order of the day, from Bartok to Cage, and from Pascal to C. Music wasn't allowed to be tonal, and computer languages weren't allowed to do fancy I/O. All the gadgetry had to be visible and exposed. Everything had to look difficult, so we got stuck in the Turing Tar Pit.

Of course, this is all oversimplified, and every language has aspects of each of these periods in it. And languages specialize in other ways: BASIC is like pop music. Tune into REXX for your easy listening classics. Tcl is fuzzy like jazz—you get to improvise a lot, and you're never quite sure who is interpreting what. Python is like MTV—it rocks, but it gets to be much of a sameness after half an hour or so.

Lisp is like church music down through the ages, adapting to what ever the popular culture is, from organ to electric guitar to synthesizer. That would make Scheme a kind of cult music, sung simply but with great fervor to an acoustic guitar.

C++ is like movie music, of titanic proportions, yet still culturally derivative by and large. Especially large. Sometimes it's hard to sit through the whole movie. And yet, as an intentionally Post-Modern language, it's kinda fun, and gets the job done.

As for Java, using a subset of movie music, it's attempting to be the basis for every good feeling everywhere, the ground of all emotional being. Muzak. It's everywhere you want to be.

Shell programming is a 1950s juke box—great if it has your song already.

And of course, any language touched by ANSI starts to sound distinctly operatic.

So where does Perl fit in to this glorious mess? Like C++, Perl is a Post-Modern language by design, unashamedly reconstructionist and derivative. Perl is neo-Baroque, neo-Classical, neo-Romantic, and even, in spots, neo-Modern.

What musical genre encompasses so much? Where can you find every thing from Wagner to “Shave and a Haircut, Two Bit”? Where can you find multiple levels of abstraction, accessible to newbies and oldsters alike? What kind of music admits everything from harmonica to accordion to pipe organ? What music is object-oriented, in good one-to-one correspondence with the main action? What music is good for programming in the small, but can be expanded to feature length as necessary? What music parodies every thing in the world, yet leaves you feeling good about the world? What music is Perl?

Why, cartoon music, of course. That's all folks!

Larry Wall doesn't mind being blamed for the invention of Perl, to the extent that you can blame any one person for it, which you can't, so there.

Archive Index Issue Table of Contents

Advanced search

AMD—AutoMount Daemon

Matthew Crosby

Issue #35, March 1997

Here's a way to make system administration easier when dealing with NFS.

The standard protocol for sharing files between Linux boxes is the Network File System (NFS). This protocol, which originated with Sun in the mid 80s, does the job, but it has many deficiencies that can cause trouble for a systems administrator. Though there are alternatives, such as the Andrew File System (AFS) that are much nicer, most of us are stuck with NFS at this time—it is standard, available on every platform under the sun and free. Fortunately, the program AMD (AutoMount Daemon) exists to make living with NFS much easier.

Why Use AMD?

AMD is an automounter—i.e., it maintains a cache of mounted file systems. At a minimum, AMD should be used wherever you use a normal NFS mount, since AMD makes your network more reliable. Because of the stateless design of NFS, any process trying to access data on an NFS partition will blocked if the partition's server goes down. AMD improves the situation by keeping track of which machines are down and which are inaccessible. Since AMD doesn't mount every partition immediately or keep them mounted, as does NFS, you save overhead that otherwise would be used for kernel and network traffic from the unused partitions, and thus improve machine performance.

Configuration and administration become much easier with AMD. Instead of requiring a different **fstab** file on each host, you can have a single, centrally maintained AMD map which can be distributed as a file with **rdist** or NIS maps or even Hesiod. As an example, we have over 100 machines with one centrally maintained AMD map. One map file is certainly easier to edit than 100.

Another convenient feature of AMD is dynamic maps that change depending on any number of criteria. A single map can point to multiple places, allowing you

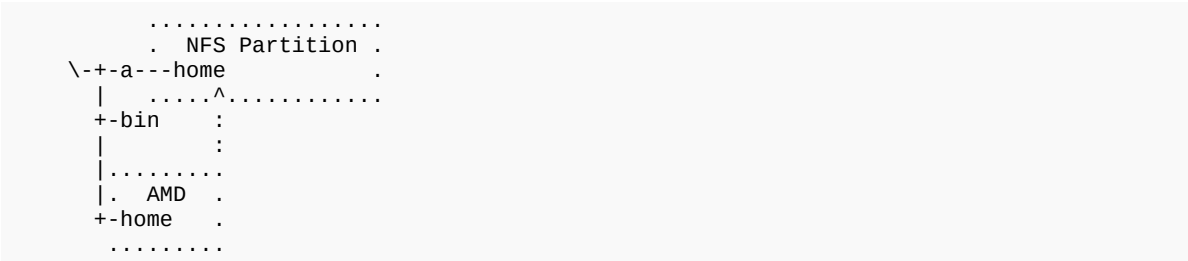
to do operations unavailable with normal NFS. For instance, if you have multiple replicated servers, you can set up a map so that if one server goes down, AMD will automatically mount files using one of the others.

How AMD Works

AMD operates by mimicking an NFS server. When a file is accessed, AMD uses its map to decide where that file actually resides. It then mounts that partition, if necessary, using regular NFS, and mimics a symlink to the actual location. All AMD actions are done transparently, so that from the user's point of view she is simply accessing a regular Unix symlink that points to a regular user's file. AMD maintains its NFS mounts beneath a temporary directory, by default called “a”, a name choice that can cause problems. For example, the actual physical path of the directory **/home/crosby** is **/a/home/crosby**, but **/a/home/crosby** exists only if someone has recently accessed **/home/crosby** (or some other path on the same partition). Therefore, users should never explicitly access files through /a.

Diagram 1 demonstrates the three types of mounts involved: the native partition, the AMD pseudo partition and the behind-the-scenes NFS partition.

Diagram 1. Three Mount Types



NFS Resources

AMD does a few other things behind the scenes to keep operations healthy. First of all, it sends out rpc requests at regular intervals to all the servers it knows to see if they are alive. If one isn't, AMD will not try to mount it. This checking also allows AMD to offer access to replicated file systems; that is, you can set up multiple redundant servers, and if one goes down, AMD will try to mount another one.

Setting Up AMD

To use AMD, you must first of all build one or more AMD maps. These maps are the configuration files that tell AMD exactly what to do. Many tasks can be done from an AMD map, and documenting them all would take more than one article. Listing 1 provides a sample AMD map with some common tasks, and with comments under each entry to explain it. In general, a map consists of two

fields: the name, which is translated to the path name underneath the AMD mount point, and the options, which specify what to do with this path name.

I have merely touched the surface of AMD features in Listing 1. The uses of AMD are almost endless—as the man page says, "A weird imagination is most useful to gain full advantage of all the features." The documentation that comes with the package gives complete instructions for writing a map.

Running AMD

To run AMD, you simply type **amd** at the prompt, providing the mount point(s) amd the map(s) on the command line. For example, if the map in listing 1 is named **/etc/map.main**, and a map named **/etc/map.silly** also exists, to execute AMD you would type:

```
amd /u /etc/map.main /silly /etc/map.silly
```

It is a good idea to include this statement in your rc files.

A number of options are available for the **amd** command. Two useful options are the ability to specify the NFS mount points and the timeout period. The program amq helps control AMD. For example, amq can force AMD to unmount a file system and to flush the cache (useful when debugging NFS problems). The man page for amq provides a complete description of all its capabilities.

NFS Considerations

Because AMD is just a front end to regular NFS, you have to worry about the same issues that you would when running NFS alone—you must ensure that exports and their options are correct. Explaining NFS is beyond the scope of this article; however, if you are unfamiliar with the basics of NFS, see the NFS Resources box on page FIXME.

How to Get AMD

Binaries and patches to port AMD to Linux may be obtained from sunsite and other sources (see sidebar). AMD has stayed relatively stable and bug free in the last few years; development is no longer active. AMD comes with excellent documentation.

AMDResources

NFS Resources

Matthew Crosby is a system administrator and student at the University of Colorado, Boulder. He has administered practically every system in existence, and has used Linux since the early .9* days. He can be reached via e-mail at crosby@nordsieck.cs.colorado.edu.

Archive Index Issue Table of Contents

Advanced search

Using the I2C Bus with Linux

Simon G

Issue #35, March 1997

Originally designed for controlling consumer electronics, the I2C bus is easily adapted to working with Linux to control a variety of devices using the I2C bus standard.

The I2C bus is a two-wire serial bus for connecting a wide range of ICs to a computer or micro-controller. It was originally developed by Phillips in the early '80s, but is now supported by a wide range of companies.

I first stumbled over this bus when I still had my old Commodore C64, while looking over some hardware books. I never built an adapter, so I forgot about it, until I started to hack around in the Linux kernel. Then I thought building an adapter and writing a corresponding kernel module to access the bus a character device would be a promising goal.

Description of the Bus

The I2C bus consists of two lines, one for the data and one for the clock. The chips connect to the bus via open collector input/output stages, which cause the lines to be high when idle.

Transmission is initiated via a start condition, after which an address byte is transmitted. The byte consists of a 7-bit device address and a direction bit. If a slave recognizes its own address, it acknowledges the transmission. Then, any number of bytes can be transmitted to the slave, until the last byte is not acknowledged. Transmission ends with a stop condition.

The data is transferred synchronously with the high state of the clock line, when the state of the data signal has to be stable. This rule is broken only for the generation of the start and stop conditions, which encapsulate a transmission. Transfer speed is not fixed, but specifications guarantee speeds

up to 100 kHz. If a device cannot cope with the transmission speed it may hold down the clock line, and thus slow the process.

When considering the data transmission rate, we can thus only assume the best case, as there are no lower limits. With 100 kHz (i.e., 100 kbps), we need 2 bits for start and stop, 9 bits for the address (including the acknowledge bit), and 18 bits for the data if we want to read two bytes. All in all, with an inter-transmission gap of an additional bit, we have an effective data acquisition rate of 100 k/30 bps, or 3333 samples per second. Therefore, one can see the I2C bus cannot be used for audio data, or other high-speed applications, but it is still fast enough to easily survey an array of sensor devices and react in under a millisecond.

Also, several extensions exist to the basic specification of the bus. One extension is a fast mode which allows transmission of data at rates up to 400 kHz. Another extension in the address space supports 10-bit wide identifiers. The Access.bus implements a software protocol on top of the I2C-bus hardware in order to communicate with computer peripherals.

From Consumer Electronics to Radio Communications

The chips available for the bus cover a wide range of needs. Originally the bus was designed for computer-controlled TV sets, resulting in a wide range of tuning circuits, colour controllers and video-text controllers.

Nowadays several micro-controller families are also on the market offering direct I2C-bus support, with A/D-D/A converters, power switches, electronic potentiometers and LCD-display drivers. These micro-controllers allow the connection of slower peripherals to the bus.

I2C bus-controlled ICs are also available for telecommunication services ranging from pagers to GSM telephones. Using these services would make it easy to start actions via a pager call to your computer.

Putting the I2C Bus and Linux Together

When I started to experiment with the I2C bus, I had to build an adapter for my PC. The simplest solution to connect them involved a TTL chip and several resistors. This adaptor occupies my second parallel port and has served me well for a long time.

In essence, transmission of data is accomplished by control of the states of the lines. This type of device is also common to many types of commercial products, such as TV-cards with teletext receivers.

More sophisticated solutions involve a dedicated adapter chip, called the PCF 8584. This chip implements most of the needed protocol, and offers some nice extras, like interrupt generation, bus monitoring and a long distance mode.

Originally, I supported only my own adapter, but I am currently implementing drivers for others.

Controlling the bus via Linux is simple. As usual with character devices, you have only to open the device file, set the slave address (via an ioctl call) and read or write data. Depending on the adapter, you can set different options.

If you don't like C, I have written a little extension to Tcl/Tk which gives direct control over the I2C bus, allowing a comfortable way to visualize acquired information. The extension is also a convenient interface to the bus for debugging.

is currently studying computer science in Linz, Austria. He can be reached at simon@tk.uni-linz.ac.at.

Archive Index Issue Table of Contents

Advanced search

The Death of Xenix

Evan Leibovitch

Issue #35, March 1997

By the time you read this, the SCO in “SCO Xenix” will stand for “Software Considered Obsolete”. Is there an opportunity here?

As of January 1, 1996 the Santa Cruz Operation (SCO) streamlined its product offerings by dropping a number of older releases from its lineup. Until last December 31, you could still buy OpenServer 3.0 even as version 5.0.2 neared release; not any more.

While few will mourn the passing of most of SCO's 1997 New Year's obsolescence list, I know of at least a few old comrades who will light a virtual candle or two for venerable SCO Xenix, that throwback to the time when Microsoft was a Unix OEM.

Ah, Xenix... My first taste of it was in mid-1985, when it was just about the only thing with genuine Unix code you could get to run on the 286s and 386s of the day. I wasn't at a college where I could play with BSD-filled VAXes, I couldn't afford the 68000-based offerings of the day from Sun and HP, and I didn't feel comfortable with the wide assortment of available Unix clones such as Coherent or QNX or OS9.

In the years that have passed, much has changed; Unix has changed hands many times, DEC and IBM discovered Unix (thus giving us the OSF and the Unix Wars), CPUs have become both more complex and more simple (i.e., RISC), and hardware such as accelerated video and CD-ROMs are commonplace. Until December, though, the one constant was Xenix.

Now before you check the cover of this magazine to wonder why you're reading so much about SCO stuff in a Linux magazine, please understand that the death of Xenix has great significance to the Linux community. It represents a major opportunity for Linux supporters to make inroads into the world of business computing. It presents us with opportunities to not only make money,

but also to build the reputation of Linux as a stable platform more than capable of running a business.

While Xenix may be going away, the reason it existed for so long won't. Xenix serves as a workhorse for small businesses—places where you might find a dozen or so dumb terminals and a couple of printers plugged into a system running FoxBase or other small database system. It's simple, small and stable, maintaining such anachronisms in the current Unix market as unlimited-user licenses and command-line administration.

Despite the appeal of GUIs, the simplicity of a command-line interface still holds an attraction for many people running small businesses. While metaphorical desktops are being moved to PCs as the dumb terminals break down, basic database tasks are still handled more than adequately by Xenix running on minimal hardware. Who needs the bloat of the X Window System on a database server anyway?

Most Xenix users will probably continue to use their systems as-is until they break, exemplifying the “if it ain't broke don't fix it” school of computer administration. But when these systems **do** break, Xenix will no longer be suitable.

For one thing, Xenix doesn't take to new hardware easily. CD-ROMs, PCI, and even mice are difficult or impossible to implement in Xenix; it just barely supports SCSI disks and tapes. Xenix's networking support, while never very functional, no longer exists as an upgrade in any case—someone wanting to add even a single Xenix box to telnet or share services is out of luck.

This is where Linux comes in. It makes a perfect drop-in replacement for Xenix; it also runs lean and acceptably fast even on slow “legacy” hardware, and it supports inexpensive peripherals very well. Furthermore, Xenix adds several new features small businesses now expect from their systems.

Most businesses have probably grown out of the kind of word processors that had been available on Xenix (ASCII-based WordPefect5; Lyrix; even Uniplex—argh!). At the very least, they've become jealous of what they've seen on PCs; even if they love the way their Unix databases are working, most want to move their word processing and spreadsheet applications onto GUI systems.

If you can talk users into considering Linux, running Caldera's Internet Office Suite or Red Hat's Applixware or even StarOffice, so much the better! But even if not, you'll know their new Linux server will be able to run more than just databases. They can use Samba to enable file and printer sharing, and all the tools are there if they want to set up an Intranet or Internet access.

SCO's answer to stranded Xenix users is to offer one of a number of upgrade schemes to OpenServer Release 5:

- Xenix to 16-user Host system (no networking): $250
- Xenix to 32-user Enterprise system (with networking): $750

Many Xenix users may be uncomfortable upgrading their unlimited-users license to a limited-user OpenServer system, or paying about $4,000 to upgrade to an unlimited-user OpenServer. Anyone wanting to upgrade a Xenix development system will pay an additional $300. Also, OpenServer systems need more RAM and better video than Xenix does to do the same job.

Compare this to the cost of the most expensive Linux distribution. To be sure, there are a few technical quirks going from Xenix to Linux (Linux doesn't have SCO's “custom” installation software, for instance) but these aren't a big problem for Linux installers. Running Xenix binaries has not been a problem for me, at least.

This means that Xenix can be the second big opportunity for Linux systems to make their way into business computing. If the Internet got Linux a foot in the door of corporate computing, upgrading Xenix installations would be like getting in an entire leg and maybe an arm as well.

Of course, all the cost savings in the world won't help if the customer doesn't believe Linux is reliable enough to be as low-maintenance as the Xenix it replaces. There's a common perception within the SCO community (at least, based on the newsgroups) that Linux still isn't robust enough for commercial use. Part of this perception is pure FUD, some is based on experience from people who tried Unix in the BC era (before Caldera), when Linux really wasn't being aimed at the commercial world.

Now that Linux is starting to approach the turf of commercial Unix vendors (not just SCO), the road is going to get rougher. Here's where we leave the realm of computer religion and enter the free market. If something you do or say is going to lose someone a sale and associated commission and profit, they're not going to put up with this quietly. Newsgroup flames are nothing compared to the vulgarities of salesmanship; it's amazing what some people will do to reach their sales quotas.

Until now, putting Linux into a company has rarely been at the direct expense of some other vendor; it's come in as an auxiliary system to do routing, run the Internet connection, or maybe serve as a power user's desktop system. Now with the release of industrial-strength software for Linux such as the ADABAS database, Linux has a legitimate shot at becoming the core for central servers

in business. As it succeeds, and as other Unix vendors start to (publicly) take Linux seriously as a competitor, a lot of mud (and FUD) will be flung our way.

The answer to flung mud is not Amiga-style evangelism, but cold hard fact—case studies, comparative analyses and companies such as Caldera and Red Hat and Crynwr prepared to support freeware-based products. The increasing volume of good coverage Linux is getting in the business computing press helps, too.

To be sure, more is needed. The Linux community *must* agree upon a single software installation and management scheme, just as it standardized file system layouts. As Linux further finds its way into commercial settings, it'll need appropriate features (provided as freeware or commercial software) such as on-line data managers and better RAID facilities. More ISVs need to be encouraged to support Linux; and as they do, they, in turn, need to be supported.

Is the community up to the challenge? I'm betting on it. In the meantime, keep an eye on all those Xenix systems out there. The time will come.

Evan Leibovitch is Senior Analyst for Sound Software of Brampton, Ontario, Canada. He's installed almost every kind of Unix available for Intel systems over the past dozen years, and this year his company became Canada's first Caldera Channel Partner. He can be reached at evan@telly.org.

Archive Index Issue Table of Contents

Advanced search

A Guide to Virtual Services

Chad Robinson

Issue #35, March 1997

In this Part 1 of 2, see how to have a single machine answer connections to multiple IP addresses and respond differently for each. This installment covers WWW services.

As Internet sites grow in number, some clients want to create a presence without dedicating a machine to the task. In many cases, a client may never get the amount of usage which would justify a dedicated machine. People like florists and civic event organizers may not have any other use for the machine, so the extra hardware would go to waste.

The solution to this problem is to use a single computer to serve multiple clients. If the machine is serving web presences, FTP directories, and e-mail services almost exclusively, a single machine with enough horsepower can easily handle the needs of several clients, especially those with low traffic.

However, this solution has a drawback, with which many are familiar. Although several domains can be easily pointed to a single host, the URLs will produce the same results regardless of which service, and which domain, was requested. For example, if tryme.com and comehere.com both pointed to the same host, http://www.tryme.com/ and http://www.comehere.com/ would produce exactly the same main page—clearly not what these organizations would have in mind.

But this problem has a solution—virtual services. A virtual service is exactly like a standard service (Web, e-mail, or FTP), but it provides different results depending on which target was selected, despite the fact that they all reside on the same machine.

This month, we examine how to set up and use a virtual web service. Next month we cover e-mail and FTP, as well as possible future capabilities. Keep in mind the Perl motto: “There Is More Than One Way To Do It.” As the Internet

has grown, several alternatives have sprung up. This document describes only one method, and by the time you read it, there may be others, so look around before you decide on any one path!

Please note, in all the solutions described, I assume that you have at least two domain names registered and operating, and two IP addresses. You should have working routes to both IP addresses, and nslookup from any outside Internet host should return the correct IP address for each domain name. You can consult other references for information on how to do this.

IP Aliasing

IP aliasing is the key to the virtual services on the level of the operating system. As the intended audience of this article is Linux users and system administrators, I will present only the information pertinent to Linux. Other operating systems may or may not have the capability to do IP aliasing, and you should consult your documentation and other publications for information on them.

Under normal circumstances, only one IP address may be tied to each network device. The virtual services described in this article rely on having a separate IP address for each domain serviced. How, then, does one manage all the traffic with a single machine? IP aliasing makes this possible.

To enable IP aliasing you must first compile your kernel with IP aliasing support. If you don't already have the kernel sources, you shouldn't continue without them. Obtain a fairly recent kernel distribution from a suitable mirror site and unpack the archive, typically into /usr/src/linux. Then go into the directory and re-make the kernel. Be sure to answer “Yes” to the question about whether or not to support IP aliasing.

If you think your kernel might already support IP aliasing, a quick way to check is to verify the existence of /proc/net/alias* files. Once you have the support enabled, you simply need to configure the interfaces in a slightly different manner. Normally this would be done with a combination of **ifconfig** and **route**. The following is a common example:

```
/sbin/ifconfig eth0 10.1.1.10 <other options>
/sbin/route add -net 10.1.1.0 gw 10.1.1.10 <other options>
```

This creates a setup for the **eth0** device (which you should replace with whatever device you use for your connectivity) and adds a route for the local network (**10.1.1.***) to go through it. (This will also automatically handle the route to the device itself.)

To handle IP aliasing, we simply add a colon and the alias number to which we want to refer. If we had obtained IP addresses 10.1.1.6 and 10.54.21.8 from our service provider, and we wanted them both to talk to our ethernet card connected to our T1 router, we would use:

```
/sbin/ifconfig eth0:0 10.1.1.6 <other options>
/sbin/ifconfig eth0:1 10.41.21.8 <other options>
/sbin/route add -net 10.0.0.0 dev eth0:0
/sbin/route add -host 10.41.21.8 dev eth0:1
```

This will set up two aliases for the Ethernet card, 0 and 1, each with a different IP address. All traffic to either IP address will be seen by any daemon listening to the Ethernet device, and traffic anywhere in the 10.* realm will be routed out through the first device, as a safety catch. You may need to modify these lines to support additional options and, perhaps, a default route. That is all there is to it!

Virtual Web Servers

The point of virtual web services is to present different document trees to users requesting pages from the same machine using different domain names. Users receive the main index page and path names associated with a particular domain name, without any knowledge of the other domains which exist on the same machine.

There are actually two solutions to this problem. The newcomer to the scene uses a fairly elegant method where the client, in its request, also specifies the exact target it was looking for. However, this works only for web services, and only with quite recently released clients from Microsoft and Netscape. If you want to support everybody without relying on the client to make your services work, you will need another solution.

The problem is fairly simple once you understand it. You need a modified HTTP daemon listening to requests coming in to a specific IP address, rather than all those directed to the current machine. Then a server is started for each virtual client, with options specifying different configuration files, document source trees, and so on.

Most web servers now support the requirements for virtual services, but some do not. You will need at least version 1.5 if you use the NCSA server. I use the Apache server, version 1.1.1. Other servers designed as “drop-in” replacements for the NCSA daemon should have this capability, but you should check your server documentation for details on configuring this feature.

To date, almost every server has a different configuration. This article covers the Apache daemon only because it is what the author uses, not because the author considers the server to be more or less capable than any other.

Setup

Once you have **ping** working on the two domain names, you can begin to configure your virtual web services. The most important thing is to select an intelligent document tree layout. If you only have a few clients, you might have a single source root with different subdirectories, one per client. Their tree would then be rooted at their respective subdirectory. If you have more clients, you may need a more complex layout. It is important to decide this **now** because changing it later can become quite messy.

In your server configuration file, you need to set up services for each domain. This is easily done in Apache by enclosing configuration statements within a **<VirtualHost>** container. For example, the following configuration for 10.1.1.6 (the IP address we obtained for www.tryme.com) would be changed from:

```
ServerName www.tryme.com
ServerAdmin webmaster@tryme.com
DocumentRoot /usr/web/tryme/docs
TransferLog /usr/web/tryme/access.log
ErrorLog /usr/web/tryme/errors.log
```

to:

```
<VirtualHost 10.1.1.6>
ServerName www.tryme.com
ServerAdmin webmaster@tryme.com
DocumentRoot /usr/web/tryme/docs
TransferLog /usr/web/tryme/access.log
ErrorLog /usr/web/tryme/errors.log
</VirtualHost>
```

This will instruct Apache (and several other similar daemons) to accept requests with those configuration parameters **only** for those requests directed to 10.1.1.6, in this case www.tryme.com.

Note that this automatically disables server-hosting, and any other targets must be set up as well, or they will not be accessible. Normally, if a machine had several IP addresses, requests directed at any address would be serviced. Including a <VirtualHost> specification prevents this activity. Also note that virtual hosting in Apache can include an optional port number (e.g., <VirtualHost 10.1.1.6:8080>) to provide services for a specific port.

Once you have this configured, start or restart the web daemon, and you should be configured for virtual web services! Next month we examine virtual e-mail and FTP services, and new techniques that provide similar functionality.

Chad Robinson is the Senior Systems Analyst for BRT Technical Services Corporation. He can usually be found behind a monitor and a keyboard, especially if they are hooked to a system running Linux. When he's not programming or administering systems, he is usually spending time with his love, Alison.

Archive Index Issue Table of Contents

Advanced search

NEdit

Dan Wilder

Issue #35, March 1997

NEdit is something new in a Linux programmer's editor.

Here is something new. Not vi, not emacs, not just a wrapper for some hackneyed old Motif widget. With a sparse but sufficient keyboard command set and full regular expression substitutions, NEdit has the best mouse integration I've seen yet in a Linux editor, free or otherwise.

Weighing in at a little over a megabyte for the static-linked Motif binary, NEdit starts up slower than the last editor I wrote about, xvile, but much faster than emacs or xemacs. Scrolling is fast, and command execution shows no perceptible delay.

It's easy to learn, economical on keystrokes due to its excellent mouse support, and not too difficult to customize using X resources; featuring multiple undo/redo, NEdit earns very high marks from me.

Mouse Support

NEdit handles both rectangular and linear regions selected by the mouse. With rectangular regions, you can:

- Select
- Resize
- Drag the region or a copy of it
- Lay region over destination text
- Shift text in the destination out of the way
- Extend or shrink region in any direction

No command or function keys are used for any of these—just mouse clicks, drags, and CTRL, ALT, and SHIFT chording. Many other operations are available via command keys, mostly CTRL- or ALT- modified, including:

- Cut or paste to cut buffer
- Left-justify region
- Replace region with typed text
- Delete
- Regular expression substitution
- Run region through command line filter

Unlike some other editors, rectangular regions are not bound on the right by the shortest line in the region. Likewise, when you move a rectangular region, you can move it as far to the right as you like.

Most things you can do with rectangular regions also work with linear regions, which start some place in a line and incorporate all intervening text until some place in another line. In fact, you can convert between linear and rectangular regions while working with them, just by pressing or releasing the CTRL button.

Linear selection is supported by an xterm, so you may paste text such as command line dialogues, listings of files, and so on, from an xterm directly into a NEdit window. This saves me a great deal of time when filing bug reports and writing release notes.

Motif secondary selection is present also, in both linear and rectangular flavors. This provides some very nice features.

Dragging mouse Button 2 or Button 3 (depending on your flavor of Motif) underlines text; when you release, the underlined text is moved or copied to the location of the I-beam cursor. The position of the SHIFT key chooses move or copy.

If there is a primary selection at this time, the secondary selection replaces it or, if you hold ALT when you release the selection mouse button, it is exchanged with the primary selection. The varieties of secondary selection are great for moving sentences around in a paragraph, re-ordering the clauses of a C language *switch* statement, or inserting text from one editor window to another.

These mouse-based capabilities alone save me perhaps 80% of the keystrokes I'd use in vi for similar text manipulation tasks.

As is customary under Motif, if you select some text, then begin typing, the typed entry replaces the selected text. This provides the handiest way to delete blocks of selected text, using the backspace key.

Design Philosophy

Mark Edel writes:

> The world of GUIs under Unix is a clash of cultures. Unix is all about flexibility and programmability. Modern GUIs are all about uniformity and standards. Finding middle ground is not easy. I'm proud that NEdit can satisfy users who want to change everything ("I want my backspace key to invoke this awk script, and I want all of my buttons blue") as well as those who want to use it right out of the box and learn at their own pace.

> Menus and Dialogs don't automatically make good GUIs. Quality really means working hard to make the program clear, efficient, consistent, modeless, and error-proof. A good GUI under X, for example, must devote a lot of code to basic stability, because critical infrastructure components are not fixed: window management, keyboard focus policy, colors, fonts, key bindings, to name a few.

Problems

NEdit is a work in progress and appears to have some memory allocation problems. I've had several segmentation violations while running it, and the file selection box sometimes loses its state entirely during a long run with many files open, requiring exit and restart before it becomes useful again. These problems have not been severe or frequent enough to prevent me from making daily use of NEdit for some months now, in preference to several other more stable but less usable editors.

It is possible the problems I've observed originate with the Motif to which my copy was linked.

Release 4.0.2 may be out by the time this article reaches press, with some bug fixes and perhaps one or two new features. [It is. Dan says that 4.0.2 is mostly a bugfix release, though syntax colorization is being actively worked on, and an alpha version of colorizing NEdit is available from ftp.fnal.gov—Ed.]

The default key bindings under Linux required some tinkering to get them to where I wanted them, with the Backspace key deleting to the left of the cursor, the Delete key deleting right, and the Insert key toggling between insert and overstrike modes. This and other customizations were accomplished using only

the information available from the help button, however, without resorting at all to the man page.

The user who is accustomed to syntax highlighting won't find it here, though Mark Edel informs me dynamic syntax highlighting is a frequently requested feature. There is a working prototype of this, with continuous incremental re-parsing, on the fly. The syntax highlighting, and also a more complete macro capability, are targeted for release early in 1997.

Volunteers

About 95% of the ongoing work on this editor is performed by Mark Edel. He's willing to incorporate volunteers, who have greatly aided the project with many bug fixes and help with porting NEdit to most flavors of Unix. However, he writes:

> Volunteers are certainly appreciated, with a couple of minor caveats. It is very important to keep the NEdit interface simple, and code that goes gonzo on obscure features faces certain rejection.

He also indicates that the cross-platform nature of this editor makes robust code difficult for many volunteers to write, often requiring rewrite by the core developers before it can be incorporated into the main distribution.

At this point, volunteers are being actively sought to provide emacs bindings for menus and translations, and several ancillary tools, such as ctags and call tree browsers.

Amenities

Keypad support is good, with the Insert, Delete, Home, End, Page Up, Page Down and arrow keys all doing reasonable things.

Ever struggle with some editor's notion of how a C comment block should look? Seems like nobody gets it quite right, and nobody handles all the oddball cases. Now, with NEdit, I can take a typical function header as in the illustration, and with just a few mouse clicks and keystrokes convert this into a much neater form.

In the first figure, I've selected a region to justify. The width of the selection gives the resulting width, with portions of lines to the right folded in. As you can see, there are tear-off menus, and almost everything you need to learn or remember about this editor is shown there.

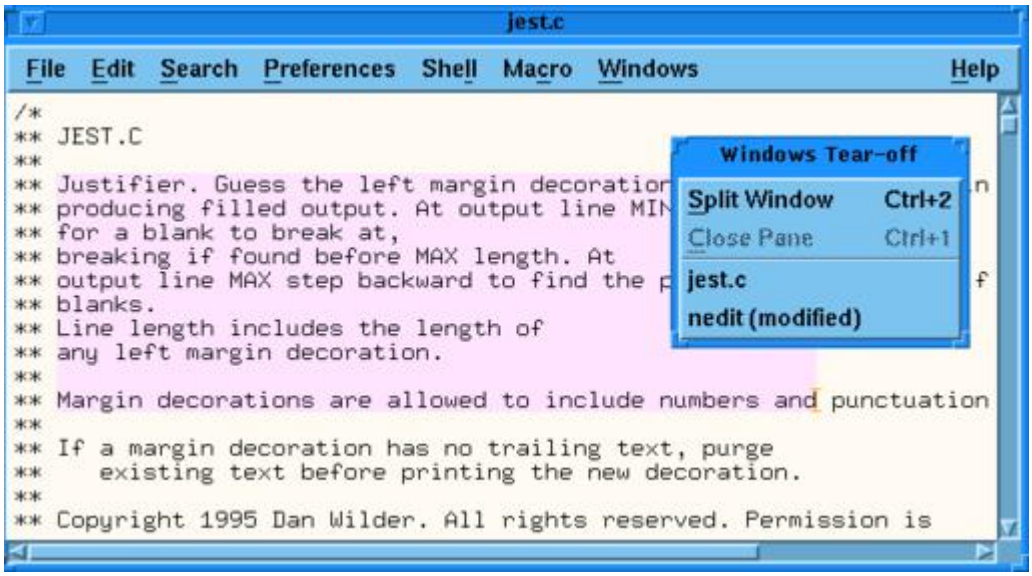


Figure 1. Selecting a Region in a C Comment Block

The second figure shows the text after justification. One justified block has been dragged, and a linear selection is shown, as I am about to hit the Backspace key to delete it.

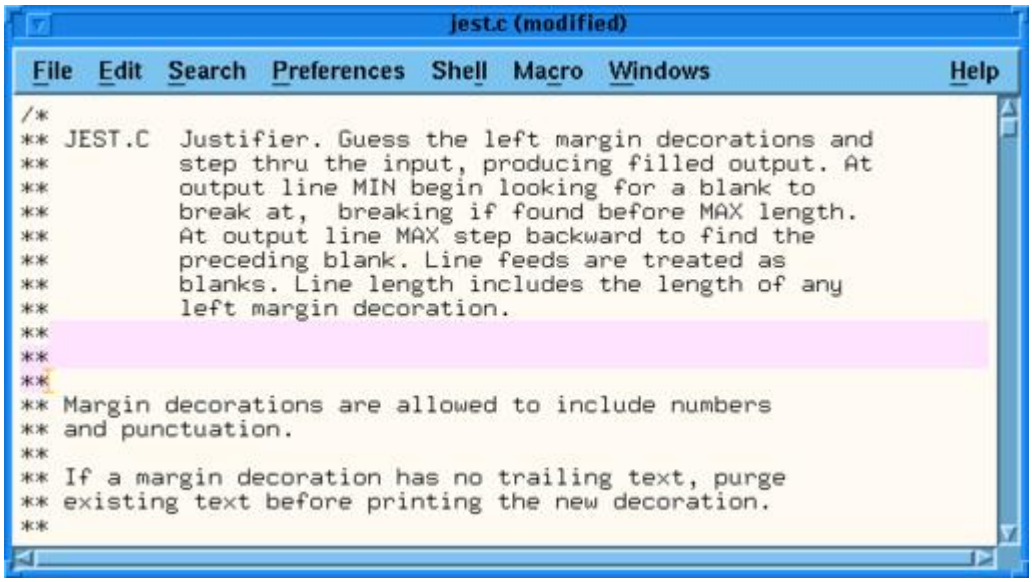


Figure 2. After Justifying the Region

Ctags are supported. Vertical and horizontal Motif scrollbars are present. Windows may be split, and multiple windows may be opened into different files. Regular expression support includes full egrep regular expressions, though find and replace are the only actions available on regular expression matches. Undo-redo extends to the beginning of session, even through file saves.

Vi users will appreciate that the ESC key mostly does nothing. After a while you get over punching ESC constantly, and at that point you might remap the key to something useful.

Some nice touches for programmers allow you to select the line number in an error listing from a compiler shown in an X window, then go to NEdit and locate that like number with just a CTRL-E. “Search” likewise allows searching for text selected in some other window. “Tag” search likewise will pick up a selection from another window. So will ''file open'', in which you select text containing a file name, then open that file using CTRL-Y.

Fairly detailed customizations including key rebinding, and some scripting may be accomplished using X resources. Minimal macro support is available with the current version (4.0.1). Shell escapes may be used to run scripts on the edit text.

Of near-production quality editors I've used, NEdit is the easiest to learn. This is partly because the keyboard command set is sparse, and partly because the menu bar at the top gives the keyboard accelerators for most of the common actions. After you've looked an action up several times in the pull-down menus, you may remember the keyboard accelerator and not need the menus. On-line help is well indexed; terse, but complete. For the most part, neither novice nor expert need refer often to the very excellent and detailed man page.

Resources

NEdit is available under GPL as source or static linked Motif binary from: ftp://ftp.fnal.gov/KITS/pub/NEdit/v4_0_1/. Building from source requires Motif libraries, not at present available as freeware. NEdit is on the LessTiff test suite, and so it should be one of the first large applications to run under this free Motif workalike.

Two discussion groups are NEdit_discuss and NEdit_announce, available by sending e-mail to mailserv@fnal.gov containing either **subscribe nedit_discuss** or **subscribe nedit_announce** in the text. To unsubscribe, send e-mail to mailserv@fnal.gov with either **unsubscribe nedit_discuss** or **unsubscribe nedit_announce** in the body of the e-mail.

Dan Wilder is a computer-literate young man who lives in Seattle, Washington. He devotes his spare time to collecting text editors and Eiffel compilers, and taking care of his mother-in-law's small apple orchard. Dan may be reached by e-mail as dan@gasboy.com.

Archive Index Issue Table of Contents

Advanced search

Setting Up UUCP

Jim Hill

Issue #35, March 1997

Does setting up UUCP scare the hell out of you? No more! Read on.

Discovering the Internet in a college environment, I was always very casual about the time I spent on-line. Since I didn't get a direct bill from the university, there was no reason to keep track of it. All that changed when I left the college world and got a commercial Internet service provider (ISP). As I read news and mail on-line, I realized the vast majority of my on-line time was spent not in transferring data, but in moving my eyes back and forth across the screen. From that simple observation began my long, strange journey into UUCP. Despite the fancy hardware and software in use today, UUCP remains an ideal way for those of us in the backwaters of the Internet to be connected. This article is

This article covers the simple setup: one computer, one phone line, one ISP, and calls originating only from the reader's machine. Those with more complicated setups are referred to *The Linux Network Administrator's Guide* by Olaf Kirch; Vince Skahan's excellent FAQs (see http://www.ssc.com/linux/howto.html) on UUCP, Mail, and News; *Managing UUCP and Usenet* by Tim O'Reilly and Grace Todino; and a competent psychiatrist.

A Brief History of UUCP

Dating back to the late 70s, UUCP (Unix-to-Unix CoPy) was developed at AT&T Bell Labs to provide them with simple dial-up networking capabilities. As is typical with such software, improvements and enhancements were quickly implemented so that, today, several distinct “flavors” of UUCP exist. The most noticeable difference to the user is in the names and locations of the various configuration files UUCP uses. The original style, Version 2, is mostly defunct and is mentioned only for completeness. The HoneyDanBer (HDB) implementation, developed in 1983, uses rather hard-to-understand configuration files, but by virtue of its age, it is in fairly heavy use. Increased

flexibility and ease of use are provided by Ian Taylor's Taylor UUCP package. As an added bonus, Taylor UUCP is capable of understanding HDB configuration files. Most, if not all, Linux distributions are released with Taylor UUCP; the current release is v1.06.

Installing UUCP

I run Red Hat Linux, but I prefer to compile my own packages, so installing UUCP was as simple as installing the SRPM file and compiling:

```
rpm -i uucp-1.06.1-4.src.rpm
```

To compile, edit the top-level Makefile.in to define your installation directories. Run **sh configure** to produce the Makefile and the configuration header, config.h. Browse through these files to make sure they say what you expected them to say. Edit the policy.h header file to customize it to your system; this is heavily commented, so the only thing to look out for is the default to HDB-style configuration files. I'd suggest changing these to Taylor format. Type **make** to build the software, then **uuchk** to verify your configuration file formats. **make install** installs the software, and now you're ready to begin configuration.

Configuring UUCP

Taylor UUCP uses a handful of interdependent configuration files which I chose to put in /etc/uucp. The first configuration file, called config, sets the most general properties:

```
# /etc/uucp/config -- UUCP main
# configuration file
hostname perrin
```

The **hostname** is the UUCP name you and your ISP agree upon—mine is perrin. If this entry is absent, UUCP will attempt to obtain it via **hostname()**, but if your UUCP name and your system name are different, you will be unable to log in to your ISP. Once you have UUCP set up, don't change this arbitrarily, as your ISP will have entries in his config files which correspond to this name. A change will prevent you from accessing his machine. Other parameters in this file can be set to override compiled defaults, but I recommend specifying the defaults in the UUCP Makefile when you compile.

The next required file is sys, which contains information about the systems to which perrin can connect. There will be only one entry in this file, since perrin will be connecting only to my ISP's machine. Its UUCP name, by the way, is sloth. Here's my sys file:

```
# /etc/uucp/sys - name UUCP neighbors
#  system:  sloth
remote-send ~
```

```
remote-receive ~
local-send ~
local-receive ~
command-path /usr/sbin
commands rmail rnews
system   sloth
time     any
phone    123-4567
port     serial1
speed    38400
chat     ogin:  UUCP_LOGIN_NAME
ssword:  UUCP_LOGIN_PASSWORD
alternate
phone    123-6789
```

The **remote-send, remote-receive, local-send**, and **local-receive** entries specify the directories in which UUCP expects to find the files it will be manipulating. By default, this is /var/spool/uucp for a Taylor UUCP configuration.

The **command-path** and **commands** entries specify what programs uuxqt is allowed to execute and where to look for them. I spent two days trying to figure out why my ISP couldn't feed me news articles before noticing I hadn't put rnews in a directory in the command-path, so look out for this item. If you want uuxqt to be able to execute other programs, you must include entries for them. For example, to allow Fred in Pittsburgh to use your expensive color laser printer, add **lpr** to the list of commands.

The **system** entry must be the UUCP name of the system you're calling, because HDB and Taylor UUCP check system names. Ask your ISP what his UUCP name is for this entry.

The **time** entry is used to specify times when connections to sloth are permitted. I don't know about the average Linux user, but when I think of running specific jobs at specific times, my thoughts turn naturally to cron. By specifying any time in the config file, I can initiate a UUCP connection via a **crontab** entry and not worry about UUCP quibbling over the time of the call.

The **phone** entry, unsurprisingly, specifies the ISP's phone number. If your ISP has several access numbers, they can be specified with the **alternate** field. In the above example, if a call to 123-4567 fails for any reason, UUCP will attempt to place a call to 123-6789, with all other configuration data remaining unchanged.

The **port** entry is not the port to use, but an entry in a file named port which specifies the port to use. For single-modem machines like mine, the port file will contain a single entry, which we'll look at in a moment.

The **speed** entry is the speed at which the port will be transferring data.

The **chat** entry contains a brief chat script used for logging in, which should be familiar to anyone who has used SLIP or PPP. The space-separated fields

represent messages which are alternately sent and received. In most cases, the only essential data which must be transferred are the login name and password. If you want a more detailed look at the messages sent out by the ISP's machine, check with your ISP.

Now the port file:

```
#  /etc/uucp/port  -  UUCP ports
port     serial1
type     modem
device   /dev/modem
speed    38400
dialer   generic
```

The **port** entry here must correspond to the port entry in config. The **type**, **device**, and **speed** entries let UUCP know the device file and speed to use. UUCP will create a lockfile based on the device name, so you should use the same device your other communication software specifies. This way, you will avoid having one process disrupt another. (If you are running UUCP via cron jobs, it's extremely likely the time to connect will come around at least once while you are already using your modem, such as for web browsing.) The last entry, **dialer**, specifies an entry in the last configuration file, called dial.

```
#  /etc/uucp/dial  --  per-dialer info
#  My modem
dialer     generic
chat       "" ATZ OK ATM0DTT CONNECT
chat-fail  BUSY
chat-fail  ERROR
chat-fail  VOICE
chat-fail  NOsCARRIER
chat-fail  NOsANSWER
chat-fail  NOsDIALTONE
```

The **dialer** entry again matches that in **port**. The **chat** entry specifies a chat script which initializes the modem and places the actual call. When this entry is read, T is replaced by the phone entry in sys. The **chat-fail** entries provide a list of conditions under which the connection fails and the whole process aborts, or tries an alternate phone number if one is provided.

I've had a problem using the phone line for voice conversation—when cron tells the system to call, the modem will dial, producing the touch tones and interrupting my conversation. Only after dialing does it check for any of the chat-fail conditions and abort. If you know of a way to make the call abort as soon as the modem activates and doesn't “hear” a dial tone, please let me know. My friends and family are growing tired of my automated machine. [I think checking for dial tone is a configurable option on most modern modems —Ed.]

Running UUCP

UUCP is actually a suite of programs to do very specific tasks. For example, uucp itself is used for copying files between nodes (the machines connected via UUCP) and uux is used for executing programs on another node. Programs exist for all kinds of maintenance, like logfile-trimming and spool-checking. For my purposes (and the purposes of this article), the most important programs are **uucico** and **uuxqt**. uucico actually places the phone call and sets up file transfers, while uuxqt tells the other machine what program needs to be run for proper handling of the files.

The following sequence of events is typical:

Typing **uucico -s sloth** causes uucico to look up sloth in config. Seeing it should use **serial1** to connect to sloth, it looks in **port** and sees that **serial1** is the modem, which is activated by the **dialer** entry. Peeking at this entry in dial, uucico initializes the modem and calls the number specified in sys. When the **CONNECT** string is received, it executes the chat script from sys and logs into sloth.

When the login procedure is complete, perrin is in “master” mode and sloth is in “slave” mode. Files to be uploaded will be in the spool directory /var/spool/uucp/sloth/D./filename. If these files exist, perrin will upload them with instructions for the slave. The instruction files will be in /var/spool/news/uucp/sloth/C./filename. When the transfer is complete, the master and slave exchange roles, with perrin now receiving any files spooled on sloth, as well as execution instructions. When both sides have transferred all the necessary files, the connection is terminated. Logging is done in /var/log/uucp, so take a look in there for an exhaustive roster of an average session's work.

When the connection is broken, the second important UUCP program is fired up: uuxqt. uuxqt looks in the UUCP spool directory for execution requests and (if permitted) executes them. For example, files consisting of mail messages must be delivered and news postings must be moved into the news spool. By default, UUCP permits only two local programs, rmail and rnews, to be executed, which not-so-coincidentally accomplish the tasks just mentioned.

With UUCP configured and tested, it's now time to set up the transfer of mail and news.

Mail via UUCP

For mail transport and delivery, the two most obvious choices are sendmail and smail. I have read that for small sites the two are roughly equivalent in configuration difficulty, but I've also seen O'Reilly's sendmail book. Nothing that

massive could possibly be required for my little project. Accordingly, I chose smail. The current release is v3.1.29—while not part of Red Hat's distribution, some kindly soul has made an RPM available in the /pub/contrib directory at ftp.redhat.com.

Installing smail

After looking at a full source tree for smail, I chickened out and grabbed a precompiled rpm from Red Hat's /pub/contrib directory, then installed it:

```
rpm -i smail-3.1.29.1-6.i386.rpm
```

Configuring smail

Two links to smail are needed: usr/bin/rmail and /usr/sbin/sendmail. The former is invoked when mail comes in for delivery via UUCP; the latter is often hardcoded into mail user agents, such as elm. To create these links, use the following commands:

```
ln -s /path_to_smail/smail  /usr/bin/rmail
ln -s /path_to_smail/smail  /usr/sbin/sendmail
```

Note that **rmail**, or a link to it, must be placed in the command-path specified in etc/uucp/sys or this will fail. Your ISP might have permission to run rmail, but if he can't find it then he'll get all sorts of error messages in his UUCP logs and might just send you a nasty e-mail message. Of course, if you've already arranged for mail to be sent via UUCP, this will backfire on the ISP, and you will quickly find yourself less than popular. Or so I hear.

The main configuration file for smail is etc/smail/config. For a site which will be doing all its mailing through a UUCP link, this is a remarkably simple file, especially since the smail package comes with very nice sample files in etc/smail/config.linux. Of the four files smail will use, I had to modify only config for my particular setup:

```
#  /etc/smail/config
smart_path=sloth
smart_transport=uux
visible_name=swcp.com
uucp_name=perrin.swcp.com
```

The **smart_path** entry is the UUCP name of your ISP's machine. This will match the system name in /etc/uucp/sys. Any non-local mail address will be shipped off to this machine for DNS resolution and delivery. Specifying uux as the transport agent will cause any outgoing mail messages to be queued in the UUCP spool to await the next UUCP connection. I will return to this when I discuss the other configuration files for smail.

The **visible_name** entry identifies your smail domain. If you have registered your UUCP name, append **:uucp** to this entry. In many cases, this is unnecessary. Using my system as an example, the **.swcp.com** portion is guaranteed by the InterNIC (and my ISP's hard-earned dollars) to be unique. Therefore, the only machines which could have already taken perrin would be connected to my ISP, who would notify me of a conflict.

The **uucp_name** entry is (surprise) your system's UUCP name. By default, smail will generate return paths from the hostname command, in my case perrin. Since I have not registered perrin, someone else might. Without this entry, any mail returned to perrin will go to that other machine. By specifying a fully qualified domain name, I am guaranteed (because of the way DNS and UUCP addressing are resolved) the message will go first to swcp.com, which will recognize perrin as my UUCP account. For a machine named “perrin”, this is a negligible concern, but a more common name, e.g., “darkstar”, might cause problems.

Running smail

For the stand-alone system under discussion, smail is best run as a daemon all by itself. Make sure this entry appears in etc/services:

```
smtp 25/tcp #Simple Mail Transfer Protocol
```

This will specify port 25 for SMTP connections, which the mail transport agent (MTA) will use for delivery.

Run smail as a daemon by having a system startup script run the following:

```
/path_to_smail/smail -bd -q10m
```

The **-bd** option causes smail to run as a daemon; **-q10m** tells it to process the message queue every 10 minutes. If you do light mailing (or do a lot during a solid block of time), consider increasing this, perhaps to two hours or more with a command line option like **-q2h**. When a connection to the SMTP port is detected (such as when UUCP hands off the latest batch of local mail), smail will fork and handle the SMTP connection.

When smail gets a wake-up call from a mailer (like rmail, for incoming UUCP deliveries, or elm, for outgoing messages), it looks in the file /etc/smail/routers to see what to do with the message. Here's my file:

```
# /etc/smail/routers
smart_host:
driver=smarthost,#What do I do with nonlocal mail?
transport=uux; # Deliver it via UUCP-path
               # to the machine specified with the
               # smart_path option in ./config
```

Upon receiving a message intended for a non-local address, smail checks this file. It sees it is to use the smart_host, which is defined in /etc/smail/config as sloth, the ISP's machine. The message is to be forwarded to sloth using the transport option, which is set to uux. uux is another program in the UUCP suite that queues the message with instructions to run mail in the UUCP spool to await the next UUCP connection.

Local mail is handled partially by an entry in the /etc/smail/directors file. The purpose of the directors file is to tell smail about some of the special options available locally, such as aliasing, forwarding to another user or to a pipe, or just about anything else you could set up. (Mailing lists on a two-user machine, anyone?) I've cut all but the entries which handle .forward files and the generic “What to do if we have a user who just wants his mail spooled?” scenario. Read the /etc/smail/config.linux/directors file for full gory detail:

```
# /etc/smail/directors
dotforward:
# general-purpose forwarding director
driver = forwardfile,
# problems go to the site mail admin
owner = Postmaster,
nobody,
# sender never removed from expansion
sender_okay;
# .forward file in home directories
file = ~/.forward,
# the user can own this file
checkowner,
# or root can own the file
owners = root,
# it should not be globally writable
modemask = 002,
# don't run things as root or daemon
caution = daemon:root,
# be extra careful of remotely accessible home
# directories
unsecure = "~uucp:~nuucp:/tmp:/usr/tmp"
# user - match users on the local host with
# delivery to their mailboxes
user:
# driver to match usernames
driver = user;
# local transport goes to mailboxes
transport = local
```

Once a delivery option is matched in directors, the transports file is searched to see what, if any, special options that particular delivery mode requires. For the trimmed directors file above, the corresponding transports file would be:

```
# /etc/smail/transports
# append message to a file
local: driver=appendfile,
# include a Return-Path: field
return_path,
# supply a From_ envelope line
from,
# insert > before From in body
unix_from_hack,
# comment out the above line for
# MMDF mailbox format and for
# use with the Content-Length
# header fields.
# use local forms for delivery
```

```
local;
# location of mailbox files
file=/var/spool/mail/${lc:user},
# group to own file for System V
group=mail,
# For BSD: only the user can
# read and write file
mode=0600,
# append an extra newline
suffix="
",
# notify comsat daemon of delivery
notify_comsat,
```

Note that there is no transport for the forwarding option. This is compiled into smail to send the message on to the forwarding address, with a few extra headers. Forwarding to a non-local address would send the message off to the UUCP queue for the next phone call.

Using UUCP to Transfer Files

Now that you've set up smail to wait for UUCP to deliver mail from the outside world and to send mail to UUCP for transfer from your machine, you need to tell UUCP when to transfer files. This is most easily done via a crontab entry. Create it by typing **su uucp**, then typing **crontab -e**. The resulting file should look something like this:

```
# Call for transfers
25 0,9,11,13,15,17,18-23 * * *
/usr/sbin/uucico -s sloth
# Trim logfile
35 03 * * * find /var/log/uucp
-size +16k -exec cp /dev/null {} ;
```

The first line calls sloth every few hours during the day and hourly through prime time. Every afternoon it trims UUCP's logfile, which can grow spectacularly with frequent transfers.

News via UUCP

Setting up mail didn't take much time, and I was feeling pretty smug when I got my first e-mail from “the outside”. Humility came back with a vengeance when I turned my attention to setting up news.

News Transport Agents

Like mail, a variety of software packages are available for getting news on and off a local machine. Again like mail, the best known is big and intimidating. Unlike mail, the lesser known is **also** big and intimidating. The “big boy” in news transport is InterNet News (INN). A close second is the C News package. I took a look at the documentation available for these two packages before deciding which to try. INN runs as a daemon and can be a disk and memory hog. While it can be configured to operate in a situation such as mine, it really is intended to

be run on dedicated news servers with high-speed connections and many users. C News is a collection of binaries and scripts that are executed by cron and consumes minimal disk resources. While there's more documentation available for INN, I decided that C News was the way to go. Besides, I just wouldn't feel right as a Linux user if I could just follow directions and have a working system.

Installing and Configuring C News

C News is sufficiently system-dependent that compiling from a source distribution is the only logical thing to do. The current release of C News can be found at ftp.cs.toronto.edu in /pub/c-news/c-news.tar.Z. After uncompressing and extracting the source tree, take a moment to read through the files in the docs/ subdirectory; there's some interesting material in there.

To configure C News for your system, follow the instructions in the file README.install. Before getting started, edit the file conf/update.ran. It lacks an initial line of **#!/bin/sh** and will cause the Makefile to exit with an error. After reading through the documentation, begin installing.

1. Run “quiz”. This brief interactive script sets up the defaults for C News based on your answers. In most circumstances, either the default is fine or you will have an intuitive grasp of a preferred alternative (such as deciding to store articles in /var/spool/news instead of /usr/spool/news). However, there are a few things to look out for:
    - The default for where to store the news control files is /etc/news. There are a handful of files in this directory that get pretty big, so if you have a small root file system you might want to put these on another file system.
    - quiz will ask about the presence or absence of 13 system functions. On my Red Hat 3.0.3 system, all but **fgetline** were present.
    - quiz will build your makefiles for you, so it needs to know what type of syntax to use for specifying include files. Specify SVR4.
    - Make sure you specify the same style of UUCP configuration files you used in setting up UUCP, HDB or Taylor.
    - Specifying the paths to rnews and inews: The paths to these programs should match the command-path line from your UUCP sys file.
    - quiz will ask in what format the system reports free disk space. I took a wild guess, considering Linux's similarity to SVR4, and answered **statfs**. Judging from the working setup, I seem to have guessed correctly.

2. Run **make** to build the software, then **make r** to run some regression tests to check out the build.
3. Create the directories you specified in quiz for the article tree, the overview tree, the binaries, and the control programs. C News calls these NEWSARTS, NEWSOV, NEWSBIN and NEWSCTL, respectively. I put the articles and the overviews in /var/spool/news, the binaries in /usr/lib/news, and the control programs in etc/news. The ownership of /NEWSBIN is largely arbitrary, but the other three should be owned by the /news administrator. I made that user **news**, group **news**. If you don't have a news user, create one with adduser or whatever tool you prefer. Some newsreaders look for news in usr/spool/news, so you might need to create a symbolic link in /usr/spool to your NEWSARTS directory.
4. Use **su** to log in as the news owner and run **make install**. You might have a permissions problem trying to write the binaries to NEWSBIN. I got around that by switching virtual consoles, giving everyone write permission to that directory, running **make install** as news, then restoring the original restrictive permissions. Run **make setup** next as news.
5. As the owner of NEWSBIN, run **make ui**.
6. C News comes with a news reader, poster and checker which can be best described as functional. You will not want to use these beyond testing your setup, but it's worth installing them for that purpose. Do this by running **make readpostcheck**.
7. Make sure NEWSBIN/input/newsspool is owned by news and in the news group. Change its permissions to **rwsr-sr-x**.
8. As **news**, change directories to NEWSCTL and configure the important control files for your particular setup:
    - **batchparms**: By default, C News will try to deliver articles as soon as they arrive. For our setup, it's better to collect the articles into a single compressed file and deliver it periodically via UUCP. This is called batching and the file **batchparms** sets up the size and number of batches. Here's mine:

      ```
      # NEWSCTL/batchparms
      # 500KB, after compress, is 4 minutes at
      # 1000cps 20 batches is somewhat arbitrary,
      # about 5MB per site
      # site class size  queue command
      # - -- -  -- --
      sloth u 500000-750000 20 batcher|
      compcun | viauux
      ```

      As with mail, sloth is my ISP's UUCP host. The **u** class option specifies that batches will be transferred via UUCP. The size of a batch is set to 500KB nominally, with a maximum size of 750KB. If a single value is specified, it is the nominal value, with the maximum being 3 times the nominal. The queue field allows up to 20 batches enqueued. The pipelined command in the final field specifies what to do with a

batch. The batcher takes the spool of outgoing news articles and assembles them into batches of nominal size. It passes the batches off to **compcun**, which compresses the batches. By default, compcun uses the **compress** function which, while not as efficient as GNU's gzip, is certain to be understood by any system. If you feel passionately about using gzip, contact your ISP to see if his system has gzip installed. (Don't laugh, some ISPs use software from their hardware providers and refuse to install any Free Software Foundation products. Now I'm serious: stop laughing.) compcun now hands off its compressed articles to **viauux**, a C News front end to UUCP, which delivers the compressed and split articles into the UUCP spool for delivery. Incoming news goes through the reverse process if it is batched: delivery by UUCP, decompression, unbatching, and off to the news spool.

- **controlperm**: The entries in controlperm specify who and what sites are allowed to send control messages for what groups. The example file included with the C News source is pretty self-explanatory, so I'll let this topic go with a word of warning: if you're running C News over a UUCP link from your Linux box, you are not important enough to send out newgroup or rmgroup control messages. Don't do it.
- **explist**: Unless you have a magic hard drive, your news spool will soon fill up if you don't throw away (“expire”) the older articles. Every day or so, you should run the program doexpire, which checks the explist file to see how long to keep articles in the spool. Entries in explist take the form:

```
grouplist  perm  times  archive
```

The **grouplist** is a comma-separated list of newsgroups to which the line is to apply. Subhierarchy expansion is automatic, so to expire the entire rec.* hierarchies, a grouplist of rec is sufficient. The **perm** entry indicates the type of group: **u** = unmoderated, **m** = moderated, **x** = either.

The **times** entry is generally a single number, though it can be a dash-separated list of three. If the former, the entry indicates how many days after arrival at your site an article should be kept. Of course, if the article itself contains an **Expires:** header which evaluates to an earlier date, it will expire then. If the **times** entry is a trio of numbers, the first represents the number of days that must pass after arrival before the article is considered “expirable”. I'm sure someone could come up with a reason why a number other than zero is desired, but I'm at a loss. The second number is the days-to-expiration as seen in the single-digit format, and the third is an absolute number of days the article is permitted to dodge expiry.

The **archive** field takes a dash if no archiving is to be done, a full path name to an archive file, or an **@** sign. If **@** is used, the archive file must be specified with the **-a** option when **doexpire** is run.

```
# NEWSCTL/explist
#grouplist perm times archive
# hold onto history lines 14 days, nobody gets
# >90 days
/expired/   x 14   -
/bounds/   x 0-1-90  -
# High-traffic or trivial groups get dumped
# quickly
control,junk,swcp,perrin x 3 -
# default:  14 days
all    x 14 -
```

Note that the earliest match is accepted, so when doexpire sees the 3-day limit on the control group, it takes precedence over the 14-day catch-all at the end. If you want to mix full group names (such as rec.arts.tv) and hierarchies (such as rec.arts.*), remember to put entries into the explist file in increasing generality.

A final comment: the /expired/ grouplist tells doexpire how long to leave article entries in the history file. For multiple feeds, duplicate articles may arrive from time to time. C News keeps a list of articles in the NEWSCTL/history file and junks any duplicates. Lines in the history file are kept the number of days specified by /expired/ so that if a duplicate arrives a few days after the original has expired, it will not be reposted. If you have a single feed which is properly configured and doesn't send multiple copies out to its leaf sites, you can keep this value small.

- **mailname**: Self-evident. My mailname file contains the single line:

```
# NEWSCTL/mailname
perrin.swcp.com
```

- **mailpaths**: Posts to moderated newsgroups are mailed to the moderator, who decides to post it or not. The mailpaths file has a line entry for each moderated newsgroup consisting of the group name, a tab, and the e-mail address of the moderator. The only moderated group I read is comp.os.linux.announce, so my mailpaths file is simply:

```
# NEWSCTL/mailpaths
comp.os.linux.announce linux-announce@news.ornl.gov
```

- **organization**: This is just a string containing the organization you are posting from. For a home site, an entry like “Private News Server” is adequate, though you might give into temptation and come up with something dazzlingly funny.
- **sys**: sys is the most important file for talking to the outside world; it specifies what hierarchies you get, from where, and what news you are willing to pass on. For our special situation, only two lines will be needed. Entries take the form:

```
site[/exclusions]:grouplist[/distlist][:flags[:cmds]]
```

Any article coming from **site** not in the (comma-separated) grouplist is junked. The **distlist** is a (comma-separated) list of distributions we are willing to forward. Any article whose Path: header contains a machine in the list of exclusions is not forwarded.

There are quite a few flags that can be set, but only one is relevant. The **f** flag tells C News to use batching. If you are using batching, the **cmd** entry contains not a command but a file name. If the file name does not begin with a slash, it is relative to /var/spool/news/out.going. If the cmd field is empty, the file is remote-system/togo.

With this in mind, let's take a closer look at my sys file:

```
# NEWSCTL/sys
ME:all/all::
sloth/sloth.swcp.com:all,!perrin/all,!local:f:
```

I gave my ISP a list of the newsgroups I wanted fed to me; he entered them into a configuration file on his end. As a result, I can safely use **all/all** in the **ME** entry and not worry about trying to download an entire Usenet feed at 14.4kbps.

The **sloth** entry again specifies my ISP's machine. By putting sloth.swcp.com into the exclusion list, I don't try to send articles back. I created a perrin.* hierarchy for some simple tests; the grouplist entry in my sys file shows I send all newsgroups except perrin up to my ISP. Similarly, articles intended for all distributions (world, usa, nm, whatever) except local are sent up. I will be using batching, and the articles are going to be stored in /var/spool/news/out.going/sloth/togo.

- **whoami**: The machine's **hostname**—in my case, perrin.

9. Return to the C News source directory and **make cmp** to see that everything is correctly installed and configured. You'll undoubtedly want to pause briefly, get up from your computer, and do a little dance when you see the message “no worrisome differences”, for the end is near.
10. Time to automate! As news, create/edit a crontab with **crontab -e**. The entries here will set up a schedule of housekeeping tasks and transfers. Here's mine:

```
#/var/spool/cron/news
09  * * * * /usr/lib/news/input/newsrun
22  * * * * /usr/lib/news/batch/sendbatches
59  0 * * * /usr/lib/news/expire/doexpire
17  8 * * * /usr/lib/news/maint/newsdaily
05  * * * *     /usr/lib/news/maint/newswatch
3000 300 100
```

At 9 minutes past the hour, **newsrun** takes the batched articles delivered by UUCP, unpacks them, and stores them into the news spool.

At 22 minutes past every hour, **sendbatches** takes the outgoing articles, batches and compresses them, and delivers them to the UUCP spool for uploading. These files will be transferred whenever the next UUCP connection takes place, so **su** to uucp and edit the crontab, as you did with mail.

At 12:59 AM, **doexpire** takes care of expiring old articles.

Every morning at 8:17, **newsdaily** tidies up some of the various log files and sends mail to the news admin if something is amiss.

At 5 minutes past the hour, **newswatch** takes a look at the system to see if there are any problems. Stale locks, full disks, and the like are reported to the newsadmin.

11. You need to have newsboot run at system bootup to clean up any messes left over after a potential crash. To preserve permissions, it needs to be run as user **news**. I tacked the line:

    ```
    su news -c "/usr/lib/news/maint/newsboot"
    ```

    onto the end of my rc.sysinit file.
12. Installing the man pages. Due to wild system variations, your best bet is just to change into the C News doc/ directory and copy them manually into your man page subdirectories.
13. Create a local test group by:

    ```
    cnewsdo addgroup my-site.test y
    ```

    and post to it. If you made **readpostcheck** you can do this with the postnews package. Look in NEWSARTS/in.coming and you should see the article, with header information thoughtfully provided by postnews. Leave the computer room and share some time with your family until after newsrun executes again. Now, your article should have appeared in NEWSARTS/my-site/test and should also have entries in the history and log files in NEWSCTL. If you installed readpostcheck, use **readnews -n my-site.test** and you should see your message. Pat yourself on the back; you've got yourself a Netnews site.
14. Stop patting yourself on the back. Unless you plan to spend your time sending messages to my-site.test and reading them a bit later, you'll want newsgroups from the Outer World. The installation directions that come with C News cover this step with “Get a feed from somebody, somehow.” That's best handled by sending your ISP a list of newsgroups you want to receive and asking for e-mail when it's set up.
15. As news, edit the files NEWSBIN/active and NEWSBIN/newsgroups. They should both have permission modes 644.
    - **active** contains the list of newsgroups, their article numbers, and the posting permissions. Each line should be of the form:

      ```
      news.group.name  highest  lowest  perms
      ```

:**Highest** and **lowest** are the upper and lower bounds of available articles in news.group.name. For setting up, set highest to 00000000 and lowest to 00001. To save time, C News never updates the lowest value, so if your newsreader (such as trn) looks at lowest, you will need to set up a crontab entry to run updatemin from time to time. The relevant perameterss are **y**: users can post to the group; **n**: users can read but not post; **m**: the newsgroup is moderated. Posts to this newsgroup will actually be sent to the moderator address specified in the $NEWSCTL/mailpaths.

As news comes in, C News will continually update the active file. Here's a snippet of mine:

```
alt.tv.homicide 0000000002 00001 y
alt.tv.nypd-blue 0000000010 00001 y
comp.os.linux.announce 0000000009 00001 m
comp.os.linux.misc 0000000504 00001 y
linux.dev.kernel 0000000048 00001 y
rec.arts.sf.written.robert-jordan 00033 00001 y
rec.arts.tv 0000000334 00001 y
swcp.test 0000000005 00005 y
```

In addition to “real” newsgroups, you should also have groups for control, junk, to.feed-site and to.my-site.

This file can be maintained by hand or by using addgroup and delgroup. Second Warning: do **not** send *newgroup* or *rmgroup* messages; these commands create groups for all of Usenet and are not for the likes of us mortals. As much as you might want to newgroup alt.fan.back-hair on your own initiative, procedures exist for creating new Usenet groups and violating them is a good way to find yourself unpopular.

Something I noticed after a day or three of running C News: control messages were responsible for about 80% of my news spool. To cut back on disk space, I had my ISP remove control from the list of groups he was feeding me. I can no longer receive newgroups, rmgroups or cancels, but it's worth it to save disk space. Forty non-binary newsgroups, most low-traffic, were eating up nearly 8MB of my disk spool after only three or days. Multiply that by the control-group factor of five and you can see why my poor 50MB spool can't afford control messages. Make sure your ISP still accepts control from you, however, as you might need to cancel a stupid question or hastily-posted flame someday.

- **newsgroups** contains a list of the newsgroups which appear active and a brief description of each. When a user reads a new newsgroup, this description should appear to provide a little newsgroup information. This can be handy for keeping pyrotechnicians from wandering into places like alt.flame. I made my newsgroups file by trimming a copy of my ISP's.

16. Final Step: do include a local test group in your feed—your ISP probably has one. Post a message to it, wait for batching and UUCP to do their things, then see if it shows up. If so, post a test message to one of the “real” newsgroups. Disguise the fact that it's a test message by making it relevant and on topic. If it shows up, you're in business. Sit back, relax, have a Coke and smile.

Jim Hill is a graduate assistant at Los Alamos National Lab who has been playing with Linux since somewhere in the .99 kernel series. Armed with total ignorance of the topic of this article, he set out to learn a little something and maybe help somebody else. Thanks to Mark Costlow at Southwest Cyberport (cheeks@swcp.com), without whose help and patience this article would have read “It doesn't work; it can't be done.”

Archive Index Issue Table of Contents

Advanced search

Accelerated-X v. 2.1 & Metro-X 3.1.2

Jon Gross

Issue #35, March 1997

This has not alleviated the pain as much as I hoped. The appearance of Metro-X and Accelerated-X in the offices of *Linux Journal*, begging for a review, has turned my worst nightmare into an exercise in rebellion: now I can reconfigure X for the hell of it, in about twenty seconds.

- Accelerated-X v. 2.1 for Linux
- X Inside Corporation
- Phone: 303 298-7478
- Fax: 303 298-1406
- E-mail: support@xinside.com
- FTP: ftp.xinside.com
- WWW: http://www.xinside.com/
- Price: US$99.95

- Metro-X 3.1.2
- Metro Link Incorporated
- Phone: 954 938-0283
- Fax: 954 938-1982
- E-mail: sales@metrolink.com
- FTP: ftp.metrolink.com
- WWW: http://www.metrolink.com/
- Price: US$99

by Jonathan Gross

I have this recurring dream about XFree86, the free X Window System that comes with Linux. I am sitting in a barren room, with a keyboard and monitor in front of me. I have the manuals for the graphics card and the monitor. I also

have a stack of blank paper and a small calculator. I sit in the room and look up numbers in my manuals, spend five minutes calculating and entering a modeline entry, and then restart X. The shadow from the weak winter sun moves across the wall behind me as I work. It is dark in the room when finally I can bring up a display in 1024x768, and I quietly rejoice... only to have the monitor transmogrify from a nice Mag into a Packard Bell.

I've been having this dream for about three years now, and I'm convinced it is the result of an attempt to configure an old version of XFree86 that took about 12 hours one winter day. XFree86, in its infancy, was a menacing nightmare to configure. In recent years, it has gotten easier: probes have been written and configuration scripts have been included to assist users. This has not alleviated the pain as much as I hoped. The appearance of Metro-X and Accelerated-X in the offices of *Linux Journal*, begging for a review, has turned my worst nightmare into an exercise in rebellion: now I can reconfigure X for the hell of it, in about twenty seconds. Excellent.

Accelerated-X

The first server I installed was Accelerated-X (AccelX) from X Inside. It comes as a tar archive, split across two floppy disks. The documentation comes in the form of an odd-sized manual and reminders to check the X Inside web site for the most current documentation. This is important, as the documentation that came with the software was a little dated. Accelerated-X requires you to have XFree86 installed already; the system requirements are fairly minimal:

- Linux 1.2.13 or later
- 4MB of RAM
- At least 3MB free disk space under /usr
- A supported graphics board (see http://www.xinside.com/bd/ for a current list).

Figure 1

To install AccelX, you simply untar the floppies (as root):

```
cd /usr/X11R6/lib ; tar -xzvf /dev/fd0
```

The configuration for AccelX is a very simple console-mode menu system in which, using arrow keys, you select the options that match your hardware (see Figure 1). All options are fairly well documented in the manual. Once I configured the server, I brought it up as a regular user and started up some applications. The one problem I had was with the mouse—a cheap Microsoft Mouse knock-off. I thought that was the trouble, anyway. It turns out that Linux kernel version 2.0.0 introduced a problem with the Async mouse feature of

Accelerated X. Async mouse is the feature that allows the mouse to move around the screen even when the X server is busy doing something else—a very useful trick Microsoft has been using for a while on their Windows product. X Inside notified the Linux kernel team and worked with them to correct the problem, so that with the release of the 2.0.26 kernel, the situation was resolved, and the mouse no longer wipes applications right off the desktop, killing the connection between the client and the server.

X Inside offers several optional enhancements, including multi-headed display, Motif, and OpenGL.

Metro-X

Metro-X is a little more full-featured than Accelerated X. It comes out of the box with multi-headed display capabilities and touch-screen support for products from three different manufacturers. It also requires that XFree86 be installed. This comes at the price of 12MB of hard drive space, however. Metro-Link also offers Motif and OpenGL at additional cost. System requirements include:

- Linux (no version was specified)
- 8MB of RAM
- 12MB of hard drive space

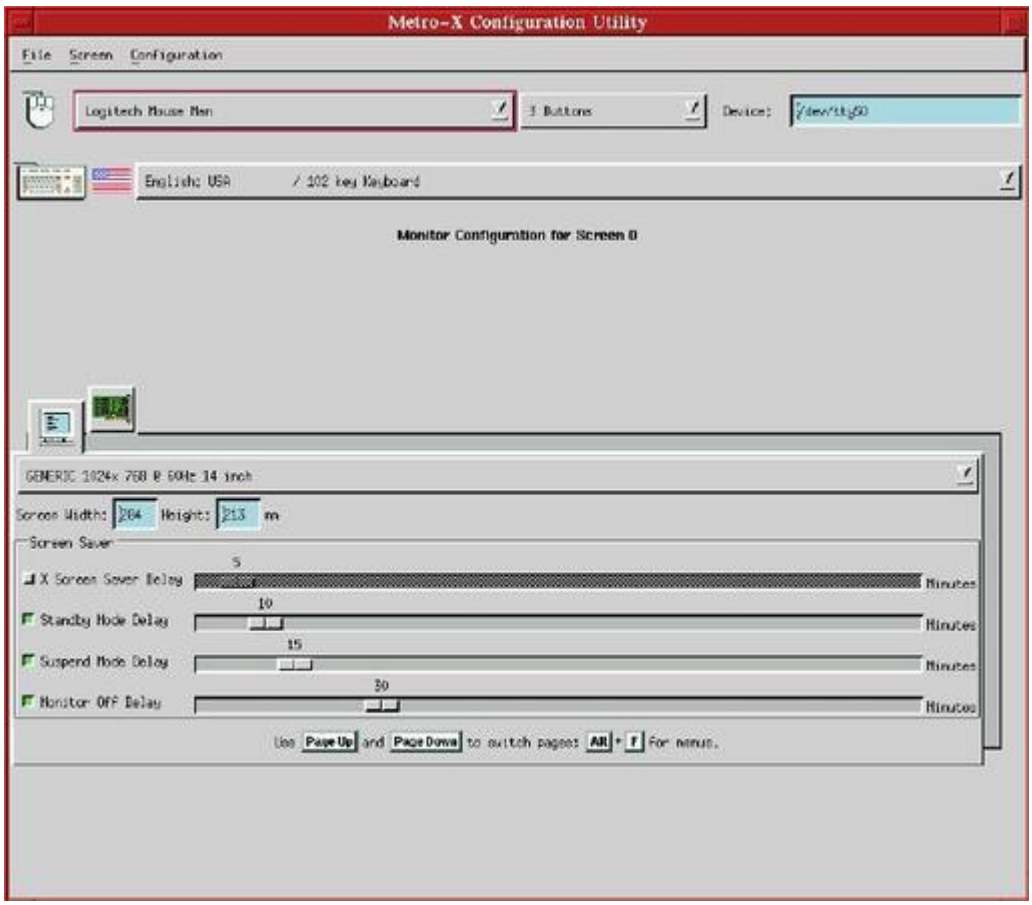


Figure 2. Metro-X Configuration Utility

Metro-X comes as a cpio archive spread across three floppy disks, or it can be downloaded via FTP from the Metro Link archive. Installing the software involved moving the tar file from the floppies via cpio, and then running a small installation script which installs the files and begins the configuration.

Metro-X brings up the X server in a low resolution to allow a GUI interface, even on an unconfigured machine (see Figure 2). Clicking on the different icons exposes options for different pieces of hardware. Several mouse clicks, and I had things configured. There is also a curses-based setup utility that is very similar to Accelerated X's setup utility in case the X server isn't available.

So Which Server is Better?

Before we get to The Chart, I did notice one interesting thing while I was playing around with the two servers. If /usr/X11R6/bin/X is a soft link to an XFree86 server, such as XF86_SVGA, AccelX will move the XFree86 server to X.LINUX. If, however, /usr/X11R6/bin/X is linked to, oh, say, Xmetro (the Metro-X server), then AccelX will turn Xmetro into a link to Xaccel, destroying the Xmetro server. Hmmmm.

Variable	Metro-X	Accelerated X	
Graphics card Support	38 manufacturers	51 manufacturers	
multi-head	up to 4 displays	Additional Cost add-on	
Hot-Keys	Yes	Yes	
Configuration	GUI and curses	curses only	
System Requirements	8MB RAM 12 MB HD space	4MB RAM, 3 MB min HD space	
Require XFree installed	Yes	Yes	
touch-screen	Yes	No	
Documentation	printed, web	printed, web	

Price	$99	$99.95	

I am currently running Metro-X. I have two friends who swear by X Inside. Which server is better sort of depends on your needs. Metro-X has more support for commercial applications, including multi-head display and touch-screen capabilities. Accelerated X supports more graphics cards, making it more likely to suit your hardware.

Bottom line? Check your hardware against both lists, and if both servers will meet your needs, flip a coin—they're both cheaper than a day spent calculating modelines.

Accelerated X Update

The mouse trouble noted in our review of Accelerated X—the mouse flinging applications off the desktop and causing a disruptions to the server connection—have been resolved.

Linux kernel version 2.0.0 introduced a problem with the Async mouse feature of Accelerated X. (This feature allows the mouse to move around the screen even when the X server is busy doing something else—a usability trick Microsoft has been using for a while on their Windows product.) X Inside notified the Linux kernel team and worked with them to correct the problem. With release 2.0.26 of the kernel, the error was resolved.

Jon Gross is a scuba diver, a marine biologist, a writer, a cyclist, a sysadmin, and works for Seattle Software Labs Inc. In his “spare” time, he likes to sleep and cook (not concurrently). He can be reached at jong@seaotter.com.

Archive Index Issue Table of Contents

Advanced search

Programming Perl

Phil Hughes

Issue #35, March 1997

If Perl is your first programming language, you will most likely want to start somewhere else—this book could lead to hopeless confusion.

- ISBN: 1-56592-149-6
- Publisher: O'Reilly & Associates
- Price: $39.95
- Authors: Larry Wall, Tom Christiansen, Randal Schwartz
- Reviewer: Phil Hughes

Programming Perl is the second edition, updated to cover Perl 5, of “The Camel Book”--the one with a camel on the cover.

This book is not a tutorial. If Perl is your first programming language, you will most likely want to start somewhere else—this book could lead to hopeless confusion.

Rather than the newbie, this book is intended for someone who is at least very familiar with Unix utilities, such as awk and sed, and even better, someone with a language such as C under his belt. In fact, C programmers will see similarities between this book and “K & R”.

Perl Programming begins with an overview of basic concepts. After that, get ready for serious work. The second chapter, titled “The Gory Details”, is exactly that—it covers the whole language in about 100 pages.

The next chapter, which is also about 100 pages, describes the functions available to the Perl programmer. C programmers would think of this as the library. Included for each function is its name, its arguments and a textual description of what it does along with appropriate examples.

The Perl language primarily deals in simple, flat data structures, but this approach doesn't work in every case. Chapter Four describes how to deal with more complicated data by using references and nested data structures.

At this point, less than half way through the book, you have seen all of the language and all the built-in functions. However, if you want to become a truly proficient Perl programmer, there's a lot more to learn; e.g., how to produce good modular code. The book therefore goes on to cover modules and packages as well as interaction with the shell and other languages. This section sets the groundwork for the standard Perl library, which is a set of Perl code included with each Perl distribution, the subject of discussion for 150 pages of the book.

The final two chapters cover debugging and error messages. In addition to how to use the debugger, common mistakes, efficiency and style are covered.

Let me note the humor content—a surprising feature in what should be a dry technical manual on a language—which is generally manifest in explanations of some obtuse detail or presentation of background information. While some might find this distracting, I found it a pleasant diversion from a lot of otherwise necessarily dry material. If a humorous bit is long, it is generally in a footnote that can easily be skipped. My only criticism of this book is that it seems to make Perl seem more complicated than it is. It is hard to put my finger on exactly why I feel this way, but perhaps it is because the book tends to tell you everything about a particular command or capability at once. This is just how the book is organized, and it makes sense for a reference work. In other words, I can't see an alternative presentation format that might improve the book as a tutorial that would not destroy the reference value.

This book is well-written, clear and accurate. While there are close to 650 pages, it contains no fluff. While the Perl language is fairly simple, its real power is in its functions and libraries, which are covered thoroughly. A two-word description of this book is “comprehensive reference”. If you are going to be programming with Perl, this book should be in your library.

Phil Hughes is the Publisher of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Perl 5 by Example

Sid Wentworth

Issue #35, March 1997

There are a large number of simple mistakes in the text of the book that may confuse the novice.

- Author: David Medinets
- Publisher: Que
- Price: $39.99 (includes CD-ROM)
- ISBN: 0-7897-0866-3
- Reviewer: Sid Wentworth

Now that Perl 5 has become the de facto standard version of Perl, publishers are getting their books to market as quickly as possible. In the case of *Perl 5 by Example*, it got there too quickly. Since I have C programming experience and have dabbled with Perl, I was looking for a book that would get me up to speed in a hurry, and learning by example seemed a good way to go.

Perl 5 by Example divides the learning of Perl into four sections: basic, intermediate, advanced and “Perl and the Internet”. The CD included with the book contains the Perl interpreter, all the examples from the book, the book itself, and two other Que books on Perl in HTML format. Each chapter contains review questions and exercises.

This approach is valid, but this book doesn't do a good job of implementation. There are a large number of simple mistakes in the text of the book that may confuse the novice. Examples of errors include spaces in the code that don't match the displayed results, the use, without explanation, of an arrow character to indicate continuation of text and the use of an incorrect technical word—“function” is used where “variable” is correct. There is also the tendency to make poor typographical decisions that could also confuse the novice, e.g., the use of different length hyphens to mean the same thing within a particular

chart or the use of italic font in titles that make operator sequences like || appear as //.

Even though I was somewhat unhappy with the book itself, I went on to take a look at the CD. It unfortunately suffers from the same problems. For example, there are links to example files, but the files are not located where the links point.

From my point of view, another drawback is that the book is clearly written for a non-Unix person—maybe less that a non-Unix person. For example, explaining the case sensitivity of variable names seems strange—what language doesn't do this?

If the typographical shortcomings and general errors were corrected, *Perl 5 by Example* would be worth serious consideration—particularly if you were working with Perl on a Microsoft-based platform. For a Linux (or Unix) user, I would recommend looking for another book.

Sid Wentworth lives in Uzbekistan, where he divides his time between UUCP hacking and raising yaks.

Archive Index Issue Table of Contents

Advanced search

Writing Man Pages in HTML

Michael Hamilton

Issue #35, March 1997

HTML is a cool way to look at the Linux man pages. Here's how to do it.

The Web invasion of the Internet continues at a breakneck pace. In the space of just a few years HTML has become one of the most widely supported document formats. Currently, there's gold-fever driven effort by a cast of thousands to reformat older sources of information for presentation on the Web. The required technology is relatively simple to understand, and is inexpensive to assemble. Linux is an ideal development and delivery environment: low cost, reliable, and necessary software is included in several competing Linux distributions.

In this article I'll discuss a solution I've written for serving up old documents in the new medium—automated translation. The documents to be converted are the Unix man pages. Man pages are highly structured documents in which major headings and references to other documents, are easily recognized. The files making up the man page system are already organized into a rigid set of hierarchies using a formalized naming system; thus, it is easy to deliver the entire existing man system and document format via the Web without reorganization of content or overall structure. I've assembled some pieces of software that translate the old Unix manual format into HTML while preserving the old style and organization. The technologies present in the Web allowed further enhancements: documents can be cross-linked; alternate forms of indices can be automatically generated; and full text searches are possible.

I've called the package I've assembled vh-man2html. It is designed to be activated as a set of CGI (Common Gateway Interface) scripts from a web server —the same technology that drives HTML forms. This means that vh-man2html can be used to serve man pages to other hosts on your LAN or on the Internet. The principle component of vh-man2html is Richard Verhoeven's man2html translator, augmented by several scripts to generate indexes and facilitate searches. vh-man2html also has some supporting scripts that allow you to

drive Netscape from the Unix command line, so that vh-man2html can actually replace “man” on systems that include Netscape.

What Does It Look Like?

If you have access to the Web, you can see vh-man2html in action at:

```
http://www.caldera.com/cgi-bin/man2html
```

where the man pages for Caldera 1.0 are available on-line.

Figure 1: Main vh-man2html Web Page

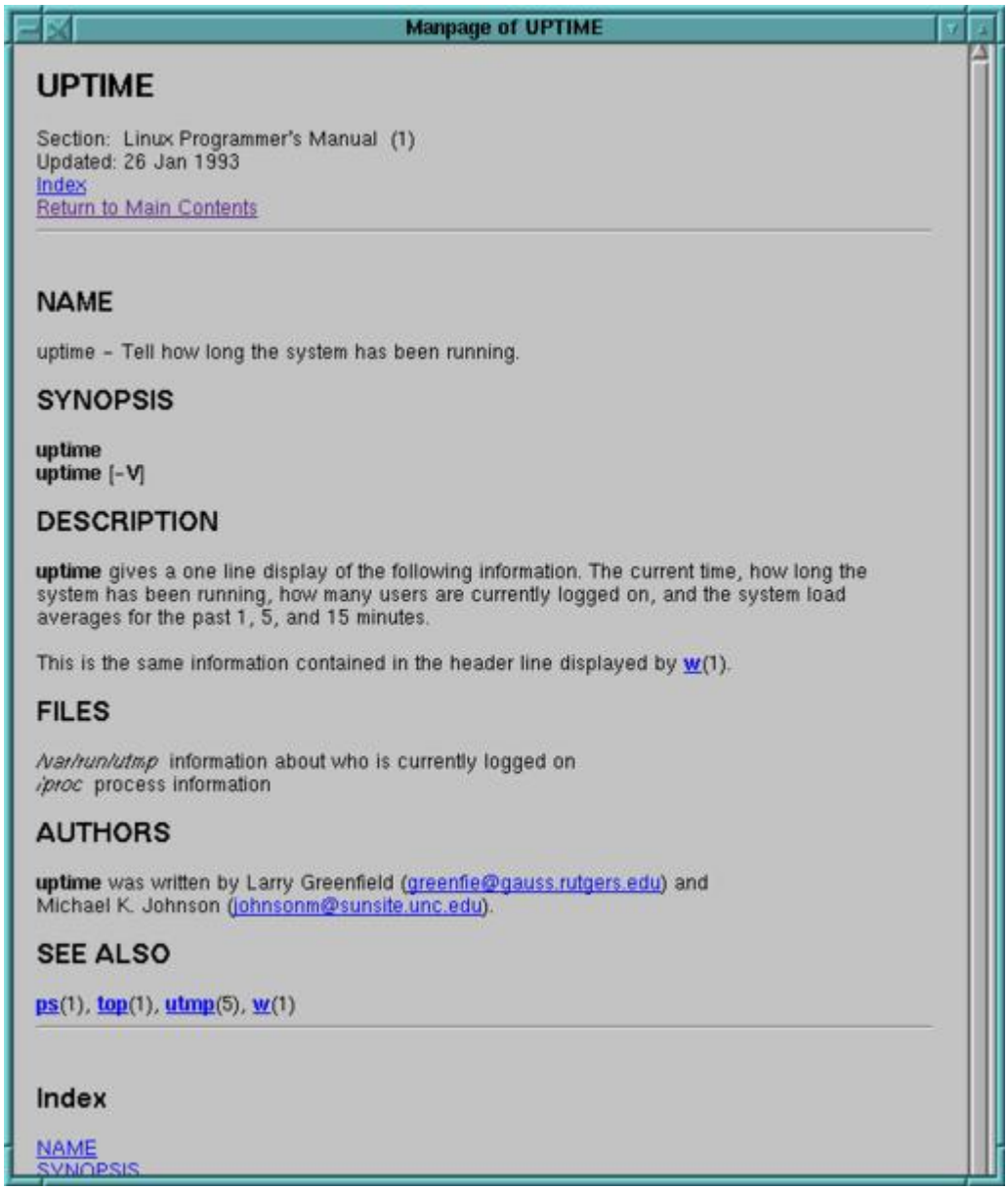


Figure 2: Man Page Generated by vh-man2html

Figure 3: Portion of Name-Description Index

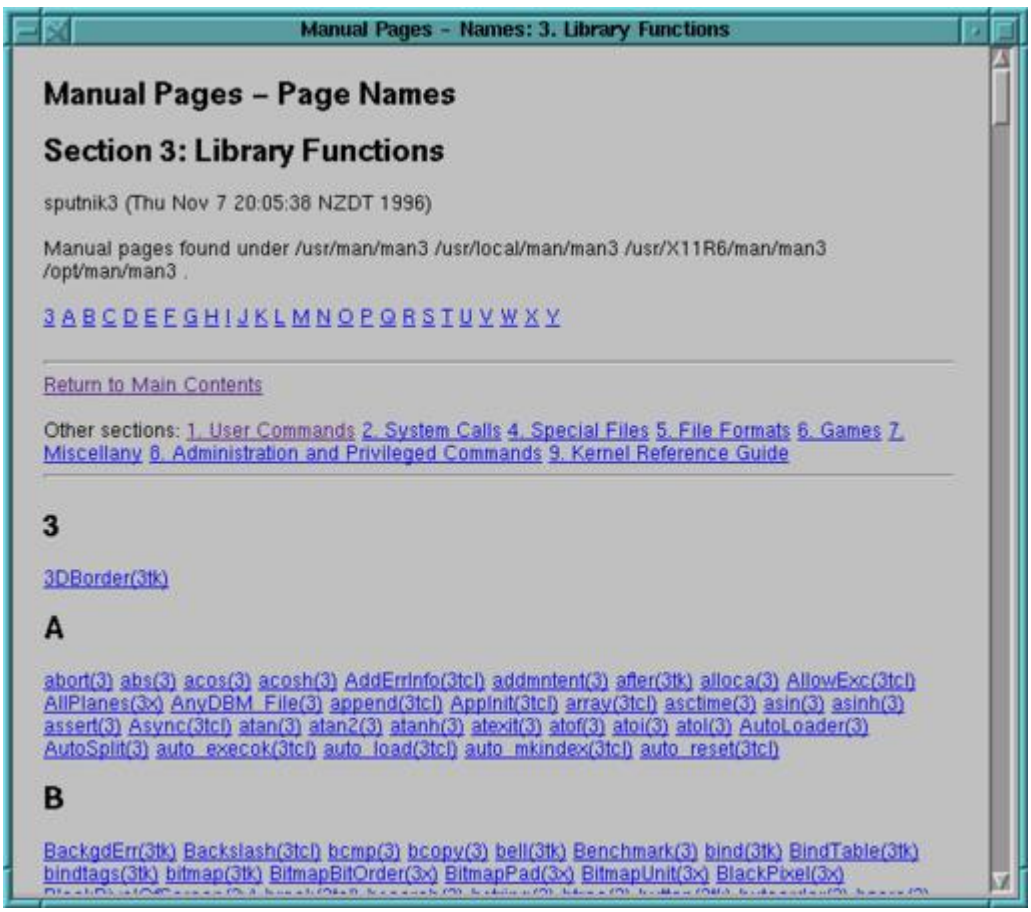


Figure 4: Portion of Name-Only Index

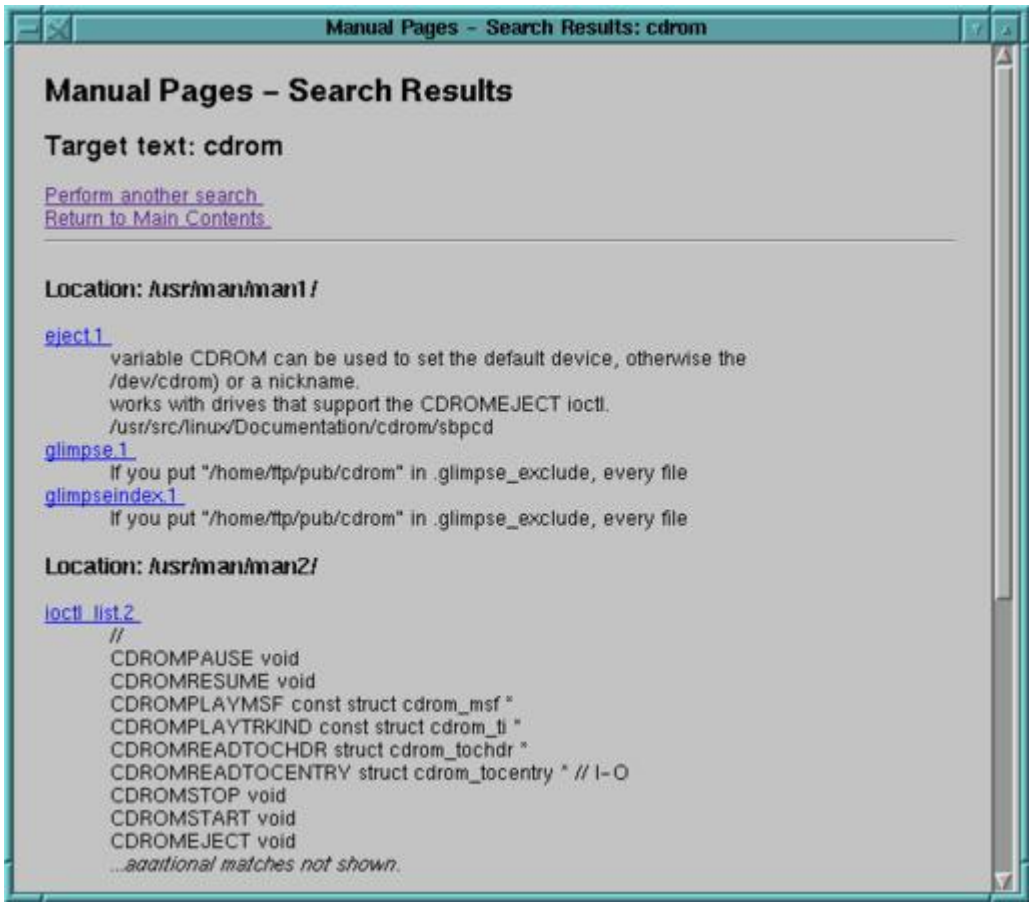


Figure 5: Full-Text Search Result

Figure 1 shows the main vh-man2html web page which provides direct access to individual man pages or to three different kinds of indices: name-description, name-only and full text search. In the main window you can enter the man page name, man page name and section number, or narrow things down even further by specifying the hierarchy or full path name.

Figure 2 shows a man page generated by the man2html converter. The converter has translated the man formatting tags to approximately equivalent HTML tags. It has also created an HTTP reference and links for any references to other man pages. The converter also generates a subject heading index, which is useful when reading larger man pages. Text highlighting and font changes are correctly translated, and if tables are present, they will be translated into HTML tables. At this time, man2html doesn't translate eqn described equations, but since very few man pages use eqn, this is not a major drawback.

Figure 3 shows part of a name-description index for section 1 of the man pages. The index includes an alphabetic sub-index and links into the other sections of the manual. The name-only index in Figure 4 is similar to the name-description index, except that it is more compact.

Figure 5 shows the result of a full text search for pages containing a reference to “cdrom”. Links are generated to each page matched.

These figures illustrate one of the key advantages of serving up the man pages via the Web: a variety of access paths can be presented in an integrated form—a one-stop-shop interface.

Obtaining and Installing vh-man2html

My development platform is Caldera 1.0, with a Red Hat 3.0.3 upgrade. If you don't have a Red Hat-based system, you can still successfully employ vh-man2html. If your system uses an unformatted man page source and you run an HTTP daemon, vh-man2html should still work; however, you may need to reconfigure it and rebuild the binaries to match your own setup.

Don't let having to run the HTTP daemon dissuade you. You can handle the security aspects of this process by restricting the HTTP daemon to serving your own host. I use the Apache HTTP daemon shipped with Caldera and Red Hat, so I just adjust the appropriate lines of my system's /etc/httpd/conf/access.conf file to prevent access from outside of my home network:

```
<Limit GET>
order allow,deny
allow from .pac.gen.nz
deny from all
</Limit>
```

(You can also specify that access should be restricted to a specific IP address.) Additionally, my system is configured with kernel firewalling, which provides an additional layer of protection.

The performance aspects of running an HTTP daemon are minimal. Most of the time it is idle—if other jobs need the memory the daemon is occupying, the kernel just migrates it to swap. To minimize the amount of startup activity and the total memory consumed, I reduced the number of spare daemons to one by editing /etc/httpd/conf/httpd.conf and changing the following:

```
MinSpareServers 1
MaxSpareServers 1
StartServers 1
```

This seems fine for my home network, where at most two users will be active at any one time.

vh-man2html is available in Red Hat package format in both source and i386 ELF-binary form from the following locations:

```
ftp://ftp.caldera.com/pub/contrib/RPMS\
        /vh-man2html-1.5-1.src.rpm
```

```
ftp://ftp.redhat.com/pub/contrib/SRPMS\
        /vh-man2html-1.5-1.src.rpm
ftp://ftp.caldera.com/pub/contrib/RPMS\
        /vh-man2html-1.5-2.i386.rpm
ftp://ftp.redhat.com/pub/contrib/RPMS\
        /vh-man2html-1.5-2.i386.rpm
```

Note that ftp.redhat.com is mirrored at ftp.caldera.com.

Also, a source tar file with ELF binaries is available from:

```
ftp://sunsite.unc.edu/pub/Linux/system\
        /Manual-pagers/vh-man2html-1.5.tar.gz
```

Addtionally, Christoph Lameter, clameter@waterf.org, has modified vh-man2html for the Linux Debian Distribution man pages. His version is available as the man2html package in the doc directory of any Debian archive.

The rpm version will install correctly in any post-2.0.1 Red Hat-based system (including Caldera). Running the following command when logged in as root will install the binary rpm:

```
rpm -i vh-man2html-1.5-2.i386.rpm
```

After installing it you can test it by firing up your web browser and using the following URL:

```
http://localhost/cgi-bin/man2html
```

Provided you haven't disabled your HTTP daemon, this should bring up a starter screen, where you can enter the name of a man page or follow the links to various man index pages.

You can use you browser to save this page as a bookmark. If you feel comfortable editing HTML files, you can insert it in the master document for your own system. In my case I edited my system's top-level document:

```
/usr/doc/HTML/calderadoc/caldera.html
```

and added the following lines at an appropriate point in the document:

```
<H3>
<A HREF="http://localhost/cgi-bin/man2html">
<IMG SRC="book2.gif">
Linux Manual Pages
</A>
</H3>
```

Red Hat users would edit:

```
/usr/doc/HTML/index.html
```

and add the following to the list of available documents:

```
<LI><A HREF="http://localhost/cgi-bin/man2html">
Linux Manual Pages</A>
<P>
```

vh-man2html makes use of some of the files in your existing man installation. It uses the “whatis” files which are used by the Unix “man -k” command as the name-description listing. These files are built by the makewhatis command. Caldera and Red Hat systems normally build the whatis files early every morning. If these files have never been run (perhaps because you turn your machine off at night), you can build them by logging in as root user and entering:

```
/usr/sbin/makewhatis -w
```

Be warned that the standard makewhatis in Caldera 1.0 takes about 30 minutes on my 486DX2-66. I have a modified version of makewhatis that does exactly the same job in only 1.5 minutes. My modified version is now available as part of man-1.4g in both rpm and tar format from:

```
ftp://ftp.redhat.com/redhat-3.0.3/i386\
        /updates/RPMS/man-1.4g-1.i386.rpm
ftp://sunsite.unc.edu/pub/Linux/system/\
        Manual-pagers/man-1.4g.tar.gz
```

Since the traditional Unix man program doesn't provide for searching the full text of the manual pages, I wanted to add this ability to vh-man2html. Enter Glimpse, a freely available program created by Udi Manber and Burra Gopal, Department of Computer Science, University of Arizona, and Sun Wu, the National Chung-Cheng University, Taiwan. Glimpse is a text file indexing and search system that achieves fast search speeds by using precomputed indices. Indexing is typically scheduled for the wee small hours of the morning, when it won't impact users.

To use the Glimpse full text searching, you must install the program Glimpse in /usr/bin. Red Hat rpm users can get Glimpse from:

```
ftp://ftp.redhat.com/pub/non-free\
        /Glimpse-3.0-1.i386.rpm
```

The Glimpse home ftp site is:

```
ftp://ftp.cs.arizona.edu/Glimpse/
```

where the latest source and prebuilt binaries (including Linux) in tar format can be found. Note that Glimpse is not freely redistributable for commercial use. I'd be very interested in hearing about any less restrictive alternatives. Having installed Glimpse, you will need to build a Glimpse index. vh-man2html expects this index to be located in /var/man2html. Building the index doesn't take very long—about three minutes on my machine. As root enter:

```
/usr/bin/Glimpseindex -H /var/man2html \
        /usr/man/man* /usr/X11R6/man/man*\
```

```
          /usr/local/man/man* /opt/man/man*
chmod +r /var/man2html/.Glimpse*
```

On Red Hat this could be set up as a cron job in /etc/crontab, e.g., (the following must be all on one line):

```
21 04 * * 1 root /usr/bin/Glimpseindex -H
   /var/man2html /usr/man/man* /usr/X11R6/man/man*
   /usr/local/man/man* /opt/man/man*i;
   chmod +r /var/man2html/.Glimpse*
```

If you don't wish to use Glimpse, you can edit the man.html file that the package installs in /home/http/html/man.html, and remove the references to full text searches and Glimpse. This file can also be edited to include any local instructions you wish.

If you're building vh-man2html from source, you will have to manually un-tar it and change the Makefile to point to your desired installation directories before issuing a make install. You can also use the rpm2cpio utility to extract a CPIO archive from the rpm, in which case you could read the package spec file to figure out where to put things.

If you don't want to use an HTTP daemon and you know a little C, you might consider using the scripts and C program to pre-translate and pre-index all your man pages. Then they can be referred to directly without an HTTP daemon to invoke conversions on demand.

Technical Overview

This section will cover a few of the implementation details of vh-man2html. It's very brief and is really intended to point out that CGI scripting is something that anyone with a little programming knowledge can do with success.

Without getting into a tutorial on CGI scripting, a CGI script is a program executed by the remote HTTP daemon (i.e., web server). A web browser can cause a remote web server to run a CGI program when you follow an HTTP link that matches its name. For example, pointing a web browser at:

```
http://www.caldera.com/cgi-bin/man2html
```

executes cgi-bin/man2html on Caldera's web server. The CGI programs that a web server is prepared to run are usually restricted to those found in cgi-bin directories on the server.

The CGI script can return output to the remote caller by writing a document to its standard output. The start of the output document must contain a small text header describing its contents. In the case of man2html the content returned is an HTML page. Listing 1 shows the HTML output from the man page to HTML converter; the header line is:

```
content-type: text/html
```

and the rest of the document is normal HTML, which consists of text marked up with HTML tags. With some web browsers, you can use options like Netscape's “View Document Source” to inspect this HTML source.

A script may create an HTML page that contains further references to other CGI scripts. In Listing 1 the following reference returns the reader to the main vh-man2html contents page:

```
<A HREF="http:/cgi-bin/man2html">Return to Main
Contents</A>
```

A CGI script receives input that may have been embedded in the original reference or that have may have been added as a result of user input. For example, in Listing 1, the “SEE ALSO” section directs the cgi-bin/man2html program to return the HTML for a specific manual page:

```
<A HREF="http:/cgi-bin/man2html?man1/from.1l">
from</A>
```

In this case the HTTP reference is supplied with a single parameter “man1/from.1l”--the name of a man page. The start of the parameter list is delimited by a “?”. If there were more than one argument, they would be separated by “+” signs (and there are conventions for how to pass special characters such as “+” and “?” as parameters). The CGI program won't see any the of delimiting characters; it just receives the parameters as arguments in its normal argument list (or optionally via standard input). This means the CGI script doesn't have to concern itself with how its input got delivered over the network, it simply receives it in the form of command-line arguments, standard input, plus a variety of environment variables.

In addition to clicking on references, the user can also enter data into input fields. The simplest way for a CGI program to introduce an input field onto a form is to include the tag <ISINDEX> in the HTML it generates. This results in a single input field, such as in Figure 1. If the user enters anything in the input field and presses **return**, the server will re-run the CGI program, passing it the input via the parameter passing conventions we've just discussed. You can also create HTML forms, but I'm not going to discuss them here.

By generating the kinds of HTML references presented above, CGI programs can perform complex interactions with the remote user. The beauty of all this is that, to get started, the only skill you need is the ability to write fairly simple code in a language of your choosing. You need to know how to process command-line arguments and write to standard output. The rest of the knowledge you need can be gotten for free from Web documents or from any

one of a number of books on HTML and CGI. CGI is a client-server that actually works. Heavy duty CGI programming languages such as Python and Perl have tools and libraries to assist you with the task.

I should also mention the issue of security. If your HTTP daemon is accessible by potentially hostile users, your CGI scripts could provide an avenue for them to attack you. Hostile users might try to supply malicious parameters to your CGI scripts. For example, by using special shell characters such as back quotes and semicolons, they might be able to get the script to execute arbitrary commands. The only way to prevent this is to carefully examine all input parameters for anything suspicious. For example, vh-man2html can be passed the full file name of a man page; however, it doesn't just accept and return any file name it is passed—it accepts only those filenames present within the man hierarchy. The program also makes sure the file name does not contain relative references such as “..” (the parent directory), and removes any suspect characters such as back quotes that might be used to embed commands in the parameter list. In languages like C, where memory bounds checking is lacking, the length of the input arguments should be constrained to fit within the space allocated for them. Otherwise the caller may be able to write beyond the allocated space into other data and change the behavior of the program to his/her advantage (e.g., change a command the program executes from gzip to rm). To help check that long input parameters wouldn't threaten vh-man2html's integrity, I borrowed some time on an SGI box and built vh-man2html with Parasoft's Insight bounds checker. Insight pre-processes a C or C++ program adding array bounds checking, memory leak detection and many other checks. One of the reasons I'm mentioning Insight is that Parasoft's Web site, http://www.parasoft.com/, lists Linux as a supported platform.

vh-man2html includes four CGI programs. They all generate interdependent HTTP references to each other.

Man page to HTML translation is handled by the man2html C program. The Unix man pages are marked with man or BSD mandoc tags which are nroff/troff macros. The bulk of the program is a series of large case statements and table lookups that attempt to cope with all the possible macros.

Listing 2 shows a typical nroff/troff marked up manual page that is using the man macro package. The macros use a full-stop, i.e. a period, as a lead-in to a one or two character macro name. troff/nroff uses two character macro names—apparently they fit nicely into the 16-bit word size of the old Unix platforms such as the PDP11 (at least that's what I was told)--a trick which man2html.c still utilizes. Some of the macros can be directly translated to appropriated HTML tags; for example, lines beginning with “.SH” Section Headings are directly translated to HTML <H1> headings.

Many troff tags limit their arguments and effects to just one line and have no corresponding end tag—where as many of the equivalent HTML constructs also require an end tag. For example, the text following a troff ".SH" section heading tag needs to be enclosed in a pair of HTML heading level 1 tags, e.g., "<H1>text</H1>". Other troff tags with a larger scope, such as many kinds of lists, have both begin and end tags, which makes translation to HTML very easy.

One tricky issue is dealing with multiple troff tags on one line; for example, tags that imply bracketing of following text or font changes. In order to correctly place bracketing, the translator can work recursively within a line. For example, the BSD mandoc sequence for an command option called **-b** with an argument called *bcc-addr* is expressed in troff as:

```
.Op Fl b Ar bcc-addr
```

which indicates the reader should see:

```
[ -
```

where **b** is in bold and *bcc-addr* is in italics. The corresponding HTML is:

```
[ -<B>b</B> <I>bcc-addr</I> ]
```

By using recursion on hitting the Op tag, we can get the square brackets on the beginning and end of the entire line.

There are some troff tags whose effect is terminated by tags of equal and higher rank; in these cases, the translator must remember its context and generate any necessary terminating HTML. Nested lists are also possible. In these situations man2html has to maintain a stack of outstanding nestings that have to be completed when a new equal or higher element is encountered.

I admire Richard's dedication in methodically building up translations of all of the tags. Adding in the BSD mandoc tags proved to be a painful experience, and in the end, the only way to get it right was to convert every BSD mandoc page I could find and pipe the output to weblint (an excellent HTML checker). For example, in tcsh/csh:

```
foreach i ( `egrep -l '^\.Bl' /usr/man/man1/* \
        /usr/man/man8/*` )
/home/httpd/cgi-bin/man2html $i > tmp/`basename $i`
end
weblint tmp/*
```

If you want to sample the spectrum of mandoc translation, look at any of the pages the above egrep locates—telnet, lpc and mail are good examples.

man2html.c also has to do minor translation fix ups, such as translating quotation marks and other special punctuation into HTML special characters.

In the end, to test the sturdiness of the translator, I converted every man page I have:

```
find /usr/man/man* -name '*.[0-9]' \
  -printf "echo %p; /home/httpd/cgi-bin/man2html\
        %p | weblint -x netscape -\n" | sh \
        |& tee weblint.log
```

These tests proved quite useful in exposing bugs.

The program also has to navigate the man directory hierarchies and generate lists of references to pages that might be relevant (e.g. a page with the same name might be present in multiple man hierarchies). The list of man hierarchies to be consulted is read from /etc/man.config, which is the standard configuration file for the man-1.4 package that ships with Redhat and Caldera. This configuration file is also consulted for details on how to process man pages that have been compressed with gzip or other compression programs.

man2html could have easily been written in Python or Perl, but you can't beat C for speed. man2html is fast enough on my 486 that I didn't think caching its output was worthwhile—each page is just regenerated on demand. However, if I was going to provide man pages from a server for a large number of high frequency users, I would probably pre-generate all the man pages as a static document set.

Two awk scripts, manwhatis and mansec, generate name-title and name only indexes for man sections and cache them in /var/man2html. manwhatis locates and translates whatis files into the desired section index, which it caches in /var/man2html. It rebuilds the cache if any whatis file has been updated since the cached version was generated. The script divides the whatis file alphabetically and constructs an alphabetic index to the HTML document, so that the the user can quickly jump to the section of the alphabet they're interested in.

mansec traverses the man hierarchy to build up a list of names; it rebuilds its cache if any of the directories in the hierarchies have been updated. mansec has to use the sort command to get the names it finds into alphabetical order. It also builds an alphabetic quick index just like manwhatis.

Both manwhatis and mansec accept an argument that indicates which section to index. They have to check the argument for anything potentially malicious and return a document containing an error message if they find anything they weren't expecting:

```
section = ARGV[1];           # must be 0-9 or all.
if (section !~ /^[0-9]$/ && section != "all") {
 print "Content-type: text/html\n\n";
 print "<head>";
 print "<title>Manual - Illegal section</title>";
 print "<body>";
 print "Illegal section number '" section "'." ;
 print "Must be 0..9 or all";
 print "</body>";
 exit;
}
```

The mansearch script is an awk script front end to the Glimpse search utility. It accepts user input, which it passes onto Glimpse, so I had to be careful to include code to check the input for safety before invoking Glimpse. This basically means excluding any shell special characters or making sure they can't do anything by quoting them appropriately. For example, in awk we can silently ignore any characters that we aren't willing to accept:

```
# Substitute "" for any char not in A-Za-z0-9
# space.
string = gsub(/[^A-Za-z0-9 ]/, "", string);
I chose awk over Python and Perl mainly because it is small,
widely available and adequate for the task. Note that I'm using the post
1985 "new awk". For larger, more complex CGI scripts I'd probably use
Python (if I had to start again without Richard's work, I think
man2html would be a Python script).
In order to make vh-man2html usable remotely, I changed man2html and my
scripts to generate HTTP references that were relative to the current server.
For example, I used:
```

```
<A HREF="http:/cgi-bin/man2html">Return to Main
Contents</A>
```

rather than

```
<A HREF="http://localhost/cgi-bin/man2html">Return to Main
Contents</A>
```

which works fine except for “redirects”. A redirect is a small document output by a CGI script. This is an example redirect:

```
Location: http://sputnik3/cgi-bin\
        /man2html/usr/man/man1/message.1
```

A redirect has no context, so the host has to be specified. man2html generates redirects when a user enters an approximate name such as “message 1”. The redirect *corrects* this to a full reference such as the one above. The server name is obtained from one of the many environment variables that an HTTP server normally sets before invoking a CGI script.

Interfacing with Netscape

Recently I wanted to be able to issue man page requests from the command line to an already running Netscape browser—I wanted to replace the command-line man with Netscape. As you're probably aware, launching a large application such as Netscape (or emacs) is prohibitively expensive, so my preference is to communicate with an already running Netscape rather than

start a new one for each request. Instructions for driving an already running Netscape from the command line can be found at:

```
http://www.mcom.com/newsref/std/x-remote.html
```

You can either use a new Netscape to pass commands to an existing one—it exits after passing them on without launching any screens—or you can use a smaller sample C program (see the HTTP reference) to do the same job. The following bash script fragment will use either (preferring the smaller utility if it's available):

```
function nsfunc () {
 # If this user is running netscape - talk to it
 if ( /bin/ps -xc | grep -q 'netscape$' ) ; then
  if [ -x netscape-remote ] ; then
   # Use utility to talk to netscape
   exec netscape-remote -remote "openURL($1,new_window)"
  else
   # Use netscape to talk to netscape
   exec netscape -remote "openURL($1,new_window)"
  fi
 else
  # Start a new netscape.
  netscape $1 &
 fi
}
```

A bash script can call this function as follows:

```
nsfunc "http:/cgi-bin/man2html?who+1"
```

By using similar techniques, systems can base their help browsers on Netscape. Hopefully, similar capabilities will be built into some of the free browsers to provide alternatives (e.g., Lynx is a fast text-based browser).

Summary

Creating vh-man2html was a rewarding spare time effort. The Internet community provided the components and feedback necessary to get it built. It demonstrates one way in which old information can be repackaged for easier access. Linux provided an excellent development platform. All of the necessary components are available in Linux distributions and via ftp.

I'd like to thank Richard Verhoeven for creating his man2html translator and making it available to the Internet community. I'd also like to thank those people who took the time to send me the feedback that assisted with the development of vh-man2html.

Why Would You Want to Do This?

How Did it All Come Together?

References

Michael Hamilton has been working as a freelance Unix C/C++ developer since 1989. He tripped over one of Linus' postings back at the beginning of 1992, and has been hooked ever since. He can be reached at michael@actrix.gen.nz.

Archive Index Issue Table of Contents

Advanced search

A Point About Polygons

Bob Stein

Issue #35, March 1997

An essay on the aesthetics of polygons and algorithms that one might see in a web image map.

Several algorithms exist in the public domain for web servers to determine whether a point is inside a polygon. They are used in the implementation of “image maps”, both of the traditional server-side variety as well as those of the more modern client-side. So who needs one more? Well, the bone this author wishes to respectfully pick is that most of the point-in-polygon code he could find is woefully over-complicated. Being a lover of simplicity and simplification, he just could not leave well enough alone.

The resulting C-language routine has just three **if** statements and no divides. Contrast that with three divides and ten **if** statements in the corresponding routine that's part of the popular Apache web server. Get the Apache distribution and search for **pointinpoly** to see the whole works. The routine from CERN/W3C's httpd is even worse, weighing in at 19 **if** statements! Search for **inside_poly** in their HTImage.c. (The URLs are shown in Table 2.)

Table 1 contrasts five different routines in the public domain for finding out if a point is in a polygon. In all cases, the polygon is specified as an array of X,Y coordinates of the corner points.

Table 1. Comparison of Point-in-Polygon Algorithms

This is a pretty casual analysis of the algorithms. I certainly didn't shy away from showing my **inpoly()** in a good light. For example, **&&** and **||** operators in C are often statements in disguise. I used one of these operators (as did most of the other folks), but that doesn't show up in the table at all. Also, some line counts are inflated slightly by comments and blank lines. But you get a rough idea.

Table 2. Sources of Public Domain Point-in-Polygon Algorithms

A Note of Reason

All judgments have a context, and I should explain mine. The primary prerogative in this article is algorithmic simplicity. This, I confess, has very little to do with the practical needs of the Web. In case I've gotten ahead of myself, a web image map is a way of carving up an image so that clicking in one particular region does one thing, and clicking somewhere else does something else. Web image maps are such a tiny fraction of the work of web servers and browsers that all of the above routines are just fine as they are. Changing from one to another is not going to make any noticeable difference in web performance. And once we're sure it works, who's going to look at the code again for 100 years? Thus I don't have any practical considerations of performance or readability to justify my cause. I'm simply championing the aesthetics of simplicity.

The point I wish to make is that problems are not always what they seem. Sometimes a simple solution exists, but you've got to take a hard look to find it. My buddy Craig had started out by porting **inside_poly()** from W3C, I think it was, for use on our web server. When I saw all the floating point math and **if** statement special cases, I thought there had to be a better way. So Craig and I started from scratch, wrestled the problem to the ground, and came up with a solution containing no floating point math, which is silly for screen pixels, and no math more tedious than multiplication. We also got rid of all the pesky special cases, except for one: polygons with fewer than three sides are excluded. What could be inside a two-sided polygon? Apache's pointinpoly() doesn't even check, and probably makes a big mess with a one-point polygon.

Now, the stated goal is simplicity, not performance, but I did stray from that course on one issue: avoiding divides. Again, performance hardly matters for image map applications, but one day someone might use this algorithm for some kind of 3D hidden surface algorithm or something. Getting rid of the divide may have, in effect, required me to use an additional **if** statement. Anyhow, what all this is leading up to is that Kevin Kenny's algorithm (see Table 1) at 29 lines and two **if** statements is by far the shortest and simplest. But mine is still better in some sense, because mine doesn't need a divide and his does.

Now let's discuss the more popular algorithm for determining whether a point is inside a polygon.

The “Crossing Count” Algorithm

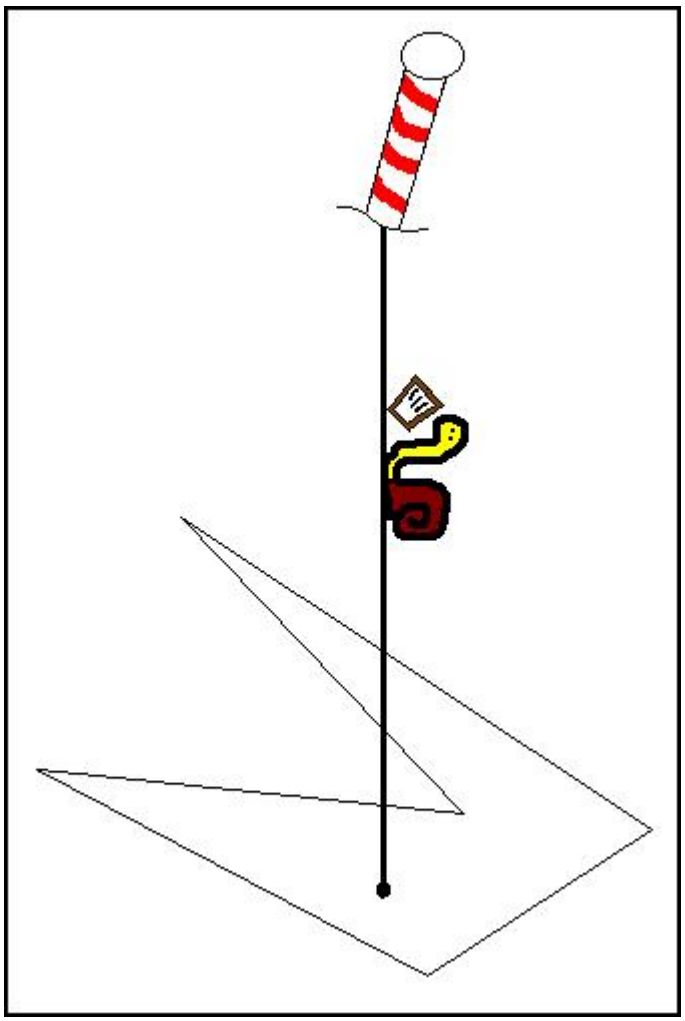


Figure 1. An Odd Number of Polygon Crossings

Imagine you could detect whether a point was in a polygon or not by placing a friendly trained snail at the point and telling him to head for the North Pole. (We're only concerned with image maps, so we exclude polygons that extend to the North Pole, and we ignore Coriolis forces.) You'd equip our intrepid friend in Figure 1 with a snail-sized clipboard and instruct him to tick off each time he crossed an edge of the polygon. He'd call you from the North Pole and report the number of crossings. An even number (including zero) means he started outside the polygon, odd means inside.

This reduces the problem to detecting whether or not line segments intersect. It's even a little better than that, because one of the line segments is simply the positive Y axis. To make that leap, just declare the snail's starting point to be the origin, (0,0), and translate all of the polygon corners so they're relative to that point.

We'll go into the algorithm a little later, but take a look at the finished code in Listing 1. The very picture of simplicity, right? If you haven't checked out the other versions, you really ought to.

Listing 1. inpoly.c

Listing 2. testpoly.c

Testing the Algorithm

The test program (Listing 2) draws a random 40-sided polygon and then picks random points to throw at the **inpoly()** routine. Points the routine says are inside the polygon it draws red, points outside are blue.

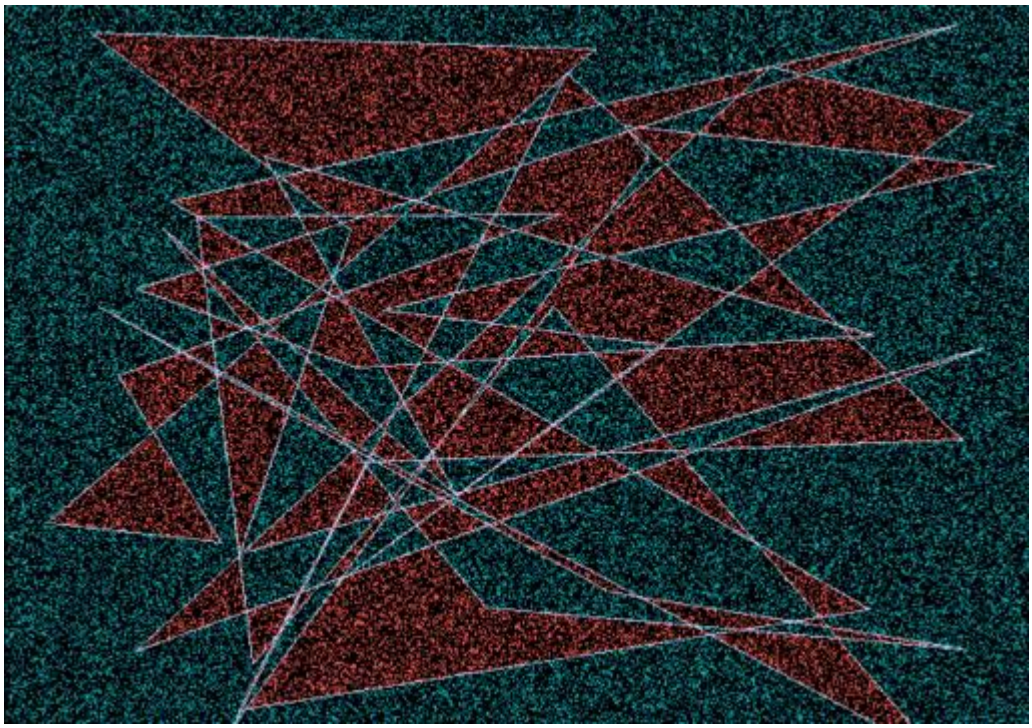


Figure 2. A Random 40-Sided Polygon

All Tripped Up

Our first rendition of **inpoly()** had a subtle flaw which the test program made evident. The full story contains an embarrassing lesson. “It'll work,” we sneered, “We don't need to waste time on a full graphical test. Besides, it'd be too much fun.” After we found out our image maps had leaks, we wrote the test program. Figure 3 shows a close-up of the flaw.

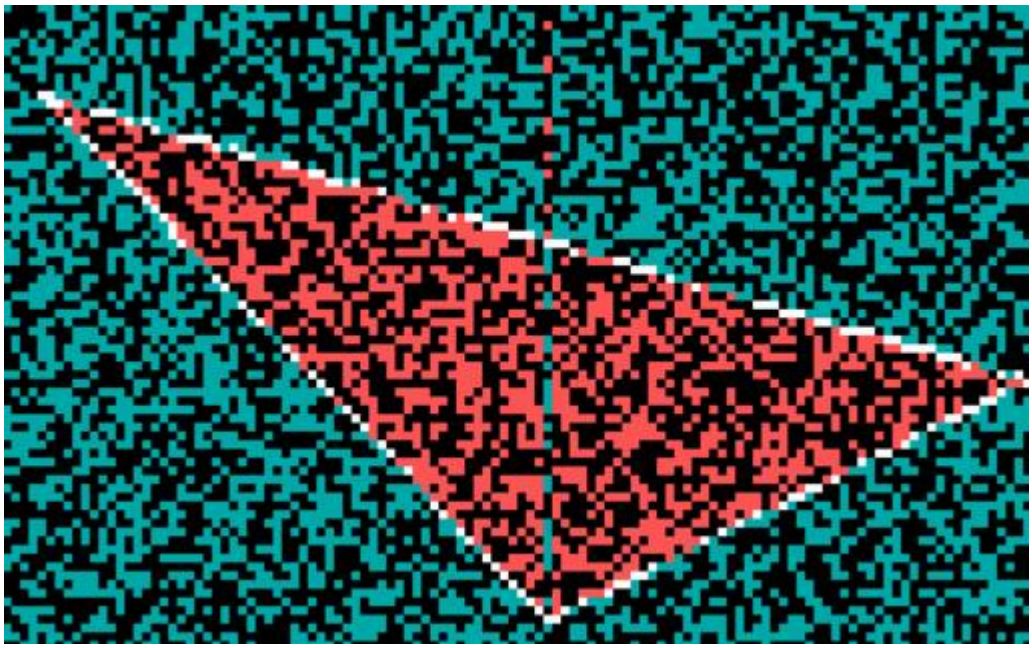


Figure 3. The flaw: The Corner is Counted Twice

Along a vertical line, all the colors are wrong. The flaw turned out to be that when our mindless mollusk crosses the bottom corner, the little hummer was counting the crossing of both edges! After that, he was always exactly wrong—he thought he was in when he was out, and he thought he was out when he was in. The solution must ensure that when our esteemed escargot crosses into the polygon corner, he counts exactly one crossing. Two is no good, and in fact, zero is just as bad—one is what we need. The reason the flaw in the close-up extends up from the corner is that the positive Y axis extends downward in screen coordinates.

I suspect this is a problem unique to the fixed-point world. I'm sure my fellow point-in-polygon smiths have either lucked out or dealt with it somehow. At least, I'd like to think so. (A lie-detector would peg me on that one. This article would be insufferably smug if I had found leaky corners in any of the other algorithms.) In my case, I realized I could not blindly count all crossings of the end point of each of the edges as a crossing. My first thought was to associate each end point with one—and only one—edge. This sounds fair and equitable, but like many things fitting that description, it just plain won't work. A problem turns up when Agent Snail just lightly nicks the corner of a polygon he's not inside at all. That's counted as one crossing, hence the snail report is bunk.

Since I abhor special cases, I sought something that would work in all cases.

Cutting Corners

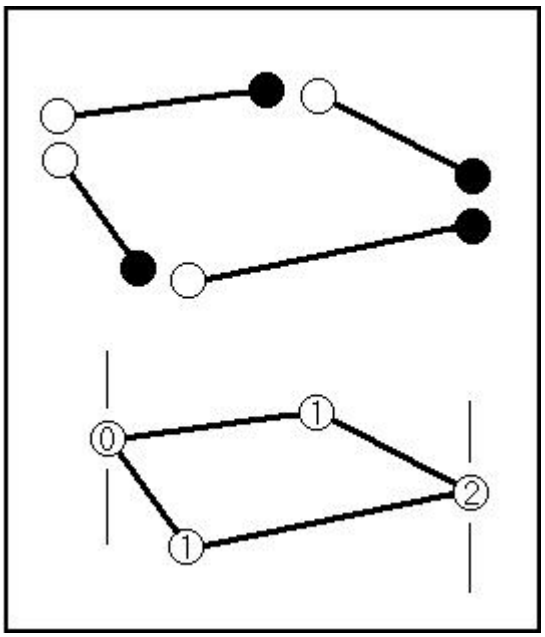


Figure 4. Counting Only the Right End of Each Edge

The scheme for getting our faithful friend to count corner crossings correctly is to always count a crossing of the right end of each edge, but never the left end (right meaning positive X). In the figure, the black circles represent points our snail will count if he crosses; the white circles he won't count. When you put the polygon together, everything ends up the way we want. Nicking the corner means he counts either 0 crossings or two crossings. We don't care which; both are even and our snail knows he's outside. The circles with ones in them represent points counted once if the snail crosses them. This is fine, just like crossing the nearby sides.

It's time to analyze the guts of the **inpoly()** routine in Listing 3. This represents a slight modification of the snail's instructions. He plays a bit of a “she loves me, she loves me not” kind of game rather than counting up the crossings and then reporting whether the total is even or odd. He starts out assuming he's outside, and complements that assumption with each crossing. So much for the **inside=! inside** statement.

Listing 3. The “Guts” of the **inpoly()** Routine

This **if** test happens inside a **for** loop that considers all of the edges of the polygon, one at a time. Each edge is a line segment that stretches between the corners (xold,yold) and (xnew,ynew). We've arranged it so (x1,y1) and (x2,y2) also represent the same edge, but the points are swapped, if necessary, to make it so x1 <= x2.

Now two things must be true for our ever-meticulous snail to count the crossing of this edge. First, the segment must straddle the Y axis (where the right end is counted but the left one is not). Second, straddling has to happen to the north of the snail's starting point. These are exactly the questions determined by the **if** statement's two pieces, on either side of the **&&**.

Now that first expression is a sneaky one, and I confess I might have preferred the less opaque code (x1 < xt && xt <= x2). You can see it does the same thing if you look carefully (very carefully—I was fooled for a while there). But I hate to fix something unless I've already broken it, if you know what I mean.

That north computation is the one I'm proud of because none of my esteemed fellow polygon smiths made one that doesn't need a divide. It does depend on the knowledge that (x2-x1) is positive. Other than that, it's just a transmogrification of that famous y=mx+b equation from high school algebra.

By the way, I've left out the case where an edge line segment stands straight up and down above the snail touchdown point. Such an edge would never be counted by Mr. Snail at all! That's because the == test would always be false, since xnew, xt and xold are all the same value. What's really wild is that's just what we want. In a sense, he's crossing three edges when we only want to count one. It turns out the adjacent line segment crossings are all we're interested in, and the rules already discussed work perfectly for them.

Life on the Edge

By the way, who cares whether the points in an image map along the edge of a polygon are technically inside or outside? As you can see in the close-up, some of the originally white pixels (representing the polygon edge) turned to red, others to blue. If a browsing user clicks on the edge of a region, he may get in, he may not. But being one pixel off is usually not an issue if your screen resolution is greater than 100 x 100. In the **inpoly()** routine, some edges are in, some are out. (I don't mind admitting to a crime after convincing everyone it deserves no punishment.)

Angling for Adders

I haven't discussed the angle-sum method used by Woods Hole Oceanographic Institution for their algorithm written in Matlab. The algorithm needs to compute arc-tangents, so it's mostly just a laboratory curiosity. The idea is that you add up the angles subtended by lines drawn from the target point to each of the corners of the polygon. If the sum is an even multiple of 360 degrees, you're out; odd, you're in. Vaguely familiar? Here's the analogy: You're in a pitch-black room with a very, very long snake all over the floor. This is a particularly rare variety of deep sea snake (Woods Hole knows all about them)

with glow-in-the-dark dots every foot or so. Oh, and he reacts to light by instantly constricting in an iron grip of death. Your question is whether you're standing inside the maze of coils at your feet or outside. You'd like to know before you turn on the light because he gets very annoyed if you step on him.

Face the head of the snake and visually trace his entire body, somehow noting as you do how your feet turn (it's a stretch I know). When you're done, face the head again. Now, if you didn't have to turn around at all, you're safely outside the snake. If you turned around twice in either direction things are fine too. Four times, and you're still OK. If you turned around an odd number of times in either direction, you're meat—no wonder folks tend to use the crossing-count algorithm.

Bob Stein has been a writer/developer at Galacticomm for nine years. He developed the web server that's part of Galacticomm's Worldgroup software which, of course, uses his inpoly.c for image map polygons.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Learning to use the httpd error log to debug CGI programs

Reuven M. Lerner

Issue #35, March 1997

This month, we will examine the error log to see what it contains, look through several errors and how they appear in the error log, and finally consider some ways in which we can use the error log to make debugging easier.

As discussed in my last column (*LJ*, Issue 34, WWW section), debugging CGI programs is often more difficult than debugging their non-CGI counterparts, if only because our programs aren't interacting directly with the user's terminal. We have only one chance to get input from the user—when the program is invoked—and only one chance to send a response back to the user—just before the program terminates, in the HTML that our program creates.

This difficulty is complicated further by the fact that running CGI programs correctly requires that a number of other items be in place. For example, you must set the permission bits correctly, the web server must be configured to serve CGI programs from your directory, and the correct version of Perl (if you are using Perl for your CGI tasks) must be installed. None of this is very difficult, and it's particularly uncomplicated when compared with the full administration of a Unix system, but it's not quite as simple as compiling and executing other, non-CGI programs.

In the last article in this series, we looked at techniques that you can use when things go wrong. But what happens when all of the permissions and directories are set correctly and you still end up getting a mysterious error on your browser?

The answer is that while web servers send a generic error message back to the user's web browser, they also keep an “error log,” a file into which information about each error is placed. This month, we will examine the error log to see what it contains, look through several errors and how they appear in the error log, and finally consider some ways in which we can use the error log to make debugging easier.

Why an Error Log?

As Unix programmers are well aware, programs typically have at their disposal three basic file descriptors: standard input (stdin), standard output (stdout), and standard error (stderr).

Standard input is typically associated with the user's keyboard; when you are entering text in Emacs or similar program, the program is probably collecting your keystrokes via stdin. Standard output, by contrast, typically goes back to your console—either the window into which you are typing or the entire screen.

Standard error is normally sent to the user's console, and thus is often confused with stdout. But the two are completely separate and can be redirected to different places. You can see this in action by trying the command:

```
ls *zz* > zz-files
```

which, on my system, produces the following output on the screen:

```
ls: *zz*: No such file or directory
```

Since there aren't any files containing **zz** in their names, the file **zz-files** is empty. This error message is sent to stderr, which is kept on the screen, while the (empty) list of files is sent to stdout, which we redirected to the file named **zz-files**.

A CGI program still has access to stdin, stdout and stderr, but these files are used somewhat differently as the program isn't running on the user's console. For a CGI program, stdin reflects the contents of an HTML form as submitted via the POST method. Stdout, not surprisingly, is used to send a response, which is typically not in HTML format, to the user. Finally, stderr is redirected to the HTTP server's error log, which is usually kept far away from visitors to the web site.

Therefore, if our program dies unexpectedly, error messages are directed to the httpd error log rather than to our browser. On the one hand, this is probably a good thing—after all, you probably don't want the world at large to know the number and kind of bugs found in your code. On the other hand, this means that you need to look at a separate file, and that you need to log on to the server computer before you can know exactly what has happened to your program.

If you are renting space on a web server run by a web space provider or Internet service provider—or more importantly, if you are looking to rent space from such a provider—make sure that you have access to these logs. You can

certainly write and debug CGI programs without access to the error log, but your life will be infinitely more pleasant if you can get to it.

Looking at the Log

Each HTTP server logs information in a slightly different way. In this column, I will describe the logs produced by Apache, a server derived from NCSA httpd, which was written and distributed by the National Center for Supercomputing Applications (NCSA). You can get a copy of Apache, as well as documentation relating to this excellent server, at http://www.apache.org/.

In my copy of Apache, the error messages are placed in the file with one line per error. Error messages are in the format:

```
[DayOfWeek Month Date Time Year] Message
```

Thus one possible entry in the error log would be:

```
[Mon Dec 2 16:05:50 1996] Coke machine needs refilling
```

As you might have guessed, this means that the server recorded the error at 4:05:50 P.M. on Monday, December 2, 1996.

Your httpd probably records all sorts of things in the error log that you probably wouldn't classify as errors. For example, my system's error log contains the following error message:

```
[Tue Nov 5 17:03:48 1996] access to \
  /home/httpd/cgi-bin failed for ahad-haam, reason: \
  script not found or unable to stat
```

The above message means that the server wasn't able to find a particular file on my web server. That's right—every time someone points his web browser to your system and requests a document that doesn't exist, an error message is recorded. This is actually a useful thing to know, both for security reasons (since it might be nice to learn if people are trying to grab sensitive files via your web server) and in order to help users who often but mistakenly request particular files. For example, if a particular file is often requested under an incorrectly spelled name, you can set up an alias or symbolic link on your system.

As interesting and useful as an error log might be for system administrators, network administrators and web masters, we are going to concentrate on the error log's role when trying to debug CGI programs. For example, try the following:

```
#!/usr/local/bin/perl5 -w
  use strict;      # Check our syntax strictly
```

```
    use diagnostics; # Tell us how to fix mistakes
    use CGI;         # Import the CGI module
    # Create an instance of CGI
    my $query = new CGI;
    # Notice that we are *not* sending a MIME
    # header, which we should always remember
    # to do.
    # Begin the HTML
    print $query->start_html(-title =>
          "Hello, world!)";
    # Print something fairly uninteresting
    print "<P>Hello, World Wide Web!</P>\n";
    # End the HTML
    print $query->end_html();
```

The above program will work just fine if run from the command line, since the only error is a missing MIME header, which is necessary only for CGI programs. But if you try to run it from a browser, you get the dreaded message:

```
Server Error. The server encountered an internal
error or misconfiguration and was unable to complete your request.
```

Not very useful for debugging your program, particularly since running it from the command line revealed no obvious errors. How can you discover what has gone wrong? By looking at the error log.

On my system, the error log contains the following entry:

```
[Mon Dec 2 17:32:04 1996] access to
/home/reuven/Text/Websmith/test-8.pl failed
for ahad-haam, reason: malformed header from script
```

Not only does Apache tell us that there was an error, and that the program died, but it even presents the reason, which is listed as “malformed header from script”. Not the most obvious message for newcomers, but it tells you what you need to know, namely that the header sent wasn't in the proper format and thus caused an error.

We can now fix the program and remove the error message by sending a MIME header before anything else:

```
#!/usr/local/bin/perl5 -w
    use strict;      # Check our syntax strictly
    use diagnostics; # Tell us how to fix mistakes
    use CGI;         # Import the CGI module
    # Create an instance of CGI
    my $query = new CGI;
    # Send the MIME header
    print $query->header("text/html");
    # Begin the HTML
    print $query->start_html(-title =>
          "Hello, world!");
    # Print something fairly uninteresting
    print "<P>Hello, World Wide Web!</P>\n";
    # End the HTML
    print $query->end_html();
```

Sure enough, the fixed program doesn't produce any new messages in the error log.

Premature Program Termination

A typical idiom in Perl might be:

```
open(FILE, "$filename") ||
        die "Cannot open $filename";
```

The above line means that **$filename** should be opened for reading, and that the opened file should be accessed via the descriptor named **FILE**. If the program cannot open the file for some reason the program should terminate, producing a message. Let's look at how that works within a CGI program:

```
#!/usr/local/bin/perl5 -w
  use strict;      # Check our syntax strictly
  use diagnostics; # Tell us how to fix mistakes
  use CGI;         # Import the CGI module
  # Create an instance of CGI
  my $query = new CGI;
  # Set this to a file
  my $filename = "/foo/bar/blah.txt";
  # Send the MIME header
  print $query->header("text/html");
  # Begin the HTML
  print $query->start_html(-title =>
        "Hello, world!");
  # Print the contents of $filename
  open (FILE, "$filename") ||
        die "Cannot open $filename ";
  print while (<FILE>);
  close (FILE);
  # End the HTML
  print $query->end_html();
```

Looks good, right? Well, the above will indeed work just fine, assuming that **$filename** exists. But if and when it doesn't, or if the program doesn't have permission to read the file, the second half of the statement is executed—and the program terminates, producing an error message.

If **$filename** doesn't exist, and you run the program from the command line, you get the following output:

```
Content-type: text/html
<HTML><HEAD><TITLE>Hello, world!</TITLE>
</HEAD><BODY>Uncaught exception from user code:
Cannot open /tmp/blah.txt at ./test-8.pl line 20.
```

But if you run it from your web browser, you get nothing at all—no error message, and no response, because the error message "**Cannot open /tmp/blah.txt**" was sent to stderr, rather than stdout. The error log, however, contains the following:

```
Uncaught exception from user code:
Cannot open /tmp/blah.txt at /home/reuven/Text/Websmith/test-8.pl line 20.
```

Note that the error message is inserted into the log without date or time information, since the text was sent directly by the Perl program (or by the Perl binary), rather than the web server. The web server automatically stamps its

own error messages with the current time and date, but your programs don't enjoy that feature—unless, of course, you include the **Carp** library included with **CGI.pm**, which redefines the **die** routine, as well as several others, such that error messages are preceded by the relevant time and date. Thus, if you include the line:

```
use CGI::Carp;
```

at the top of the above program, the program still fails as expected, but your error log now contains the following entry:

```
[Tue Dec 3 11:55:14 1996] test-8.pl:
Cannot open /tmp/blah.txt at
/home/reuven/Text/Websmith/test-8.pl line 21.
```

Now we know not only what went wrong, but also when it happened. The Carp library can be very useful when debugging CGI programs, especially if you find yourself running the same program repeatedly. This way, there isn't any question whether the error occurred the first or fifth time that you ran the program, since the time at which the program ran is written right next to the error message.

Once you have installed Carp, you can use it for recording non-fatal errors, as well. For example, if you are trying to keep track of the value of a particular variable, you can include the line:

```
carp 'variablename = "'. $variablename . '"';
```

Because it includes the time and date information, this is better than:

```
print STDERR 'variablename = "', $variablename,
        '"';
```

Although using Perl's built-in syntax checkers and warnings (with the **-w** flag and the **strict** and **diagnostics** modules) is a great idea, realize that loading the **CGI::Carp** module will set off warning bells because it redefines several subroutines. You can ignore these warnings, but don't be surprised when they appear.

What About the User?

All this time, we have ignored our users, who are really supposed to be the focus of our efforts. It's all well and good that we are improving error messages for the purpose of debugging programs, but none of the ideas that we have looked at until now have done much for what the user sees.

The problem is that if you use **die** to output a message to the error log, the program dies off before it has a chance to produce any sort of “whoops” message for the user. If you're building an interactive web site, the last thing

you want to do is present a user with a blank results page; at the very least, you should tell him that something has gone wrong, and that you are trying to fix the problem.

One easy way to do this is to replace the line:

```
open (FILE, "$filename") ||
        die "Cannot open $filename ";
```

with the following:

```
open (FILE, "$filename") ||
        &log_and_die ("Cannot open $filename ");
```

This new version opens the file as necessary, but invokes a subroutine named **log_and_die**, if the file cannot be opened. You can then define the subroutine as follows:

```
sub log_and_die
  {
        # Get arguments
        my $string = shift;
        # Begin the HTML
        print $query->start_html(-title => "Error");
        # Give a message
        print "<P>Sorry, but this program ";
        print "has encountered an error. The system ";
        print "administrator has been ";
        print "informed.</P>\n";
        print "<P>Would you like to return to the;
        print "<a href=\"/\">home page</a>?\n";
        # End the HTML
        print $query->end_html();
    # Now die with the error message
    die $string;
  }
```

The above routine is obviously a bit Spartan for most tastes, but the point is obvious: By wrapping errors into a separate routine, you can have our cake and eat it, too. The user gets an error message directing him or her to another site, and apologizing profusely for the program's errors. And the system administrator or web master gets a note in the error log describing exactly what went wrong, making it possible to fix the program more easily.

You could obviously add to this routine. For starters, a good idea might be to e-mail the system administrator or programmer to let him or her know that an error had occurred. If the program were part of a critical system, then perhaps it would even be a good idea to send e-mail to a system tied into an alphanumeric paging system.

Sending MIME Headers First

In all of the above examples, we call **die** only after having already sent the MIME header to the browser (with the built-in CGI.pm “header” method). Failing to do so can create all sorts of odd problems, and results in the dreaded message:

```
Server Error. The server encountered an internal
error or misconfiguration and was unable to complete your request.
```

For example, running the following CGI program:

```
#!/usr/local/bin/perl5 -w
  use strict;      # Check our syntax strictly
  use diagnostics; # Tell us how to fix mistakes
  use CGI;         # Import the CGI module
  use CGI::Carp;
  # Create an instance of CGI
  my $query = new CGI;
  # Don't do very much
  die "This program crashed on purpose ";
```

The above program dies upon reaching the last line, and inserts an appropriate message into the httpd error log. What will the user see on his or her browser? In the above example, the user will get the dreaded:

```
The server encountered an internal error or
misconfiguration and was unable to complete your request.
```

You can alleviate this problem by sending a MIME header before performing any serious computations:

```
#!/usr/local/bin/perl5 -w
  use strict;      # Check our syntax strictly
  use diagnostics; # Tell us how to fix mistakes
  use CGI;         # Import the CGI module
  use CGI::Carp;
  # Create an instance of CGI
  my $query = new CGI;
  print $query->header("text/html");
  # Don't do very much
  die "This program crashed on purpose ";
```

Note: while the documentation for CGI::Carp claims to alleviate this problem, experiments with the version installed on my system (from early September 1996) didn't seem to work quite this way. I thus suggest that you output MIME headers as close to the beginning of your programs as possible, simply to alleviate such problems.

Despite the colorful language that might occur to you when your program produces an error message, good warnings and logs are extremely helpful when writing and debugging code. This is especially true for CGI programs, where much of the work takes place behind the scenes and the only output to the user comes via a web browser. Learn to use the httpd error log effectively, and while your programming problems certainly won't disappear, your mistakes should be easier to identify and correct.

Reuven M. Lerner has been playing with the Web since early 1993, when it seemed more like a fun toy than the world's Next Great Medium. He currently works from his apartment in Haifa, Israel as an independent Internet and Web consultant. When not working on the Web or informally volunteering with school-age children, he enjoys reading (on just about any subject, but especially computers, politics, and philosophy—separately and together), cooking, solving

crossword puzzles, and hiking. You can reach him at reuven@the-tech.mit.edu or reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Gary Moore

Issue #35, March 1997

I just read your PageSat article. We've been waiting for our system for three months. Unfortunately if you dial the main PageSat number you get another number to call, 415-493-9592—which says PageSat is history, gone, no longer in service. (It was a good article anyway.)

Another Satellite News Solution?

I just read your PageSat article. We've been waiting for our system for three months. Unfortunately if you dial the main PageSat number you get another number to call, 415-493-9592—which says PageSat is history, gone, no longer in service. (It was a good article anyway.)

It's too bad. I was eagerly waiting for this, since our Newsfeed is very poor. Do you have any other suggestions for an alternate newsfeed? Ed Longstrom edl@rock.spectra.net

Author Responds

Rich Myers, who wrote the article in question, recently sent us e-mail on this very question:

NCIT (aka Pagesat) has ceased operations effective 12/4/96. Norman Gillaspie is starting a new business, PC-Sat, to continue the satellite newsfeed for both current and future customers.

We should have a web page on-line by the time you read this. The location of the web page will be: http://www.pc-sat.com.

Would you please pass on this info to the interested parties?

Thanks in advance for your assistance! Rich Myers rich@webworks.net

Lack of NFS lockd And statd a Problem

First let me say that I enjoy *LJ* very much and get a lot out of every issue.

I'm considering using Linux in some commercial installations and have held off because of a glaring omission in the networking subsystem. I can find no evidence of Linux support for the NFS lockd and statd network lock managers which are required to guarantee exclusive file access over NFS. Our application support software uses a shared NFS mounted file system and must have this to work. I believe the lack of this capability will impede Linux's acceptance in enterprise computing environments that depend on robust networking support. Other than that, it seems Linux offers the most bang for the buck for Unix computing and networking. Toby L. Kraft tkraft@krafte.com

More On “The Politics of Freedom”

I read with interest your editorial (“The Politics of Freedom”, Phil Hughes) in the October 1996 *Linux Journal*, #30, and thanks for webbing Richard M. Stallman's letter for background. Will you entertain a comment from an outsider?

It's too bad the word “political” has such a bad flavor. Virtually everyone wants to make things better in his community (the *polis*) and tries to persuade others to cooperate in doing so, i.e., acts politically... the catch, of course, being how to decide what is “better”.

I don't use Linux. I make a living writing commercial software for commercial platforms. I have also written, published, and placed in the public domain a few utility programs. I did so for at least three reasons I'm consciously aware of (not necessarily in priority order):

- to make useful tools available to others
- to share my pleasure in solving a problem elegantly
- to show off what a clever bugger I am

At least one of these motivations—the desire to benefit my fellows—RMS also feels. Perhaps he feels the others as well. Nothing wrong with any of 'em.

You suggest RMS “feels that everyone has to believe exactly the same things he does.” I read his letter somewhat differently.

First, let's dispose of the myth of “free” software. They ain't no sech animal. Even if you obtain it without exchange of valuta, you must still invest time and skullsweat to use it—both scarce and valuable commodities, perhaps more so than money. So, any system competes with all others for your investment, whether cash changes hands or not.

Perception is reality. If Linux and GNU do not share a name, they are more likely to be perceived as distinct rather than variants of the same thing (one of RMS's points, and I agree). If they are distinct, they compete.

And “free” software already competes with the notion “if it really was valuable, you'd be charging us for it”—as Heinlein pointed out, why do you think they take collection in church? Added competition between Linux and GNU can't help but drain both against the “common enemy”, which I think is what he is saying.

I'm sorry to have to disagree with Linus Torvalds, but it **does** matter what people call Linux, and this is why. Since Adam's day, to name a thing is to have power over it... and to be trapped by its paradigm.

(Insert clever pun about “Paradigms Lost” here. Oh, never mind.) That's my two pfennigs, anyway. Hope you enjoyed it.Davidson
CorryDAVIDSCO@Attachmate.com

Phil Hughes points out RMS asked for and was offered the opportunity to respond, but so far, we've not heard from him.

OpenGL Does Perl

The November 1996 issue (#31) contains a very nice article on the OpenGL libraries by Jörg-Rüdiger Hill [“Linux Goes 3D: An Introduction To Mesa/ OpenGL”].

In that article it is stated, “Mesa can be called from C and Fortran routines”, which is true, but I felt obligated to let your readers know OpenGL and MesaGL are also callable from Perl 5. Stan Melax has an OpenGL module which is described at http://www.arc.ab.ca/vr/opengl/.

I have just installed it and the requisite MesaGL library and headers on a Solaris box, and I must say it is plenty impressive (the clip script is only 116 lines long—if you don't count shebang and comments, it is only 70 lines of Perl!). The **examples/** directory of the OpenGL module for Perl also includes a **tk_demo** script that uses the *Mainloop()* event handler of the Tk extension to Perl (whereas the other demos do their animation with built-in Perl loops). I know Perl's Tk400.200 extension compiles on most major Linux distributions, and I am fairly certain the OpenGL module does as well.

Thanks for a great issue on graphics. Peter Prymmer
pvhp@lns62.lns.cornell.edu

A timely note for our focus on Perl this issue. Thanks!

LJ to the Rescue

Last week I was int eh field, working on a piece of equipment controlled witha computer runnin iRMx, a “unix-like” Intel product from about 1980. I needed logfiles from the machine. Using all of my clues, I was unable to write on the 3.5" diskette without first using RMX to format it. I turned the equipment back over to the customer and went away witha n unreadable diskette containing data the factory needed right away. Certainly you cannot read it on a Windoze box!

Next I tried it with my llnux system (Debian 2.0.24). While Gates's garbage reads only one kind of file system, maybe RMX is readable as one of the many file systems Linux will support...Load up the modules and hack away: Sorry, no luck!

Then I remembered my *LJ* issue 32, which I picked up on the way to the airport. Out it came and straight I went to Sam Chessman's excellent article, “What is dd?” I ran Sam's “Example 1a” to make a disk image. Voila! I have a file? Mov e the image file to the FAT partition, kill Linux and run Windoze to e-mail it home, along with instructinos for the Unix guys at home to run Sam's “Example qb” to recreate the diskette.

They tried it but did not succeed (you know, th edd supplies with SunOS may not be as capable as ours). So I resolved to hack on the image file. As it turned out, using **CTRL**-K a few timmes in Emacs turned it into radable plain text log files! —D.W. Wieboldt dwiebold@Aus.etn.com

Thanks for letting us know you found the article useful (and thanks to Same Chessman for writing it in the first place). We love to hear about Linux being useful (even as frequently as it is).

Oracle Client Under Linux?

Regarding “Stop the Presses” in *Linux Journal* #32, December, 1996, two items:The first concerns SCO UNIX “Give-away”. I bit, paid, the 20-some bucks and got the CD. It was primarily for work; I wanted to do a demonstration of what Unix could do. The IBMers are quite leery of Linux—software with no huge corporation after thier money. They just don't know how to relate.

Anyhow, I got a PC loaner from network services and was going to load SCO up and I couldn't get to first base! Turns out, SCO doesn't seem to believe IDE hard-drives are used. The installation couldn't see I had 1.6 gigs of space—just said I didn't have a hard drive and so it couldn't proceed. Yeah, I'd be worried about that company if I wanted to see Linux thrive (giggle...).

Afterwards, I brought in Red Hat 4.0. It installed without a hitch. The only thing holding Linux back at my work site is the poor token-ring support. I just haven't been brave enough to even consider trying to develop new drivers.

The second item is a question: Have you heard of anyone running a UNIX Oracle client under LInux? Or getting access to Oracle data through ODBC or other means? Just curious.

I do appreciate your magazine. It's fun and provides a lot of useful information. Keep up the good work. —Dean Siewertdsiewert@execpc.com

Being fair, we plan to subject Free SCO OpenServer to a review in an upcoming issue of *Linux Journal*. It's real “UNIX” so it'll be lots better than Linux, right?

As I've done earlier in this month's letters. I refer your question regarding Oracle clients on Linux to the LJ readers.

More Thoughts on Free SCO

I have taken what I consider to be a very critical look at the SCO offering and I would like to convey the following observations:

1. SCO's product is a compliment to the existing Unix community. It provides casual and concerted users another opportunity to gain insight and respect for an already proven great operating system.
2. SCO's product will **not** infringe, impinge or detract on pico-iota from the existing community of Linux users.

In this MS-day and MS-age, it is imperative alternative operating sytems be encourages to grow and flourish. One of the most effective means of giving a boost to alternative operating systems—and Unix specifically—is to make them truly accessible to the common user. To this end, SCO is making a significant contribution.

And almost as quickly the contribution ends. SCO's offering is sexy, slick and almost inaccessible. It provides all of the Unix tools and almost none of the more necessary understanding. Having started life as a DOS user, I view SCO's contribution as cold and disenfranchizing as Apples' Systme 7 or Windows 95. It has all of the functionality while offering no real understanding of the contents of the operating system's “black box”.

Linux, my first exposure to Unix, provided the all-important tool for true inderstanding : accessibility. I am able to lift the hood, kick the tires and generally test drive the OS functionality in a hands-on fashion. In my naive and

simplistic view, this is the best enfironment for learning. Let SCO offer but know that Linux provides! —William B. Meloney VII bmeloney@mindspring.com

Archive Index Issue Table of Contents

Advanced search

Linux Means Business to the City of Garden Grove

Pyng Chang

Charles Kalil

Issue #35, March 1997

Garden Grove, California is run by Linux—read how it happened here.

Background

The City of Garden Grove has been a Pick shop since implementing a Microdata Reality in the late 1970s. The original system supported 24 users. In 1984 the Reality was replaced by a Honeywell/Ultimate machine with over 100 serial ports for terminals and printers. In 1990, the City needed a larger system, so it replaced the Ultimate with a 512 user Data General running DGUX /Advanced Pick. At the same time, the City was beginning to use personal computers (PCs) in the workplace, leading to the need for a PC network. A consultant was hired in 1993, but his equipment recommendation was declined due to budget constraints.

Layout of the Computer System

Evolution

Still needing a network, the Information Systems department needed to develop a phase-in plan, so under the direction of the IS manager, Robert Shingledecker, in June 1993 a mini-lab was set up to evaluate different network solutions for a multi-server environment in. The first network chosen for testing was Novell 4.0, which promised seamless integration of multiple servers. Two Novell network engineers took three weeks to set up the first server and then an additional three weeks to set up a second one. Even then, the integration was not as promised.

Disillusioned with Novell, Bob decided to go out and purchase the newly released Windows NT. The IS staff took only 3 days to get NT up and running. Unfortunately, at that time NT's speed was unimpressive, Netbeui (the only

available protocol) was not routeable, and NT was unable to share printers with the existing Unix system.

Staff Group Picture

Since the City was already using Unix, Unix was the next system considered for networking. After researching Unix networking on the Internet, the decision was made to try out a NFS network. Unfortunately, four different commercial NFS clients failed to work with Microsoft's newly released Word 6.0. The first NFS product to solve the Word problem was XFS, a shareware NFS client from Germany, available on the Internet. The Internet soon became an invaluable source of both information and software. The City installed its first production network in July 1994, in the Public Services Department, using an SCO Unix 486 File Server and 16 XFS/Windows 3.1 clients. The PCs had dual connections—Ethernet connections to the local SCO server and serial/mux connections to DGUX/Pick.

In September 1994, IS learned that the City would be moving to a new building, and that our network project would have to be accelerated. The computer systems for each campus building, especially that of the Police Department, would have to be able to “stand alone and be fully functional.” XFS did not run on Windows 3.11 at that time, so we continued to search the Internet for a network solution. That's when we discovered Samba. Samba emulates SMB networks (LanManager, WFW, Windows NT) on Unix, and provides an even easier solution for networking since **no** additional non-Microsoft software had to be loaded on the clients. The next production network was installed in December of 1994 in the Housing Department, using an SCO Pentium File server and 20 Windows for Work groups (WFW) clients. The Samba network proved faster than any network previously tested, and in addition, it was much easier to maintain. The server was also running a 16 user license of Advanced Pick (AP), so now the clients had only one connection to the server—Ethernet. Immediately following this success, we set up another Samba network for the Police Department that included a 48 user license of AP running on an SCO Pentium file server with 100 WFW clients.

Front of the New City Hall

In February 1995, having taken care of all the satellite buildings, we prepared to move City Hall, connect all local buildings via Fiber and plan a Wide Area Network (WAN) including a T1 connection to the Public Services Department. Meanwhile, the discovery of Samba led to the discovery of the Unix Platform it was developed on, Linux. Linux had everything SCO had—plus much, much more including Internet tools. Testing then began on the use of Linux. Linux is amazingly simple compared to SCO, its setup and performance is better than

SCO and it's free. In April 1995, the Public Services Network was converted from SCO/XFS to Linux/Samba with noticeable performance improvements. In June 1995, discussion began on what to do with the Data General (DG), as the City planned to move into its new building in November.

Pick systems released a Beta copy of AP/Linux and the City started testing it in September 1995. After discussions with Pick Systems regarding support of AP/Linux, it was decided to leave the DG behind and run the City with the successful Intel/Linux/Samba/Pick network. The City also established a presence on the Internet in September by setting up a web page (http://www.ci.garden-grove.ca.us) hosted by Deltanet Internet Services, with future plans to bring it in house.

In November, the DG was abandoned, and the new City Hall is now run by two Linux Pentium File servers. One runs Samba and serves the PC network, and the other runs the Advanced Pick database. Together the servers handle 150 WFW clients. Housing's SCO/AP/Samba server was changed over to Linux followed by the conversion of the Police server from SCO to Linux in March. Concurrently, the Fiber backbone was installed throughout the Campus buildings. IS was now prepared to set up an intranet. By this time the network consisted of 8 servers (5 production Linux servers, 2 lab [test] Linux servers and 1 Sun server for imaging). The City Hall Samba Server was now running NCSA's web Server and the City's intranet was born.

The IS staff then began to work with HTML, Perl and Python to create CGI applications on the Web. Next, we decided to get the Pick Server involved. The first web page to extract data from the Pick Server did so with HTML, CGI and Perl. It was done by having the Perl script call a Pick application which wrote to a temporary file in Unix that was then read by the calling Perl script. This method worked, but proved to be slow. Because of the networking features of Linux and AP, we decided that the speed could be improved if the two servers communicated over sockets. The new speed was incredible. Almost instantaneously, the data requested by the web page was returned by the Pick server. The City's intranet was undergoing a growth spurt, and a GUI solution for Pick applications had been found.

In June of 1996, the City added a T1 speed Frame Relay connection to Intelenet Internet Services, adding yet another Linux Server that acts as a firewall between the new Frame Relay line and the City's network. IS is currently in the process of bringing the City's WWW page in house. We can now write applications to make information on the Pick Server available to the world via the World Wide Web. IS is also creating an imaging system based on the WAIS server—so development continues.

Conclusion

Without Linux and Samba, the project could never have come in within budget nor made the time-line for the move. In just a few short months, the City switched from a mini-computer with dumb terminals to a total network environment. The Linux machines run 24 hours a day and have proven to be very stable and reliable. Our thanks go out to all the Internet community for providing such a wonderful environment.

Equipment

The city now has over 300 486 and Pentium PC92s running WFW, Microsoft Office and Netscape Navigator. All are connected with 10baseT Ethernet cards (NE2000 and 3COM). Microsoft Telnet is used to connect to the Pick Server for legacy applications.

Linux servers are Intel-based Pentiums (100-133) with 64MB RAM, (2) 1GB SCSI drives and a 2GB DAT tape drive.

Printers are all HP LaserJets connected with an internal Jet Direct Card (Ethernet).

The network includes Category 5 cable with a Fiber Optic backbone, T1 line to Public Services, and 1544 Kbps Frame Relay to the Internet, 3COM hubs, patch panels and switch, Prelude and ADC DSU/CSU92s, and a Cisco Router.

IS Staff

The Informations Services staff consists of a manager, 2 systems analysts and a technician. Robert Shingledecker is the Information Systems Manager for the city of Garden Grove, California. He has written early machine code and assembly language programs. He uses the Internet regularly to research solutions for the city's IS needs and future developments. He can be reached at roberts@exo.com. The authors are the systems analysts. Victor Chang, Information Systems Technician, supports the PC client's needs—he installs and supports software (Windows, MS Office, Netscape, etc.), troubleshoots and fixes hardware problems.

Pyng Chang is a Senior Systems Analyst with the city of Garden Grove, California. Both authors have Pick OS backgrounds, and have gradually moved into the Unix and PC environment since the inception of the project. They are both involved with all aspects of Unix network administration, and are currently writing the GUI interface to the Pick routines with the HTML, CGI and Perl.

Charles Kalil is a Senior Systems Analyst with the city of Garden Grove, California. Both authors have Pick OS backgrounds, and have gradually moved into the Unix and PC environment since the inception of the project. They are both involved with all aspects of Unix network administration, and are currently writing the GUI interface to the Pick routines with the HTML, CGI and Perl.

Archive Index Issue Table of Contents

Advanced search

New Products

Margie Richardson

Issue #35, March 1997

Cyclone 6 UART Serial Board, Visual SlickEdit Version 2.0 for X Windows, FootPrints, Web-based Helpdesk and more.

Cyclone 6 UART Serial Board

GTEK, Inc., announced the Cyclone 6 intelligent UART serial board. The Cyclone 6 features 64 byte transmit and receive FIFO buffers, automatic handshaking and flexible addressing options. GTEK has also released a new version of their BlackBoard-8A intelligent UART serial board. Drivers are available for Linux. For pricing for both products contact GTEK.

Contact: GTEK, Inc., P.O. Box 2310, Bay St. Louis, MS 39521-2310, Phone: 800-282-GTEK, E-mail: spot@gtek.com.

Visual SlickEdit Version 2.0 for X Windows

MicroEdge, Inc. announced today the release of Visual SlickEdit v2.0 for X Windows. Visual SlickEdit is a programmer's editor, possessing cross platform functionality and compatibility with nearly any language. New features include API Apprentice, a C/C++/Java code beautifier, difference editing, selective display, code block selections and hex editing. Visual SlickEdit v2.0 is available for Linux for a price of $195.

Contact: MicroEdge, Inc., P. O. Box 18038, Raleigh, NC 27619, Phone: 800-934-3348, Fax: 919-303-8400, E-mail: sales@slickedit.com, URL: http://www.slickedit.com/.

FootPrints, Web-based Helpdesk

UniPress Software, Inc. announced the availability of FootPrints, the first web-based helpdesk system designed to record and track problems, solutions, bugs, change requests, and to make the information available to anyone with access

to the Internet or Intranet. FootPrints creates databases of related information that includes priority, status, description of each entry. This information can be viewed and edited by the users. FootPrints requires a Unix-based or NT-based web server. Starter License includes the server software and three user licenses for $1,995.

Contact: UniPress Software, Inc., 2025 Lincoln Hwy, Edison, NJ, 08817, Phone 908-287-2100, Fax: 908-287-4929, E-mail: info@unipress.com, URL: http://www.unipress.com/.

MagiCaster Multimedia Interface

DIMACC, Inc. and Boomerang Communications Corporation have released MagiCaster, the first customizable multimedia interface for user navigation within a web site. MagiCaster utilizes animated button graphics for site navigation, streaming text for new updates, and supports live and streaming audio reception. Best of all, its free. MagiCaster is fully compatible with all Shockwave enabled web browsers, and can be downloaded from http://www.magicaster.com/.

DIMACC,Inc., Amherst, NY, Phone: 716-636-6001, URL: http://www.magicaster.com/.

WebTech C++ Library Supporting HTML 3.2

WebTech from Tech Development and Consulting is a C++ class library for creating web pages and CGI applications without requiring any knowledge of HTML or CGI. It is built on top of the Standard Template Library (STL), which increases portability and efficiency. Java applets, JavaScript, VB Script and ActiveX Script are all supported. WebTech is available for Linux at a price of $399(US).

Contact: Tech Development and Consulting, 2308 Houma Blvd. Suite 810, Metarie, LA 70001, Phone: 504-456-8826, E-mail: tparent@techdevelopment.com, URL: http://www.techdevelopment.com/.

InterCam Telerobotic System

Perceptual Robotics, Inc announced InterCam, the first telerobotic system to offer live, interactive video over the World Wide Web. InterCam is capable of running on Unix-based Internet networks, and gives remote viewers complete control over what they see.The technology can be tried out online at http://www.nethomes.com/intercam/. InterCam is available for approximately $25,000.

Contact: Perceptual Robotics, Inc. 1840 Oak Avenue, Evanston, IL 60201, Phone: 847-475-0512, Fax: 847-866-1808, E-mail: pri-cooper@nmu.edu, URL: http://www.nethomes.com/InterCam.

OpenPath Web 2.0

Trilogy Technology International has released OpenPath Web 2.0 to help developers create sophisticated web applications. It is a template-based language that generates powerful IS applications on the web.OpenPath Web runs faster because it is compiled and provides a rich development environment. The Lite version is a free download at http://www.openpath.com/. Full version for Unix is $295.

@Contact:Reach Trilogy, 17870 Castleton Street, City of Industry, CA 91748, Phone: 800-585-8668, E-mail: maini@openpath.com, URL: http://www.openpath.com/.

NetTerminal

Boundless Technologies, a subsidiary of SunRiver Corp., announced the NetTerminal for color text applications over a TCP/IP network. NetTerminal is a low-cost device that functions as a network alternative to PCs running terminal emulation software and serial terminals connected to TCP/IP terminal servers. It combines color VGA compatibility with enhanced performance for faster throughput of data. NetTerminal model 100 is available for $699 and model 200 for $799. It is Linux compatible.

Contact: Boundless Technologies, Echelon IV, Ste 200, 9430 Research Boulevard, Austin, Texas 89759-6543, Phone: 512-349-5802, Fax: 512-346-7062, E-mail: craig.parks@boundless.com, URL: http://www.boundless.com/.

VirtuFlex

VirtuFlex Software Corp. announced expansion of its VirtuFlex Partnership Program. VirtuFlex is a versatile tool for building sophisticated dynamic web sites that integrate databases, fax, e-mail, telephony and CGI scripts. The Partnership Program offers free training and support, as well as product discounts. VirtuFlex runs on standard Unix workstations including Linux.

Contact: Virtuflex Software Corp., 930 Mass. Ave, Cambridge, MA 02139, Phone: 617-497-8006, Fax: 619-492-0486, E-mail: comments@virtuflex.com, URL: http://www.virtuflex.com/.

Java-based Website Monitor

Aquas announced the release of the Bazaar Analyzer Pro version 1.0. Bazaar is a web monitoring and analysis tool developed in Java, and is the first real-time interactive product to monitor the productivity of commercial web sites. With single point installation at the server, deployment and maintenance is fast and easy, particularly in widely distributed user environments. Bazaar Analyzer Pro is available for Linux and sells for $990.

Contact: Aquas, Inc., 599 North Mathilda Ave., Ste. 210, Sunnyvale, CA 94086, Phone: 408-737-7122, Fax: 408-737-1292, URL: http://www.aquas.com/.

View Designer/X

Dirk Laessig announced the View Designer/X, a new Motif Interface Builder for Linux. It enables applicaton developers to design user interfaces with Motif 2.0 widgets and to generate C and C++ code. The VDX provides an interactive WYWIWYG View and a Widget Tree Browser which can be used to modify the structure of the user interface. All resources are adjustable by Widget Resource Editor, Bredex, GmbH, Germany is distributing the View Designer/X via Web Service.

Contact: Bredex GmbH, Germany, E-mail: dirk@unifix.de, URL: http://www.bredex.de/EN/vdx/.

Archive Index Issue Table of Contents

Advanced search

Features of the TCSH Shell

Jesper K. Pedersen

Issue #35, March 1997

LG #35

In this article, I will describe some of the main features of TCSH, which I believe make it worth using as the primary log in shell. This article is not meant to persuade bash users to change. I've never used bash, so I know very little about it.

As some of you know, I've created a configuration tool called The Dotfile Generator, http://www.imada.ou.dk/~blackie/dotfile/, which can configure TCSH. I believe that this tool is very handy for getting the most out of TCSH without having to read the manual a couple of times. Therefore, I'll refer to this tool several times throughout this article to show how it can be used to set up TCSH.

Why is the Shell So Important?

The shell is your interface to executing programs, managing files and directories, etc. Though very few people are aware of it, they use the shell frequently in daily work, e.g., completing file names, using history substitution and aliases. The TCSH shell offers all of these features and a few more, which the average user seldom optimizes.

With a high knowledge of your shell's power, you can decrease the time you need to spend in the shell, and increase the time spent on original tasks.

Command Line Completions

An important feature used by almost all users of a shell is *command line completion*. With this feature you don't need to type all the letters of a file name—just the ambiguous ones. This means that if you wish to edit a file called

file.txt, you may need to type only **fi** and press the TAB key, and the shell will type the rest of the file name for you.

Basically, one can use completion on files and directories. This means that you cannot use completion on host names, process IDs, options for a given program, etc. Another thing you cannot do with this type of completion is to complete directory names when typing the argument for the command **cd**.

In TCSH the completion mechanism is enhanced, so that it is possible to tell TCSH which list to use to complete a particular command. For example, you can tell TCSH to complete from a list of host names for the commands **rlogin** and **ping**. An alternative is to tell it to complete only on directories when the command is **cd**.

To configure user-defined completion using The Dotfile Generator (TDG), go to the TDG page **completion -> userdefined**; this will bring up a page which looks like Figure 1.

Figure 1. TDG Completion/Userdefined Page

For the command name, you tell TDG which command you wish to define a completion for. In this example it is **rm**.

Next you have to tell TDG to which arguments to the command this completion applies. To do this, press the button labeled **Position definition**. This will bring up a page, which is split into two parts as shown in Figures 2 and 3.

Figure 2. TDG Position Definition Page

In the first part, you tell TDG the position definition that should be defined from the index of the argument to be completed (i.e., the one where the TAB key is pressed). Here you can tell it that you wish to complete on the first argument, all the arguments except the first one, and so forth.

2066f3.gifFigure 3. TDG Pattern Definition Page

The alternative to “position-dependent completion” is “pattern-dependent completion”. This means that you can tell TDG that this completion should only apply if the current word, the previous word or the word before the previous word conform to a given pattern.

Now you have to tell TDG which list to complete from. To do this, press the button labeled **List**. This will bring up a page where you can select from a lot of different lists, e.g., aliases, user names or directories.

Files and Directories

Four of the lists you can select from are **Commands, Directories, File names** and **Text files**. If you select one of these, only elements from that directory are used.

Predefined Lists

There are two ways to specify completion from a predefined list. One is to mark the option **predefined list**, and type all the options in this list.

This solution is a bad idea if the list is used in several places (e.g., a list of host names). In that case, one should select the list to be located in a variable, then set this variable in the **.tcshrc** file.

Output from Command

In many cases the list should be *calculated* when the completion takes place. For example, a list of users located at a given host or targets in a makefile would need to be calculated.

To set up such a completion, first develop the command which returns the list to complete from. The command must return the completion list on standard output as a space-separated list. When this is done, insert this command in the entry **Output From Command**.

Here's a little Perl command which finds the targets in a makefile:

```
perl -ne 'if (/^([^.#][^:]+):/) {print "$1 "}'
        Makefile
If this is inserted in the entry, you can complete on targets from the file
called Makefile in the current working directory.
If someone should think I describe TCSH through it only in order
to promote TDG,(s)he should take a look at the following line, which is the
generated code for the make completion:
```

```
complete make  'p@*@`perl -ne \
        '"'"'>if (/^([^.#][^:]+):/) \
        {print "$1"}'"'"'Makefile`@'
```

Restrict to Pattern

With user-defined completion, you can restrict the files which are matched for each command. Here are two very useful examples:

- Restrict latex to *.{tex,dtx,ins} The latex command will complete only on files ending in the extensions **.tex**, **.dtx** or **.ins**.
- Restrict **rm** to ^*.{tex,html,c,h}.` This means that you cannot complete **rm** to a **.tex**, **.html**, **.c** or **.h** file. I've done that a few times, when I wanted to delete a file called important.c~. Since the file important.c existed, TCSH

completed only to that name, and... I deleted the wrong file, because I was too quick.

Additional Examples

Additional examples can be obtained by loading the export file distributed with TDG. Please note that if you wish to keep the other pages, you have to tell TDG to import only the page **completion/userdefined**. This is done on the **Details** page, which is accessible from the **reload** page.

Configuring the Prompt

Configuring the prompt is very easy with TDG. Just enter the menu called **prompt**. On this page you can configure three prompts:

1. **prompt**: the usual prompt, which you see on the command line, where you are about to enter a command
2. **prompt2**: is used in **foreach**, and **while** loops, and at lines continuing lines ended with a slash
3. **prompt3**: is used when TCSH tries to help you, when it meets commands it doesn't know—i.e., spell checking

The prompts are mixed with tokens and ordinary text. The tokens are inserted by clicking on them in the menu below the scrollbar, and the ordinary text is simply typed in. When a token is inserted, an indication will be shown in the entry. Figure 4 is an example of how this may look.

Figure 4. Prompt and Text Mix

Some of the prompts may be positioned in the xterm title bar instead of on the command line. To do this, choose **font change** and select **Xterm**.

History

The history mechanism of the shell makes it easier to type similar commands after each other. To see a list of the previously executed commands, type **history**.

The following table lists the *event specifiers*:

!n	This refers to the history event, with index **n**

!-n	This refers to the history event which was executed **n** times ago: **!-1** for the previous command, **!-2** for the one before the previous command, etc.
!!	This refers to the previous command
!#	This refers to the current command
!s	This refers to the most recent command whose first word begins with the string **s**
!?s?	This refers to the most recent command which contains the string **s**

With these event specifiers, you can re-execute a command, e.g., just type **!!**, to re-execute the previous command. However, this is often not what you want to do. What you really want is to re-execute some part of a previous command, with some new elements added. To do this, you can use one of the following *word designators*, which is appended to the event specifier with a colon.

0	The first word, i.e., the command name
n	The **n**th word
$	The last argument
%	The word matched by an **?s?** search
x-y	Argument range from x to y
*	All the arguments to the command (equal to ^-$)

Now it's possible to get the last argument from the previous command, by typing **!!:$**. You'll often see that you need to refer to the previous command, so if no event specifier is given, the previous command is used. This means that instead of typing **!!:$**, you need type only **!$**.

More word designators exist, and it's even possible to edit the words with different commands. For more information and examples, please take a look at

the TCSH manual found on-line at http://www.imada.ou.dk/Technical/Manpages/tcsh/History_substitution.html.

It is possible to expand the history references on the command line before you evaluate them by pressing ESC-SPC or ESC-!. (That is: first the escape key, and next the space bar or the **!** key). On some keyboards you may use the meta key instead of the ESC key, i.e., M-SPC (one keystroke).

Patterns

Many operations in the shell work on many files, e.g., all files ending with **.tex** or starting with **test-**. TCSH has the ability to *type* all these files for you, with file patterns. The following list shows which possibilities exist:

*	Match any number of characters
?	Match a single character
[...]	Match any single character in the list
[x-y]	Match any character within the range of characters from x to y
[^...]	Match elements which do not match the list
{...}	This expands to all the words listed. There's no need that they match
^...	^ in the beginning of a pattern negates the pattern

Examples

- *.tex: match all files ending with .tex
- ^*.tex: match all files which does not end with .tex
- xxx{ab,cde,hifj}yy: match xxxabyy xxxcdeyy and xxxhifjyy
- *.[ch] or *.{c,h}: match all .c and .h files

The Shell Expand Patterns

An important thing to be aware of is that it is the shell which expands the patterns, and **not** the program, which is executed with the pattern.

An example of this is the program **mcopy** which copies files from disk. To copy all files, you may wish to use a asterisk, as in: **mcopy a:* /tmp**. However, this does not work, since the shell will try to expand the asterisk. Since it cannot find any files which start with **a:**, it will signal an error. So if you wish to send a asterisk to the program, you have to escape the asterisk: **mcopy a:* .**

There are two very useful key bindings which can be used with patterns. The first is **C-xg**, which lists all the files matching the pattern, without executing the command. The other is **C-x***, which expands the asterisk on the command line. This is especially useful if you wish to delete all files ending in .c except **important.c**, **stable.c** and **another.c**. Creating a pattern for this might be very hard, so just use the pattern ***.c**. Then type **C-x***, which will expand ***.c** to all your .c files. Now it's easy to remove the three files from the list.

Aliases

When using the shell, you will soon recognize that certain commands are typed again and again. One of the top ten is surely **ls -la**, which lists all files in a directory in long form.

TCSH has a mechanism to create aliases for commands. This means that you can create an alias for **ls -la** called **la**.

Aliases may refer to the arguments of the command line. This means that you can create a command called **pack** which takes a directory name and packs the directory with tar and gz.

Aliases can be a bit hard to create since you often want history/variable references expanded at the time of use, not at the definition time. This can be done more easily with TDG; go to the page **aliases** to define aliases. If you end up with an alias you cannot define on this page, but you can in TCSH, please send me e-mail (blackie@imada.ou.dk). For more information about aliases, see the TCSH manual at http://www.imada.ou.dk/Technical/Manpages/tcsh/Alias_substitution.html.

Timing Programs

Have you ever needed to know how long a program took to run, i.e., how much CPU it used? If so, you may recognize the output from the TCSH built-in time command:

```
  0.020u 0.040s 0:00.11 54.5%     0+0k 0+0io 21pf+0w
```

Informative? Yes but... the GNU time command is a bit more understandable:

```
0.01user 0.08system 0:00.32elapsed 28%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+0minor)pagefaults 0swaps
```

In TDG you can configure the output from the time command on the page called **jobs** shown in Figure 5.

Figure 5. TDG Jobs Page

References

As you may have guessed, TDG and this article will help you a lot in the use of TCSH, but you may need to read a bit more to get even more out of TCSH. Here are a few references:

- The TCSH manual page at http://www.imada.ou.dk/Technical/Manpages/tcsh/top.html
- *Using csh & tcsh* by Paul DuBois, published by O'Reilly & Associates, 1995: http://www.primate.wisc.edu/software/csh-tcsh-book/
- The TCSH mailing list at tcsh@mx.gw.comr (to join send mail to listserv@mx.gw.com with body text **SUBscribe TCSH *your name***)

Jesper Pedersen lives in Odense, Denmark, where he has studied computer science at Odense University since 1990. He is a system manager at the university and also teaches computer science. He is very proud of his "child", The Dotfile Generator, which he wrote as part of his job at the university. In his spare time, he does Jiu-Jitsu, listens to music, drinks beer and has fun with his girlfriend. He loves pets, and has a 200 litre aquarium and two very cute rabbits. His home page can be found at http://www.imada.ou.dk/~blackie/", and he can be reached at blackie@imada.ou.dk. This article first appeared in Issue 12 of Linux Gazette.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #35, March 1997

Our experts answer your technical questions.

Removing a Boot Manager

How can I remove the boot manager from the master boot block of my hard disk? —Ralph Wu

A Linux Solution

A backup copy of your MBR is stored in **/boot** when first installing LILO. You can restore it with:

```
dd if=/boot/boot.0300 of=/dev/hda bs=446 count=1     (IDE-Disk)
dd if=/boot/boot.0800 of=/dev/sda bs=446 count=1     (SCSI-Disk)
```

—Klaus Franken, S.u.S.E. GmbH kfr@suse.de

A DOS Alternative

Find any MS-DOS boot disk with **fdisk.exe** on it. Execute fdisk with the undocumented parameter **/mbr**:

```
fdisk /mbr
```

It may appear not to work since this usage will return no messages and will exit immediately. It will overwrite your master boot record with a fresh copy, and LILO, Bootlin or whatever other boot manager you may be using will be gone.

—Chad Robinson, BRT Technical Services Corporation
chadr@brtgate.brttech.com

E-mail Error Message

I have a small network of Sun SPARC Solaris 2.4-based machines and a couple of Linux systems. The Suns use NIS+ and the Linux systems use **/etc/hosts** files to know about other hosts on the net. My problem is that I cannot send e-mail from a Linux system to another host. I have no problem when I am connected to the Internet using DNS to resolve host names. Why can't sendmail resolve a host name from **/etc/hosts**? I have no problem with telnet or FTP, but sendmail always says **unknown host**.

—Tim Bower

Configuring Sendmail to Use DNS

Your sendmail must be configured to not use DNS.

If you use the m4-macros, try /usr/doc/packages/sendmail/cf/linux.smtp-nodns.mc:

```
include(`../m4/cf.m4')
VERSIONID(`linux for smtp-only without dns setup'dnl
OSTYPE(linux)
FEATURE(nouucp)dnl
FEATURE(always_add_domain)dnl
FEATURE(nodns)dnl
MAILER(local)dnl
MAILER(smtp)dnl
```

If you edit your /etc/sendmail.cf, try the following parameters:

 # service switch file (ignored on Solaris, Ultrix, OSF/1, others) **O ServiceSwitchFile=/etc/service.switch**

```
# hosts file (normally /etc/hosts)
O HostsFile=/etc/hosts
```

Then you have to create a file /etc/service.switch:

```
hosts   files
aliases files
```

—*Klaus Franken, S.u.S.E GmbH* kfr@suse.de

Modem Won't Hang Up

We are attempting to set up a dial-in server for the Internet site at our community college. We are using Getty_ps and have it up and working fine but cannot figure out how to get the modems to hang up automatically when the user exits the system. Any suggestions on how we can accomplish this? --Barre BullPrince George's Community College

Use Mgetty Instead

Getty-ps may be capable of what you are trying to do, but since you are using it specifically for modem dial-in lines, you should take a hard look at Mgetty. Mgetty is a getty replacement specifically designed for use with modems. It supports almost all of the capabilities that Getty-ps supports alone.

I recommend it since it is designed explicitly for a modem. You should find it much better suited to your task.

You can get Mgetty from any Sunsite mirror. You will find that it also has support for FAX and voice capable modems.

—*Chad Robinson, BRT Technical Services Corporation* chadr@brttech.com

Root Password Won't Change

I've just loaded Linux and I can't add/change my root password. It says that I'm denied access, but when I do change attributes and reboot the system, the system defaults back to where it was: root access with no password protection.

—Dan Sapach

Try This

Have you possibly cleaned up files in your [cw]/etc[ecw] directory? Several files there are not meant to be cleaned up, and doing so will cause change-password attempts to fail. You should have /etc/passwd, /etc/passwd.OLD and /etc/passwd.old.

Also be sure your /usr/bin/passwd is setuid to root. If it isn't, try:

```
chown root.bin /usr/bin/passwd
chmod 4711 /usr/sbin/passwd
```

Finally, make sure that only root can modify /etc/passwd:

```
chmod 644 /etc/passwd
```

—*Chad Robinson, BRT Technical Services Corporation* chadr@brttech.com

Another Possible Solution

It sounds like your root file system isn't getting remounted read-write when the system boots up. Try logging in as root and running **mount -w -n -o remount /** by hand and see if there is a useful error message, or if you can then change the password.

—*Steven Pritchard, Southern Illinois Linux Users Group* steve@silug.org

Can I Use a ZIP Drive?

Is the IOMEGA ZIP Drive a supported device?

—David Jones

Linux Supports ZIP and JAZ Drives

Yes, both the SCSI and parallel port drives are supported under the 2.0.x kernels. You will need either the parallel support compiled in, or support for your SCSI controller. You can alternatively use modules to get the support you need. There are documents covering topics such as these in the Documentation directory at the top level of the kernel sources.

You should also get the latest eject package from sunsite.unc.edu or contrib on ftp.redhat.com. It can handle software ejecting of ZIP and JAZ drives under Linux.

— *Donnie Barnes, Red Hat Software* redhat@redhat.com

ZIP Drive HOWTO

You may want to read the Linux ZIP Drive mini-HOWTO. It is available from any Linux Documentation Project mirror, including www.silug.org/LDP/HOWTO/mini/ZIP-Drive.

—*Steven Pritchard, Southern Illinois Linux Users Group* steve@silug.org

Cannot Execute Binary File

I've been having this problem with some binary files I downloaded recently. When I try to execute the binary file, I get the following error:

```
If the shell is bash : cannot execute binary file
If the shell is tcsh : Exec format error. Wrong Architecture.
```

Along with this, there will be a message in /var/adm/messages saying:

```
"N_TXTOFF < BLOCK_SIZE. Please convert binary."
```

I've installed Slackware 96 with the Linux kernel upgraded to 2.0.24.

—*Tushar*

Support Needed in Kernel

My first guess is that you are trying to run a.out programs without a.out support in the kernel. Make sure that when you run **make config, make menuconfig**, or **make xconfig** in your kernel source directory you answer yes to the following question:

```
Kernel support for a.out binaries (CONFIG_BINFMT_AOUT)
[Y/m/n/?]
```

and, for that matter, to this one as well:

```
Kernel support for ELF binaries (CONFIG_BINFMT_ELF)
[Y/m/n/?]
```

While the majority of Linux software is now based on the ELF binary format, there is still a lot of software based on the older a.out format.

—*Steven Pritchard, Southern Illinois Linux Users Group* steve@silug.org

Archive Index Issue Table of Contents

Advanced search