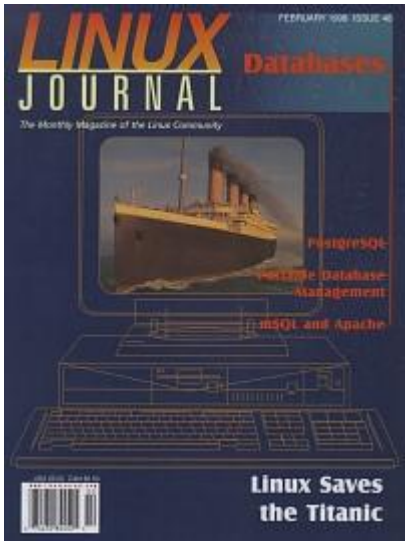
Linux Journal Issue #46/February 1998



Features

PostgreSQL—The Linux of Databases *by Rolf Herzog*
A close look at the PostgreSQL database, including programming interfaces and using it for WWW applications.

The Qddb Database Suite *by Eric H. Herrin, II and Gilbert J. Benson, Jr.*
An introduction the freely available database suite called Qddb.

Beagle SQL, A Client/Server Database for Linux *by Rob Klein*
Mr. Klein introduces us to a database called Beagle SQL that he developed as a learning experience.

Portable Database Management with /rdb *by Ed Petron*
Web server analysis logs and mailing list management is made easy by using the /rdb database system—here's how to do it.

News & Articles

Linux Network Programming, Part 1: BSD Sockets *by Ivan Griffin and John Nelson*
This is the first of a series of articles about how to devlop networked applications using the various interfaces available on Linux.

Linux Helps Bring Titanic to Life *by Darryl Strauss and Wook*
First article in a two part series on using Linux for visual effects in “Titanic”. This article will focus on the technical aspects of the project.

The Quick Start Guide to the GIMP, Part Four *by Michael J. Hammel*

Our series winds up with a detailed description of the toolbox, plug-ins and keyboard acceleration.

A Partner's Survival Guide *by Telsa Gwynne*

A view of life with a hacker brought to us by a mischievous spouse who should know—Ms. Gwynne is married to Alan Cox.

by Craig Oda

Tokyo Linux Users Group Grows Up

Reviews

Personal Empress Database *by David Weis*

Review of personal Empress RDBMS for Linux.

S.u.S.E. V5.0 *by Stuart Green*

The Essential Perl Books *by Eric Raymond*

WWWsmith

Web Counting with mSQL and Apache *by Randy Jay Yarger*

Learn all about Apache modules and mSQL programming using a web counting program as an example.

Linux Works for Me and You *by Maan Bsat*

A high school student tells us about using Linux as a server at school, at home and at work.

At the Forge Attaching Files to Forms *by Reuven M. Lerner*

Mr. Lerner shows us a way to use file elements to allow web site visitors to upload information or program files to the site.

Columns

Letters to the Editor

From the Editor Databases *by Marjorie Richardson*

From the Publisher Needed: Linux Banking Software *by Phil Hughes*

Stop the Presses COMDEX/Fall '97 *by Carlie Fairchild*

Linux Apprentice Setting Up E-mail *by Jonathan Walther*

Setting Up E-mail This article will give you a properly working e-mail setup and an overview of various pieces of e-mail software.

Take Command ispell: Spelling Checker *by Marjorie Richardson*

ispell: Spelling Checker Don't know how to spell? This is the command for you.

Linux Means Business United Railway Signal Group, Inc. *by Lester Hightower and Hank Leininger*

United Railway Signal Group, Inc. The story of how Progressive Computer Concepts has turned United Railway into a Linux shop.

New Products

Best of Technical Support

Archive Index

Advanced search

Linux Journal

AND Search OR Search
Phrase Search Show results per page

Powered by Sphider

LINUX JOURNAL

PostgreSQL—the Linux under the Databases

Rolf Herzog

Issue #46, February 1998

A close look at the PostgreSQL database, including programming interfaces and using it for WWW applications.

One milestone in the spreading of Linux for general use is the availability of office applications. Besides text-processing, spreadsheets and graphic applications, databases are an important part of this type of application. Until now, the big players in the database market like Sybase and Oracle haven't released a Linux version of their products; nevertheless, there are several database packages available for Linux, mostly commercial applications. There is one database which deserves special attention, because it is now developed similarly to Linux. This database is PostgreSQL, formerly known as Postgres95. Last October, version 6.2.1 was published, and this is a good time to take a closer look at this project.

PostgreSQL has a long history beginning in the year 1986 when Michael Stonebraker began its development as a successor to the Ingres database system. The main goal of his project was to show that a relational database system could cope with modern demands of extensibility as well as an object-oriented system. The resulting product was called an object-relational database because it was a relational database system with some object-oriented features such as inheritance and user-defined functions, operators and access methods. In 1994, with the release of version 4.2, the work on Postgres stopped because Stonebraker's project ended. However, some staff members decided to continue with their work and in 1995 released Postgres95 with SQL as the query language rather than PostQuel. Now the development is carried on under the name PostgreSQL by a team of volunteers on the Internet, coordinated by Marc G. Fournier.

The package is not released under the GPL. It has its own license that allows the distribution of modified releases even without source code, as long as the copyright notice remains untouched.

Features of PostgreSQL

- Client/Server Database for a multi-user environment
- Networking with TCP/IP
- Three Authentication methods: Kerberos, host/user based and username/ password authentication
- SQL as query language (It is not fully ANSI SQL92 compliant; options not available are nested subqueries, HAVING clause in an ORDER BY statement, OUTER JOIN, PRIMARY and FOREIGN KEY statements during table creation.)
- Multiple index types, unique indexes and multi-column indices
- User-defined functions (SQL, C), operators and data types
- User-defined sequences and trigger functions
- Language interfaces for C, C++, Objective-C, Java, Perl, Tcl/Tk and Python
- Available third party ODBC driver
- Ported for Linux/Intel, Linux/Solaris, Linux/Alpha, AIX, DEC Alpha AXP, FreeBSD, BSDI, DG/UX, HP-UX, Nextstep, Solaris x86, Solaris Sparc, SunOS Sparc, SGI Irix, SCO, Intel SVR4, Ultrix

The Client/Server Model

Client/Server databases are different from the monolithic systems familiar to users of MS-DOS and Windows machines. It is not sufficient to start only one program to work with the database. There are at least three different applications running simultaneously. One is the server, which in PostgreSQL is called the postmaster. It is a daemon which observes a TCP port (usually port 5432) for a connection from the second: a user client. This user client could be **psql**, a command-line tool for sending SQL queries that comes with PostgreSQL, or specialized applications, like a CGI program that wishes to store data gathered from an HTML form. Once a client connects, the postmaster starts the third application: the back-end server. This is the actual database engine, because only the back-end server has direct access to the stored records of the database. The postmaster connects the client with the back-end server and then waits for other connections. In the meantime, the client sends its queries to the back-end and receives its reply, usually a lot of data. When the client has no other query, the connection drops and the back-end server exits.

The consequence of this procedure is that the client, the program that the user works with, has no direct access to the database. It doesn't know how many records the back end sends in response to a query. It doesn't know if the inserted data were successfully processed, unless the back end signals success. Therefore, fancy table editors such as those in dBase or Access are difficult to implement and don't currently exist.

Installation

The installation of PostgreSQL is a little tedious at present, but it becomes easier with every release. Version 6.2.1 comes with configure-support and detailed and comprehensive installation instructions with specific notes for Linux. I will mention some possible pitfalls during the installation process. Make sure you have a recent version of **flex** installed on your system. In particular v2.5.3 does not work with PostgreSQL whereas v2.5.2 and v2.5.4 work fine. For the client psql, you need the developer libraries from GNU Readline, which are not installed by default on some Linux distributions.

On Linux, PostgreSQL is compiled with a shared library named libpq.so, which is used by all client applications. You need to make sure that the dynamic linker finds this library by adding the directory $POSTGRESDIR/lib to the file ld.so.config. Especially with older versions of ld-linux.so (e.g., v1.8.5), it is necessary to also add this directory to the environment variable LD_LIBRARY_PATH. I do this in my rc.local file, which starts up the postmaster, so it looks like this:

```
echo "starting PostgreSQL postmaster..."
export LD_LIBRARY_PATH=/opt/lib/pgsql/lib
su postgres -c "postmaster -D/opt/lib/pgsql/data \
-d 1 &> /opt/lib/pgsql/postmaster.log &"
```

In the contrib/linux directory of the source distribution, a more sophisticated startup script named postgres.init can be found, that integrates the postmaster startup into the RedHat runlevel system.

Before starting the postmaster, edit the file pg_hba.conf in the data directory. This file holds information, indicating which hosts and users are allowed to connect to the database back end. The minimal file necessary for local operation is:

```
# TYPE  DATABASE  IP_ADDRESS  MASK       USERAUTH  MAP
  host    all      127.0.0.1   255.255.255.255   trust
# The above allows any user on the local system to
# connect to any database under any user name.
```

As mentioned in the documentation, it is essential not to start the postmaster as root, but rather to start it in a special postgres user ID.

For clients to work, they must know where to find the databases. Therefore, you have to set the environment variables PGDATA and PGLIB for every user. I do it in /etc/profile. Additionally, every user who should have access to PostgreSQL needs to have appropriate access permissions in the system databases. This is done with the **createuser** program, run by the postgres account. **createuser** offers you the option of giving the user the right to create databases and/or the right to add other users. It is possible to say no to both

questions, thus giving the user access only to the existing databases. If you are running PostgreSQL as a WWW database back end, you must set access permissions for the user ID of the web server.

Short Introduction to SQL

The best way to get started with PostgreSQL is to read the user manual. It is written for Postgres95 v1.0 and dates back to September 1995, but the information provided there is still useful. Other valuable sources of information are the tutorials, the man pages and the various files in the /doc directory. I will be using the examples from the tutorial in this article whenever possible.

To begin, create a database, i.e., a named container for several tables and accompanying data such as indices and views. To use the database named tutorial, type this command:

```
createdb tutorial
```

Now, you are automatically the owner of the database and have full access to it. Other users have access only if you grant them the appropriate rights.

Next, connect that database to a client program. We will use psql, which comes with PostgreSQL. If you prefer a graphical interface, there is also a Motif client available called **mpsql** (with pre-compiled binaries for Linux). **mpsql** has great editing capabilities but doesn't provide special local commands for listing existing tables and databases. It also lacks a help system that is available in psql. To use psql, type:

```
psql tutorial
```

This command provides you with a shell-like environment in which you can issue SQL commands. Due to the Readline support, you have command history and file name completion with the same key bindings as the bash shell. You can also enter local commands which are processed by the client first if you prefix them with a backslash. Enter **\?** to get a list of local commands. All other commands are sent directly to the back end. Commands can be typed on separate lines. They will be stored in a local buffer until you enter a line terminated with a **;** (semicolon), then the buffer is sent to the back end. Help on SQL commands is available by typing **\h**.

SQL commands can be passed on the command line for shell scripting. The **\i** command reads a file from disk and executes its contents as SQL commands. Be sure to always use absolute pathnames with psql, because the back end knows nothing about the current working directory of the client.

Next, create the following two tables:

```
create table cities (
  name  text,
  population    float8,
  altitude      int--this is a comment
);
create table capitals (
  state char2
) inherits (cities);
```

The text type is a string of characters of variable length. If you enter **int**, you get a four-byte integer value. PostgreSQL comes with 43 predefined data types including several types for time and date values, many types for geometrical objects such as point, circle and polygon and a boolean type. Arrays are also supported. All types are described in the pgbuiltin(l) manual page. If you need additional types, you can add your own. Note that identifier names have not been case sensitive since v6.1.

This example also illustrates a special feature of PostgreSQL: object inheritance. The second table inherits all the fields from the cities table and adds one more field. Later I'll show how to take advantage of that feature.

One often needed feature is missing from PostgreSQL, that is, the ability to define primary keys in the create clause. Primary keys are used to define a default sort order for the tuples and to ensure that the field with that key can't hold duplicate values. It is not supported due to the method used to store the records (or tuples). Every tuple in the database gets a unique object identifier (oid) value, which is unique not only in the table but also in the whole database. There is no way to guarantee a specific order in the table. As of version 6.0 of PostgreSQL it is possible to create a unique index, so that the same effect can be achieved with indices. To create a unique index for our example table, type:

```
create unique index on cities
using btree (name);
```

There are three methods available for index creation: btree, rtree and hash. The method can be specified after the keyword “using”. Only the btree method allows multiple key indices with up to seven keys. Note that not all data types are supported by all index types. In particular, rtree indices are available only for geometric types. If no index type is specified, btree is used as the default. Indices increase the access speed to tables significantly and should be used whenever possible.

The maximum size of a tuple is 8192 bytes. In reality, it is somewhat smaller, because PostgreSQL needs some place for storing internal data. The amount of this space varies from platform to platform. If you need larger fields, use the large objects interface, which provides unlimited fields of transparent data, like MEMO or BLOB fields in other databases; however, you need special functions to access them.

To enter data in the tables, use the **insert** command:

```
INSERT INTO cities VALUES ('San Francisco',
        7.24E+5, 63);
INSERT INTO cities VALUES ('Las Vegas', 2.583E+5,
        2174);
INSERT INTO cities VALUES ('Mariposa', 1200,
        1953);
INSERT INTO capitals VALUES ('Sacramento',
        3.694E+5, 30, 'CA');
INSERT INTO capitals VALUES ('Madison', 1.913E+5,
        845, 'WI');
```

To get the data out of the tables, use the **select** command. It's a very powerful command, so I will demonstrate only some of its characteristics.

```
-- this will return all records in the table
select * from cities;
select * from capitals;
-- to get also the records of the
-- inherited tables, use this syntax:
select * from cities*;
-- here are some variants to limit the returned
-- data:
select name, altitude
from cities
where altitude > 500;
```

To change some values in the table, use the **update command**, which has a similar syntax as the select command:

```
update cities
-- population grows by 10%
set population = population * 1.1
where name = 'Mariposa';
```

When you update your data regularly, you will notice that the tables grow continuously, even if you haven't added new tuples. This is not a bug—it's another special feature called time travel. PostgreSQL keeps a history of all data changes in the table. To access this data, you have to use a special qualifier:

```
select name, population
from cities['epoch', 'now']
where name = 'Mariposa';
```

This example will list all the values of the two fields, name and population of Mariposa, from the creation of the database up to the present. If you don't wish to retain the history data, you can delete it using the **vacuum** command. In the future version 7.0, the time-travel feature will vanish, but at this time you need to vacuum your databases regularly. The vacuum command also has the additional purpose of updating the internal data in order to make faster querys possible. Therefore, it is a good idea to define a **cron** job that runs vacuum every night.

Time and Date Functions

Time and date values are handled in a very flexible manner. I will demonstrate this on a database that I use for tracking my telephone costs for connecting to

my Internet provider. The structure of my database and a sample query look like this:

```
create table telephone (
 datum    abstime,
 online_secs int4,
 units    int2,
 costs    float8
);
select * from pppcosts
where datum >= 'yesterday'::abstime;
datum                   |online_secs|units|cost
Tue Jul 08 00:16:22 1997 MET DST|  486|  3|0.36
Tue Jul 08 20:18:52 1997 MET DST| 1476| 10| 1.2
Wed Jul 09 01:06:33 1997 MET DST| 3317| 14|1.68
```

This query returns my on-line events from yesterday at 00:00:00 until now. The term **'yesterday'::abstime** is an explicit type conversion that makes sure the parser gets the right type for literal values. In this case it is not necessary, but sometimes the input type might be guessed wrong and an error returned. Explicit notation avoids such errors.

It is also possible to get aggregates of data. From my telephone costs table, I can get the costs for the last 30 days by giving the following query:

```
select sum(online_secs) as seconds,
    sum(units) as units,
    sum(costs) as costs
 where datum > ('now'::abstime -
        '30 days'::reltime);
```

The keyword **as** sets the column titles to the specified values; otherwise, they would all read **sum**.

Also supported is the SQL date type which accepts dates in the form mm/dd/yyyy. There is even a mechanism to change the date format for foreign languages (currently only European and US formats are supported). Use the **set** command as follows:

```
set datestyle to 'european';
set datestyle to 'us';
```

The date format is changed to dd/mm/yyyy and then back to mm/dd/yyyy. In v6.1 a comma had to follow the keywords “european” and “us”. In later releases this bug has been fixed.

Sequences

Another nifty feature is the sequence command. A frequent demand to a database is that some column contains a sequence of numbers which is automatically increased when new records are added. To do this, create sequences by giving the following statements:

```
create sequence id start 1000 increment 10;
insert into table values (nextval(id),...);
```

This example will create a sequence, with the name ID, which starts at 1000 and increases by 10 every time the function **nextval()** returns a unique number.

User-Defined Functions

The next example shows how to implement a user-defined function. This technique can be used to overcome a limitation of PostgreSQL—the database system is not able to process subqueries. Querys that contain another query, usually in a where clause, are called subquerys. PostgreSQL supports them only indirectly, i.e., when the subquery is hidden in a function.

```
-- I will use the cities table again
create function big_city() returns set of cities
 as 'select * from cities* where population > 700000'
 language 'sql';
select name(big_city()) as highpop;
```

In the example, the function returns a tuple from the cities table. It is also possible to return only single values by writing e.g., **returns int4**. The language command in the example tells the parser that this is a query-language function. With programming-language functions (e.g., **language 'c'**), you can program your own functions in C and load them as dynamic modules directly into the back end. This mechanism is also used to create user-defined types. (Examples can be found in the tutorial directory of the source tree.) A function is permanently stored in the same way as a table. So, when it is no longer needed, it must be explicitly deleted by giving the **drop function** command.

I have presented only some highlights of the SQL language here, with an emphasis on special features of PostgreSQL. I also omitted examples for joining data from different tables, for creating views and for enclosing your operation in transaction blocks to ensure that either all actions are performed or none at all. I also omitted mentioning cursors, which allow you to limit the amount of tuples returned from a query. These and other features can be learned best from a book about the SQL query language. I have used the book *A Visual Introduction to SQL* from J.H. Trimble, D. Chappel and others, published by Wiley in 1989 for my learning. It is aimed at novices and easy to understand.

Programming Interfaces

One advantage of the long history of PostgreSQL is the availability of many programming interfaces. An Interface for C, C++, Perl5, Tcl and Java JDBC is shipped with the release. Also available are packages for Python and Objective-C (the latter from the GNUStep project). With these libraries you can develop your own client applications.

The C-library is the basic interface to PostgreSQL, because it is used by most of the other libraries. The best example for using it is the psql client. If you are writing clients in C, this program will teach you much of the internal working method of the C-API.

I will now discuss a small program that writes all tuples from the cities table to the screen. (Source code can be obtained from ftp://ftp.linuxjournal.com/pub/lj/listings/issue46/2245.tgz.)

The C-Interface is located in the libpq library. If you have installed PostgreSQL properly, the linker should find the library. Link your program with the **-lpq** options set. In the source code, you have to include the header files with the line:

```
#include <libpq-fe.h>
```

Make sure the compiler finds the header files; they are usually located in the $POSTGRESDIR/include directory.

The first step in the dialog with the back end is to establish a connection to the database. This is done with the command:

```
char* dbname;
strcpy(dbname, "tutorial");
conn = PQsetdb(host, port, options, tty, dbname);
```

All parameters are of type char*. If one is NULL, a default value is used. Usually it is only necessary to specify the database name and a hostname if it's not the local host. The function returns a pointer, which must be used for further access to this connection.

To test the success of a PQsetdb operation the following code can be used:

```
if (PQstatus(conn) == CONNECTION_BAD) {
 printf(stderr,
"Connection to database '%s' failed.\n", dbname);
 fprintf(stderr, "%s", PQerrorMessage(conn));
 PQfinish(conn);
 exit(1);
}
```

PQerrorMessage returns a detailed error message. With PQfinish the connection is terminated and all internal buffers are freed.

After a successful connection, a query can be sent to the database in this way:

```
result = PQexec(conn, "select * from cities");
if ((!result) || (PGRES_TUPLES_OK !=
               PQresultStatus(result))) {
  fprintf(stderr,
"Error sending query.\nDetailed report: %s\n",
PQerrorMessage(conn));
```

```
    PQfinish(conn);
    exit(1);
  }
```

Again, the result of the operation has to be checked, now with the function PQresultStatus which returns different codes dependent on the database operation that is to be performed. In our case, we expect to get some tuples back so we test against this condition. Another possible value is PGRES_COMMAND_OK, which is returned when a query such as INSERT was sent that returns no data. The returning pointer points to a structure within the record data. The structure could be very large. Don't try to access this structure directly; use the functions of the libpq instead, because the internal working methods of the library is subject to change. As a demonstration, here is code that prints out the contents of the cities database:

```
  printf("name               population      altitude\n\n");
  for (i = PQntuples(result)-1; i >= 0; i--) {
   printf("%s        %s              %s\n", PQgetvalue(result,i,0),
   PQgetvalue(result,i,1), PQgetvalue(result,i,2));
  }
```

The function PQgetvalue returns the data as null-terminated ASCII strings, regardless of the field type. To determine the field type, the function PQftype could be used. This function is a weakness of the library, because it returns only the internal coding of the type which is difficult to handle. In order to interpret it correctly, one has to know that all data types are stored as separate tuples in the system database of PostgreSQL. The return value of PQftype is the OID, the unique identifier, of that tuple. To identify the type, you have to query the system database:

```
  -- get the type with internal number 16 (bool)
  select oid, typname from pg_type
  where oid = '16'::oid;
```

Other Programming Interfaces

If you prefer a language other than C, there is a good chance that it is supported by PostgreSQL. The programming techniques are very similar to the C-library; in fact, some object-oriented interfaces make the programming simpler.

One such interface is the libpq++ library, written in C++ and included in the distribution. It has undergone a major revision in the v6.1 release. The class PgEnv is provided for manipulating the environment such as setting the port number for connections and the pathnames for the libraries and databases. The main class is PgDatabase, used to establish a connection and execute querys. It provides very convenient methods to make internal checks for successful operation. There are also classes for accessing large objects (PgLargeObject), for using transaction blocks (PgTransaction) and for defining

cursors (PgCursor). The libpq++ is not built by default, you have to compile it with the provided Makefile.

The Perl5 interface Pgsql_perl5 also must be compiled with the Makefile. It provides two different interfaces. One is nearly identical to libpq so that porting C applications to Perl is easy. The other interface uses the object-oriented features of Perl5. Listing 1 demonstrates the use of that interface. Its purpose is to give a listing of all users who are owners of databases. To run this script, you need to have the proper access rights in the system database. As you can see, with the object-oriented interface, it is not necessary to close the connection explicitly. The module handles this automatically when the connection object is destroyed.

PostgreSQL and the WWW

Together with the Apache web server and some separately available Apache modules, PostgreSQL can be used to serve databases to the Web. One of these modules is mod_auth_pg95, which allows user authentication with PostgreSQL databases. To make it work, you have to install the web server with its own user and group ID rather than with the default “nouser” and “nogroup”. This can be done with the configuration statements **User** and **Group** in the file httpd.conf. It is then necessary to tell PostgreSQL of this account, so that it accepts connections from the web server with **createuser**. In the Apache configuration file access.conf, you must tell the web server where to find the authentication data. The following is a sample configuration:

```
<Directory /DocumentRoot/MySecrets>
Auth_PGhost localhost
Auth_PGport 5432
Auth_PGdatabase www
Auth_PGpwd_table apache_user
Auth_PGuid_field user
Auth_PGpwd_field password
AuthType Basic
AuthName My Secrets
require valid-user
</Directory>
```

Now, create a database named “www” with a table “apache_user” that contains the fields user and password. I suggest you do not create the table under the web server account for security reasons. If you create the table under another account and grant access with the command:

```
grant select on apache_user to apache;
```

then the server can't be used to insert new users or to delete the whole table. To populate the table, you can use the Perl script pg95passwd.pl, which ships with mod_auth_pg95. It also encrypts the passwords for you.

The second Apache module I want to introduce is PHP/FI. It is also available as a stand-alone CGI-Script, but it is more secure to compile it as a module. PHP/FI allows you to embed scripts in your web pages in a similar way to Microsoft's Active Server Pages. It can be configured to allow script access to PostgreSQL. PHP/FI uses its own script language which is easy to learn. The PostgreSQL part is based on libpq. If you know libpq, you should have no problems using PHP/FI for PostgreSQL access. I can't present all the features of PHP/FI, but I will provide an example script in Listing 2, which reads the contents of the cities table and puts it into an HTML table environment.

PHP/FI provides automatic conversion for some data types. Unlike the C interface, table values are returned in their proper format for the types integer, boolean, oid, float and real. Arrays are always returned as strings.

When you compile PHP/FI for use with PostgreSQL, defining the compiler switch MAGIC_QUOTES in php.h makes life easier. This option causes all single and double quote characters in GET and POST data from HTML forms to be automatically escaped.

The Further Development of PostgreSQL

PostgreSQL is now developed by a couple of volunteers, who coordinate their efforts via the Internet. A mailing list is used for discussion of implementation details. At the FTP site (ftp://ftp.postgresql.org/), a **tar** archive file with the latest sources is provided every night. The programmers use the FreeBSD utility **sup** to synchronize their source trees. There is also a documentation project where the man pages and the user guide are

It is planned to achieve full SQL92 compliance in the future and to constantly speed up the database operation. Version 6.2.1, which was released in October 1997 brings PostgreSQL a big step nearer to this goal. Now default values and constraints can be specified at table creation, to check newly inserted data for specific conditions. It is also possible to write trigger functions, that are executed whenever a row of table data is selected, inserted or updated. There are also new string functions like trim() substring() and position() which makes manipulating of strings very convenient. A new Server Programming Interface gives the users the aility to write server stored procedures and to implement integrity checks with triggers.

Compared to the widely used mSQL, PostgreSQL is much slower as long as only simple queries are involved. But for queries that include complex joins of multiple tables, PostgreSQL should be faster. mSQL is written as a program that gives programmers or webmasters a small and fast tool for simple database operations. In contrast to this, PostgreSQL is a full featured database system. It

provides more data types, better extensibility and, with the two available query tools psql and mpsql, more user friendliness.

There are commercial database systems which are available for free as personal versions, e.g., Solid and Yard-SQL. These systems are fully SQL92 compliant and, in comparison to them, PostgreSQL's functionality is limited. The features that make PostgreSQL stand out among these competitors are that it is freely availabe and anyone who is interested can participate in its further development.

Resources

Rolf Herzog was fascinated by computers since his first steps with the early TRS-80. He came to Linux with kernel version 1.2.3, when searching for an alternative to Windows. Now he is working as a computer consultant for the Steinheim Institute in Duisburg, Germany. He is pursuing a degree in Political Science as the theory of social systems stimulates him as much as computer systems. He can be reached via e-mail at rolf@culthea.gun.de.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Qddb Database Suite

Eric H. Herrin, II

Gilbert J. Benson, Jr.

Issue #46, February 1998

An introduction the freely available database suite called Qddb.

Have you ever needed a program to manage some data? Perhaps you have tried one of the many relational database products that require a significant amount of effort to set up a few simple tables and wondered why there isn't one that is easy to use.

When looking at the design of the available database products, notice that most of the tools are very cumbersome. You must carefully define your fields, select those that require a separate table, define link fields between the tables, choose the important fields for searching, then finally build some sort of interface. In many cases, people decide to use flat files because a real database is too time consuming. There are many tasks that can benefit greatly from a program that makes it easy to design simple databases.

What is Qddb?

Qddb is a multi-user database suite that allows you to quickly design custom databases. If you know the data your database should contain, you can design your database and be up and running almost as fast as you can type the field names. Skeptical? Let's work through an example.

There are two basic steps to building a Qddb database. First, you must create a blank database with Qddb's **qnewdb()** command:

```
bash$ qnewdb Properties
```

qnewdb() builds a new database directory (named Properties in this case) and populates it with several files for Qddb's use. Next, you must edit the file Properties/Schema and define the format of the individual records:

```
# This is a comment
# Qddb Schema file for a landlord's property
# database
 Street City State ZipCode
 Apartments (
 AptNumber type integer
 Rented defaultvalue "N"
 Lease (
 Begin type date # start date of lease
 End type date  # end date of lease
 )
 Person (
 First Middle Last ID
 PrimaryRenter defaultvalue "Y"
 )*
 Comments*
 )*
```

The first thing you notice about the Schema file is its free-style format. Each word (other than the keywords **type** and **defaultvalue** followed by a data type or string) defines a field in a record. By default, each field's value in a record may contain an arbitrary-length string. You can also explicitly assign data types of **string**, **real**, **integer** and **date** to each field.

Some fields are *structured*, that is, they contain other fields. For example, Apartments contains the subfields AptNumber, Rented, Lease, Person and Comments. Apartments.Lease and Apartments.Person are also structured and contain subfields.

Some fields are *expandable*, meaning each record in the database can contain multiple values for that field. Expandable fields are denoted by an asterisk (*) at the end of its definition.

The sample Schema above defines a database that contains one record for each property owned by a landlord. Each property contains multiple apartments. Each apartment has a number, whether it is rented, lease dates, a list of people living there and a list of comments.

nxqddb()--Qddb's Generic Database Browser

Now, you are ready to use your Properties database with Qddb's generic X-based application **nxqddb**.

```
nxqddb Properties
```

Figure 1. Standard nxqddb Screen

Figure 1 shows the standard nxqddb screen for the Properties database. Once started, nxqddb() allows you to:

- Add new records
- Search for records and view the results

- Modify records
- Delete records
- Generate letters, postcards, e-mail, reports, barcharts and graphs

To add records, select the Modes menu button and choose Add Mode. Once in Add Mode, you can simply fill in the fields. If you want to add multiple values for an expandable field, you click the mouse on the Add button next to the field's label. You can also view all the values in an expandable field by clicking the View button. When you are ready to save a record, select the File menu button and choose Save. Now you are ready to add another record. Note that Qddb does not clear the entry boxes for the next record by default; you can choose *auto-clear* under the configuration menu if that is the behavior you want.

Once you have added some records, you can go to Search Mode and perform simple, fast database queries. For example, let's say you want to find all the people renting apartments whose first name is Sally. Just type “sally” in the Apartments.Person.First entry box and press **return**. If you want to confine your search further, to the last name “Jones”, type “jones” in the Apartments.Person.Last field and press **return**. A list of all the properties matching the criteria will appear in a Search Results window as shown in Figure 2. If you want to edit one of the records, click the mouse on the entry and you will be switched to Change Mode for that record.

Figure 2. Search Results Window

You will quickly find that Qddb, by default, splits field values into searchable components. A *separator* is a character that separates the searchable components in a field. You can specify your own set of separators for any field in the schema:

```
MyField separators ",.;"
```

The separators schema option above tells Qddb to index the field MyField using only the separators “,” (comma), “.”(period) and “;” (semicolon). The default list of separators includes all ASCII non-alphanumeric characters. (Note that only fields of type **string** use separators.)

Qddb's simple search mechanism supports ranges, regular expressions, numbers and dates. A fully functional expert search and report generator round out the searching facilities of Qddb.

How to Configure the Search Results

Now, that you know how to perform simple searches, you may want to add or omit certain fields from your results. By default, nxqddb displays only the first five fields in the search results window.

If you click the Configure menu button, you will notice many configuration options. This is the current state of all your nxqddb settings. The search results configuration (see Figure 3) determines what fields are present in the search results, how those fields are displayed, and in what order. You can also add user-defined columns to the search results.

Figure 3. Search Results Configuration Window

After configuring the search results, you may choose to save the configuration. **nxqddb** recognizes two configurations on startup: the global configuration and the personal configuration. The global configuration is the default for any user of a particular database. The personal configuration overrides any global settings for an individual user.

Expert Search

While the simple search mechanism is extremely handy for quick lookups, it cannot perform all types of searches. The expert search is designed to handle generic searches and includes the ability to save the search criteria for later use. Figure 4 shows the expert search window. From this window, you can combine and nest queries in any way you like. Ranges use two entry boxes; other search types use only the leftmost entry.

Figure 4. Expert Search Window

Suppose you want to find all the primary renters whose lease expires this month. The idea is to provide a list of the search criteria to the expert search engine. Our search then should be all Apartments.Person.PrimaryRenter == "Y" and Apartments.Lease.End == "@firstofmonth(this month)-@lastofmonth(this month)".

The expert search window always contains at least one field for search criteria. To restrict our search to the particular fields specified above, we click the down arrow next to the entry box, choose Apartments.Person.PrimaryRenter as the attribute name and dismiss the pull-down menu, then type **Y** in the entry. Now we have to specify a second criterion, so we:

1. Click the Append Node button to add a new entry.

2. Click the down arrow to select a date range search on the Apartments.Lease.End attribute.
3. Dismiss the menu.
4. Type **@firstofmonth(this month)** in the left entry and **@lastofmonth(this month)** in the right entry.

Now, we can click the Search button, and we will see a search results window that contains all the matching rows using the current search results format.

Generating Reports

Ad-hoc reports can be easily produced by executing these two steps:

1. Configuring the search results.
2. Performing a simple or expert search.

More detailed reports require the Qddb report generator. The Qddb report generator can produce postcards, letters, e-mail, summaries of numeric fields, graphs, bar charts and tables. The basic idea is the same as producing an ad-hoc report, but you generally want to save the settings for both the search criteria and results format for later use.

Figure 5. Report Generator's Window

Figure 5 shows the report generator's main window. From this window you can define multiple expert searches and a search results configuration. After defining what you want to see and how you want to see it, you can choose any combination of the six output formats.

For example, suppose we want to send a postcard to each renter whose lease expires this month. First, we need to configure the format of our return and destination addresses. Choose the Report Format Defaults option under the Configure menu button. You will notice two options in the cascaded menu: Return Address and Destination Address. Choose Return Address and fill in the return address as you want it to appear on postcards and letters. Next, choose Destination Address. This option will format arbitrary fields for the destination address on all postcards and letters. Once you have configured the addresses, you may wish to save your configuration so these options become the default.

We have already seen how to perform the expert search, so let's just use the previous example for defining what you want to see in your report. Next, we have to define the relevant fields for the report via the Results Format button. To generate a postcard, we must include all address information, plus we probably want to include the lease's expiration date for inclusion in the message. Now, we can choose the type of reports we wish to generate. If you

want a listing as well as postcards, you can choose both options and click the Run Report button. Since we chose listing as well as postcards, the listing appears first and can be printed out. After printing the listing, we are prompted for the postcard message as shown in Figure 6. Notice that the postcard message is personalized for the recipient by using data from each record.

Figure 6. Postcard Window

Qddb and the Relational Model

In the simple case, Qddb uses a single set of *tuple trees* instead of flat tables. A tuple tree is simply the collection of prejoined rows from the relational tables in a one-many relationship. That is, each record represents some entity, and each entity can have an arbitrary number of values for some subset of the fields. More complex cases (such as many-many relationships) can be handled with a set of Qddb schemas.

We can use standard relational tables to model any Qddb schema. For example, our Properties schema requires four tables: one table for the address information and one table for each expandable field. The tables require a plethora of link information for the purpose of joining the rows.

It is the complexity of managing these tables that prevents normal users from building their own databases with standard relational tools. Even programmers avoid fields that can have multiple values because a separate table is required for each such field. With Qddb, users and programmers can allow multiple values for any field by simply appending an asterisk to the field's definition in the schema.

The relational model requires that field values be *atomic*. That is, when searching, field values cannot be broken into smaller searchable components. By default, Qddb disobeys this rule by breaking field values of type **string** at the separators. Most users do not care about the relational model, they simply want to do their work. Sometimes this work involves searching the database for a particular record. By relaxing this restriction, Qddb can search for words in textual fields such as comments or abstracts. For string fields where atomicity is important, you can specify an empty list of separators for that particular field.

Storage Format and Indexing

Qddb uses several interesting storage techniques. If you look at the contents of the files in your database directory, you will see that all Qddb data and index files are stored as readable text. The Database file contains the field values for each record in the stable part of the database. Empty fields require no storage and are omitted from the Database file.

If you browse the Database file, you will also notice that each record is stored contiguously. Qddb's records are prejoined rows from the relational tables defined in the schema. When you perform a search, the matching tuple trees in the Database file are expanded into the equivalent relational rows for viewing purposes.

Qddb uses inverted indices for searching. When you perform a search, the criteria are quickly translated into the file offsets for the corresponding tuple trees. Each matching tuple tree can then be read from the disk with one disk read per tuple tree.

Since Qddb currently uses inverted indices for indexing, you should periodically reindex the database so Qddb can retain its speed. Only changes and additions are stored in a secondary location and can be reindexed on the fly. The **qstall** command is used for this purpose:

```
qstall Properties
```

After stabilization, all changes, additions, and deletions are committed to the Database file.

History and More Information

Resources

Eric Herrin was a co-creator of the first release of Qddb in 1989 and has since guided its development. He holds a Ph.D. in computer science from the University of Kentucky and his current research interests include operating systems and information retrieval. Dr. Herrin can be reached via e-mail at eric@hsdi.com.

Gil Benson is a software developer and holds a B.S. in computer science from the University of Kentucky and an MBA from Xavier University. He has been using Qddb since its inception in 1989 and can be reached via e-mail at gil@hsdi.com.

Beagle SQL, A Client/Server Database for Linux

Rob Klein

Issue #46, February 1998

Mr. Klein introduces us to a database called Beagle SQL that he developed as a learning experience.

I'd like to introduce you to a client-server database package that I've been developing on Linux since May 1996. I picked Linux because it is one of the most robust development environments available for the development of advanced client-server applications. Beagle SQL began life as a learning project. I've always had a fascination with figuring out how things work under the hood. After years of working with many major (and minor) database packages, I decided to build my own. Since then I've received a lot of comments from people who are looking for more choices in database packages, particularly ones that support Linux. I have benefited greatly from the hard work so many people have contributed to the development of Linux and other freely available tools. Beagle SQL (BSQL) is my way of giving something back.

Basic Architecture

Database management systems come in many different forms. Here I'll discuss the client/server architecture of BSQL. The three base components of this architecture are the client process, the Connection Manager and the Database Manager.

The client process is the user application that sends requests to the Database Manager. The client is simply a program written using the available API (Application Programming Interface) provided by the DBMS to access the data in the database. BSQL comes with both C and Perl APIs.

The Connection Manager handles all incoming connections to the Database Manager. When the client program issues a connect request, the Connection Manager spawns a copy of the Database Manager to handle all subsequent requests from the client. Once the client and the Database Manager are talking,

the Connection Manager is free to handle connection requests from other clients.

In this scenario, each client process is serviced by its own server process on the remote machine. One advantage of this type of architecture is that the server process only needs to worry about its own client, making the communication between the client and server processes easy to handle. Unfortunately, this architecture is memory-intensive as a server process is spawned for each client.

Another disadvantage is that the locking algorithms become more complicated as each server process needs to be aware of the other server processes updating the database. Database management systems typically incorporate one of two locking methods, coarse locking and fine locking.

Coarse locking, also known as table-level locking, is the easiest of the two to implement. It requires that each client process writing to a table requests a lock be placed on the entire table and its associated indices. Once the database manager grants this lock, the client process has permission to write to the table. Any other clients needing to write to the same table must wait until the first client is done. Typically, the duration of the lock is handled using a transaction. In its simplest form, a transaction would be a single UPDATE or DELETE statement. Some database managers give the client the ability to expand the size of a transaction using a keyword to block several statements together. This can be very critical in systems where data is replicated throughout several tables in a database. The main problem with this type of locking is the bottleneck that can be created when several clients are trying to update the same table at the same time.

Fine locking, also known as row-level locking, is much more complicated to implement. When a client writes to a table, it requests a lock only for the row in the table it is currently updating. This allows several clients to update the same table at the same time as long as they are not trying to lock the same row. The complexity comes in when a client is trying to update a table that has indices associated with it. BSQL uses an indexing method known as B-trees. Whenever a client updates, deletes or inserts rows into a table, the B-tree indices for the table may need to be re-balanced. Concurrent balancing of B-trees is way beyond the scope of this article, but there have been many books and papers dedicated to the topic.

Currently, BSQL uses the client**->**connection**->**database architecture without locking. I plan to implement coarse locking first, eventually evolving into fine locking as time allows.

Client Process

The client process is usually a user-written program that accesses the database using the provided API, in this case, BSQL's C API. For those who prefer Perl, a demo client with full Perl source is provided with the BSQL distribution downloadable from http://www.beaglesql.org/. The first thing the client program needs to do is request a connection to the server process using the API function BSQLConnect(). The connect function returns the file handle needed by all subsequent communications with the server. Next, the database you want to manipulate is set using the BSQLSetCurrentDB() function call, passing the file handle returned by BSQLConnect() and the name of the database to which you wish to connect. The following code example illustrates how a client process connects to a server process running on the same machine:

```
s = BSQLConnect (host);
if (!BSQLSetCurrentDB (s, "test")) {
   fprintf (stdout,
      "\nCan't send current database");
   exit (s);
}
```

Once you are connected to the database, you can begin sending SQL queries using the BSQLQueryDB() function, passing the file handle assigned to the connection and a string containing your SQL query. A pointer to a result structure is returned that contains the status of your request. Status information includes whether or not the query succeeded and, in the case of an SQL SELECT, how many records or tuples returned to the client process. The code fragment in Listing 1 shows the results of a SELECT statement sent to the Database Manager.

In the above example, the BSQLnfields() function returns the number of fields per record returned by the SELECT statement. The BSQLFieldName() function returns a string containing the field name of the *n*th field returned. The function BSQLntuples() returns the number of records that match the SELECT's WHERE clause. The omission of the WHERE clause in the above example tells the server process to return all records from the table phone book. The call to the BSQLFieldValue() function returns a string containing the data from the *n*th field of the *i*th record. Because the result structure returned by the BSQLQueryDB() function is dynamically allocated, it must be freed after you are finished with it. The BSQLFreeResult() function does just that. The last BSQL API function called in any client program should be the BSQLDisconnect() function. When called, it passes an exit message to the server process so it can terminate the connection and exit cleanly. Without it, you will litter your system with stray server processes that eat up system resources.

Connection Manager

This is probably the most straightforward piece of the client-server puzzle. The Connection Manager is simply a loop waiting for incoming messages from client processes. First a socket is opened for the “beagled” service (defined in your /etc/services file), so that the Connection Manager can listen for incoming connections. Then an endless loop is entered. Once the Connection Manager receives a signal from a client processes, a call to accept() returns the socket number that the client and server processes will communicate through. At this point, the Connection Manager fork()s the Database Manager, passing the socket number returned by accept(). After the Database Manager is successfully started, the Connection Manager starts listening for the next incoming connection.

Database Manager

The Database Manager does all the work. The basic components of the Database Manager are the expression parser, the query optimizer (currently, no query optimization is done in BSQL), the index manager, the locking manager and the low-level I/O manager. The SELECT statement is the most complex operation performed by the Database Manager. As BSQL supports explicit joins, a single SELECT statement can search through several tables to return the requested information. The expression parser must be intelligent enough to tell which fields in the SELECT list belong to which tables. If you are joining two tables with duplicate field names, the SELECT statement must explicitly state which field belongs to which tables. Wild cards are allowed. When the expression parser sees wild cards in the field list, it inserts the appropriate field names into the list.

There are three examples in the sidebar to give you an idea of how the expression parser does its work. The statement in Example 3 fails because **field1** is ambiguous. The expression parser can't tell if it belongs to table A or table B as both have a field called **field1**.

Examples

When joining tables, the SELECT statement's WHERE clause can contain several different parts that need to be treated separately. When joining two tables these parts can include the conditionals for the first table, the conditionals for the second table and the join condition. The Database Manager searches each of the two tables using the appropriate conditionals from the WHERE clause. Next, it joins the two tables into a temporary table using the fields in the SELECT field list as well as the fields used in the join condition. Last, the records in the temporary table are matched with the join condition and the appropriate records are made available for retrieval by the client process.

This operation can get quite complicated and time consuming when dealing with large and multiple tables. This is where the query optimizer comes in. Its purpose is to determine the most efficient order to search and join the tables. BSQL currently doesn't do query optimization and joins the tables from left to right as they appear in the SELECT statement. This leaves it up to the person writing the SQL statement to put some thought into the order the tables appear in the join.

When performing searches, the Database Manager uses a set of low-level I/O routines to retrieve records from the database. Most commercial database vendors use proprietary file systems to house their databases. In the case of BSQL, the Linux file system sufficed. A future enhancement will be a flexible file format that can allow for such things as BLOBs, images, text documents and anything else. (A BLOB is a large binary datatype used to store images, sound bites, programs, etc. in a database table.)

The method used to store these variable-length records will significantly impact the performance of the Database Manager. When a record is written to the database, it is broken down into fixed-size segments. The database administrator can set the size of these segments for each database. If a record containing 850 bytes is written to a table that uses 256 byte segments, it is broken down into four segments that are chained together. If at a later time the record size is changed to 1200 bytes, an additional segment is added to the chain. If the record is reduced to 700 bytes, the unneeded segments are marked for reuse. One drawback here is that over time the database can become fragmented. Routine maintenance using a de-fragmenting utility should be performed on databases that see a lot of UPDATEs and DELETEs. This utility will be provided with the first official release of BSQL v1.0.

Conclusion

Hopefully, I've given you some insight into how client server databases work and the many late nights that go into their development. For more information on Beagle SQL, point your web browser to http://www.beaglesql.org/. Here you can follow its development history from day one to present as well as download the code to try it out for yourself. Also, be sure to look into the other freely available database resources on the Web.

Rob Klein has been a Developer/Administrator for 11 years. Although software development can be fun, his main interests are football, baseball and spending quiet evenings with this wife Cathy (absolutely not in that order). He welcomes your comments sent to rvklein@ober.com.

Archive Index Issue Table of Contents

Advanced search

Portable Database Management with /rdb

Ed Petron

Issue #46, February 1998

Web server analysis logs and mailing list management is made easy by using the /rdb database system—here's how to do it.

The /rdb database system from Revolutionary Software is a compact, low-cost relational database management system that has been available for Linux and other Unix variants for several years. It distinguishes itself from most (if not all) other Unix database systems by functioning as an extension of the Unix shell.

The Unix Approach to Application Design

The typical Unix distribution contains a rich set of shell commands (often referred to as *filters*) such as **awk**, **grep** and **sed** that operate primarily on text data. In addition, *pipes* can be used to route the output from one command into the input of another command. These commands can either be standard commands like those mentioned above or be created by the user. This system of pipes and filters simplifies the development of complex applications by enabling the reuse of components that have already been tested and proven.

Most filters are designed to operate on ASCII text data, so it is natural to represent logical data records as ASCII text lines. Data elements within the records are separated by a field delimiter character (such as a colon or tab) specified by the application. Alphanumeric data elements such as names, addresses and descriptions can be as long or as short as needed. Numeric data elements can have unlimited magnitude and precision, and their representation is architecture independent. This scheme provides complete portability of data between platforms.

Other Approaches

Approaches to application design inherited from other platforms typically use fixed-length fields with predetermined field lengths. These lengths result in

wasted file space due to the unpredictability of an application's actual storage requirements. Numeric data is often stored in binary format which makes it highly platform specific and nonportable. Data elements such as “description” or “comment” fields can vary greatly in size, and the use of fixed-length fields forces the application designer to either allocate enough storage to cover all cases (often impossible) or to dedicate separate data files to their storage (often making application logic clumsy, complex and difficult to manage).

The result of these approaches is the allocation of disk space that is never used and application programs that are difficult to maintain. Although recent innovations such as SQL have simplified the problem of data access from applications, many programs are tightly bound to storage layouts determined at compile time. This binding of code to data forces recompilation and relinking of application code each time a change in record format is required. Worse yet, many applications use “filler fields” to avoid recompilation, wasting even more disk space and I/O bandwidth.

Application Design Using /rdb

While a great many Unix systems (database systems included) use the approach outlined in the last section, /rdb augments the capabilities already provided by standard Unix shell commands. Many /rdb commands operate as front ends for standard Unix commands such as awk. This enables users already familiar with Unix to create /rdb applications without learning a new programming language.

Table Format

The format of /rdb tables is extremely simple. They are ASCII text files in which a row is a line of data. The column elements within each row are delimited by a column separation character (a tab by default but any other character except the newline can be used). /rdb commands use a **-f** command line switch to specify a column separator in cases where it is not a tab. Note the similarity between this approach and typical Unix system configuration data files such as /etc/passwd. The only difference is that the first line of /rdb tables is a delimited set of field names and the second line is a line of dashes. These first two lines are known as a header.

Part of the beauty of /rdb tables is that, for some applications, they need not even be stored in disk files. Instead, they can be passed between commands using pipes. These commands can be /rdb commands or standard Unix shell commands. The /rdb commands are designed to output tables through **stdout**, and most accept input tables through **stdin**. Although some standard Unix commands will be “confused” by the header lines, /rdb provides commands for removing and reattaching these headers (**headoff** and **headon**, respectively).

/rdb Filters

Most /rdb commands operate on tables that are read through stdin. The resulting table is written to stdout. Here are a few examples:

```
row 'Cost < 50' inventory
```

selects the rows in the inventory table where the Cost field is less than 50.

```
column 'Description' < inventory
```

reads the inventory table and outputs a table containing only the column Description.

```
addcol 'Value' < inventory
```

reads inventory and outputs a new table that is identical to inventory except for the addition of a new column called Value. At this point, all of the fields in the Value column are initialized to empty strings.

```
compute 'Value = Cost * Quantity'
```

reads an input table and, on each row, sets Value to Cost * Quantity. If Value is a column name, the computed value is stored in the table; otherwise, it could still be referenced in other statements within the same compute statement.

Putting these commands together, the Bourne shell script in Listing 1 will output a table describing inventory items which cost less than $50 US and showing the value of each.

These commands are only a few of the filters that /rdb provides. For a complete listing, see the on-line man pages listed in Resources.

The row and compute filters are actually front ends for awk. Since awk doesn't understand field names as defined by /rdb, these commands convert field names into column numbers which awk does understand. As a result, the user who is familiar with awk can immediately use these commands and many others provided by /rdb.

/rdb Table Editing

Because /rdb tables are ASCII files, they can be created and maintained by any text editor. This will work in many cases but will be awkward in others, in particular, with wide tables or those containing a large number of columns. /rdb provides two table editors, **ve** and **jve** for this purpose. A complete

description of these editors is beyond the scope of this article, but the following is a partial list of the capabilities provided:

- **Forms based data entry:** Screen files, ASCII files describing data-entry screen layout, can be specified using the **-s** option and are used to associate screen field positions with table column names and to provide initialization and write protection for certain fields.
- **Data validation:** The **-v** option is used to specify a validation file to provide data edits based on allowed and disallowed ranges of characters, column length limits, table lookup and command-based validation. With command-based validation, the values can be passed to any arbitrary command or shell script. The exit status indicates to jve whether or not the data is considered valid (zero for success, non-zero for failure). If a non-zero status is received, jve will beep and display the first line of standard error at the bottom of the screen.
- **Audit file creation and maintenance:** An audit file tracks changes to a table and can be used by the **rollback** command to restore a table to a specific point in time.

Web Server Log Analysis

In order to illustrate how /rdb operates in conjunction with other Unix shell commands, the script in Listing 2 shows how a log file can be analyzed to report a server's top five accessors. This script assumes that stdin is a log file with space delimited fields with the host name as the first field (this is typical for Apache and several other web servers). The following is a step by step description of the process:

1. The combination of awk and the **header** command produces a table with host names in order of their appearance in the log file. The output from awk is not in /rdb table format, so header is used to complete the production of the table.
2. The **addcol** command followed by compute is used to add a column named n and initialize its value to 1.
3. The table is then sorted by host and the n values are totaled using host as a control break to produce a table of the host names and the total access counts for each, i.e., the value of n.
4. Since we are interested in seeing the five hosts with the highest access counts, the table produced in step 3 is sorted by n.
5. The top seven lines from the table in step 4 (two header lines plus five rows) are filtered out by the **head** command and passed to the **justify** command which vertically aligns the header and columns and produces a format better suited to printing.

The script in Listing 2 produces the following output:

```
host n
wilbur.leba.net 106
gateway.amat.com        24
208.219.77.9    20
ras3.fsxnet.com 14
ohnp6.m50.nrc.ca        14
```

Actually, it wasn't necessary to create and store any /rdb tables.

Mailing List Processing

Now we look at a slightly more complex example: the generation of mailing labels. Assuming that stdin is an /rdb table with fields named **Salut** (Mr., Mrs., etc.), **Fname**, **Lname**, **Address1**, **Address2** (sometimes blank), **City**, **State** and **Zip**, we create the script shown in Listing 3.

For each row in the table, this script creates a mailing label with either three or four lines (depending on whether or not the second address line was used). After adding columns to represent the lines in our final output, the compute command determines their contents. The resulting table is sorted by zip code so that the labels are grouped for bulk mailing purposes. The **report** command formats the output using the report.frm file shown here:

```
<Line1>
<Line2>
<Line3>
<Line4>
```

This file is a very simple example of a report form file used by /rdb to generate reports. A field name within brackets indicates the substitution of the contents of the named field.

Other report file options are available as well, although we won't describe them here. One of the most significant options is the placement of output from shell commands. In this example, the output from report is a series of labels in a single column. This data is piped into **pr** which arranges it into two columns.

Other /rdb Features

There are five access methods provided by /rdb: *sequential* (used by the above examples), *record*, *binary*, *inverted* and *hashed*. An **index** command is provided to build index files, and a **search** command is used to execute keyed retrievals.

A set of library functions, librdb.a, is provided for use by those who require access through C programs. Complete C source code for these functions is also included.

Conclusion

/rdb is a powerful but low cost package suitable for a large variety of database management tasks. It offers the following benefits:

- **Low cost**: Only $149US for Linux, SCO and other Intel Unix implementations, $495US for RISC platforms.
- **Short learning curve**: Users already familiar with Unix become proficient quickly.
- **Low Hardware Requirements**: /rdb requires under 5MB of disc space to install. /rdb commands are compact since they function mostly as front ends for standard Unix tools. The mailing list application presented above is a “real world” program that was previously done using a dBase file. Since the dBase file used fixed-length fields, they contained a lot of trailing blanks which were eliminated when converted to an /rdb table. As a result, the /rdb table was roughly one third the size of the original dBase file.
- **Integrates well with other Unix commands**: This point is illustrated in the web server log analysis example. One could easily create generic interface programs for other systems (including other databases) and using /rdb to provide data entry and validation on the input side and report generation on the output side.
- **Portability of data and code**: This is becoming increasingly important as networked computing environments become more common.

The only disadvantage is that platform-independent systems generally run slower than those that are machine specific. Also, variable- length records tend to be more difficult to manage than fixed-length ones. Even so, /rdb performs well even with large databases (see Resources for pointers to demos available on the Web). Even in the extreme cases where a more traditional database system may be needed, /rdb can still be used as a front end as shown in Listing 2.

Finally, the economic considerations involved in the hardware independence versus performance tradeoff are clear. Hardware price/performance ratios are dropping rapidly while the cost of programmer time is constantly rising. This being the case, hardware independence is the uncontested winner.

Resources

Ed Petron is a computer consultant interested in heterogeneous computing. He holds a Bachelor of Music from Indiana University and a Bachelor of Science in computer science from Chapman College. His home page, The Technical and Network Computing Home Page at http://www.leba.net/~epetron, is dedicated to Linux, The X Window System, heterogeneous computing and free software. Ed can be reached via e-mail at epetron@wilbur.leba.net.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Linux Network Programming, Part 1

Ivan Griffin

PhD. John Nelson

Issue #46, February 1998

This is the first of a series of articles about how to devlop networked applications using the various interfaces available on Linux.

Like most other Unix-based operating systems, Linux supports TCP/IP as its native network transport. In this series, we will assume you are fairly familiar with C programming on Linux and with Linux topics such as signals, forking, etc.

This article is a basic introduction to using the *BSD socket interface* for creating networked applications. In the next article, we will deal with issues involved in creating (network) daemon processes. Future articles will cover using remote procedure calls and developing with CORBA/distributed objects.

Brief Introduction to TCP/IP

The TCP/IP suite of protocols allows two applications, running on either the same or separate computers connected by a network, to communicate. It was specifically designed to tolerate an unreliable network. TCP/IP allows two basic modes of operation—connection-oriented, reliable transmission and connectionless, unreliable transmission (TCP and UDP respectively). Figure 1 illustrates the distinct protocol layers in the TCP/IP suite stack.

Figure 1. TCP/IP Protocol Layers

TCP provides sequenced, reliable, bi-directional, connection-based bytestreams with transparent retransmission. In English, TCP breaks your messages up into chunks (not greater in size than 64KB) and ensures that all the chunks get to the destination without error and in the correct order. Being connection-based, a virtual connection has to be set up between one network entity and the other

before they can communicate. UDP provides (very fast) connectionless, unreliable transfer of messages (of a fixed maximum length).

To allow applications to communicate with each other, either on the same machine (using loopback) or across different hosts, each application must be individually addressable.

TCP/IP addresses consist of two parts—an IP address to identify the machine and a port number to identify particular applications running on that machine.

The addresses are normally given in either the "dotted-quad" notation (i.e., **127.0.0.1**) or as a host name (**foobar.bundy.org**). The system can use either the /etc/hosts file or the *Domain Name Service* (DNS) (if available) to translate host names to host addresses.

Port numbers range from 1 upwards. Ports between 1 and IPPORT_RESERVED (defined in /usr/include/netinet/in.h—typically 1024) are reserved for system use (i.e., you must be root to create a server to bind to these ports).

The simplest network applications follow the *client-server* model. A server process waits for a client process to connect to it. When the connection is established, the server performs some task on behalf of the client and then usually the connection is broken.

Using the BSD Socket Interface

The most popular method of TCP/IP programming is to use the *BSD socket interface*. With this, network endpoints (IP address and port number) are represented as *sockets*.

The socket interprocess communication (IPC) facilities (introduced with 4.2BSD) were designed to allow network-based applications to be constructed independently of the underlying communication facilities.

Creating a Server Application

To create a server application using the BSD interface, you must follow these steps:

1. Create a new socket by typing: **socket()**.
2. *bind* an address (IP address and port number) to the socket by typing: **bind**. This step identifies the server so that the client knows where to go.
3. *listen* for new connection requests on the socket by typing: **listen()**.
4. *accept* new connections by typing: **accept()**.

Often, the servicing of a request on behalf of a client may take a considerable length of time. It would be more efficient in such a case to accept and deal with new connections while a request is being processed. The most common way of doing this is for the server to *fork* a new copy of itself after accepting the new connection.

Figure 2. Representation of Client/Server Code

The code example in Listing 1 shows how servers are implemented in C. The program expects to be called with only one command-line argument: the port number to bind to. It then creates a new socket to listen on using the **socket()** system call. This call takes three parameters: the *domain* in which to listen to, the *socket type* and the *network protocol*.

The domain can be either the PF_UNIX domain (i.e., internal to the local machine only) or the PF_INET (i.e., all requests from the Internet). The socket type specifies the communication semantics of the connection. While a few types of sockets have been specified, in practice, SOCK_STREAM and SOCK_DGRAM are the most popular implementations. SOCK_STREAM provides for TCP reliable connection-oriented communications, SOCK_DGRAM for UDP connectionless communication.

The *protocol* parameter identfies the particular protocol to be used with the socket. While multiple protocols may exist within a given protocol family (or domain), there is generally only one. For TCP this is IPPROTO_TCP, for UDP it is IPPROTO_UDP. You do not have to explicitly specify this parameter when making the function call. Instead, using a value of 0 will select the default protocol.

Once the socket is created, its operation can be tweaked by means of socket options. In the above example, the socket is set to reuse old addresses (i.e., IP address + port numbers) without waiting for the required connection close timeout. If this were not set, you would have to wait four minutes in the TIME_WAIT state before using the address again. The four minutes comes from 2 * MSL. The recommended value for MSL, from RFC 1337, is 120 seconds. Linux uses 60 seconds, BSD implementations normally use around 30 seconds.

The socket can linger to ensure that all data is read, once one end closes. This option is turned on in the code. The structure of **linger** is defined in /usr/include/linux/socket.h. It looks like this:

```
struct linger
{
        int l_onoff;   /* Linger active */
        int l_linger;  /* How long to linger */
};
```

If **l_onoff** is zero, lingering is disabled. If it is non-zero, lingering is enabled for the socket. The **l_linger** field specifies the linger time in seconds.

The server then tries to discover its own host name. I could have used the **gethostname()** call, but the use of this function is deprecated in *SVR4 Unix* (i.e., Sun's Solaris, SCO Unixware and buddies), so the local function **_GetHostName()** provides a more portable solution.

Once the host name is established, the server constructs an address for the socket by trying to resolve the host name to an Internet domain address, using the **gethostbyname()** call. The server's IP address could instead be set to INADDR_ANY to allow a client to contact the server on any of its IP addresses—used, for example, with a machine with multiple network cards or multiple addresses per network card.

After an address is created, it is bound to the socket. The socket can now be used to listen for new connections. The BACK_LOG specifies the maximum size of the listen queue for pending connections. If a connection request arrives when the listen queue is full, it will fail with a connection refused error. [This forms the basis for one type of denial of service attack —Ed.] See sidebar on TCP **listen()** Backlog.

Having indicated a willingness to listen to new connection requests, the socket then prepares to accept the requests and service them. The example code achieves this using an infinite **for()** loop. Once a connection has been accepted, the server can ascertain the address of the client for logging or other purposes. It then forks a child copy of itself to handle the request while it (the parent) continues listening for and accepting new requests.

The child process can use the **read()** and **write()** system calls on this connection to communicate with the client. It is also possible to use the buffered I/O on these connections (e.g., **fprint()**) as long as you remember to **fflush()** the output when necessary. Alternatively, you can disable buffering altogether for the process (see the **setvbuf() (3)** man page).

As you can see from the code, it is quite common (and good practice) for the child processes to close the inherited parent-socket file descriptor, and for the parent to close the child-socket descriptor when using this simple forking model.

Creating the Corresponding Client

The client code, shown in Listing 2, is a little simpler than the corresponding server code. To start the client, you must provide two command-line arguments: the host name or address of the machine the server is running on

and the port number the server is bound to. Obviously, the server must be running before any client can connect to it.

In the client example (Listing 2), a socket is created like before. The first command-line argument is first assumed to be a host name for the purposes of finding the server's address. If this fails, it is then assumed to be a dotted-quad IP address. If this also fails, the client cannot resolve the server's address and will not be able to contact it.

Having located the server, an address structure is created for the client socket. No explicit call to **bind()** is needed here, as the **connect()** call handles all of this.

Once the connect() returns successfully, a duplex connection has been established. Like the server, the client can now use read() and write() calls to receive data on the connection.

Be aware of the following points when sending data over a socket connection:

- Sending text is usually fine. Remember that different systems can have different conventions for the end of line (i.e., Unix is **\012**, whereas Microsoft uses **\015\012**).
- Different architectures may use different byte-ordering for integers etc. Thankfully, the BSD guys thought of this problem already. There are routines (**htons** and **nstoh** for short integers, **htonl** and **ntohl** for long integers) which perform host-to-network order and network-to-host order conversions. Whether the network order is little-endian or big-endian doesn't really matter. It has been standardized across all TCP/IP network stack implementations. Unless you persistently pass only characters across sockets, you will run into byte-order problems if you do not use these routines. Depending on the machine architecture, these routines may be null macros or may actually be functional. Interestingly, a common source of bugs in socket programming is to forget to use these byte-ordering routines for filling the address field in the sock_addr structures. Perhaps it is not intuitively obvious, but this must also be done when using INADDR_ANY (i.e., htonl(INADDR_ANY)).
- A key goal of network programming is to ensure processes do not interfere with each other in unexpected ways. In particular, servers must use appropriate mechanisms to serialize entry through critical sections of code, avoid deadlock and protect data validity.
- You cannot (generally) pass a pointer to memory from one machine to another and expect to use it. It is unlikely you will want to do this.
- Similarly, you cannot (generally) pass a file descriptor from one process to another (non-child) process via a socket and use it straightaway. Both BSD and SVR4 provide different ways of passing file descriptors between

unrelated processes; however, the easiest way to do this in Linux is to use the /proc file system.

Additionally, you must ensure that you handle short writes correctly. Short writes happen when the write() call only partially writes a buffer to a file descriptor. They occur due to buffering in the operating system and to flow control in the underlying transport protocol. Certain system calls, termed *slow* system calls, may be interrupted. Some may or may not be automatically restarted, so you should explicitly handle this when network programming. The code excerpt in Listing 3 handles short writes.

Using multiple *threads* instead of multiple processes may lighten the load on the server host, thereby increasing efficiency. *Context-switching* between threads (in the same process address space) generally has much less associated overhead than switching between different processes. However, since most of the slave threads in this case are doing network I/O, they must be kernel-level threads. If they were user-level threads, the first thread to block on I/O would cause the whole process to block. This would result in starving all other threads of any CPU attention until the I/O had completed.

It is common to close unnecessary socket file descriptors in child and parent processes when using the simple forking model. This prevents the child or parent from potential erroneous reads or writes and also frees up descriptors, which are a limited resource. But do not try this when using threads. Multiple threads within a process share the same memory space and set of file descriptors. If you close the server socket in a slave thread, it closes for all other threads in that process.

Connectionless Data—UDP

Listing 4 shows a connectionless server using UDP. While UDP applications are similar to their TCP cousins, they have some important differences. Foremost, UDP does not guarantee reliable delivery—if you require reliability and are using UDP, you either have to implement it yourself in your application logic or switch to TCP.

Like TCP applications, with UDP you create a socket and bind an address to it. (Some UDP servers do not need to call bind(), but it does no harm and will save you from making a mistake.) UDP servers do not listen or accept incoming connections, and clients do not explicitly connect to servers. In fact, there is very little difference between UDP clients and servers. The server must be bound to a known port and address only so that the client knows where to send messages. Additionally, the order of expected data transmissions is reversed, i.e., when you send data using **send()** in the server, your client should expect to receive data using **recv()**.

It is common for UDP clients to fill in the sockaddr_in structure with a sin_port value of 0. (Note that 0 in either byte-order is 0.) The system then automatically assigns an unused port number (between 1024 and 5000) to the client. I'm leaving it as an exercise to the reader to convert the server in Listing 4 into a UDP client.

/etc/services

In order to connect to a server, you must first know both the address and port number on which it is listening. Many common services (FTP, TELNET, etc.) are listed in a text database file called /etc/services. An interface exists to request a service by name and to receive the port number (correctly formatted in network byte-order) for that service. The function is **getservbyname()**, and its prototype is in the header file /usr/include/netdb.h. This example takes a service name and protocol type and returns a pointer to **struct servent**.

```
struct servent
{
        char *s_name;   /* official service name */
        char **s_aliases;       /* alias list */
        int s_port;     /* port number, network<\n>
                         * byte-order--so do not
                         * use host-to-network macros */
        char *s_proto;  /* protocol to use */
};
```

Conclusions

This article has introduced network programming in Linux, using C and the BSD Socket API. In general, coding with this API tends to be quite laborious, especially when compared to some of the other techniques available. In future articles, I will compare two alternatives to the BSD Socket API for Linux: the use of *Remote Procedure Calls* (RPCs) and the *Common Object Request Broker Architecture* (CORBA). RPCs were introduced in Ed Petron's article “Remote Procedure Calls” in *Linux Journal* Issue #42 (October, 1997).

Resources

TCP **listen()** Backlog

Major System Calls The next article in this series will cover the issues involved in developing long-lived network services (*daemons*) in Linux.

All listings referred to in this article are available by anonymous download in the file ftp://ftp.linuxjournal.com/lj/listings/issue46/2333.tgz.

Ivan Griffin (ivan.griffin@ul.ie) is a research postgraduate student in the ECE department at the University of Limerick, Ireland. His interests include C++/Java,

WWW, ATM, the UL Computer Society (http://www.csn.ul.ie/) and, of course, Linux (http://www.trc.ul.ie/~griffini/linux.html).

Dr. John Nelson (john.nelson@ul.ie) is a senior lecturer in Computer Engineering at the University of Limerick. His interests include mobile communications, intelligent networks, Software Engineering and VLSI design.

Archive Index Issue Table of Contents

Advanced search

Linux Helps Bring Titanic to Life

Daryll Strauss

Wook

Issue #46, February 1998

Digital Domain uses Linux to create high-tech visual effects for the movies.

Digital Domain is an advanced full-service production studio located in Venice, California. There, we generate visual effects for feature films and commercials as well as new media applications. Our feature film credits include *Interview with the Vampire*, *True Lies*, *Apollo 13*, *Dante's Peak* and *The Fifth Element*. Our commercial credits are challenging to count, much less list here (see the web site at http://www.d2.com/). While we are best known for the excellent technical quality of our work, we are also well respected for our creative contributions to our assignments.

The film *Titanic* (written and directed by James Cameron) opened in theaters December 19, 1997. Set on the Titanic during its first and final voyage across the Atlantic ocean, this tale had to be recreated on the screen in all the splendor and drama of both the ship and the tragedy. Digital Domain was selected to produce a large number of extraordinarily challenging visual effects for this demanding film.

Figure 1. Computer Generated Image of the Titanic

The Task

Digital visual effects are a large portion of our work. For many digital effects shots, original photographic images are first shot on film (using conventional cinematic methods) and then scanned into the computer. Each “cut” or “scene” is set up as a collection of directories with an “element” directory for all the photographic passes that contribute to the final scene. Each frame of film is stored as a separate file on a central file server.

A digital artist then begins working on the shot. The work may involve creating whole new elements such as animating and rendering 3D models or modifying existing elements such as painting out a wire or isolating the areas of interest in the original film.

This work is done at the artist's desktop (often on an SGI or NT workstation). Once the setup for this work is done, the process is repeated for each frame of the shot. This batch processing is done on all the available CPUs in the facility, often in parallel and requires a distributed file system and uniform data overview. A goal of this processing is to remain platform independent whenever possible.

Finally, once all the elements are created, the final image is “composited”. During this step the individual elements are color corrected to match the original photography, spatially coordinated and layered to create the final image. Again, the set up for compositing work is usually done on a desktop SGI, and the batch processing is done throughout the facility.

Since building a full-scale model of the Titanic would have been prohibitively expensive, only a portion of the ship was built full size (by the production staff), and miniatures were used for the rest of the scenes. To this model we added other elements of the scene such as the ocean, people, birds, smoke and other details that make the model appear to be docked, sailing or sunk in the ocean. To this end, we built a 3D model and photographed 2D elements to simulate underwater, airborne and land-based photography.

During the work on *Titanic* the facility had approximately 350 SGI CPUs, 200 DEC Alpha CPUs and 5 terabytes of disk all connected by a 100Mbps or faster network.

The Analysis

Our objective is always to create the highest quality images within financial and schedule constraints. Image creation is accomplished in two phases. In the first phase, the digital artist works at an interactive workstation utilizing specific, sophisticated software packages and specific high-performance hardware. During the second phase the work is processed in batch mode on as many CPUs as possible, regardless of vintage, location or features to enhance interactive performance.

It is difficult to improve on that first, interactive phase. The digital artists require certain packages that are not always available on other platforms. Even if similar packages are available, there is a significant cost associated with interoperating between them.

Another problem is that some of the packages require certain high-end (often 3D) hardware acceleration. That same quality and performance of 3D acceleration may not be available on other platforms.

In the batch-processing phase, improvements are more easily found, since basic requirements are high-bandwidth computation, access to large storage and a fast network. If the appropriate applications are available, we can improve that part of the process. Even in cases where only a subset of the applications are available on a particular platform, using that platform gives us the ability to partition work flow to improve access to resources in general.

We rapidly concluded the DEC Alpha-based systems served our batch-processing needs very well. They provide extremely high floating-point performance in commodity packaging. We were able to identify certain floating-point-intensive applications as port targets. The Alpha systems could be configured with large amounts of memory and fast networking at extremely attractive price points. Overall, the DEC Alpha had the best price/performance match for our needs.

The next question was which operating system to use. We had the usual choices: Windows/NT, DEC UNIX and Linux. We knew which programs we needed to run on the systems, so we assembled systems of each type and proceeded to evaluate their suitability for the various tasks we needed to complete for this production.

Windows NT had several shortfalls. First, our standard applications, which normally run on SGI hardware, were not available under NT. Our software staff could port the tools, but that solution would be quite expensive. NT also had several other limitations; it didn't support an automounter, NFS or symbolic links, all of which are critical to our distributed storage architecture. There were third-party applications available to fill some of these holes, but they added to the cost and, in many cases, did not perform well in handling our general computing needs.

Digital UNIX performed very well and integrated nicely into our environment. The biggest limitations of Digital UNIX were cost and lack of flexibility. We would be purchasing and reconfiguring a large number of systems. Separately purchasing Digital UNIX for each system would have been time consuming and expensive. Digital UNIX also didn't have certain extensions we required and could not provide them in an acceptable time frame. For example, we needed to communicate with our NT-based file servers, connect two unusual varieties of tape drives and allow large numbers of users on a single system; none are supported by Digital UNIX.

Linux fulfilled the task very well. It handled every job we threw at it. During our testing phase, we used its ability to emulate Digital UNIX applications to benchmark standard applications and show that its performance would meet our needs. The flexibility of the existing devices and available source code gave Linux a definitive advantage.

The downside of Linux was the engineering effort required to support it. We knew that we would need to dedicate one engineer to support these systems during their set up. Fortunately, we had engineers with significant previous experience with Linux on Intel systems (the author and other members of the system-administration staff) and enough Unix-system experience to make any required modifications. We carefully tested a variety of hardware to make sure all were completely compatible with Linux.

Numerology and Fate

The Linux distribution used was Red Hat 4.1. At that time Red Hat was shipping Linux 2.0.18, which didn't support the PC164 mainboard, so the first thing we had to do was upgrade the kernel. During our testing we tracked down a number of problems with devices and kept up with both the 2.0 and 2.1 series of kernels. We ended up sticking with 2.1.42 with a few patches. We also decided on the NCR 810 SCSI card with the BSD-based driver and the SMC 100MB Ethernet card with the de4x5 driver. It turned out to be a very stable configuration, but there was one serious floating-point problem that caused our water-rendering software to die with an unexpected floating-point exception.

This turned out to be a tricky problem to fix and didn't make it into the kernel sources until 2.0.31-pre5 and 2.1.43. The Alpha kernel contains code to catch floating-point exceptions and to handle them according to the IEEE standard. That code failed to handle one of the floating-point instructions that could generate an exception. As a result, when that case occurred, the application would exit with a floating-point exception. Once fixed, our applications ran quite smoothly on the Alpha systems.

The Implementation

At this point, the decision was made to purchase 160 433MHz DEC Alpha systems from Carrera Computers of Newport Beach, California. Of those 160 machines, 105 of the machines are running Linux, the other 55 are running NT. The machines are connected with 100Mbps Ethernet to each other and to the rest of our facility.

The staff at Carrera was extraordinarily helpful and provided inestimable support for our project. This support began at the factory, with follow-up support through delivery, support and repair.

We created a master disk, which we provided to Carrera, along with a single initialization script that would configure the generic master disk to one of the 160 unique personalities by setting up parameters such as the system name and IP address. Carrera built, configured and burned-in the machine, then logged in as a special user causing the setup script to execute. When the script completed, the machine automatically shut down.

This process made configuring the machines easy for both Carrera and us. When the hosts arrived, we just plugged them in and flipped the switch, and they came up on the network. All 160 machines are housed in a small room at Digital Domain in ten 19 inch racks. They are all connected to a central screen, keyboard and mouse via a switching system to allow an operator to sit in the middle of the room and work on the console of any machine in the room.

Figure 2. Digital Domain Computer Room

The room was assembled in a time period of two weeks including the installation of the electrical, computing and networking. The time spent creating the initialization script was extremely well spent as it allowed the machines to be dropped in place with relatively little trouble. At that point we began running the *Titanic* work through the "Render Ranch" of Alphas.

The first part of this work partition was to simulate and render the water elements. We knew that the water elements were computationally very expensive, so this process was one of the major reasons for purchasing the Alphas.

These jobs computed for approximately 45 minutes and then generated several hundred megabytes of image data to be stored on central storage servers. Intermediate data was stored on the local SCSI disk of the Alpha. The floating-point power of the DEC Alpha made jobs run about 3.5 times faster than on our old SGI systems.

As the water rendering completed, the task load then switched to compositing. These jobs were more I/O bound, because they had to read elements from disks on servers spread around the facility and combine them into frames to be stored centrally. Even so, we still saw improvements of a factor of two for these tasks.

We were extremely pleased with the results. Between the beginning of June and the end of August, the Alpha Linux systems processed over three hundred thousand frames. The systems were up and running 24 hours a day, seven days a week. There were no extended downtimes, and many of the machines were up for more than a month at a time.

The Problems

We addressed a number of different problems using a variety of techniques. Some of the problems were Alpha specific, and some were issues for the Linux community at large. Hopefully, these issues will help others in the same position and provide feedback for the Linux community.

Hardware compatibility, particularly with Alpha Linux, is still a problem. Carrera was very cooperative about sending us multiple card varieties, so that we could do extensive testing. The range of choices was large enough that we were able to find a combination that worked. We had to pay careful attention to which products we were using, as the particular chip revision made a difference in one case.

The floating-point problem (discussed above) was the toughest problem we had to address. We didn't expect to find this kind of problem when we started the project. This was a long-standing bug that had never been tracked down—we attribute this fact to the relatively small Alpha Linux community.

Linux software for Alpha seems to be less tested than the equivalent software for the Intel processors—again, a function of the user-base size. It was exacerbated by the fact that Alpha Linux uses glibc instead of libc5, which introduced problems in our code and, we suspect, in other packages.

We had a number of small configuration issues with respect to the size of our facility. Most of these were just parameter changes in the kernel, but they took some effort to track down. For example, we had to increase the number of simultaneously mounted file systems (64 was not sufficient). Also, NFS directory reads were expected to fit within one page (4K on Intel, 8K on Alpha); we had to double this number to support the average number of frames stored in a single directory.

Boot management under Linux Alpha was more difficult than we would have liked. We felt the documentation needed improvements to make it more useful. Boot management required extensive knowledge of ARC, MILO and Linux to make it work. ARC requires entering a reasonably large amount of data to get MILO to boot. MILO worked well and provided a good set of options, but we never managed to get soft reboots to operate correctly. We've been working with the engineers at DEC to improve some of these issues.

The weakest link in the current Linux kernel appeared to be the NFS implementation, resulting in most of our system crashes. We generally had a large number of file systems mounted simultaneously, and those file systems were often under heavy load. When central servers died or had problems, the Linux systems didn't recover. The common symptoms of these problems were stale NFS handles and kernel hangs. When all the servers were running, the Linux boxes worked correctly. Overall, the NFS implementation worked, but it should be more robust.

Conclusion

The Linux systems worked incredibly well for our problems. The cost benefit was overwhelmingly positive even including the engineering resources we devoted to the problems. The Alpha Linux turned out to be slightly more difficult than first expected, but the state of Alpha Linux is improving very rapidly and should be substantially better now.

Digital Domain will continue to improve and expand the tools we have available on these systems. We are engendering the development of more commercial and in-house applications available on Linux. We are requesting that vendors port their applications and libraries. At this time, the Linux systems are only used for batch processing, but we expect our compositing software to be used interactively by our digital artists. This software does not require dedicated acceleration hardware, and the speed provided by the Alpha processor is a great benefit to productivity.

Feature film and television visual effects development has provided a high-performance, cost-sensitive, proving ground for Linux. We believe that the general purpose nature of the platform coupled with commodity pricing gives it wide application in areas outside our industry. The low entry cost, versatility and interoperability of Linux is sufficiently attractive to warrant more extensive investigation, experimentation and deployment. We are currently at the forefront of that development within our industry and hope to be joined shortly by our peers.

Why Risk Linux? A Production Perspective

by Wook

Currently, Digital Domain's core business is as a premier provider of visual effects creativity and services to the feature film and commercial production industries. As such, we often take a conservative approach to changes in infrastructure and methodologies in order to meet aggressive delivery schedules and the most demanding standards of product quality.

During the course of work on several recent feature film productions, we encountered situations where our installed base of equipment was not adequate to meet changing production schedules and dynamic visual effects requirements (in terms of increasing magnitude of effort and complexity). We needed to meet these challenges head on without impacting the existing pipeline and without creating new methodologies or systems which would require re-engineering or re-training. Linux Alpha helped us overcome these challenges both cost effectively and quickly (a rare combination).

Selecting Linux as part of the production pipeline for the film *Titanic* required several goals to be met. If we had not met these requirements, it is unlikely we would have been able to deliver sufficient computing resources in a timely fashion to the production. We needed interoperability and, to a certain degree, compatibility with our SGI/Irix-based systems. Interoperability and compatibility with Linux had been demonstrated during a previous effort (*Dante's Peak*). We ported critical infrastructure elements (to support distributed processing) to the Linux environment in days, not weeks, using existing staff. The developers of these tools were able to rapidly deploy to the Linux environment, demonstrating that we could leverage that environment in short order. We needed performance, as the schedule for the production, as well as the magnitude of the work implied a 100% or more increase in studio processing capacity. As we had shown that Alpha Linux provided a factor of three to four over our SGI systems (see main article), it was possible to deliver that increased level of performance while physically constrained (air, power and floor space) within our current facility.

As to cost effectiveness, we would have needed more than twice as many Intel machines as Alphas to meet our performance goals. SGI was a valid contender, but could not compete on a price per CPU basis. We also needed a viable structure for delivery, installation and support. Carrera Computers had proven their ability to supply and support us in a timely and cost-effective manner prior to this order, and that company continued to provide an extraordinary level of service throughout the *Titanic* project.

All things considered, this risk paid off in substantial dividends of project quality and time. Because the urgency of the situation demanded that we think “outside the box”, we were able to deliver a superior solution in a framework that was entirely compatible with our normal operating

Daryll Strauss (daryll@d2.com) is a software engineer at Digital Domain. He has been hacking on Unix systems of one variety or another for the last 15 years, but he gets the most enjoyment out of computer graphics. He has spent the last five years working in the film industry doing visual effects for film.

Wook has been a software engineer for over 20 years, having discovered computers and became a complete geek at the age of 14. He has worked for many companies over those years, finally coming to rest at Digital Domain, where he was considered unfit for the task of software engineering and has been relegated to the position of Director of (Digital) Engineering.

Archive Index Issue Table of Contents

Advanced search

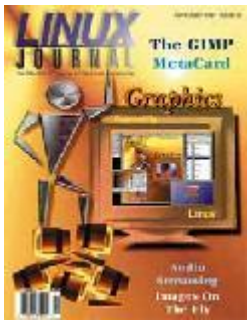
LINUX JOURNAL

The Quick Start Guide to the GIMP, Part Four

Michael J. Hammel

Issue #46, February 1998

Our series winds up with a detailed description of the toolbox, plug-ins and keyboard acceleration.



Last month we talked about the Image Window and Layers, two very important topics when learning about the GIMP. Image Windows are where image processing work is displayed and where you pick the elements of the image to be worked on. Layers provide even more control over those selections. This month we take a look at the Toolbox, the set of buttons that drives most of the major processing tools, including various styles of selections. We also look at using text and filters, and briefly discuss scripting.

The Toolbox

The most recent developer's release of the GIMP used for this article included a more compact version of the Toolbox than earlier versions. None of the tools were dropped, but the excess blank areas in the buttons and menu bar were removed. This saves some screen space on smaller displays, although I found it a little hard, with my aging eyes, to make out some of the buttons on an 1152x900 display. Keep in mind that most tools and many features in the GIMP are also accessible via keyboard accelerators, also known as keyboard shortcuts. I'll only mention a few, but take note of them when you experiment, as they help speed your work tremendously.

Figure 1. The Toolbox

The Toolbox is made up of buttons set in a seven row by three column box, with a menu bar at the top and a foreground/background color selection box at the bottom. The menu bar holds a File menu similar to the one in the Image Window (although with fewer options) and an Extensions menu. The latter allows you access to the Script FU scripting interface. We'll get to that a little later. The tools are selected by a single left-button mouse click over the appropriate button. A double click opens the Tool Options dialog window that provides access to configurable options for the selected tool, if any are available.

Figure 2. Selection Tools

The first row of buttons and the first two buttons in the second row are the Selection Tools: Rectangular, Elliptical, Free Hand, Fuzzy and Bezier. Rectangular and Elliptical draw the respective type of outline as you drag the cursor, left mouse button down, around the Image Window. Holding the **shift** key down while dragging will form a uniform box or circle. Free Hand will draw an outline which follows the path of the cursor as you drag it. Note that when you release the mouse button while working with the Free Hand selection, the selection is closed between the starting point and ending point of the cursor's path. The Fuzzy select will select a region of an image based on the colors in the region where you click with the left mouse button. This sort of selection works best for images that do not contain gradients or wide ranges of colors over very small areas. The Bezier selection tool allows a region to be outlined in precise detail. The Bezier tool was used to select points on the outline of the monitors (after each point was selected, a straight line was drawn connecting the last two points) on the cover of the November issue, back when part one of this series ran. Once a complete outline is selected, a single click on the inside of the selected region activates that region and the marching ants began their dance around the selection's outline. Bezier selections can also be used to select outlines of very jagged regions, but the method for doing so is beyond the scope of this article. Note that all the Selection Tools have configurable options. They all support anti-aliasing and user-defined feathered regions.

The third button on the second row is the Intelligent Scissors button. At the time of this writing that feature was not yet supported.

Figure 3. Move, Magnify and Crop Tools

The third row of buttons holds the Move, Magnify and Crop tools. The Move Tool allows a selected region or regions to be moved. The Move tool is more sophisticated than it might seem at first. When used with the Layers and

Channels dialog, it becomes possible to select various pieces of an image to adjust them within the framework of the complete image.

The Magnify Tool is used to zoom in and out of the image. This is useful for finding the edges of nearby objects in an image. At times, you might bucket-fill a region only to find the fill process covered more of the image than you expected. Using the zoom tool might show that a tiny region of the fill area is not enclosed by colors different from the fill region, indicating that the fill process “leaked” into adjoining regions.

The Crop tool selects a region by boxing it out; hold down the left mouse button, drag and release to select a region. A dialog box will open giving the dimensions of the selected region. At this point the region can be cropped; that is, the area outside the boxed region is discarded and the Image Window is resized to the area defined by the Crop Tool. All Layers are cropped to the new dimension. Note that there are image filters which can also perform various forms of cropping.

Figure 4. Transform and Other Tools

Button one in row four is the Transform Tool. Select a region or layer of the image and then click on the Transform Tool button. Next, click inside the selected region or layer. The area will be outlined with a rectangle and the corners will be outlined with small, wireframe boxes. By default, the Transform Tool is configured to perform rotations on the selected region. Double clicking the Transform Tool button brings up the Tool Options dialog where you can select one of Rotate, Scale, Shear or Perspective for the selected region or layer. Scaling will work with any selected area but be aware that scaling of text can, if enlarged too much, cause aliasing (jagged edges) to occur. Click, hold and drag the mouse inside the selected region in order to perform the current transform.

The middle button of row four is the Flip tool. Click on this button to flip a layer, image or selected region either horizontally or vertically. The Tool Options dialog for this tool provides the mechanism for selecting the direction of the flip. A flip reverses the pixels of the area to be flipped—the rightmost pixels end up on the leftmost side in a horizontal flip, and the topmost pixels end up at the bottom in a vertical flip.

On the outside of the fourth row is the Text Tool. Clicking once on this tool will enable the tool, but you need to click inside an Image Window to bring up the Text Tool dialog. This dialog is used to select the font characteristics of the text you wish to insert in the image. The color of the text will match the current foreground color, so be sure you have set the foreground color (described in a

later section on Colors) before clicking on the OK button in the Text Tool dialog. Once you click on OK, the Text Tool dialog closes, and the text is inserted in the image with the marching ants (if enabled) dancing around the edges of the text. At this point you can click on the Move Tool to move the text to the desired location. Once you're familiar with the GIMP, you'll find that it's not necessary to click on the Move Tool. However, when first starting out, it's important to be familiar with the current mode of GIMP operation by explicitly selecting the Tool of interest.

An important note about adding text: text is inserted into an image as a *floating selection*. Floating selections have their own, temporary layer in the Layers and Channels dialog. They require the user to select a disposition for them before any further work can be done on other layers. A floating selections disposition can be one of “Anchor” or “New”. Anchoring a layer composites it with the current layer. Marking a floating selection as a new layer adds it to the top of the set of layers that make up the current image.

The next row is made up of the Color Picker Tool, the Bucket Fill Tool and the Blend Tool from left to right. Colors in the GIMP can be selected through the use of the Color Selection dialog which allows fairly detailed selection of the Red, Green and Blue levels along with HSV (hue, saturation and value) levels. However, if you need to use a color that already exists in your image, the Color Picker Tool, which resembles an eye dropper, is a much quicker way to set the foreground color to the desired color. Click on the Tool's button and then move the mouse over the pixel in the image window which contains the color you wish to use. Important note: be sure the layer in which that pixel lives is currently active. When you click in the image window the foreground color is set to the color of the pixel over which the mouse sits. The foreground color's setting is shown in the box at the bottom of the Toolbox. The two large boxes represent the current foreground (upper left) and background (lower right) colors.

Filling a region of the image with either a solid color or a pattern can be done using the Bucket Fill Tool. First, select a region of an image using one of the Selection Tools (or use **ctrl-A** to select the entire active layer). Next, double click the Bucket Fill Tool's button to bring up the Tool Options dialog. If you want a solid color to fill the selected region, click on the Color Fill toggle button. Select a color for the foreground by double clicking on the foreground box at the bottom of the Toolbox. The Color Selection dialog will open. Set the color to Red, for example, and then click on OK in the Color Selection dialog. Now, you're ready to fill in the selected region. Simply click on the selected region, and the GIMP fills it with a solid red pattern. You can also fill regions without selecting them using the Fill Threshold slider in the Tool Options dialog for the Bucket Fill Tool. The transparency amount can also be set prior to doing the fill.

Instead of a solid color fill, you can also fill a region with a given pattern. There are a host of default patterns available with the GIMP. To choose a pattern you first click on the Pattern Fill toggle in the Tool Options dialog for the Bucket Fill Tool. Then place the mouse over the Image Window which will be filled and type **ctrl-shift-P** to open the Patterns dialog. Click on one of the patterns to select it. Then click on the selected region to have the GIMP fill it with the selected pattern. If you create an interesting pattern you wish to use for Pattern Fills, you can save it as a PAT file from the Save Image dialog.

Figure 5. Gradient Editor

The last tool in the fifth row of the Toolbox is the Blend Tool. This tool is used for filling a region in a direction the user specifies with a blended color pattern. The colors used can be blended from foreground to background, foreground to transparent or from a custom gradient provided by the Gradient Editor. The Gradient Editor can be opened from the Image Windows pop down menus as "Dialogs**->**Gradient Editor". There are a number of default gradients that come with the distribution as well as a number that have been created or collected by the Gradient Editors author, Federico Mena Quintero, known affectionately to the GIMP developers as Quartic. The resources section of this article lists Quartic's web pages which are considered the definitive resource for finding anything related to the GIMP, including more custom gradients.

A full description of gradients would take many pages, but to get an idea of what can be done with them, consider the cover art for the November 1997 *Linux Journal* shown at the beginning of this article. The gold background is one gradient, the colors of the stick man are another. Gradients are very useful for creating all sorts of special effects.

Figure 6. Pencil, Paintbrush and Other Tools

The next four buttons are the closest thing to drawing tools the GIMP provides. They are, in order: the Pencil Tool, the Paintbrush Tool, the Eraser Tool and the Airbrush Tool. Each of these requires that a brush type be set. (There is a default setting, but you'll find you'll wish to change this fairly often.) The Brush Selection dialog allows selection of the brush type. Once selected the brush type is applicable to each of the drawing tools. The Pencil Tool draws solid lines that follow the shape of the brush using hard edges. Using a brush like the Duck or Guitar results in an effect similar to using a stencil. The Paintbrush works similarly, although the edges of the stencil are much softer, having been antialiased.

Figure 7. Brushes

The Eraser Tool works just the opposite of the Pencil Tool in that the shape of the brush is used to remove the underlying pixels. Note that this does not mean the new color for the pixels is the background color; instead, the pixels become transparent. Finally, the Airbrush Tool works similarly to the Paintbrush but acts as if the pattern is being blown onto the canvas at a given rate and pressure.

All of the drawing tools use the current foreground color, to do the drawing. If the brush type is not a solid pattern, the darker areas of the brush come out in the foreground color, and the lighter areas of the brush come out transparent (with the current pixels showing through the area the brush doesn't cover). This behavior can be modified using the Mode and Opacity settings on the Brush Selection dialog. The dialog also allows you to specify how fast the brush is applied using the Spacing setting. Lower spacing values cause the brush to be repeated quickly, overlapping the individual applications of the brush shape. Higher settings cause the brush to be applied less often as the mouse is moved around the Image Window.

Aside from the Airbrush tool, the bottom row of the Toolbox also contains the Clone Tool. A rubber stamp icon represents this tool in the Toolbox. This handy tool allows you to use one part of an image to paint or “smudge” over another part of an image. This is useful, for example, for removing the edges between pasted objects or for clearing the edges in a tileable image. Tileable images are ones which, when repeated side by side and top to bottom, have no seams between the images, thus creating the illusion of a solid image. Tileable images are often used as background images for web pages.

To use the Clone Tool, click on the middle mouse button on the bottom row of the Toolbox. Now, place the mouse over the Image Window on the spot you wish to use as the beginning source location. Hold down the control key and press the left mouse button over this spot. A crosshair is displayed beneath the mouse pointer to signify the source has been selected. Now move the mouse to where you wish to begin cloning. The distance and angle between the source and destination points will remain constant as you hold down the left mouse button and drag it around the image. The area used as the source will be the same size and shape as the current brush type, so be sure to set it before beginning the cloning operation. When removing edges in an image, use a brush type that has soft edges; that is, edges that fade from black to white. The Nove brush is good for this type of work. Note that the clone tool can also use a pattern type, just like the Bucket Fill Tool. The use of the pattern or the image as the source can be selected from the Clone Tool's Tool Options dialog.

The last tool in the Toolbox is the Convolver Tool. This tool can be used to sharpen or blur areas of an image; the choice is made from the Convolver

Tool's Tool Options dialog. Just select the tool from the Toolbox and then start dragging the mouse around the area of the image you wish to sharpen or blur. The operation is not very obvious using the default settings, which have a Pressure setting of 50 in the Tool Options Window. Play with it a bit to get the hang of how much dragging is required for the desired effect. Increasing the Pressure setting will cause the blurring to be more obvious with less work on your part. The Convolver, along with the Clone Tool, can be used to clean up sludges or to remove unwanted lines and streaks from images.

At the bottom of the Toolbox window, you'll find the Foreground/Background color selection box. This box contains two large boxes, a two-ended arrow and two smaller boxes. The large boxes are the foreground color and background color (top left to bottom right, respectively). Clicking on either will open the Color Selection dialog allowing you to set the current color. The two-ended arrow allows you to swap the foreground and background colors. The smaller boxes, below the foreground color box, reset the colors to their default values.

Most operations use the foreground color for fill, paint and drawing operations unless a pattern has been set. The color should be set prior to clicking on the tool of interest or the results may be unexpected. New layers can be created using the background color as well, so be sure you've chosen the color you want prior to creating the new layer.

Selections

One of the most important aspects of any tool like the GIMP is the ability to select regions of an image to be processed. Knowing the types of selections that are available and how to use them will greatly enhance your use of the GIMP.

The simplest selection is the one which selects the entire current layer. This particular selection can either be done via the Image Window menus or by using the **ctrl-A** keyboard accelerator. Once selected, the region is surrounded by the marching ants. Once a region is selected, it is important to know how to turn off the selection. One method is to click on one of the Selection Tools in the Toolbox and then click the left mouse button in a region of the image that is not currently selected. In the case where **ctrl-A** was used to select the entire layer, you need another way to turn off the selection since there is no section of the image that is not selected. Again, the Image Window menus and keyboard accelerators provide the answer. **shift-ctrl-A** turns off the selection. Note that this will turn off any selection, not just those created with **ctrl-A**.

The Toolbox offers the five basic selection methods: Rectangular, Elliptical, Free Hand, Fuzzy and Bezier. For those of you who have used Adobe Photoshop, the Rectangular and Elliptical selections are accessed from Photoshop using a

single tool called the Marquee. The Free Hand selection tool in the GIMP is referred to as the Lasso in Photoshop. All of these work basically the same, although the Tool Options dialogs in the GIMP look different than their counterparts in Photoshop.

The Bezier selection tool in the GIMP is accessed via the Lasso tool in Photoshop using Paths. In the GIMP the user creates the paths for the selection using the tools in much the same manner as the Free Hand tool except that control points are used to refine the path of the selection. Fuzzy selections are based on the range of pixel values over which the user drags the mouse. A fuzzy selection can be used to select all regions with the same single color or all regions which encompass a range of values.

There are a few additional features common to all of the selection methods. First, all selections can be “feathered”. A feathered selection means that the edge of the selection turns into a fading pattern, from current image to transparent. The feathering starts at the edge of the selection and continues out for the specified number of pixels. Also, all Toolbox selections allow for antialiasing, which reduces the jagged appearance of the selection in some cases. Some selection tools provide a “Sample Merged” option in the Tool Options dialog. This option allows the selection to be taken based on the visible image and not just the currently active layer.

Clearing a selection (available from the Image Window menus as “Edit**->**Clear”) removes the selected region. The cleared area is transparent if the active layer contains an Alpha channel. If it doesn't, the cleared area takes on the background color. Filling a selection (“Edit**->**Fill” from the Image Windows menus) will fill a selection to the background color. Selections can be extended by holding down the **shift** key while performing another selection. They can be reduced or otherwise have sections removed by holding down the control (**ctrl**) key while performing another selection.

There are a number of other options available for selections, such as selecting regions by color, or extending and shrinking selections. All of these are available from the Selections submenu in the Image Window's pop-down menu. One of the newer features is the ability to snap selections to the Ruler Guides. The “View**->**Snap To Guides” menu option toggles the snap action on or off.

Selections can be moved, rotated, flipped, filled, cut, pasted and have any of the filters applied. The selection applies only to the active layer, but whatever process was applied to a single layer can be repeated by simply selecting a different layer, leaving the selection outline in place, and then choosing the “Edit**->**Redo” option from the Image Window menus. Selections, once moved in

any manner, become floating layers in the Layers and Channels dialog. You must make sure this floating layer is given a final disposition—either making it a new layer or anchoring it the currently active layer—before continuing with your work.

Text

One of the most useful aspects of the GIMP is its ability to manipulate text. Logos, cover art and birthday cards are all possible given the many features provided by the GIMP. In order to add text to your images, you first need to understand how the GIMP gets access to text fonts.

The GIMP is based on Xlib and as such has access to the font rendering engines within the X server. This normally means that the GIMP has access to one of two types of fonts: Bitmap and Postscript Type 1 fonts. The former are not widely distributed (such as on commercial CDs from a software store like CompUSA or Egghead Software). My experience has been that you shouldn't bother with these types unless you happen to be given one. Conversely, Postscript Type 1, more commonly referred to simply as Type1, fonts are quite common. I've purchased at least four different CDs with literally thousands of Type1 fonts on them for use with Linux and X Window System applications. In order to make use of these fonts, you need to install them on your system and inform the X server. My *Graphics Muse* column in *Linux Gazette* (Issue 14, February 1997) had an article on how to do this titled “Adding Fonts To Your System”. Note that an additional format, TrueType, can also be used if you have a font server available for use. Caldera ships a font server with their packages, but most of the other distributions do not. Font servers have, in general, always been an additional package purchased separately for X servers on Linux systems.

Figure 8. Text Tool Dialog Window

Once you've gotten the fonts installed and available under X, you're ready to use them in the GIMP. The Text Tool in the Toolbox provides access to the fonts and is used to insert text into images. Figure 6 shows the Text Tool dialog window. When you click on the Text Tool button in the Toolbox and then click in the Image Window, the Text Tool dialog opens. If you have a large number of fonts installed on your system, the dialog will not open immediately the first time you access it—wait a moment or two as it gathers information about the available fonts. Subsequent invocations of the tool will not take so long, as the GIMP caches the information about fonts for better efficiency.

The Text Tool dialog provides a scrollable window that lists the available fonts by name, sorted alphabetically. Below this window is a window for typing the text you wish to add to the image. A limitation in the current implementation

requires the text to be a single line (no carriage returns), so multiline text has to be created one line at a time. Fortunately, the addition of Guides (from the Image Window's Rulers) allows a fairly good way to line the text up.

The only other part of this dialog that you'll find yourself using regularly is the Size setting. The size value is a text field that you can type in to set the size of the font. Next to this is an options menu. This menu has two options: Pixels and Points. The difference isn't completely clear to me, so I use Pixels most of the time. I expect, when I become more educated as to the use of these settings, I'll find a need for the Points option. The Size value, however, is an item I use quite often. Note that when you type in a new size and press **enter**, the text field at the bottom of the dialog is adjusted to reflect the new size. Also note that the text in this field is not antialiased, even if the Antialiasing toggle button in the dialog is set (it's set when the toggle appears depressed). The antialiasing setting only affects the way the text is displayed in the image.

When you have finished setting up the font type, size and text, click on the OK button to apply the text to the image. As I stated earlier, the text is applied to a floating layer and can be moved around that floating layer. When the text is applied in this way, it is presented in the Image Window with the marching ants outline and a transparent (i.e., the underlying image is visible) body. When the cursor is placed over the transparent section (anywhere inside the marching ants) the text can be moved. For some fonts and sizes, it becomes very difficult to find the body of the text with the mouse pointer. In these cases, click on the Zoom Tool and then click by the text to zoom in on it. Once the text is more accessible to the pointer, you can select the Move Tool from the Toolbox and proceed to move the text around the image. When you have finished, be sure to create the new layer or anchor it to the current layer.

Filters aka Plug-Ins

After all this, we've finally come to a place where we can talk about filters. Much of your image work will be based on filters, also referred to as Plug-Ins. These are external programs called by the GIMP to process part or all of the current layer of a given image window. Filters use the GIMP's Plug-In API (application programming interface) to access image data via shared memory. Although the filters are external programs, they cannot be used outside the GIMP.

Quite a number of filters are now delivered as part of the core GIMP package. They can be placed either in the system plug-in directories, usually /usr/local/lib/gimp/*release*/plug-ins or in the users .gimp/plug-ins directory. When the GIMP starts, it will query the plug-ins for necessary information so that the menu options can be built. Consequently, when you've added large numbers of plug-ins to these directories, the GIMP can take a bit of time to get started.

Using the default set of filters, start-up time isn't too bad on a Pentium 133MHz system.

All of the filters are accessed through the Image Windows Filters menu option. Filters are broken into a number of categories ranging from Blurs to Distorts to Rendering options. Some filters work on the existing image directly, modifying the existing pixels based on the plug-ins function. For example, a blur looks at the pixels in an image and samples those that are near each other to produce a blurred effect. The pixels sampled can be either the whole image or a selected region. Other filters, such as IFSCompose, create new images in the current layer, so you may wish to either start with a completely new image or at least a new layer before running these filters. Experimentation is the key—play with the filters on stock images to see how they effect it and its layers.

Figure 9. Bump Map Plug-In

Figure 10. Bump Map Image

I've included samples of just two of the filters: IFS Compose and Bump Map. The latter is an interesting 3D effect that is easily added to 2D images. The IFS Compose is a fractal-based tool that can be used to create some rather stunning images. I've played with this tool quite a bit and am still amazed at the range of effects that are possible. One important thing to remember about IFS Compose—you can add and delete the control triangles. For the longest time I simply played with the default three triangles. Adding triangles allows even greater flexibility to the effects.

Figure 11. IFSCompose Plug-In

Figure 12. IFSCompose Image

Script FU

An extension to the filters is the builtin scripting language, Script FU. This is a Scheme-based scripting language that allows you to string sets of filters and other GIMP internal commands and tools together to create interesting effects. A set of predefined scripts is distributed with the core GIMP package. Many of these are used to create very interesting logos.

Scripts make use of an internal database of routines that have been registered for use by plug-ins and scripts. This database is called the Procedural Database or PDB. There is at least one script available for browsing the database. All the routines in the database are registered by plug-ins or the GIMP itself using a set of parameters that allow it to be self documenting. The PDB browser prints this

information in a window showing the inputs and outputs for the routine, the author and a short descriptive blurb.

I'm not a big fan of Scheme, but that's mostly because I'm getting old, crotchety and tired of learning the language du jour. There is hope, however, for people like me. The GIMP's scripting language is just one example of a GIMP Extension. Extensions can be written by anyone, much like plug-ins, but the details are a bit beyond my understanding right now. I do know that at least one other scripting language extension has been written—for Tcl. Unfortunately, I don't know that language either. Ah, well.

Keyboard Accelerators and GTK

Finally, I think I should mention one of the lesser known but really nifty features of the GIMP. Actually, it's not really part of the GIMP, it's part of GTK, the windowing toolkit upon which the GIMP is based. Most windowing systems have keyboard accelerators, or keyboard shortcuts, that allow you access to window features directly from the keyboard. This is also true of GTK; however, GTK goes one step beyond. Other toolkits allow developers to predefine the accelerators in the source code or allow users to define them in configuration scripts used by the toolkits at start up time. GTK allows the user to set the accelerators *at run time*--you simply move the mouse over the menu item you wish to set, making sure the menu option is active (don't click the mouse on it, just move the cursor over it) and press the keystroke you wish to use for the keyboard accelerator. For example, moving the mouse over the “File**->**Print” option in the Image Windows menus, I press **ctrl-P** to set the keyboard accelerator for printing. Now, instead of having to go to the menu option, I just type **ctrl-P** over the image I wish to print.

Changes to the accelerators are saved between sessions, so you don't have to redo them each time you start the GIMP. You should be able to disable the accelerator by typing the same keystroke, but that didn't seem to work in the version I had. If a menu option already has an accelerator, just type the new key strokes to change it. This is a pretty nifty feature, as it puts more control in the hands of the user. As an X programmer by trade, I can say this is definitely a design goal of windowing application developers.

Resources on the Net

Michael J. Hammel (mjhammel@csn.net) a Computer Science graduate of Texas Tech University, is a software developer specializing in X/Motif living in Denver, Colorado. He writes the monthly Graphics Muse column in *Linux Gazette*, maintains the Linux Graphics mini-HOWTO, helps administer the Internet Ray Tracing Competition (http://irtc.org/) and coauthored *The Unix Web Server Book*, published by Ventana Press. His outside interests include running, basketball, Thai food, gardening and dogs.

Archive Index Issue Table of Contents

Advanced search

A Partner's Survival Guide

Telsa Gwynne

Issue #46, February 1998

A view of life with a hacker brought to us by a mischievous spouse who should know—Ms. Gwynne is married to Alan Cox.

I have just celebrated my fifth wedding anniversary to a hacker. Happily, there aren't many things I would change. I feel the past five years have given me some experiences that are only fair to pass on to those contemplating marriage to a Linux hacker—a few little tips on how life will be. I have used the pronoun “he” for the hacker because my hacker is male, but I have been assured by others that the following description is a fairly standard one regardless of gender.

When you shout three times for your spouse and require an answer, don't expect the answer to return immediately. He will reply, but only when his brain has finished with the problem it is currently working through. If this problem is conveyed to the brain via a computer screen, it becomes a “Very Important Problem” and requires up to an hour of grokking before answering questions such as:

- What do you want for tea?
- It's your mother on the phone—do you want a word?
- Are you coming to your parents' silver wedding anniversary party? If so, the train is leaving in five minutes.

Your spouse may have problems with rational time-management—or, more accurately, with the less-than-rational world. As far as a hacker is concerned, if it is Friday night, it is *not* the time to consider the quantity of clean work clothes which must be available for Monday morning. In fact, if it is Friday night, it is time for an extended hacking session because getting up for work the next day is not necessary. Saturday mornings, consequently, are a great time for getting things done which do not require help from your spouse.

The concentration powers of a hacker are a problem in everyday life. If Dr. Who is on the television, your hacker is not going to notice that it is raining and the clothes on the outside line are getting wet. After all, Dr. Who requires maximum concentration.

If you take your hacker to any films which involve computers in any way whatsoever, do not expect any appreciation of the plot, the music or the direction. I took my husband, a hacker-friend and my non-hacker sister to see *Beauty and the Beast*. I was glad my sister was with us to share my discussion of the film's qualities, as when the film had finished, the other two began discussing ray-projection and its application to just about every scene in the film. Similarly, we left *The Net* early, because it was boring (to me) and non-logical (to him). And as for that wretched *Terminator*...

Food provided by your hacker-partner will either be sloppy and messy (he was thinking hard at the time) or incredibly precise and technically excellent. Washing up, however, is a one-banana job to be left to the operator (spouse).

In a romantic interlude at a restaurant, be very sure that you actually want to know the answer when you gaze into his distant eyes and ask “What are you thinking about?” For the answer will most likely not be “You.”

When asked “do you want to do this or do that,” your hacker partner will consider it perfectly acceptable to answer “Yes” rather than selecting one of the options. Do not panic! This can be extremely useful at times. Consider the following example: you ask, “Do you wish to pay for the paper or the new bookshelves?” He says, “Yes.” Later, when you present him with the bookshelf bill and he wonders why, you say, “I asked which you wanted to pay. You said yes and didn't pay the papers, so I assumed you wished to pay for the new bookshelf.”

Some other partners may be able to explain gardening with a hacker better than me, but I know there are three vital things to note in this area. The first is to ensure your partner understands that Nature has root privileges—Nature doesn't have to make sense. The second is to let him know that planting seeds in a straight line is boring. The third thing he should understand is that three miniscule crumbs of soil dropped on a kitchen surface do not constitute a public health emergency. If your partner is foolish enough to mutter “that surface ought to be sterilised before using it again,” point him in the direction of the cupboard under the sink and invite him to acquaint himself with the process, since you've been doing it unnoticed every week.

As to house maintenance, does it involve problem-solving? If so, your hacker can safely be left to deal with the planning (for the amusement value, if nothing

else). However, intervention may be required. Your hacker may wish to adapt a household item to another use for which it was not intended, as he does with chunks of code. Sadly, however, /home/tools, once edited, cannot be returned to its original state and purpose.

Finally, here's one tip for all you hackers. Remember: while root can do most everything, there are certain privileges that only a partner can grant.

Telsa Gwynne is a registered mental health nurse and the wife of Alan Cox. The two are merely coincidental. Her computing time is spent playing nethack and hating majordomo. She can be reached via e-mail at hobbit@lxorguk.ukuu.org.uk.

Archive Index Issue Table of Contents

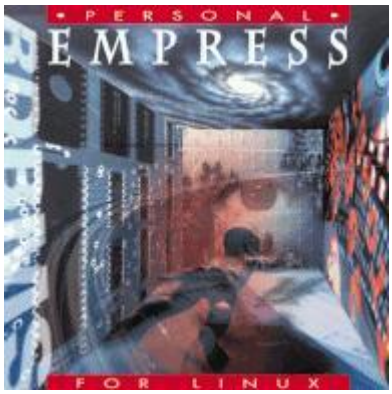
Advanced search

Personal Empress Database

David Weis

Issue #46, February 1998

Personal Empress includes a subset of the features found in the full version, including the database kernel, the Empress HLI Dynamic SQL C Extension, the Empress 4GL, the Empress GUI Builder and a beta version of the Empress Hypermedia WWW toolkit.



- Manufacturer: Empress Software, Inc.
- E-mail: sales@empress.com
- URL: http://www.empress.com/
- Price: $149 US
- Reviewer: David Weis

Personal Empress for Linux is a smaller version of Empress RDBMS (reviewed September 1997). It includes a subset of the features found in the full version, including the database kernel, the Empress HLI Dynamic SQL C Extension, the Empress 4GL, the Empress GUI Builder and a beta version of the Empress Hypermedia WWW toolkit. Only two user processes are allowed, and no networking capabilities are present. ODBC and other advanced features are not included.

Installation

Installation of the package is very easy. It comes on a single CD with a 12-page booklet of instructions. You mount the CD and run the **install** script—that's all there is to it. The script determines whether to install the ELF or the a.out version. You are also prompted to install the Empress Hypermedia toolkit.

Documentation

Since this is the “Personal” version, it doesn't come with any printed documentation. The full set of documentation does come in compressed PostScript format on the CD. I wish Empress had left the files uncompressed or in PDF format for easier viewing. On the plus side, the documentation is very complete. The command **empdocs** presents a table of contents for the documentation set, from which you chose the section to print or display using Ghostview.

Performance

My testing Platform is an IBM PS/2 Model 90 XP 486-DX2/50 with 16MB RAM, 1.5GB DEC DSP3160S on an IBM SCSI Adapter with 512KB cache and Red Hat Linux 4.0 2.0.29 kernel.

The performance of Empress was impressive, especially considering my hardware. When using a 30,000-record database the query seemed to finish before I could press return. The speed of the Hypermedia toolkit was similarly impressive.

I wasn't able to thoroughly test the X Window System interface on the tool due to an unsupported video card, but I did examine it briefly on another machine. The sample programs included in the package are sufficient to illustrate the power of the software.

Graphical administration tools similar to SQL Server are absent, but on the other hand, I didn't run into any situation that required them. Also, the system resource requirements are much slimmer for Personal Empress than for many other commercial RDBMS programs. The entire package takes only 45MB of disk space.

Conclusion

Empress positions Personal Empress as a solution for developers who are going to move completed applications to the full version of Empress. It is also a good performer for general database work. You get a lot of software for the

money, including a report writer, a 4GL and an X interface designer. Overall, I would rate Personal Empress as an excellent RDBMS.

David Weis is a computer science student at the University of Northern Iowa. He enjoys spending time with his fiancée and can be reached at weisd3458@uni.edu.

Archive Index Issue Table of Contents

Advanced search

S.u.S.E. V5.0

Stuart Green

Issue #46, February 1998

S.u.S.E. is well worth the $49 US price tag.

- Manufacturer: S.u.S.E.
- E-mail: info@suse.com
- URL: http://www.suse.com/
- Price: $49 US
- Reviewer: Stuart Green

When I arrived home following the ALG (Austin Linux Group) meeting of August 28, there in the kitchen sat a rather large box from S.u.S.E. Being of sound disposition and sober manner, I tore the box open. Inside were a boot diskette, four CDs in a single jewel box and a 380 plus page manual. The packaging was of good quality and shrink-wrapped as if it were sitting on a shelf at a computer retailer.

Documentation

The manual is excellent. This is the finest piece of comprehensive documentation I have seen. It includes practical and detailed setup instructions for services, such as firewalling, rational sendmail configuration, emulator setups (and many emulators such as DOS, Atari, Nintendo, CBM from the PET to the VIC-20, Sinclair/Timex, Gameboy and others).

The highly detailed table of contents and comprehensive index are rare in this type of documentation. Not only is information accessible, but it is presented very clearly. There are minor errors in translation from the German, including the presence of some characters unique to that language being left as is, in particular, in the names of individuals. These mistakes are easy to overlook. An unusual and very appreciated aspect of this manual is that it consists of original material and does not contain reprints of the LDP (Linux Documentation

Project) HOWTOs. The manual's thoroughness is aptly complemented by the clarity and accessibility of the material. The manual alone is worth the price of admission.

Software

The sheer amount of code available within this distribution is daunting—three of the four CD-ROMs are installation media. The fourth is a live file system that can be loaded in a demo mode that uses very little disk space. The explanation of how to permanently set up a small Linux partition and how to use the live file system to have a complete installation contains an accessible and cogent discussion of inode density, or how to cram 40,000 symbolic links into a partition.

S.u.S.E. has included utilities from Caldera and Red Hat; specifically Netware connectivity and Red Hat's RPM (Red Hat Package Management). The utilization of RPM, which is evolving into the de facto standard for installation, maintenance and replacement of software packages, is admirable.

Installation

I installed it on a variety of machines: a P166, a K6-166 and a PPro-II running at 300MHz. Kernel compile times were:

- P-166: 7 minutes 15 seconds (64MB memory)
- K6-166: 6 minutes 12 seconds (64MB memory)
- PPro-II: 2 minutes 43 seconds (96MB memory)

The system-level information available is the most thorough I have encountered in any operating system. The disk information, for example, contains the actual disk geometry as well as the LBA translations, the amount of physical cache on the disk and the manufacturer's information down to the device serial number. The same can be said for this utility's output regarding any physical device, e.g., the screens for the networking card go to the level of reading the registers for networking statistics on the card. I find this information of great value in diagnosing networking problems, as well as for installation ease. The S.u.S.E. environment contains only software available under the GPL (GNU Public License); there are no commercial X servers or window managers. The overall product seems to be quite complete.

As is the usual case the printed documentation is behind the actual code, so don't rely on the printed instructions to be consistent with what is on the screen at all times. The process is well documented on the screen. The only confusing point is the order of the process. Like all Unix installations, the first step is preparing the storage media by partitioning and/or making file systems.

Under S.u.S.E. this process is accomplished within a cleanly presented screen that requires using the function keys for selection of mount points, formatting options (inode density and bad block checking), etc. I found that the level of options is far above other distributions and, unlike some, S.u.S.E. does install.

The installation is simplicity itself. Once the machine is booted using the enclosed diskette, a series of questions is asked. The install process begins by prompting for the insertion of the first CD-ROM. It loads all selected packages from that CD-ROM and then boots into the newly installed system to complete the installation. It works well—it is a solid installer. The installation process is quick—a base installation taking a little over half an hour with reboots and all. The entire process is as smooth as an installation of this sort on Wintel architecture is going to be. Linux will not install with the ease of Solaris or Irix; but those are captive machines, where the manufacturer has total control of the hardware and choices are rather small.

Configuration

The best part is available after the installation is completed. Bringing up YaST (yet another setup tool) and choosing “Configuration” brings up a selector menu that looks like the “Check Config” utility in SGI's Irix and is similar to the configuration tools within System V Release 4. On systems running an unlinked kernel, most configuration is done this way. S.u.S.E. has managed to bring the same, sane method to Linux. In order to use this feature the S.u.S.E.-modified sources need be loaded.

I have not reorganized my disks to accommodate another Linux distribution yet, so I started with the “demo” mode using the file system on the CD. It worked, albeit with some pains—especially speed—almost all the user-level programs are on the CD. Regardless of which keyboard is chosen in YaST, once the live file system is loaded, the keyboard reverted to a German layout. I found this to be a real irritant. X configuration is simple and quick with the XFree86-3.3 version of XF86Setup. Once the X server is up, the environment presents a wide range of options—including dynamically switching window managers from **fvwm2** to **olvwm** to an **mwm** clone to AfterStep.

Conclusion

S.u.S.E. is well worth the $49 US price tag. It is also available as a subscription, for $34 US a pop with 3 releases per year expected, that can be canceled at any time. This is a full-featured distribution without commercial packages. There are quite a number of demos of commercial software included, but no fully functional packages, which keeps the price low. The installation is fast and fairly easy to use. With few reservations I'd recommend this system for users with experience ranging from modest to advanced. The installation requires slightly

more Unix exposure than Red Hat 4.2, and since there is an exceptional number of options, it is better suited to those who have done some hacking. It is stable.

Perceived performance is on par with Red Hat 4.2 and Caldera Standard. X installation is surprisingly quick, especially when one considers the amount of additional programming that S.u.S.E. loads. (S.u.S.E. loads and configures multiple window managers and desktop paradigms). The kernel tuning process is identical with other distributions. There are two sets of kernel sources—the straight Linux code and the S.u.S.E.-modified code. I used the S.u.S.E.-modified sources so I could access the configuration utilities, which are most certainly worth loading. S.u.S.E. updates are built in RPM format and are available at their FTP site. The lag time between bug reports and updated code seems to be in the two week range, which is as good as, if not better than, most commercial operating systems. This is definitely a competitive system worth consideration.

S.u.S.E. v5.1 News

Stuart Green (stugreen@bga.com) started in the industry 24 years ago doing serial communication. He has gone from packet switch design, to symbolic processing experiment and, finally, to multi-processor fault-tolerant Unix at Tandem Computers. He has been involved with Linux about four years. A year and a half ago he and two others enthusiasts formed the Austin Linux Group. They have been working on developing a Linux proficiency program that will eventually reside on the group web site, once built. He is the founder and president of Linux Systems Development Labs (LSD-Labs), a network design and security firm that also does system integration and support.

Archive Index Issue Table of Contents

Advanced search

The Essential Perl Books

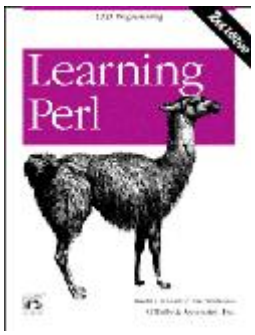
Eric Raymond

Issue #46, February 1998

All three books are well written, lucid and (in the best hacker tradition) quite witty and funny as well.

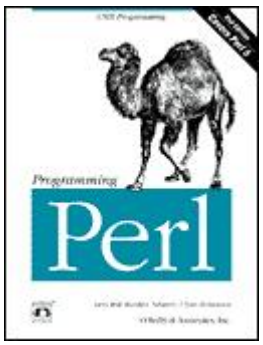
Perl has become one of the most important languages in the Unix/Internet world and is by far the most popular choice for CGI scripts on the World Wide Web. It is still growing rapidly and spreading to new applications. Perl is even making headway in non-Unix environments—a fact well attested to by the Microsoft NT presentations at the first Perl Conference, held August 19-21, 1997, in San Jose under the auspices of O'Reilly and Associates.

If you don't yet know Perl, learning it would be a good idea. For many kinds of end-user and system-administration applications (especially those involving text processing or the gluing together of several standard Unix tools), Perl is beyond price. Of the major interpretive Unix languages (shell, Tcl, Python, Emacs Lisp) only Python is truly competitive with Perl over Perl's entire range of applications, and even Python cannot match Perl's facilities for regular-expression-based text handling (though on the other hand Python probably scales up to very large and complex projects better).



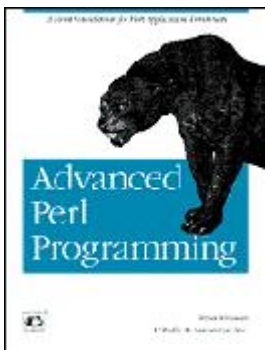
The vitality of Perl as a language is matched by the strength of these three books, which between them do a remarkably complete job of documenting the language. You probably need to own two out of three of these: the dilemma is deciding which two, for they are each aimed at rather different audiences.

Learning Perl (familiarly known as “the llama book”) is the introductory volume. It aims to teach elementary Perl quickly and appears to be targeted at people with little programming experience. While it does an excellent job of covering the basics, it omits a number of advanced topics including references and Perl's object system. It includes many exercises and could profitably be used as a textbook.



Programming Perl (well known as “the camel book”) is the intermediate-level volume and the best self-contained exposition of Perl. It describes the entire Perl language but provides only teasers on various Perl extensions (such as Perl/Tk for GUI programming) and the techniques for embedding Perl in C applications.

If you've seen the first edition of *Programming Perl*, you'll note many changes in the second edition besides just the coverage of the new Perl 5 features. The rather bizarre and haphazard organization of the first edition has been fixed—a huge plus. On the other hand, most of the whole-program Perl examples have been removed in the interest of keeping the book's page count below 700. (Tom Christiansen confirmed at the San Jose conference that he is working on a separate Perl cookbook.)



Advanced Perl Programming is a tome of esoterica for experienced Perl-heads. (It's fair to say you must be thoroughly versed in the material of the camel book to benefit from this “panther book”.) Within the language, it offers in-depth coverage of references, the Perl object system, typeglobs, eval, closures, modules and persistent-object techniques. It also covers the use of Perl/Tk to

construct GUIs. The last three chapters cover extending Perl, embedding Perl and Perl's internals, each in detail.

The author of *Advanced Perl Programming* illuminates Perl by connecting it to an impressively broad range of issues in computer science and programming language design. At the same time, his discussion of the way Perl does things is refreshingly concrete. His diagrams of Perl's internal data structures do much to demystify such knotty topics as typeglobs. The end-of-chapter sections systematically comparing Perl features to analogs in other languages are extremely valuable. Finally, though Mr. Srinivasan clearly loves Perl, he is not afraid to point out its uglinesses and occasional design shortcomings. As a specialist in computer language design myself, I found his comments uniformly intelligent, incisive and tasteful.

The only flaw I detect in this book is a couple of chapters that describe favorite Perl hacks of the author's in a way not strongly motivated by the rest of the text. Even so, *Advanced Perl Programming* is an astonishingly sustained tour de force and, if not the best book of the three, certainly the most intellectually stimulating.

All three books are well written, lucid and (in the best hacker tradition) quite witty and funny as well. If you are going to buy only one Perl book, it should be *Programming Perl*, which serves quite well as a desk reference for the language. Serious programmers will find a rich feast in *Advanced Perl Programming*. *Learning Perl* serves as a satisfactory and nonthreatening introduction for those who lack the hacker nature.

All three books also do an excellent job of transmitting the Perl culture—the attitude, the jokes, the sense of mission, the deep connection to Unix tradition and the free-software culture. As with Linux, Perl's true strength is the collective talents of its enthusiasts, and those are well on display in these books.

Indeed, as Perl continues to spread to NT and Windows environments, it's not too much to hope that the spread of Perl culture will imply a lot of quiet subversion that prepares people for the Linux way. Even if these books were not excellent in many other ways, they would earn a warm welcome in *LJ*'s pages on that account.

Resources

Eric S. Raymond (esr@thyrsus.com) is a semi-regular *LJ* contributor who thinks Perl is pretty neat even though he still carries a torch for Scheme. You can find more of his writings, including his paper for the San Jose Perl conference, at http://www.ccil.org/~esr/.

Archive Index Issue Table of Contents

Advanced search

Web Counting with mSQL and Apache

Randy Jay Yarger

Issue #46, February 1998

Learn all about Apache modules and mSQL programming using a web counting program as an example.

This article is not about web counters. Oh, I know what the title says, and I am going to be showing you how to use mSQL to make a fast and effective counter. But that is just icing on the cake. The real purpose of this article is to explore the fascinating worlds of Apache modules and mSQL programming.

Of course, counters are far from the newest thing to hit the streets. I'd be willing to bet that you have one on your home page. However, the vast majority of counters that people use are third party off-site counters. These counters are usually implemented by including an <IMG> tag on your page to call a remote site which keeps track of the number of accesses and returns a counter image containing the current count. The problems with this method are well known: it can be expensive, it doesn't include all of your hits (especially hits from text-based browsers), and it isn't customizable.

Being Linux users, we don't settle for the easy way of doing things—we do them the right way. Every major distribution of Linux comes with the Apache web server. This gives every Linux user with a network connection the ability to run their own web site without the constraints of using third-party sites for anything, not even counters.

The Apache web server is far and away the most widely used web server on the planet. One of the features that makes it so popular (besides that fact that it is free and always will be) is its modular design. The server can easily be enhanced by writing modules that compile right into the main program. These modules are usually written in C, but there are even modules which allow you to write modules in other languages, such as Perl.

The good news is that writing an Apache module is not hard to do. Even a novice C programmer like me was able to get the hang of it fairly quickly. With that said, let's get down to the business of writing a counter module for Apache.

There are several good reasons for making your web counter a module that is a part of the server. Counters that work via CGI (such as the ones described above) do not record all possible hits to your page. On the other hand, counters that work by analyzing log files are hard to keep up to date, and they can also miss some hits (e.g., it may catch /~myhome/index.html but miss /~myhome/ or /~myhome). A counter directly integrated into the web server, however, is guaranteed to catch all possible hits since all files requested from your web server must be sent through the server. In addition, going through the web server allows you to use the name of the file that was actually sent to the user, so that instead of /~myhome, /~myhome/ and /~myhome/index.html, all hits are labelled /~myhome/index.html. The other big advantages are speed and customization. You know that the counter is going to be up to date, since every file going through the server also goes through your counter.

Now, all we have to do is decide how to store the data. As a matter of fact, a friendly member of the Apache community has done some of our work by writing an Apache module called mod_cntr.c (see Resources) which is a counter that uses plain text files or DBM databases to store the data.

As far as making a counter module goes, that one is really all you need. Download the mod_cntr.c file, put it in your Apache source directory and recompile. But being discerning Linux users, we want more power—that power is going to come from the mSQL database.

The mSQL engine is very fast at the expense of being an incomplete SQL language, but it has enough for our needs. Running our counter module through an mSQL module removes the need for cumbersome text files. All we do is call up the nearest mSQL server (whether on the same machine as the server or not) and send it the instructions needed to maintain the database.

I won't go into the details on how to obtain and install mSQL here. Complete instructions are available on the mSQL web page (See Resources). To compile our mSQL-based counter, you need the libmsql.a library in your libraries path and the msql.h header somewhere accessible.

Of course, the best way to handle this would be to add mSQL capabilities to the already existing mod_cntr.c. And in fact, this is the way I do it for my production server. But as a simple case, I'm going to rewrite the module so that it only

supports mSQL. This will remove a bit of clutter and be a bit more understandable.

Listing 1 contains the code for the mSQL-based counter. Let's first look at this from the perspective of an Apache module. The Apache module API contains very simple rules for creating modules. Lines 403-419 contain the module definition that the web server reads to define the module. There are many options here, but for our needs (and for the needs of most modules), we only need to define four of them. The first two, defined on lines 406 and 408, are **create_cntr_dir_config_rec** and **create_cntr_srv_config_rec**. These are the names of functions which initialize the variables used by our modules. Notice that there are two of these, one containing the characters “dir” and one with “srv”. Apache allows users to customize most aspects of the server on a per-directory basis using .htaccess files; this configuration is accessed with create_cntr_dir_config_rec. The main server configuration files (access.conf and httpd.conf) are accessed with create_cntr_srv_config_rec.

The code for the create_cntr_dir_config_rec and create_cntr_svr_config_rec functions are located on lines 132-146 and 148-162, respectively. The two sections of code are almost identical. Each simply initializes the counter information, contained in the structure **cntr_config_rec** (lines 106-112), with the defaults defined in lines 122-127. If you were writing a module for any purpose, you should include two functions similar to these to initialize any default variables for both the server-wide and per-directory configurations.

The next definition for our module is **cntr_cmds** at line 410. This is the name of an array of type **command_rec** which is defined in the Apache header files. These are the options which users of your module can enter into the Apache configuration files. There are six fields which must be filled out for each command_rec:

1. The name of the configuration option the user will enter. For instance, the command_rec on line 240 has the name **CounterType**. It is general practice to preface the name with **Server**, if it is an option for the server-wide configuration files.
2. The name of the function called to give the user's configuration information to the module.
3. A null pointer used to pass additional information to the function given previously.
4. A flag indicating where the configuration option can appear. In our module, we use **ACCESS_CONF** which means that the option can appear in a directory section of the server's access.conf file or in .htaccess file and **RSRC_CONF** which means that the option may appear only in the global access.conf or httpd.conf.

5. A flag indicating how many arguments the configuration option requires. In our module, we use **TAKE1** which indicates one argument (pre-parsed by the server) and **TAKE2** which indicates two arguments separated by white space. Other flags, as well as full documentation on the whole server API, can be found at http://www.apache.org/docs/misc/API.html.
6. A string describing the use of the configuration option. It is displayed when the user enters the wrong number or type of arguments for the option.

Our module defines six configuration options. **CounterType** defining the type of database to use (mSQL only, in this case, but I left the line in for easy transition to the original version of this module with text and DBM databases). **CounterAutoAdd** allows the user to decide whether or not a URI (the portion of the URL relative to the host) encountered for the first time should automatically be added to the database. **CounterDB** is the name of the mSQL database used, followed by the table within that database. There are three similar options that do the same operations for the server-wide configuration.

Each of the options defined has an associated function (**set_cntr_type**, **set_cntr_autoadd**, etc.). These functions check to see if the user supplied a value for that option. If so, that value is added to the cntr_config_rec structure for the module, replacing the default.

The final bit of information about our module is on line 417. This is the name of the function which does all of the work. In our case, that is the function **cntr_update** located on lines 331-400.

That is all there is to creating an Apache module. Once you create a C file with a module structure that has two functions to initialize default variables, a command_rec structure with user-definable options (along with functions to insert those options into your module) and a function to perform the action, you have everything you need. Of course, to do anything useful you have to code the action into that last function, so let's take a look at cntr_update.

The first interesting bit of this function occurs on lines 343 and 344. The structure **r** is of type **request_rec** (defined in the Apache headers) and is passed to us from the web server. This structure has all of the information we need about the request the server is currently handling, including the URI of the document and whether or not the document has been fully resolved. In lines 343-344, we skip all of the unresolved steps the server goes through to get the final file, so that the URI we receive is the name of the file actually sent. This way, the URIs entered into our database are of the form /directoryname/ index.html instead of /directoryname, /directoryname/ or whatever name the client-end user actually typed. On lines 348-351, we abort if there is no URI (the

user made a typo or the file doesn't exist) or the file is a server-side include (no need to keep a counter for those). We then call the function **get_module_config** for both the server-wide and per-directory cases. This is a function provided by the web server which gathers all of the user-supplied options and calls the appropriate functions (that we have provided) and returns a cntr_config_rec structure with the initialized variables.

Next, we check to make sure that the user has provided a database and table for us to work with. Assuming everything is set, we then call the **cntr_inc** function to increment the counter for the given URI. After that we set a couple of environment variables which can be used by CGI programs that may be interested in the counter (e.g., an image generator that makes an odometer graphic).

The main purpose of the cntr_inc function (lines 310-329) is to do a bit of preprocessing of the URI and then call the appropriate database incrementing function. In our case we only have one type of database, so this is a very short function. First it strips double slashes (//) from the URI and then calls the **cntr_incmsql** function to do the actual work.

As far as creating an Apache module goes, we're done. Pretty cool, eh? I certainly never imagined that creating an extension to a web server could be so easy. Now all we have to do is code the cntr_incmsql function to do the work.

The cntr_incmsql function (lines 253-308) is the heart of the module. In 55 lines we implement a fast, effective web counter using an SQL database server. The C API provided with mSQL includes several functions that allow easy access to all of mSQL's features. The first one we use is on line 264. There we call **msqlConnect** to connect us to the local database server. You can also call msqlConnect with a non-NULL argument to connect to a remote database server. In the next two lines, we check to make sure we connected successfully and return an error if we didn't. Line 269 calls **msqlSelectDB** to select the database supplied to us by the configuration options. Again, if the function does not return successfully, an error is generated.

Now that we have connected to the appropriate database, we can send our first SQL query to find out if the given URI already exists in the database. The table we query is given by the user and must have a **uri char** field which is a non-null unique index, a **cntr_count int** field and a **cntr_date char** field. The query we send is

```
select cntr_count,
cntr_date from tablename where uri='uriname'
```

This query returns the existing count of the URI (if any) and the last time it was reset. We send this query using the **msqlQuery** function and check for errors as usual. The next step is to retrieve the results (if any) from the server. We do this with the **msqlStoreResult** function. This function returns a structure which allows us to retrieve the results row by row, along with other information about the results such as the number of rows retrieved. In line 281 we use the **msqlNumRows** function to see how many rows were retried. The number should be one or zero since each URI is unique. (A full list of the SQL queries supported by mSQL can be found on the mSQL home page listed in Resources.)

If there already exists an entry for that URI, we enter the block on lines 282-293. First we use the **msqlFetchRow** function to retrieve the number of counts for the URI. Since we know we only have one row of data, we only need to call the function once. If you were expecting multiple rows of data, you can keep calling msqlFetchRow to retrieve them until the data runs out. The msqlFetchRow function returns an array which contains each field that you requested. In this case, we asked for the count and the date for the URI. We then increment the count and place the count and date into our results structure so that CGIs or other interested programs can access them. Finally, we send the update query:

```
   update
```

(with the appropriate count and uri in place). After checking for errors, we are free to close the database (using **msqlClose**) and deallocate any database memory using (**msqlFreeResult**). We can then return success and everyone is happy.

If the URI we have does not exist in the database, we have to enter it. First we check to see if anyone has defined the **CounterAutoAdd** option in the server's configuration files. If they have, then we just skip this URI and go home. If CounterAutoAdd is not defined, we have to add the URI into the database with a count of 1 and the current date. The current date is set using the server-defined function **ht_time** which returns a pre-formatted string. We do this with the following query:

```
   insert into
   cntr_date) values ( 'myuri', 1, 'currenttime' )
```

(with appropriate values for *tablename*, *myuri* and *currenttime*). Then we always check for errors. If there are no errors, the module is finished. We now have a working hit counter built right into the web server.

This module is neat, but it is not perfect. A problem known as a race condition arises if you are running a high volume server. Note that on line 275, we check the database to see if the URI already exists. If it doesn't, then on line 296 we insert the URI into the database. Suppose that somewhere between line 275

and 296, another hit comes in for the same URI. The database indicates on return that the URI does not exist; it doesn't know the first hit is about to add it to the database. By the time the second hit reaches line 296, the URI has now been entered into the database by the first, and the database returns an error due to a non-unique URI.

In a fully-featured SQL server, this problem is solved by a technique called “transactions” where multiple commands can be entered into the database together—before any other commands are processed. Hopefully, mSQL will support transactions in the near future. Until then, one workaround for this problem is to initialize your database with all of the URIs for your site and a count of 0. This way, the counter will never be told that a URI doesn't exist.

Resources

Randy Yarger (randy@yarger.tcimet.net) is the Systems Administrator and Chief Programmer for H-Net, Humanities and Social Sciences On-line (http://www.h-net.msu.edu). His sign is Virgo, his favorite color is blue, he wants to work on world peace and is currently practicing for the interview portion of the Miss America pageant.

Archive Index Issue Table of Contents

Advanced search

Linux Works for Me and You

Maan Bsat

Issue #46, February 1998

A high school student tells us about using Linux as a server at school, at home and at work.

I know a lot of people have already talked about how you can use Linux in your business. Well, I'll tell you about how Linux helped me. But first of all, you have to know that I'm 16 years old and attending high school. I live in Geneva, Switzerland.

Linux at School

At my school, officials decided to hook up to the Internet with a single 64K ISDN line. At first this may sound fast but for 50 computers, it's not. We found that when a class wanted to find some information on the Internet, it was usually stored on one server. As a result we decided to create an Intranet web server on which we would store the desired site, and everybody could access it from there. That would solve the speed problem. Since our resources were limited, I proposed that we install Linux on the machine as our operating system, and I did the install. It turned out to be a great solution in addition to being very cost-effective. I then installed Squid, a cache program, to make the connection even faster.

Linux At Home

At home, I have three computers, all networked, but only one modem to connect to my ISP. I used to use Wingate for Windows 95, but it wasn't the best solution. Someone told me about a feature called IP masquerading in Linux. With only one account, the three computers can now access the Internet. All of this software is free, of course.

Linux At Work

A company would probably not use Linux in the same way I do. Lately, there has been a lot of talk about Intranets; however, for a small company an Intranet might cost too much. Server software is quite expensive. If you want messaging capabilities with industry standard protocols, you have a choice of two solutions: commercial software for big bucks, or free software on Linux. Linux comes with sendmail, an SMTP mail server. You can access the server with a POP3 mail program, such as Netscape, Internet Explorer, Eudora, etc. Note that the client computers do not need to run Linux. They can be Macintoshes, or PCs running Windows 3.1 or Windows 95. The most basic use of a server is usually a file server. You can access the server via FTP, but doing so is not very user friendly. Another alternative is to use Microsoft's SMB protocol, or on a Linux server, you can install Samba (also free). SAMBA will allow all Windows for Workgroups and Windows 95 machines to access the file server. Files can the be stored and retrieved as if they were local.

Of course, to follow the Internet trend, you can install Apache (http://www.apache.org/). Apache is an easy to install, easy to use web server. It will allow you to present the most basic pages, and you'll also be able to put up some Java applets. Many companies use Java applets to access databases. Solid Technology supports a SQL server for Linux, and there are many free Linux databases, as seen in other articles in this issue.

If you decided to make an installation as I have described on Windows NT, you would have to pay at least 1000 Swiss francs. With Linux, apart from the cost of the computer, the same installation is nearly free. If you choose to buy a CD-ROM with a complete Linux distribution, it'll cost you about 25 Swiss francs. You make the choice.

Maan Bsat is a student at the International School of Geneva (http://www.ecolint.ch/). He can be reached at bsat@iprolink.ch.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Attaching Files to Forms

Reuven M. Lerner

Issue #46, February 1998

Mr. Lerner shows us a way to use file elements to allow web site visitors to upload information or program files to the site.

Here's a relatively easy question for anyone who has been working with the Web for a while: How would you allow visitors to your site to send you their name and address? The easiest solution would be to use an HTML form containing one or more text fields. The contents of the form would be sent to a CGI program on the site's server, which would retrieve the fields' contents.

What if we were interested in sending more than just a few words or phrases? What if we were interested in allowing visitors to our site to enter large volumes of text? We could use a <textarea> tag, which gives the user much more room in which to write. But <textarea> elements, like all HTML form elements, are still a bit clunky. Wouldn't it be nice if there were a way to simply attach a file to an HTML form, much as we can attach files to e-mail messages?

Using the relatively unknown file element, we can do just that. File elements are in many ways similar to text and hidden elements, in that they contain strings of text, rather than simple on-off indicators (as with check boxes) or one of a number of possible strings (as with radio buttons and selection lists). In addition to sending the name of the file, file elements send the contents of the file along with the HTML form.

Before we get to some practical uses for the file element, let's look at a simple example of what is possible. Our initial form shown in Listing 1 contains a single file element and a submit button. It is similar to forms that we have seen in previous installments of *At the Forge* and is probably quite similar to forms that you have seen on other web sites. The form begins with a **<form>** tag indicating the method that should be used when sending the data (**POST**) and the CGI program to which the data is to be sent (/cgi-bin/upload-file.pl). We then have a single form element, as well as a “submit” button for sending the data.

There are two differences between this form and the forms that we have seen before. For starters, the form element has a third attribute (**ENCTYPE**) that we can generally ignore, because the default value (**application/x-www-form-urlencoded**) is sufficient for most purposes. However, URL-encoding (in which characters are replaced by a percent sign followed by their ASCII code in hexadecimal, e.g., the space character becomes %20) is inefficient when used on large files, particularly when those files contain a large number of characters that require the coding. In addition, we want to separate the form elements from the file (or files) being uploaded, and we want to tag the uploaded file with a MIME-style content-type header indicating the type of data that is being sent.

For all of these reasons, using the form element requires the use of a new ENCTYPE setting (one defined in RFC 1867 available on the Web at http://www.internic.net/): **multipart/form-data**. With this encoding type, the contents of each uploaded file will be sent separately, without URL-encoding and with a "Content-type" header describing the type of data contained within. Aside from having to remember to set the encoding type explicitly at the top of any forms containing file elements, we do not have to worry too much about the way in which files are submitted to our CGI programs.

The other new element in the above form is the file element itself. When presented in HTML source code, a file element appears to be quite similar to a "text" element. We assign it a name, and the value will presumably come from the user.

The file element is different from other elements in two ways. First of all, it tells the user's browser to send not just the file name specified in the file element, but also the contents of the file associated with that name.

More obvious to the user, however, is the fact that a file element appears in the user's browser as the combination of a text field and a button. The user can enter a filename by typing into the text field, or—and this is the unusual part—she can browse through the file system using a dialog box brought up by the "browse" button. When the user selects a file to upload by using the "browse" button, the name of the file is entered into the text field, as if the user had typed it.

Receiving the File

Now that we know how to set up the form for uploading files, let's look at a small CGI program that will accept the file that was uploaded. For starters in Listing 2 we will simply have our program print the uploaded file on the screen.

Let's run through this program, in case you aren't completely familiar with CGI programs written in Perl. First of all, we start up Perl with the **-w** flag to warn us

if we are doing something particularly stupid. We also turn on diagnostics so that Perl will give us a verbose error message if and when it detects an error.

Normally, any program I write includes the line **use strict** to catch potentially dangerous or foolish constructs that I might have built. However, as you will see, we will be playing some games with references later on, and we must turn off the strict package when dealing with references so that our program does not crash. Immediately after importing the "strict" module, we thus turn off strict checking on references by using the "no" pragma (a construct for telling Perl how to handle your program).

Then, we load CGI.pm, the package that takes care of most of the dirty work for CGI programs. We create an instance of CGI and use the "header" method to send an initial MIME-style header to the user's browser, indicating that we will be sending HTML-encoded text in our response.

Next, we retrieve the value of the file name entered by the user from the form element named **userfile** and put it into a variable named **$userfile**. Until now, **$userfile** could have come from a text or hidden element as easily as from a form element.

Now comes the wild part. We use **$username** as a file handle, and iterate over it using the **<>** operator to retrieve the contents of the file. I must admit that when I first wrote programs that took advantage of uploading files, I was floored—could I really use the variable that I had assigned as a file handle? The answer is that it does indeed work rather well.

Checking the Uploaded File's Type

There is at least one problem with our program. What happens if the user uploads a GIF or JPEG image? We will end up displaying a good deal of garbage on the user's screen, since the image will be sent and then displayed as if it were HTML.

One solution is to use the **accept** attribute that can be used with the file element. In theory, **accept** should be set to one or more file types that the user should be allowed to send via the form. Thus, if we were only interested in receiving HTML files, we could say:

```
<P>File to upload: <INPUT NAME="userfile"
        TYPE="file" value=""
        accept="*.html"></P>
```

This statement would restrict the user to uploading files with the .html extension, which would presumably be HTML files. In practice, I have found that while the **accept** setting changes the filter in the window brought up by the

browse button, the **accept** setting is not enforced, and users can enter whatever they might like in the text field.

If we are truly interested in ensuring that only HTML files are uploaded to our program, we need to modify our CGI program such that it checks the Content-type header of a file before displaying it. If the file has a Content-type of **text/html**, it is considered acceptable and printed; if not, a short error message is displayed.

We can check the headers associated with a file by calling **uploadInfo** on the file name ($userfile), which returns a reference to a hash, which in turn contains all of the headers associated with a particular file. Listing 3 is a slightly modified version of our previous program which prints the headers before the file.

Once we have retrieved the headers as shown in Listing 3, it becomes relatively easy to receive only certain kinds of files. We could, for instance, add the following lines to our program to ensure that uploaded files have a Content-type of **text/html**:

```
if ($headers{"Content-Type"} ne "text/html")
        {
        print "<P>Sorry, only HTML files.</P>\n";
        exit;
        }
```

Doing Something with the File

Uploading files is fine and dandy, but the point of uploading files is to use them, not simply display them on the screen for users to see. Now that we have seen how to upload a program from the user's browser to a CGI program running on our Web server, let's try to use this program for something practical.

Let's take a simple example, one which comes from a program that I wrote for a site. Our site sat on a server rented from a web space provider, meaning that while we had control over our individual HTML files and CGI programs, system administration (including user names) was controlled by the company from which we rented the space.

The problem began when those of us working on the site decided that we wished to allow members of various affiliated organizations and subgroups to add HTML files within particular directories. That is, we created a directory for each group affiliated with our site, and expected that each group would be able to add and modify HTML files as necessary.

However, we only had a single login for our site, and we certainly didn't wish to jeopardize the site's security by giving out that user name and password to each of the 40 or 50 affiliated organizations. At the same time, given that each

organization would change no more than ten HTML files in a given month, it seemed an extreme and costly measure to order user names for each group.

We finally decided to allow users to upload files into their organization's directory—and only into that directory—using an HTML form similar to the one we saw above. Whereas the above form only requested that the user enter a file name (either by typing it in directly or by clicking on the browse button), we now asked for three additional pieces of information: the directory into which the file should be deposited, the name that the file should be given once it reaches that directory and the password for that directory.

We ask for a directory name and password to ensure that users only deposit files into the directory for which they have been given permission by the system administrators. Passwords are not a perfect security system, but they work relatively well, are portable and are easy to understand. In this way, members of group A can upload into the A directory and members of group B can upload into the B directory, and both groups can be sure that no one else is modifying the files in their directory.

One possible version of the HTML form that could be used to upload files to a CGI program is in Listing 4. Notice that in that file, I have used HTML tables to separate elements. That choice was made for purely aesthetic reasons, so that each of the form elements would line up with one another.

When the user enters a file name, directory name, password and destination name in the form and clicks on the “Upload file” button, the four form elements are sent to the CGI program (/cgi-bin/upload-file.pl) along with the contents of the file named in the file element, named userfile. We want to write a program that takes the contents of userfile and saves it in the appropriate directory (the section element) with the appropriate name (the file name element). Of course, all of this happens only if the user enters the correct password for that section.

Writing a program that does this sounds pretty straightforward, right? Well, it is; take a look at Listing 5 to see what a basic version might look like.

The password system in this program is a simple hash whose keys are the different sections and whose values are the passwords for those sections. If you have a small number of sections on your site, you can set the passwords within the program , as shown in Listing 5.

Even though it is easy to add new sections and passwords in this way, it is not a good idea for you to modify the source code for something this simple. It is a good idea to put password information in a text file, a DBM-style file (which is basically a hash saved to disk), or even a small SQL database (as we saw in a

series of *At the Forge* installments earlier this year). Then again, if the number of sections is small and doesn't change very often, you might simply stick with the example system displayed here.

In order to ensure that users do not abuse the uploading system, we remove everything in the uploaded file name up to and including the first slash. This makes it rather difficult for someone to try to deposit one of their files in someone else's directory by taking advantage of the “..” directory name, an option that means “use this directory's parent”.

After checking to make sure that we have received all of the required information, we compare the correct password with what the user entered:

```
&log_and_die("Incorrect password")
        unless ($PASSWORD{$section} eq $password);
```

Now that we have established that we have all of the needed information and that the user is authorized, we save the file to disk using a simple “while” loop that reads from $userfile (which can be treated as a file handle) and printing to FILE, the handle that we created for saving information to disk in Listing 5:

```
open (FILE, ">$saved_filename") || &log_and_die(
           "Cannot write to $saved_filename: $! ");
        while (<$userfile>)
        {
        print FILE;
        }
close (FILE);
```

Finally, let's look at **log_and_die** shown at the end of Listing 5, a subroutine I include in many CGI programs, which allows us to die relatively gracefully with a reasonable message sent to the user and the error log. This is a far better way for the program to crash than producing the unfriendly “500 Server error” message that is all too common on web sites these days.

When we execute statements such as:

```
open (FILE, ">$saved_filename") || &log_and_die(
        "Cannot write to $saved_filename: $! ");
```

we are saying, “Open the file $saved_filename for writing, and allow me to write to $saved_filename using the FILE file handle. But if you cannot open the file, send a message to the user indicating that you cannot open it, along with $!, Perl's special variable containing the most recent error message.” Not only is this message better for visitors to your site, it can be more useful to you when debugging your program.

There are a few drawbacks to this program that deserve mention. The lack of an intelligent backup system (which would be relatively easy to add), the

inability to deal with subdirectories (useful when you want to separate images from text) and a fairly primitive user interface are all strikes against this system. But over time, this has proved to be a good compromise between insecurity (i.e., giving everyone affiliated with the site access via the same user name and password) and expense (i.e., buying a new account for each affiliated group, even though the accounts will rarely be used).

The file tag is not something that you will use every day, but when you need to allow users to upload information to your server, it can be quite handy.

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. You can reach him at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #46, February 1998

Readers sound off.

The Fujitsu 420D

The Fujitsu 400 series [*Lifebook 420D Notebook Computer*] that I reviewed in November has recently been discontinued in favor of the Fujitsu 700 series. I have been assured by Fujitsu Technical Support that they will continue to honor their warranties and support "legacy" systems such as these. Anyone who succeeds in picking up one extra cheap can be reasonably sure of support for the foreseeable future. Still, it's a bit alarming that a six-month-old machine can be thought of as a "legacy" system. This industry is just nuts sometimes.

—Michael Scott Shappe mshapp19@idt.net

I2O Restrictions

I read the comments about the new I2O device-driver architecture by Phil Hughes just days before my *Embedded Systems Programming* magazine arrived. In it was an article by Larry Mittag describing I2O in some detail. He did mention the NDA and license restrictions. He reported that the reason for these restrictions is not the I2OSIG itself, but the lawyers and owners of the software patents for this architecture. Apparently, this architecture has been used before in mainframes and the techniques were patented. Thus, it cannot be freely implemented by others unless a license fee is paid.

For this reason, an e-mail campaign to I2OSIG will not be successful. After all, they must comply with the law. However, it sure would have been nice of them to communicate with *Linux Journal* to clarify the situation.

The League for Programming Freedom (http://www.lpf.org/) is fighting to overturn the concept of software patents. This would not prohibit programmers from copyrighting their code; it just means that an idea—which is

what an algorithm is—could not be patented. Their belief and mine is that patents are for physical devices and processes, not for mathematical algorithms.

The only way I see to create a unified driver environment is for developers to refuse to use the I2O specification.

—Jeff Root

Noweb

It was nice to see an article presenting noweb, one of my favorite tools [*Literate Programming with Noweb*, Andrew Johnson and Brad Johnson, October 1997].

Although the article did a good job of describing the technical intricacies of creating a noweb-literate program, it did not properly present the idea behind literate programming (LP), nor did it convey the idea that LP can be used by serious software developers. Noweb does not require that the source be in a single file; my preference is to put each project component in a separate directory and to use one noweb source file per subcomponent.

One of the key LP advantages is that the documentation is next to the code, in the same file. Other LP tools that also support multiple programming languages are nuweb and FunnelWeb; check the LP FAQ.

Another noweb feature is that the tangled (extracted) code is readable, indentation and line breaks are respected; therefore, the tangled code can be distributed as if it were the actual source.

Users unfamiliar with LaTeX might be pleased to know that there are several noweb modes for Emacs (I wrote one of them), and that with color highlighting the source file becomes quite readable.

Those interested in LP applied to software engineering can check the low traffic moderated newsgroup comp.programming.literate, and the following books: Knuth's *The Stanford GraphBase* (Addison Wesley 1993), Fraser and Hanson's *A Retargetable C Compiler: Design and Implementation* (Benjamin/Cummings 1995) and Hanson's *C Interfaces and Implementations* (Addison Wesley 1997).

Noweb works perfectly under Linux (or any Unix variant) and there are also versions for Windows 95. Wouldn't it be nice if the full Linux kernel sources were available in noweb format and published as a book? After merging with the Kernel Hacker's Guide it could be used in Operating System courses and perhaps become the next standard format for kernel sources.

—Alexandre Valente Sousa avs@ismai.pt

Comments on *LJ* issue 43

I was looking forward to receiving my copy of *Linux Journal* Issue 43 (November 1997). After checking the *LJ* web site I was drooling about reading of the GIMP and faxing from Macs. However, when my copy arrived I was disappointed at seeing no examples of the GIMP in use. How on earth can you cover a graphics program without any graphics? Sadly, this article seems to have set a precedent as the later article on *Linux as a Telephony Platform* by David Sugar was also without any illustrations. I hope that this trend does not continue.

Our office FAX machine is antiquated and in need of replacement with something that we can use from our desktop Macs. So I was also expecting good things in *Faxing From a Web Page* (using HylaFAX on the Mac) by David Weis, but sadly the short article did not address the issues in any depth. An example of the web page used would have been preferable to reproducing just the HylaFAX logo.

They say that if you're going to criticize something or someone then you have to make three positive comments. First, the *Linux Means Business* article [*Highway POS System*, Marc L. Allen] was interesting to read. I've worked on EPOS systems so I know some of the pitfalls especially when trying to use MS-DOS machines. Although short, this article did capture the author's obvious enthusiasm for Linux. This series of articles has always been inspiring and thought provoking. Second, the update on IP Masquerading was very helpful. Keeping up to date with all that is happening with Linux kernel issues is not easy, so this article was a timely reminder of what is actually happening. (It also had some illustrations, examples and figures to support the text.) Third, the *Take Command, ssh: Secure Shell* [Alessandro Rubini] article also served as a timely reminder to be careful out there.

—Trevor Jenkins Trevor.Jenkins@suneidesis.com

Three negatives and three positives—a well-balanced letter. Michael sent in a very long article on the GIMP that just wouldn't fit in one issue, so we requested that he split it into four parts. Our November cover was built with the GIMP and used the graphic that went with this first purely introductory article. We always request graphics and images to go along with articles, but it doesn't always work out—this was the case with the two other articles you mention.

November Issue

I have every issue of the *Linux Journal*. The issues vary from good to great, which is really just another way of saying that the subject matter of some issues is of more interest to me than others. However, with the November issue, you outdid yourself. I am not sure I can complete it before the December issue arrives. Every article and column was worth a read and then a re-read. Excellent, keep up the good work.

—Richard Parry rparry@qualcomm.com

Newbie

Please allow me to describe my feelings about all of this: Linux is over my head.

I'm no slouch when it comes to computers. I'm freshly trained, and now on my own as a system administrator with a few Suns, an SGI, an Intergraph and 1 PC (Win95) on a subnet. While my knowledge base may be limited, I'm catching up rapidly. I'm also not a programmer, nor is anyone I work with—they are all just operators. So, hacking a kernel, writing code and everything else that seems to be necessary to use Linux at home is beyond my capabilities and those of most people I know.

I like the Unix environment at work and would love to have the same control at home. As long as I could do on Linux the simple stuff I enjoy on a Windows machine, I'd be happy. Things like surf the web, balance my checkbook, pay bills and e-mail my parents. I know I could if I could only code, but I can't; so, Linux doesn't seem to be for me. Until it is accessible to people like me, the vendors who support Linux will never make the money that that “other” OS does.

—Searoy@aol.com

More and more applications are being written to make Linux more accessible to the average user. E-mail can be done through your ISP using Eudora or other mail programs. Surfing the web has been available for some time. The majority of web servers are Apache servers running on Linux boxes, and there are many browsers that work with Linux—even Netscape. Applixware and StarOffice take care of word processing and other office needs. Databases abound as this issue proves. Check it all out—it may be easier than you think.

MagicFilter

I have been getting *Linux Journal* since the first issue, and it has always been a great resource. Although I have twenty years experience with Unix and systems

work, I still have days like last Tuesday. All I wanted to do was move my printer to a new machine that had a new version of Slackware. I spent all day and never got it working. Argggghhhh! Yesterday, my new *LJ* [November 1997] came in, I read Bill Cunningham's article *Power Printing with MagicFilter* last night, came to work this morning and got the printer up and running in no time. Thanks again for the right article at the right time.

—Dennis Director dennis@is3.com

November cover

I just wanted to comment on the November 1997 cover of *Linux Journal*. It's by far the best one to date, really fantastic! The articles aren't bad either.

—John Wagner jwag@together.net

Kudos from India

This off-and-on *LJ* subscriber in faraway India feels a terrible urge to let you know that he thinks *LJ* #43 (November 1997) is a great issue.

I am a Linux user who has had to get things the hard way. When I started (with an SLS distribution on diskettes), Linux was unheard of here in India. As times went by, its popularity grew, but resources were hard to find—no Internet in India until 1995 was one of the main problems.

I subscribed to *LJ*, and bless you guys every day (I am a datacomm consultant for huge multinational corporations operating in India and Linux is my main battle axe).

In the past two years, India's #1 computer magazine (*PC Quest*,http://www.pcquest.com/) has distributed Linux on its cover CDs twice. This has helped the number of Linux users in India shoot up in a near vertical fashion. India has always been known as “Unix country”; since we have the world's largest pool of technically-qualified, English-speaking professionals, knowledge of any kind of Unix is a major plus point when one goes job hunting. Many kids are now playing with Linux, which gives them a huge edge when their time comes to pass out resumes.

Unfortunately, *LJ* is not a “stand-mag” around here, and no local subscription points is another problem (subscribing in US$ is a major pain here). It would be great if you guys could tie up with someone in India, so that more people could get the point. Here, as in many countries, a field/application/OS is treated as *serious* only if it has publications associated with it.

Keep up the good work. You have an excellent magazine, and your work for the Linux cause is not going unnoticed—even here in far-away India.

—Atul Chitnis achitnis@cbconsulting.com

Imagine1 URL

The URL for downloading the free Linux F compiler from Imagine1, Inc. was left out of my article, *Portability and Power with the F Programming Language*, published in the October issue of *LJ*. The correct URL is http://www.imagine1.com/imagine1/.

—David Epstein david@imagine1.com

Comment on your article

I'd like to comment on Mr. Hughes' article in the November issue of *Linux Journal*.

He offered four suggestions that would make Linux a better solution. Two of them were application programs or packages. I agree that creating new applications would be useful, but I'm also concerned with existing Unix freeware that has yet to be ported or compiled on Linux. There is a huge amount of this freeware available but getting it to build on a Linux box is something else again.

What is needed is a software porting project: a central site to collect information about porting specific applications to Linux. Information such as what changes are needed in the makefile, imakefile or prog.tmpl to get the program to compile. Additionally, information about the problems encountered in trying to compile the app would be very useful. This kind of information would help others who are trying to port applications to Linux. So, the site would be a learning resource as well, and this is perhaps most important.

As the knowledge and ability to port software to Linux increases, the number of applications ported will increase at a faster rate. The site wouldn't need to archive binaries or source code, just the specifics about how to compile and the problems encountered. In addition, it would be useful to include information on who's currently working on which application. This would allow communication between groups or individuals working on the same problem and prevent duplication of effort. More applications will make Linux a better solution, and the easiest way to get them is to port existing Unix freeware. A dedicated site would greatly facilitate the endeavor by effectively harnessing the knowledge and ability of the Linux community.

—George Timmons gwtim@2xtreme.net

vi, most used text editor

First of all, let me tell you how much I appreciate *LJ*, for the accuracy of the information it gives and the nice tone it uses.

I write you in reaction to the *1997 Readers' Choice Award* that the text editor **vi** received in the December issue. I'm an old addict of this program, and I still find it fast and useful. But I have always promised myself that I would learn Emacs one of these days, thinking I couldn't remain an old dinosaur using such an old tool. The problem is that I always found Emacs far too complicated. So when I saw the Award, I was genuinely surprised so many people think like me and I had a good laugh.

In short, after reading *LJ*, I decided that, despite vi being an old and ugly editor, I'll keep using it without any remorse for being an Emacs loser.

—Pierre-Philippe Coupard coupard@mipnet.fr

Archive Index Issue Table of Contents

Advanced search

Databases

Marjorie Richardson

Issue #46, February 1998

Most of the articles discuss freely available databases; however, we don't want to forget all of the commercial databases available for Linux.

Our focus this month is databases. Most of the articles discuss freely available databases; however, we don't want to forget all of the commercial databases available for Linux. So, I thought I'd mention a few here, along with the month we published reviews or articles about them to give you a point of reference for more information. Solid Technology's Solid Server won our Readers' Choice Award for favorite database in December 1997 and was reviewed in our September 1997 issue. Also, in September, we had reviews of Empress RDBMS for professionals and Just Logic/SQL RDBMS. The Raima Database Manager and Velocis Server were reviewed in December and, in October, were featured in the *Linux Means Business* column about Grundig TV.

There are, of course, more commercial databases that we have not reviewed such as Adabas, C-tree Plus and Yard (upcoming)--and those aren't all. It's quite amazing when you start counting just how many databases do support Linux. A database is one of the basic “must have” applications for any business, so it is good to see so many supporting Linux.

As to the “big 2”, Sybase and Oracle: Sybase sells an official Linux version but refuses to support it, and Oracle has had an “in-house” port to Linux for some time that they neither support nor sell. Those of you who work for Linux companies that need the size and robustness offered by these two databases should let the companies know that there is a market for Linux versions and that supporting them would be worthwhile.

In the September, October and November installments of *At the Forge*, Reuven Lerner told us all about another free database, MySQL. The programmers working on MySQL (http://www.tcx.se/) have been working on a comprehensive benchmark suite. It is written in Perl using the DBI/DBD interface so that it can

be used easily for a wide range of databases. All of the benchmarks generate tables of data and are configurable to fit different needs. For more information visit the MySQL web site or write David Axmark at david@detron.se.

Linux in the News

A very interesting article about Linux appeared in *Network World*, October 13, 1997. The article is "Linux Flexes its Internet Muscle" by John Cox and can be found on the Web at http://www.nwfusion.com/news/1013linux.html. In it, Mr. Cox discusses the Corel Computer Corporation announcement that Corel is using Linux as the operating system for its new Video Network Computer. He also mentions other companies that have switched to Linux and why. I am very happy to see Linux receive this kind of publicity and to see companies talking about their use of Linux and why they chose it over other operating systems.

The Corel Computer announcement has been quite exciting to everyone in the Linux community—I've received a lot of mail from people making sure I heard about it. More information on Corel Computer's use of the Linux operating system can be found in their press release at http://www.corelcomputer.com/news/press/vnc_october97.htm. We will be publishing an interview with Corel's president, Mr. Eid Eid, in our April issue which will focus on workplace solutions.

86Open

Another interesting item that has been brought to my attention is the 86open project. This group has come up with a standard for creating software that will run without modification or emulation on any Unix/Intel platform, including Linux, BSDI, FreeBSD, SCO OpenServer, Sunsoft SolarisX86 and SCO UnixWare. For more information on this project, see the web site http://www.telly.org/86open/.

University of Washington Computer Fair

For those of you who will be in the Seattle area March 18 and 19, UW is holding its 24th annual Computer Fair. It is the oldest and largest computer show in the Pacific Northwest, bringing together people from the University and the community for presentations and demonstrations of state-of-the-art computer equipment, software and support materials. Exhibits and seminars are designed to show how computer and network technologies can change the way we learn, communicate and work. Better yet, it is free. For more information write compfair@u.washington.edu or visit the web site at http://www.washington.edu/compfair/.

Archive Index Issue Table of Contents

Advanced search

Needed: Linux Banking Software

Phil Hughes

Issue #46, February 1998

Instead of approximately 30 keystrokes to do my transaction (deposit three checks with cash back), I saw as many keystrokes interspersed with about half a dozen mouse operations.

Last weekend my credit union (I won't mention their name in order to avoid embarrassing them) did a computer conversion. Conversion is their word—they didn't say upgrade. They replaced their aging Data General system with a Windows 95 system. You can guess the kind—big video displays, stereo speakers on the display, keyboard in a drawer and, amazing as it seems, a mouse.

This isn't a little credit union—they have multiple offices, their own set of ATMs and, at my local branch, six teller windows. What was the first thing that made the conversion obvious? It was the long line—not something I expected, because it's usually not there. I talked to a teller, who thought the line was her fault because she wasn't used to the new system. She was wrong.

I watched as she did my transaction, as I have watched tellers at this credit union for over 10 years. (I even witnessed the conversion to the Data General system.) What I saw this time was a new way of doing transaction processing. Instead of approximately 30 keystrokes to do my transaction (deposit three checks with cash back), I saw as many keystrokes interspersed with about half a dozen mouse operations.

I told her it was impossible to do the job quickly when a mouse is involved, pointing out how easily a selection mistake can be made. I even told her I had been thinking about applications such as a credit union transaction-processing system and how an operating system called Linux was a better platform for building such a system. She asked me to let her know when I had this new system available for use.

Now, I'm doing my job as a Linux evangelist, and *Linux Journal* is doing its job to show people that some businesses have found the right use for Linux. What's missing is someone to get all the necessary applications together.

I expect it is possible to write a well designed transaction-processing system on MS Windows, but it certainly would leave out all of that wonderful drag-and-drop stuff. In a past life as a systems design specialist, I was the guy who had to evaluate the user's requirements and come up with the proper solution. Asking a bank teller to use a mouse when a typical transaction involves entering an account number, some check numbers and some dollar amounts (and nothing else) does not make sense. Doing so means you must build a system which includes the overhead of a mouse-oriented graphical operating system—overhead that is just not necessary in this case.

If this same system was built with Linux, the hardware and software costs would be reduced, as well as the programming effort. And, bottom line—we would have Linux out there solving yet another problem.

If any of this strikes home, you should take a look at the discussion groups on the *Linux Journal* web page (http://www.linuxjournal.com/). We put them there to encourage the community to get talking and get programming on the applications that will make Linux the best solution for a variety of problems. Oh, and if you already have a Linux-based credit union package, let me know—there's this credit union in Seattle that could use your help.

Changes at *Linux Journal*

On a totally different subject, there have been some changes at *LJ*. Margie told you about the subscription outsourcing fiasco last month. With subscription processing returned to our office and Linux systems, Ellen Dahl has returned to handle your subscriptions (info@linuxjournal.com).

Another change is that Jennifer Davies has joined the *LJ* staff as my assistant. We expect the combination of her Masters in Library Science and her winning personality will make her the right addition to the team.

Finally, almost four years ago, Lydia Kinata applied for a job with us doing layout. We didn't have an opening in layout at that time, so Lydia was hired on in Order Processing and since then has done almost every other job available in the company, including customer service, products development, A/R, A/P, payroll and office management. She has handled all of these jobs efficiently and well. A current opening in layout will give her the opportunity to work in her area of expertise. Beginning with the March issue, Lydia will be our new Layout Artist.

COMDEX/Fall '97

Carlie Fairchild

Issue #46, February 1998

Not only did most people we spoke with know about Linux, but many of them are using it and very excited with their results.

Photo Album

Las Vegas, Nevada is host each year to one of the largest technology trade shows in the U.S. —COMDEX/Fall. This year nearly 220,000 industry professionals lined up to find, test and research the latest technologies from the leading industry vendors.

Earlier in the year the staff of *Linux Journal* volunteered to coordinate the COMDEX/Fall Linux Pavilion. Coordinating the event turned out to mean hours of preparation, and, luckily, vendors were quick to lend a hand. Kit Cosper of Linux Hardware Solutions managed to talk the spirit of Linux into Softbank, the sponsor of the COMDEX show. As a result, COMDEX personnel were very cooperative and worked with us to ensure that the floor space for the pavilion was in the best possible site; that is, we weren't hidden away in a back corner.

Attendees seemed pleased to find many of their favorite Linux vendors in one convenient and easy-to-find area. Vendors present included Caldera, Linux Hardware Solutions, Enhanced Software Technologies, S.u.S.E., Red Hat Software, Hard Data, Quant-X, InfoMagic, LinuxMall, Linux International and, of course, *Linux Journal*.

Jon “maddog” Hall barely held his own against the hordes of Linux enthusiasts visiting the Linux International booth. Several members of the Linux community kindly volunteered their time to staff the Linux International booth, answering questions and spreading the word about Linux. Volunteers included Marc Merlin, Ira Abramov, Dan Peri and Richard Demanowski.

Red Hat Software announced the December 1 release of Red Hat Linux 5.0. To mark the event, Red Hat balloons filled the Linux Pavilion area of the convention center. The Linux mascot, Tux the penguin, was carried away in all of the excitement (see photo).

S.u.S.E., a popular European Linux vendor, also announced the latest release of their Linux distribution, S.u.S.E. 5.1. This was S.u.S.E.'s first appearance at COMDEX, and considering their rapid growth in the U.S. market, it will most likely not be their last. Their distribution demonstrations proved to be great crowd pleasers, compliments of Bodo, Rolf, Michael and James Gray, the President of S.u.S.E. U.S. (See review of S.u.S.E. in this issue.)

Clarica Grove, Britta Kuybus and I staffed the *Linux Journal* booth. We were quite pleased with the turnout of this year's show. During last year's COMDEX, we were kept busy explaining what Linux is to all comers. We were pleased to find that this year's COMDEX attendees had remembered and done their homework from last year. Not only did most people we spoke with know about Linux, but many of them are using it and very excited with their results. It goes to show that the popularity of Linux is indeed growing. Linux is being looked at more than ever as a cost-effective, viable operating system. Thanks to years of dedicated work by all of the Linux vendors, Linux International and the Linux community, we are now able to begin enjoying the success of Linux. This year's COMDEX Linux Pavilion was a showcase of this success.

Linux Journal would like to thank everyone involved with this year's show—we look forward to seeing you there next year.

Carlie Fairchild is an Advertising Representative and Marketing expert for *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Setting Up E-mail

Jonathan Walther

Issue #46, February 1998

This article gives step-by-step instructions to set up your e-mail configuration properly and an overview of various pieces of e-mail software.

In my time on IRC (Internet Relay Chat), I found that the number one problem newbies have, after setting up PPP, is e-mail setup. Many people get their e-mail half-working and leave it at that, for fear of breaking it and not knowing how to fix it.

Mail handling is split up between several types of programs, instead of having everything handled by a single program (such as Eudora). Trust me, it is conceptually easy, and once you understand it, very quick and easy to set up. It also gives you a lot of flexibility.

I've included three sidebars which you may wish to read first; in particular, the Glossary for definitions of terms I will be using. Although there are alternative programs, I'll discuss the ones that I use.

Glossary

Prerequisites

First, you need to get an e-mail account set up by your ISP with mail delivered to their machine. Then, you need to know your ISP's domain and your password on the mail server (usually the same as the password you use when you start up PPP). For the examples in this article, we will call the ISP domain foo.com, the user name barney and the password f00bar.

To begin, you need the following programs or their alternatives:

- pine [or elm]
- either smail or sendmail

- fetchpop [or fetchmail]
- procmail

To find out if these programs are installed on your system, use the command **which**. If you type **which *program*** at the prompt, the pathname of that program, if it exists in your path, will be returned to the screen. For example, **which** might return the path /usr/bin/*program*, letting you know that *program* is in the /usr/bin/ directory. All of the basic distributions (Slackware, Debian and Red Hat) usually include all of these programs except fetchpop, so you should have no problem finding and installing them.

fetchpop sources can be downloaded from http://snakepit.wasteland.org/fetchpop.tgz (special thanks to TheAsp for hosting this prepatched version on his server so downloading it from Sunsite is unnecessary).

Pine

For this example, Pine is your MUA (mail user agent). You use it to read and write mail. Start up pine by typing **pine** at the prompt. When you see the opening screen, type **s** to call up the setup menu, then type **c** to go to the configuration screen. Now do the following steps.

1. Press **enter** and type in the personal name you wish to be associated with your actual e-mail address. Mixed case and spaces are allowed (e.g., Barney Fallon). In this case, other people will see your address as Barney Fallon (barney@foo.com).
2. Set smtp-server to localhost to ensure that mail gets handed to your local MTA (mail transport agent), where you don't have to worry about it. The MTA then sends it off the next time you connect. (See Glossary and Generic Mail Flow sidebars.)
3. Leave the other options set to the defaults for now.

To send mail from the prompt or in a script, type:

```
pine person@address.com < message.file
```

Smail/Sendmail

Use smail or sendmail as the MTA (mail transport agent). You use it to transport mail between machines. People have written 600 page books on setting up these programs. The horror stories are numerous. For this simple application, in most cases, you don't have to do any set up. If you use Slackware, Debian or Red Hat, the default setups for smail and sendmail work fine right out of the box.

Some installation scripts may ask you whether to use a smarthost, which means it will hand mail to your ISP's mail server so it can do the delivery. That decision is up to you. There's not much difference. If you plan to receive mail on your machine, you might want to look at the /etc/aliases file some day.

If your user name at your ISP is barney@foo.com and you wish this name to show up in the From: header, use sendmail to handle it by taking the following steps:

1. Create a user named barney on your system.
2. Edit the /etc/sendmail.cf file line that begins with the characters #DM by changing it to masquerade the Domain (DM—Domain Masquerade), e.g., DMfoo.com. If your machine has a hostname, but not a domain name (i.e., it is not networked except via your ISP), you may also need to edit the line #Dj$w.Foo.COM to remove the # and change Foo.COM to your ISP's domain name. Finally, edit the DS line to name the Domain Smart mail host, if you wish your ISP to actually forward all your mail.
3. Once the changes are made and saved, just execute **SIGHUP<\! s>sendmail**, so sendmail will re-read the sendmail.cf file.

Sometimes smail or sendmail will die. You will then get a pine error message saying the SMTP connection is not available . If this occurs, log in as root and type **smail -bd** or **sendmail -bd**. If this type of failure happens consistently, the first thing to check is your /etc/rc* directories to be sure that your SMTP server is started at boot up.

Checking the Mail

Sendmail and smail include two very handy programs for mail checking:

- **mailq** lets you see any mail that is in the queue (i.e., not sent). If you aren't connected, you can use mailq to check the queue and to remove a mail, if you decide not to send it.
- **runq** gives smail and sendmail a “kick in the pants” to start sending out the mail that's in the outgoing queue. This command is useful if your sendmail or smail is set up to send the mail at some specified interval, and you want to send it earlier. Many installations have the default interval set to 60 minutes. Not all distributions include runq. Another way to accomplish the same result is to type **sendmail<\!s>-q** at the prompt.

Fetchpop

I chose fetchpop because I think it's the easiest program to set up. Download the sources (since it comes with none of the major distributions), compile it and install it as root. If you are brave, you can download it from Sunsite and apply

the patch yourself, but the source at the URL given earlier has the patch already applied, so it will work for everyone. Without the patch, it will fail to connect to certain POP (Post Office Protocol) servers, with no damage.

Now, run fetchpop without parameters. (Substitute your personal information in the place of the example names used below.) Fetchpop will prompt you for your POP server. Enter **foo.com**. Then, it will ask for your user name and password. Enter **barney**, then **f00bar**. Fetchpop will then write the information to the ~/.fetchhost file. Then, to get your mail, type:

```
fetchpop -arbp
```

In the testing stages, to reassure yourself that it works, use only the **-a** option. If you use the **-r** option, any messages you fetch are removed. The **-p** option causes fetchpop to filter the mail through procmail. Without the **-p** option, the mail is dumped in your default mailbox (see Mailboxes sidebar), /var/spool/mail/barney.

For more information on fetchpop, read the man page—it explains a lot of other details.

Procmail

Procmail is an MDA (mail delivery agent) and is used for “sorting” e-mail. The standard method is for the MTA to pass mail to procmail one message at a time, where it then checks its configuration file and decides what to do based on what's in the mail. Normally, it puts it in an appropriate mailbox. If you are a concerned parent, you could use procmail to direct all mail containing swear words to /dev/null. If you do a lot of hacking, you can set procmail up so a special “trigger” e-mail would encrypt your hard drive—a quick e-mail containing the code word will lock the box up tight. Or suppose you receive an e-mail from someone important; mail from a particular address could invoke a script to beep you on your pager.

To ensure that your incoming mail is filtered through procmail, create a file called ~/.forward containing the following line:

```
|exec /usr/bin/procmail
```

Then, change permissions on this file using the command:

```
chmod 644 ~/.forward
```

Check the sendmail man page for the particulars on other uses of the ~/.forward file.

Edit ~/.procmailrc by adding the following lines at the beginning:

```
MAILDIR=$HOME/mail
DEFAULT=misc
```

The default misc folder is the mailbox that receives all mail not put in another box. Generally, this will be personal mail.

Let's put our first “filter” in the file. Here is your basic “recipe” in all its glory:

```
:0
* ^.*linux-kernel@vger.rutgers.edu
linux-kernel
```

Let's break it down line by line.

:0 indicates the start of a new recipe. You have to use it whenever you put in a new rule for sorting mail.

Each rule line must begin with an * (asterisk) followed by a space. What does **^.*linux-kernel@vger.rutgers.edu** mean? The **^** (caret) means “at the beginning of a line”. The **.** (period) means “match any character”. The * means match any number of the preceding character(s). So, **.*** means match any number of characters that don't match the following characters. linux-kernel@vger.rutgers.edu is the name of the Linux Kernel mailing list. Altogether, that rule matches any line in the e-mail header that contains linux-kernel@vger.rutgers.edu—if a match is found, it's a safe bet the e-mail came from that mailing list. That rule has worked fine for me for several months now.

Now, the final line, **linux-kernel**, tells procmail that if the rule above was matched, to store that piece of mail in the mailbox named linux-kernel.

That was easy, eh? Let's do another example.

```
:0
* ^Reply-To:.*mindanao-l@MINDANAO.COM
mindanao-list
```

Much like before, except to pick out mail from this mailing list, I only need to look for lines that have the mindanao address in the Reply-To: field of the header. If I put *** ^.*mindanao-l@MINDANAO.COM**, I might find that personal e-mail that were carbon copied to that mailing list would also get sent to the mailing list mailbox instead of my private one.

Here's one final example. Sometimes, two separate rules are needed to sort all the mail from one mailing list. If I were more knowledgeable about regular expressions, I could easily condense these two rules into one. But I'm not, so here they are:

```
:0
* ^To:.*wmaker@eosys.com
WindowMaker
:0
* ^Cc:.*wmaker@eosys.com
WindowMaker
```

Can you see what these rules do? They ensure that all mail, having lines that begin with **To:** or **Cc:** and contain wmaker@eosys.com elsewhere in the line, is put into the WindowMaker mailbox. Look at the man pages for **procmailrc** and **procmailex** to get more information.

If you already have mail and want to filter it through procmail for resorting, use the commands **cat** and **formail** as follows:

```
cat mailbox | formail -s procmail
```

This command is handy if you discover mail has been sorted into the wrong folder. Tighten up the "rules" in your ~/.procmailrc file (in the same manner as shown above) and run it through again. For instance, if I find mail in my mailbox which should have gone into the debian-list folder, I type:

```
mv misc temp; cat temp|formail -s procmail
```

I can then check the misc file to see if I tightened the rules enough to result in the proper sorting.

To do thorough filtering, I recommend that you learn to use regular expressions effectively. An excellent resource is the book from O'Reilly & Associates, *Mastering Regular Expressions*, Jeffrey E. F. Friedl, 1997. The man page for **egrep** can provide a quick reference for regular expressions.

Conclusion

You should now be able to send, receive, sort, read and reply to e-mail on your very own Linux machine. From here, I would recommend you read the EMAIL-HOWTO, the various man pages and documentation for each program.

Once your setup is working, try experimenting with different programs. There are several alternatives to procmail, e.g., **deliver** and **mailagent**. Instead of fetchpop, you could use the popular programs **fetchmail** or **popclient**. Many other MUAs, such as **mail** and **elm**, work quite well. As a substitute for sendmail or smail, install **qmail**, which is quite popular due to its speed, reliability and simplicity compared to sendmail.

Generic Mail Flow

Mailboxes

Credits

Jonathan Walther enjoys hanging out on MOOs and coding various little programs. Currently, he's looking for any Unix sysadmin/coding related work at an entry level, hopefully somewhere in western Canada where he lives. He just LOOOOOOVES getting e-mail at krooger@kurgo.ml.org. Look for SirDibos on IRC and the various MOOs.

Archive Index Issue Table of Contents

Advanced search

ISPELL: Spelling Checker

Marjorie Richardson

Issue #46, February 1998

Not a perfect speller? This is the command for you.

As a former Technical Editor, I know how easy it is to miss incorrect spelling when proof-reading, especially if the word “looks” right, e.g., compatability (sic). For this reason, a good spelling checker is a must. The command **ispell** does a good job and has special features to help it do even better. The Man page for **ispell** is very comprehensive, so I won't go into all its options—only my favorites.

When **ispell** has been invoked and it finds a misspelled word, options are displayed in one line across the bottom of the screen:

```
[SP] <number> R)epel A)ccept I)nsert L)ookup
 U)ncap Q)uit e(X)it or ? for help
```

All you have to do is press the space bar (accept this time only) or *A* (accept for rest of document) to accept the spelling as is, press *I* to insert the word in the dictionary, or press the appropriate number or *R* to replace it. The main thing to watch out for is the right time to use *R*. When a misspelled word is found and the spelling choices are offered, the tendency is to press *R* for *replace* and enter the number of the correct choice—doing this results in the number replacing your word. Instead, enter the number of your choice immediately, and since *replace* is the default, the correct spelling will replace the incorrect in the text. Use *R* only when a correct spelling is not offered by **ispell**.

Most of SSC's reference cards and command summaries use troff text formatting; other manuals use TeX. Use the option **-n** with troff text or **-t** with TeX or LaTeX, and **ispell** will ignore formatting commands, thereby returning fewer “misspelled” words for you to *accept*. While an option is not available to designate a Quark file, you can always insert the QuarkXPress formatting

commands into your personal dictionary the first time they come up and not be hassled again.

In fact, the personal dictionary is probably the neatest feature of all. The very first time you select *I* to *insert* a word it doesn't recognize, **ispell** sets up a personal dictionary named ispell_english in your home directory. After that, any word you select will be added to this dictionary, and you will never be told it is misspelled again. This feature is particularly handy for proper names, buzz words and abbreviations unique to your business. Hashed dictionaries for other languages (that have been installed) can be specified using **-d**. In addition, you can set up special dictionaries for particular projects. For example, when I was editing the Java Reference Cards, I set up a special dictionary named ispell_java just for Java terms in my work directory. Afterwards, whenever I ran **ispell**, I specified the command line as:

```
ispell -n -p ./ispell_java java.troff
```

As a result, **ispell** knew class names like **getFontList** were spelled correctly, and that **getFontlist** was not. By the way, don't forget that the command line specification must include the directory of the dictionary (**./** in the above example); otherwise **ispell** will look for it in your home directory.

Another handy feature to remember is how to check a single word instead of a complete file by using the **-a** option. For example, if you specify:

```
echo compatability | ispell -a
```

ispell will return the one line message:

```
&compatability 3 0: comparability, compatibility,
computability
```

This message tells you “compatability” is misspelled, and gives you a list of 3 *best guesses* in alphabetical order. If you prefer not to have the list sorted alphabetically, use the **-S** option, and it will be sorted by *best guess*.

All in all, **ispell** is an effective and easy-to-use all-purpose spell checker.

Marjorie Richardson is Managing Editor of *Linux Journal* and the Editor of *Linux Gazette*. She had been a programmer in the oil industry for 20 years before coming to SSC. She likes to quilt, read science fiction, watch action movies and musicals, go to the opera and camp with her husband, Riley. She can be reached via e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

United Railway Signal Group, Inc

Lester Hightower

Hank Leininger

Issue #46, February 1998

The story of how Progressive Computer Concepts has turned United Railway into a Linux shop.

I remember being amazed and somewhat impressed in September of 1995 when I made my first trip to the main offices of United Railway Signal Group, Inc. in Jacksonville, Florida. I was impressed that their computer network worked at all, much less that they actually got quite a bit of work done using it. United had the longest single run of 10Base-2 I had ever seen, connecting approximately 30 computers to two Novell Netware servers that lived in a halon fire-protected room at one end of the offices.

The primary server was Netware 3.12 on an Intel P90 with 64MB of RAM, three Adaptec 2940 SCSI controllers, two one gigabyte SCSI hard disks and a 20 cartridge, 26GB Maxoptix magneto-optical jukebox. One of the 1GB disks was exported from both IPX/SPX and Netware NFS as “ursgpub”, a shared network file system containing shared data and a collection of custom FoxPro applications to track such information as project flow and time sheets.

The magneto-optical jukebox was being used at about 80% capacity. Corel SCSI for Netware controlled the jukebox and used one of the 1GB disks as its “cache volume”. A DOS TSR was needed on the client side for the PCs to see the jukebox as a contiguous 26GB file system. United's Unix CAD stations needed access to data on the jukebox, as did the PC CAD stations and office PCs.

Netware NFS had no knowledge of the jukebox, as the jukebox was handled entirely by the Corel SCSI NLMs. In order to allow the Unix CAD stations some limited access to that data, the jukebox's "cache volume" had to be NFS exported. NFS exporting of the cache volume from underneath Corel SCSI proved to be the source of many problems and a constant headache.

The other server, an Intel P75 with 32MB of RAM, ran Netware 4.1. Its sole responsibility was to do backups. ArcServe, the package United used for backups, could never be made to run on the Netware 3.12 server. Thus, United's only option was to purchase a 60-user license Netware 4.1 server for ArcServe and to install the WinAgent client software on each PC. ArcServe never seemed to run properly on this backup server either, crashing almost nightly with out of memory errors or hanging when a client PC's WinAgent software hung.

In mid-October of 1995 we, Progressive Computer Concepts, connected United's main office to the Internet via a dedicated ISDN line using an Ascend Pipeline 50. We installed Linux 1.2.13 on an Intel P90 with 32MB of RAM, a BusLogic 956C and a 1GB SCSI disk to handle DNS, e-mail, WWW and FTP service.

From my time spent at United performing this work, it became obvious to me that the Netware solution was falling apart. The Netware server aborted about once per week (and checking the jukebox's file systems on reboot took hours). Nightly backups via ArcServe failed in some manner almost daily, and hours were wasted each day by United's Netware administrator manually reviewing logs and checking the contents of tapes. I began expressing my opinion that Linux would be a better solution to United's CEO, Mike Wilson.

Over time, the reliability of the one Linux server that handled the Internet operations became more and more clear. I would e-mail Mike an uptime report every 30 days or so. The pivotal point in moving United away from Netware to Linux came when United's Netware system administrator resigned. The door was now opened for us to step in.

Where We Went

In June 1996 we began the formidable task of moving United's entire operation from Netware to Linux. The first to go was the backup server—a Linux boot disk and about an hour turned that little P75 into United's first Linux file server named ursgfs2. We immediately installed SAMBA and **smbfs** and began writing backup scripts. After removing WinAgent and setting up an administrative share on each PC, we did a full floor backup to the 4mm tape drives in the new Linux server that very night; the backups completed through every machine without error.

We purchased three BusLogic 958 SCSI controllers and four 4GB fast wide SCSI II Quantum Atlas drives in external cases and attached them to the P75 Linux server, ursgfs2. Now, we needed a way to move many gigabytes of CAD files and corporate data off the 26GB magneto-optical jukebox connected to the Netware server, at that time accessible only through a DOS machine using Netware drivers plus a TSR program. We tried various unsuccessful methods.

All of our attempts on DOS clients failed with out of memory errors while trying to **pkzip** or **xcopy** files from the jukebox cartridges. The Corel TSR would load under NT but would crash and die at random points during the copy process. We never got NT to successfully copy a single cartridge. Using **ncpmount** we could mount the jukebox from the Netware server under Linux, but without the TSR the Netware server would kick us off within 60 seconds. The solution was DOSEMU. DOSEMU, when installed on ursgfs2 allowed us to run the Corel TSR, attach to the jukebox on the Netware server and then copy directly to the attached 16GB of new disk space using the xcopy command.

Due to an inefficiency in the FAT (finessed automatic transfer) file system, the FAT tables on the jukebox cartridges were filling long before the cartridges were actually full of data. We were able to store all of the jukebox data onto 16GB of disk space. After the transfer of all of the jukebox data was complete, we blew Netware off the larger server, moved ursgfs2's SCSI cards and disks to the new hardware, and renamed the server ursgfs1. A fifth disk was added a short time later.

The new Linux file server used SAMBA for exporting to PCs and NFS for Unix workstations. The server had three 4mm tape drives. Our backup scripts used **smbmount** to mount each PC in the building and archive it on tape using **tar**. Soon, an eight-port Lantronix 10/100 switch was installed, and ursgfs1 was moved to a dedicated 100MB port.

United has a number of CAD stations in each of its offices, where CAD operators work each day on various engineering projects. While using the Netware/ArcServe system, each night ArcServe would copy CAD files from a specific directory hierarchy on each CAD station, in-turn, to the read-only CAD file hierarchy on the Netware server. The potential existed for two different projects inside United to involve the same CAD files. This situation is particularly dangerous when the two CAD operators involved are unaware that they are both working on the same set of files. Under the old Netware/ArcServe system the fact that two CAD operators had been working on the same CAD files, simultaneously, could only be detected by a human and was often not discovered for days or weeks. Much redo CAD work (recadding) would have to be done when those situations were discovered.

To solve this problem, our next software project for United was to write a custom file retrieval and archive commit program that would be, in effect, a revision control system. Every night, the working directories of each CAD station are copied by ursgfs1 to scratch space. Files are then put through a number of sanity checks to detect duplicate works-in-progress, verify file revision and time stamps, file sizes, etc. Files passing all criteria are copied into the read-only CAD file hierarchy; currently, existing files replaced by this process are put into a daily incremental backup. Sixty days' worth of these incrementals are kept in mid-line storage, and any version of any file can be rolled back if needed. Summary reports of committed files and rejected files (if any) are e-mailed to the administrators and to a hypermail archive each night. This system is written entirely in Perl 5 and has been in place, working successfully, since October 25, 1996. The system has recently been expanded to include United's Omaha office.

After the back-up work was completed, we began developing custom Intranet applications for United. We replaced most legacy FoxPro LAN applications with more fully featured and more tightly integrated Intranet programs. The Intranet system, the URSG Daily Operations Control System as it is called, is written entirely in Perl 5. URSGDOCS originally used MiniSQL as its back-end database, but has been ported to and using MySQL for many months now. Once all of the legacy applications like time-sheet entry and project management had been replaced by the Intranet system, we upgraded United's main office Internet connection to 1.536MB T1. United's remote offices in San Francisco, Omaha and Jacksonville (the manufacturing facility) immediately began using URSGDOCS via dial-up Internet connections.

Figure 1. Internet Screen for URSGDOCS

Later, we added a “Fax This Page” button to the bottom of all the reports that a user might wish to retrieve from United's Daily Operations Control System. A retired 386DX40 was given 32MB of RAM and an eight-port Comtrol Rocketport board. It now runs multiple PPP dial-in sessions, various network sniffers and all of the URSGDOCS faxing subsystem. The URSGDOCS faxing subsystem is a custom Perl script wrapped around the **efax07a** package, a virtual X server, a few Netscape “-remote” commands and Ghostscript. The result is the ability to fax any URSG Daily Operations Control System report directly to any fax machine in the world, just by clicking on that button.

United's Omaha office grew to the point that dedicated T1 connectivity was deemed necessary. A Linux server was installed in that office in December 1996. The custom file retrieval and archive commit program now runs in Omaha as well. Furthermore, **ssh** (Secure Shell) is used to move those files automatically to the main server in Jacksonville.

We ran across a great deal on some DEC Alpha UDBs (universal desktop box) and initially picked up four of them for United. Red Hat Alpha Linux allowed us to spread some of the server tasks across those boxes. The URSG Daily Operations Control System was moved to alpha2, for example. Alpha4 was assigned the role of running the old 26GB magneto-optical jukebox which had been collecting dust for a few months. By wrapping our own custom backup scripts around Gerd Knorr's jukebox disk-changer package, we have almost eliminated the need for United to perform tape-based backups. Better yet, nightly incrementals of everything imaginable happen automatically to mid-line storage. Detailed reports of what was backed up, what incrementals were pruned due to age, and the disk-usage status of everything are waiting for the system administrators each morning.

Linux On The Desktop

As United continues to grow and the employee count rises, additional desktop computers are continuously needed. Having worked at places like the Supercomputer Computations Research Institute on the campus of Florida State University, where four system administrators support hundreds of Unix users (a large percentage through X Terminals), we knew that an X Terminal model would work for United. We began installing Linux-based X Terminals everywhere new desktop computers were needed. We are able to sell United new Alphas with 32MB of RAM, 15 inch monitors and no hard disks for $1000US each. This is a much cheaper alternative to deploying new Windows 95 boxes considering the cost of hardware, software, setup time and recurring maintenance. The hardware was also cheaper, and in our opinion better than even the dedicated X Terminal/Network Computer equipment that we originally investigated for this task.

The Linux/Alpha X Terminals boot over the network, NFS mount their root and usr file systems, and then open an **xdm** (X window manager) session on an XDMCP server. The window manager chosen for United's X Terminals was FVWM95. The engineers who have received these X Terminals rarely use MS Office type applications—Applixware is used to fill that occasional need. Netscape Navigator is used to access the URSGDOCS and Internet e-mail.

Figure 2. ApplixWord on X Terminal

The X Terminal model is working well. All of URSG's engineers have migrated to X Terminals. This has allowed United to continue to extend the useful lifespan of aging 386/486 class equipment by redeploying it in both X terminal and fat-client capacities in the wiring facilites and branch offices.

Our calculations show significant cost savings through utilizing slightly higher-end Linux-based X Terminals with fast Ethernet in place of the FAT- client CAD

stations that United currently uses and deploys. PCC was a beta test site for Bentley's port of Microstation 95 to Linux. The port seemed flawless and was very promising; an academic release version has been shipping for several months. However, Bentley has not yet received the level of demand from Linux users that it deems necessary to support Microstation commercially on the Linux platform. We plan to continue to work with Bentley and to encourage the commercial release and support of Microstation for Linux.

More Recent Projects

An NCD Wincenter Multiuser NT server has been installed at URSG and more Alpha X Terminals deployed. Even administrative staff now have Linux-based X terminals on their desk. These staffers use Linux Netscape for Inter/Intranet access and e-mail, and they run office productivity applications (accounting, MS Word, MS Excel, etc.) on the Multiuser NT server.

URSG's Jacksonville wiring facility has moved to a new location, tripled in size, and had its connectivity to the main-office LAN upgraded from the 128K ISDN mentioned earlier to a 1.536MB T1.

PCC has added enhanced extranet functionality to URSGDOCS including a system designed specifically for use by CSX Transportation and all of its railway signal design contractors. This particular extranet section of URSGDOCS allows fast and well documented business transactions between CSX Transportation and its signal design contractors. The system allows contractors to make requests for circuit plans and CAD files, CSXT staffers fulfill those requests by uploading CAD files into URSGDOCS, then contractors can download those files. All transactions inside the system generate e-mail notifications to appropriate persons and to an e-mail alias that is used to archive all transactions. The transactions archive is full-text searchable and browseable. The system has been in operation since July 10, 1997 and averages approximately 2000 transactions and 500MB of compressed CAD file transfers per month.

As new systems and functionality have been added to URSGDOCS, it has been migrated to a Dual 200MHz Pentium Pro Linux server with 128MB RAM. The ursgfS1 machine has been upgraded to a uni-processor 200MHz Pentium Pro Linux server with 64MB RAM. The Wincenter server resides on Dual 150MHz Pentium Pro hardware with 128MB RAM.

The Moral of the Story

Unix, in particular Linux, combined with the GNU tools, can change a company's technology spending focus allowing it to move away from a larger, under-skilled IS staff to a smaller, higher-skilled staff or to outsourcing. A move

to either results in a company receiving more reliable and more customized solutions that can easily evolve to the company's changing needs over time.

Yes, Linux means business. The work that we at Progressive Computer Concepts have done with United Railway Signal Group is a wonderful example.

Lester Hightower (hightowe@progressive-comp.com) is Vice President of Development at Progressive Computer Concepts, Inc. in Jacksonville, Florida. He has an extensive background with a wide variety of Unices. He is involved in all aspects of Unix and Internet consulting including installation, administration and systems programming.

Hank Leininger (hlein@progressive-comp.com) is Senior Systems Integrator at Progressive Computer Concepts, Inc. in Jacksonville, Florida. He has an extensive background with a wide variety of Unices. He is involved in all aspects of Unix and Internet consulting including installation, administration and systems programming.

Archive Index Issue Table of Contents

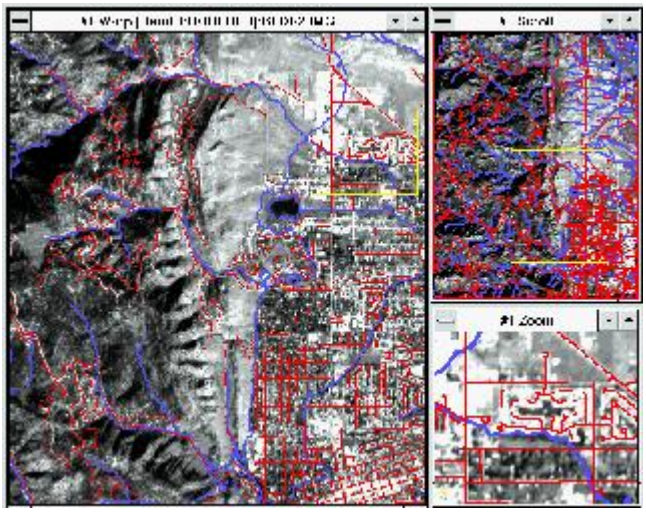
Advanced search

New Products

Amy Kukuk

Issue #46, February 1998

ENVI Vector GIS Feature, Xbase for Linux, Empress Hypermedia V2.10 and more.



ENVI Vector GIS Feature

ENVI 3.0

Research Systems Incorporated announced the release of ENVI (Environment for Visualizing Images) 3.0. Among the new features of this release are GIS tools and new routines for orthorectification of air photos and satellite images. Other features include support for the Spatial Data Transfer Standard (SDTS), support for U.S. and world location maps and true type scalable fonts. Personal computer pricing starts at $3350 US and workstation pricing starts at $5750 US.

Contact: Research Systems, Inc., 2995 Wilderness Place, Boulder, CO 80301,Phone: 303-786-9900, Fax: 303-786-9900, E-mail: info@rsinc.com, URL: http://www.rsinc.com/.

Xbase for Linux

StarTech has introduced a DBase-compatible C/C++ library for reading, writing and updating Xbase format database files and indeces. Xbase for Linux is available for review or download for free from their website.

Contact: StarTech, 1615 Timber Ridge Lane, Roanoke, TX 76262,Phone: 817-431-8546, E-mail: startech@gbob.com, URL: http://www.gbob.com/.

Empress Hypermedia V2.10

Empress announced the release of Empress Hypermedia V2.10. The new release features new tools for creating database-driven web pages. Enhancements to this Internet applications toolkit include the ability to establish a continuous session to the Empress RDBMS, simple syntax for switching database tools, the ability to utilize multiple languages in addition to English, an easier method for establishing joins between two data tables, the option to use in-line CGIs and the ability to insert data into a database from a file on the client.

Contact: Empress Software, 6401 Golden Triangle Drive, Greenbelt, MD 20770, Phone: 301-220-1919, E-mail: sales@empress.com, URL: http:// www.empress.com/.

SCREADER

SCREADER is a freely available screen reader using a software text-to-speech tool. No extra hardware is required, but you do need a sound card and a text to speech (TTS) synthesizer. It is a modification of the **Screen** package and most of the features of Screen are still available. Future versions of SCREADER will not be implemented in Screen. At that time it will read from each virtual screen. This version only works on the virtual console where SCREADER is executed.

Contact: Jos Lemmens, URL: http://leb.net/blinux/betas.html#scr.

ascp0.8.2

The release of ascp0.8.2 (AfterStep Control Panel) has been announced. New features include finished background utility, file browsing, JPEG support, root modes for backgrounds and added context sensitive help. ascp0.8.2 is intended to provide an intuitive graphical front-end for configuring AfterStep and is freely available.

Contact: Elumaze Nwanua, URL: http://hubble.colorado.edu/~nwanua/htmldir/ascp.html.

FontScope

CurveSoft announced the release of FontScope. Features of FontScope include Multiple Master font support, well-defined API, two sample C++ applications to illustrate the use of the API and the ability to be statically linked into an application or installed as a dynamically linked shared library. A free demo version can be downloaded at ftp://ftp/blueneptune.com/pub/users/ram/demos/demo-x86.tar.gz.

Contact: CurveSoft, Inc., 2053 Grant Road, Suite 555, Los Altos, CA 94024,E-mail: info@curvesoft.com, URL: http://www.curvesoft.com/.

Pure Java JDBC Driver

Solid Information Technology Ltd. announced the 100% Pure Java JDBC driver for the SOLID Server database management system. The new native driver offers Java applications access to Solid's database engine. SOLID Server is a standards-compliant and compact database engine requiring minimal administrator attention. Its database components are embedded in packaged software, application development tools, web sites, hand-held devices, point of sale systems, industrial products and more.

Contact: Solid Information Technology Ltd., Huovitie 3, FIN-00400 Helsinki, Finland, Phone: +358-9-477 4730, Fax: +358-9-477 473 90, E-mail: info@solidtech.com, URL: http://www.solidtech.com/.

QMASTER Batch Management

QMASTER Software Solutions Incorporated announced the release of its batch management system which can manage occasional batch processing on a single server up to high-end scheduling. Some features include interval submission of jobs, automatic re-submissions, global variables for interprocess communication, job grouping and more.

Contact: QMASTER Software Solutions Inc., 1730-840, 7th Avenue SW, Calgary, Alberta T2P 3G2, Canada, Phone: 403-264-8322, Fax: 403-265-5307, E-mail: info@qmaster.com, URL: http://www.qmaster.com/.

QMASTER Output Management

QMASTER Software Solutions Incorporated announced the release of its output management system which manages output from file to device to messaging. The software allows central management of departmental printing. Some features of the new software include partial printing, routing to multiple

printers, reprinting of “lost” printouts, transparent pre-processing and load-balancing over printers.

Contact: QMASTER Software Solutions Inc., 1730-840 7th Avenue SW, Calgary, Alberta T2P 3G2, Canada, Phone: 403-264-8322, Fax: 403-265-5307, E-mail: info@qmaster.com, URL: http://www.qmaster.com/.

Babylon

SpellCaster Telecommunications Inc. announced Babylon. Babylon is a Multilink PPP solution that provides support for both modems and ISDN adapters in a single package. Some features include multi-link and authentication on a workstation or server, and telecommunication applications at 64Kbps, ISDN and higher bandwidths. Babylon is device independent and supports both in-bound and out-bound connections. Babylon is available from SpellCaster Telecommunications at a list price of $199US for the 256Kbps version. CD-ROM copies of Babylon also include the Red Hat Linux 4.2 operating system.

Contact: SpellCaster Telecommunications Inc., 73 Laird Drive, Suite 206, Toronto, Ontario M4G 3T4, Canada, Phone: 800-238-0547, Fax: 416-425-0854, E-mail: info@spellcast.com, URL: http://www.spellcast.com/.

Corel Video Network Computer

Corel Computer Corporation announced its new Video Network Computer. It contains a Linux-based operating system and a Java-based multimedia communications suite. Key benefits of this new technology include a lower cost, persistent caching and synchronization, minimal network traffic, energy efficiency and ergonomic design. Pricing for corporate solutions is available upon request.

Contact: Corel Computer Corporation, 150 Isabella Street, Suite 1000, Ottawa,Ontario K1S 1V7, Canada, Phone: 613-788-6000, Fax: 613-230-8300,E-mail: sales@corelcomputer.com, URL: http://www.corelcomputer.com/.

W3Control Filter

W3Control announced the W3Control Filter. W3Control is a web request filtration package designed to meet the demands of commercial accounts. The product is a server-based filtration solution which blocks access to Internet sites drawn from a list of 350,000 sites deemed to be either unproductive from a business perspective or a potential source of legal liability. The software also keeps an active log of all Internet requests down to the group or individual user level.

Contact: W3Control Inc., 260 Chapman Road, Suite 208, Newark, DE 19702,Phone: 302-451-1698, Fax: 302-454-1718, E-mail: sales@w3control.com,URL: http://www.w3control.com/.

Samba: Integrating Unix and Windows

Specialized Systems Consultants, Inc, announed the publication of “Samba: Intagrating Unix and Windows” by John D. Blair. The book is a combination of technical tutorial, reference guide and how-to manual. It contains a CD-ROM of 1.9.17 and 1.9.18alpha versions of the Samba server, a library of tools and scripts and the Samba mailing list archives. The price of the book is $29.95US.

Contact: SSC, Inc., P.O. Box 55549, Seattle, WA 98155-0549, Phone: 206-782-7733, Fax: 206-782-7191, E-mail: info@linuxjournal.com, URL: http://www.ssc.com/.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #46, February 1998

Our experts answer your technical questions.

Using Two Monitors in Linux

Is it possible to set up Linux to use two monitors? If so, where can I find out about it? —Greg Bell

It is possible to use multiple monitors but to do so you will need an X Server that supports it. Accelerated X from Xi Graphics and Metro-X from Metro Link both support this feature (known as multi-headed). Based on comments from friends I would go with Accelerated-X for performance and stability reasons. To find more information, visit their web sites :Xi Graphics: http://www.xig.com/ Metrolink: http://www.metrolink.com/ —Mario de Mello B. Neto mneto@buriti.com.br

FTP No Longer Works

My machine no longer accepts logins via FTP (anonymous or login). It accepts the connections but no user name is good enough for it. It worked fine until I moved my /usr directories onto their own file system. The next day, people said they couldn't log into my FTP server. —Anonymous

From your description, I can't tell what the problem is for certain, but I suspect it is incorrect /etc/passwd entries. To debug this problem, run **strace** on the process generating the error, in this case **ftpd**. You'll find which operation causes the connection refusal by attaching strace to the process before the error occurs, that is, before typing your user name. —Alessandro Rubini alessandro.rubini@pluto.linux.it

Corrupted Terminal

When I **cat** a binary file (**cat xxxx.tgz**) to a virtual console (for example tty1) all of the characters are changed on the screen. For example the “a” character become a big rectangle. I can continue to type commands but when I use the mouse to cut and paste in another tty the words are stripped of their vowels. How can I solve this problem without a reboot? —Thierry Neusius Slackware 3.2

The cat program was designed to handle ASCII files. When used on a binary file, it attempts to display it just like any ASCII file. Unfortunately, this usually results in the terminal getting corrupted because cat interprets a lot of the binary file as control sequences. These control sequences almost always result in a lot of unprintable characters and a messed up terminal. Logging out usually solves the problem of a corrupted terminal. —Keith Stevenson ktstev01@homer.louisville.edu

You can run the program **reset** to switch back to normal characters. (**reset** is part of the **ncurses** distribution and should already be on your system.) —Samuel Ockman, VA Research ockman@varesearch.com

Clustering Linux Servers

I run a small web site hosting business and am wondering if there is any efficient way to cluster two or more Linux servers. I want it so that if one server goes down, all the hits will just go to the active server. I have experimented with round-robin DNS, but that doesn't give me the fault tolerance I want. I'd welcome any suggestions. —Tim ArcherRed Hat 4.1

You can use IP aliasing to solve this problem. Each machine should run a script that watches the status of the other. When a machine stops responding, the working machine adds an IP alias for the down machine's IP number. Now the up machine answers for both itself and the down machine. When the original machine comes back on-line, the alias is removed.

What makes this tricky is that you must make sure that you never have both machines responding to the same IP address at the same time. You can do this in several ways. One way is to have a separate communication path between the two machines. Here, again, you can use IP aliasing. First, assign each machine a primary, unique IP number. No other machine will ever use that number. Next, assign an IP alias to each machine to handle the Web traffic. Your round-robin DNS cycles through the aliases, not the primary machine addresses.

Now, the two machines can always talk to each other using the primary addresses regardless of who is responding to the IP aliases. The machines

monitor each other on the primary addresses and install or remove the aliases depending on each other's state.

You may lose a few hits during the time it takes the first machine to figure out that the other is down and switch over, but most people generally find the number of missed hits acceptable. —Larry Augustin, VA Research lma@varesearch.com

Playing Internet Games Through A Firewall

With the release of more Internet-capable games like Diablo, Netstorm and Quake, what needs to be done so that the IP packets are routed through a firewall?

We use a Linux firewall to connect to our Internet service provider via PPP with dynamic IP numbers. The other members of the household would like to connect to the various servers available for games. I've looked though the documentation and nothing seems to work. I've tried to add services to the /etc/services file, (following up in /etc/inetd.conf, /etc/hosts.allow, /etc/sockd.conf and a few others). Our firewall works by setting firewall on in the kernel and IP forwarding turned off. —Aaron Hicks Slackware 2.0

You need to allow the packets to be forwarded for Internet hosts to see machines on your local LAN. To keep the firewall protection active in your arrangement, find out the port numbers the various games use.

Suppose you wanted to allow only TELNET connections through your firewall. TELNET uses port 23. Using **ipfwadm** you can specify forwarding rules to allow packets destined for port 23 to pass through your firewall. For additional protection, you can specify entries only for those machines that should receive these packets.

Read the ipfwadm(8) man page for more information on command-line syntax for this tool. Note that this applies only to standard Linux firewalling systems (using the kernel facilities, rather than external programs). In this case, the kernel handles all firewall functions, and ipfwadm is only used to configure the rules. —Larry Augustin, VA Research lma@varesearch.com

Compatibility Between Linux Distributions

I have been feeling a bit confused lately about the different Linux distributions. Specifically, if they are compatible with one another. The two distributions I use are Slackware and Red Hat. I prefer the way Slackware handles some things and prefer the Red Hat method for certain other things. What I would like to know is if these two distributions (as well as the others) are interchangeable in

any way. For example, would it be possible to install **pkgtool** on a Red Hat system and could an XF86_SVGA server from Red Hat work on a Slackware distribution? Could a Slackware kernel be used to boot a Red Hat system? — Steven M. St. Hilaire

The core of this question centers around Linux itself. As far as binary compatibility goes, that depends on your platform, not your distribution. If you use x86 binaries from all distributions, then yes, they are indeed compatible.

The problems you are likely to run into are conflicts in directory locations and library versions, which can happen in any binary distribution. The kernels from the various distributions will boot properly, but once Linux boots, it needs to perform some work to get the entire system mounted and operating.

Take two examples from your question. The XF86_SVGA server would most likely operate properly, if you had installed the entire Xfree86 installation from the Slackware set. If, on the other hand, you wanted to use only the X server from the Slackware distribution and use the Red Hat distribution to install the rest of the X system, you would likely run into trouble.

The second example is the Red Hat control panel. Yes, it would have difficulty with those items which you installed from Slackware. But you would not have trouble using it to control packages which you installed from the Red Hat distribution.

If you do this, be aware that you could be in for long nights working out the various incompatibilities. Certain things may work without trouble, especially third-party packages (such as some database systems) that come in specific formats for your convenience. Unless you are an experienced administrator, I would not recommend installing system-level facilities using a mixed distribution.

Instead, why not select one distribution and install the remaining items by compiling the source distribution? It will probably take somewhat less time to do because of the headaches you will avoid, and you will gain a much stronger understanding of how the various programs, system libraries and kernel facilities interact. —Chad Robinson, BRT Technologies chadr@brt.com

Archive Index Issue Table of Contents

Advanced search


[#301 August 2019](#) Feature: The DevOps Issue


[#300 July 2019](#) Feature: The Command Line


[#299 June 2019](#) Feature: From Mac to Linux

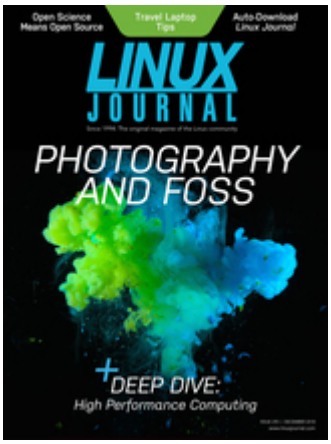

[#298 May 2019](#) Feature: The Kernel Issue


[#297 April 2019](#) Feature: The 25th Anniversary Editio


[#296 March 2019](#) Feature: Single Board Computers


[#295 February 2019](#) Feature: The Security Issue


[#294 January 2019](#) Feature: Distributions (Distro Roundu




[#293 December 2018](#) Feature: Photography and FOSS


[#292 November 2018](#) Feature: Monitoring


[#291 October 2018](#) Feature: Programming


[#290 September 2018](#) Feature: Linux Gaming


[#289 August 2018](#) Feature: Containers


[#288 July 2018](#) Feature: Git


[#287 June 2018](#) Feature: Do It Yourself


[#286 May 2018](#) Feature: Privacy


[#285 April 2018](#) Feature: The Cloud


[#284 March 2018](#) Feature: Blockchain


[#283 November 2017](#) Feature: Control a Heterogeneous Ser SSH Agent


[#282 October 2017](#) Feature: Programming


[#281 September 2017](#) Feature: Full Stack Development Proje


[#280 August 2017](#) Feature: Filesystem Events with inotif


[#279 July 2017](#) Feature: Build your own Cluster


[#278 June 2017](#) Feature: Test Security with an Internal Campaign


[#277 May 2017](#) Feature: 3D Imaging of Heart Activity


[#276 April 2017](#) Feature: The Space Robotics Challeng


[#275 March 2017](#) Feature: Big Data, Hadoop and R


[#274 February 2017](#) Feature: Detect Man-in-the-Middle Ce

Linux Journal

AND Search OR Search
Phrase Search Show results per page

Powered by Sphider