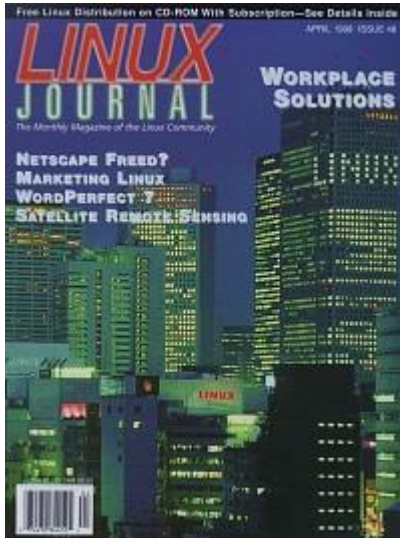


Advanced search

*Linux Journal Issue #48/April 1998*



*Features*

Using Linux in a Control and Robotics Lab by Jon Davis

How a lab at Queen's University is using Linux to develop programs and control hardware experiments.

Biomedical Research and Linux by Roger S. Flugel

Linux is readily establishing itself in the biomedical field as a powerful and reliable system for research computing.

Latvian Government Uses Linux by Dmitrie Komarov

Mr. Komarov tells us how he used Linux to give an old database new capabilities and thereby saved his government money.

Satellite Remote Sensing of the Oceans by Simon J. Keogh, Emmanouil Oikonomou, Daniel Ballestero and Ian Robinson

Presented here is an overview of the kind of remote sensing that is done at Southampton University and how Linux has helped improve our productivity.

Small Business Marketing of Linux by Cliff Seruntine

Linux is a good business product. This article deals with the why, how and who of selling Linux.

*News & Articles*

Building Projects With Imake by Otto Hammersmith

Here's an explanation of how Imake works and how you can use it to build your executables—an article for programmers with C and Unix programming skills.

Linux Network Programming, Part 3: CORBA: The Software Bus by Ivan Griffin, Mark Donnelly and John Nelson

This month we are presented with an introduction to the networking of distributed objects and the use of CORBA.

Financial Calculation Programs for Linux by James Shapiro

Mr. Shapiro shows us how to write a program to compute internal rate of return using three programming languages supported by Linux—Perl, C and Java.

LJ Interviews Mr. Eid Eid of Corel Computer by Marjorie Richardson  
Helping Netscape Make History by Eric S. Raymond

Netscape source is now free, who would have thought it? Eric Raymond, that's who. Here are his insights into this momentous event.

### *Reviews*

**Product Reviews** Visual SlickEdit: A Commercial Editor for Programmers by Larry Ayers

**Product Reviews** WordPerfect 7 for Linux by Michael Scott Shappe

**Product Reviews** TeraSpell 97 for Emacs by Daniel Lazenby

**Book Reviews** Practical Programming in Tcl and Tk by John McLaughlin

**Book Reviews** Protecting Your Web Site with Firewalls by Leam Hall

### *WWWsmith*

**At the Forge** Using What We've Learned by Reuven M. Lerner

This month Mr. Lerner shows us how to set up a web site using many of the techniques he's taught us over the past months.

### *Columns*

Letters to the Editor

From the Editor Workplace Solutions by Marjorie Richardson

Stop the Presses The Software world—It's a Changin' by Phil Hughes

New Products

**System Administration** Managing your Logs with Chklogs by Emilio Grimaldo

Managing your Logs with Chklogs An introduction to a program written by Mr. Grimaldo to manage system logs.

**Kernel Korner** Writing a Linux Driver by Fernando Matia

The main goal of this article is to learn what a driver is, how to implement a driver for Linux and how to integrate it into the operating system. An article for the experienced C programmer.

Linux Gazette Configuring procmail with The Dotfile Generator by Jesper Pedersen

Best of Technical Support

Archive Index

Advanced search

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

## Using Linux in a Control and Robotics Lab

**Jon Davis**

Issue #48, April 1998

How a lab at Queen's University is using Linux to develop programs and control hardware experiments.

The Mathematics and Engineering program at Queen's University in Kingston, Ontario operates a control and robotics laboratory as part of the course offerings in the control systems area. The lab experiments use our custom-built electro-mechanical setups and require the students to write algorithms in C for controlling the hardware.

The lab began using C under DOS as the software environment for the lab experiments, but not all students had an easy time with the environment. Generally, it was too easy for configuration files to be inadvertently changed with frustrating consequences. Subsequently, an integrated experiment environment known as **dlxlab** has been developed for simulating and running control lab experiments. It consists of a single program, run as **dlxsim** for simulations or as **dlxrun** for controlling hardware experiments. The program was developed using the XView toolkit under Linux (see "Programming with XView" by Michael Hall, *LJ*, March 1998), and operates in the lab on a variety of PC-compatible hardware running Linux 1.2.13.

### **dlxlab Design Goals**

The students taking the lab vary widely in computing background and skill. The primary intent of the control labs is to investigate the application of control theory to actual motors, carts, inverted pendula and so on, without having software operation dominate the experience. On the other hand, understanding low-level interfacing code is also a desirable outcome, so the hardware interface has to be "visible".

In order to design a control algorithm for a physical system, one must have a mathematical model of the system to be controlled in the form of either

differential or difference equations, and knowledge of the physical parameters in the model. The user interface to dlxlab was designed so that, as far as possible, this is the *only* information that must be supplied by the user.

This goal is attained for the case of the program running in simulation mode. For the situation where actual hardware is being controlled, information describing the hardware interface must also be provided, although it can be largely hidden from the user through header files.

The user input to the program takes place through interactive construction of a system file which describes the system under investigation.

### System Files

To simulate a system, one invokes dlxsim with a system description file as argument.

```
dlxsim sim.sys &
```

One of the lab experiments consists of a pair of track-mounted carts, coupled by springs and driven by a servomotor. A simple system file for simulating such a pair of spring-coupled carts is shown in [Listing 1](#). The format of the system file is a sequence of **begin ... end** delimited blocks. The blocks are of two types:

1. Definition blocks establishing identifiers for variables (including parameters)
2. Code blocks containing C code sequences which are executed by the program to initialize variables, as well as numerically integrate the governing differential equations to simulate the system

## Executing A Simulation

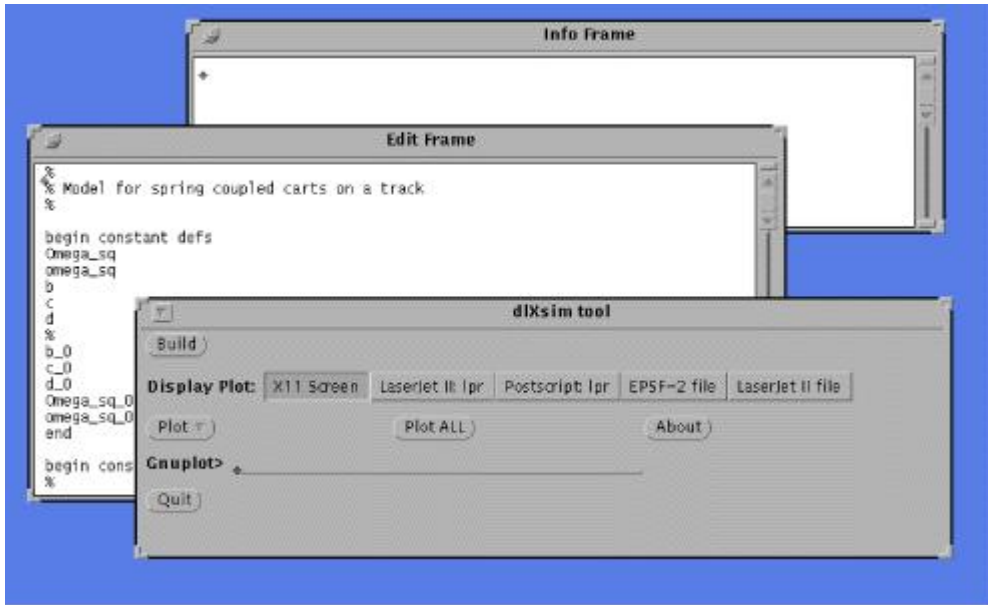


Figure 1. Main Program Panel and Other Windows

The main program panel (see Figure 1) contains a “Build” button, which when pressed causes processing of the system file. That is, the user system file is converted into a series of C code files by a parsing process. The files are compiled to a shared object file by **gcc**, and the contents of the resulting shared object module are dynamically linked into dlXsim as it runs. The linked code contains not just the system differential equations, but also modules for interactively manipulating parameters and plotting results on the basis of the variable names provided in the system file.

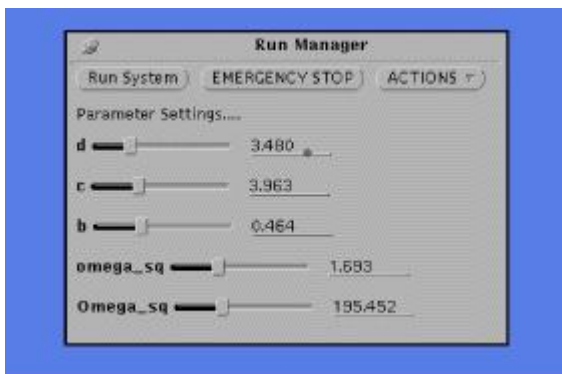


Figure 2. Run Manager

Assuming that the system file contains no syntactic errors, the program log window contains only progress messages, and a pop-up panel for controlling simulations appears (Figure 2). As long as only parameter changes are made, a series of simulation runs can be made. Plotting and printing is handled by **gnuplot** running as a child process.

If the system file contains errors, the error location is reported in the log window, and the pop-up does not appear. The errors are caught either at the parsing level or within the C code segments. In the latter case, the error messages from gcc refer to lines in the user system file, since the generated C files include **#line** statements referring to the user system file. Since the dlxsim system file edit window is an XView **textsw**, it inherits the line searching menus associated with XView applications.

Simulations can be run and plotted, as long as the system file contents (such as variable names, equations of motion and so on) are not changed. If such changes are made, the “ReBuild” button must be invoked to cause freeing of resources, recompilation of the dynamic module and relinking of the generated codes.

### Controlling Actual Hardware

The system file for an actual experiment contains code sections which mediate between floating-point program variables and the binary hardware levels, in addition to the requirements of a simulation file. A file for measuring the response of a servomotor to a constant voltage input is shown in [Listing 2](#). This file illustrates the use of so-called utility code and definition sections in the system file. There is really no restriction on the type of code placed here. The program user guide contains an example of such code written to carry out a recursive least squares identification algorithm using measured data resident in files. The example in Listing 2 is much more modest and uses library include files to abstract the interface board data access.

The include files serve to hide the actual hardware interface behind port access macros like **set\_dac()** for setting digital-to-analog converter levels and **get\_encoder()** for reading the count of the optical quadrature position encoder from the interface boards. The code blocks using these macros are converted to dynamically linked subroutines and repeatedly called by the program **main** real-time control loop.

This example is really “open loop control” and primarily illustrates the hardware interface provided by the program. Feedback controllers typically employ filtering of the measured data, and the system file for such a controller includes a system code section which implements the dynamics of the filters.

### I Thought Real-time Linux Wasn't Running?

True, certainly we would be in trouble trying to run a print server and a copy of the Apache WWW daemon at the same time that we were balancing an inverted double pendulum. However, in the lab environment, the window manager (preferably Open Look olwmm) and the control environment dlxrun

are the only user level applications running. The program dlxlab is an XView application, written in the explicit dispatch mode. This means the timing of the control loop is under control of the programmed loop and not the XView notifier.

As long as the lab machines are provided with enough memory to avoid swapping during the experiments, the effect of timing jitter has a smaller effect than, for example, pretending that the behavior of servomotors is entirely linear. It was originally thought that selectively killing and restarting certain daemon processes would be necessary, but our experience has shown that this is not the case. In any event, one of the aims of control design is to produce controllers which are robust against unmodelled disturbances, and timing jitter provides an example of such a disturbance.

### Equipment Description

We run lab experiments on machines ranging from a 12MB 486SLC-66 to a Pentium P5-166. There is no problem running the experiments on our set of 486DX2-66 boxes, with sample rates up to several thousand samples per second. The lab machines are on a local network with 10Base-2 coaxial cable, with a salvaged 386DX-16 staggering along as the resident print server.

### Further Information

The Linux machine in my office is running the Apache web server and has a WWW page for our control and robotics lab. The address is <http://jhd486.mast.queensu.ca/>. The lab page has photos of the lab equipment and links to my home page where documentation, sources and binaries for the dlxlab programs are available.

### Conclusion

The dlxlab environment described began life as an **awk** script that turned a system file (ancestor of the ones above) into an XView control system simulation program, running under SunOS-4.1. After a Linux conversion experience, I came upon a version of the *Kernel Hacker's Guide* by Michael Johnson and discovered that user level I/O port access was possible under Linux. This allowed the program to accomplish hardware control as well as simulation tasks.

The low cost and wide availability of interface boards for PC-compatible machines make them ideal for a lab such as the one we have set up. The complete openness of the Linux system made it possible to undertake program development with the confidence that it could be made to work. It is also



helpful to have the same operating environment in the office, at home and in the robotics and control lab.

Virtual beers all around.



**Jon Davis** is an Associate Professor in the Department of Mathematics and Statistics, Queen's University, Kingston, Ontario. His interests are in applied mathematics, especially control and communication systems. His interest in computing goes back to high school, where he cannibalized some pinball machines to make a science project computing device. He can be reached via e-mail at [jon@mast.queensu.ca](mailto:jon@mast.queensu.ca).

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

[Advanced search](#)

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

## **Biomedical Research and Linux**

**Roger S. Flugel**

Issue #48, April 1998

Linux is readily establishing itself in the biomedical field as a powerful and reliable system for research computing.

The boom in biomedical research over the past decade has occurred as a result of increased demands for better health care and greater understanding of the biomolecular mechanisms responsible for many diseases. Fueling many of the scientific advances in medicine have been the corresponding developments in computational processing power and speed. In addition to the large-scale data collection and processing required to understand complex biological and physiological processes, our ability to analyze and model medical phenomena at microscopic and macroscopic levels is reliant upon the systems used for these computationally intensive jobs. Linux systems are receiving recognition as ideal workhorses for performing many of these tasks.

To better understand the emerging biomedical applications of Linux, it is first necessary to examine the typical computational requirements and resources of many biomedical research centers. The popular perception of biology and medicine is that they are wet and gushy sciences, involving little or no quantitative or numerical analysis. In fact, this couldn't be farther from the truth. There is a concerted effort among scientists to bring the biomedical sciences into the realm of other quantitative disciplines. This effort is illustrated by the recent emergence or increasing prominence of fields such as bioinformatics and genomic-sequence databases, biomolecular modeling and computer-aided drug design, diagnostic medical imaging, epidemiology and biostatistical analyses of patient populations, as well as medical informatics and patient record databases.

The systems used by researchers in these fields span a spectrum of different platforms. One typically finds an abundant assortment of Intel and Power PC-based systems running the Windows and Mac operating systems for performing basic tasks such as text processing, routine data analysis, e-mail

and Internet access. In addition, the recent trend among manufacturers has been to interface a broad range of biomedical research equipment with PCs for the ease of GUI instrument control and data collection. The choice of which platform should be employed for such interfaces is usually the decision of the equipment manufacturer and not the researcher.

Laboratories may also contain a few Unix workstations which are used to process large jobs, drive complex instrumentation or function as servers. Typically these machines are dedicated to specific tasks and are unavailable for general use.

One unfortunate aspect of this heterogeneous situation is that it creates headaches for system administrators. Such a wide variety of systems performing a multitude of different tasks and operated by a large number of users with a wide range of computer abilities, frequently spells disaster.

Linux serves as an ideal system for biomedical research laboratories, where there is an abundance of PCs, strict limits on the allocation of research funds and a need for significant processing power in conjunction with straightforward and flexible system customization. The stability of Linux, its cost-effectiveness and large user community make it attractive to both novice and experienced system administrators confined by tight research computing budgets. In addition, the ability of the various incarnations of Linux to run on a wide variety of platforms provides a uniform OS and networking capabilities for the mixture of different machines already in existence at many biomedical research centers.

Utilization of Linux systems is emerging within the Biomolecular Structure Group here at Harvard Medical School. Intel-based systems running Red Hat Linux have recently been implemented for simultaneous control and data collection from x-ray diffraction instruments (Figure 1). Individual data sets obtained with these instruments are about 100MB in size and are used to understand the atomic structure of biomolecules and for drug-design studies at the molecular level. Although the source code for driving and collecting data on the x-ray instrument had to be recompiled for Linux and the kernel was rebuilt, the result was a very stable system for performing this specialized and complex task. In this instance, the Linux system has successfully replaced a more expensive Silicon Graphics workstation.

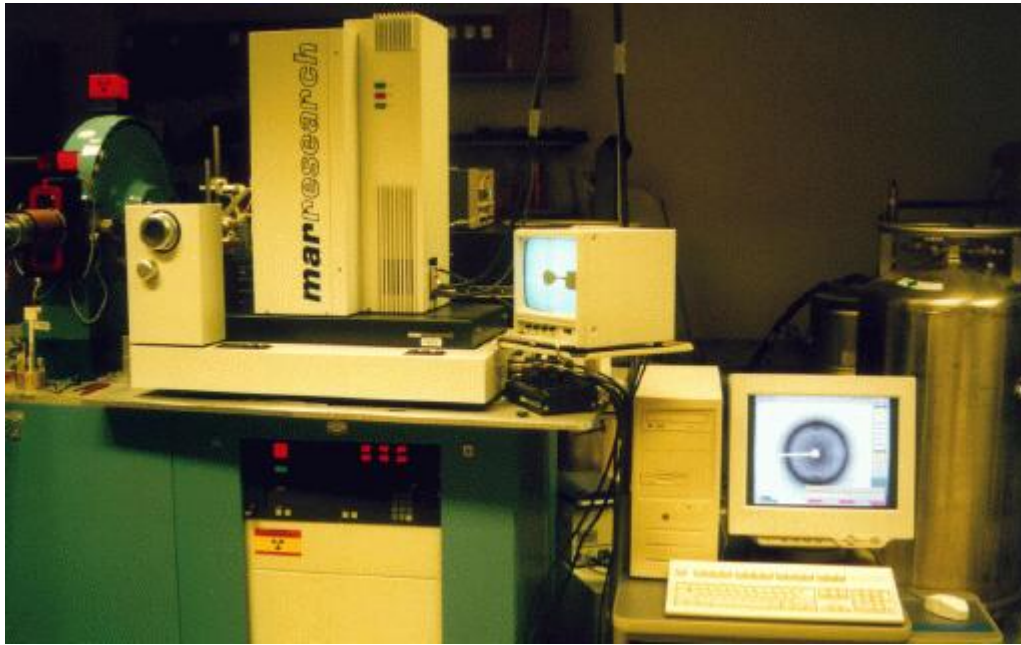


Figure 1. X-Ray Diffraction Equipment

Granted, the above application of Linux is rather specific, but Linux is also gaining more widespread usage among individual researchers. Installation of Linux on PCs in conjunction with either the Windows or Mac operating systems gives users the choice of the appropriate system with which to complete a job, while sitting at the same machine. For example, in a laboratory with an abundance of Macintoshes, the recently developed MkLinux for Power PC has proven to be beneficial in providing greater access to remote Unix systems and applications. More individuals are now able to use popular GUIs for genomic databases, bioinformatics applications, and biomolecular visualization software located on remote servers, while taking advantage of the multitasking capabilities of Linux (Figure 2). In addition, through implementation of a simple Tcl/Tk script, general users have the ability to reboot the machine back into Mac OS with a click of a button. In some cases, individuals are using the Linux systems as nothing more than X terminals. However, this use alone is valuable, since it provides increased access to applications located on a limited number of servers without having to purchase additional X terminals or X terminal emulation software.

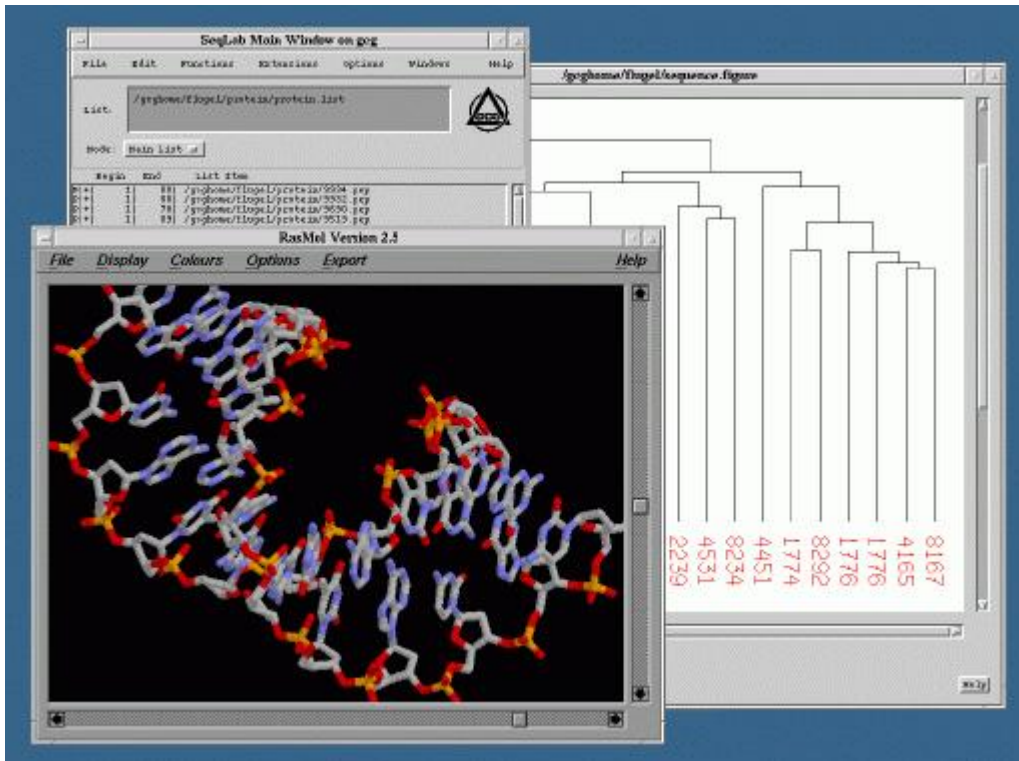


Figure 2. Multitasking with Linux

Linux is bound to have a more prominent role in biomedical research computing in the future. The recent installations of Linux systems described above have already been recognized as highly successful and even superior to other systems, in terms of both price and performance. As the computing demands of biomedical research increase, scientists, system administrators, software developers and instrument manufacturers will all be considering Linux.



**Roger S. Flugel** is a Ph.D. student in Molecular Biophysics at Harvard Medical School. He spends far too much time as a system administrator and hacking on MkLinux, and far too little time actually working towards finishing his degree. He can be reached via e-mail at [flugel@walsh.med.harvard.edu](mailto:flugel@walsh.med.harvard.edu).

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

[Advanced search](#)

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

[Advanced search](#)

## Latvian Government Uses Linux

**Dmitrie Komarov**

Issue #48, April 1998

Mr. Komarov tells us how he used Linux to give an old database new capabilities and thereby saved his government money.

Recently I've found a new unusual use for Linux, which I have used successfully at work. Here is a list of machines and software I had to work with:

- Local Ethernet network
- Novell NetWare 4.10 file server
- About 15 DOS/Windows workstations
- Linux box as an IP router, mail server, etc.
- A database application written in Clipper for MS-DOS, which contains more than 400,000 records

### The Problem

This database application was written for a stand-alone DOS workstation about five years ago, and the most powerful computer we had at that time was a 386SX with 2MB of RAM. At one point we patched this program to add a little network support. The main database migrated to the NetWare file server, and the program worked with the server as if it were a local drive (Z:). Data is added to this database every day, and it has grown to more than 300MB in size. There are now 15 users connected to it, but the program itself remains the same as it was five years ago.

Each time a user searches the database, his workstation uses about 30 to 100MB of network traffic. The NetWare server shows 100% utilization and searches can last for hours. One day (it was a beautiful spring morning to be precise) my boss asked me to connect 10 more workstations to the database and to add remote access to the database via the Internet and dial-ups. I just about had a heart attack.

### What Needed to be Done

Of course, we needed to rewrite our database from scratch with client server and Intranet support. Yes, we *had* to, but we are a government organization with a very limited budget. When I asked local software developers about the possible cost for this rewrite, they told me that because it would be a mission-critical client-server application, with extended network and Internet support, as well as maintenance, the price would be about \$20,000 to \$40,000. Now my boss had the heart attack. That's not all—our database is very special. It was written for very specific tasks and was tested for a long time with bug fixes added during the experimentation period. We just couldn't afford to write such an application ourselves from the beginning, going through all the same problems once again. And even more—we needed to add remote access to the database immediately, so we needed another solution.

### The Solution

I have used our Linux box to solve some of these problems in the following ways:

- I moved the database from the NetWare file server to Linux.
- I installed the most recent version of the DOS emulator (dosemu-0.64.3) available at that time.
- After configuring and starting up the **dosemu**, I mounted the directory with my database as a local disk (Z:) in dosemu and installed the application.
- I started the application to be sure it worked—it did.
- I configured dosemu to use the exact amount of memory my application needed, and I checked the terminal specific configuration lines in the dosemu configuration file.
- I made a TELNET connection to my site and started dosemu with my application—this also worked.
- I made a dial-up connection to Linux and started dosemu from the shell prompt with the application—it worked again.
- I started X using **xdos**—another success. Oops, now I was running four copies of my application at once.
- Last, I shared the database directory via Samba or MARS\_NWE (NetWare server emulator for Linux), mounted it on the other workstations as a normal database disk, and it worked that way too.

### Perceived Gains

Our DOS/Windows workstations can continue using the database in the old manner. All that has changed is that the file server for the database is now

Linux, and clients can be connected to the database not only via NetWare protocol but also via the Microsoft Network using Samba—not bad.

Since more than one copy of the DOS emulator can be run at one time on Linux, it is possible for users to access the database from the Linux console, the X Window System, an X terminal, a serial terminal, a remote TELNET connection and a remote dial-up connection. Note that if the database application is started from dosemu, it works 10 times faster because, in this case, it uses data not on the network but on a local drive.

After using this configuration for about two months, I've found running the application from dosemu on Linux is more stable than running it from Windows 3.11. There are some tricks to prevent data loss in case of a broken remote connection to dosemu. By the way, while experimenting with MARS\_NWE, I found it to be very stable, powerful and fast. Data transfer speed from Linux via MARS\_NWE remains almost as fast as it was from the native NetWare 4.10 file server, so users connected from DOS/Windows workstations do not see much difference. And all this was accomplished during one night without any programming or additional capital investment.



**Dmitrie Komarov** is the Network and System Administrator of the Police Department of Riga, Latvia. He wrote his first programs in BASIC when he was 11 years old and had never seen a computer. He has been a Linux fan since kernel 1.2.13 and expects to remain one forever. He can be reached via e-mail at [dmit@rgpp.gov.lv](mailto:dmit@rgpp.gov.lv).

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

[Advanced search](#)

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



Advanced search

## Satellite Remote Sensing of the Oceans

**Simon J. Keogh**

**Emmanouil Oikonomou**

**Daniel Ballestero**

**Ian Robinson**

Issue #48, April 1998

Presented here is an overview of the kind of remote sensing that is done at Southampton University and how Linux has helped improve our productivity.

The remote sensing group at Southampton University Department of Oceanography (SUDO) works on many different aspects of Earth observation for applications such as climate monitoring, pollution control and general oceanography. Linux first appeared in the Oceanography Department in 1994 and has steadily become the workhorse OS.

Earth observation is now a familiar topic to almost everyone. Many of us are used to seeing satellite images of the Earth every day whether it's in the news, TV documentaries or just the national weather bulletin. More and more Earth observation satellites are being launched all the time. Here at SUDO we deal primarily with observation of the world's oceans, and we use many types of satellite data in studying various surface features. Satellites produce vast amounts of data, and large satellite images can be very difficult to use for this reason. The days when a low-powered PC could churn through a satellite dataset are long gone. Today we are required to process more and more images as the speed of the hardware gets faster.

We use IDL (Interactive Data Language) to process all our satellite and meteorological data. This package is very good as it allows us to read, warp and manipulate images easily as well as providing lots of useful built-in math functions for detailed analysis of our data. IDL (<http://www.rsinc.com/>) is now on its fifth release and is extensively used by the satellite community all over the world. IDL places a lot of demands on the system, so on anything less than

the quickest 486, IDL will be incredibly slow. These days IDL is available for almost every type of platform.

Sooner or later, when a machine is pushed to the limits to do its job, you may have to worry about the operating system (OS) you are using. When you are running jobs that take hours or even days to run, the last thing you need is your system to crash or hang. Similarly, when your system isn't fully loaded, you would like to be able to run several jobs to make more efficient use of your resources. A few years ago we discovered that by using Linux we could accomplish all of this. As usual though, the change had to come from within in order to establish Linux here and show its strengths against systems such as MS Windows.

### **How We Found Linux**

When I arrived in the Department, it became apparent that I would need a reliable method of wading through the enormous amounts of data I was required to process. I became particularly frustrated with the Microsoft Windows environment which frequently crashed and gave everyone headaches. In our research we can't afford to have our programs crashing, particularly since they take so long to run. In those days you could perhaps process one image in an hour, if your top-of-the-line 486 was up to it, so frequent multiple crashes could quite easily eat up much of the day. Of course, our single SPARC station never had these kinds of problems so there was fierce competition between colleagues to use it. We tried at first to use the machine remotely from our PCs using Vista Exceed. This proved very successful at first, but eventually the SPARC station became so overloaded with users that it was quicker to go back to using Windows again.

Then one of our Mexican Ph.D. students, Miguel Tenorio (now at CISESE, Ensenada in Mexico), told us of an exciting OS called Linux which could turn a humble PC into a powerful Unix workstation. I was very skeptical at the time and refused to believe that the Slackware version he had running on his 386 could possibly improve our efforts, especially since he was not able to run X because of memory limitations. However, he proved us all wrong later when he installed a full Slackware version with X on one of our better 486s, and since that time we have never looked back. Now our group has a suite of high-spec 486 and Pentium machines running Caldera Linux and Slackware Linux and utilising the powerful processing and data analysis power of IDL. IDL is the software we use most for our data processing, although occasionally we have to write the odd C program or Unix script. IDL has been around on Unix and Windows for a long time and was recently fully ported to Linux, much to our relief.

## Remote Sensing Applications—Global Warming

Global warming is a phenomenon that might very well affect us all one day. By using data from infrared radiometers in space, we can see how the average sea surface temperature (SST) changes over time so we can tell from the satellite image archives whether or not we are seeing changes in the Earth's climate. Because the ocean has such an immense thermal capacity, even a small change (e.g., 0.1 degrees C) in the average SST can imply a huge change in the Earth's heat budget. Unfortunately, the SST as measured from a satellite is a measure of the temperature of only the top millimeter of the ocean and does not always reflect the true sea surface temperature a few centimeters below because of the cooling effects of the wind and evaporation. Sometimes the “skin temperature” of the ocean can be as much as half a degree cooler than that of the bulk temperature just a few centimeters below. It's therefore important that we understand how the skin of the ocean behaves under different meteorological conditions, so that we can apply a correction to the satellite measured SST to account for the variability of the temperature of the ocean skin layer. After all, the skin temperature variability (up to 0.5 degrees C) is larger than the sort of changes in SST we are looking to measure for global climate research (0.1 degrees C).

The biggest source of error in estimating SST from space is the atmospheric absorption due to water vapour in the atmosphere. One method of dealing with this is employed by the Along Track Scanning Radiometer (ATSR) on board the ERS-2 remote sensing satellite. This radiometer views the sea at two different angles, 0 and 55 degrees to the vertical so that there are two images for every patch of sea. By looking at the difference between these two images, taken through two different thicknesses of atmosphere, a correction factor can be calculated to adjust the images for atmospheric absorption.

To study the atmospheric effects and the skin effect on measuring SST from space, I use marine infrared radiometers to measure SST at the same time as the satellites pass overhead, thereby getting simultaneous SST measurements. The ship I use is a ferry vessel, the MV Val de Loire (Figure 1) which sails regularly across the English Channel. Figure 2 shows a thermal infrared satellite image of the English Channel for January 1997 at which time the MV Val de Loire was approaching the French coast. The ship's radiometer and meteorological data are currently being used to evaluate the effect of the cool ocean skin on remotely sensed SST under a wide range of environmental conditions.



Figure 1. MV Val de Loire Ferry European Space Agency

### **Figure 2. Infrared Satellite Image of English Channel European Space Agency**

Using IDL under Linux I have found I can process all my data simultaneously rather than running batch jobs that eat up the whole DOS machine. One of the future ideas for this project is to have the ship data telemetered back to base here in Southampton, so that it can be processed and archived in real time rather than collected by hand. However, we have to await further funding for that to happen.

#### **Remote Sensing Applications—Synthetic Aperture Radar Images**

Synthetic Aperture Radar (SAR) imaging of the Earth is becoming increasingly popular due to the fact that these radars can be used regardless of the atmospheric conditions—they easily penetrate clouds, whereas clouds absorb almost all the sea surface infrared signals of infrared imaging instruments. The SAR gathers global information by emitting a beam of microwave radiation towards the sea surface of the world's oceans.

### **Figure 3. SAR Imaging European Space Agency**

If the sea surface is smooth, i.e., calm conditions, the surface acts like a mirror reflecting the incident beam in a direction away from the satellite. If the sea surface is roughened by wind or currents, then some of this incident beam is reflected back to the satellite and is received by the SAR. Thus, the stronger the backscattered signal, the rougher the sea surface.

Marine surface pollution is something we see all too often on television and in the news. The enduring images of stricken sea birds and baby seals on oil-soaked beaches put a lot of public pressure on environmental agencies to monitor marine pollution and catch the culprits who may be illegally dumping oil/waste off our shores. There is work going on in our group related to the observation of sea surface slicks, both man-made and natural. This type of work is suited to "SAR" studies.

The SAR on board the ERS-2 satellite sends images of the ocean surface back to Earth receiving stations on a regular basis. The SAR images reveal a surprising

amount of structure on the sea surface reflecting just how rough the sea actually is at the time the image is acquired. Where the sea surface is rough, the radar beam is strongly scattered back to the satellite antenna; and where it is smooth, the beam is strongly reflected from the sea surface away from the antenna.

Slicks have the effect of calming the sea surface and damping wave motion thereby resulting in most of the radar beam being reflected away from the antenna. On a SAR image, slicks typically appear as dark patches on the sea surface corresponding to calm conditions. Using IDL under Linux, sophisticated processing routines have been developed that reveal slick-like features on the sea surface and provide information about their position. Wind speeds can also be determined from the radar backscatter which helps to predict where the surface wind-driven currents may move the slick. The main problem with processing this data is the image sizes. The raw SAR images are 130MB in size and take a long time to process before they can be displayed on the screen. Linux has proved to be much more reliable at handling SAR data than MS Windows, and it is much faster too.

#### **Figure 4. ERS SAR Coast of Greece, 1996 European Space Agency**

The scene in Figure 4 is a good SAR image of the Gulf of Thermaikos in the Aegean Sea, just off the coast of Greece. The image was taken by the ERS-2 SAR on 25 May 1996 at 20:43 GMT. A number of oceanographic features evident in this image which are of general interest. The numerous black swirls and bands across the water surface correspond to surface slicks. Much of the slick material comes from a river outflow at the edge of the city of Thessaloniki, seen at the top left in the image. This riverine material is concentrated around the top of the bay and is then distributed throughout the gulf by eddies, tides and wind-driven currents. Incidentally, the dark square patches around the town are probably rice fields which give very little radar return signal. In the bottom right of the image is the edge of Mount Olympus, mythical home of the Greek gods.

#### **Figure 5. ERS SAR Coast of Greece, 1995 European Space Agency**

Figure 5 shows the same area one year earlier, but this time there is very little of oceanographic interest in the image. The sea appears to be bright, which implies that most of the radar signal had been backscattered towards the SAR. This suggests that the sea surface is very rough (wind speeds greater than 10 meters per second), rough enough to break up any slick material and destroy the surface signatures of eddies and weak currents. Around some of the Eastern coastline are dark patches which correspond to areas of water which are sheltered from the effects of the wind by the mountains and are therefore calm, backscattering very little of the radar beam towards the SAR.

The full potential of SAR is yet to be realised, and the European Space Agency is keen to see the SAR data it produces fully used. Military applications include looking for surface vessels and the surface signatures of submarines, although here at SUDO we deal strictly with the oceanographic science.

### **Remote Sensing Applications—Ocean Colour**

Modeling of phytoplankton blooms and the subsequent chlorophyll concentrations is also done here at the University in conjunction with satellite ocean colour data. This data reveals information about pigment concentration that is a measure of the biological activity in the water. The pigments are part of the phytoplankton's biological strategy for getting energy from sunlight (photosynthesis) so they can live. The study of phytoplankton blooms is very important for the study of the carbon cycle and its global warming implications.

#### **Figure 6. Ocean Colour Image/Airborne or CZCS**

#### **Figure 7. Thermal Image from AVHRR Satellite**

Figure 6 shows an interesting image from the Coastal Zone Colour Scanner (CZCS) which is an instrument which flew on the Nimbus 7 satellite. The instrument is no longer functional but worked well between 1978 and 1985. The image data were acquired on 14/9/80 and show the Western coast of the Iberian Peninsula. The image shows pigment concentration during a strong upwelling event. (Equatorward winds push the water away from the coast, and cool water from beneath the surface is drawn upwards near the coast.) The pigments are produced by phytoplankton.

The subsurface waters are generally cooler in the coast than the surface and in Figure 7 this is shown on a coincident thermal image from the AVHRR satellite as the blue, cool area. So the high pigment concentrations in the CZCS image can be explained by the fact that the upwelling event observed in the thermal image has led to the pigments being brought closer to the surface where they are more visible to the CZCS satellite instrument. Also, nutrients upwell with the phytoplankton and as they are closer to the surface, where there is more light, they are able to photosynthesise more effectively and thus form large blooms. This multi-sensor approach to oceanography (using complementary data from different sources, e.g., WAR, thermal and visible imagery) provides a more comprehensive view of a region than would be obtained using only one source of data.

## **Specification of Workstations**

For serious image processing you need a fast machine with good graphics support. For satellite images you also need vast amounts of storage. So I will talk about these in turn, bearing in mind that cost is always a factor.

### **Motherboards and Processors**

At the moment Intel P200 and AMD K6 processors are very fashionable although price-wise a P166 will give comparable performance for much less money. It's difficult to make price comparisons though because here in the UK electronic components are generally more expensive than in most other countries. The Intel 430TX motherboard is generally the one I would choose at the moment, USB and Ultra DMA support being standard.

### **Monitors and Graphics Cards**

Depending on the amount of time you spend using your machine for graphics I would recommend at least a 17-inch colour monitor. We do have some Illyama 21-inch monitors, but at the moment those extra few inches double the price of the monitor. A fast graphics card with lots of on-board RAM will make your machine update the display much faster, especially if you are using large images. Any S3 card (e.g., S3 trio v64+) with 2MB+ on board should give you enough to cope with most demands, although a 4MB card should give plenty of scope for dealing with vast displays, especially when using the monitor at its highest resolution.

### **Disk Space**

Our group has about 10GB of storage space allocated on the network server, which is almost enough. If you need speed, you need a lot of disk space local to the machine. The local hard disks of workstations are rarely backed up, so beware of depending on it too much. About 3GB of hard disk space is sufficient, and these days E-IDE is about as quick as SCSI and certainly cheaper. New IDE disks have Ultra DMA which allows a 33MB/s transfer rate, double that of the old IDE, although you will need at least the 430 TX motherboard to take advantage of this rate.

### **CD-ROM**

Many images are now distributed on CD-ROM because it is such a cheap way to distribute large quantities of data. A 12-speed CD with ATAPI controller will suffice for most requirements, although the speed of CD-ROM drives is getting faster by the month.

## Software

As I have mentioned, IDL is our favorite package for dealing with satellite images even though many are available. Most tend to be inflexible and tailor-made for doing specific types of image analysis. IDL can do most of the same operations more cheaply and flexibly while allowing you to interact with the data and merge data from several sources. However, IDL is a programming language in its own right, so there is a learning curve inherent to using it.

The choice of other system components is not as critical as choosing the video card, motherboard, processor and monitor. When it comes to backing up our data, an Exabyte drive suits us nicely for backing up anything less than 5GB in size.

## Further Points

Again, IDL is the main processing software for this type of work. Many users feel they are comfortable using LaTeX under Linux too, although this appears to be a point of contention. Most of us are still locked into using MS Word for want of a cheap Linux word processor that is Word compatible and can handle multiple data formats and equations. We are now looking at some alternatives like Applixware.

Linux has reduced the computing cost of the Oceanography Department's satellite remote sensing group in terms of both time and money. Its stability has been its greatest asset in converting die-hard Windows fanatics into potential Linux gurus. For many of us Linux has enabled us to embark on a voyage of discovery into the computing world. We now have a deeper understanding of how our PCs work as Linux has brought us closer to the machine. We are currently reviewing the hardware and software requirements of the group for the next couple of years. We plan to continue using Linux well into the next millennium and are quite happy with the decision.

## Acknowledgements

The SAR images in this article were obtained free from the European Space Agency.





**Simon Keogh** ([sjk2@soton.ac.uk](mailto:sjk2@soton.ac.uk)) is a graduate in Astrophysics from the University of Leeds and is now a NERC-sponsored Ph.D. student at the University of Southampton studying the oceanic thermal skin. In his spare time he enjoys golf, soccer and travelling.



**Emmanouil Oikonomou** ([oikono@soton.ac.uk](mailto:oikono@soton.ac.uk)) Emmanouil Oikonomou is a Greek Ph.D. student studying SAR imaging and fluid dynamics. He spends his spare time directing and scripting short movies for fun.



**Daniel Ballester** ([dab2@soton.ac.uk](mailto:dab2@soton.ac.uk)) is a Costa Rican Ph.D. student studying ocean colour.



**Dr. Ian Robinson** ([ian@corp.u-net.com](mailto:ian@corp.u-net.com)) is the Head of the Oceanography Department's satellite remote sensing group, and his interests on the subject vary from SAR to Altimetry, Ocean Colour, Thermal Imagery and general remote sensing.

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

[Advanced search](#)

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

Advanced search

## Small Business Marketing of Linux

**Cliff Seruntine**

Issue #48, April 1998

Linux is a good business product. This article deals with the why, how and who of selling Linux.



I have a standard Linux sales pitch that goes like this: "Linux is not what it was, but what it has come to be. Born of an idea by Linus Torvalds, evolving from infancy as an OS just powerful enough to control a disk drive, it grew into a very powerful, Unix-like operating system with the ability to meet the needs of such demanding users as medical centers, theoretical physicists and communication corporations, as well as the more modest needs of small businesses and end users. Linux offers you the power to accomplish whatever task you may have at hand with unparalleled stability and reasonable ease of operation. Software is abundant, help is as close as the Internet, and the potential is limited only by your imagination and computing skill. The bottom line is that this incredibly flexible and powerful OS is free."

What I like about selling Linux systems is that my sales pitch is absolutely true. I haven't embellished anything in order to score a sale. I have given the client the whole truth and nothing else. From that modest beginning, I can listen to the client's needs and design a Linux system tailored to meet them. Tucked into a spare bedroom, my small business, *cybertronics* (<http://cwolf.uaa.alaska.edu/~asacs7/main.htm>), is not a Microsoft-sized operation by any means. From my home workshop I conduct repairs of various electronic equipment from TVs to computers, build customized systems and maintain contact with the array of suppliers, clients and persons to whom I contract out various jobs. My partner, my wife, manages the paperwork end of the business, and I run the technical side.

One would think in this day and age of big conglomerates taking over the technology industry, that small businesses like mine would be rapidly shoved out of the way or swallowed up, but Linux is an asset that has allowed us to remain competitive. I offer an unusual, powerful product. The open source code allows this OS to be customized for whatever specific uses a client might have. And the fact that it is free allows my prices to remain extremely competitive.

I have come to specialize more and more in Linux systems, and I hope within a year or two to focus on Linux systems as my primary commercial product. That will all depend on how Linux continues to evolve, as the literature I have read on the Internet indicates the OS is again at a crossroads and experiencing the associated growing pains. On the Internet, two opposing sides can be seen arguing over the way Linux should go in the future as it continues to grow and become more powerful. On the one hand, those opposed to the simplification of Linux usage to expand its mass appeal argue the OS should remain complex to force the user to learn something and ensure that only dedicated computer hobbyists and aficionados will use it. On the other hand there are those that would have the development of Linux travel down a road of utter ease so that it resembles Windows 95. At *cybertronics*, I have drawn my own conclusions (and sell Linux systems by them) based on the hard practicality that makes business work:

- Linux is a powerful OS. It has been so for a long time. But it is only recently, due to advances made largely by Red Hat and Caldera in making Linux more user friendly, that it has become a truly marketable product to a diverse client base.
- The growth of Linux and related software depends on the mass acceptance of the OS. The more users there are, the more people and companies will be turning out applications and making improvements.
- The option of simplification is a good thing, as long as Linux is left with enough complexity to make the full power of the OS available for those users who desire or demand that kind of control. It is also desirable to leave that power and intimate control capability in place, right down to manipulation of the source code, to maintain the unique flavor and spirit of Linux.

Linux is now at this crossroads. Simplification has been added so the end user can use the system, while full control can be accessed by the experienced user for greater power and enhancement of the OS. This is what makes Linux different and marketable today to a broad range of clients. I can sell it to an average end user who simply wants a good, inexpensive system to balance his books, do word processing or access the Internet—and this represents the majority of computer users—because it has become relatively easy to use. I can

also sell it to a client who must maintain a huge database of corporate records or run a powerful web server because that power is also easily accessible. Because Linux currently maintains a healthy balance of end user simplicity and ultimate user control, it is a highly attractive product to a wide range of people.

The second aspect of Linux which I emphasize strongly to my clients is the incredible stability of the system. I own three computers: an AMD-based system, a Cyrix-based system and an older Intel-based system. Linux has never crashed on any of these. On a couple of occasions, applications have misbehaved or have closed down, but this always turned out to be the result of a poorly programmed application, a user error, or the program's use of some new, experimental aspect of Linux that hadn't yet been tested and proven. But never, in all the time I have used Linux, have I seen it crash a system due to an instability problem. For businesses that believe time is money, stability is a major advantage. When computers crash, businesses lose not only the unbacked-up data on the system, but also time paid to employees while they wait for the system to come back on-line. If a network server crashes or becomes unstable, it can cost many thousands of dollars as computers throughout the company are affected. Even worse, customers tend to have little tolerance for businesses with faulty equipment. They will take their business elsewhere, sometimes for good.

A third and extremely important point in selling Linux is all of the software that is available, most of which is free. I have tried on several occasions to search the Internet for all Linux software to develop a comprehensive list for my clients. But the list is always growing and changing. And there is just so much software out there, much of which is good yet hard to find, that I've given up creating a comprehensive list. I doubt there is enough time in a month of Sundays to gather up Internet addresses to all the software and evaluate everything I can find for Linux that is free. This brings up an important point.

From the standpoint of business, professional software that is designed and sold commercially for Linux is extremely affordable, such as the office suites sold by Caldera (Star Office) and Red Hat (Applixware). In many cases, a company can purchase and outfit an entire network with professional quality productivity software that takes advantage of the full networking, stability and processing power of Linux for less than a thousand dollars. This selling point is truly great. The trouble is, it sounds too good to be true. You know the old adage, "You never get something for nothing." Well, business people tend to be very cautious with money (especially the small to medium-sized businesses I often work with), and they become dubious at hearing how Linux can offer so much for so little. This point in the pitch is a good place to have references and

demo software available, and it is also an especially important place to understand the aspects of Linux that make such high quality so affordable.

- The Linux OS is free and freely developed by computer program developers around the world. Therefore, there are no royalties to be paid by software companies making professional business software, and development costs are cut to almost nothing.
- Companies producing Linux software for profit are engaged in hard competition with software giants, including Microsoft, and are turning out superb products to gain an edge since they cannot compete in financial and advertising resources.
- The spirit of free software is active even in the businesses involved with Linux. While those businesses are not running charities, they help promote the growth of Linux by making their products more widely available through reasonable prices—and anyone familiar with the Unix business knows that the price tag behind that OS and its software has killed its presence in the end-user market (a sad way for such a fine OS to go).

The only downside I have found is that for the private consumer wanting a more powerful computer system, professional Linux software is often too expensive. More competitively priced items are available for Windows for many applications—but this is changing. Caldera makes Star Office freely available to private users. Also, I have found a number of practical, good applications available at more reasonable prices within the last year—everything from web design software to data backup tools.

Moving on, it is important to note in selling Linux that there are many less obvious reasons for a client to turn to a Linux system.

- The way Linux manages memory is very effective. No need to worry about high memory and low memory—Linux uses it all.
- Linux's ability to communicate with other operating systems enhances its flexibility.
- Linux's efficient and powerful programming enhances its speed.
- Source code availability allows Linux to be entirely customized to fit the needs of the user.
- A good distribution of Linux comes with so much free software that a business can often find most of the applications it needs on the Linux CD.

For the business person considering marketing any product, a question as important as “What product shall I sell?” is “Who shall I sell it to?” One would think with the broad potential for client usage I have described above, that anyone who uses a computer would be a potential client. In some ways I have

found that to be true. I have sold Linux to casual end users who want more power than an old DOS-based system has to offer. I have also sold it to people who just want to learn what makes their computers tick. I have found that to compete, Linux must and does appeal to certain groups of people in particular. Since advertising costs what it does, it is important to target your business' ads and make each dollar count.

Is Linux a gamer's OS of choice? I am sorry, but it's not. Yes, there are a lot of games out there for Linux. Unfortunately, most of them are reminiscent of the earlier days of computing when two dimensional arcade-style games like *Asteroids* and *Space Invaders* were the rage. There are some other, more advanced games, I'll grant you, like *Quake*. But Linux is really lacking in the intense, power demanding, multimedia games that are available these days for other systems. And, let's face it, currently Linux is too demanding for the needs of someone who only wants a super gaming machine. And that is, in my opinion, too bad, because Linux has so much power and stability to offer. On other platforms, games crash or move jerkily. Both are frustrating, and I believe that Linux harnessed for such applications would make the ultimate gamer's computer.

Nor is Linux the first choice for the end user who wants as little to do with his computer as possible. One of my best customers, an elderly gentleman who has been coming to me for a couple of years now, wants his computer to get his stock quotes and publish his letters—that's all. The learning curve of Linux is still too high for him. He wants to do his work, and then get off his computer. He calls me from time to time with frustrations about how his current OS has led to another computer crash, but the occasional crash is an easier burden for him to bear than learning even the simplest administration of a Linux system. Until a shell for Linux is created which greatly simplifies its usage for the least demanding of end users, Linux will not appeal to this market.

My primary sales strategy is aimed at end users who want a system strictly for some form of business, be it personal bookkeeping, writing or running their small business. They like Linux because of the strong sales points I have already mentioned. Linux's stability ensures against losing time or work and is the biggest selling point for these individuals. Often they run their businesses with little or no help. They need to get things done. They don't have time to waste. That stability helps ensure that they will be able to do their jobs in the least amount of time so they can move on to other projects.

For representatives of larger companies, power is the factor to emphasize. Knowing that they can take this OS and put it to serious work, thereby saving their companies thousands of dollars by not buying commercial operating systems such as Windows NT or some other Unix is very impressive. Of course,

the stability of Linux is also appealing to them. It is like icing on an already delicious cake.

Certainly, there are many other people for whom Linux could prove to be a useful and desirable product. However, the two groups I have just mentioned form the lion's share of my Linux business, and they are the market segment where, in my experience, advertising dollars should be aimed. At this time, these two groups comprise the broadest consumer market to which Linux seems most appropriately tailored. Both of these groups represent practical people with things to do.

I have found a brief, one page brochure with lots of illustrations that quickly and plainly point out the facts is a great way to introduce clients to the idea of a system based on Linux. If they are interested, I will happily take a system to their place of work and demonstrate it along with the features that make it particularly beneficial to them. I keep these presentations to fifteen minutes unless they indicate a desire to go deeper into the system. I can expand these presentations as needed to give potential customers all the information they want.

Since Linux is a complicated system, I deliver a Linux computer to the client already configured. I maintain good communication with these busy people by regular follow-up to make sure everything is going according to their expectations, and I go the extra mile to help them incorporate their Linux computer into their business as a vital and necessary component of administration. The most common complaint I hear from any of my customers is, "I've found something I just can't understand," so I offer up to six free hours of consultation to help my clients over the obstacles that inevitably crop up.

Finally, you wouldn't attempt to fly without a little formal training. Nor would you buy a new car and not receive an owner's manual so you could learn how to take care of it. Well, in the same way, I don't turn over systems without documentation. My systems come with a copy of an easy to read, good reference on Linux. Que's *Using Linux* by Jack Tackett, Jr. and Dave Gunter (1997) has proven especially useful in that respect. To get my clients excited about their Linux system, I also give them a free issue of *Linux Journal* as a way of introducing them to the growing world of Linux users everywhere that they have just entered.

Linux is a fine and marketable product. At *cybertronics*, I foresee a bright future for the OS and for businesses marketing and selling it. It is my hope that Linux will continue to evolve, becoming easier for everyone to use and maintain while retaining all of its power and accessibility. With this combination and a little enthusiasm, Linux can rekindle the days of the eighties when computer

companies sprung from home basement workshops and pioneered new frontiers.



**Cliff Seruntine** owns and operates cybertronics, a computer and electronics service center, from his home in Anchorage, Alaska. Currently a student at the University of Alaska in Anchorage, Cliff spends his spare time engaged in writing, reading, hiking and boating. Cliff can be reached via e-mail at [asacs7@uaa.alaska.edu](mailto:asacs7@uaa.alaska.edu).

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

[Advanced search](#)

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



## Building Projects With Imake

**Otto Hammersmith**

Issue #48, April 1998

Here's an explanation of how Imake works and how you can use it to build your executables—an article for programmers with C and Unix programming skills.

Imake is tool for configuring the X Window System and its components for different platforms and compilers. Imake allows you to create a generic description of how your project should be built and leaves the system-specific details to centralized configuration files.

Put simply, Imake is built on the idea of applying the C preprocessor to Makefiles, rather than C programs. Much of the redundancy in common Makefiles has been turned into preprocessor macros, which have the same effect macros have in C programs—making things easier.

### Imake Commands and Configuration

Imake should have come with your installation of X. If you run one of the common distributions of Linux, you need to check that some of the “development” packages are installed. For instance, I run Red Hat 4.0 on my system, and the Imake system is contained in the XFree86-devel RPM.

If you aren't sure if you have Imake installed or have it configured properly, it is easy to check. Just run these commands:

```
touch Imakefile
xmkmf
```

There should now be a new file called Makefile in the current directory. There is quite a bit of “stuff” in the new Makefile, mostly generic rules and information generated from the configuration files.

If you've gotten this far, Imake is installed and configured just fine. Now, you should be able to run **make** and get output. For example on my Red Hat 4.2 system, I type **make** and get the following output:

```
make: Nothing to be done for 'emptyrule'.
```

### Supporting Programs

Besides containing many programs, there are several programs that Imake depends on to function properly. However, most of them are basic things like **bash**, **mv**, **rm** and so on. If these basic programs are missing, you have more severe problems than whether Imake works properly. Still, there are a couple of major components that aren't vital to a working Linux system that are vital to Imake.

### make

**make** isn't necessarily vital to the functioning of Imake, but the generated Makefiles are absolutely useless by themselves. If you don't have make, you should install GNU make. For more advanced uses of Imake, it is good to have a thorough understanding of how make works. At the end of this article are some references for more information on make.

### GCC

Obviously, for a Makefile generation program to be useful, you need a compiler. However, that is not the only reason for having GCC installed. Imake relies on an external C preprocessor, installed in `/lib/cpp` by default. If you have GCC installed and configured, you should also have **cpp** (the C preprocessor) installed.

In case you don't have a `/lib/cpp` file, the simplest way to solve the problem is to make a symbolic (soft) link (use **ln**) from it to GCC's `cpp` executable. For example, on my system, `/lib/cpp` is a symbolic link to `/usr/lib/gcc-lib/i486-linux/2.7.2/cpp`.

### Imakefile Contents, a Jump Start

The major user-visible component of the Imake system is the file `Imakefile`, in which the project-specific information needed by Imake to build your project is kept.

At a basic level `Imakefile` consists of rule invocations, variable definitions and variable invocations. The first, rule invocations, resemble C-function calls. The second resemble C-variable assignments. These two should be familiar to anyone who has used macros in C code. On the other hand, variable

invocations are nothing like their C counterparts, but should be familiar to anyone who has done any work with plain make. Here's a simple example:

```
VARIABLE = value  
ARule(arg1, arg2, $(VARIABLE))
```

As with programming in general, there are many of syntax pitfalls with Imake. The following two sections describe a few of the problems to keep in mind.

### Argument Warnings

Imake (actually, the underlying C preprocessor) is fairly strict about how the various rules must be invoked. First, there must be absolutely no extraneous white space around the commas between arguments. For example, this rule is **incorrect**:

```
ARule(arg1, arg2, arg3)
```

and this version is **correct**:

```
ARule(arg1,arg2,arg3)
```

To confuse things even more, in some cases, it is valid for the argument to have space. For example, many rules take a list of files as one of the arguments, where each file is delimited by a space. Typically, this is fairly intuitive, and several of the examples later in this article show the most common cases.

The second problem with argument passing in Imake is empty arguments. Not all arguments are necessary for every rule, but the C preprocessor breaks if an argument is left out or has no value. To solve the problem, Imake provides the macro `NullParameter` to allow empty arguments in a rule. For example:

```
ARule(arg1,NullParameter,arg3)
```

### Comment Warnings

There are two different ways to include comments in your Imakefile. The first is standard C-style comments, opening with the `/*` characters and closing with the `*/` characters. Anything between the two is completely ignored. The second style of comment uses the string `XCOMM` to start the comment, and a line break ends it.

The basic difference between the two is what appears in the final generated Makefile. C-style comments don't show up at all in the generated Makefile, while `XCOMM` comments do. Put simply, instances of `XCOMM` are replaced with a `#` character. One item to remember about `XCOMM` comments, since the C preprocessor doesn't know to ignore them, it will gladly process anything it

sees as a macro inside an XCOMM comment. In other words, an invalid rule invocation that looks to be commented out can cause the Makefile generation to fail.

## The (Common) Rules

There are many rules provided by the Imake configuration files. Some of the most common and useful are described here. Note that the rules used in the presented examples are not necessarily compatible with each other. Do not intermix rules from the different examples.

### Rule 1. SimpleProgramTarget()

This rule, as you can tell by its name, is the simplest one that Imake provides. It generates a simple Makefile that compiles a single program from a single source file. Say you have a source file, `prog.c`, and you wish to compile it into an executable called `prog`. You need only a single line in the Imakefile such as this one:

```
SimpleProgramTarget(prog)
```

To try it out, write a short “hello world” type program in C and an appropriate Imakefile. Then, run **xmkmf** and `make`. Your “hello world” program should compile without problem.

### Rule 2. ComplexProgramTarget()

Since it's difficult to write complex programs with only a single source file, this rule was created to allow a single executable to be built from multiple source files. However, with the added flexibility comes a cost, you have to add information about which source files to use. For example, if you have a project with `foo.c` and `bar.c` that will create an executable called `foobar`, your Imakefile should look like this:

```
SRCS = foo.c bar.c
OBJS = foo.o bar.o
ComplexProgramTarget(foobar)
```

There are three other rules that are similar to `ComplexProgramTarget()` but not quite the same. They are called `ComplexProgramTarget_1()`, `ComplexProgramTarget_2()` and `ComplexProgramTarget_3()`. The basic idea behind these rules is to allow a single Imakefile to generate targets for more than one program.

Although, these rules are similar to `ComplexProgramTarget()` there are some minor differences. The first and most obvious one, is the different way to specify source files. Instead of setting `SRCS` and `OBJS`, you use `SRCS1`, `OBJS1`,

SRCS2, OBJS2, SRCS3 and OBJS3. The second is the rules take different arguments. These rules take an additional two arguments. The second and third arguments are a way to specify local and system libraries that must be linked into the final executable. Here is an example of how to use these rules:

```
PROGRAMS = foobar1 foobar2 foobar3
SRCS1 = foo1.c bar1.c
OBJS1 = foo1.c bar1.c
SRCS2 = foo2.c bar2.c
OBJS2 = foo2.c bar2.c
SRCS3 = foo3.c bar3.c
OBJS3 = foo3.c bar3.c
ComplexProgramTarget_1( foobar1, NullParameter, \
    NullParameter)
ComplexProgramTarget_2( foobar2, NullParameter, \
    NullParameter)
ComplexProgramTarget_3( foobar3, NullParameter, \
    NullParameter)
```

Note that the last two rules will not function properly without the first. You cannot have a `ComplexProgramTarget_2()` or `ComplexProgramTarget_3()` without having a `ComplexProgramTarget_1()`. The reasons for this are related to the internals of the rule definitions. To discover the reason why, read the O'Reilly *Imake* book mentioned in Resources.

### Rule 3. `NormalProgramTarget()`

This rule is by far the most useful. It can be used an arbitrary number of times and can use an arbitrary number of source files to build the executable. Again, along with added flexibility, there is added complexity. The usage of this rule given by `Imake.rules` is as follows:

```
NormalProgramTarget(
    locallibs, syslibs)
```

This is how the arguments are defined:

- **program:** name of the executable being built
- **objects:** names of object files to be built from source. This argument replaces the functionality of the `OBJS` variables in the various `ComplexProgramTarget()` rules.
- **deplibs:** libraries that the executable needs
- **locallibs:** local (to the project) libraries to link
- **syslibs:** system libraries to link

Note that in the last three arguments there is some redundancy. The *deplibs* argument is a list of the library file names which the program needs. This list is used by `Imake` to build a proper Makefile that will rebuild the executable if any of those files change. The last two arguments, *locallibs* and *syslibs* are the same libraries, but expressed as command-line options to tell the compiler to link

those libraries. Naturally, this is done so that Imake knows how to properly build the executable with all the necessary libraries.

Listing 1 uses `NormalProgramTarget()` to reproduce the functionality of the `ComplexProgramTarget()` example. The advantage, of course, is that program targets can be added or removed without affecting each other.

Note that the use of the variables `SRCS1`, `OBJS1` and so on are simply for convenience. Unlike `ComplexProgramTarget()`, these variables are not used in the definition of the rule. When it comes time to change the list of sources and objects, it's easier to find them at the front of the Imakefile as a variable definition than to search through the rule invocations to find all the instances.

On the other hand, the variable `SRCS` is a different case. Notice two new rules, `DependTarget()` and `LintTarget()`. These two rules create the targets **depend** and **lint**, respectively. To be able to do their job, these rules need the variable `SRCS` set to a list of the source files in the project. Note that in the previous examples the `depend` and `lint` targets were created automatically; the function of these targets is explained later.

There is one more new rule in this example, `AllTarget()`. Again, because `NormalProgramTarget()` does less work than the other rules, you have to do more work. Quite simply, each instance of `AllTarget()` adds another dependency to the "all" target in the final Makefile. `AllTarget()` is called by default for each program to be built.

### The Files

There are various files involved in the Imake build system; most reside in the Imake configuration directory. Though only one file, `Imakefile`, is really important to simple uses of Imake, these descriptions are included as a sort of a road map to more in-depth understanding of how Imake works.

- `Imakefile` is the file used to generate a Makefile. This file contains your project specific information.
- `Makefile` is the final output file. There are many references to help you understand the contents of the Makefile. The easiest to obtain is probably the GNU Info pages on GNU make, which come with the GNU make distribution. See the References section for specifics on information available about make.
- `Imake.tmpl` is a generic template used to fill in information not provided by the `Imakefile`. Most template files are created for large projects that need to customize Imake beyond what is possible through the `Imakefile`. (The FVWM source as an example.)

- Imake.rules and other .rules files hold the C preprocessor “defines” that make up the bulk of the Imake rules. Things like SimpleProgramTarget() and DependTarget() are defined in this file.
- .cf files contain system specific configuration information. Imake.cf is the file that “chooses” the proper platform-specific .cf file.

### Putting Imake to Work

Once you've written a basic Imakefile, it's time to put Imake to work. Fortunately, the difficult part is over. Just type the following two commands:

```
xmkmf -a  
make
```

If you've written your Imakefile properly, your project will build with no problems.

**xmkmf** is a Bourne shell script that wraps up the **imake** command. It calls imake with convenient default arguments and does some cleanup, such as copying the old Makefile to Makefile.bak. This program generates the Makefile from your Imakefile.

**imake** is a program that you almost never need to run by hand. It's the program that handles the brunt of running the C preprocessor, as well as some minor tweaking (i.e., XCOMM comments) for compatibility with make.

Without any arguments, xmkmf just saves the old Makefile and runs imake to generate a new Makefile from the Imakefile. If you give xmkmf the **-a** switch, it runs the following commands in the given order:

```
make Makefile  
make includes  
make depend
```

See the xmkmf(1) man pages for more details on xmkmf capabilities.

### The Targets

Below is a list of the common make targets contained in an Imake generated Makefile. Precisely which targets end up in your Makefile depend on which rules you used in your Imakefile, of course. A given target always does the same thing regardless of the project. (That is, it will if the Imakefile has been written properly.) In fact, these targets are standard beyond Imake. For example, the GNU Coding Standards specify much the same targets.

- **all**: Build the whole project. Usually the default target. This is the target to which the AllTarget() rule adds a dependency.

- **clean:** Remove buildable files and backup files. Includes all object files, executables, backup Makefiles and files commonly used by editors as backup files (\*~).
- **Makefiles:** regenerates all the Makefiles using Imake.
- **depend:** generates the dependences between source (\*.c) and header (\*.h) files and ensures that all object files that need recompiling are rebuilt.
- **tags:** builds the tags database used by vi and Emacs. The tags database is simply a list of C-language symbols and their location in the source. It allows for quickly and automatically locating function definitions and other such objects. See the man pages for etags(1), ctags(1), vi(1) and the Info documentation that comes with Emacs.
- **install:** installs the final binaries into the main file system. Defaults to a directory under /usr/X11R6/ depending on each type of component. For example, executables go in /bin; man pages go into the proper section directory under /man.
- **lint:** This target isn't particularly useful on Linux systems, since there isn't a commonly used **lint** program. **lint** is a program that checks C code for common style problems and other non-syntactic mistakes. The GNU compiler has moved much of the functionality of lint into the compiler, compile your code with the **-Wall** flag set and you'll have most of lint's functionality. With a lint program installed on the system, you can use this make target to run lint on all the source files of the project.

### Custom Imake Rules

In some cases, it is useful to write your own custom rules for generating Makefiles. The Imake system makes it easy to specify a different set of configuration files than the usual /usr/X11R6/lib/X11/config. This kind of customization is beyond the scope of this article, but O'Reilly publishes a great book on Imake that has several chapters devoted specifically to this topic. See Resources for more information on that book.

### Resources

Otto is a developer at Red Hat Software who is currently very busy with the next release of Red Hat Linux. He can be contacted by e-mail at [otto@redhat.com](mailto:otto@redhat.com)

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

[Advanced search](#)



[Advanced search](#)

## Linux Network Programming, Part 3

**Ivan Griffin**

**Mark Donnelly**

**PhD. John Nelson**

Issue #48, April 1998

This month we are presented with an introduction to the networking of distributed objects and the use of CORBA.

In the last few articles in this series, we dealt with basic low-level network programming in Linux, and with the issues involved in developing network servers (daemons). However, coding at this low level is extremely tedious and prone to error as a result.

Nevertheless, distributing an application over the network provides significant benefits over a monolithic, stand-alone structure—and these benefits make the additional development investment (in terms of time and money) worthwhile.

In this article, we will introduce an approach intended to reduce the added complexity in writing a network application by applying object-oriented techniques to the development and coding—namely, the use of the *Common Object Request Broker Architecture* (CORBA).

In the next article, we will present different software packages, available for Linux, which implement the CORBA standards. In addition, a basic introduction to developing applications with CORBA will be given.

Note that this article assumes a basic familiarity with C, C++, object-oriented development, the BSD socket API and RPCs. It presents the key concepts of CORBA which need to be understood before developing applications. These concepts, and the associated terminology, comprise much of the learning curve of CORBA. This article is necessarily theoretical in order to develop a vocabulary to be used when developing with CORBA.

## The Benefits of Distributed Computing

The motivations for structuring a system as a set of distributed components include:

- An increased ability to collaborate through better connectivity and networking
- The potential to increase performance and efficiency through the use of parallel processing
- Increased reliability, scalability and availability through replication
- A greater cost-effectiveness through the sharing of computing resources (for example, printers, disk and processing power) and the implementation of heterogeneous, open systems

Along with these impressive advantages comes additional complexity to the development process. The programmer now has to handle situations like the following:

- Network delay and latency
- Load balancing between the different processing elements in the system
- Ensuring the correct ordering of events
- Partial failure of communications links, in particular with regard to immutable transactions (i.e., transactions which must yield the same result if repeated—for example, the accidental processing of a bank transaction twice should not erroneously affect the account balance)

One of the earlier techniques in reducing the programmer's work (to develop these distributed applications) was the remote procedure call (RPC).

### Remote Procedure Calls

RPCs have already been introduced in *Linux Journal* in an excellent article by Ed Petron, "Portable Database Management with /rdb", October 1997. The motivation behind RPCs is to make network programming as easy as making a traditional function call. Linux provides the Open Network Computing flavor of remote procedure calls (ONC-RPC), which was originally developed by Sun Microsystems (Mountain View, CA) for use in its Network File System (NFS) implementation.

RPCs do nothing more than hide the implementation details of creating sockets (network endpoints), sending data and closing the sockets as required. They are also responsible for converting the (remote) procedure's (local data) parameters into a format suitable for network transportation and again into a format that can be used on the remote host. This network-transparent

transmission of data ensures both end machines can interpret the information content correctly.

Quite often, different representation issues are involved in this process (for example, byte ordering, ASCII to EBCDIC, etc.). Additionally, pointers (references to address locations in a virtual memory space) cannot be passed directly from one process/machine to another. Instead, a “deep copy” needs to be performed, copying the data pointed to by the pointer which is then sent “across the wire”. The entire procedure of transferring the data from the native format of a machine into a network-transparent format is called “marshalling”. The reverse procedure is called “de-marshaling”. The ONC-RPC uses a special type of marshaling called external data representation (XDR). Figure 1 shows the sequence of events involved in a remote procedure call. The mechanisms and semantics of ONC-RPCs are discussed in much greater detail in RFC 1831, RFC 1832 and RFC 1833.

## **Figure 1. Remote Procedure Call (RPC)**

### **An Object-Oriented Approach**

As mentioned, RPCs follow the traditional functional model. State information may need to be maintained independently at both client and server (depending on the type of application). This data is often repeatedly re-transferred across the network on each remote function call.

An alternative architecture is to use techniques from object-oriented development and to partition the system into a set of independent objects. Each object is responsible for maintaining its own internal state information.

By using an object-oriented approach to your network software development, you can promote certain beneficial traits in your code:

- Encapsulation: ensuring a clear separation between the interfaces (through which the objects in your system interact with one another) and their implementations
- Modularity, scalability and extensibility
- Re-usability (of code and, perhaps more importantly, of design)
- Inheritance and specialization of functionality and polymorphism

The act of sending a message from one entity on a network to another is remarkably similar to one object invoking a method on another object. The integration of distributed network technology and object-orientation unites the features of a basic communications infrastructure with a high-level abstraction of these interfaces and a framework for encapsulation and modularity—

through this, developing applications which inter-work is significantly more intuitive.

### **The Object Management Architecture**

In 1991, a group of interested parties joined to form the Object Management Group (OMG)—a consortium dedicated to the standardization of distributed object computing. The OMG supports heterogeneity in its architectures, providing the mechanisms for applications written in any language (running on any operating system, any hardware platform) to communicate and collaborate with each other—in essence, the development of a “software bus” to allow for implementation diversity, as a hardware bus does for expansion cards.

The OMG architecture which permits this distributed collaboration of objects is called the *Object Management Architecture (OMA)*. Figure 2 shows the object management architecture.

### **Figure 2. Object Management Architecture (OMA)**

*CORBA services* provide the basic functionality for the management of objects during their lifetime—for example, this includes:

- Naming (uniquely specifying a particular object instance)
- Security (providing auditing, authentication, etc.)
- Persistence (allowing object instances to be “flattened” to or created from a sequence of bytes)
- Trading (providing objects and ORBs a mechanism to “advertise” particular functionality)
- Events (allows an object to dynamically register or unregister an interest in a particular type of event, essentially decoupling the communication from the object)
- Life-cycle (allows objects to be created, copied, moved, deleted)

*Common Facilities* provide the frameworks necessary for application development using distributed objects. These frameworks are classified into two distinct groups: horizontal facilities (commonly used in all applications, such as user-interface management, information management, task management and system management), and vertical facilities (related more to a particular industry, for example telecommunications or health care).

The CORBA standard specifies an entity called the Object Request Broker (ORB), which is the “glue” that binds objects together to enable higher-level distributed collaboration. It enables the exchange of CORBA requests between local and

remote objects. Figure 3 shows the architecture of CORBA. Figure 4 shows the invocation of methods on different remote objects via the ORB.

### **Figure 3. Common Object Request Broker Architecture (CORBA)**

### **Figure 4. Calling a method within a specific object instance**

#### **More about CORBA**

In the OMA, objects provide services. Clients issue requests to different objects for these services to be performed on their (the client) behalf. The repetitive transmission of state information that is common with RPC applications is avoided since each object is responsible for maintaining its own state. In addition, the objects interact through well-defined interfaces and are unaware of each others' implementation details. As such, it is much easier to replace or upgrade an object implementation, as long as the interface is maintained. The objects in an OMA/CORBA system may take on many different roles in relation to one another: peer-to-peer, client/server or publish/subscribe, etc.

Before an object can issue a request to invoke a method on an object, it must have a valid reference for that object. The ORB uses this reference to identify and locate the object—thus providing location transparency. As an application writer, you need not be concerned with how your application finds an object, the ORB performs this function for you transparently. In a similar fashion to how RPCs use XDR, CORBA specifies the *common data representation* (CDR) format to transfer data across the network.

An object reference does not describe the interface of an object. Before an application can make use of an object (reference), it must somehow determine/know what services an object provides.

Interfaces to objects are defined via the *Interface Description Language* (IDL). The OMG IDL defines the interface of an object by means of the various methods they support and the parameters these methods accept. Various language mappings exist for the IDL (for example, C, C++, Java, COBOL, etc.). The generated language stubs provide the application with compile-time knowledge which allows these interfaces to be accessed.

The interfaces, alternatively, can be added to a special database, called the *interface repository*. The interface repository contains a dynamic copy of the interface information of an object, which is generated statically via the IDL. The *Dynamic Invocation Interface* (DII) is the facility by which an object client can probe an object for the methods it supports and, upon discovering a particular method, can invoke it at runtime. This involves looking up the object interface,

generating the method parameters, invoking the method on the remote object and returning the results.

On the “server” side, the *Dynamic Skeleton Interface* (DSI) allows the ORB to invoke object implementations that do not have static (i.e., compile time) knowledge of the type of object it is implementing. All requests to a particular object are handled by having the ORB invoke the same single call-up routine, called the *Dynamic Interface Routine* (DIR). The *Implementation Repository* (as opposed to Interface Repository) is a runtime database of information about the classes the ORB knows of, its instantiated objects and additional implementation information (logging, security auditing, etc.).

The Object Adapter sits above the core ORB network functionality. It acts as a mediator between the ORB and the object, accepting method requests on the object's behalf. It helps alleviate “bloated” objects or ORBs.

The Object Adapter enables the instantiation of new objects, requests passing between the ORB and an object, the assignment of object references to an object (uniquely naming the object), and the registering of classes of objects with the Implementation Repository.

Currently, all ORB implementations must support one object adapter, the *Basic Object Adapter* (BOA).

All of this talk about interoperability is not useful unless ORBs from different developers/vendors can communicate with one another. The *General InterORB Protocol* (GIOP) is a bridge specifying a standard transfer syntax and a set of message formats for the networking of ORBs. The GIOP is independent of any network transport.

The *Internet InterORB Protocol* (IIOP) specifies a mapping between GIOP and TCP/IP. That is, it details how GIOP information is exchanged using TCP/IP connections. In this way, it enables “out-of-the-box” interoperability with IIOP-compatible ORBs based on the world's most popular product and vendor neutral network transport—TCP/IP.

### **Multi-tier Network Computing**

CORBA is an example of what is termed “middleware”—a technology that enables the separation of applications into three distinct sections (see Figure 5):

1. The presentation, or user interface, tier
2. The business logic, or control, tier
3. The data storage tier

## **Figure 5. Three-Tier Network Computing**

The presentation layer could be an HTML form or a Java applet (as in Figure 5). Extracting the control logic from the presentation allows you to network-enable the presentation of your application.

It also allows the end user to use less expensive hardware to interact with the application, since their equipment is now responsible only for rendering the information supplied to it by the control logic. In addition, the presentation system does not need to have any knowledge of where the data originally came from or in what format it is stored in the database—all it needs is the interface to the middle layer.

The control logic can perform access management, alter the view of the information as required to enable different users to view different subsets of the same data—perhaps for security reasons. For example, a doctor in a hospital may want to see the patient's medical history, whereas someone from the finance department should only be able to see billing information. In Figure 5, either Java or CORBA performs the role of the business logic.

By separating the control logic from the data store, you gain the benefits of distributed computing. Your logic can encapsulate database access, providing you with scalability and fault-tolerance for mission-critical data. All sources of corporate information can be integrated via the control logic to achieve what is termed a “data warehouse”—allowing all the information to be accessed via a single interface (depending, of course, on security clearance).

Legacy systems can be encapsulated, thereby protecting your existing investments. By standardizing the interface between the business logic and the data, you can more easily replace or upgrade database systems. The desktop machines (responsible for presentation) do not need to be modified. The task of replacing a database becomes solely concerned with that action—moving data from one database to another, without affecting the other components of the system.

The control logic can also augment the capabilities of the data storage system, performing additional features, such as searching the information for non-obvious trends (a process called “data-mining”).

The separation of application systems into a number of distinct tiers, and standardizing the interfaces between these tiers, ensures that when you make a modification to one layer, the effect of this change on your entire architecture is localized.

## Summary

In this article, we introduced the Common Object Request Broker Architecture, a developer's tool in implementing applications based on distributed object technology. We also discussed the benefits of an object-oriented approach to network programming over traditional functional approaches, such as the use of RPCs. Finally, we introduced one of the main interests in CORBA technology—enabling the deployment of business applications on a network using a multi-tiered approach.

The next article will discuss the various ORBs available for Linux and how to begin programming with CORBA.

## Resources

### Acronymns and Abbreviations



**Ivan Griffin** is a postgraduate student in the ECE department at the University of Limerick, Ireland. His interests include operating systems, broadband networks and multimedia. He maintains a page of Linux WWW resources at [oak.ece.ul.ie/~griffini/linux.html](http://oak.ece.ul.ie/~griffini/linux.html), and his e-mail address is [ivan.griffin@ul.ie](mailto:ivan.griffin@ul.ie).

**Mark Donnelly** is also a postgraduate student in the ECE department at the University of Limerick, Ireland. Mark is interested in Aikido, Linux, CORBA, Distributed Agents and Alpha World. His e-mail address is [mark.donnelly@ul.ie](mailto:mark.donnelly@ul.ie).

**Dr. John Nelson** is a senior lecturer in Computer Engineering at the University of Limerick, Ireland. His research interests include telecommunications (mobile/broadband), VLSI design, and software engineering. His e-mail address is [john.nelson@ul.ie](mailto:john.nelson@ul.ie).

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

[Advanced search](#)

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



## Financial Calculation Programs for Linux

**James Shapiro**

Issue #48, April 1998

Mr. Shapiro shows us how to write a program to compute internal rate of return using three programming languages supported by Linux—Perl, C and Java.

The world of finance is rife with mathematical formulas. We are all familiar with the elementary ones like those for compound interest or tallying up the value of a savings account if we add a certain amount each month over a period of time. These kinds of formulas are fairly simple and, in general, they can be handled with nothing more than a calculator. Loan amortizations and their inverse, sinking funds, are a little more difficult, but even these types of financial problems succumb to either a calculator or a simple computer program.

In this article I would like to consider a more difficult problem, usually referred to as finding the internal rate of return (IRR). Stated simply, if we make a number of expenditures at different times and receive a number of payments, also at different times, what is the effective interest rate? Because the times are not constrained to regular intervals, the simple formulas referred to above are inadequate. In fact, amortizations, such as car and home loans, are a special case of IRR where the payments and time intervals are both constant. For example, a standard four-year car loan with fixed monthly payments is really an IRR problem where you receive one lump sum at time zero (the loan value) and pay it back in 48 installments. These installments happen to be equal and happen to occur at equal time intervals, namely monthly.

We will first develop the equation for the IRR, then solve it with three equivalent programs in Perl, C and Java. This way you can compare the same algorithm in three different languages. I have found this an interesting and efficient way to learn computer languages. As an added treat we will handle both the discrete and continuous compounding cases in each program. When we are done, you

will have a very general program (three programs, actually) which can handle almost any financial interest rate problem.

The trick to working with money problems is to understand a concept called "the time value of money". If someone agrees to give you \$100 a year from now, that promise is, for financial purposes, worth less than \$100. And, if the \$100 is promised two years hence, it is worth even less. Conversely, if you agree to give someone \$100 a year from now, you can start with less than that now and let your money earn a year's interest before parting with it. People are willing to pay a premium to be able to have things sooner rather than later, and this idea leads to the concept of interest and the time value of money.

Let us consider, for the moment, only discrete compounding, and let us further restrict the discussion to annual compounding, at say eight percent. Our \$100 invested for one year increases to \$108. Leave it invested for another year and we have \$116.64. So, we can see that the general formula for determining the value **V** of **D** dollars invested at interest rate *i* percent for *t* years is

$$V = D * (1 + i / 100)^t$$

or, in our case,

$$V = 100 * (1.08)^t$$

In general, we multiply by  $(1 + i / 100)^t$  to find values in the future and divide by the same factor to find past values. And this explains why and how the state lotteries can give you the choice of taking your winnings as either a series of checks at future time intervals or as one lump sum now. The lump sum results from reducing each of those checks by the above factor (calculated separately for each time interval) and summing the resulting amounts. The lump sum is, of course, always smaller than the total of all those checks, and that is the amount you sacrifice to have the money now rather than later. We will come back to lotteries later after we develop our programs.

We are now in a position to state the rule that governs financial transactions that take place over a period of time: the sum of all the transactions, corrected to the same (any) time using the IRR must total zero. If you think about this for a minute it makes perfectly good sense because, if the interest rate was zero, all of the correction factors would be one and this rule would state no more than the simple fact that expenditures must equal income. Interest compounds (no pun intended) the problem by introducing those pesky correction factors to account for the increase of money with time.

First, the transaction values have to be signed, with inflows typically positive and outflows negative (although reversing the signs consistently will not affect

the resulting IRR). Let us take an easy example. You deposit \$100 in the bank, withdrawing \$40 one year later and \$70 after two years. Correcting all transactions to the present, the IRR is the value of  $i$  that satisfies the equation:

$$-100 + 40 / (1 + i / 100) + 70 / ((1 + i / 100)^2) = 0$$

I will leave it to the reader to solve this one with the hint to define  $x = (1 + i / 100)$  and solve for  $x$  first, then  $i$ . Congratulations, if you got  $i = 6.02\%$ . You probably solved a quadratic equation, and if so, you may have noticed the reason for the quadratic—three different transactions: now and one and two years hence. These kinds of problems get increasingly hard as the number of time periods increase and that provides the motivation for our program(s).

Before I present an algorithm to address financial problems, let us expand our scope to include not only discrete compounding but continuous compounding as well. This turns out to be very easy. Our factors become exponentials instead of powers, so  $(1 + i / 100)^t$  is replaced by  $\exp(i * t / 100)$ . Money compounded continuously grows faster than money compounded discretely at the same rate and for this reason IRRs for continuous compounding are smaller than IRRs for the same discrete problem. You may wish to try recomputing our \$100 bank account problem for continuous compounding. The IRR for this problem is 5.85%.

Let us agree on a straightforward input format with one dollar, time pair per line with both dollars and times as floating-point numbers for complete generality. For our little bank account problem the input file is:

```
-100 0.0
 40 1.0
 70 2.0
```

Note the negative sign on the investment.

I chose Newton's method to solve this problem, mainly because I know that the powers in the problem and their derivatives are well-behaved functions. More importantly, however, we have a handle on the IRR. Interest rates are usually in the 3% to 20% range, allowing us to choose a starting value that is, for most real world problems, going to ensure convergence.

Initially, I used a straightforward algorithm, applying Newton's method directly to the IRR equation. While I was successful, I noted that for some data sets the number of iterations required was disturbingly large. I plotted the IRR equation against the interest and quickly discovered the problem. Newton's method owes its simplicity to the neglect of all terms beyond linear ones, and thus, it works best when higher-order terms are relatively small. Another way of saying

the same thing is that the closer the function is to a straight line, the quicker the linear Newton's method converges.

My plots revealed significant curvature of the IRR function, so I tried manipulating the equation into a “flatter” form. The trick is to collect the time-corrected terms into separate negative (expenses) and positive (income) groups. If we denote the sums as `pos_sum` and `neg_sum` we have equality:

$$\text{pos\_sum} = \text{neg\_sum}$$

where both sums are positive numbers. Dividing both sides of this equation by `pos_sum` and taking the logarithm of the result we have:

$$\ln(\text{neg\_sum} / \text{pos\_sum}) = 0$$

The logarithm flattens the function and makes for very quick convergence. The slight trade-off of having to calculate the log is more than compensated for by the reduction in the number of iterations. I made the following substitution:

$$\exp(-u) = 1 + i$$

and solved for **u** first, then **i**, which simplifies the calculations even more.

Three listings are available for anonymous download from <ftp://linuxjournal.com/pub/lj/listings/issue48/2545.tgz>: [Listing 1](#) is the Perl program, `irr.pl`; [Listing 2](#) is the same program in C, `irr.c`; and Listing 3, `irr.java`, is the equivalent Java code. Due to space considerations only the Perl code is printed here (see Listing 1).

Let us look at the Perl program first. It is the shortest of the three programs, in part because of Perl's many built-in functions and in part because of Perl's built-in memory management. It displays a simple usage message and exits should the user forget the expected format. There are three constants, one to control the maximum number of iterations before exiting the Newton loop, one to decide if convergence has occurred, and a starting value for the IRR (**u = i = 0.0**).

The data file is read in a loop, using the arrays **@pos\_d**, **@pos\_t**, **@neg\_d** and **@neg\_t** to hold the income and expenses and their respective times. Note that Perl dynamically allocates memory for the arrays as the data is read. The write function performs formatted output using the formats at the end of the program. Before starting the Newton algorithm loop we make sure the input data contains both income and expenses—the algorithm will not converge unless it does.

The **for** loop on the variable **\$iters** is the Newton algorithm, where **\$d\_neg** and **\$d\_pos** are the derivatives with respect to **u** of the numerator and denominator, respectively. The program concludes with a display of the IRR if convergence is attained. Note that scalar variables start with a dollar sign and arrays with an @ sign. Variables and arrays do not have types and can contain strings or numbers with the type being determined by usage.

Note that Perl's **print** function accepts variable names and interpolates the actual value, whereas **printf** works pretty much like its C counterpart. Also, you can put "if" tests after the statements they control.

As a non-trivial example, consider the data set in Listing 2. In this example you spend \$1000 today, get back \$500 in a year, only to spend another \$2000 two years from today, etc., with a total of six transactions spread out over five years. We can run the Perl program with this data set by typing:

```
irr.pl irr.dat
```

This results in a display of the input data followed by the line:

```
IRR = 10.6952% (discrete) = 10.1611% (continuous) after 3 iterations.
```

Note that it takes a higher interest rate when compounding is discrete to give the same return as continuous compounding. You will, of course, get the same results from the other two programs.

The C program is longer than its Perl equivalent—about twice as long. In the C program, I chose to define a structure for the dollar and time pairs. One complication—I had to scan the input data twice: the first time to determine the number of pairs so that I could use **malloc** to allocate memory for the array of structures, the second to actually load the data into memory. In this program I placed the Newton algorithm in a static function called **irr**. This function is of a defined type, **ITERATOR**, and returns either **OK** or **NO\_CONVERGE** depending on whether the algorithm converges. The iterating function looks very much like the equivalent Perl code.

The Java program for IRR is longer yet, partly because of the exception checking, but also because I defined several classes in addition to my public **Irr** class. There are no structures in Java, so I put the input data into its own **Expenses** class. I used the interface **Do\_irr\_returns** to hold the return values as the equivalent to the C enumerations for the same purpose. This organization of the program into classes and interfaces tends to increase the amount of code, but leads to clear and easy-to-read programs. The **MyDouble** and **MyInt** classes serve another purpose, as well.

In Java, native variables are passed by value. There is no concept of passing by address, at least for individual built-in variables. One way around this limitation is to create classes as wrappers and then to pass the classes to methods. The **MyDouble** class is endowed with a constructor, so that we can set an initial value for **du**.

Note that there was no reason to scan the input data twice in this case. The variable **in\_data** of type Vector grows automatically as elements are added—no allocating or reallocating of memory is necessary.

I contacted my state lottery in Colorado and came up with a good IRR problem. If you win the lottery, you have a choice of taking a lump sum equal to 40% of your winnings or 25 payments starting with 1/40 of your winnings followed by checks that increase by 3.7% each year. For example, if you win a million dollars you can walk away with a check for \$400,000 (less federal and state withholding taxes of 28% and 4%, respectively) or receive yearly payments starting with \$25,000, then \$25,925, etc., through the last check for \$59,790.22 (withholding applies to these checks, too). Add up those 25 checks and, indeed, you get your million dollars. (For you purists, or if you just like to count your pennies, you will note that the exact total is \$1,000,066.48. The true increase each year is slightly less than 3.7%.) The question faced by every lottery winner is whether it is better to take the lump sum or the payments over time. From a strictly financial standpoint the question is answered once the IRR is determined. That is, at what interest rate would you have to invest the \$400,000 to be able to write yourself the same 25 checks? The answer will be in my next article.

We have looked at the problem of determining the effective interest rate for a series of expenditures and returns of arbitrary amounts and occurring at arbitrary times. We developed the equation for the internal rate of return and manipulated it into a form that is suitable for application of Newton's method. We looked at equivalent programs in Perl, C and Java. You can use these programs to solve for interest rates and to compare investments for any problem involving transactions over time.

Jim has been programming since 1967 and using Linux for about three years. He uses Perl, Java and C to work on telecommunications and GIS applications in his professional life and number theory and cryptography problems in his other life. He is trying, as always, to make his squash and programming skills commensurate. He can be reached via e-mail at [jnshapi@easthub.mnet.uswest.com](mailto:jnshapi@easthub.mnet.uswest.com).

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

[Advanced search](#)

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

## **LJ Interviews Mr. Eid Eid of Corel Computer**

**Marjorie Richardson**

Issue #48, April 1998

Mr. Eid began work for Corel Corporation in 1989 and eventually rose to the position of Vice President of Technology.

In October of last year, Corel Computer Corporation announced that its newest computer, the Corel Video Network Computer, would come with the Linux operating system installed. Wanting to know more, I asked for and was granted an e-mail interview with Corel's President, Mr. Eid Eid. This interview took place December 1, 1997.

Mr. Eid began work for Corel Corporation in 1989 and eventually rose to the position of Vice President of Technology. As Vice President, Eid was in charge of the development of all Corel products, including CorelDRAW, the Corel WordPerfect Suite and CorelVIDEO. In mid-1997, Mr. Eid was appointed to the presidency of the newly formed Corel Computer Corp., located in downtown Ottawa, Canada.

**Marjorie:** The Linux community is quite excited about your new Video Network Computer. Exactly what is a Video Network Computer? What are some of its features?

**Eid:** I want to try to resolve some of the misinformation about network computers before talking about what we are doing at Corel Computer. The concept of a network computer is very simple; let me explain.

- A network computer should be based on an open software and hardware architecture.
- It should have a lower cost of ownership than a PC.
- It should allow for centralized software distribution and centralized hardware management.
- It should increase network and data security.



Corel Computer has adhered to these concepts and developed a number of features that enhance the image of an NC. In fact, our goal was to dispel the reputation that network computers lack the performance and power of PC products.

The Corel Video Network Computer includes a Digital StrongARM processor that is clocked at 233MHz and delivers 250 MIPS. This is equivalent in horsepower to the Pentium II processor but at the power consumption of a night-light.

The processing power of the machine allows us to deliver high-end multimedia to a networked desktop. We have included a Persistent Cache or hard drive that minimizes network traffic, improves performance and allows users to work off-line should a server go down. We have designed and integrated software for greater performance and consistency. And what amazes us most is the size of the device: it measures about the size of a small hardcover book.

**Marjorie:** What factors influenced you to choose Linux as your operating system of choice?

**Eid:** Corel Computer reviewed many operating systems from most major vendors but selected Linux for some tangible and some intangible reasons. The concrete reasons why Linux has millions of supporters, including Corel Computer, are because it is:

- Highly stable
- Robust
- Multitasking and multithreading
- Small of footprint
- Supported by thousands of developers worldwide

However, Linux offers some benefits that go beyond technology. Primarily, Linux is a highly accessible operating system. Sure it is freely available, and in the case of Linux, this means that the users and developers have access to the best technology as soon as they need it, not when a big software developer gets around to releasing the next upgrade. For example, when Corel Computer decided to use the StrongARM processor, Linux was the only serious OS that had binaries available for this chip.

**Marjorie:** What is your targeted market? Who do you expect will be buying this machine?

**Eid:** In the short term we are marketing the machine in three fundamental ways. First, we are positioning the machine as a corporate and institutional

product that addresses the need for low-cost, high-value, network computing. Customers will have a choice of a Java-based user interface or a Linux-based browser interface. The main applications for such a device will be to access and gather information, communicate and collaborate. We expect that some companies will use the Corel Video Network Computer to replace aging terminals or older PCs, and that other companies will use the network computer to extend computing services to employees who do not currently have access to any.

Secondly, we are leveraging our existing communication experience and offering a compelling solution for LAN-based communications. In response to customer feedback, the device is being offered with a Java-based, integrated communications suite and user interface. Customers will be able to centralize audio, video and data communications into a single networked device.

Finally, and most importantly, we are going to position the device as a Linux workstation for Linux and Java developers. The machine will include a software development kit, documentation, support and discounts on the hardware for developers. We plan to announce the details of our developer's program early in the new year.

**Marjorie:** How much will these computers cost? When will we be able to buy one?

**Eid:** The initial design will cost about \$1,000 US and will ship early in 1998. This product will include a keyboard and mouse, and integrated Java-based software running on the Linux OS. However, what we want to sell is the idea that the device will cost less to operate, manage and maintain than a PC. For example, we are currently evaluating the energy consumption of our NC compared to a PC. Our initial results indicate that the savings on an electric bill could be as much as \$200 per year per network computer. Furthermore, the productivity enhancements achieved by users will all but pay for the initial cost of the machine. For example, users will no longer be responsible for the management of "corporate information" residing on their computer. Because files are stored on a central server, the loss to the company should a machine be accidentally damaged is the cost of a new network computer. However, if a PC is damaged, all the information that is stored on the PC could be lost for good.

The Linux workstation that I mentioned before will ship in the spring of 1998. We are planning to include more memory, a larger hard drive and a faster chip in the Linux workstation. Although we have not set the price of this device, it may be closer to \$1,500 US because of these enhancements.

**Marjorie:** What flavor of Linux will be on the workstations?

**Eid:** As a basis for our development, we used two sources under GPL. A large part of the OS is based on pieces from Red Hat and Russell King, an independent developer who has ported much of the recent Red Hat releases to the ARM chip. Corel Computer has been developing, modifying and porting the two sources to the StrongARM chip and configuring the OS to run the Corel Computer NC peripherals. Corel Computer will follow all the rules of GPL and will publish the source code and diff files when the machine is shipped.

We are committed to returning the development work that we have done to the Linux community.

**Marjorie:** How many of Corel's products support Linux at this time? Will those products not currently supported be ported to Linux soon?

**Eid:** Currently, Corel WordPerfect 7 has been ported to Linux 2.0.25. Corel is continuing to study this market with input from partners like Caldera and input from customers. The feedback to this point has been that Linux users are looking for unique and compelling products, not necessarily product ports from Windows.

What is highly encouraging is that Java running in the Linux environment offers the opportunity to begin to address some of these customer demands. Corel Computer has been spearheading development work on Linux running on the StrongARM chip. This includes championing development on Sun's Java Application Environment for this combination. What this means is that Linux users can begin to develop for, or take advantage of, new Java-based products.

**Marjorie:** Tell us a bit about your Java-based multimedia software.

**Eid:** Corel Computer will continue to use the Java programming language to integrate products onto the Corel Video Network Computer. The first product, codenamed "Cabot", is one part of a much larger strategy being implemented by Corel Corp. Corel is dedicated to being an industry leader in the field of web and Java-based communications and groupware applications. The products that we have announced are being designed to leverage network-centric, thin-client computing.

Besides advancing the Linux OS and developing new tools for Java development, Corel Computer has created a software product that automatically increases the value of the network computer. Cabot is an HTML and Java-based user interface and communications suite for the Corel Video Network Computer. The product includes:

- A Java-based browser

- A PIM
- An HTML editor
- An e-mail client
- A file manager and file browser
- And much more...

Our parent company, Corel, will offer the Corel Video Network Computer and its software as part of a broader corporate solution. Corel is designing new software to complement its complete business, graphics and Internet applications. First, Corel is developing Java technology that will allow existing applications such as Corel WordPerfect or Microsoft Office to run on a server and be accessed via a thin client on any Java Virtual Machine. This product will ship early in 1998. Second, Corel is producing a new line of products that will focus on information management and knowledge gathering and publishing. This product will be modular and lightweight, and will ship in the summer of 1998.

**Marjorie:** Any other exciting projects involving Linux (or not) coming up?

**Eid:** We hope to be able to announce a formal relationship between Corel Computer and Sun Microsystems in the coming weeks. This announcement will pertain to the Java Application Environment running on Linux. We also hope to formalize our relationship with Caldera and announce exciting new projects that are in the works between our two companies.

**Marjorie:** How about a short history of Corel Corp. and Corel Computer? Where you started, where you are now...

**Eid:** The history of Corel is very interesting, but very extensive. There are many highlights from the past 12 years, but I think that the most notable are:

- **1985:** Corel Systems Corp. founded by Dr. Michael Cowpland. The company was producing a PC system for desktop publishing using Ventura and WordPerfect. As the saying goes, Michael liked the products so much he later bought the companies.
- **1989:** A drawing product was designed by Corel engineers and was expected to sell a couple hundred copies. CorelDRAW has sold more than 7 million stand-alone copies, and millions more have been shipped with PCs since its launch.
- **1992:** CorelDRAW 3 is the first software suite ever sold. The package included DRAW, PhotoPaint and a raft of multimedia accessories. It also marked the first product to have mass distribution on CD-ROM.

- **1995:** CorelDRAW 6 is the first major 32-bit application to ship for Windows 95.
- **1996:** Corel buys WordPerfect from Novell. Since that time, Corel has shipped two major upgrades to the product in addition to increasing cross-platform support by offering new versions on DOS, Windows 3.1 and Unix.
- **1997:** Launch of Corel Computer Corp. We are currently 90 employees including students, contractors and full-time employees. We have separate offices in Ottawa but are connected to Corel through desktop video conferencing, direct dial telephone and a shared WAN. Dr. Michael Cowpland is the chairman of the new company.

**Marjorie:** What do you see in the future for Corel and Linux?

**Eid:** It is very exciting to be involved with an active and stimulating community. It is a diverse community with a breadth of experience. I hope that Corel Computer can help champion the operating system and the development language and steward new directions based in Java. I know many of us at Corel Computer look forward to traveling to meet with user and developer groups over the coming months.

**Marjorie:** Thank you very much for your time.



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

[Advanced search](#)

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

[Advanced search](#)

## Helping Netscape Make History

**Eric S. Raymond**

Issue #48, April 1998

Netscape source is now free, who would have thought it? Eric Raymond, that's who. Here are his insights into this momentous event.



One of the strangest and most disorienting feelings to experience is waking up and realizing you're making history. I had that feeling the day of Netscape's bombshell announcement on January 23, 1998. They announced they would be releasing the source for Navigator 5.0. I think the Linux community as a whole is having that same experience now.

Netscape's Chief Technology Officer told me, "On behalf of everyone at Netscape, I want to thank you for helping us get to this point in the first place. Your thinking and writings ["The Cathedral and the Bazaar", <http://www.earthspace.net/~esr/writings/cathedral-bazaar/>] were fundamental inspirations to our decision." That made me feel pretty good—I think the best part wasn't the praise for me, it was the astonishing victory for the Linux culture that it represents.

My paper only explains the bazaar development model; I didn't create or discover it. The bazaar model was grounded in old-time Unix tradition that has been brought to full fruition by Linus Torvalds and the thousands of Linux volunteers out there who made it work. Netscape's move is a victory for *all* of us—and, perhaps, a harbinger of greater victories to come if we work hard and smart.

Yes, I know Netscape's maneuver was partly one of desperation. Microsoft's Internet Explorer has been winning the marketing war. Netscape's stock price

had been dropping. Netscape had to change the rules to stay in the game. But forced or not, the change matters a lot—and Wall Street knows it, because Netscape's stock has since been rising.

Netscape people chose *our* rules because they believe in them. Ten days after the announcement, I flew to Silicon Valley at Netscape's invitation for a day-long strategy meeting at which we tried to hammer out solutions to the problems the company will face in making the bazaar model work. One thing I heard over and over from their people was a huge sense of relief—almost gratitude that market conditions had gotten so bad that they could justify doing what they wanted to do anyway.

Within hours after the strategy session, I was briefing a standing-room-only crowd of over 200 at the Silicon Valley Linux Users Group meeting. I told them that within the next eight weeks, Netscape would be posting its proposed source license on the Web for public comment from the hacker community.

Also, I explained why I flew out to help Netscape without charging them a dime—because, like it or not, Netscape has put *our* prestige on the line, too. If Netscape screws up its implementation, the free-software model will be so thoroughly discredited that it could be more than a decade before corporate America is willing to try it again. We've all got to be Netscape's allies now, because for our sake they *must* win.

Before and during my trip to the Valley, I got together with a good many of the Linux community's tribal elders, including Linus Torvalds, John “maddog” Hall, Bruce Perens, Larry Augustin, Dan Quinlan, Phil Hughes and others. We're all acutely conscious that it's crunch time—the long-awaited breakout of free software into the commercial world is happening, it's happening *now*, and the decisions we make in the next few months will be absolutely critical.

One conclusion we've already reached is that the term “free software” is itself a problem. The problem with it is twofold. First, it's confusing; the term “free” is very ambiguous (something the Free Software Foundation's propaganda has to wrestle with constantly). Does “free” mean “no money charged”? Does it mean “free to be modified by anyone”? Or something else?

Second, the term makes a lot of corporate types nervous. While that does not intrinsically bother me in the least, we now see an opportunity to convert business people rather than thumbing our noses at them. Netscape's announcement and the buzz in *Wired*, *Wall Street Journal* and elsewhere means we have a window of opportunity. If we don't exploit it, the window will close—but if we do, and do it right, we can make serious gains in the mainstream

business world. A better label will position us to do that without compromising our ideals and commitment to technical excellence.

We did a lot of brainstorming about this issue and came up with a happy result. We suggest that everywhere we have previously talked about “free software”, we now say “open source”—open-source software—the open-source model, the open-source culture, the Debian Open Source Guidelines. Eventually, creating an open-source branding program for commercial vendors who want to play.

We should publicly explain the reason for the change. Linus has been saying in his “World Domination 101” speech that the open-source culture needs to make a serious effort to take the desktop and engage the corporate mainstream. Of course he's right—and this re-labeling, as Linus agrees, is part of the process. It says we're willing to work with and co-opt the market for our own purposes, rather than remaining stuck in a marginal, adversarial position.

Bruce Perens of Debian has volunteered to register “open source” as a trademark and hold it in trust through Software in the Public Interest. Richard Stallman has agreed to use the term, provided he can be convinced that the new “Open Source” version of the Debian Free Software Guidelines won't weaken them.

Over the next several months I'm going to be seriously pitching the open-source concept to corporate America. I'm going to be talking about the three business models I see for making money from open sources:

1. The Cygnus/Red Hat/BSDI model: give away source, sell service and support.
2. The Netscape model: use open-source software as a loss leader and market positioner for commercial software.
3. The hardware-store model: outsource development of the device drivers and interface tools for your widget to a bazaar effort centered on your web site and led (but not monopolized) by in-house programmers.

I'm also going to be invoking the idea of “peer review” a lot. Open-source software is peer-reviewed software, and thus more reliable and respectable than that icky closed stuff.

In the near future, I think our best tactic is to go after companies that fit the “hardware-store” model. For those outfits, software development is strictly a cost rather than a profit center or part of their corporate identity. If we can show them a way to get much better software and better market position for minimum effort, they'll have every reason to go for it.



A couple of easy wins in that arena will help Netscape and other early pioneers feel less lonely and maybe even convince Wall Street that it's seeing a trend. This will position us to go after the big guys—Sun, SGI, Apple and Novell.

Every CEO and CTO I talk with is going to hear the same thing. First, I'll ask if they have any strategic problem bigger than "Microsoft is going to crush me within five years." Then I'll point out that nobody has ever beaten Bill Gates playing by Bill Gates' rules. Then I'll tell them, "The only way to survive is to change the rules—and I'm here to show you how."

I hope everybody reading this will help. You are the people who got open-source software this far; I'm just your tribune, a sort of accidental ambassador to the suits. Frankly, I'd rather be hacking; however, this is a job that needs doing that I'm fairly well equipped for, so I won't shirk it. If you want to help, and you work for a company you think might be a good target, here are some things you can do:

1. Point your boss to <http://www.earthspace.net/~esr/writings/cathedral-bazaar> and ask him to do the same with his boss. Hey, it's worked so far!
2. If you can get the ear of your Chief Executive Officer or Chief Technology Officer, e-mail and tell me. Then work with me to get me in the door, so I can make the open-source pitch for us.
3. If you work for the trade press, talk to me. The more positive publicity we get, the better chance we have of creating a positive climate of opinion that will make it easy for CEOs to say "yes".

I don't want to be the only ambassador, but the ambassadors at least all have to be telling the same story. Once the effort starts taking clearer shape, there will be a mailing list and a web page and other resources aimed at helping open-source advocates reach out to corporate America during this critical time.

We techies have been trying to win for Linux/Unix and open standards/open source by quiet pressure from below for twenty years, and for the most part it hasn't worked. Now, after Netscape, we've got another chance—to break in from above. Let's do it together!

**Eric S. Raymond** 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/>. Eric can be reached at [esr@thyrsus.com](mailto:esr@thyrsus.com).

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

[Advanced search](#)

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

[Advanced search](#)

## Visual SlickEdit: A Commercial Editor for Programmers

Larry Ayers

Issue #48, April 1998

This editor is cross-platform, with versions available for all three Windows variants, OS/2 and a variety of Unix platforms.



- Manufacturer: MicroEdge, Inc.
- E-mail: [sales@slickedit.com](mailto:sales@slickedit.com)
- URL: <http://www.slickedit.com/>
- Price: \$195 US
- Reviewer: Larry Ayers

There is no shortage of high-quality free editors for Linux. A commercial editor would have to be quite powerful and easy to use to attract prospective customers in the Linux community, where the tradition of using freely-available software is deeply rooted. Visual SlickEdit, developed and marketed by MicroEdge, fits these two criteria. This editor is cross-platform, with versions available for all three Windows variants, OS/2 and a variety of Unix platforms. Recently MicroEdge released version 3.0 of SlickEdit, which brings several new enhancements to an already powerful editor. The closest free equivalents to SlickEdit are GNU Emacs and XEmacs, so in this review I will use these two editors as benchmarks for comparison.

In contrast to the two Emacs variants, SlickEdit is strictly an X application, although it originated as a character-mode editor. The character-mode console version is still available but is marketed separately.

### **Installation**

Visual SlickEdit comes on five floppy disks. The installation is initiated by directly unpacking (using tar) the first disk's contents into a temporary directory, setting execute permission on an installation binary, then running the binary. From this point questions concerning preferred paths are asked, and eventually the option of whether or not to install the uncompiled macro files is offered. These \*.e files aren't necessary for normal editor operation but can be installed if a user wishes to modify them. Not installing them will save some disk space.

I consider the choice of installation directories to be a very desirable feature, as each user will most likely have a different partition or directory with enough space. A full installation occupies about 18MB. Network installations are possible with each user sharing a common executable but running the program with personal initialization files and home directories.

### **Documentation**

The 250-page printed manual supplied with the software is well written and complete, though the well-indexed on-line help system (see Figure 1) offers more than enough guidance while learning to effectively use the editor. About a third of the printed manual is a reference guide to the Slick-C language, which is invaluable for advanced users needing to create new macros and routines.

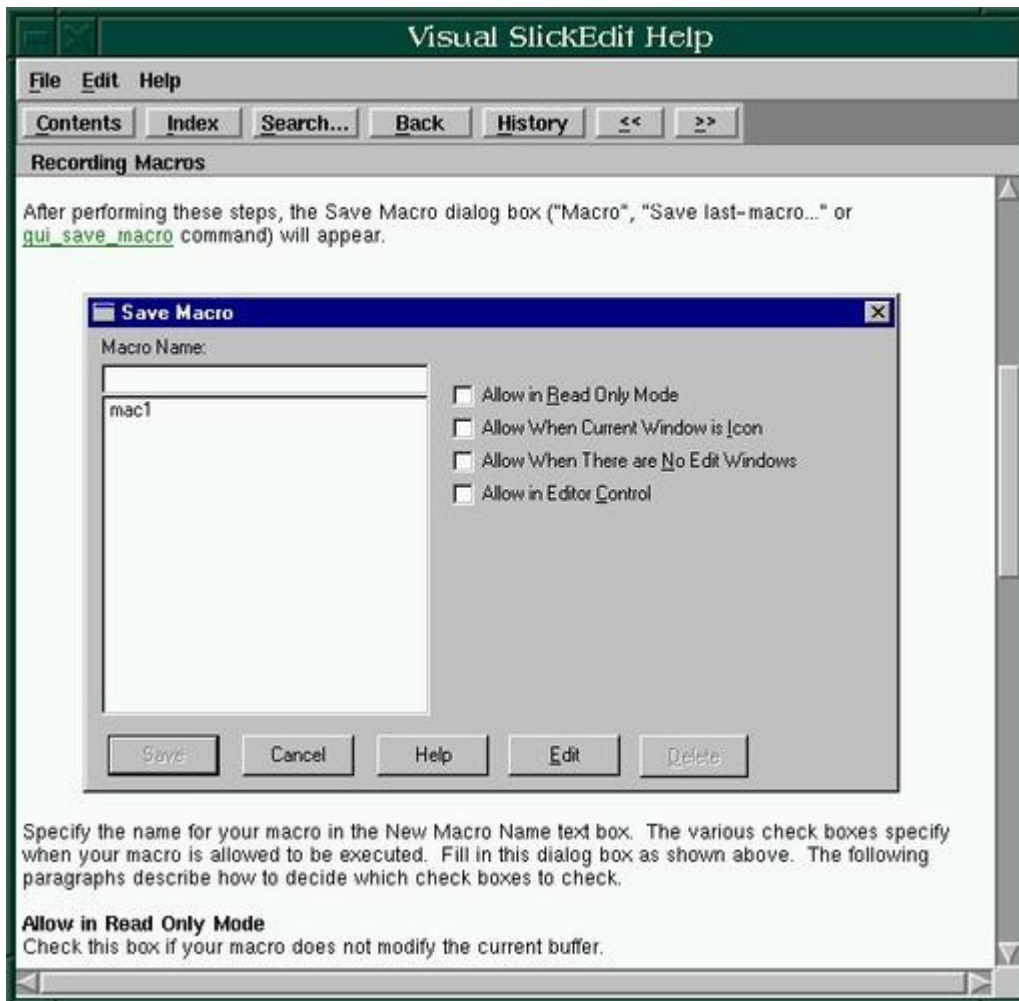


Figure 1. Help Window

## Features

The MicroEdge developers have written a new macro programming language for this editor called Slick-C, which is derived from C++. The syntax is very close to that of C++, but the language has built-in features which are useful for building graphical user interfaces. Slick-C is analogous to Emacs LISP in that many of the routines and functions of the editor are implemented as compiled macros which are called as needed by the main executable.

The developers of Slick-C have attempted to address several shortcomings of C++, such as the lack of built-in types. Boolean, string and container types can be useful when developing editing functions, and Slick-C is designed to work with such types. As with any application-specific programming or macro language, the disadvantage is the necessity of learning yet another language if extensive customization is desired. Slick-C is close enough to C++ that the learning curve for C or C++ programmers shouldn't be too steep, but it's still there.

Just as the Emacs LISP \*.el files are ASCII text compiled for efficiency into \*.elc files, the Slick-C \*.e files can be compiled into binary \*.ex files. The advantage of a separate macro language is that new routines and functions can be written without the need of a new executable, and macro routines tend to be easier to write and debug than an entire editor with the new functions included.

Clipboard Inheritance is an interesting feature of the language, allowing new code to be added to existing controls without affecting the original. That is, parts of existing dialog boxes can be copied to the clipboard and pasted elsewhere; if new code is added to the original dialog box, it is inherited by its offspring.

A feature which programmers will find useful is called SmartPaste. This is an intelligent, syntax-aware version of the standard editing function. If a block of code is cut or copied and then pasted into another source file, the indentation of the pasted block will automatically match that of the surrounding code. I'm unaware of an equivalent Emacs function, but evidently there hasn't been a strong demand or (I surmise) someone would probably have written one by now.

SlickEdit uses its own versions of several Unix/Linux utilities such as **diff**, **find** and **grep**, among others. This feature would probably be more desirable for users with the Windows or OS/2 versions of the editor. The only advantage under Linux is that the interface is via dialog boxes, and the utilities are optimized for use with the editor. In version 3.0 the traditional Unix regular expression syntax is supported.

Support for a variety of version-control systems is included, and unlike the Emacs editors, various commercial version-control programs are supported, which is an advantage for programmers working in a commercial environment.

The command-line window can be used to enter shell commands as well as native editor commands. The command can be executed in a separate terminal window, such as an xterm, if the output needs to be monitored or if interaction is required. Some basic shell conveniences, such as aliasing and command history, are built into this interface.

Like the various Emacs editors (and few others) SlickEdit has an incremental search command, finding the next match of a search word, phrase or regular expression as it is typed. In most cases, the search finds the desired pattern before it has been completely typed in, speeding up the process.

Most modern Integrated Development Environments, such as Borland's, allow all of the files involved in a programming project to be accessed as a unit.

SlickEdit includes this feature. A SlickEdit project includes the working directory, compile/make command lines and a list of pertinent files. Tagging (like Emacs' Etags program) is supported, enabling quick navigation through function definitions and other programming constructs.

The syntax-highlighting support is well done and easy to configure, once again with dialog boxes. Several color schemes are included with the default installation, and they provide good starting points for further modification.

SlickEdit has a number of nicely implemented formatting features, such as the ability to indent or un-indent selected lines with the **tab** key.

Although this editor has its own file-manager window, the list of most recently accessed files found in the File drop-down menu is very convenient. A few other editors, such as Nedit and FTE, save a similar history of files opened in previous sessions, and it's a time-saving feature if there are certain files which are repeatedly loaded.

Some of the other useful features are:

- three-way file merging
- hex editing
- built in spell-checker and dictionary (not quite as extensive as **ispell**)
- spell-checking can be restricted to only the comments and strings in source files
- normally invisible characters such as tabs, spaces and end-of-line characters can be made visible
- multiple clipboards with dialog interface
- preconfigured support for many languages
- C++ and Java code beautifiers
- syntax expansion for several languages
- file size up to one gigabyte
- API Apprentice (help for APIs)
- search and replace in multiple files, buffers or directories
- multiple-level code folding, allowing comments, functions, etc. to be hidden and revealed at will
- easily redefinable **Enter**, **Backspace**, **Delete**, **Home** and **Tab** keys
- a variety of windowing treatments: tiling (horizontally or vertically), cascading, or a basic MDI interface

## Customization

SlickEdit is very customizable, rivaling Emacs and XEmacs in the scope and variety of ways the editor can be molded to a user's preferences. The main difference between the Emacs and the SlickEdit approaches is in the user interfaces. SlickEdit makes extensive use of dialog boxes, windows which allow a user to type in preferences or define macros and keybindings. The Emacs approach tends towards editing the various init files, such as `~/.emacs` or `~/.xemacs-options`, adding entries which load various packages, functions and keybindings. If it weren't for the large corpus of Emacs packages available, most provided with explicit instructions for installation and usage, the SlickEdit method would have a definite edge.

Open a source code file in SlickEdit and (with the default set-up) a variety of windows will appear along with the expected basic view of the file. A narrow vertical project window contains links to every procedure or class (in a C++ or Java file) and clicking on one will scroll the editing window to the corresponding location in the file. This window can also display a list of source files in a project; a file can be loaded into the editing window, then the project window can be toggled so that it will display all procedures or classes in the file. In effect this updated and graphical approach to the "tags" file functionality is familiar to Emacs and vi users. Another available window will display the output of child processes such as compilation.

In keeping with SlickEdit's tendency towards maximum configurability, all of these child windows can be easily customized or disabled via a menu-accessed dialog box. Entirely new toolbars and subwindows can be created and saved within this dialog box, and the visibility of any of the child windows or toolbars can be toggled to suit the user's current needs. These toolbars can be either free-floating or "docked" to a specified edge of the main window (see Figure 2).



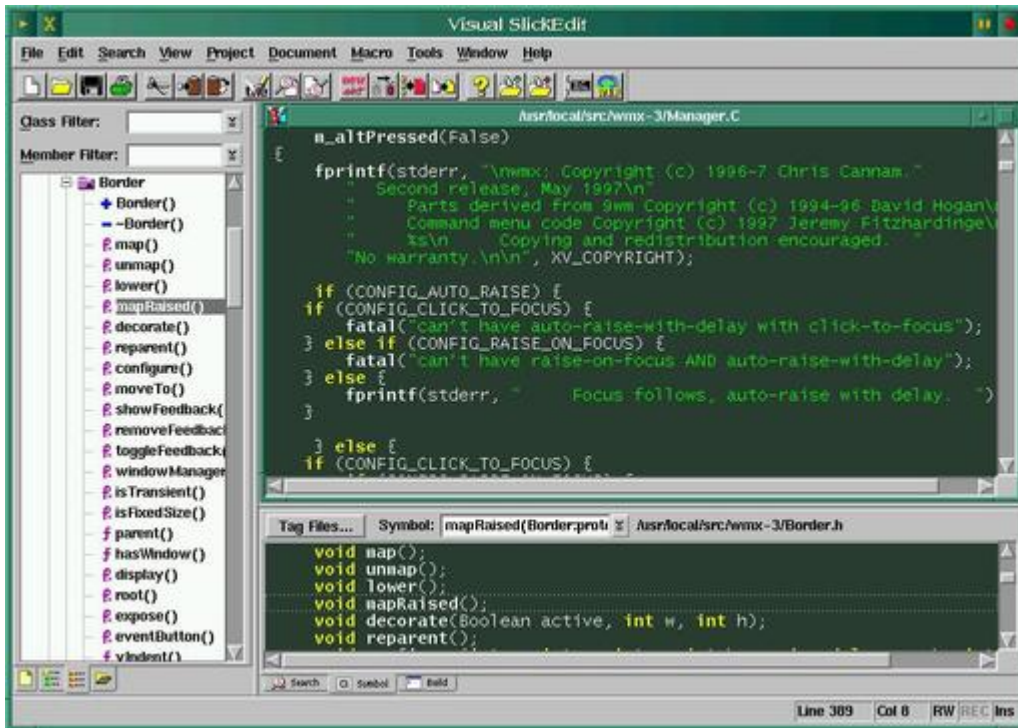


Figure 2. SlickEdit Main Window

The drawback I've found with customization of SlickEdit is that there aren't as many configuration packages available, so even though you have nice graphical windows in which to enter your information, in most cases you have to start from scratch. As an example, if you want to have a useful set of hot-keys for the various HTML tags, in Emacs you have a choice of several key-binding packages, such as HTML-Helper-Mode or HM-HTML-Menus. SlickEdit doesn't come with such bindings, so you have to set them up yourself. Version 3.0 does have a useful new feature for HTML-editing: the spell-checker is aware of HTML tags and will ignore them.

The editor can easily be set up to use any of several keyboard emulation packages, such as Emacs, VI or Brief. The default is SlickEdit's own CUA/Windows keymapping. Of course, any and all keys can be remapped to the user's preference. Keyboard macros are easily and conveniently set up via a dialog box and can be saved for subsequent reuse. A macro is automatically translated into the Slick-C language and compiled when it is saved, speeding its subsequent execution.

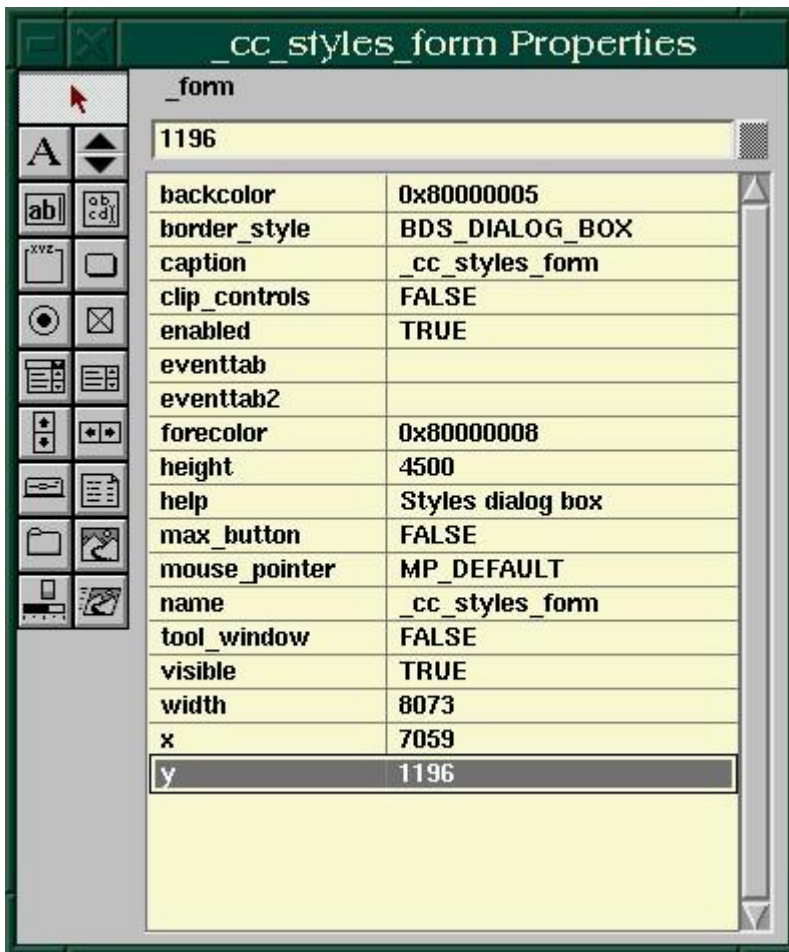


Figure 3. Creation of Dialog Box Screen

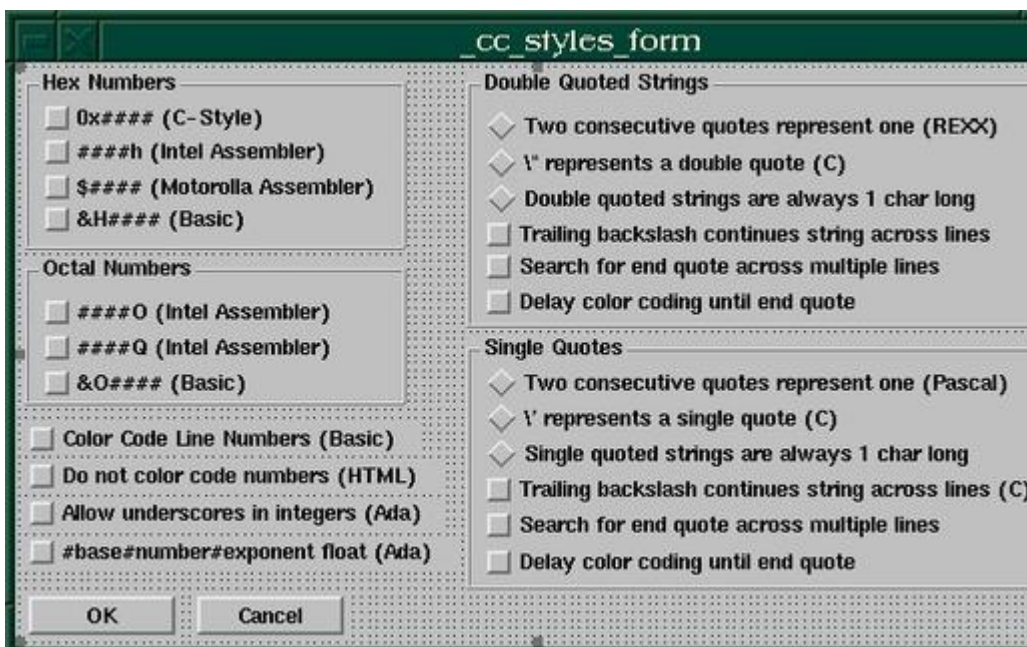


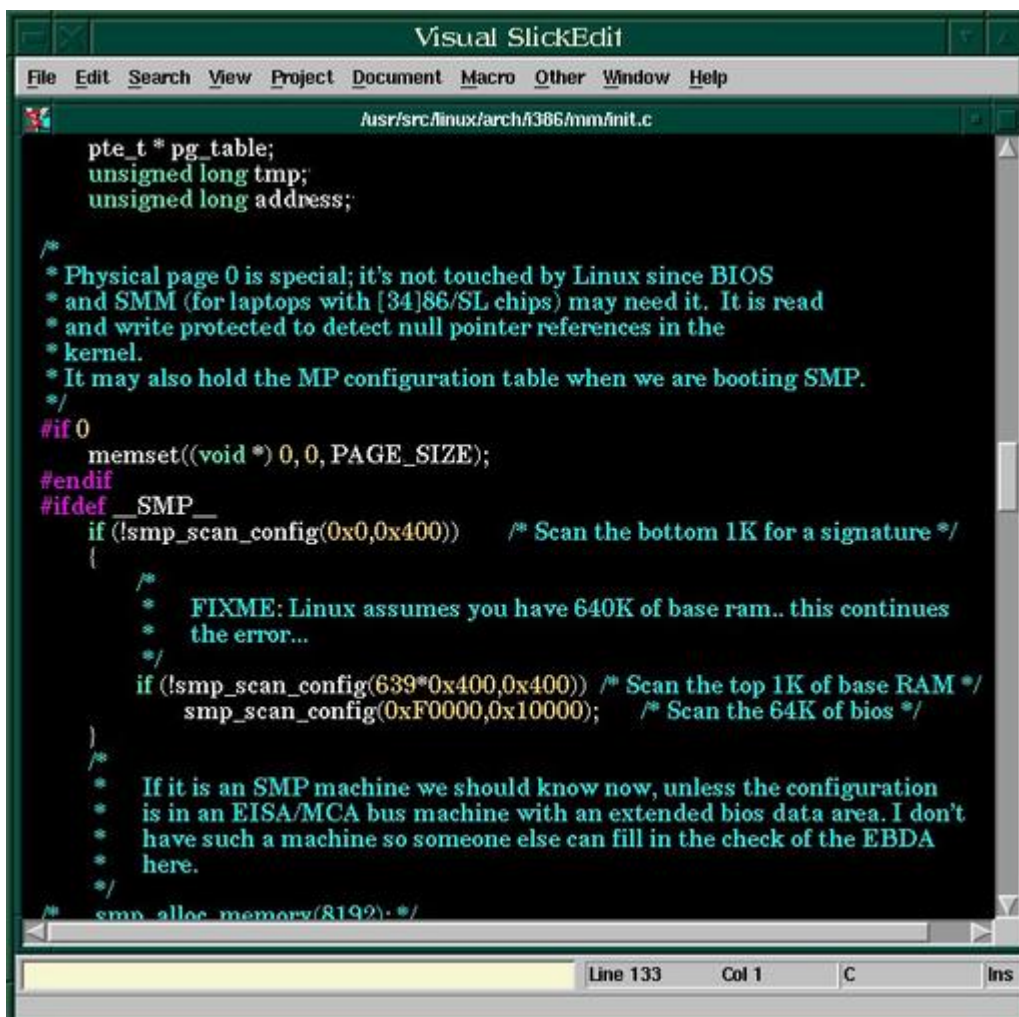
Figure 4. Editing of Dialog Box Screen

One of the most impressive features of this editor is the dialog box editing and creation facility (see Figures 3 and 4). The interface is reminiscent of Visual Basic and allows any of the existing dialog boxes to be changed to suit your

preferences. On first exposure, it seemed that this was customization carried to an extreme. Is it really necessary to be able to shift a button or menu in a dialog box to another location in the box or change its font and color? After further experimentation the real value of this feature becomes apparent. The dialog boxes are written in the Slick-C language, and the dialog editor is in effect a visual development environment for the language, specialized for dialog-box creation. Any of the supplied dialogs can easily be edited into an interface for any editing function a user might desire.

This feature could be a time-saver in a development firm employing many programmers; dialog-boxes could be created specifically for the local language and type of project, then propagated to the individual developers' machines.

Any of these same Slick-C commands could be typed in the command-line entry field at the bottom of the editor window, which illustrates the hybrid nature of this editor. The concept of performing tasks via the mouse and dialogs is reminiscent of the Windows and Macintosh interfaces, both of which evolved as alternatives to the spare command-line environment of Unix or DOS. SlickEdit has a foot in each of these worlds, allowing a user either to intermingle the two approaches or ignore one while using the other.



```
Visual SlickEdit
File Edit Search View Project Document Macro Other Window Help
/usr/src/linux/arch/i386/mm/init.c
pte_t * pg_table;
unsigned long tmp;
unsigned long address;

/*
 * Physical page 0 is special; it's not touched by Linux since BIOS
 * and SMM (for laptops with [34]86/SL chips) may need it. It is read
 * and write protected to detect null pointer references in the
 * kernel.
 * It may also hold the MP configuration table when we are booting SMP.
 */
#if 0
    memset((void *) 0, 0, PAGE_SIZE);
#endif
#ifdef __SMP__
    if (!smp_scan_config(0x0,0x400)) /* Scan the bottom 1K for a signature */
    {
        /*
         * FIXME: Linux assumes you have 640K of base ram.. this continues
         * the error...
         */
        if (!smp_scan_config(639*0x400,0x400)) /* Scan the top 1K of base RAM */
            smp_scan_config(0xF0000,0x10000); /* Scan the 64K of bios */
    }
    /*
     * If it is an SMP machine we should know now, unless the configuration
     * is in an EISA/MCA bus machine with an extended bios data area. I don't
     * have such a machine so someone else can fill in the check of the EBDA
     * here.
     */
    smp_alloc_memory(8192);
#endif

Line 133 Col 1 C Ins
```

Figure 5. Program Editing Window

### Performance

I found SlickEdit to be a very responsive editor on my Pentium 120 machine with 32MB of memory. Its memory footprint and loading time are comparable to those of GNU Emacs, and about half of what XEmacs requires. Search and replace operations are quick, and file operations (loading, switching from one buffer to another, etc.) seem to be as fast as can be expected.

### Availability and Support

MicroEdge maintains a web site for SlickEdit users and prospective customers at <http://www.slickedit.com/>. Patches for registered users are available there, as well as macros and tips from other users and the MicroEdge developers.

At around \$200 US for the Linux version, the price will probably be too steep for many Linux users, especially with so many high-quality free editors available. The program is intended for professional developers, in particular those working in a cross-platform shop. A programmer routinely collaborating on large projects with colleagues using Windows, OS/2 or another Unix variant would find this editor very useful, as project files and macros are compatible across platforms.



**Larry Ayers** lives on a small farm in northern Missouri, where he raises sheep, shiitake and shell-scripts. His e-mail address is [layers@marktwain.net](mailto:layers@marktwain.net).

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

[Advanced search](#)

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



[Advanced search](#)

## WordPerfect 7 for Linux

**Michael Scott Shappe**

Issue #48, April 1998

WordPerfect 7 for Linux, I'm pleased to say, is very good—certainly good enough for me to finally abandon Windows as a word processing environment.

- Manufacturer: Corel Corporation (ported by Software Development Corporation)
- E-mail: [custserv2@corel.com](mailto:custserv2@corel.com) ([info@sdcorp.com](mailto:info@sdcorp.com))
- URL: <http://www.corel.com/> (<http://www.sdcorp.com/>)
- Price: \$199 US (add \$19.95 US for CD\_ROM)
- Reviewer: Michael Scott Shappe

Since the push for commercial applications under Linux began, one of the flagship applications has been the “grand old man” of PC word processing, WordPerfect. Originally, Caldera had ported WordPerfect 6 to their commercial Linux distribution. This port apparently did well enough to convince Corel, the current owners of the product line, that there was a market worth their efforts. They farmed out the port to Software Development Corporation, but Corel markets the results, WordPerfect 7 for Linux, directly as well as through SDC's web site.

WordPerfect 7 for Linux, I'm pleased to say, is very good—certainly good enough for me to finally abandon Windows as a word processing environment. It has few problems (no show-stoppers) and a reasonable price tag.

### Getting the Program

There are two ways to obtain the program: downloading off the Net or ordering a CD. In both cases, you get a completely functional program with all its features but only a 15-day evaluation license. I thought this was a nice compromise between the commercial nature of the program and the freely available software model the Linux community is used to. This evaluation

version is entirely free if you get it off the Net; you can also pay about \$30 US to have a CD shipped to you. I tend to prefer to have the disk around, so I tested both the CD and the network installation.

If you decide to keep WordPerfect, a single-user license is \$199 US—about \$100 US less than for current versions of WordPerfect on other platforms, including other Unix platforms. This pricing also seems to be aimed at a compromise between the two mindsets. While not willing to give the license away, Corel is interested enough in the Linux community to offer an attractively low price.

## Installation

Whether you grab it off the Net or opt for the CD, installation is fairly easy, as you would expect these days. If you choose to download, you will need to do a little more work. You will also need a lot more disk space to store both the distribution and the installation, but the former can be erased or archived as soon as you're satisfied with your setup. You will need between 50 and 150MB of disk space available for the final installation, depending on the options you select.

## Installing from the Net

Getting the necessary files off the Internet is just a matter of linking to the right web site. In this case, you want to go to <http://www.sdcorp.com/> or <http://www.corel.com/>. It's a little faster to navigate SDC's web site. SDC's main page has a link for "WordPerfect for Linux" that leads to the registration and download pages for the evaluation copy. It also leads to pages that allow you to actually purchase a license or purchase the CD from one of their resellers.

Once you have the files you need (and it will take a while, even on a fast link), make a directory and unpack the files (**tar xzf** should do the trick for each of them). Note that, unlike most GNU packages and other software you get off the Net these days, these tar files do not create a directory for you, so be careful.

When done, you will have a number of cryptically named files and a completely obvious "Runme" file. Execute "Runme" as root, and it will walk you through the installation with either a decently thought-out graphical interface or a bearable character interface, depending on whether or not your X server is running at the time and whether root currently has permission to display. If you're running XDM with its default configuration, and log in as a regular user and use **su** or **sudo** to get root access, root will not have permission to display by default. You can either use the command:

```
xhost `cat hostname`
```

before invoking su to add the entire local host, or use this command:

```
xauth merge ~yourusername/.Xauthority
```

as root to give root access to the necessary keys. Both are kind of cheating, and you'll probably want to undo them (by using **xhost -** or removing /root/.Xauthority) when you're finished. Both methods are almost identical in what they let you do.

When the installation is complete, you will find that all of those cryptically named files now have .bk extensions added. The files themselves haven't changed at all, but if you find you need to reinstall or wish to change installation options, you will need to move these files back to their original names before running "Runme" again.

### Installing from the CD

Installation from the CD is much simpler. Mount the CD the way you would any other CD-ROM, change to the mount directory and execute install.wp as root. This is basically the identical program to "Runme" above, except that the distribution is formatted differently. The resulting installation will be identical if the same options are selected.

### Running the Program

First of all, even if you installed via the character-based interface, make sure your X server is running when you go to run the program. WordPerfect for Linux absolutely requires the X Window System to run. Most other WordPerfect for Unix distributions include a character-based version (ah, function keys), but the Linux distribution does not at this time. It hasn't been ruled out for the future; it's just not currently included. I gather they want to see how the "main" product does first, and then see what demand there is for the character-based version. I, for one, wouldn't mind having the option, but I don't fault them for their caution.

Once the program is installed, make sure you have its binary directory in your path (if you installed in /usr/wp, you want /usr/wp/wpbin/ in your path) and run **xwp**. You should see a splash screen within a second, a small window with two menu items shortly after that, and a large new document window a second or so later. To add a comparative note here, I found WordPerfect's start-up time to be remarkably fast compared to StarOffice.

### The Good

At this point, if you've ever used any of the modern Windows or Macintosh word processors, you should be right at home. WordPerfect for Linux has all of

the features of WordPerfect 7 for other platforms, including simple drawing and charting modules, spelling and grammar checking, auto-correct and spell-as-you-go highlighting, tables, mail merge, outlining, lists and style sheets. This is a complete implementation of the feature set people have come to expect in a word processor.

For all of that, somehow, the program manages to not be a complete memory hog. While typing this review, I found that the program had a resident set size of only 5.5MB, while a small daemon that runs concurrently (**wpexc**) took up a mere 416K—not exactly svelte, but not bloated by today's standards either. These numbers basically mean that I can run the program on my small 16MB laptop without swapping. To get comparative again: while I do not remember the exact resident set size of StarOffice, I do remember quite clearly that it swapped like crazy when handling anything more than just typing.

The secret seems in part to be that SDC has taken advantage of the Unix model and split a lot of the “under-the-hood” functionality out into separate programs. The grammar and spell checkers, for example, are each a separate binary, and so are many of the file conversion filters. Some of these can even be run independently of WordPerfect from the command line. The result is that most of the time all that needs to be in memory is the basic interface to handle commands and typing.

### **Ease of Use**

If you need to get started quickly, WordPerfect is a definite winner. Although the default text style is not the best for a document, reformatting is simple enough that you can just start typing and worry about it later.

At any given time, the right mouse button will bring up a handy menu of common formatting commands. In addition, the paragraph the mouse pointer currently points to will always have a small button next to it. Pressing this button brings up a small window with just the paragraph formatting commands.

If you have Spell-As-You-Go (the name of the feature that highlights misspelled words) turned on, the menu that appears when you right-click over a highlighted word changes to provide suggestions for replacements. You also are given the choice to add the word to the dictionary right then or to bring up the full spell checker.

### **Word Processing with Style**

Even the simplest word processors these days have style sheets, although they all handle them a bit differently. WordPerfect's handling falls into the middle



range, in my opinion, for ease of use. What keeps it from a better rating is the sheer complexity of the underlying style-sheet model, and an interface that does not seem to have been completely thought out from the user's point of view. There are several different kinds of styles: character, paragraph and document, and several different "locations" where style sheets can reside: single document, personal style-sheet library, system library. Users have to pay very careful attention to these details, particularly if they want a single style sheet to be used in several documents (for example, the chapters of a book). In some cases, however, it's not entirely obvious how to set the options you need.

The other complication is an historical artifact of WordPerfect itself. WordPerfect is still code driven; that is, it performs stylistic changes by transparently marking up text with bracketing tags, much like HTML. It is, in fact, still possible to tell WordPerfect to show you these codes in your document—WordPerfect old-timers will be familiar with the "Reveal Codes" feature.

The window that lets you actually manipulate a style either shows you these codes to tell you which features you've selected or shows you nothing at all. To a user more familiar with Microsoft Word or StarOffice, this can be quite confusing; the more so since the user really doesn't need to know about the codes. It would have been just as useful to list which features had been set up for a given style, leaving "Reveal Codes" as an option for the power user.

To WordPerfect's credit, there is a "QuickStyle" option that simply creates a style based on the current formatting under the insertion point. So, a user could type a document, change the style of one paragraph using the normal formatting commands until satisfied, then invoke the Styles feature and select QuickStyle. After that, a Select All followed by selecting the new Style would apply the style throughout the document.

### **The Need for Speed**

I'd like to be able to say that WordPerfect is a complete speed demon, but it's not quite. Many operations are fast, to be certain (spell-check, for example, is quite speedy), but some others seem slow, at least on my laptop (a Pentium 120, 16MB, Xfree86 and later Accelerated X LX 4.1). Just typing seemed to lag under Xfree86, although it's much improved under Accelerated X. Even under that server, however, typing is sometimes sluggish, especially when inserting into the middle of a paragraph. No characters are actually lost, but it's not quite as fast as I would expect.

I found there were certain things I could do to speed it up. Turning off Spell-As-You-Go was a big help. Also, reconfiguring the status bar at the bottom of the

window to not display full position information helped by eliminating a redraw for every character.

I've also found that plugging in my laptop, rather than running off the battery, speeds things up; this is not surprising, since most portables have an automatic "slow down" mode for saving power. This, of course, is not WordPerfect's problem. Even so, word processors have been keeping up with people since long before the Pentium era, so I wouldn't necessarily expect the slight power-saving slow down to be so noticeable.

My speed problems were not entirely consistent, and I did not do a very scientific examination of exactly what was going on when the program became slow.

Even when WordPerfect is feeling sluggish, however, it's still thoroughly usable. In fact, if you are the sort of typist who has been trained not to look at the keyboard or screen while typing but rather at your source material or some other neutral point, you probably won't even notice.

### **Clunkers**

Although I find the program mostly satisfactory, with all the features I might want and quite a few I can take or leave, I have found a few problems that I can only call "clunkers". None of them prevent the successful use of the program as a whole, but they do make certain features difficult or impossible to use, and that, of course, detracts from the overall attractiveness of the product.

The first problem I noticed was that the on-line help system did not seem to work on my portable, although it had worked fine on another system I'd tried it on. It turned out that my locale information was set to "us" rather than C, which is the usual default. The on-line help seems to crash when faced with a locale it doesn't know about. This is arguably a case of user error, but it seems to me that the help program could simply fall back to a reasonable default rather than crashing.

The second problem I had is that the HTML export feature does not seem to work consistently. When I tried exporting this document to HTML, it worked just fine. However, when I begin by importing some (but not all) existing HTML documents, I find that I absolutely cannot export them—the file comes up blank. I've reported this problem, and hopefully it will be tracked down soon.

### **The Last Word**

WordPerfect 7 for Linux is an excellent effort, fairly stable, reasonably fast and contains everything people have come to expect in a word processing program.

It works fine in relatively low memory conditions (a rarity in commercial applications these days) and on relatively low-end hardware. In fact, from the low-end standpoint, the only thing that would make it better would be to include the character-based version, so that low-end users would not have to use the X Window System at all.

While it does have its problems, WordPerfect successfully proves that it is possible to write commercial software for a free operating system and also provides one more excellent weapon in the struggle for freedom from the Microsoft Empire. I'm pleased to say that, with WordPerfect now installed on my Linux partition, I have only two reasons to use That Other Operating System at all: Dramatica (an excellent program for working out the details of a story, which I am trying to convince Screenplay Systems to port to Linux), and Jedi Knight.

Software Development Corporation and Corel: my hat's off to you. Good work.



**Michael Scott Shappe** is a senior programmer for AetherWorks Corporation, a technology startup in St. Paul, MN. He has spent the last 10 years hammering on Unix systems of various stripes and addicting the unsuspecting to the Internet. You can peruse his personal web page at <http://www.publiccom.com/web/mikey/> or send him e-mail at [mshapp19@idt.net](mailto:mshapp19@idt.net).

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

[Advanced search](#)

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

Advanced search

## TeraSpell 97 for Emacs

**Daniel Lazenby**

Issue #48, April 1998

TeraSpell is used to spell check a word, line, region or buffer within Emacs.

- Manufacturer: Teragram Corporation
- E-Mail: [info@teragram.com](mailto:info@teragram.com)
- URL: <http://www.teragram.com/>
- Price: \$9.99 US
- Reviewer: Daniel Lazenby

TeraSpell 97 for Emacs is a shareware version of the Teragram Corporation's Teragram Spell 97 for Windows, Mac and Unix. Naturally there are some differences between the shareware and the full product. TeraSpell for Emacs provides the Emacs user with a flexible multi-language tool for spell checking documents. As a test, I decided to use TeraSpell 97 and Emacs to write this review.

### What Does It Do?

TeraSpell is used to spell check a word, line, region or buffer within Emacs. The evaluation copy contained only an American English spelling dictionary. Several other languages including British English, French and Canadian French, Dutch, German, Italian and Spanish were listed as possible language dictionaries. This software also allows users to define multiple custom dictionaries of their own.

The feature I like most is the phonetic spell checking. Spelling is not one of my fortes. Often I will spell a word the same way I pronounce it. Needless to say, my pronunciation of the word is not always correct, and hence neither is my spelling. TeraSpell did an excellent job of offering suggestions for my often abused spellings.

## How Do You Use It?

Using TeraSpell is as easy as opening the Emacs editor. TeraSpell appears as the left-most item on the Emacs menu. TeraSpell menu options include spell checking a region, buffer or word and spell checking words on the fly. You may also alter several default settings. Preferences that may be adjusted include the default dictionary, the foreground and background color used to highlight misspelled words, the underlining of misspelled words and the placement of the suggested spelling's floating menu.

There are two ways to set individual preferences. One is to start Emacs and make selections from the TeraSpell Menu. The menu method lasts only for the duration of the Emacs session. The other is to customize your `~/.emacs` file which will cause TeraSpell to start with your desired preferences. Changes to the `~/.emacs` file involve appending **setq** variable definitions to the bottom of the `~/.emacs` file.

I configured TeraSpell to display misspelled words in red with a yellow background (see Figure 1). To correct a misspelled word you place the mouse cursor over the word and click the right mouse button. This right click action displays a pop-up menu with some suggested spellings or alternate words (see Figure 2). I noticed an odd cursor behavior when correcting misspelled words. The cursor seems to act as an insertion point when it is within a misspelled word. Instead of replacing the misspelled word, the new word was inserted into the misspelled word.

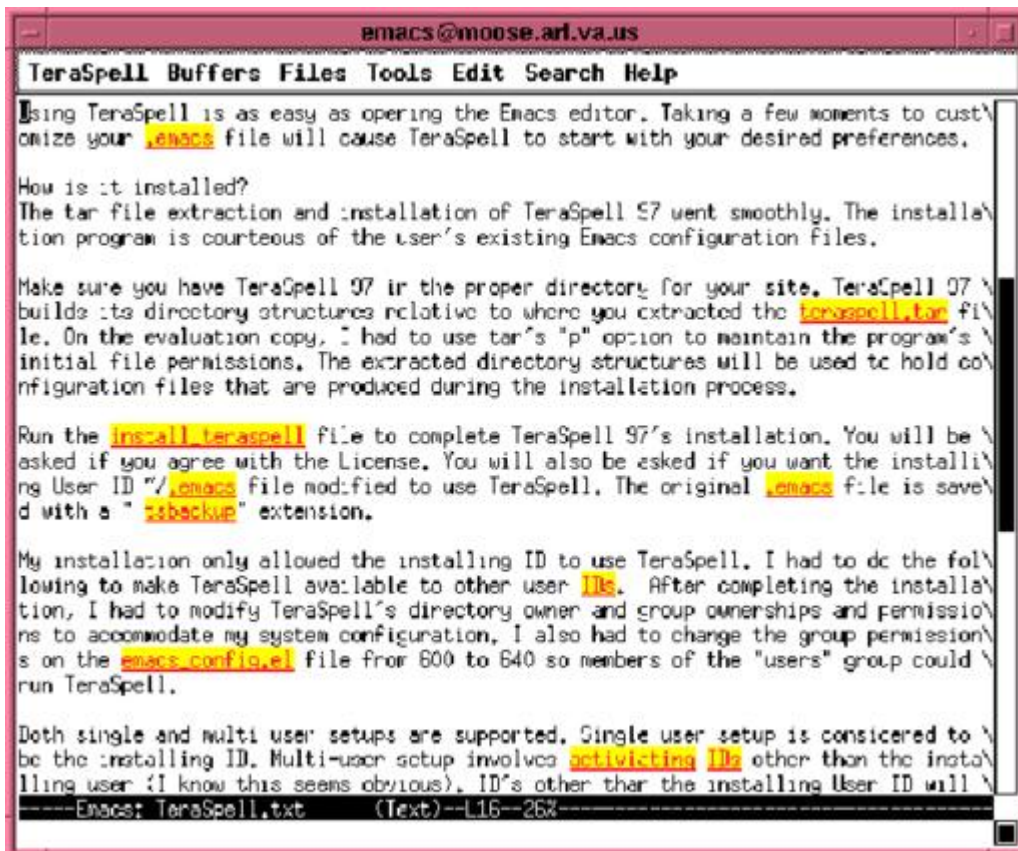


Figure 1. Emacs Screen Shot

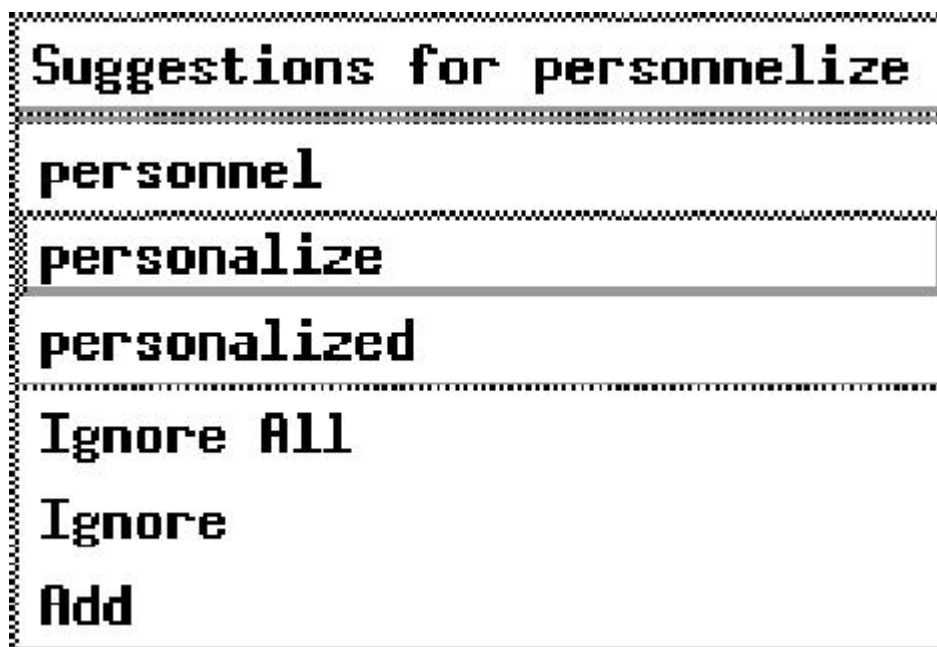


Figure 2. TeraSpell Pop-up Menu

### How is it Installed?

TeraSpell came as a gzipped tar file. The tar file extraction and installation of TeraSpell 97 went smoothly. The installation program was courteous of my existing Emacs configuration file.

Make sure you have TeraSpell 97 in the proper directory for your site. TeraSpell 97 builds its directory structures relative to where you extract the teraspell.tar file. On the evaluation copy, I used tar's **p** option to maintain the program's initial file permissions. The extracted directory structures are used to hold configuration files produced during the installation process.

Next I ran the **install\_teraspell** program to complete TeraSpell 97's installation. I was then asked if I agreed with the License and if I wanted the ~/.emacs file of the installing UID modified to use TeraSpell. The modification is a line that starts TeraSpell. The startup line is appended to the ~/.emacs file and the original ~/.emacs file is saved with a **-tsbackup** extension.

Both single- and multi-user setups are supported. Single-user setup is considered to be the installing UID. Multi-user setup involves activating UIDs other than the installing user (I know this seems obvious). UIDs other than the installing UID will have to run a program called **ts\_adduser** or manually modify their ~/.emacs files to load TeraSpell with Emacs.

My installation only allowed the installing UID to use TeraSpell. To make TeraSpell available to other users, I had to modify TeraSpell's directory group owner and some group permissions to accommodate my system configuration. I also had to change the group permissions on the emacs\_config.el file from 600 to 640, so members of the users' group could run TeraSpell.

Product Requirements Documentation for the reviewed copy of TeraSpell indicated it runs on Linux, HP-UX, IRIX, Sun4 and Sun5. I did not see a minimum required version of Emacs in the documentation. I did notice the installation process verifies that an Emacs release 19.29.1 or newer is installed.

### **Who Makes It?**

The Teragram Corporation provides advanced text processing tools and linguistic resources to manufacturers, developers and researchers for inclusion in their applications. Teragram may be reached at: Teragram Corporation, 236 Huntington Avenue, Boston, MA 02115-4701, 617-369-0100 (voice).

### **Evaluation Platform**

The platform used to review this product was an Intel 486/66 with 20MB RAM and a 16MB swap file. This platform is running the Slackware96 distribution of Linux with a 2.0.30 kernel and XF86Free V3.1.2 with the FVWM window manager. Emacs version 19.31.1 provided the Emacs editor environment.

**Daniel Lazenby** holds a BS in Decision Sciences. Daniel has been working with specialized and general purpose computer systems for more than 20 years. He

first encountered Unix in 1983 and discovered Linux in 1994. Today he provides Engineering Support for a wide range of platforms running Linux, AIX and HP-UX. His e-mail address is [lazenby@ix.netcom.com](mailto:lazenby@ix.netcom.com).

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

[Advanced search](#)

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



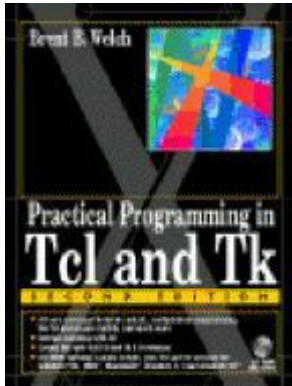
Advanced search

## Practical Programming in Tcl & Tk

**John McLaughlin**

Issue #48, April 1998

If you're curious and want to learn more about Tcl or if you just want to become more comfortable with the language, *Practical Programming in Tcl & Tk* by Brent B. Welch is a book you'll want to read and refer to often.



- Author: Brent Welch
- Publisher: Prentice Hall
- E-mail: [sales@prenhall.com](mailto:sales@prenhall.com)
- URL: <http://www.prenhall.com/>
- Price: \$42 US
- ISBN: 0136168302
- Reviewer: John McLaughlin

If you are a Linux user, you have most likely used a Tcl/Tk application. A large number of the applications under Linux use Tcl/Tk (here after referred to as “Tcl applications”) as the GUI front end. Tcl has become popular under Linux for a variety of reasons: flexibility, ease of use, availability of source code, liberal license policy and the gorgeous look to the Tk widget set. If you're curious and want to learn more about Tcl or if you just want to become more comfortable with the language, *Practical Programming in Tcl & Tk* by Brent B. Welch is a book you'll want to read and refer to often.

A member of the development team for Tcl at Sun, Mr. Welch is perhaps best known to the Linux community for his work developing EXMH, the best mail reader under Linux (small personal bias inserted). In this 2nd revision of his book, he uses 613 pages to lead the user from the fundamental paradigm of Tcl scripts, to using Tk and writing extensions in C.

The book is divided into seven sections covering Tcl Basics, Advanced Tcl, Tk Basics, Tk Widgets, Tk Details, Tcl and C, and Changes. Each chapter begins first with a short paragraph introducing the subject, followed by detailed discussions, tables and a plethora of examples. A helpful feature of this book is the "Hot Tips" marked in the margins, these indicate something particularly useful, tricky or non-obvious.

The section on the Tcl language pleased me immensely; often books fly through the fundamentals of Tcl in order to get to the more "glamorous" Tk section. Tcl is powerful and rich, but can sometimes be a confusing language especially if you don't fully understand the fundamentals. The author's thorough explanation of the basic constructs and group-substitute-evaluate paradigm of the Tcl language was nicely done and clarified my understanding. Anyone who writes Tcl scripts would benefit from reading the first section a few times.

The sections on Tk widgets clearly explain not only the syntax and use of the widgets, but through the extensive use of examples really drive home how to most effectively use them. The book also discusses techniques on how to solve, in a simple manner, problems that face many Tk programmers (such as having a scroll bar attached to two listboxes).

The sections on the Tcl C API are very well done and excellent reading for anyone who is trying to integrate Tcl into a larger application. Not only does the author give a nice overview of the API from a task perspective; he also provides several tricks. My favorite is **Tcl\_Invoke** a C procedure that allows the invocation of Tcl commands without the overhead of substations that come with **Tcl\_Eval**.

Included with the book is a CD-ROM containing all of the examples from the book and a small browser application to help access and run the examples. Perhaps I am being a bit overly critical, but I was a little disappointed with the included browser application. It would have been nice to include a really glitzy Tcl browser to really show off the strengths of the language. Sadly, the included application, although usable, was a little stark and would certainly not inspire a new user of Tcl to get excited about the language. Also included were a collection of Tcl scripts and extensions from the various FTP sites; again it would have been nice if a few more of them had been unpacked and ready to run off the CD.

Although this book is described as being useful for the beginner, as well as expert Tcl user, it really shines for people with at least a little programming knowledge. In some ways the breadth and detail of the book (not to mention its weight) could be a little intimidating for a newcomer. For the true beginner I still like John Ousterhout's *Tcl and the Tk Toolkit* (although it's getting a bit dated).

Throughout the book the author pulls no punches. Covering the language in great detail and providing many useful tricks, he is equally quick to point out potential pitfalls, inefficient commands and constructs that although legal “probably aren't a good idea”.

A great amount of attention is spent throughout the book delineating features that apply to a specific release of Tcl or a specific computer platform (Unix, Windows, Mac). This is particularly helpful if you are interested in writing truly cross-platform scripts. Whenever possible, attention is given to writing code that will work across multiple platforms and revisions of Tcl. Because the author is part of the design team at Sun, he was able to cover some of the new features introduced in Tcl 8.0 including namespaces, binary string support and the on-the-fly compiler.

Without a doubt this book lives up to its name. It provides an extremely practical view of Tcl and Tk, pointing out its features and flaws while focusing on writing scripts that “get the job done.” Mr. Welch has written a book that is essential reading and reference material for anyone who writes Tcl code or integrates it into a larger application.

**John McLaughlin** is a project manager at Hewlett Packard working on test equipment for wireless chipsets. A Linux user since 0.99.13, he lives in the San Francisco Bay area with his wife Noel and enjoys traveling, diving, skiing, hacking with Tcl and writing EXMH extensions (under Linux of course). He can be reached at [johnmcl@sr.hp.com](mailto:johnmcl@sr.hp.com).

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

[Advanced search](#)

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

Advanced search

## Protecting Your Web Site with Firewalls

**Leam Hall**

Issue #48, April 1998

The author provides good, basic information on what the security needs may be and how to balance cost of security against availability of services.



- Author: Marcus Goncalves
- Publisher: Prentice Hall PTR
- URL: <http://www.prenhall.com/>
- Price: \$49 US
- ISBN: 0-13-628207-5
- Reviewer: Leam Hall

If you can configure a DNS nameserver on your coffee break, this book is not for you. However, if you need a platform-independent general overview of Internet/Intranet security issues, Mr. Goncalves uses a conversational tone and an abundance of resources to help you get a firm grasp of the basics.

There are many web sites that are unprotected and unsecured. This may be due to a system administrator who is over tasked and does not have enough research time, who is new to a particular operating system or who just does not know how important security is. The author provides good, basic information on what the security needs may be and how to balance cost of security against availability of services. Individual chapters give introductions to financial issues,

strategies for different web site platforms and implementing services like conferencing, e-mail, FTP, NNTP and HTTP.

There is also plenty of material written in non-technical language. This can be very useful when trying to educate upper management on basic security issues and why they need to be funded. The whole book provides many information sites, including 77 pages of appendices giving resources for firewall products, web server products, authentication, password and other administrative tools. This may be the best part of the book, to help you quickly see what sort of material is available for your operating system.

There are a few things I did not like about the book. The editing and proofreading could have been better. I generally read too quickly to find spelling or grammatical errors, but I found some here. Also, several sections left me wondering what I was supposed to have learned. My most abundant gripe is the nondifferentiation between “hackers” and “crackers”. I consider myself a junior hacker, yet one of the most law abiding people I know. Hackers play with code and try to create; crackers intrude—hopefully the media will soon understand the difference. There is also the ubiquitous CD-ROM; this one contains an evaluation copy of an NT-based logging and reporting system. Nothing for the Unix/Linux crowd.

The book's target audience is those who need a very basic introduction with a lot of resources. It meets that market well, but most *Linux Journal* readers will be somewhat more advanced. For \$49 US, get this book only if you are certain it meets your particular needs. Otherwise, do a lot of web surfing and find a book tailored to your specific operating system and level of expertise.



**Leam Hall** considers his wife and faith to be the cornerstones of his success. He stewards a Promise Keepers group, calls getting beat up “martial arts training” and considers Linux the mental tool of the sophisticate. Leadership mentoring and feeding Catbert are major daily tasks. He can be reached at [gershom@spidernet.it](mailto:gershom@spidernet.it).

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

[Advanced search](#)

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

## Using What We've Learned

**Reuven M. Lerner**

Issue #48, April 1998

This month Mr. Lerner shows us how to set up a web site using many of the techniques he's taught us over the past months.

Over the last year, we have looked at ways to create sophisticated web sites using little more than CGI programs written in Perl. Among other things, we have explored HTML/Perl templates (for separating design from programs), HTTP cookies (to identify returning users) and relational databases (to store information in a readily retrievable format while gaining robustness and security).

This month, we will see how we can use all of these techniques together. Such a combination of techniques is used on many commercial sites and is the basic idea behind several web server projects, including Microsoft's Active Server Pages (ASPs), Vignette's StoryServer, AOL's freely distributed AOLServer and the freely distributable PHP/FI.

Note that many of the examples in this month's installment refer to techniques and ideas that were discussed in previous issues of *Linux Journal*. If you are new to the magazine or to the ideas, please see the "Resources" sidebar, which should give you some good starting points for learning about these subjects.

Our example site will revolve around a single table tracking users' birthdays. Once we have created the table for the birthdays, we will write a CGI program that allows users to enter their birthdays into the table. Finally, we will combine the use of cookies and Perl/HTML templates to create personalized home pages drawing upon the information stored in the database.

Assuming that MySQL is our relational database server and that our table will be in the "test" database, at the prompt type the command:

```
mysql test
```

This starts the MySQL client, allowing us to enter SQL queries interactively. The “test” database used in these examples comes with MySQL and is completely insecure—it cuts down on the space required to explain how to secure new databases created with MySQL. As a result, you should seriously consider creating separate databases and users for each of your web applications, so as to reduce the risk of unauthorized users modifying or viewing data on your system.

First, we must create a table with information about our users and their birthdays. Here are some simple SQL commands that create such a table:

```
mysql> create table birthdays (person_id int unsigned
auto_increment,
->  firstname varchar(15) not null,
->  lastname varchar(15) not null,
->  email varchar(50) not null primary key,
->  birthdate date not null,
->  key id (person_id));
```

This creates the “birthdays” table in the “test” database. The table has five columns (person\_id, firstname, lastname, email and birthdate), none of which may be blank. Each row in the table represents a distinct user whose birth date we wish to track; we ensure that no user is entered twice by setting the “email” column to be a primary key, which is a fancy way of saying that no value of “email” may be repeated. Since users may have multiple e-mail addresses, we cannot ensure that a user will not enter his or her birthday twice. However, this is likely to reduce such repetition, and is better than using the person's name, which is rarely unique.

The first column, person\_id, will be set automatically by MySQL each time we add a user to the database. The first entry in the system will have person\_id set to 1, the second entry will have it set to 2 and so forth. Because person\_id is of type **int unsigned**, our system can accept no more than 4,294,967,295 unique entries—smaller than your particular database might need, but large enough for most of my purposes.

We can get a good picture of our database using the “describe” command at the “mysql” prompt, as follows:

```
mysql> describe birthdays;
Field      Type      Null      Key      Default  Extra
person_id  int(10)  unsigned  MUL      0        auto-increment
firstname  varchar(15)
lastname   varchar(15)
email      varchar(50)  PRI
birthdate  date        0000-00-00
```

### Entering data into the table

Now that we have created the infrastructure, we need applications that will allow people to enter their birthdays. The easiest way to do this is to create an

HTML form whose contents are submitted to a CGI program that takes information from the form and saves it to the database. One such form is shown in [Listing 1](#).

The form is relatively straightforward, although it might seem a bit daunting because of the long selection lists defined using the “select” and “option” tags. Using such lists, rather than allowing the user to enter a birthday into a text field, reduces the number of errors that a user might enter. It is certainly possible, though, that someone could create an HTML form with identical field names, using text fields instead of selection lists, and thus circumvent our system. The moral, then, is that you should always try to reduce the number of errors that users might enter, but always check to be certain that data were entered correctly.

As you can see from looking at Listing 1, our form gathers six pieces of information: first name, last name, e-mail address and user's birth month, day and year. The latter three pieces of information are separated so that we can simplify the user interface; as we will soon see, combining these to create a valid MySQL “date” type is fairly simple.

Our form's “form” tag indicates that data should be submitted using the POST method to a CGI program named “enter-birthday-info.pl”.

We take the user's input and create a SQL query that creates a new entry in the table:

```
# Now that we have the basic information, create
# an SQL query
my $command = "insert into birthdays ";
$command .= "(firstname, lastname, email, birthdate) ";
$command .= "values ";
$command .= "(\"$firstname\", \"$lastname\",
 \"$email\", \"$birthdate\")";
```

Of course, you do not need to store the command in the variable **\$command**. Indeed, you can create the command directly when using **\$dbh->query**, rather than putting it together and then passing **\$command** as an argument to **\$dbh->query**. Putting the query together in this fashion makes it easier to read when programming and easier to send the SQL query to the screen if bugs appear.

After we send our query to the database server, the row is probably added. However, we do not want to just assume it was added because something serious might have happened, and we wish to give a correct indication of the outcome to the user.

First, we check to see if **\$dbh->errno**—the value of an error returned by MySQL—was set to 2000. This is the specific error code returned when trying to insert



a row that conflicts with another row. Since we have defined “email” to be a primary key, the odds are rather high that if `errno` is set to 2000, then we have tried to enter a duplicate e-mail address:

```
if ($dbh->errno == 2000)
{
    &log_and_die(
        "There is already an entry in\ the database for \"
        $email\". Try another\ e-mail address!");
}
```

If this was not the case, then we should check for any other error. The easiest way to detect errors is to see if `$sth` is undefined; if it has not been given any value, then an error occurred, which we identify for the user and in the error log. Note that our general error-catching mechanism needs to come after the mechanism for catching error 2000.

```
elseif (!defined $sth)
{
    &log_and_die("MySQL error " . $dbh->errno .
        "\ on command \"$command\"<P>" . $dbh->errmsg)
    unless (defined $sth);
}
```

Finally, if no errors occurred, then we can print a message indicating success:

```
else
{
    # Return something to the user
    print "<P>Done!</P>\n";
}
```

## Tracking Users

Now users can enter information about their birthdays into the system. But wait—at the beginning of the column, I said that we were going to make it possible to keep track of users' comings and goings. One easy way to do that is to use HTTP cookies, small pieces of data stored on the user's computer that are sent to the site that set them. CGI programs can set cookies whenever they return an HTTP response to a user's browser. Whatever cookies were set are then returned upon each subsequent visit to that web server. In this way, they can be used as variables, albeit variables that might be modified or removed by users worried about their privacy, or who might come to our site from a friend's computer.

Our cookie will have to be a unique identifier that can be used to bring up the user's entry from the “birthdays” table. We thus have two options—the user's e-mail address, which is guaranteed to be unique because it is a primary key, and `person_id`, which is automatically incremented each time a new entry is added to the table. We will use `person_id`, but there is no reason why you could not use the “email” column. Indeed, given that “email” is a primary key, you could even do away with the `person_id` column—except that if you were to create

other tables that refer to individual users, keeping track of an integer (such as `person_id`) is much more efficient than using their full e-mail address.

How can we retrieve the ID that was added to a row that we might add? The most obvious way is to retrieve the row that we just entered, creating and sending an SQL query to the MySQL server. But MySQL has an easier way to do this—after sending our query, we can ask for `$sth->insertid`. `$sth` is the “statement handle,” an object that allows us to send and retrieve information about an individual SQL query and statement, and `insertid` is one of the methods that `$sth` provides.

Once we know the value of `person_id`, we can create a cookie with the following two lines of Perl:

```
my $cookie = $query->cookie(-name => "person_id",
                           -value => $sth->insertid);
```

The cookie (now stored in `$cookie`) is sent as part of the HTTP header returned by the CGI program to the user's browser. Thus, the original version (see [Listing 2](#)) of our program sent a basic MIME header at the beginning of the program's execution with the command:

```
print $query->header("text/html");
```

But the new version of our program will have to move that to a later portion of the program, after we have already sent our query to the database. In addition, we will have to modify our statement such that it sends the cookie along with the MIME information in the header. This can be accomplished with the following code:

```
print $query->header(-type => "text/html",
                   -cookie => $cookie);
```

When the above code runs, it sends both the MIME header that describes the type of output that we are sending to the client, and the information describing the “`person_id`” cookie that we want to set. Every time this user visits our site in the future, we will be able to retrieve the value of “`person_id`”, and thus look the user up in a table in our database.

You can see the results of changing our program in [Listing 3](#). The modifications are fairly small, but they ensure that a cookie is returned to the user's browser whenever we successfully add a row to the database. (When no row is added to the database, the header remains as before, sending the MIME header but nothing else.)

## Retrieving the Data

The final part of our system will bring cookies and databases together with another technique we discussed several months ago, Perl/HTML templates. Templates are pages of HTML with small snippets of Perl interspersed between the HTML tags, accessed via a short program that uses the Text::Template module to turn the Perl into text. This allows HTML pages to perform database lookups, calculations and other elements that require computation too difficult to accomplish with server-side includes. Unlike straight CGI programs, templates can be edited by your site's designers and HTML coders, removing the programmer as a bottleneck to changing the HTML that a program produces.

In [Listing 4](#), you can see wrapper.pl, a CGI program that turns templates into HTML. (A version of wrapper.pl published in a previous issue of *LJ* was not tested thoroughly, and contains several bugs that were fixed in this version.) Assuming that wrapper.pl is installed on your system and that the file /home/httpd/html/birthdayhp.tpl is a valid template, you should be able to request

```
/cgi-bin/wrapper.pl?/home/httpd/html/birthdayhp.tpl
```

In other words, wrapper.pl should be invoked with a single argument, the name of the template that should be evaluated and then returned.

This version of wrapper.pl allows us to pass variables to the template using the *LJ* package (as described in the manual pages for Text::Template). This allows us to pass variable values from wrapper.pl to templates. In general, we would not want to pass variable values from wrapper.pl to a template, since the template should be somewhat isolated from its surroundings and should be allowed to assign its own variables. But in this case, we do want to pass one value, that of \$query (the CGI instance), which allows us to access information passed to wrapper.pl per the CGI specification. This includes the "cookie" method, which allows us to retrieve cookies that our server has set in the past.

I have included a simple version of birthdayhp.tpl in [Listing 5](#), so that you can see how easy it is to include Perl inside HTML. There is a performance penalty for serving documents in this way, since you are forcing an invocation of Perl each time a template is viewed on your system. But that drawback is often counterbalanced by a template's versatility and ease of inclusion in a site. A large Web site's editorial and production staffs can thus modify the site's content and design without disturbing programs necessary for the site to run. The programs are surrounded by curly braces, making them easy to spot in an HTML file.

One thing to remember when dealing with templates is that the output from each Perl fragment is inserted into the resulting HTML. If you wish to send the text “Hello, there” to the user's browser from within a Perl fragment, in order for things to work correctly you must use:

```
{
  "Hello, there"
}
```

or a slightly more formal version:

```
{
  my $outputstring = "";
  $outputstring .= "Hello, there";
  $outputstring;
}
```

Do not make the mistake of trying to use **print** from within a template, as in this case:

```
{
  my $outputstring = "";
  $outputstring .= "Hello, there";
  print $outputstring;
}
```

The contents of `$outputstring` will indeed be sent to the user's browser thanks to `print`, but the text will be sent ahead of the rest of the template. In place of the Perl block containing this code, the `Text::Template` module will insert the result from `print`—the result will have a value of 1 when printing is successful. Setting strings, rather than printing them directly, is the norm in the case of templates, but takes some getting used to if you are an experienced Perl programmer.

### Putting It All Together

The trick now is to create a template that does the following:

- Grabs the value of `person_id` from any cookie that the user's browser might send.
- Uses the value of `person_id` to retrieve information about the user from the database.
- Uses the information from the database to create a personalized HTML page greeting the user.

Getting the value of the user's cookie is almost as simple as setting it. Just as our CGI program can set one or more cookies when it returns an HTTP header to the user's browser, the value of the cookie is sent to our server in the HTTP headers accompanying any request we might receive. Once our `person_id` cookie is set on the user's computer, every visit to our site will be prefaced with the name and value of `person_id`. We can retrieve the values of all cookies on

our system with a call to the “cookie” method, just as we used it to create our cookie.

Listing 6 contains a very short CGI program (show-cookies.pl) that prints the names and values of all cookies sent to a particular server. Remember that cookies are only sent to the server that originally set them—so while your browser might contain a large number of cookie name-value pairs, the output from show-cookies.pl will only display those created by your server.

In show-cookies.pl, we iterate through each of the cookies sent to the web server. But for our particular purposes, we are only interested in a single cookie, “person\_id”. We can retrieve that by using the “cookie” method in the following way:

```
my $person_id = $query->cookie("person_id");
```

Assuming that a cookie named “person\_id” was sent to the web server (meaning that the “person\_id” cookie had been set by a program on that server in the past), its value would now be available in the variable \$person\_id. We can then use \$person\_id as a unique key in our “birthdays” table, allowing us to retrieve information about returning users.

Among other things, we must ensure that the template handles the possibility that a user with a cookie might not appear in the database, or that a user might go to the customized home page without a cookie. In the test template (cookie.tpl), this is handled in a fairly crude way, printing out both the value of the user's “person\_id” cookie and the number of rows matching this value of person\_id in the database. In the example template shown in Listing 7, we include the following code:

```
if (($person_id == 0) || ($sth->numrows == 0))
{
    $outputstring .=
        "<P>Error retrieving information.</P>\n";
    $outputstring .= "<P>person_id (cookie) = \"
        $person_id\".</P>\n";
    my $numrows = $sth->numrows;
    $outputstring .=
        "<P>Rows returned from table = \"
        $numrows\".</P>\n";
    $outputstring .=
        "<P><a href=\""/>";
    $outputstring .=
        "Enter your birthday</a></P>\n";
}
```

This code checks to see if \$person\_id (which is sent as a cookie from the user's browser and should correspond to a single row in the “birthdays” table) is equal to 0, which can also mean that it is not set. If \$person\_id is 0, then we have no cookie on record for this user. This does not necessarily mean that the user has never visited our site before—some users reject cookies because of privacy

concerns, others use multiple browsers (each of which keeps its own list of cookies) and still others might be accessing the web from multiple computers. But our system does ensure that users accessing our site from the same computer (and the same browser) will see their birthday displayed whenever they come to our system.

We also compare **`$sth->numrows`** with 0 to see if no rows were returned from the database. It is quite possible that a user might have visited our site long ago, and that the cookie from that visit remained on that user's computer—but that all early visitors' entries in the database were somehow deleted. Under such circumstances, **`$sth->numrows`** will return 0 (meaning that no rows had a `person_id` column matching **`$person_id`** from the user's cookie), and we have to request a new birthday entry from the user.

If the query did return a row (and we know that it will return one row at the most, since `person_id` must be unique), then we have to grab that row with **`$sth->fetchrow`**, and then read the values of the resulting array into our variables. In this particular case, we do nothing more than print them out:

```
{
    while (my @arr = $sth->fetchrow)
    {
        my ($firstname, $lastname, $email,
            $birthdate) = @arr;
        $outputstring .=
            "<P>firstname =
\"$firstname\"</P>\n";
        $outputstring .=
            "<P>lastname =
\"$lastname\"</P>\n";
        $outputstring .=
            "<P>email = \"$email\"</P>\n";
        $outputstring .=
            "<P>birthdate =
\"$birthdate\"</P>\n";
    }
}
```

Of course, if we were interested in doing something a little more interesting, we could do so by taking the values returned by **`$sth->fetchrow`** and using the resulting variables in the title of the HTML page or a comparison of today with the user's birth date. The point is that the database is a means for storing information between invocations of the CGI program. Once the information has been read into the CGI program from the database, we can use that information just as easily as if we had assigned the variables at the beginning of its invocation.

## Summary

Most people do not need to be reminded of their birthdays. And indeed, the use of birthdays in this example was simply for demonstration purposes. Even with the limited information we stored in our database, we can create a bare-

bones personalized home page that displays the user's name in the title. With a little more work, we could print a special message on this user's birthday, or an indication of how many days remain until the user's next birthday.

And because we have stored all users' birthdays in our database, we can create applications that access other birthdays on the system. For instance, we could create a CGI program (or a Perl/HTML template) that finds other users on the system with your birthday. The possibilities are endless, and putting the information into templates means that you (as the programmer or webmaster) can concentrate on writing the code necessary to make things run, while the site's editorial and production staffs can make things look pretty and ensure that they are grammatically correct as well.

With that, we end our whirlwind (albeit longer than usual) tour of integrating multiple techniques into a single web site. Web sites based on databases are increasingly popular, for good reason. The largest and best-known web sites combine back-end databases with templates and cookies to give each user a personalized experience; now that you have seen how it can be done, create some on your own sites.

## Resources



**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](mailto:reuven@netvision.net.il).

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

[Advanced search](#)

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

Advanced search

## Letters to the Editor

### Various

Issue #48, April 1998

Readers sound off.

#### **“Linux Means Business: Highway POS System”**

In *Linux Journal*, November 1997, there is an article by Marc Allen about Linux running a gasoline station. Mr. Allen, an employee of the vendor of the system, Schlumberger, states: “We are, however, the only Unix-based one.”

Not a bad story, but not completely true. At the gasoline station where my brother works, the whole system (pump control, sales, e-cash) has been working under SCO Unix for more than 1 year—24 hours a day, 7 days a week, running on a 80486 PC.

The company that makes the software is Sofitam of Belgium (Satam is the gasoline station equipment).

—Groeten Ben Erkens, The Netherlands berkens@airchips.xs4all.nl

#### **Linux as Proxy Server**

I would like to point out that in the article called “Linux as a Proxy Server” by Peter Elton in the December issue of *LJ* there was absolutely no mention of Linux's built-in firewalling and proxying support. As I think most seasoned “roll your own” kernel compilers will know, Linux can do almost all of what Peter describes without resorting to any extra user-level daemons or utilities. I am speaking specifically of the kernel's IP Masquerading, Transparent Proxying, Accounting and Firewalling features. From the included Listing 1, it seems that Peter was using a very antiquated kernel indeed and perhaps these features did not exist. When writing about bleeding-edge operating systems one should do the proper homework to be aware of current OS features.

—Oliver Jones orj@ihug.co.nz



## Something Funny

I have been following *LJ* for over a year now and can't believe how good it's getting. I like the design changes, the way you shortened the "What is Linux" part, and every issue is a better mix of topics than the last one.

If you want to hear something funny, my brother works at a *major* American software company, and he tells me that developers there are bringing up the topic of porting their products to the Linux platform more and more frequently. Apparently there is a lot of Linux quietly installed around the world (enough to be seen as a significant market by a large American software player).

—Harold Sinclair [morelife@stealth.net](mailto:morelife@stealth.net)

## Microway Screamer 533 Review

I'm writing in response to the review of the "Microway Screamer 533" by Bradley Willson which appeared in the January issue of *Linux Journal*. This review is especially timely for me as the research group I work with is purchasing a new computer for development work, and the price and performance of the Alpha-PC running Linux are extremely attractive. As I started to read the article, however, my excitement quickly waned. Nowhere in the review of the Screamer 533 are there meaningful benchmarks. Moreover, the benchmark that the author put together is worthless for precisely the reasons mentioned in the article: so why include it?

The author would have done better for *Linux Journal* and Microway if he had included results from the SPEC benchmarks (SPEC benchmarks run on the Screamer 533, *not* benchmarks from Digital with the 533/21164A chip running Digital Unix in some form of an Alpha Station), the BYTE and Bonnie benchmarks and, perhaps, some of Microway's own programs. All benchmarks have shortcomings but they are much more useful to convince coworkers and bosses with than stories of fast cars.

Please *Linux Journal*, give us meaningful hardware reviews.

—Rich McClellan [rich@chemistry.ucsc.edu](mailto:rich@chemistry.ucsc.edu)

I wrote to Ann Fried and asked her if she would make 533 benchmark numbers available on their web site. I have not received a reply yet.

For you, the article did not satisfy your questions and for that I apologize. I wrote the article for a less technical audience because of market demographics. There are simply more Intel/Linux users than Alpha/Linux people. The focus of the piece was to direct their attention to a machine they might otherwise

overlook because of price, technical astigmatism or both. The academic and technical audiences generally have resources that allow them to own an Alpha and understand it, as you have demonstrated.

I realize that this does not help you sell the machine to management, and to that end I will continue to work with Microway to get some meaningful numbers to you.

—Bradley J. Willson [cpu@ifixcomputers.com](mailto:cpu@ifixcomputers.com)

### Evil Cookies

In January's "At the Forge", Mr. Lerner writes that cookie information cannot be shared between different web sites. True, but cookies still present a danger to people's privacy. The large banner ad sites all use cookies, and their ads appear as in-line images on many web sites. So it doesn't matter if you visit Lycos, the *New York Times* or some other site—as long as the banner ad is there, the banner ad site will be able to use its cookie to track where you've been and tailor the ads it shows you appropriately.

—Joey Hess [joey@kitenet.net](mailto:joey@kitenet.net)

### Linux Ports

I am writing in response to Dave Blondell's letter in December, where he says "The sad truth of the matter is that Bentley, and for that matter most other software companies, don't get enough requests for Linux ports to justify the production costs."

Perhaps it's true for ports from non-Unix environments, but it surely is not true otherwise. In the same issue, a look at "Linux Makes the Big Leagues" by Sam Williams and the "A Place for Linux" presentation by Chip Richards (mentioned by Sam) shows exactly how I persuaded our company to start using Linux. For only \$250 we could have Linux with Metro Link Motif—what's more, we could use a cheap PC clone that put our HP715 to shame in the performance stakes.

Since we associate closely with some of our clients, they often visit and get to see some of the new enhancements that are under development. Often they noticed how fast Linux was compared to other platforms, and switched to Linux.

And the best part is that I never need to change a line of code when compiling across platforms—I use simple shell scripts that are used as CC and LN. The combination of Linux[*Intel*] with its BIG ENDIAN architecture and HP-PA RISC with its nice LITTLE ENDIAN (same as networking) provides a nice combination

of test beds to ensure both byte swapping and 64/32 bit compatibility are tested.

At the end of the day it is no extra effort to provide a Linux solution. Probably the biggest deterrent is the *loud* anti-commercial chorus. Those folks who don't mind paying for software should be more vocal.

While not everyone may appreciate or use any of the free software that I have contributed to the Linux community, some of the credit must go to my employer (who does not provide free software as a rule), for the skills and resources I used to create my free software were gained from them; in return, they use some of my free software.

—Ross Linder ross@mecalc.co.za

### **Author Photos**

Did you guys happen to notice how much the author Richard Sevenich bears a resemblance to the author Dean Provins? By golly, they could be brothers.

—M. David Gelbmand gelbman@npiny.com

Yes, we did notice. Unfortunately, at the same time you did—when the magazine arrived from the printer, not before we sent it to be printed. We apologize to both Mr. Sevenich and Mr. Provins for the mixup —Editor

### **Articles on Beowulf Systems**

Thanks for the articles on the Beowulf systems and PVM. I have known Patrick Goda of the Loki system for three years now, since he came to our local Linux User's Group this past spring to talk about Loki.

I was a little disappointed that there was no mention of what I consider to be the ultimate in "Freely Distributed Systems": the "Stone SouperComputer". The Stone SouperComputer is a project at the Oak Ridge National Labs, the same laboratory that gave us the PVM software.

It seems that a project at Oak Ridge needed some computer power, but had no budget for it. They decided to implement a Beowulf system using *donated* computers. By using older 486 and low-end 586 systems donated to them, they were able to create a *no-cost* supercomputer. It now has 48 system boxes working together to do real-world work. You can find the web pages at <http://www.esd.ornl.gov/facilities/beowulf/>.

I am very excited about the "Stone SouperComputer" project, because for the first time it demonstrates that any university, college, high school or even grade school can put together its own "Soupercomputer" using a freely distributed operating system and allow students to tackle the most difficult type of coding, that of a parallel environment. It also paves the way for low-cost, highly available, high throughput systems for research or even administrative work.

—Jon "maddog" Hall, Linux International maddog@zk3.dec.com

### Microstation 95 Update?

We have contacted a Bentley re-distributor in our area who informed us that they could in fact secure us a copy of Microstation 95 for Linux. We have also checked with the supplier of our current add-on software to make sure that their software for Microstation would run on the version for Linux, and we were informed that not only did it run on Linux, but that they had customers already doing just that. While it is possible that the other customers may include academia, I don't believe they all are. This leads me to believe that, with the help of a good re-distributor, users outside academics should be able to get a copy. Although the distributor stressed the "unsupported version" heavily, with a demand for the Linux version this may disappear. Please keep up the great work.

—Medina County Engineers lfilak@ohio.net

### Red Hat CDE Article

As an owner of Red Hat/TriTeal CDE, I was interested in the CDE article in the January 1998 issue of *Linux Journal*. However, I'd like to add one thing to the article concerning the root dtlogin "problem". This isn't actually a problem, per se, it's a "feature". Refusing dtlogins by root was chosen intentionally for security reasons. However, this feature can be disabled, as I discovered when I asked Red Hat support about it. Here are the three things that they suggested could be done:

1: Add **dt** to the `/etc/securetty` file.  
2: Remove the `/etc/securetty` file.  
3: Remove the line in the `/etc/pam.d/dtlogin` file referencing `secure tty`.

I simply did number 1, and now I have a nice X root login.

—Matt Harrell mharrell@voyager.net

### URL Error

For your information, there is an incorrect URL in your “Ricochet Modem” article by Randolph Bentson (January 1998)—[www.metricom.net](http://www.metricom.net) does not work. The correct address is <http://www.metricom.com/>.

—Thomas K. Pedersentpedersen@kraft.com

### Comments about January 1998 Issue

I am happy to see an issue devoted to parallel processing, but I am curious why you did not include information on the Linux SMP project which is at <http://www.linux.org.uk/SMP/title.html>? This project is directly related to parallel processing.

Also, will *LJ* make any future effort to make an archive CD-ROM of all the past issues? I hope so, that would be great.

—Steel Viper sviper@solli.inav.net

I wish we had room for everything about our focus but there are only so many pages in a magazine. I would like articles about SMP and the Stone Soupercomputer program mentioned above, but they will have to be in the future. As to your second question, an archive CD-ROM of 1994 and 1995 will be available in March —Editor

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

[Advanced search](#)

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

[Advanced search](#)

## Workplace Solutions

**Marjorie Richardson**

Issue #48, April 1998

This month we focus on workplace solutions and give you five feature articles of this type.

### Workplace Solutions

Linux is spreading like wildfire through businesses these days. The proof is in the number of articles and article suggestions that I receive every month. The majority of these articles describe how Linux is being used at work by a surprisingly varied group of authors from all over the world. Usually we print only one of these "Linux Means Business" articles each month, but this month we focus on workplace solutions and give you five feature articles of this type. We have even more but there is just not enough space for all of them, so we will continue to publish one each month. We also have a few product reviews to help you select office applications that support Linux.

Last year there were several news items proving that Linux is being taken seriously by the big companies. At the LISA'97 Conference in San Diego in October, *LJ's* publisher, Phil Hughes, talked to a programmer who is porting Linux to SGI hardware. UMAX Technologies invested in VA Research, a company that sells its computers with Linux installed, even though UMAX is also a manufacturer of PowerMac clones. In Canada, Corel Computer Corporation announced that Linux will be the operating system installed on their new Video Network computer. We have an interview with Corel's President, Mr. Eid Eid, in this issue.

Another item of note is that Netscape has announced that Communicator 4.04 will include JDK 1.1 with support for Linux. This is definitely a step in the right direction, since Netscape has not provided any kind of support for Linux before. This is most likely due to the fact that Caldera is providing a Linux version of Netscape in their OpenLinux Standard.

## A Brief Note

As I am writing this in January, Microsoft is again being charged with unfair business practices, this time for bundling Internet Explorer with its Windows operating system. I must admit to brief sympathy on this one—I mean, Microsoft did announce in the beginning that this was their plan, and no one objected until it became a reality. However, I lost all sympathy when yesterday (and I paraphrase here) the Microsoft spokesman was asked if he truly thought that when the judge said Microsoft should supply a Windows version without IE, the judge meant for Microsoft to supply a version that didn't work, and the spokesman said "Yes." You have to wonder what these guys are thinking.

## Upcoming Events

Some upcoming Linux events that you might be interested in are:

- Spring Comdex Linux Pavilion, April 20 through April 23, 1998, Chicago, IL. For more information see <http://www.comdex.com/> or visit the *Linux Journal* WWW page (<http://www.linuxjournal.com/>).
- USENIX Conference on Object-Oriented Technologies and Systems (COOTS), April 27 through April 30, 1998, Sante Fe, NM, El Dorado Hotel. For more information see <http://www.usenix.org/events/coots98/>.
- 4th Annual Linux Expo, May 28 through May 30, 1998, Chapel Hill, NC, Bryan Center at Duke University. For more information see <http://www.linuxexpo.org/>.
- 23rd Annual USENIX Technical Conference, June 15 through June 19, 1998, New Orleans, LA. For more information see <http://www.usenix.org/events/no98/>.

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

[Advanced search](#)

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

[Advanced search](#)

## The Software World—It's a-Changin'

**Phil Hughes**

Issue #48, April 1998

By far the most interesting part of Netscape's announcement for the Linux community is the fact they will release the source code for Communicator starting with 5.0.

Welcome to my attempt to write a "Stop the Presses" with a subject that really asks to be editorialized on. First, let me set the scene: today is January 22. The important events of this day are:

- Bill Clinton is once again accused of a sexual impropriety.
- The Pope is in Cuba.
- Netscape has announced that their browser is now free and that they will freely distribute the source code for it.
- Microsoft has somewhat folded in its browser battle with the U.S. Justice Department.

The first two items are just for context—it has been an exciting day. I am really here to discuss the last two items.

Let's get the Microsoft information out of the way first. My understanding is that a compromise has been reached between Microsoft and the justice department—the Internet Explorer icon will not appear on the desktop, but the browser itself will still be included. As the easiest way to get a new browser is to download it off the Internet and 90% of all personal computers today come with Microsoft Windows, it seems that all we have done is made it a little harder for Internet Explorer to be on 90% of the desktops. Hopefully, there will be further developments in the Microsoft vs. the U.S. Justice Department game.

The Netscape item has two parts. The first, making the browser available for free really is a necessity; 90% of new personal computers come with Windows and, thus, Internet Explorer. Whether IE is better than anything Netscape offers



or not isn't the issue if one comes with your computer and you have to go buy and install the other one. Numbers back up this statement. Netscape used to account for about 90% of the browser market while 60% is probably the case today.

The good news for Netscape is that they have managed to shift their revenue stream away from stand-alone client software. Their own numbers show that in the fourth quarter of 1997 these revenues were only 13% of the total, down from 45% a year earlier.

By far the most interesting part of Netscape's announcement for the Linux community is the fact they will release the source code for Communicator starting with 5.0. Sure, this will also make a change for them in the Windows arena and may force Microsoft to make some brave decision as well, but let's look at what this does for the Linux community.

The first thing I see is talk on the Gnome mailing list about a version of Navigator using Gnome. Call it Gnomescape, it is potentially a full-featured browser with a look and feel that is likely to become the Linux standard. [For more on Gnome see the "KDE and Gnome" article by Larry Ayers in issue 24 of *Linux Gazette*, January 1998.]

Netscape claims they are releasing the code to allow the Internet community to contribute to the development. (I expect Linux helped them realize that is possible.) For us, this can mean that instead of complaining about Netscape bugs, we can fix them. I expect, based on Linux history, the best, most bug-free version of Netscape will appear on Linux systems first.

Free Communicator and free source code mean that Linux systems become a much cheaper choice for "Web Appliances". It also means inexpensive kiosks at shopping malls, car dealers, etc. While I am sure Netscape made this decision to help their competitive position with Microsoft, I think we will see a huge impact on the Linux scene. Of course, if Linux replaces Windows as the operating system installed on 90% of the PCs sold today, Netscape will be as happy as the Linux community.

What's still up in the air is what sort of license the source code will fall under. GPL is one choice; a license more like that of BSD is another. Check out "Linux News" and the discussion groups on our web site [<http://www.linuxjournal.com/>] to get up-to-the-minute information on what is happening.

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

[Advanced search](#)

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

Advanced search

## New Products

**Amy Kukuk**

Issue #48, April 1998

DupliDisk-PCI, Visual MATCOM 1.0, Scientific Astronomer and more.



DupliDisk-PCI

ARCO Computer Products, Inc. has announced the release of DupliDisk-PCI. DupliDisk-PCI is a driverless disk mirroring adapter designed for the PCI bus. The price of this product is \$225 US. The DupliDisk-PCI can mirror up to two pairs of IDE, E-IDE or U-IDE drives of any capacity when inserted into a PCI bus slot. It connects via a standard IDE cable either to the IDE connector built into most PCI motherboards today or to an add-in IDE adapter installed on the system. The DupliDisk-PCI is a hardware-based dual-channel adapter. It requires no TSRs or operating system-dependent device drivers.

Contact: ARCO Computer Products, Inc., 2750 North 29th Ave., Hollywood, FL 33020, Phone: 954-925-2688, Fax: 954-925-2889, E-mail: arco@arcoide.com, URL: <http://www.arcoide.com/>.

### **Visual MATCOM 1.0**

MathTools, Ltd. has announced Visual MATCOM 1.0. Visual MATCOM allows you to use MATLAB and MATCOM development from within the Visual C++ 5.X environment. Visual MATCOM gives the ability to edit, compile, debug and view m-files within the Developer Studio. Some features of MATCOM 1.0 include keyword syntax coloring, automatic translation of m-files into C++ and compilation into stand-alone executable and viewing matrix data. The price for a single commercial license is \$699 US and an educational license is \$349 US. An evaluation copy is available from the MathTools web site.

Contact: MathTools Ltd., P.O. Box 855, Horsham, PA 19044-0855, Phone: 888-628-4866, Fax: 212-208-4477, E-mail: [info@mathtools.com](mailto:info@mathtools.com), URL: <http://www.mathtools.com/>.

### **Scientific Astronomer**

Wolfram Research has announced the release of Scientific Astronomer. Scientific Astronomer is a software package that allows users to determine which constellations, planets, galaxies and comets are visible from their location and to pinpoint when and where to look for them. The product includes five star charts, two and three-dimensional planet plotting, eclipse calculations and satellite tracking. The commercial price for the software package is \$295 US.

Contact: Wolfram Research, 100 Trade Center Drive, Champaign, IL 61820-7237, Phone: 217-398-0747, E-mail: [info@wolfram.com](mailto:info@wolfram.com), URL: <http://www.wolfram.com/>.

### **Cyclades-PR3000**

Cyclades Corporation has announced the Cyclades-PR3000. Cyclades-PR3000 is a mid-range, full-featured, multiprotocol router. The Cyclades-PR3000 is based on the PowerPC MPC860MH. It relies on a dual-processor architecture and delivers more than 50 MIPS of raw performance. The PR3000 can also be configured with multiple serial ports (up to 64 per box) and can perform the functions of a router and remote access server in one box. Prices start at \$2449 US for the end user.

Contact: Cyclades Corporation, 41934 Christy Street, Fremont, CA 94538, Phone: 888-CYCLADES, E-mail: [sales@cyclades.com](mailto:sales@cyclades.com), URL: <http://www.cyclades.com/>.

### **ARIA 2.5**

Andromedia has announced the release of ARIA 2.5. ARIA 2.5 is “logless” web site activity tracking software. Some of the product's new features include data compression and firewall compatibility, network-level monitoring, dynamic site tracking, trend analysis and data mining capabilities. The new version tracks activity on all major web servers and includes a database export utility, ARIA Xporter.

Contact: Andromedia, Phone: 415-278-0700, E-mail: info@andromedia.com, URL: <http://www.andromedia.com/>.

### **NetTracker 3.5**

Sane Solutions LLC has announced the release of NetTracker 3.5. Some new features include support of gzipped log files, built-in web server for analysis on client computers, standardized Internet and Intranet configurations and grouping and tracking of users by department and local keyword summary. NetTracker 3.5 is priced at \$295 US. Demonstration copies of the new version are available at the company's web site.

Contact: Sane Solutions LLC, 3 Preston Drive, Wickford, RI 02852, Phone: 401-295-4809, E-mail: info@sane.com, URL: <http://www.sane.com/>.

### **S.u.S.E. 5.1**

S.u.S.E. LLC has announced the release of S.u.S.E 5.1 which includes four CD-ROMs, a reference book and “YaST”. Some new features include the addition of Adabas D Personal Edition, Applixware 4.3 demo, a revised manual and several new packages. The new version also includes the XSUSE X Server and a fax server. YaST contains updates via FTP and source packages in srpm format. The price of S.u.S.E. 5.1 is \$49.95 US.

Contact: S.u.S.E. LLC, 458 Santa Clara Avenue, Oakland, CA 94610, Phone: 510-835-7873, Fax: 510-835-7875, E-mail: info@suse.com, URL: <http://www.suse.com/>.

### **Samba 1.9.18**

The Samba Team has announced Samba 1.9.18. Samba 1.9.18 is a file and print server suite for corporate network integration with Microsoft Windows clients. Samba now implements the opportunistic locking feature of the SMB/DIFS protocol. The new version provides for dynamic code page support and full support for servers with multiple network interfaces. Samba 1.9.18 has no client license fees and may be used without cost on any compatible server

operating system. Commercial support for Samba is available at <ftp://samba.anu.edu.au/pub/samba/Support.txt>.

Contact: The Samba Team, URL: <http://samba.canberra.edu.au/pub/samba/>.

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

[Advanced search](#)

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

## Managing your Logs with Chklogs

**Emilio Grimaldo**

Issue #48, April 1998

An introduction to a program written by Mr. Grimaldo to manage system logs.

One of the attributes that characterizes Unix systems, and therefore Linux, is true multitasking. Because of it, there are usually many processes running on your machine, including the not-so-evil daemons and other important programs such as **uucico** (Unix-to-Unix copy-in copy-out). If your system is properly configured, those programs leave traces of their activities in the system logs. These logs usually contain lots of data that must be filtered to generate more readable reports. In the case of system problems, these logs are a valuable source of information for tracking, and possibly solving, the problem.

Years ago, when I was faced with the prospect of spending some of my free time looking through raw logs and trimming them so that they wouldn't eat up disk space, I decided it was time to take action—I wrote the program **Chklogs**. ChkLogs is an acronym for “Check Logs” in the Unix tradition.

Whether you are an experienced Linux user, system administrator or a newcomer to the Linux world, you will certainly find this subject of interest. Although it is mainly a system administration tool, it also has applications in the user world.

In this article I will introduce you to version 2.0 of Chklogs which should be out by the time this is published. Currently version 1.9 release/build 2 (1.9-2) is available, which is the last Perl 4.0x compatible version. Version 2.0 and higher require Perl 5.003—which is not much to ask considering about 99% of registered users have that Perl version.

## Features

Here are some of the features offered by Chklogs:

- Individual specification of thresholds and action(s) for each log
- A choice of compression program
- Log management by size or age
- Addition of user extensions
- Logical Log Groups with pre- and post-processing
- Global and local repositories (Alternate Repository feature)
- Log shuffling (also known as log rotation)
- Directory lumping
- Requires no programming experience
- User resource file
- Fully based on Perl 5.003, except for the user interface
- Nice Tcl/Tk user interface, available separately (See Resources at the end of this article.)

Logical Log Groups are groups of logs that have something in common; this could be a collection of UUCP logs, INN logs or anything you consider a group. You can use virtually any name; the only restriction is that the name must be valid to create a directory. Chklogs has two built-in groups: syslog and common (reserved).

Alternate Repository (AR) is a directory where the archived logs are stored after they have been processed if they met the threshold condition. By default, archived logs are created with **tar** and GNU **zip**. A local AR is a directory named OldLogs under the directory in which the log resides. A global AR is a directory hierarchy (you define the root of this AR) where log archives are divided into logical groups. Here, you will always see the "common" subdirectory where all "orphan" logs go, i.e., those that have not been explicitly declared to belong to a group.

Log Shuffling is well known and is also called log rotation. The "phased-out" log is assigned a number or tag each time until a maximum is reached, at which point the oldest one is removed.

Directory Lumping is another nice feature. Instead of specifying each log separately by name, you supply a directory name. Chklogs treats any non-archived, non-directory file (as recognized by Chklogs) as a log and acts on it according to the action specification. A real-world application of this option is a site that gives UUCP access to a large number of subdomains (very



cumbersome to do by hand), and a separate UUCP transfer log is kept for each site. Chklogs always determines first if you have specified a directory or a file.

### Installing Chklogs

Every time I make the first release/build of a new version (e.g., v2.0-1), I submit it to Sunsite and Funet FTP sites. Whenever a fix is required (and therefore no new features), I put out a new release/build (e.g., 2.0-2) only on the primary site, my ISP. To make sure that you have the latest version, check out the primary site and/or the Chklogs web page at <http://www.iaehv.nl/users/grimaldo/info/>.

Now, get “version 2.0 build 1” installed on your system. Unpack it and go to the root of the directory tree:

```
gunzip chklogs-2.0-1.tar.gz
tar xvf chklogs-2.0-1.tar
cd chklogs-2.0-1
```

Under the root tree is the bin directory containing the scripts and modules which comprise the Chklogs package. The /doc directory contains all the necessary documentation, including man pages in both troff and HTML format. A plug-out directory contains some extra utilities to scan your logs—use them as is or as examples to build your own. Last is the /contrib directory. In the root of the tree there is the README file, release notes, a makefile for installation and, most important, the GuideMe script. Type:

```
GuideMe
```

And it will do just that or at least make the attempt . This script performs a probe into your system and indicates which configuration parameters must be changed in which files. Follow its advice closely. In the end, it will ask you if you wish to send in your registration. If you select “no” that's fine. If “yes”, you will receive mail regarding updates and major fixes.

Assuming you have made the necessary configuration parameter changes (mailer, compress program, administrative account, library location, Perl location etc.), you are now ready to actually make Chklogs work for your system. I will also assume it has been installed (see Makefile) in /usr/local/sbin and /usr/local/lib/Degt/ by the **make** command (**make install**). The same library directory is shared by the graphical user interface.

If you are not yet sure you wish to commit your logs to Chklogs, you should make a back-up copy of them in a local directory. Run Chklogs on those copies until you feel safe, and you will. Doing this requires use of the resource file (see below).

## Chklogs Configuration

The Chklogs package consists of an administrative program (**chklogsadm**), the main program (**chklogs**) and some Perl modules (Sntp.pm, Chklogs.pm, Interp.pm). For the second part of the configuration, we need the Chklogs configuration file (chklogs.conf) and, optionally, the resource file.

### The Personal Resource File

The resource file (.chklogsrc) resides in your home directory. You need it if you wish to first test on a copy of your logs or if you don't have root permissions (therefore, not used for the system logs but for others). When Chklogs runs, it first gets configuration information that is coded in the scripts configuration variables (the configuration section is clearly marked in the source), then it looks for a Personal Resource File (PRF) and then the command-line options, in that order.

In the PRF you can override some variables from the script configuration, for example, the admin user and others. Here is an example provided in the distribution:

```
# Personal resource file for Chklogs 2.x
set ChklogsConf ~/devel/test/chklogs.conf
set ChklogsDb ~grimaldo/devel/test/.chklogsdb
set VarRun ~/devel/test
set RelativePath MyOldLogs
set Admin grimaldo
mode ignore on
set SyslogConf /etc/syslog.conf
```

You will notice it has a syntax similar to Tcl. That is, with the exception of the mode lines, everything between **ignore on** and **ignore off** is not interpreted. The Tcl/Tk GUI does have a dummy mode routine, but all variable settings are performed as this ignore mode is only meaningful in the Perl mode. Personally, I find the dollar sign in front of normal variables annoying. This same resource file (with some extensions) is used by the GUI. The PRF example from above does not show the variables that are used exclusively by the GUI. However, for the examples in this article, I will assume you are working on the actual logs and not a copy.

In the above script, all variables are overrides for the internal configuration variables. ChklogsConf indicates the location of the configuration file (introduced in the next section), ChklogsDb is the internal database (you must never edit or move this file around), and VarRun is the directory where the PID lock files are kept (including where to find Syslog's and Chklogs' PID file). RelativePath is used when you choose the local repository, and Admin is the e-mail address where the reports are sent when the mail command-line option is

specified. Normally, I don't need this file when running Chklogs, but I do use it during testing as a non-privileged user.

### The Configuration File

Listing 1 is a sample configuration file demonstrating the various features. This file contains three types of lines: comments/spacers, directives and specifications. It is divided into two major sections, the header and the body:

- A **directive** line begins with the two characters **#:**  and because this pattern is checked first, it is not confused for a comment.
- A **comment** line is introduced by the **#** character. Note that you cannot have comments at the end of a specification line.
- The **specification** line is any line that is not empty or a directive or comment. It defines the attributes for each of the logs you wish ChkLogs to manage.

The header is comprised of the comment lines that identify this configuration for a particular host, and the two directive lines which must appear in the order shown. The header can be created with either of the following two administrative commands:

```
chklogsadm newconf -global /var/log/Chklogs
```

or

```
chklogsadm newconf -local
```

The first directive (**options**) is used to set global options for this configuration. At present only two option names are recognized, and both deal with the Alternate Repository Feature explained earlier. The two acceptable options are **global** and **local**. The second directive (**global**) is also related to the Alternate Repository. It must be set to the root directory where the logs are going to be archived (i.e., those with "archive" attribute) by group hierarchy.

In the instruction section or body, we find mostly specification lines. These lines follow a particular syntax:

```
LogName Threshold Action [Parameter(s)]
```

The first field is a fully-qualified log name, the second is a threshold that can be a size or age. The size must be in bytes or, optionally, append the K qualifier for kilobytes. If you opt for an age, it must be in days by appending a D (e.g., 7d) or in months by appending an M (e.g., 2m). The qualifiers D, M and K are case insensitive. The log is always compared to the threshold, if it is reached then the specified "Action" is taken.

The Action field can have a value of archive, execute or truncate, which has no arguments, Chklogs uses the **truncate** system call to chop the file to a length of zero. Use this option if you don't care about this log or after you have used the log with either the archive or execute options as shown in Listing 1.

The archive action takes one parameter that indicates how many logs to keep archived before rotating/shuffling. If no parameter is given, an internal one is used (see documentation). Use this option, if you plan to look at this log at a later time.

The execute action is used to spawn a plug-out, that is, an external program that will operate on the log. This is useful if you have a particular log scanner that will filter the log and give you a human-readable report. You can specify as many parameters as you wish, the special parameter **%L** is replaced by Chklogs with the full name of the log and is used to inform the plug-out which file to process.

Listing 1 also shows a Logical Group definition. There, a group named NetNews is created so that, if a global alternate repository is used, there will be a NetNews directory under it in addition to the /common directory. For each group, you can associate an external program to execute in the plug-out interface. Parameters are allowed, but **%L** is meaningless in that context. Pre and Post specify which program to execute before and after the group has been processed respectively. A group definition always starts with the three directive lines in that order, then one or several logs and, finally, an empty line or a comment line. Do not include any of these in between the lines of the group definition.

In the case of the sample NetNews group, we want the daemon to stop using the logs before we touch them and restart (or continue) after we are finished. This is particularly useful for INN (which has a program to control the daemon) and HTTP. Chklogs handles syslog automatically; it knows what to do and how to do it.

Finally, notice that the last specification line refers to a directory as explained earlier in the Directory Lumping section.

### Setting Up the Environment

Once you have created your first configuration file, you need to have Chklogs initialize its internal database (not to be confused with configuration) specified by either the configuration variable ResrcFile (for historic reasons) or the Personal Resource File's ChklogsDb variable. Remember, initialization overwrites any previous history so should be done only after the first

installation (rather than when you make changes to add/remove logs/groups). Use the following administrative command:

```
chklogsadm init
```

Then, issue the administrative command:

```
chklogsadm initrepos
```

to initialize all the alternate repositories. Unlike **init**, you must execute this command each time you create a new group or change the location of your alternate repository (the global parameter). All of these steps are explained in detail in the documentation provided with the package.

Finally, the **sync** option of `chklogsadm` is needed whenever you make a modification to the configuration file (`chklogs.conf`). This is not done automatically because it is not very effective if several modifications are made to the file consecutively. This operation is as simple as typing:

```
chklogsadm sync
```

### The Plug-out Interface

Chklogs comes with several plug-outs. I use them to generate statistics on my UUCP logs, listserver log, etc. You can build your own plug-outs, and in fact I would like to hear about any log scanners or filters you have. Available plug-outs are stored in the `/plug-out` and `/contrib` directories.

A plug-out is any external program executed by Chklogs in the Pre/Post group directives or the execute actions. This program must not generate any output on `stdout/stderr` as it will clutter the report, and if you, like most users, run Chklogs with a **cron** job, then this sort of behavior is undesirable. So, do you have a nice scanner that *does* write to `stdout/stderr` and don't want to hack it up? Or don't know a thing about programming? Simply use the **cdkwrap** wrapper provided in the distribution, and it will capture all the output and mail it to you.

Also, the plugged-out child inherits certain environment variables that provide useful information:

- **CDKLOG**: The fully-qualified log name
- **CDKROOT**: The archive repository
- **CDKMAILTO**: The e-mail address for mailing the report

## Running Chklogs from the Command-Line

Now, we have finished our setup; we have a configuration file, a resource file and initialized repositories. The administrative tasks are done for now (you can make changes later, if needed), and we are ready to get down to the action. Chklogs has several command-line options, some of which can be combined to achieve a particular effect. I won't cover all of them, or even all of the possibilities, but I will discuss enough of them to enable you to begin, and to acquaint you with a few of its capabilities. Note that we will now be using the chklogs program, not the administrative program, chklogsadm.

Basically, there are four major actions you can perform:

1. Check the configuration file.
2. Get an overview of all the logs that are archived.
3. Obtain a report of actions to be taken when you execute the program.
4. Perform the actions as directed on the configuration file.

To check the correctness of the configuration file (although the check is not very thorough) use the following command:

```
chklogs -w
```

For a quick overview of which logs are archived into the repositories, there is the **-l** command line option:

```
chklogs -l
```

This gives you an indication of how many logs you need to scan or filter and needed information in case something suspicious happens on your system.

To get the usual log report indicating whether a log is still within its threshold, and if not, what action would be performed, use the **-w** option:

```
chklogs -w [-m]
```

Alternatively, you can also specify the **-m** (mail) option to mail the report. You cannot put them on the same switch (**-wm**).

Finally, when you want Chklogs to actually process your logs as specified in the configuration file, simply use it without any of the above options:

```
chklogs [-m]
```

A report is produced on standard output unless you use the mail (**-m**) option. Mail is the most frequently used option.

Most users make an entry into the `/etc/crontab` file so that Chklogs runs every day at a particular time and mails the report. A typical crontab entry looks like this:

```
# System Cron Tab
45 23 * * * root /usr/local/sbin/chklogs -m
```

### Summary

I hope that you, too, will find Chklogs to be a useful package and will soon incorporate it into your system administration tool box. It is used by the Linux community and by most of the other major Unices (I counted 9 different flavors in my last scan of the registration database), for small home systems, service providers and other major network providers. I am always available by e-mail for suggestions, and if there is a problem I always give an answer. A standard form for reporting problems is provided with the distribution to ease my task. I would like to thank all of those who have sent suggestions and encouraging e-mail, as well as problem reports. It has been a great experience to provide this free software to the community. Chklogs is “postcard-ware”; I collect them.



Emilio specializes in the joys and tears of developing software for embedded systems. He is currently working as a software consultant at a conditional access systems provider in the Netherlands. Besides trying to convert more people to Linux and experimenting with his home network, he also enjoys reading and writing. He can be reached at [grimaldo@panama.IAEhv.nl](mailto:grimaldo@panama.IAEhv.nl).

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

[Advanced search](#)

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

## Writing a Linux Driver

**Fernando Matia**

Issue #48, April 1998

The main goal of this article is to learn what a driver is, how to implement a driver for Linux and how to integrate it into the operating system—an article for the experienced C programmer.

The concept of an operating system (OS) must be well understood before any attempt to navigate inside it is made. Several definitions are available for an OS:

1. An OS is the set of manual and automatic procedures which allow a set of users to share a computing system in an efficient manner.
2. The dictionary defines an OS as a program or set of programs which manage the processes of a computing system and allow the normal execution of the other jobs.
3. The definition from the Tanenbaum book (see Resources): An operating system is [the program] which controls all the resources of the computer and offers the support where users can develop application programs.

It is also very important to clearly distinguish a *program* from a *process*. A program is a block of data plus instructions, which is stored in a file on disk and is ready to be executed. On the other hand, a process is an image in memory of the program which is being executed. This difference is highly important, because usually the processes are running under OS control. Here, our program is the OS, so we cannot speak about processes.

We will use the term *kernel* to refer to the main body of the OS, which is a program written in the C language. The program file may be named `vmlinuz`, `vmlinux` or `zImage`, and has some things in common with the MS-DOS files `COMMAND.COM`, `MSDOS.SYS` and `IO.SYS`, although their functionality is different. When we discuss *compilation* of the kernel, we mean that we will edit the source files in order to generate a new kernel.



Peripheral or internal *devices* allow users to communicate with the computer. Examples of devices are: keyboards, monitors, floppy and hard disks, CD-ROMs, printers, mice (serial/parallel), networks, modems, etc. A *driver* is the part of the OS that manages communication with devices; thus, they are usually called *device drivers*.

### What is a Driver?

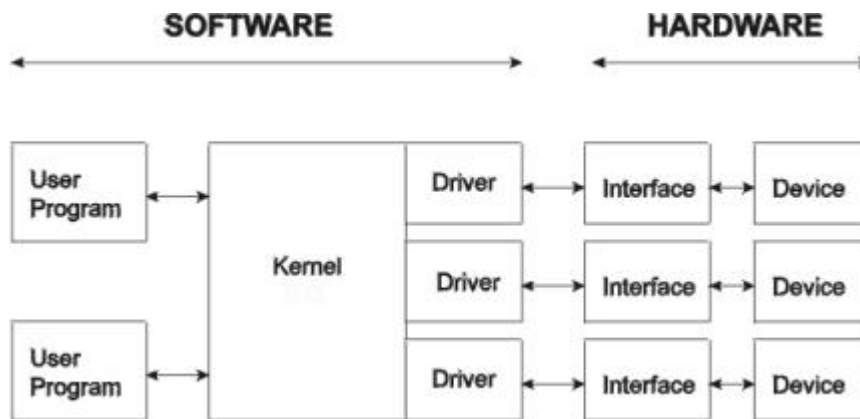


Figure 1. Software/Hardware Scheme

Figure 1 shows the relation between user programs, the OS and the devices. Differences between software and hardware are clearly specified in this scheme. At the left side, user programs may interact with the devices (for example, a hard disk) through a set of high-level library functions. For example, we can open and write to a file of the hard disk calling the C library functions **fopen**, **fprintf** and **fclose**:

```
FILE *fid=fopen("filename", "w");
fprintf(fid, "Hello, world!");
fclose(fid);
```

The user can also write to a file (or to another device such as a printer) from the OS shell, using commands such as:

```
echo "Hello, world!" >
echo "Hello, world!" > /dev/lp
```

To execute this command, both the shell and the library functions perform a call to a low level function of the OS, e.g., **open()**, **write()** or **close()**:

```
fid = open("/dev/lp", O_WRONLY);
write(fid, "Hello, world!");
close(fid);
```

Each device can be referred to as a special file named `/dev/*`. Internally, the OS is composed of a set of drivers, which are pieces of software that perform the low-level communication with each device. At this execute level, the kernel calls driver functions such as **lp\_open()** or **lp\_write()**.

On the right side of Figure 1, the hardware is composed of the device (a video display or an Ethernet link) plus an interface (a VGA card or a network card). Finally, the device driver is the physical interface between the software and the hardware. The driver reads from and writes to the hardware through ports (memory addresses where the hardware links physically), using the internal functions `out_p` and `in_p`:

```
out_p(0x3a, 0x1f);  
data = in_p(0x3b);
```

Note that these functions are not available to the user. Since the Linux kernel runs in protected mode, the low memory addresses, where the ports addresses reside, are not user accessible. Functions equivalent to the low-level functions `in` and `out` do not exist in the high-level library, as in other operating systems such as MS-DOS.

### Features of a Driver

The main features of a driver are:

- It performs input/output (I/O) management.
- It provides transparent device management, avoiding low-level programming (ports).
- It increases I/O speed, because usually it has been optimized.
- It includes software and hardware error management.
- It allows concurrent access to the hardware by several processes.

There are four types of drivers: character drivers, block drivers, terminal drivers and streams. *Character drivers* transmit information from the user to the device (or vice versa) byte per byte (see Figure 2). Two examples are the printer, `/dev/lp`, and the memory (yes, the memory is also a device), `/dev/mem`.

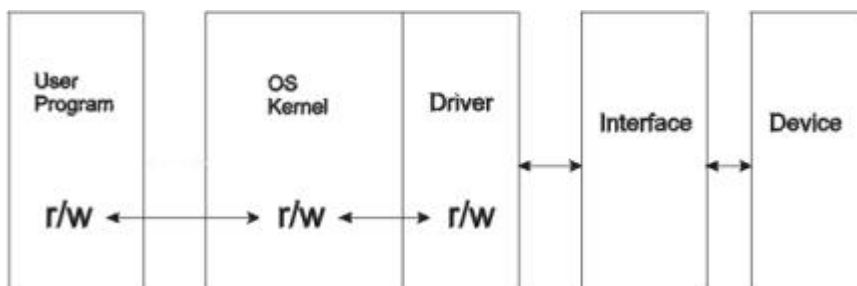


Figure 2. Character Drivers

*Block drivers* (see Figure 3) transmit information block per block. This means that the incoming data (from the user or from the device) are stored in a buffer until the buffer is full. When this occurs, the buffer content is physically sent to the device or to the user. This is the reason why all the printed messages do

not appear in the screen when a user program crashes (the messages in the buffer were lost), or the floppy drive light does not always turn on when the user writes to a file. The clearest examples of this type of driver are disks: floppy disks (/dev/fd0), IDE hard disks (/dev/hda) and SCSI hard disks (/dev/sd1).

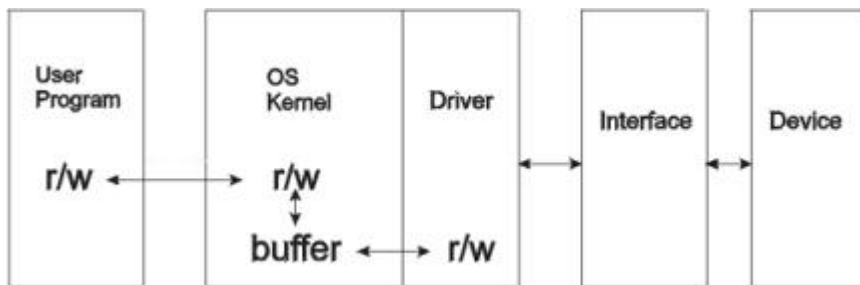


Figure 3. Block Drivers

*Terminal drivers* (see Figure 4) constitute a special set of character drivers for user communication. For example, command tools in an open windows environment, an X terminal or a console, are devices which require special functions, e.g., the up and down arrows for a command buffer manager or tabbing in the bash shell. Examples of block drivers are /dev/tty0 or /dev/ttya (a serial port). In both cases the kernel includes special routines, and the driver special procedures, to cope with all particular features.

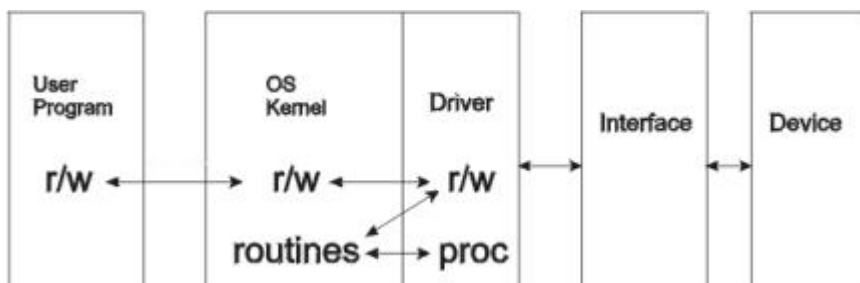


Figure 4. Terminal Drivers

*Streams* are the youngest drivers (see Figure 5) and are designed for very high speed data flows. Both the kernel and the driver include several protocol layers. The best example of this type is a network driver.

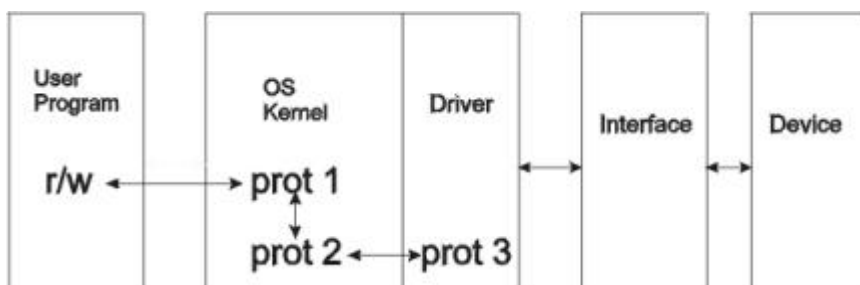


Figure 5. Stream Drivers

As we have said, a driver is a piece of a program. It is composed of a set of C functions, some of which are mandatory. For example, for a printer device, some typical functions only called by the kernel, may be:

- **lp\_init()**: Initializes the driver and is called only at boot time.
- **lp\_open()**: Opens a connection with the device.
- **lp\_read()**: Reads from the device.
- **lp\_write()**: Writes to the device.
- **lp\_ioctl()**: Performs device configuration operations.
- **lp\_release()**: Interrupts connection with device.
- **lp\_irqhandler()**: Specific functions called by the device to handle interrupts.

Some additional functions are available for particular applications, like **\*\_lseek()**, **\*\_readdir()**, **\*\_select()** and **\*\_mmap()**. You may find more information about them in Michael Johnson's *Hacker's Guide* (see Resources).

### **Do I Really Need to Write a Driver?**

There are several reasons for writing our own device driver:

- To solve concurrency problems when two or more processes try to access a device at the same time.
- To use hardware interrupts: as the kernel runs in protected mode, the user cannot manage interrupts directly from a program.
- To handle other unusual applications, such as managing a virtual device (a RAM disk or a device simulator).
- To obtain satisfaction as a programmer: writing a driver increases personal motivation as well as control over the computer.
- To learn about the internal parts of the system.

Conversely, there are also several reasons for not writing our own driver:

- It requires a good deal of mental preparation.
- It requires low-level programming, i.e., direct management of ports and interrupt handlers.
- In the debug process, the kernel hangs easily, and it is not possible to use debuggers or C library functions such as **printf**.

In order to understand the following explanation, you must know the C programming language, the basic I/O procedures, a minimum about the internal architecture of a PC and have some experience in the development of software applications for Unix systems.

Finally, we must add that writing our own device driver is only necessary when the device manufacturer does not supply a driver for our OS or when we wish to add extra functionality to the one we have.

### **An Actual Example of a Driver**

The first question we answer is: why use Linux as an example of how to write a driver? The answer is twofold: all the source files are available in Linux, and I have a working example at my lab in UPM-DISAM, Spain.

However, both the directory structure and the driver interface with the kernel are OS dependent. Indeed small changes may appear from one version or release to the next. For example, several things changed from Linux 1.2.x to Linux 2.0.x, such as the prototypes of the driver functions, the kernel configuration method and the Makefiles for kernel compilation.

The device we have selected for our explanation is the MRV-4 Mobile Robot from the U.S. company Denning-Brach International Robotics. Although the robot uses a PC with a specific board for hardware interfacing (a motor/sonar card), the company does not supply a driver for Linux. Nevertheless, all the source files of the software, which control the robot through the motor/sonar card, are available in C language for MS-DOS. The solution is to write a driver for Linux. In the example, we use kernel release 2.0.24, although it will also work in later versions with few modifications.

The mobile platform is composed of a set of wheels coupled with two motors (the drive and the steer), a set of 24 sonars which act as proximity sensors for obstacle detection and a set of bumpers which detect collisions. We need to implement a driver with, at least, the following services (**init**, **open** and **release** are mandatory):

- **write:** to send linear and angular velocity commands
- **read:** to read sonar measures and encoder values
- **three interrupt handlers:** to store sonar measures when a sonar echo is received, to implement an emergency stop when a bumper detects a collision and to stop the steer motor when the wheels are located at 0 (zero) degrees and a *go to home* flag is active
- **ioctl commands:** *go to home* which sends a constant angular velocity to the wheels and activates the *go to home* flag; and configuration of motors and sonars

The *go to home* service allows the user to stop the wheels at an initial position which is always the same (0 degrees). The incoming values from sonars and

encoders, as well as the velocity commands, might be part of the main loop of the control program of the robot.

Returning to the initial scheme (Figure 1), the device is the MRV-4 robot, the hardware interface is the motor/sonar card, the source file of the driver will be `mrv4.c`, the new kernel we will generate will be `vmlinuz`, the user program for kernel testing will be **`mrv4test.c`** and the device will be `/dev/mrv4` (see Figure 6).

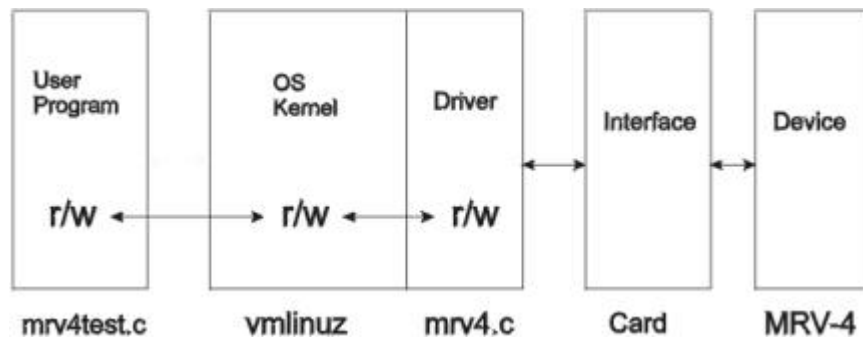


Figure 6. `mrv4hard` MRV-4 Scheme

### General Programming Considerations

To build a driver, these are the steps to follow:

1. Program the driver source files, giving special attention to the kernel interface.
2. Integrate the driver into the kernel, including in the kernel source calls to the driver functions.
3. Configure and compile the new kernel.
4. Test the driver, writing a user program.

The directory structure of the Linux source files can be described as follows: the `/usr/src` contains subdirectories such as `/xview` and `/linux`. Inside the `/linux` directory, the different parts of the kernel are classified into subdirectories: `init`, `kernel`, `ipc`, `drivers`, etc. The directory `/usr/src/linux/drivers/` contains the driver sources, classified into categories such as `block`, `char`, `net`, etc.

Another interesting directory is `/usr/include`, where the main header files, such as `stdio.h`, are located. It contains two special subdirectories:

- `/usr/include/system/`, which includes system header files, such as `types.h`
- `/usr/include/linux/`, which includes the Linux kernel headers such as `lp.h`, `serial.h`, `mem.h` and `mr4.h`.

The first task when programming the source files of a driver is to select a name to identify it uniquely, such as `hd`, `sd`, `fd`, `lp`, etc. In our case we decided to use `mr4`. Our driver is going to be a character driver, so we will write the source

into the file `/usr/src/linux/drivers/char/mrv4.c`, and its header into `/usr/include/linux/mrv4.h`.

The second task is to implement the driver I/O functions. In our case, **`mrv4_open()`**, **`mrv4_read()`**, **`mrv4_write()`**, **`mrv4_ioctl()`** and **`mrv4_release()`**.

Special care must be taken when programming the driver because of the following limitations:

- Standard library functions are not available.
- Some floating-point operations are not available.
- Stack size is limited.
- It is not possible to wait for events, because the kernel, and so all the processes, are stopped.

The OS functions supported at kernel level are, of course, only those functions programmed inside it:

- **`kmalloc()`**, **`kfree()`**: memory management
- **`cli()`**, **`sti()`**: enable/disable interrupts
- **`add_timer()`**, **`init_timer()`**, **`del_timer()`**: timing management
- **`request_irq()`**, **`free_irq()`**: irq management
- **`inb_p()`**, **`outb_p()`**: port management
- **`memcpy_*fs()`**: data management
- **`printk()`**: input/output
- **`register_*dev()`**, **`unregister_*dev()`**: device management
- **`*sleep_on()`**, **`wake_up*`**: process management

Detailed information on these functions is given in Johnson's *Guide* (see Resources) or even inside the kernel source files.

### **Low-Level Programming**

Access to the hardware interface (the card) is provided through low-memory addressing. The I/O registers of the card, where we can read/write information, are physically connected to memory addresses of the PC (i.e., ports). For instance, the motor/sonar card of the MRV-4 mobile robot is associated with the address **`0x1b0`**. Sixteen registers are used in this card, so the port map includes addresses from **`0x1b0`** to **`0x1be`**. A typical list of port addressing is shown in Table 1.:

#### **Table 1. Typical Port Addresses**

A free address region must be found to allocate the ports for the new card. In Table 1, addresses from **1b0** to **1be** were free. The source code example, `foo.c`, is available on the SSC FTP site (see end of article) and includes a call to a system function that allows us to see the previous table of addresses. Finally, access to the ports is granted via the functions `inb_p` and `outb_p`.

Interrupts are the other main topic when talking about low-level programming and hardware control. Interrupt handling versus polling has the main advantage that hardware is usually slow. We cannot stop all processes in a computer until a printer finishes a job. Instead, we can continue with normal work until the printer finishes, then send an interrupt signal that is handled by a specific function.

Continuing with our example, we need three handlers, one for each of the hardware interrupts that the card can generate: sonars handler (at irq **0x0a**), home handler (at irq **0x0b**) and bumper handler (at irq **0x0c**). As example of what the source code must do, we show the structure of the `sonar_irq_hdlr` function. Each time an echo from a sonar is received, it must:

1. Disable hardware interrupts.
2. Read sonar value from its port and store it in a driver internal variable.
3. Enable interrupts again.

If a user program wants to read the incoming data from the sonars, it must perform a `mrV4_read` operation, which returns the data stored in the internal variables of the driver.

### Implementation of Driver Functions

Although we will explain the guidelines to implement each of the driver functions, when programming your own driver it is a good idea to use the driver most similar to yours as an example. In our case, the models for `mrV4.c` and `mrV4.h` are `lp.c` and `lp.h`, respectively.

The file `mrV4.c` includes the initialisation and I/O functions. The initialisation function `mrV4_init` must follow these steps (see guidelines in file `foo.c`):

1. Check in the device.
2. Get a free region for port addressing.
3. Test if hardware is present.
4. Test if irq numbers are free.
5. Initialise driver internal variables.
6. Return an OK status.



If an error is detected in any of these steps, it must undo all previous operations and return an error status. To implement the I/O functions, the following structure (or similar) must be defined and initialized in `mrsv4.c`:

```
static struct file_operations mrsv4_fops = {
    NULL, /* mrsv4_lseek */
    mrsv4_read, /* mrsv4_read */
    mrsv4_write, /* mrsv4_write */
    NULL, /* mrsv4_readdir */
    NULL, /* mrsv4_select */
    mrsv4_ioctl, /* mrsv4_ioctl */
    NULL, /* mrsv4_mmap */
    mrsv4_open, /* mrsv4_open */
    mrsv4_release /* mrsv4_release */
};
```

Pointers to all existent I/O functions must be set in this structure. Then, the I/O function code can be implemented, following the guidelines shown in the sidebar.

The available commands are defined in the file `mrsv4.h` (see guidelines in file `foo.h` also available on the FTP site):

```
#define MRV4_MAGIC, 0x07
#define MRV4_RESET _IO(MRV4_MAGIC, 0x01)
#define MRV4_GOTOHOME _IO(MRV4_MAGIC, 0x02)
#define MRV4_RESETHOME _IO(MRV4_MAGIC, 0x03)
#define MRV4_JOYSTICK _IOW(MRV4_MAGIC, 0x04,
    unsigned int)
#define MRV4_PREPMOVE _IOW(MRV4_MAGIC, 0x05,
    unsigned int)
#define MRV4_INITODOM _IO(MRV4_MAGIC, 0x06)
#define MRV4_SONTOFIRE _IOW(MRV4_MAGIC, 0x07,
    unsigned int)
```

The `_IO` macro is used for commands without arguments. The `_IOW` is used for commands with input arguments. In this case, the macro needs the argument type, for example a pointer might be of type **unsigned int**. The magic number must be chosen by the programmer. Try to select one not reserved by the system (see other header files at `/usr/include/linux`). Constants are defined in the file `/usr/include/linux/mrsv4.h`, which must be included by both the driver (`mrsv4.c`) and the user programs. In general, the `mrsv4.h` file can include:

- Constants and macros definitions
- ioctl commands
- Port names
- Type definitions
- Data structures to be exchanged between the driver and the user
- `mrsv4_init()` function prototype

## Driver Integration in the Kernel

The task of integrating the driver into the kernel includes several steps:

- Insert kernel calls to the new driver.
- Add the driver to the list of drivers.
- Modify compilation scripts.
- Re-compile the driver.

The insertion of the OS call to `mrsv4_init()` is done in the `/usr/src/linux/kernel/mem.c` file. The other driver function calls (`open`, `read`, `write`, `ioctl`, `release`, etc.) are user transparent. They are carried out through the `file_operations` structure. A driver major number must be added to the list located at `/usr/include/linux/major.h`. Search for a free driver number; for example, if number 62 is free, you must add one or both of the following lines to the file, depending on the Linux release:

```
/dev/mrv4 62
#define MRV4_MAJOR 62
```

Each device is referenced by one major and one minor number. The major number represents the number of the driver. The minor number distinguishes between several devices which are controlled by the same device (e.g., several hard disks controlled by the same IDE driver: `hd0`, `hd1`, `hd2`).

The next step is to create a logical device to access the driver. You must use the command **mknod** in this way:

```
mknod -m og+rw /dev/mrv4 c 62 0
```

where **62** is the major, **0** the minor (only one physical device) and **c** indicates a character device. Set the permissions as necessary, although you can modify them later with the command **chmod**. For example, enable **rw** if you want to allow all users to access the device:

```
crw-rw-rw-2 bin bin 62, 0 Mar 12 1997 /dev/mrv4
```

## Driver Compilation and Testing

To allow driver compilation within the kernel, the following lines must be added to the script file `/usr/src/linux/arch/i386/config.in`:

```
comment 'MRV 4'
bool 'MRV 4 card support' CONFIG_MRV4
```

and the following lines to `/usr/src/linux/drivers/char/Makefile`:

```
ifdef CONFIG_MRV4
    L_OBJS += mrv4.o
endif
```

It is recommended that the driver be compiled alone, before linking the kernel. This method will save time testing syntax errors:

```
cd /usr/src/linux/drivers/char
gcc -c mrv4.c -Wall -D__KERNEL__
```

And when all is well, delete the object file:

```
rm -f mrv4.o
```

Next, configure the kernel by typing:

```
cd /usr/src/linux
make config
```

Answer **yes** when the script asks you about installing the MRV-4 driver (this sets the constant **CONFIG\_MRV4**). Finally, insert an empty floppy disk and re-build the kernel by typing the following commands:

```
make zdisk    # generate a bootable
# floppy disk
dev -R /dev/fd0 1  # disable writes to<\n>
# floppy
```

Once you are sure that the kernel works, you can overwrite the file `vmlinuz` with the new kernel. To test the new kernel, restart the system (type **reboot**) and ... good luck! There are no debuggers available.

If the kernel seems to work, you might test the driver by writing one or more user programs, i.e., `mrv4test.c` which call the driver functions:

```
fid = open("/dev/mrv4", ...);
read(fid, ...);
write(fid, ...);
ioctl(fid, ...);
close(fid);
```

### How to Obtain Additional Information

You can obtain privileged documentation at [sunsite.unc.edu](http://sunsite.unc.edu) (see Resources). But of course, you will never be able to write your own driver using only the general guidelines of this article. To facilitate this task, we supply the source files for a dummy driver for Linux 2.0.24, which is a model for character driver development. It simulates the equation  $y = ax$  and includes an example of interrupt management (which does not work since it is not associated with any hardware). Its name is `foo`, since Linux already has a driver called `dummy`. These files are:

- **README**: summary of instructions to install it
- **foo.c**: driver source file

- **foo.h**: driver header file
- **footest.c**: program for driver testing

You can obtain these files via anonymous FTP at <ftp://ftp.linuxjournal.com/pub/lj/listings/issue48/2476.tgz>.

## Guidelines

## Resources

Fernando Matía is an Associate Professor at the Universidad Politecnica de Madrid (UPM). He was born in Madrid, Spain, in 1966. He became an Industrial Engineer at UPM in 1990 and received his Ph.D. degree at UPM in 1994 in the area of Control Engineering. He works at the Systems and Automatic Control Engineering Division (DISAM). His main activities are Intelligent Control, Fuzzy Control, Robotics and Computer Sciences. He can be reached at [matia@disam.upm.es](mailto:matia@disam.upm.es).

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

[Advanced search](#)

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

[Advanced search](#)

## Configuring procmail with The Dotfile Generator

**Jesper Pedersen**

Issue #48, April 1998

Here's a follow-up article on TDG to show you the easy way to configure the procmail program.

In this article, I'll describe how to configure procmail using The Dotfile Generator (TDG for short). This will include:

- How to sort mail coming from different mailing lists
- How to setup an auto reply filter when you are on vacation
- How to change some part of a letter, e.g., remove the signature
- How to avoid lost mail

If you haven't downloaded the program, now is the time to do it. The home page of TDG at <http://www.imada.uo.dk/~blackie/dotfile/> provides a list of mirror sites. An earlier article, giving details about the program and the installation of TDG, appeared in *Linux Journal* October 1997, Issue 42.

### Starting TDG

To start TDG with the procmail module, type **dotfile<!\s>procmail**, and the window shown in Figure 1 will appear. As you can see, the module is split into three pages. The first two are very simple, so let's start with the page called General Setup. This page is shown in Figure 2.

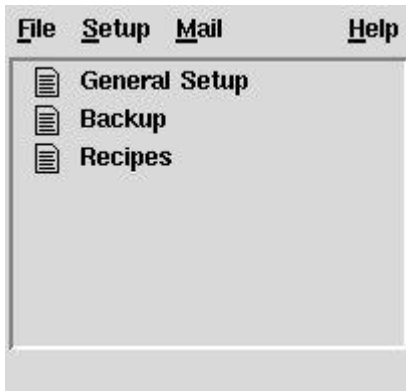


Figure 1. TDG procmail Window

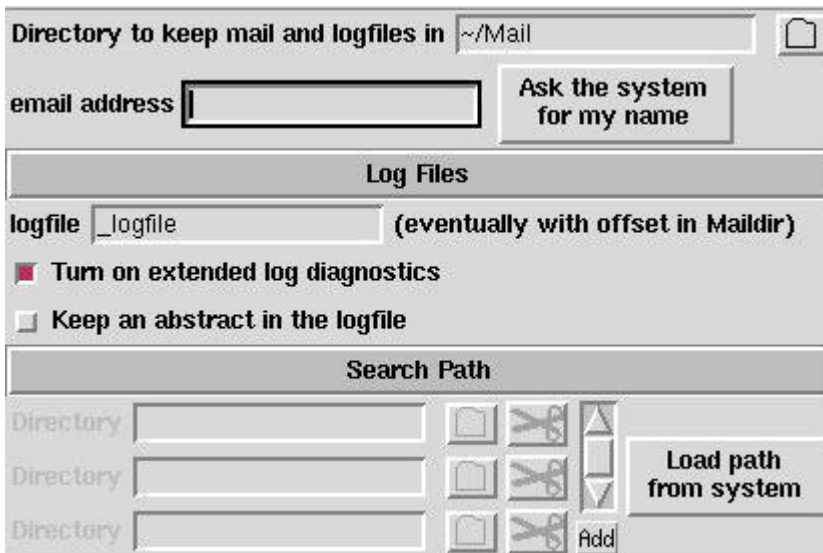


Figure 2. General Setup Page

On this page there are four items to configure:

1. The directory to use as prefix for all file operations. This is simply for convenience, since all file operations may be given using the file's full prefix.
2. Your e-mail address, which is used in preventing loop-backs.
3. Configuration of log files. These files are very useful when you wish to investigate mail destinations. If you turn on *abstract logging*, you may find the program **mailstat** very useful. (See the log file below.)
4. The search path, in which procmail may find the programs it needs. Note this is only the programs, which you specify in filters etc.

### Avoiding Lost Mail

Since procmail handles your incoming mail, security is very important to this module. You can back up your incoming mail in three different ways. To do this, go to the page called Backup shown in Figure 3.

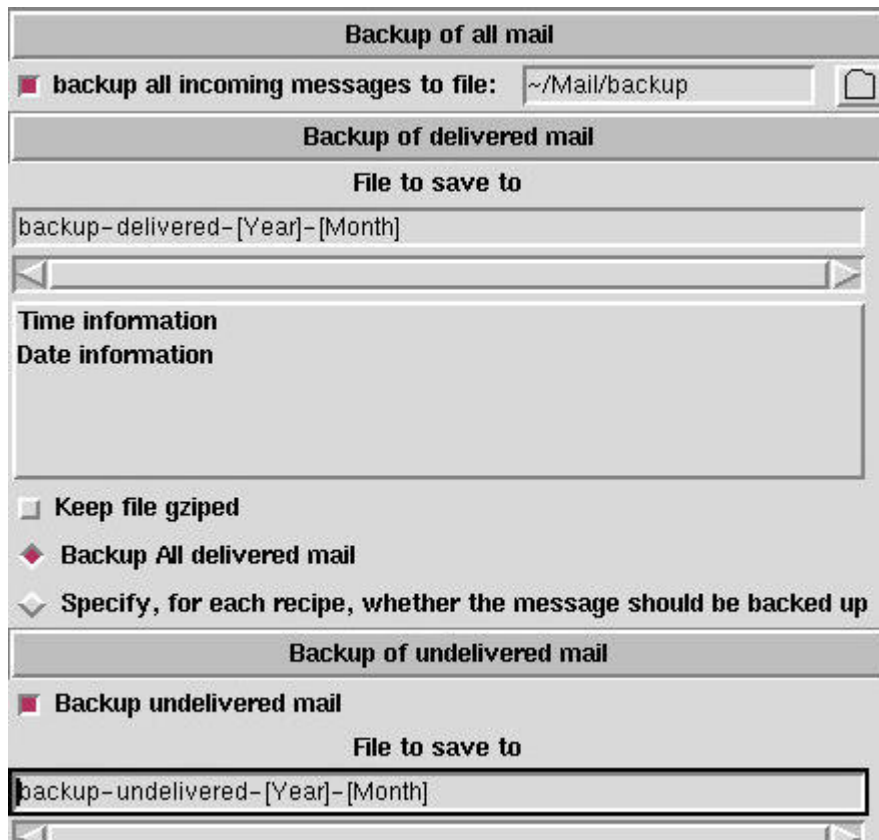


Figure 3. Backup Page

The first backup category is “back up all incoming mail”. The code needed by the procmailrc file to do this is typed in the very first field. This is done to avoid any errors in the generated procmail file that might cause any of your mail to be thrown away. This sort of backup is only a good idea when you first start to use the generated procmail file. The main drawback is that all incoming mail is saved in one file, so this file can become very big, very quickly.

The second method is to backup all incoming mail that is delivered by procmail. This method makes it easy to verify that mail is sorted into the right places.

The third method is to back up all mail that makes it to your incoming mailbox. This mail is often personal mail; that is, it did not come from a mailing list, and it is not junk mail.

In the first method, the full file name must be specified. This is because this method has to be 100% foolproof. In the other two methods, you may build the file names from the current date and time. This makes it possible to save this sort of mail to folders for the current year/month/week, e.g., a folder called backup-delivered-1997-July. As an additional feature, you may compress the files as gzipped files.

The backup of delivered mail can be specified for each individual recipe or for all recipes at once. (See Figure 4, check box 9.) The FillOut elements, which configure the saved file are discussed in the previous article.

### Setting Up the Recipes

In procmail a central concept is a recipe, which is a set of conditions and a set of actions. All of the actions are executed if all of the conditions are fulfilled. A few examples of conditions appear below:

- The letter comes from president@white.house.com.
- The subject is subscribe.
- The size of the letter is greater than 1MB.
- The letter contains text.

A list of actions includes:

- Reply to the sender that you are on holiday.
- Forward the letter to another person.
- Save the letter to a file.
- Change some part of the letter (e.g., add a new header field or add some text to it).

A procmail configuration is a sequence of recipes. When a letter arrives, each recipe is checked to see if all of its conditions are fulfilled. If they are, the actions of the recipe are executed.

Procmail will finish testing recipes when one is matched, unless a flag is set to tell it that this recipe should not stop delivery (see Figure 4 check box 8). This means that the order of the recipes is important, since only the first recipe to match will process the letter.

If none of the recipes are fulfilled, or if the ones which are fulfilled have check box 8 in Figure 4 set, the letter is delivered to the incoming mailbox as if the procmail filter did not exist.

You configure the recipes on the page called "Recipes". This page can be seen in Figure 4.



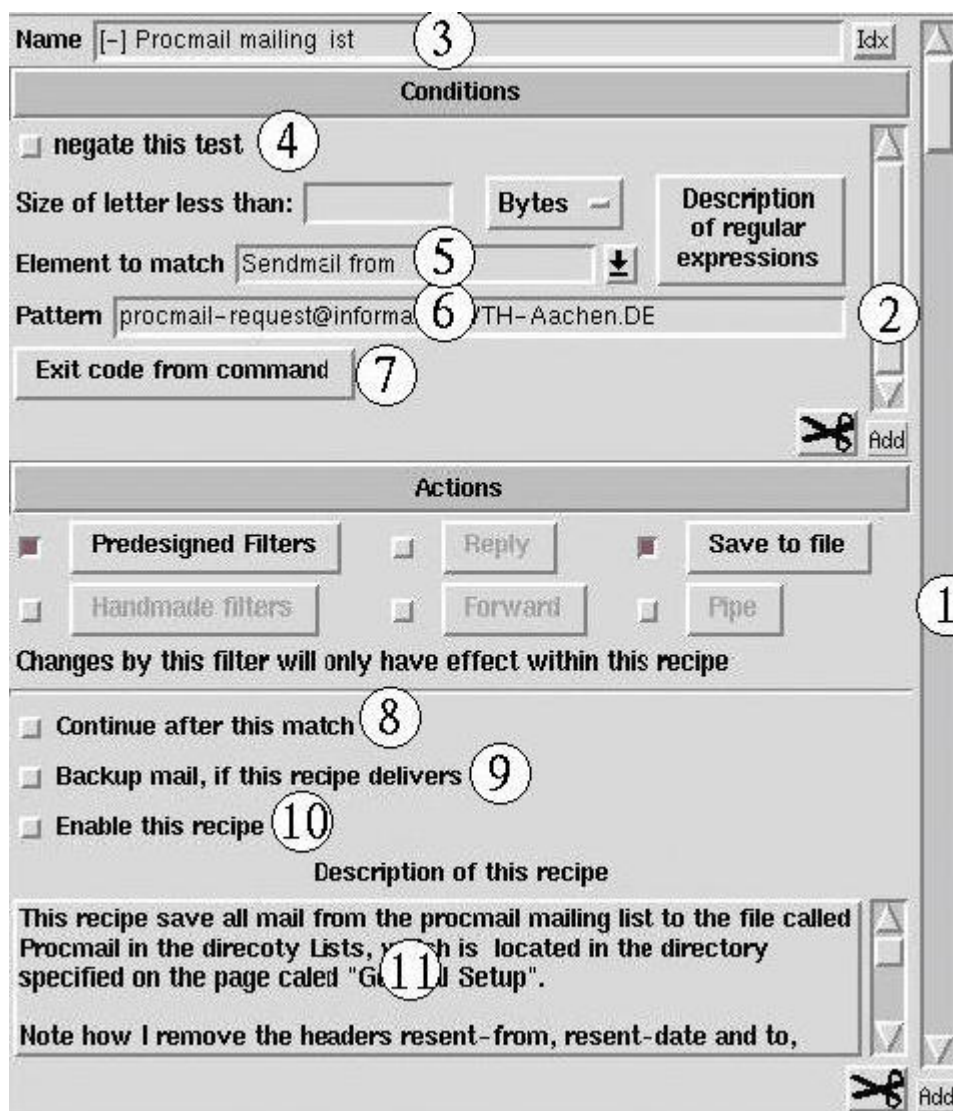


Figure 4. Recipe Page

What you see here is an ExtEntry. An ExtEntry is a widget, which repeats its elements as many times as necessary (just like a list box repeats the labels). Everything on this page is one single recipe. To see a new recipe, you have to scroll the outer scroll bar (1). To add a new recipe, you have to press the Add button beneath the scroll bar.

As described above, a recipe is a set of conditions. This set is also represented with an ExtEntry (2). To scroll to another condition in a recipe you must use the scroll bar (2), and to add a new condition you must use the button below scroll bar (2).

Each recipe can be given a unique name to make it easier to find a given recipe. This name is also written to the file with mail delivered by recipes (method 2 above), so that you can see which recipe matched the actual letter. To give a recipe a name, use entry (3). A button labeled Idx is located at the right side of the entry. This is a quick index to the outer ExtEntry (i.e., the recipes). If you

press this button, a list box drops down from which you can select one of the recipes by name.

### The Conditions of a Recipe

The most common condition to set up is to match one of the header fields with a given regular expression, or to match the body of the letter with a given regular expression. A typical header is shown in [Listing 1](#).

The very first line of the header is special and has been written by the program (often **sendmail**) sending the letter. This header field is always the same for a given mailing list, so to sort mail from a mailing list, it is a good idea to view the letter with an ordinary file reader (a mail reader will seldom show this line). Copy this information to the pattern field (Figure 4, label (6) ). As the element to match, you have to select "**Sendmail from**" in entry (5).

Three special macros exist in procmail. These may be used when matching header fields:

1. TO: This macro matches every destination specification.
2. FROM\_DAEMON: This macro should match when the letter comes from a daemon (which includes mailing lists). It is useful to avoid creating a mail loop with some mailing lists.
3. FROM\_MAILER: Another regular expression, which should produce a match when the letter comes from the mail daemon.

To see what these macros stand for, please refer to the procmailrc man page.

There are many header fields to choose between in the pull down menu (5), but if the one you wish to select isn't located there, you may type it yourself.

The check box (4) may be used to negate the condition, i.e., if the pattern does not match, the condition is fulfilled.

### Regular Expressions

So far, I have mentioned that you can type a regular expression in box (6). In most cases, it may not be necessary to know anything about regular expressions since the procmail module will take care of most of it for you. One thing may be worth knowing, and that is that you may match "anything" with **".\*"**. This means that **abc.\*def** will match anything which starts with **abc** and ends with **def**, e.g., **abcdef** or **abcXXXXdef**. To read a more detailed description of the set of regular expressions used by procmail, press the button labeled **Description**.

One common pitfall is to forget to match everything at the start of the line. If you wish to set up a regular expression for the **From:** field above, it is not enough to give the pattern: **rick@helix.nih.gov**, since this is not at the start of the line. Instead, to tell procmail that every mail message which includes the text **rick@helix.nih.gov** is to be handled, insert **.\*** in front of the e-mail address.

### External Programs

A final way to set up a condition is by using an external program to verify some conditions. This is done by pressing button (7) which brings up a window with a FillOut field like the one in Figure 3. This time, however, the entry has been replaced with a text box. In this text box, you can type some commands to read either the header or the body on standard input. These commands can refer to any header fields from the letter. The lines (separated by a newline) are joined together with a separating semicolon, making each line a separate command.

Procmail will consider the condition fulfilled only if the exit code from the program is 0. This behavior can be changed with the check button (4) in Figure 4.

### Actions

The actions that this module can handle are split into six parts. These are described in detail below. To activate an action, you first have to select the check box that is located next to it, making it clear which actions are enabled for a given recipe.

#### 1. Predesigned Filters

To set up a filter, press the button labeled **Predesigned Filters** in the window. This filter can change the header fields, add new header fields and/or remove existing header fields.

On this page you will find one custom-made filter: **Remove signatures**. With this filter, you may specify a signature for each e-mail address. If the text you specify is found (exactly), it will be removed from the letter. My intention is to add more custom-made filters as users send me their ideas and filters.

#### 2. Handmade Filters

If you wish to create your own filter, you must go to the page **Handmade Filters**. On this page, you may send the header and/or the body of a letter through a command.

As an example, you may remove the header with the command

```
cat ->> /dev/null
```

or add a message to the body of a message with the command:

```
echo This letter has been resent to you, by my\  
procmail filter!; cat -
```

If only the filter action is selected, the filter will change the letter permanently, i.e. the changes will affect the subsequent recipes (even on the delivered letter, if no recipes match). This may be useful if you use a mail reader that does not support MIME, and you have a filter to convert MIME-encoded text to 7-bit ASCII. If, however, one of the other actions is also enabled, the changes affect only this recipe.

### 3. The Reply Action

With the reply action you can set up a reply mechanism, which sends a letter back to the sender with a message that you specify. One feature of this mechanism is that you can specify how often a reply should be sent. You have the following choices:

- Send a reply to each letter.
- Send a reply only once.
- Send a reply only if it is more than a given number of days since the last reply was sent.

This action is useful if you leave on vacation, and wish to send a message that you will not read your letter at once.

The reply is sent only if the letter does not come from a daemon, to avoid sending a reply to every message on a mailing list.

### 4. The Forward Action

With the forward action, you may forward letters to other e-mail accounts.

### 5. The Save to File Action

With this action, you can save the letter to a file. The file name is specified with a FillOut widget, just as you specified the name of a file for backups. This time, however, you have two additional features: you can use the content of a header field, or you can use the output from a command. In Figure 5, you can see how to select a header field to extract as part of the file name.

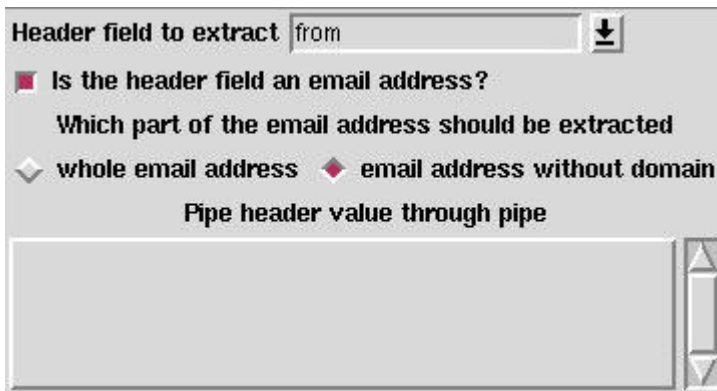


Figure 5. Header Field Selection

E-mail addresses can be specified in three ways:

- real name (e-mail)
- e-mail (real name)
- e-mail

If you specify that the field is an e-mail address, you may also specify whether you wish to extract the user name with or without the domain name.

Finally, you can pipe the header field through a specified command. This command can read the value of the header field on standard input and write to standard output.

## 6. The Pipe Action

With the pipe action, you can specify a command to take care of the letter. This command can read the letter on standard input but cannot write anything (it is ignored).

### The Log File

The procmail file generated from TDG contains lots of comments to make it easy for you to find a specific recipe.

Should something go wrong, you may turn on the extended diagnostic option. This will write additional lines to the log file to show you what it does. For debugging, you must read both the log file and the procmail file.

If you use the *log abstract* options, you will find the program mailstat very useful. It tells you how many letters have been delivered where. One line in the output from the mailstat programs is fake: **/bin/false**--it may be safely ignored. When you wish to delete a letter in a way that you can explicitly see that it has been removed, you should deliver it to the file called `/dev/null`. Please note that

you can only use the mailstat program if the extended diagnostic option is turned off.

## The End

Before procmail starts filtering all your incoming mail, you must add the following line (i.e., no break) to the file called ~/.forward:

```
"|IFS=' ' &&exec /usr/local/bin/procmail -f-||exit 75
```

with the correct path name for procmail, and **username** replaced by your e-mail address.

## Resources

## Acknowledgments



**Jesper Pedersen** lives in Odense, Denmark, where he has studied computer science at Odense University since 1990. He is a system manager at the University and also teaches computer science. In his spare time, he does Jiu-Jitsu, listens to music, drinks beer and has fun with his girlfriend. His home page can be found at <http://www.imada.ou.dk/~blackie/> and he can be reached via e-mail at [blackie@imada.ou.dk](mailto:blackie@imada.ou.dk).

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

[Advanced search](#)

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

[Advanced search](#)

## Best of Technical Support

### Various

Issue #48, April 1998

Our experts answer your technical questions.

### Using ZIP Drive and Printer

I want to use my ZIP drive as well as my printer with Linux. My friend suggested using **kernelcd** to make modules of the ZIP drive and my printer so that I can load the ZIP drive, access it, then unload it to use my printer. How would I go about doing this?

—Scott Bell Red Hat 4.2

There is no need to do this. All the modules you need are already available from the stock install. To access your parallel port ZIP drive simply run:

```
insmod ppa
```

*as root. You will then have your ZIP drive available as /dev/sda if you don't have any other SCSI devices on your system. When done with the ZIP, make sure everything is unmounted. Then run:*

```
rmmod ppa
```

*as root. You can then unplug it, plug your printer in and use the printer as you normally would (kernelcd should load that module automatically). If you have further questions or need help, install the kernel-source RPM and see /usr/src/linux/drivers/scsi/README.ppa and the SCSI-HOWTO for more details.*

—Donnie Barnes, Red Hat redhat@redhat.com

## Unwanted Displays

The gremlins at work are killing me. I have a few co-workers that keep sending me `xmelt`, `xroach` and `xsnow`. Unfortunately, I need to keep all my network connections open. Is there a way to find and kill processes that are sent to my display? Also, is it possible to reroute these processes back to the display they came from?

—Ray Banez Red Hat 4.2

Type `ps -ax` to get the process ID which is on the left side of the output, then type `kill -9 <process ID>`. Now to keep this from happening again make use of the `xhost` command. It will allow you to deny X sessions from hosts you don't want (`xhost -unwanted_machinename`).

—Mark Bishop, Vice President Southern Illinois Linux Users Group  
mark@vincent.silug.org

You should restrict access to your display. If you access your display locally you could just deny remote access to shell accounts on the workstation; otherwise, look at the docs about `xauth` and use it to authenticate graphic programs. This way only your own programs will be able to use your graphic display. If you share your account with other people, on the other hand, there's no solution to the problem.

Routing the processes back to the display they came from is not possible. You must block intruders before they get in or kill them afterwards.

—Alessandro Rubini rubini@linux.it

## kcore File

What is the `kcore` file in `/proc` directory? The file size is growing out of control. Can I delete it?

—Kai Lien, Pharm.D. Red Hat 4.1

This is a “virtual” file, reflecting your memory. It doesn't exist at all on your hard disk, like all of `/proc`, so it can't and shouldn't be deleted!

—Ralf W. Stephan ralf@ark.franken.de

Perhaps a short explanation of the `/proc` file system is in order. The `/proc` file system is a “virtual” file system. It doesn't actually reside on any sort of physical device. `/proc` is a means of examining what is going on inside the Linux kernel



without having to resort to a lot of programming. **/proc/kcore** is actually all of the memory that is in use on your system. Even if you could delete it, you wouldn't want to. Don't worry about the "size" of kcore. It's actually not affecting any of your drives.

I strongly encourage you to explore the files in the /proc file system, preferably as a non-root user. By looking inside these files, you can learn a lot about how your system is configured, what it is doing and how certain things work. As long as you aren't poking around as root, it's very difficult to mess anything up.

—Keith Stevenson k.stevenson@louisville.edu

### **Migrating to Linux**

I am currently trying to migrate an ISP's radius authentication server from FreeBSD to Linux (not distribution specific, but using Debian). The /etc/passwd file from FreeBSD is using MD5 encryption. The default scheme for Linux is the DES-like scheme. FreeBSD states, correctly, that the scheme may be switched to MD5 by changing the sym-links in /usr/lib from libcrypt to libcrypt. I cannot find a solution of this nature for any Linux distribution, though at this level there should not be any distribution specificity. I know of at least one ISP with a similar problem between BSD and Linux. I am not alone.

—Michael Roark Generic

Transferring passwords between different operating systems can be a major problem. The difficulty lies in the fact that passwords are encrypted in a one-way fashion. You have to know the password in order to decrypt it. Here is what I have done in a similar situation.

- 1) Write a program wrapper around your login program that will capture the userid and password of your users before passing the information to your authentication server.
- 2) Use this file of clear-text userid and password combinations to set up the authentication database on the new system.

If you are unable or unwilling to do this, find out whether or not freeBSD and Linux use the same crypt function for storing passwords. If so, set freeBSD to use the crypt function instead of MD5. Accelerate the rate of password expirations so all of your users have to change their passwords (assuming you use password expiration). After all of the passwords have been changed, simply copy the encrypted passwords from the freeBSD box to the appropriate place on the Linux box. This will not work unless freeBSD and Linux use the same crypt function.

You may also want to take a look at Red Hat Linux. I think that the PAM security system bundled with it may support MD5 passwords. If so, you can copy them directly.

—Keith Stevenson k.stevenson@louisville.edu

### **minicom Configuration Problem**

My machine is a p75 with a Fujitsu hard drive and a Hayes Accura 56kflex external modem. I have 4 partitions: MS-DOS, Caldera, Red Hat and swap. I have a mouse on com1 and a modem on com2. The modem works fine in DOS. In Linux, when I go into X and start **seyon**, an AT returns “ok” and I can dial out and get a response from a BBS or my ISP.

When I go into minicom, it does not dial out. I wait for a couple of retries and then exit minicom. As soon as I exit, minicom dials out.

If I open minicom in a vt the same thing happens. In all cases the modem init light comes on, but there is no dialing until I exit from minicom.

Can someone please tell me where I am going wrong? No one has been able to help—not Hayes, not the Caldera help line, not any of the lists where I posed the question. Any suggestions will be much appreciated.

—Ted Wager

Ok, you have a problem with minicom's config somewhere. Now I could be more helpful if I knew if an init string came up or just the familiar “Press **ctrl-A Z** for help on special settings.” My advice to you is go through the settings and make sure minicom is set up as it should be. (Usually it is set to /dev/modem assuming /dev/modem is a symbolic link to /dev/cua1.) Check your serial port settings and check your init string.

—Mark Bishop, Vice President Southern Illinois Linux Users Group  
mark@vincent.silug.org

### **syslogd messages**

On my home PC, the /var/adm/messages file was getting pretty big. I deleted it, then created a new one with **touch**. The new file's permissions are identical to the original.

But now, /usr/sbin/syslogd will not run for more than about two minutes. No data is logged to /var/adm/messages anymore. What have I done?

—Bill Cunningham Slackware

**syslogd** sometimes doesn't like it if the files it has open for writing are modified. Sending SIGHUP (**kill -HUP**) is usually enough to make it start writing to the file again.

Incidentally, I use the program **logrotate** to manage my syslog files. It trims them as necessary, archives old files and restarts syslogd as needed after working on the log files. It has made log file management much easier for me.

—Keith Stevenson k.stevenson@louisville.edu

### **Multi-threaded Applications**

I want to program multi-threaded applications. Kernel threads are available, I know. But how about thread-safe versions of much used libraries (libc and tcpip communications)?

—Peter Boncz Generic

You can try the LinuxThreads library, which is a free, kernel-level implementation of POSIX 1003.1c threads under Linux (based on the **clone** system call). For information about compatible libraries, check out <http://pauillac.inria.fr/~xleroy/linuxthreads/>.

—Pierre Ficheux, Lectra Syst pierre@rd.lectra.fr

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

[Advanced search](#)

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