Linux Journal Issue #6/October 1994

Features

News & Articles

Reviews

Product Review Motif 1.2.3 Runtime and Development System *by Dale A. Lutz*
Product Review Unix Interactive Tools *by Clarence Smith, Jr.*
Product Review Crisp Text Editor *by Robert Broughton*

Columns

Letters to the Editor
Stop the Presses
New Products

Archive Index

Advanced search

Messages - A Multi-Media Mailer

Terry Gliedt

Issue #6, October 1994

In this article, Terry Gliedt continues his tour through the Andrew project—this time into a new area, multi-media mail.

Introduction

AUIS had its roots in 1982 when Carnegie Mellon University and the IBM Corporation decided to jointly develop a campus computing facility based on personal computers to replace the time-sharing system then on campus. IBM provided not only generous funding, but also some talented individuals and access to IBM development programs.

The result was a graphical user interface we know as the Andrew User Interface System and a file system, the Andrew File System (www.transarc.com/Product/AFS/FAQ/faq.html). The file system formed the basis of Transarc Corporation's (http://www.transarc.com/) *Distributed File System* (DFS) and is as part of the Open System Foundation (www.osf.org) software.

The Andrew Consortium (www.cs.cmu.edu/afs/cs.cmu.edu/project/atk-ftp/web/andrew-home.html), composed of a number of corporations and universities, funds the current development of AUIS. AUIS is available on a wide variety of platforms including Linux, AIX, Solaris, Ultrix, HP UX as well as others.

In early June, version 6.3 of the Andrew User Interface System (AUIS) was released by the Andrew Consortium. This led to the release of auis63L?-wp.tgz (ftp://ftp.andrew.cmu.edu/pub/AUIS/bin-dist/linux) which contains just a small portion of AUIS that is suitable as a word processor.

Now another package has been released, auis63L?-mail.tgz, to sunsite.unc.edu (ftp://sunsite.unc.edu/pub/Linux) in /pub/Linux/X11/andrew. This package provides the Linux community a powerful mail system which leverages the

power of the tools we see in the word processing package to provide multi-media mail.

Multi-Media Mail

Electronic mail is becoming enormously popular and common these days, but most of the tools we use have hardly changed since the earliest e-mail systems were invented. A major effort in the Andrew project was the development of the Andrew Mail System (AMS) and its mail reading tool, **messages**. The developers at CMU wanted a mailer which not only met the needs of traditional mail systems, but could also be used for both public and private bulletin boards. Today at CMU today one can follow over 7500 bulletin boards with **messages**. The scale of the data is a major factor in the design of **messages**.

Another goal for the developers of **messages** was to provide *the* multi-media mail system. The AUIS data-stream was designed from its inception to be mail-able. This means it uses only 7-bit ASCII data which isn't “too long” for the various mail systems one might encounter. From a technical standpoint, this design has worked well.

MIME

Early versions of **messages** suffered from the “chicken-and-egg” syndrome. Since there was no standard at the time, most people were reluctant to use **messages** because not enough other people used it. On the other hand, if a large number of people used it, then everyone would love to use **messages**. For instance, if telephones were very uncommon and only 1% of people had one, your business would be unlikely to install a telephone. On the other hand, no business today would think of not having a telephone, since virtually everyone has a telephone.

Everyone seemed to think **messages** was great—but at the time you could only send AUIS-mail to others who used **messages**. On large homogenous sites like CMU, this worked fine. But how to convince the rest of the world to convert?

After years of effort, the answer became obvious: you can't. Not everyone was going to convert to **messages**--no matter how wonderful it might have been. One of the principle architects of **messages**, Nathaniel Borenstein, decided to take another tack. He proposed an architecture called MIME (Multipurpose Internet Mail Extensions) that would allow one to send non-traditional mail (like pictures) in an architected format. Nathaniel retrofitted **messages** to send mail in MIME format instead of native Andrew Mail format. When receiving MIME mail, he modified **messages** to call a new program, **metamail**, to display non-text on your machine. After demonstating that this all worked, Nathaniel

provided similar updates to several common mailers to support MIME in the same manner.

For example, if I send mail using **messages** with a picture to someone who uses some other mailer on a SUN workstation, the recipient's mailer might end up calling **xv** to display the picture. In a non-X environment, perhaps some other tool might be called. Finally, if the picture just cannot be displayed, then the mailer might "convert" the picture into some text like "a picture was provided here, but cannot be displayed".

Messages

Messages is actually just one part of a larger system known as the Andrew Mail System. This system supports reading and posting to bulletin boards and delivery between cells in the Andrew File System. I will not discuss any of these topics here, but rather just describe **messages** as a conventional mail user agent.

To view your mail, invoke the command **messages** from an **xterm** window. In Figure 1 you can see that the resulting **messages** window has three parts. The top window shows a list of folders where you can save mail. The folder "mail" is where your incoming mail is saved until you move it elsewhere or delete it. When you use **messages** the first time, your only folder will be a mail folder. To begin, select a folder (probably **mail**) by pointing to a name and clicking with the left mouse button.

The middle window will now display a summary of the mail in that folder. Select one of these and you'll see the summary line in bold and, in the bottom window, the text of the message. In Figure 1 you will notice several things. The headers of the mail (From:, To:, etc.) are show in bold. When you delete a piece of mail, the summary line is changed to a smaller font, but the actual file is not deleted until you explicitly delete it. More discerning eyes will notice the scroll bar shows that there is something above the **From:** line. This data is the normal mail headers which you normally do not want to see. If you look at the headers for this particular piece of mail, you'd see that it has the lines:

```
MIME-Version: 1.0
Content-Type: multipart/alternative;
   boundary="Interpart.Boundary.ohyS2z9z0001I1RlEF"
```

indicating this is MIME mail and consists of several parts. If you look at the summary line for this mail, you'll notice the phrase "(60+1)", meaning the body consists of 60 characters and contains one inset.

In the body of the mail, you can see the phrase "Information SuperHighway" is in a larger bold font. MIME's metamail or another low-end MIME reader would

convert this to simple text on systems which do not have font capabilities. On some systems the picture might show up in a separate window. Since **messages** has all the capabilities of AUIS, the text and images are shown in-line, as they were intended.

Sending Mail

To compose and send mail to someone, select *Send/Post Message* on the **messages** menucard. This will present a window as shown in Figure 2. In many ways, this is just another **ez** session. (**ez** is the basic editor in AUIS and was described in issue 4 of *Linux Journal*.) In this particular example, I selected the mail we saw in Figure 1 and then I selected *Reply to Sender* on the **messages** menucard. This brought up the **messages-send** window with the **Subject:** and **To:** fields already filled out. I then selected the first sentence of the original mail (so it was shown in reverse video) and then in the **messages-send** window, selected *Excerpt body* on the *Other* menucard. This resulted in the indented and italized *Excerpts* lines that you see.

Had we not marked any text and simply selected this menu card, the entire original message body would have been copied, indented and italized.

Notice that the excerpted area may contain more than simple text, for example the large, bolded **Information SuperHighway**. In fact we could have also included the image you saw in Figure 1. To send the mail, select *Send/Post* on the *messages-send* menucard. If the mail contains multi-media (anything except simple plain ASCII text), you will be prompted as shown in Figure 3. If you select *Remove formatting and send*, **messages** will do some simple text conversion. The excerpt will have the font information removed and the excerpt text itself will be prefixed with “>” as is common with many mail readers. As you can see in Figure 3, you may choose to send the mail in the original AUIS format or in MIME format. You can avoid the question altogether and always send in MIME format with an entry in your **$HOME/preferences** file like this:

```
        *.mailsendingformat: mime
```

Issue the command **auishelp preferences** and read about the *mailsendingformat* setting for more information.

Options

Everyone wants to do mail their own way—and the same is true with **messages**. **Messages** has dozens of options you may set—so many that it provides its own interface to setting and querying these. If you select *Set Options* on the *Other* menucard, **messages** will present a list of the options and their current setting.

I'll bet as soon as you have **messages** working, it won't be long before you will be poring through the options, trying them out.

Compatibility with Other Mail Readers

The designers chose to not keep the **messages** database compatible with that of other Unix mail readers like **elm**. In conventional Unix mail user agents (e.g., **mail**) when mail is received, it is first stored in **/usr/spool/mail/$USER** as a simple "flat" file. All the mail is mashed together in one physical file. It's up to the mail reader to sort these out. This works fine when the file is small, but when you have 100 pieces of multi-media mail, each 50K in size, it starts getting unwieldy and slow.

Now when you invoke your mail reader (e.g., **elm**), the reader shows you what is in **/usr/spool/mail/$USER**. When you save the mail to a folder, the mail is appended to some file in **$HOME/Mail** (e.g. **~/Mail/tpg** in my case). Just as with the mail in **/usr/spool/mail**, this is also a simple file containing many logical files (pieces of mail). It will suffer even more from performance problems as you keep more and more mail around. Most mail readers allow you to create separate folders for categories of mail—but each still uses one monolithic file.

In the world where AUIS was developed, it is not unusual for one to have hundreds of pieces of mail (and large multi-media mail at that). So another, incompatible approach was taken. **Messages** keeps each individual piece of mail as a **separate file** in a folder (i.e., directory) and builds an index so it can quickly show what's in the folder. Each of these folders is kept in the directory **$HOME/.MESSAGES**.

The drawback to this approach is that you cannot directly switch between **elm**, for example, and **messages**. Any mail received by **messages** is not available in your next **elm** session. Similarly, mail you received in the past with **elm**, is not directly available in **messages**. However, that does not mean it cannot be done —but you must do it "manually". I have developed some techiques to allow you to "convert" your **elm** mail for use in **messages** and vice versa. I will not describe these in detail, but rather direct you to read about it in **/usr/andrew/README.ez.mail** which is created when you install the **auis63L?-mail** package.

Printing and Previewing

Just as AUIS prints other text documents, **messages** will print your mail using PostScript. In this version AUIS objects all generate **troff** output—along with copious amounts of embedded PostScript. The **troff** is then processed to generate the necessary PostScript. The default print command will invoke a shell, **/usr/andrew/etc/atkprint**. The default preview command calls the shell **/usr/andrew/etc/atkpreview**. Each of these shells will invoke the **groff** formatter

to generate the PostScript output. In **atkpreview** the **groff** output is directed into **ghostview**.

For More Information

A mailing list is available at **info-andrew@andrew.cmu.edu** (mail to info-andrew-request@andrew.cmu.edu for subscriptions). The newsgroup comp.soft-sys.andrew is dedicated to the discussion of AUIS. A World Wide Web home page can be found at www.cs.cmu.edu:/afs/cs.cmu.edu/project/atk-ftp/web/andrew-home.html. A book, *Multimedia Application Development with the Andrew Toolkit*, has been published by Prentice-Hall (ISBN 0-13-036633-1). An excellent tutorial is available from the Consortium by sending mail to info-andrew-request@andrew.cmu.edu and asking about the manual, *A User's Guide to AUIS*.

Terry Gliedt (tpg@mr.net) left Big Blue last year after spending over twenty years with IBM. Although he has worked with Un*x and AUIS for over six years, he is a relative newcomer to Linux. Terry does contract programming, teaches classes in C/C++ and Unix and writes the occasional technical document.

Archive Index Issue Table of Contents

Advanced search

Mobile Computing with Linux

Marc E. Fiuczynski

Issue #6, October 1994

This article serves as a buyers' guide for people about to purchase a Linux-capable laptop and points out existing software support for Linux.

The perfect laptop is energy-efficient, lightweight, fast and upgradeable. These features are addressed in product reviews which abound in popular PC magazines. The product reviews are geared towards DOS/Windows users, and they tend to ignore hardware support issues which would arise in other environments. A valuable source of information for users about to purchase a Linux-capable laptop is the Linux Laptop Survey, which can be found in **tsx-11:/pub/linux/ packages/laptops/laptop-survey**. You will find over fifty entries describing make and model, disk size, processor, battery, screen types and general comments that come in handy. One highly overlooked issue addressed by the survey is that some laptops employ proprietary components, such as RAM modules which can only be upgraded by the original manufacturer. Other proprietary components to look out for are PCMCIA adaptor chips, sound support, and special displays.

Getting more power

There are many factors that determine how long your laptop will run tetherless, but probably the most important is the type of battery your laptop uses. A few years back NiCd (Nickel-Cadmium) batteries were the battery of choice for laptops. However, NiCd batteries do not recharge properly unless they are completely discharged, a behavior known as the *memory problem*.

Current generation laptops use batteries composed of NiMH (Nickel Metal Hydride) which do not have the NiCd memory problem and provide more energy. The next-generation battery beyond NiMH uses lithium-ion as its power source which provides even more energy.

Most modern laptops have a hot battery swapping feature, which allows you to replace a drained battery with a charged one while you are working without having to shutdown your system or lose data. If your laptop supports battery swapping, then it has a small internal battery that provides enough power to refresh the DRAM.

Some laptops can even replace their floppy drive with an additional battery, allowing you to operate your laptop for extended tetherless periods. Others allow you to add extra battery packs on the back, which have the added benefit of stabilizing your system when you try to hold it on your lap.

Virtuous misers

Advanced power management (APM) is a specification from Microsoft that allows the system software and the system BIOS to cooperate in reducing overall power consumption. Laptops that conform to the APM specification (i.e., their BIOS has APM) may use software support to increase battery life by reducing the power consumption of the system without degrading performance.

There are a myriad of x86 processors made by different vendors for laptops, marked with the SL acronym. These processors implement a set of system management mode (SMM) features to assist in power management, but Linux does not yet take advantage of them. Are these features worth their price, or are they simply a marketing ploy by the processor manufacturer? One argument in their favor is that the SL processors use 3.3-volt circuitry, which consumes less than half the power of traditional 5-volt systems. However, newer DX models (e.g., DX-4) also use 3.3-volt logic, so the value of a SL processor is questionable.

Hard disk storage capacity ranges between 80 and 340MB. Most modern laptops use second generation IDE drives which support various power management features such as spinning down and turning off unused components. By default, many laptops will automatically spin down the disk after some idle period when running off of the battery.

Desktop vs. Laptop

Money!

Flat panel displays drive the overall price of the laptop. There are different types of flat panel displays: active-matrix color, dual-scan passive-matrix color, and passive-matrix monochrome. Active matrix has the best color display, with immediate refreshes that yield a crisp picture. The down side of active matrix displays is that they are expensive, require more battery power, and add to the

overall weight and size of the system. Also, unlike monochrome displays, active matrix displays are barely viewable in direct sunlight. The opposite extreme is passive matrix monochrome, with its low cost and low power requirements. However, it doesn't refresh the screen as nicely, is harder to read at an angle, and leaves streaks of former images (known as “ghosting”). If cost is a limiting factor, but color is required for your application, then dual-scan passive-matrix is a reasonable alternative.

There are a few potential problems with laptop displays. Some laptops may use new display chip sets that XFree86 does not fully support, resulting in very low resolution and/or unacceptably low speeds. Also, even more so than on desktop systems, the X-Windows setup of Xconfig has to be done with extreme care to avoid possible damage to LCD pixels. Pointing devices such as an external mouse, trackball, or EZ-point (IBM mini-joystick in the middle of the keyboard) are generally supported by Linux, since they usually emulate one of the popular mouse types. At the moment there is no support for pen input used on machines like the Compaq Concerto.

Personal Computer Memory Card International Association (PCMCIA) is a standards body for an IC card standard. The PCMCIA standard is becoming the form factor of choice in laptops and the PCMCIA devices released include modems, ethernet adaptors, radio LAN adaptors, radio alphanumeric pagers, hard drives, SCSI adaptors, multimedia, serial, parallel, GPS, and various types of memory cards.

Most modern laptops have PCMCIA support and use a PCMCIA adaptor chip provided by one of several vendors. The most commonly found adaptor chips are the Intel 82365 and Databook TCIC/2. There are others from Cirrus, Motorola, HP, IBM, Databook, Toshiba, etc. Think of the PCMCIA adaptor chip as a built-in adapter for the PCMCIA bus, just as there are various SCSI adapters. Unfortunately, each chip requires a special driver.

Linux Support for Laptop Features

Power management is not really supported under Linux. Based on the Linux Laptop Survey results, a laptop running Linux will typically last for two hours on batteries. Although an APM support package does exist, it only provides the initial hooks into the APM BIOS for power management. At this point the APM support just recognizes when the system returns from sleep mode and updates the internal clock, which is important to keep the system sane. APM applications that display the current battery level and execute the “shutdown” command when the battery runs low are also available. This software may be retrieved from **tsx-11:/pub/linux/ packages/laptops/apm**.

As mentioned earlier, the SL processor series has SMM features to power manage the processor. However, a portable power management solution that doesn't require SMM support is to halt the processor in the scheduler's idle loop. The x86 asm(hlt) instruction suspends the processor until there is an interrupt from the system (e.g. a key stroke). Linux versions above v1.1.10 implement this simple and effective solution, which does not degrade performance at all. It has the added benefit (even to non-laptop users) of increasing the expected life of your CPU.

PCMCIA provides plug-and-play capabilities to the laptop. Currently support exists for the Intel 82365 adaptor chip, PCMCIA modems (Megahertz, IBM, Intel, AT&T, and others), and ethernet cards (D-Link 650, Linksys, IBM credit card, and 3COM 3c589). The PCMCIA support in Linux is still in its alpha stages and is available from **tsx-11:/pub/linux/packages/laptops/pcmcia**. PCMCIA support that meets the PCMCIA Unix specification is in the works and you may expect it to be part of the standard Linux distribution by the end of the year.

Future Topics

A future article will discuss operating system implications of mobile computers.

Bibliography

F. Douglis and B. Marsh, *Low-Power Disk Management for Mobile Computers*, Matsushita Information Technology Lab, 2 Research Way, Third Floor, Princeton, NJ, 1993, MITL-TR-53-93

Fred Douglis and P. Krishnan and Brian Marsh, *Thwarting the Power-Hungry Disk*, Proceedings of the 1994 Winter USENIX Conference, January 1994, (Also Matsushita Information Technology Lab Technical Report MITL-TR-61-93)

Proceedings of the 1994 Winter USENIX Conference, January 1994

Marc E. Fiuczynski (mef@cs.washington.edu) is a computer science graduate student at the University of Washington in Seattle. His research interests are distributed systems, communications, operating systems, and mobile computing.

Archive Index Issue Table of Contents

Advanced search

Learning C++ With Linux

Jeff Tranter

Issue #6, October 1994

Jeff gives us a head-start for learning C++ on our own.

Object-Oriented programming is a hot topic in the computer industry these days, and most experts agree that C++ is the predominant object-oriented programming language. Many programmers are familiar with the C programming language and would like to move to C++, but feel they lack the necessary tools and resources, particularly if the training has to be done on their own time.

It should therefore come as welcome news to learn that Linux makes an ideal platform for learning C++. This article covers some of the C++ programming tools available under Linux and refers the reader to additional resources, many of them freely available on the Internet.

Resources

By resources, I am referring to sources of information that will help you learn C++ and solve programming problems.

A large number of books on C++ and object-oriented design and programming are available, some better than others. The three I suggest here are among the most popular of those that are specific to C++; you may be able to borrow a copy from your local public, school, or corporate library. As these concentrate more on the C++ language itself, you may wish to supplement them with books that cover object-oriented analysis and design.

- *The C++ Programming Language* (2nd edition), Bjarne Stroustrup, Addison-Wesley, 1991
- *Annotated C++ Reference Manual*, Bjarne Stroustrup and Margaret Ellis, Addison-Wesley, 1990
- *C++ Primer* (2nd edition), Stan Lippman, Addison-Wesley, 1989

There are a number of publications related to object-oriented programming, including the following:

- *C++ Report*
- *Object Magazine*
- *Journal of Object-Oriented Programming*

For those with access to Usenet, these newsgroups are relevant and can be very useful, both as a source of answers to specific questions and for picking up the latest developments in C++ and object-oriented programming:

- **comp.lang.c++**
- **comp.std.c++**
- **comp.object**

The following Frequently Asked Question (FAQ) lists are periodically posted to Usenet, and are available by anonymous FTP from Internet archive sites (listed in brackets):

- C++ FAQ(**sun.soe.clarkson.edu:/pub/C++/FAQ**)
- G++ FAQ(**rtfm.mit.edu:/pub/usenet/news .answers/g++-FAQ/plain**)
- comp.object FAQ(**rtfm.mit.edu:/pub/usenet/news .answers/object-faq/***)

Tools

Listed here are a number of useful C++ programming tools. If you use one of the standard Linux distributions you probably have most of these already, otherwise you can get them from a major archive site. To save time and disk space, I suggest obtaining the Linux binaries rather than building them from source.

The standard Linux C++ compiler is GNU g++ from the Free Software Foundation. It follows the evolving ANSI C++ standard and supports most features found in AT&T's cfront 3.0 compiler, including templates. It does not yet support exceptions.

Unlike cfront, which is a preprocessor, g++ generates native code. As the compiler is evolving quickly, I recommend getting the latest version. (At the time of writing most Linux distributions included version 2.5.8; version 2.6.0 had just been released.)

Gdb is the GNU symbolic debugger; you have probably used it already for debugging C programs under Linux. It supports most C++ data types and language constructs, and transparently handles C++ “name demangling”. Gdb

runs well inside Emacs, or you can use the xxgdb graphical user interface under X. The documentation for gdb, in info format, describes the features specific to C++ debugging.

The programmer's editor of choice, Emacs, has a C++ mode that assists in editing. It works well in conjunction with gdb and g++, allowing you to compile and debug from within the editor.

Class Libraries

If you want to run any meaningful programs, such as examples from a textbook or code of your own, you will want some class libraries. A number of C++ class libraries are available under Linux.

The GNU libg++ library provides the standard C++ iostream class. It also includes a number of additional useful classes, from complex numbers to general-purpose stack, queue, and set objects. Since the source is freely available, you can read it to understand how the libraries were implemented. Libg++ is well documented in the included info pages.

InterViews is an object-oriented toolkit for graphical user interface programming in C++. It is included as the “iv” series in the Slackware distribution of Linux. A new version of InterViews is included in the recently released revision 6 of the X Window System (X11R6), under the name Fresco. Fresco has been sucessfully compiled with GNU g++ version 2.6.0.

NIHCL, the National Institutes of Health Class Library, is a portable C++ translation of the Smalltalk-80 class library, by Keith Gorlen of NIH. The source is available on the Internet from FTP site alw.nih.gov in the directory /pub/NIHCL. At time of writing NIHCL would not compile under g++; this should be corrected in a future release.

ObjectBuilder is a graphical user interface builder for the OI C++ toolkit. It is designed to simplify the development of X11-based user interfaces. The Linux version is freely distributed in binary format; the same product is sold commercially by ParcPlace Systems for other computing platforms.

If, as some experts recommend, you want exposure to a “pure” object-oriented language, you can investigate GNU Smalltalk. The GNU gcc compiler also supports Objective-C, another object-oriented language based on C and Smalltalk. Both of these tools are available as packages under Slackware Linux.

Conclusions

Becoming proficient in C++ takes time and effort, but you can gain a career advantage by learning it on your own using Linux. I suggest trying an iterative approach which alternates between reading books and writing your own code. You may also want to consider some formal training.

Ideally you will be able to find a mentor to guide you and answer questions as you embark on your C++ education. More likely, you will find yourself the mentor for co-workers who will learn from your mistakes. I wish you success in your journey.

(Jeff.Tranter@Software.Mitel.com) works as a professional programmer. His Linux-related interests currently include C++, LaTeX, and multimedia.

Archive Index Issue Table of Contents

Advanced search

The Joy (and Agony) of SLIP

Warren Baird

Issue #6, October 1994

Have you ever wanted to connect your computer to a network? Perhaps you have two computers that you want to run Mosaic, ftp, or X11 between? Do you want to let your friends experience the full power of your magnificent new Linux box? SLIP is one way you can do all these things.

Have you ever wanted to connect your computer to a network? Perhaps you have two computers that you want to run Mosaic, ftp, or X11 between? Do you want to let your friends experience the full power of your magnificent new Linux box? SLIP is one way you can do all these things.

by Warren Baird

SLIP (Serial Line Internet Protocol) is a protocol that allows you to make a TCP/IP connection over a serial line. TCP/IP stands for Transmission Control Protocol/Internet Protocol, and is the protocol used on the Internet and on most local ethernets. Many common Unix programs (like X11, Mosaic, gopher, IRC, talk, telnet and ftp) use TCP/IP to communicate. If you want to use programs like this to talk to the rest of the world over the Internet, or if you want to connect two (or more) local machines, SLIP is a good option.

There are actually two variants of SLIP: normal SLIP, and CSLIP (which stands for Compressed SLIP). CSLIP compresses various TCP/IP headers to give a higher throughput on a serial line. (Note: CSLIP doesn't compress the data that is being transmitted, just the headers added by TCP/IP.) Unfortunately not all systems support it. Even with CSLIP, the throughput on a serial line is quite low compared to ethernet and other Local Area Networks (LANs). It's possible to run high bandwidth applications like Mosaic and xv over a SLIP link, but the response times may be quite low.

One of the most common uses of SLIP is to connect to a commercial Internet provider. There are many companies that provide full access to the Internet by allowing you to make a SLIP connection using a modem.

Another use of SLIP is to connect two or more machines using a null-modem cable. Since SLIP is implemented for many different platforms, you can connect your Linux box to many different types of systems. For example, I have an Amiga 2000 connected to my 386 running Linux 1.0.8.

If you are really ambitious, you can even set up your Linux box as a SLIP server. You could do this to give your friends access to your machine, or as part of a BBS setup.

Configuring SLIP under Linux

Configuring something like SLIP under Unix is always very dependent on the flavor and version of Unix that you are running. The details given here are for the Slackware 1.2.0 (Linux 1.0.8) distribution. The exact details may vary somewhat with other versions of Linux and other distributions.

What do all these files do?

There is a whole host (no pun intended) of files that control the behavior of a Unix system, and a number of them need to be modified to set up SLIP.

Looking up host names: **/etc/hosts**, **/etc/resolv.conf**, and **/etc/host.conf**

The **/etc/hosts** file maps hostnames (for example acme.gadgets.com) to IP addresses (for example 129.50.49.234). If you are using SLIP to connect two local machines, your /etc/hosts file only needs to contain the hostnames and IP addresses of your two machines. If you are connected to a larger network, but talk to only a few sites, you can put their names and IP addresses in your /etc/hosts file.

If you are connecting to many hosts (or using a program like Mosaic that connects to many hosts), you'll need to tell your system who it should ask to find IP addresses for unknown hostnames. There are systems called nameservers that will lookup an IP address for a given hostname. The /etc/resolv.conf file tells your system which nameserver(s) to use. It also tells your system what your domain name is. The domain name is your full hostname with the machine name removed—for example, if your hostname is acme.gadgets.com, your domain name would be gadgets.com.

If your domain name is gadgets.com, and you have a nameserver at the IP address 128.34.54.27, your **/etc/resolve.conf** file could read:

```
domain gadgets.com
nameserver 128.34.54.27
```

It's possible to list multiple nameservers in a **resolve.conf** file. The **/etc/host.conf** file tells the system what order it should use to lookup hostnames. Normally this file contains

```
order hosts,bind
multi on
```

which tells the system to check the **/etc/hosts** file first, and then to try any nameservers it knows about.

What needs to be done at startup?

The names of the Internet startup files vary even between different Linux distributions. On most Linux systems the files used are **/etc/rc.d/rc.inet1** and **/etc/rc.d/rc.inet2**. Whichever startup file is used, there are a number of daemons that must be running for any TCP/IP program to work. The NET-2 HOWTO describes which daemons are available, and what they do. Minimally, you need to have the inetd daemon running. The inetd daemon coordinates the startup of many other TCP/IP daemons.

Additionally, if you are setting up a permanent link to another computer over a dedicated line, you may want to actually initialize the SLIP link in the startup files.

How do I connect to another machine using SLIP?

There are two primary ways that you can connect to another machine using SLIP. If it is a link over a leased line, or a null-modem cable, you should use slattach. If you are connecting over a modem (or allowing other computers to connect to yours over a modem) you should use dip.

To configure a SLIP connection with slattach, you need to execute the commands:

```
/sbin/slattach -p prot -s baud dev &
/sbin/ifconfig sl0 iplocal pointtopoint ipremote up
/sbin/route add default dev sl0 &
```

where prot is the protocol to use (normally slip or cslip), baud is the baud rate to connect at, dev is the device to use, iplocal is your local IP address, and ipremote is the remote IP address. For example, if your IP address is 129.45.43.76, and you want to connect to a machine with IP address 129.50.49.234 using slip at 19200 baud on /dev/ttyS1 (COM2), you would give the commands:

```
/sbin/slattach -p slip -s 19200 /dev/ttyS1 &
/sbin/ifconfig sl0 129.45.43.76 pointtopoint\
   129.50.49.234 up
/sbin/route add default dev sl0 &
```

These commands can either be added to a startup file, if you always want the SLIP connection to be initialized, or you can create a simple shell script to execute them for you. You'll have to run this shell script as root for **slattach** to work. If, instead of a shell script, you create a C or **perl** program owned by **root**, you can make it setuid (always run as root) by typing **chmod u+s file** (where file is the name of the program). Take note of the "&" following the /sbin/slattach line. Older versions of the NET-2 HOWTO omitted the ampersand, which caused Linux to lock up when booted if the commands were added to the startup files.

The dip program is a little more complicated to use. To use it to connect to another machine, you must create a script file telling dip how to call the machine, log onto it, and start slip (if slip isn't started automatically). The NET-2 HOWTO and the dip(8) man page both give sample dip scripts (dip(8) means that the man page for dip is in chapter 8 of the man pages, so you can read the man pages with the command **man 8 dip**). Dip can also be used to allow other people to connect to your machine using SLIP. To do this, you must create a special userid that has dip as its shell. Instructions for doing this are in the **dip(8)** man page.

Troubleshooting: Some common problems and how to fix them

Linux locking up at boot time

As I mentioned before, if you start slattach in your system startup files, you must add an ampersand ("&") after the slattach line to run slattach in the background. Otherwise your system will lock up when you reboot.

Packets disappear without a trace

There can be many causes of this. The first thing to check is that your serial link is correct. If you are using a null-modem cable, make sure that it's plugged in at each end. If you are connecting over a modem, make sure that the modem is working correctly. If you are using a null-modem cable, and you have easy access to both ends of the link, try starting a terminal program on one end, and see if anything shows up if you ping the machine running the terminal program. If garbage appears in the terminal program, then you know that your serial link is working.

There seems to be a problem with CSLIP in some versions of Linux. If you are using CSLIP and packets seem to be disappearing, try using SLIP instead.

Unknown host errors occur

The first thing to do is to try using an IP address instead of a hostname. If you were trying to type ping acme when you got the Unknown host error, find the IP address of the machine acme, and ping it instead. For example, if acme has an IP address of 128.155.123.6, type

```
ping 128.155.123.6
```

instead. If using the IP address works, you have a problem with translating hostnames into IP addresses. Make sure your /etc/hosts file has a correct entry for your hostname, or if you are using a nameserver, make sure that your system isn't having trouble contacting the nameserver. As a temporary fix, adding an entry for the hostname into your /etc/hosts should solve the problem.

If the IP address doesn't work either, you may need to use the route command to tell your system how to find the remote system. The NET-2 HOWTO and the route(8) man page show you how to use the route command.

“Network is Unreachable” Errors

If you see messages like:

```
ping: sendto: Network is unreachable
```

when you are trying to use ping, the most likely cause is incorrect routing. Using the /sbin/route command by itself shows you the current routing table. If the routing table is configured correctly, you should see something like:

```
Destination  Gateway  Genmask  Flags  Metric  Ref  Use  Iface
127.0.0.0    *       255.0.0.0 U     0       0    0    lo
default      *       *         U     0       0    3    sl0
```

If the last line isn't present, and you are using slattach, make sure that the /sbin/route line described above has been used. If the last line is present, or if you are using dip, see the NET-2 HOWTO, and the dip(8) and route(8) man pages.

What alternatives are there to SLIP?

There are other programs that provide functionality similar to SLIP. Here is a brief overview of some of the most popular.

If you want to connect two local machines, the best way is to use ethernet, which gives far higher throughput than a serial link. Unfortunately, ethernet cards can be fairly expensive. If you can't afford a pair of ethernet cards, or if

you want to connect to another machine using a modem, there are other options. SLIP isn't the only protocol that provides TCP/IP (or TCP/IP-like) connectivity over a serial line. PPP (Point-to-Point Protocol) is a newer protocol that serves a purpose similar to SLIP. PPP was designed specifically to be an Internet standard, and is documented in a number of Internet RFCs (Requests for Comments). This design process makes PPP more standardized than SLIP. Additionally, PPP was designed to address known weaknesses of SLIP in areas of security and robustness. Unfortunately, since PPP is a lot newer and more difficult to implement than SLIP, it isn't available on as many systems as SLIP is.

Another common program is term. Term does not provide a full TCP/IP connection, but it does allow some similar functionality. It will let you run multiple terminal sessions over a single serial line (like SLIP does), but it won't let you run TCP/IP programs (like ftp, Mosaic, or telnet) on your Linux box without recompiling them to include term support. It will let you display remote X11 programs on your local machine, so you could, for instance, run Mosaic remotely and display it on your Linux box. You can also use term to connect ports on the local machine to ports on the remote machine, which allows you to read news and mail on your Linux box. The biggest advantage that term has over SLIP is that it can be set up on both ends without requiring root privileges. Unlike SLIP, you don't have to change any system configuration files to use term.

PLIP (Parallel Line Internet Protocol) is a SLIP-like protocol that runs over a parallel line. PLIP is only intended to connect two machines locally, but it gives higher transfer rates than a comparable SLIP connection at the cost of using more CPU time than SLIP. To use PLIP you need a special parallel cable, like those used in a number of DOS PC-to-PC file transfer packages.

If you only want to connect two local machines, you might want to consider just running a terminal program over a serial link. The configuration is certainly simpler than setting up SLIP (you need to start a getty process on the appropriate serial port), and if you run a program like screen (which allows multiple sessions in a manner similar to virtual consoles), you might get all the functionality that you need. Certain operations (like transferring files) are more efficient when just using a terminal program, since there isn't the overhead imposed by SLIP. The disadvantage is that you can't run X11, or any other TCP/IP programs through a terminal program.

Recommended reading

The Serial and NET-2 HOWTOs have a lot of useful information about networking and serial links with Linux in general. The NET-2 HOWTO has an entire section devoted to SLIP. The HOWTO files can be found in /usr/doc/faq/

howto on most Linux systems, or on the ftp site sunsite.unc.edu in the directory /pub/Linux/docs/HOWTO.

The Usenet newsgroups comp.os.linux.help and comp.os.linux.admin both discuss SLIP quite regularly. Reading these newsgroups will often answer simple questions about SLIP (and many other things). It's a good idea to read the appropriate HOWTOs and FAQs (Frequently Asked Question lists) before asking any questions, since some people become quite irate when you ask questions that are answered somewhere else.

There are a number of books that address administrating TCP/IP connections. The best I have come across is TCP/IP Network Administration, by Craig Hunt, published by O'Reilly & Associates, ISBN 0-937175-82-X.

SLIP isn't the newest TCP/IP protocol around, but it is commonly available and is usually quite stable. It's a good choice for many serial TCP/IP links.

Warren Baird runs Linux on a 386 DX-33, and has been hacking various flavours of Unix for five years. He is on the verge of getting a B.Math joint-honours degree in Computer Science and Com-binatorics & Optimization at the University of Waterloo, Ontario, Canada. He can be reached at wjbaird@uwaterloo.ca , or baird@asc.on.ca .

Warren Baird (wjbaird@uwaterloo.ca) or (baird@asc.on.ca) runs Linux on a 386 DX-33, and has been hacking various flavours of Unix for five years. He is on the verge of getting a B.Math joint-honours degree in Computer Science and Com-binatorics & Optimization at the University of Waterloo, Ontario, Canada.

Archive Index Issue Table of Contents

Advanced search

Tutorial: Emacs for Programmers

Matt Welsh

Issue #6, October 1994

Ever wanted an all-in-one program development, compilation, and debugging environment? Look no further than Emacs.

Those of you who tuned in last month will recall “Emacs: Friend or Foe?”, a tutorial for people who can't stand anything but **vi**. “All right,” you're asking yourselves, “What is this card-carrying **vi** fundamentalist doing writing yet another article on Emacs?” Sounds fishy, doesn't it?

The truth is, once you get the hang of it, Emacs can greatly simplify editing, especially editing program source code. I now routinely use Emacs for developing and debugging programs.

It dramatically reduces turnaround time during the dreaded **edit-compile-curse-debug-edit** cycle. Here's how to put Emacs to use in this manner.

The bulk of this tutorial assumes that you are familiar with Emacs, as well as with customizing your Emacs environment (as discussed in last month's tutorial, in *Linux Journal* volume 1, issue 5). As long as you know how to add code to your .emacs startup file (or, as per last month's discussion, **~/emacs/startup.el**), you're set.

The functions and comments described here work with GNU Emacs 19.24.1. By the time you read this article, newer versions may be available, in which case your mileage may vary.

Editing C Code

As you know, Emacs has several major modes associated with programming. For example, C Mode is used for editing C source, Perl Mode for Perl, and so on. First off, I'll discuss the features of C Mode, and then explain how to compile and debug C programs within Emacs.

The command **M-x c-mode** is used to enter C Mode. (Recall from last month: **M-x** is **Meta-x** where meta is usually the <esc> key.) However, Emacs is usually able to determine that C Mode should be used for a C source file, either by the filename extension **.c**, or if the magic string

```
-*- C -*-
```

appears on the first line of the file. See the Emacs documentation on major modes if you're interested in how this works.

Within C Mode, code is automatically indented according to the values of several variables. These variables include **c-indent-level**, **c-continued-statement-offset**, and so on. The Emacs Info pages describe these variables in gory detail; however, I find that the default values work quite well, for a number of indentation styles. Unless you have a particularly unique artistic flair when it comes to indenting your code, I suspect that you won't have to fiddle with Emacs' indentation variables.

A line is indented appropriately when you press TAB anywhere on the line. This does not cause a tab character to be inserted; it just indents the line according to the variable values mentioned above. If you want to actually insert a tab character, prefix it with **C-q**.

To see how this works, start up Emacs and edit a file called foo.c. Type a few lines of bogus C code, pressing RET after each. Then go back and press TAB on each line to see the results.

Pressing LFD instead of RET is equivalent to pressing RET followed by TAB—that is, you start a new line, and it is automatically indented for you. I find it particularly useful to bind RET to this function, which is **newline-and-indent**, so that I don't have to use LFD when writing code. In my Emacs configuration file, I include the line:

```
(define-key c-mode-map "\C-m' -newline-and-indent)
```

One feature that you may have noticed is that closing braces and parentheses automatically **blink** their opening counterparts. This can help you to check that parentheses are balanced in your code. If you don't like this feature, you can turn it off using

```
(setq blink-matching-paren nil)
```

in your **.emacs** file (or **~/emacs/startup.el**, for those of you using the method described last month. Hereafter, we will refer only to **.emacs**, but keep in mind that all of these customizations can be used with both methods).

If you're running Emacs under X, the paren library will cause matching parentheses and braces to be highlighted whenever point is on an opening brace/parenthesis, or after a closing brace/parenthesis. Simply including

```
(load-library "paren")
```

in your **.emacs** file will enable this feature. Balanced parentheses are highlighted in the region face; you can change the color or font used with commands such as **set-face-foreground**, **set-face-font**, and so on.

For example,

```
(set-face-background -0region "pink")
```

will set the background color for this face to pink. The region face is also used to display the current region when **transient-mark-mode** is enabled. We'll talk a bit more about faces below.

You'll also notice that typing a closing brace (on a line by itself) will exdent the line containing the brace. When

typing code such as:

```
int foo() {
  /* Your code here */
```

After pressing RET, the next line will be indented relative to the comment above it (assuming, of course, that you have bound RET to **newline-and-indent**). Now, after typing the closing brace, you'll end up with:

```
int foo() {
  /* Your code here */
}
```

Braces are bound to the function **electric-c-brace**, which inserts the brace, and corrects indentation on the current line. The indentation of braces, and the text enclosed by them, is controlled by the Emacs variables **c-brace-offset**, **c-imaginary-brace-offset**, and so on.

In general, your code should follow the indentation style set forth by Emacs. Adding comments is one exception. Many programmers like to set comments out towards the right margin of the display, as in

```
int floof(struct shoop *s, int i) {
   s->fnum = i;     /* You are not expected
                       to understand this */
   return 0;
}
```

Now that TAB has lost its natural ability to add whitespace, how can we add such a comment? Emacs provides the **M-;** command, which begins a comment starting at the column specified by the variable **comment-column**, which is set to 24 by default. Of course, you can always add comments by typing

```
/* ... */ by hand.
```

You can use **M-x comment-region** to comment out all lines in the current region. (For Emacs neophytes, the region is defined by moving point to a particular location, using **C-Space** to set the "mark", and then moving point elsewhere. The region is the block of text between point and mark. There are various other ways to set the region; for example, under X, dragging mouse-1 over a portion of text will define the region.) Likewise, **M-C-** (that's **meta-control-backslash**) will indent the current region.

C Mode defines several new moving commands as well. **M-C-a** will move point to the beginning of the current function. Similarly, **M-C-e** will move point to the end of the current function. Note, however, that the "current function" is denoted by an opening or closing brace in the first column of text. If you use a C indentation style such as

```
int foo() {
  /* Your code here */
}
```

Emacs won't be able to find the beginning of the function, as the opening brace is at the end of the line. For these commands to work properly, you should indent your code as so:

```
int foo()
{
  /* Your code here */
}
```

To select the region as the text of the current function, you can use **M-C-h**. This provides a convenient way to manipulate entire functions. For example, the quick key sequence **M-C-h**, **C-w**, **M-C-e**, **C-y** will move the current function below the following one. Impress your friends!

The keys **M-a** and **M-e** can be used to move to the beginning or end of the current C statement (block, semicolon-delimited expression, etc.).

One last important C Mode feature: macro expansion. If you run **M-x c-macro-expand**, Emacs will run the C preprocessor on the current region and display the results in another buffer. For example, given the following code:

```
static XtResource resource_list[] = {
  { RES_N_doputimage, RES_C_doputimage,
    XtRBoolean, sizeof(Boolean),
```

```
      XtOffset(app_data_ptr,do_putimage),
      XtRImmediate, (XtPointer) FALSE,
    },
  };
```

Selecting this text as the region, and calling c-macro-expand gives us:

```
  static XtResource resource_list[] = {
    { "doPutImage", "DoPutImage",
      ((char*)&XtStrings[1561]), sizeof(Boolean),
      ((Cardinal) (((char *)
      (&(((app_data_ptr)0)->do_putimage))) -
      ((char *) 0))),
      ((char*)&XtStrings[1695]), (XtPointer) 0,
    },
  };
```

This can be useful if you're trying to debug complex macros, or need to know the definition of a given preprocessor symbol.

Many other modes exist for particular languages, such as Perl Mode, Emacs LISP Mode, Prolog Mode, and so on. Most of these modes share the basic features described above. The best way to learn about a new mode is to enter it (with a command such as M-x perl-mode) and use M-x describe-mode to get a rundown on its features.

Using faces

Emacs-19 provides support for faces, which allow different kinds of text to be displayed in various fonts and/or colors. In last month's tutorial, we described how to configure faces under Emacs; because use of faces is particularly helpful with respect to editing source code, it bears repeating here.

The command **M-x list-faces-display** will display the current faces, and their associated names, in another window. Faces are given names such as bold, bold-italic, and so on. These names don't necessarily have anything to do with how the faces appear—for example, your bold face needn't be in a bold font.

The functions **set-face-foreground** and **set-face-background** can be used to set the foreground and background colors for a face, respectively. **set-face-font** sets the font used for a particular face; **set-face-underline-p** specifies whether a particular face is displayed with an underline.

Faces are used most commonly within Font Lock Mode, a minor mode which causes the current buffer to be “fontified”--that is, the text is displayed in various faces depending on context. For example, when using Font Lock Mode with C Mode, function names are displayed in one face, comments in another, preprocessor directives in another, and so on. This is a pleasant visual effect when editing source code; you can easily identify function names and comments by glancing at the display.

The following function can be used in your Emacs startup file to enable Font Lock Mode and to set the colors for various faces.

```
defun my-turn-on-font-lock ()
  (interactive "")
  ;;; Color the faces appropriately
  (set-face-foreground -bold "lightblue")
  (set-face-foreground -bold-italic "olivedrab2")
  (set-face-foreground -italic "lightsteelblue")
  (set-face-foreground -modeline "white")
  (set-face-background -modeline "black")
  (set-face-background -highlight "blue")
  ;; Turn off underline property for bold and underline
  (set-face-underline-p -bold nil)
  (set-face-underline-p -underline nil)
  (transient-mark-mode 1)
  (font-lock-mode 1))
```

Note that in addition to turning on font-lock-mode, I enable **transient-mark-mode**. In this mode, the current **region** is shaded using the **region** face. This can save you a great deal of time trying to remember where the current mark is set.

The above function is called by:

```
(defun my-window-setup-hook ()
   (set-foreground-color "white")
   (set-background-color "dimgray")
   (set-mouse-color "orchid")
   (set-cursor-color "red")
   (my-turn-on-font-lock))
(add-hook 'window-setup-hook 'my-window-setup-hook)
```

That is, the Emacs **window-setup-hook** (which is executed at startup time) calls **my-window-setup-hook**, which first sets the foreground and background colors for the window, and then enables Font Lock Mode.

You must enable Font Lock Mode separately for each buffer that you wish to use it in. For this reason, I have Emacs call **my-turn-on-font-lock** whenever I enter C Mode, Emacs LISP Mode, or Perl Mode:

```
(add-hook 'c-mode-hook 'my-turn-on-font-lock)
(add-hook 'emacs-lisp-mode-hook 'my-turn-on-font-lock)
(add-hook 'perl-mode-hook 'my-turn-on-font-lock)
```

The best way to determine how to configure faces to your liking is to experiment with the code given above. There are several variables which control which faces Font Lock Mode uses for particular kinds of code. For example, **font-lock-comment-face** is the face used for comments. By default, its value is italic, which we set above to use the foreground color of lightsteelblue. You can either set the face properties for bold, italic, and so on directly, or you can operate on **font-lock-comment-face**, **font-lock-function-name-face**, et cetera. Using **M-x apropos** and entering **font-lock** will give you a list of functions and variables associated with Font Lock Mode.

Using tags

Emacs has a number of features dealing with tags, which are simply marked locations in your source code. The most common use of tags is to mark the beginning of a function definition. You can then jump directly to that function definition, no matter what source file it lives in.

To handle tags, Emacs uses a tags file, which is (by default) named TAGS in the directory where your source files live. Before experimenting with tags, let's create a tags file. From the shell prompt, use the command

```
etags filenames...
```

where filenames are the names of the source files in the current directory. For example,

```
etags *.c *.h
```

This will create the file TAGS, based on the C source and header files in the current directory.

Let's say that we have three source files: grover.c, oscar.c, and telly.c. These files might contain code such as:

```
/* grover.c ********************/
int grover() {
  /* Code for grover... */
}
/* oscar.c *********************/
int oscar() {
  /* Code for oscar... */
}
/* telly.c *********************/
int telly_monster() {
  /* Code for telly_monster... */
}
int main(int argc, char *argv[]) {
  /* Code for main... */
}
```

Running etags on these three source files will create tags for each function in the three files. (Using **etags** with the **-t** option will also include any **typdef**s found in the source.)

Now, we can use commands such as **M-.** (that's meta-dot) which will find a given tag. When we press **M-.** while editing one of these source files, Emacs will ask us:

```
Find tag: (default oscar)
```

You can enter the name of a tag (function name), such as **telly_monster**, and the source file containing that tag will automatically be opened, and point set to

the line containing the tag. This is a very quick way to move between source files when editing.

The default tag for **M-.** is set based on whatever word point is currently on. Therefore, if point is currently over a call to the function oscar(), pressing **M-.** followed by **RET** will take us directly to the definition of **oscar()**.

M-x find-tag-regexp will find the tag matched by the given regular expression. Therefore, using **find-tag-regexp** and giving a portion of the function name will take you to that function (assuming that the regular expression that you specified was unique for that function). If you have a set of similarly-named functions, using **M-0 M-.** (that's meta-zero meta-dot) will take you to the next tag matched by the previous use of **find-tag-regexp**.

Similarly, you can use **M-x tags-search**, which will search for the named regular expression in any of the files named in the current TAGS file. That is, **tags-search** does not limit its search for tags-it will search for any text in the files listed in TAGS. You can use **M-,** to search for the next instance of the given regular expression.

Another useful feature is tags completion. Pressing **M-TAB** will attempt to complete the current word based on functions listed in the current tags file. Therefore, when calling the function telly_monster, we can type **tel M-TAB** which will complete the name for us. If a given word has more than one completion, a ***Completions*** buffer will be opened, listing all possible choices. Under X, pressing mouse-2 on a completion will select it.

There is one caveat associated with using tags—you will occasionally need to refresh the TAGS file, in case you have done major reorganization of your code. Emacs doesn't depend on the TAGS file being 100% accurate—it will search for a tag if it is not found in the exact location given in the file. However, if you mangle your code considerably, re-run etags to refresh the tags database.

Also note that Emacs can use more than one TAGS file at a time. Most tags-based functions assume use of the file TAGS in the current directory. If you are editing source files spread across several directories, **M-x visit-tags-table** can be used to load another TAGS file into Emacs' list of known tags. Alternately, you can set the variable **tags-table-list** to a list of files or directories where TAGS files can be found. For example, I might want Emacs to always know about tags found in common library routines. In my Emacs startup file, I would use something like:

```
  (setq tags-table-list '("~/lib" "~/src/lib" "~/common"))
```

The TAGS files found in the named directories would be used in addition to TAGS in the current directory.

Updating the Change Log

Many programs are accompanied by a ChangeLog, which describes updates and modifications to the source on a day-to-day basis. Emacs allows you to semi-automatically update the ChangeLog, using the command **M-x add-change-log-entry** (or, **C-x 4 a**, which will do the same in another window).

For example, let's say that we're editing the source file grover.c, and we add a bit of code to grover(). To document this change, we use **C-x 4 a**, which will open a window containing:

```
Sun Jul 24 14:39:50 1994 Matt Welsh (mdw@loomer.debian.org
)
        * grover.c (grover):
```

This command determines that we are within the function grover(), in the file grover.c, and indicates this at the beginning of the entry. We can now enter a new log entry and save the file in the usual way. Each source file that you add entries for will be given its own item.

Compiling and debugging code

You can compile programs, and even run a debugger, entirely within Emacs. The most basic compilation command is **M-x** compile, which will run **make** (or another command of your choice) in the directory of the current buffer.

The default compilation command is **make -k**.

(The **-k** switch will prevent make from halting on an error which has no bearing on other targets in the makefile.) When using **M-x compile**, you will be prompted for the compilation command to use, and also whether you wish to save any buffers that have changed. If you wish to change the default command, set the variable **compile-command** to another value. For example,

```
(setq compile-command "make")
```

will cause **M-x compile** to run **make** without the **-k** argument.

You can also set a value for **compile-command** for a particular source file. This is done by including “local variable definitions” within the source file itself. For example, we could include the following comments within a C source file:

```
/* Local Variables: */
/* mode: C */
```

```
/* compile-command: "make" */
/* End: */
```

These comments set the values of the mode and compile-command variables for the buffer containing this code. When the file is opened by Emacs, it recognizes the line containing Local Variables: and uses subsequent lines, until the line containing End:, to assign values to variables for this buffer alone. You can use this feature to set values for any Emacs variables specific to this buffer.

Now, when we use **M-x compile**, Emacs runs the given compilation command (here, **make**) in another window, with which we can monitor the progress of the compilation. To kill the compilation process, type **C-c C-k** in the compilation buffer.

Once the compilation completes, we can use the error messages printed (if any) to automatically visit the source which caused the errors. For example, let's say that we use **M-x compile** and the following errors result:

```
cd /amd/noon/c/mdw/test/lj/
make
gcc -O -O2 -I/usr/include -I. -c main.c -o main.o
In file included from main.c:12:
libpx.h:30: image.h: No such file or directory
libpx.h:31: misc.h: No such file or directory
make: *** [main.o] Error 1
Compilation exited abnormally with code 1 at Sun Jul 24 16:32:17
```

Instead of manually locating libpx.h and jumping to the line in question, you can move point to the error message in the compilation buffer, press **C-c C-c**, and the source corresponding to the error is automatically visited. You can then correct the bug, move to the next error, and repeat. Under X, pressing **mouse-2** on an error message in the compilation buffer will jump to the corresponding source line.

If you have a large number of error messages, pressing **M-n** in the compilation buffer will move to the next error message (in addition to viewing the corresponding source). To cause **M-n** to have this behavior within the C source buffer as well, you can use the following command in your Emacs startup file: (**define-key c-mode-map "\M-n" 'next-error**)

To simplify the compilation process, I use the following code within my Emacs configuration file:

```
(defun my-save-and-compile ()
  (interactive "")
  (save-buffer 0)
  (compile "make -k"))
(define-key c-mode-map "\C-c\C-c' 'my-save-and-compile)
```

This defines a new function, **my-save-and-compile**, which will automatically save the current buffer and run **make -k**. This saves me the hassle of answering

the various prompts given by **M-x compile** alone. Now, using **C-c C-c** within a C Mode buffer will save the source and compile it.

Once you get used to the above mechanism, fixing bugs and recompiling code becomes quite painless—you can concentrate on debugging and let Emacs locate the errors, run make, and so forth.

Running gdb

gdb is the GNU debugger. It is indispensable for run-time debugging for programs written in nearly any compiled language, most notably C. **gdb** can also be used for post-mortem examination of a crashed program using a core file.

Not surprisingly, Emacs provides a number of features which allow you to run gdb within an Emacs buffer, interacting with the corresponding source buffers to view and edit code. While gdb deserves a tutorial of its own, here we will introduce you to the Emacs-specific gdb features. gdb provides extensive online help, which can fill in the gaps left here. For the rest of this tutorial, we assume that you have basic familiarity with gdb, or a similar debugger such as dbx.

Let's take the following short program, which will un-doubtedly cause a segmentation fault on most systems:

```
 #include
int main(void) {
  int i; int *data = NULL;
  data[0] = 1;
  data[1] = 2;
  for (i = 2; i > 30; i++) {
    data[i] = data[i-1] + data[i-2];
  }
  printf("Last value is %d0,data[29]);
}
```

As you can see, we're attempting to write data into a NULL pointer. Sure enough, when we compile and run the program, we obtain:

```
loomer:~/test/lj/crashme% ./crashme
Segmentation fault (core dumped)
```

Let's use gdb to inspect the problem. Using **M-x gdb** gives us the prompt:

```
Run gdb (like this): gdb
```

Here, you should complete the gdb command line. In this case, we want to run gdb on the executable crashme, with the core file core. So, we complete as so:

```
Run gdb (like this): gdb crashme core
```

Emacs should open two windows—one containing the gdb interaction session, and the other containing the source file crashme.c. The gdb session will look something like:

GDB is free software and you are welcome to distribute copies of it under certain conditions; type **"show copying** to see the conditions. There is absolutely no warranty for GDB; type **"show warranty"** for details.

```
GDB 4.12,
Copyright 1994 Free Software Foundation, Inc...
Core was generated by "crashme".
Program terminated with signal 11, Segmentation fault.
#0  0x22bc in main () at crashme.c:5
(gdb)
```

We can now issue gdb commands to inspect the crash. Immediately, we notice that the crashme.c buffer contains an arrow pointing to the current source line, as so:

```
=>data[0] = 1;
  data[1] = 2;
  /* ... */
```

This arrow is not part of the source text. It can't be selected, modified, or deleted. You are free to edit the code in the source buffer; this imaginary arrow will not be saved with the edited code. The arrow only exists to let us know what gdb's idea of the current source line is. Note, however, that adding or deleting lines from the source buffer will cause gdb's information about the location of source lines to be out-of-sync with the actual code.

We can see that the crash was caused by a segmentation fault on line 5, pointed to by the arrow. Using the where command in the gdb buffer will give us a stack trace, and so on. You can correct the code in the source buffer, recompile, test, and re-run gdb (if necessary), all within Emacs.

gdb can also be used to inspect running programs. For example, we can run crashme under gdb's control, and step along a line at a time. First, however, let's correct the bug by changing the definition of data to

```
int data[30];
```

(Otherwise, crashme would crash on the first line of code, and we'd have scant little to go on by way of demonstration.)

First, we should set a breakpoint on the first line of code. Within the gdb buffer, we can use list to display the first few lines:

```
 (gdb) list
1       #include <stdio.h>
2       int main(void) {
```

```
3         int i;
4         int data[30];
5
6         data[0] = 1;
```

The command break 6 will set a breakpoint at line 6:

```
(gdb) break 6
Breakpoint 1 at 0x22b0: file crashme.c, line 6.
```

Now, the run command will begin execution of the program, but will halt immediately on the first line of code. A buffer for crashme.c will be opened, with our friendly arrow pointing at the line containing the breakpoint.

Now, we can employ gdb's various commands directly, by entering them in the gdb buffer—or, we can use the Emacs key equivalents. Placing point on a line of code in the source buffer and typing **C-x C-a C-b** will set a breakpoint at that line. Similarly, **C-x C-a C-d** will delete all breakpoints on that line. (The gdb command info break will list the current breakpoints.) After setting a breakpoint, you can use **C-x C-a C-r** to resume execution.

All of the above commands can be used within the gdb buffer as well, using **C-c** instead of **C-x C-a** as the prefix. For convenience, **C-x SPC** in either buffer will set a breakpoint on the current source line.

If you find these key bindings unnecessarily lengthy, as I do, you might consider rebinding the functions **gud-break**, **gud-remove**, and **gud-cont** within **c-mode-map**. For example, I use the commands

```
(define-key c-mode-map "\M-b" 'gud-break)
(define-key c-mode-map "\M-d' 'gud-remove)
(define-key c-mode-map "\M-r" 'gud-cont)
```

Of course, this negates the previous meanings of **M-b**, **M-d**, and **M-r** within C Mode.

The following additional macros are available within the gdb buffer, as well as within C Mode by changing the prefix from **C-c** to **C-x C-a**.

- **`C-c C-s'** Step one line of code, descending into function calls. (The gdb step command.)
- **`C-c C-n'** Step one line of code, without descending into function calls. (The gdb next command.)
- **`C-c <'** Move up one stack frame. (The gdb up command.)
- **`C-c >'** Move up one stack frame. (The gdb down command.)
- **`C-c C-f'** Run until the completion of the current function, and then stop. (The gdb finish command.)

Again, you may wish to bind these to shorter key sequences (such as **M-s**, **M-n**, and so on).

Another interesting command is **C-x C-a C-p**, which will (within the source buffer) take the C expression around point and pass it to gdb's print command, which evaluates the expression and prints its value. This is very handy way to examine variables, data structures, and so forth within the debugger. You can even use this command to call functions and print the return value, if you're executing the debugged program within gdb.

For example, placing point on the line

```
printf("Last value is %d\n",data[19]);
and pressing C-x C-a C-p will cause the following to be printed in the
gdb buffer:
(gdb)
$1 = 16
```

In this case, data[19] is 0, because we haven't executed the calculation loop yet. Nevertheless, we can call functions within the program (or, in fact, any arbitrary function, using the print command manually) and examine the return value.

Emacs also allows you to define your own functions for interacting with the debugger. For example, we might want to move the point to a line of code, and run the gdb until function, which will cause execution to continue until that line is reached.

This is accomplished with the **gud-def** function. For example, we can use (**gud-def my-until-line "until %f:%l""\C-u" "Run until current line."**)

This will define the function my-until-line which sends the string

```
until %f:%l
```

to the gdb process, where **%f** is replaced with the current source filename, and %l is replaced with the current line number. The new function will be bound to the key sequence **C-x C-a C-u** (in the source buffer), and **C-c C-u** (in the gdb buffer). The final argument is the documentation string for the command, printed using **describe-function**.

Now, we can place point on a line of code in the source buffer, type **C-x C-a C-u**, and execution will continue until that line of code is reached.

We can customize interactions with the debugger in another way. For example, gdb lacks the inherent ability to automatically step along code, allowing us to monitor the execution of the program without interruption. A similar effect can

be achieved by using the step command many times in succession, but we'd like Emacs to automate the process for us.

This can be accomplished using the following function:

```
(defun gdb-step-forever (arg)
  (interactive "NTime between steps: ")
  (while -t
    (progn
      (sit-for arg)
      (gud-step 1))))
```

Running this function as **M-x gdb-step-forever** will prompt us for the amount of time to sleep between steps, in seconds. (This need not be an integral number of seconds—you can specify real values such as 0.5.) The function will then pause for the given amount of time, run **gud-step**, and repeat, ad infinitum. To interrupt the function, you can use the Emacs quit key, **C-g**.

A more general extrapolation of this idea would allow us to run a “hook” function between steps, which would allow us to print the values of variables, and so on.

Note that the above function isn't very intelligent—if it runs into a breakpoint, or the program ceases execution for some reason, it will continue to loop naively. In these cases, you can simply interrupt the function by hand.

Given the above tour, you should be ready to tackle the other programming features that Emacs provides—including version control, customized indentation styles, and so forth. Perhaps a future issue of *Linux Journal* will cover these aspects of Emacs.

I welcome all suggestions, comments, and corrections for the material presented here. Please feel free to send correspondence to the author, c/o *Linux Journal*, or via electronic mail at mdw@sunsite.unc.edu .

Happy hacking!

Matt Welsh (mdw@sunsite.unc.edu) is a programmer at the Cornell University Robotics and Vision Laboratory.He is a writer and programmer who uses vi almost exclusively—perhaps more by accident than design. He spends his free time homebrewing virtual beer and playing the blues.

Archive Index Issue Table of Contents

Advanced search

Selecting a Linux CD

Phil Hughes

Issue #6, October 1994

Confused over the many flavors of Linux CDs? Here, Phil Hughes gives us a brief summary of what's out there.

As the Linux movement has grown, so has the size of a Linux distribution. Buying a CD-ROM is now the only practical way to get a complete distribution. Fortunately, as the amount of code has grown, so have the buyer's choices in terms of distributions on CDs. And it continues to grow every month.

Linux is a very stable product with some people running distributions many months old. (Our server system at *Linux Journal* is running a distribution from October, 1993.) But each new release offers new features and new bug fixes. This means that new CDs are appearing virtually weekly. Here are some shopping hints.

Distributors, Packagers and ?

Although the name Linux really applies to the kernel, packages that include this kernel with lots of utilities and applications are commonly called Linux packages. Some of these packages are designed for a specific (and generally non-commercial) purpose. Others are designed specifically to be a retail product and others are a combination of the two.

To add to the confusion, there are the companies that develop the content of the CD and have them manufactured and there are companies that just distribute CDs manufactured by others or include mostly a distribution developed by someone else.

As the distinction between someone who has developed their own package and one who just sells code gets muddled more and more, the real issue comes up: is the product supported? Will the company that sold you the CD (or the manufacturer of the CD) offer you support if you need it?

This doesn't necessarily mean free support or support included with the product. The amount of support you can expect with a $20-$50 package is minimal, although that minimal “hand-holding” to get the package initially running might be important.

Some people are buying Linux to “hack”. If this is the case, support is probably not an issue. But if you plan to have the operation of your company depend on Linux (which is happening more and more today), make sure you can get the support you need before you jump in. That said, on to what is available.

Yggdrasil Computing is clearly the biggest of those who make their own package. Yggdrasil put together their own package (the first was called LGX and the current product is called Plug-and-Play) by selecting from available code. They then developed their own installation package running under X-Windows and made other decisions independent of most other Linux development.

Yggdrasil (and Yggdrasil distributors) sells a package which includes a boot disk and 90-page manual for $34.95. On the plus side, you get a shrink-wrapped complete package. But on the down side, you get Yggdrasil's idea of the system layout and utilities, making it harder to add other software at a later time.

If you want to follow the development mainstream, there are choices. The most popular Linux package today is Slackware, produced by Patrick Volkerding. It followed the general idea of the SLS package, which was to have an easy installation method that would allow the addition of more packages. The package has been freely distributed on the Internet, which means that it has most of the kinks worked out. Pat never intended to make such a package (he just wanted to clean up a few problems with SLS) but it just happened because it worked so well.

Many commercial packagers jumped on the Slackware bandwagon, including Trans-Ameritech, Morse Commu-nications and InfoMagic. But they did it in different ways. Trans-Ameritech produces a CD that includes Slackware plus BSD. The CD comes with bootable kernels and programs to write the floppy boot disks. The CD is $29.95.

Morse Communications produces The Linux Quarterly. They try to add value to each CD produced. The Spring 1994 edition includes programs that run under Microsoft Windows that allow you to read documentation and build the boot disk. It also includes the complete contents of the **tsx-11.mit.edu** archive site. This CD is $29.95.

Morse also has a Linux Professional package that consists of Pat Volkerding's Slackware and Matt Welsh's Installation and Getting Started book in a shrink-

wrapped package. [At the time of this writing we haven't seen this package; it may contain other things as well.] It sells for $49.95.

InfoMagic has decided to offer everything available in the way of code at a reasonable price. Their current distribution consists of two CDs which include all the Linux archives on both **tsx-11.mit.edu** and **sunsite.unc.edu** as well as the GNU archive from **prep.ai.mit.edu**, all of the Linux HOWTO documents with a browser that runs under Microsoft Windows, and the complete Slackware 2.0, SLS 1.0.5, Debian 0.91 beta and TAMU 1.0A distributions. The two-CD set sells for $20.00.

Other vendors who offer Linux on CD-ROM (but haven't yet sent us their products to review) include Red Hat Software, Nascent Technologies, S.u.S.E. GmbH (Germany) and Unifix Software GmbH (Germany).

Good Shopping

This information is current as of the end of July. By the time you read this there will probably be a new player or two and the current manufacturers may have new products. The current “production” kernel for Linux is 1.0.9 and the development kernel is 1.1.32. (If the second digit is even it is a production kernel built for stability. If it is odd it contains new features and is more likely to have bugs.)

If you want to be on the leading edge of development you may need to do some serious shopping. If, however, you just want a stable version of Linux, any of the distributions that contain a 1.0.x kernel should fill the bill. In any case, welcome to the exciting Linux movement.

Phil Hughes is the publisher of *Linux Journal*. He is a DeadHead who claims he's 33-years-old, and that he'll move to Montana as soon as he gets his staff trained.

Archive Index Issue Table of Contents

Advanced search

Report from the Front

Magnus Y. Alvestad

Issue #6, October 1994

The secratary of this volunteer group gives us the scoop on two Linux distributions.

The Linux Review Group is a group of Linux users willing to donate some of their spare time to testing Linux distributions. We have approximately forty testers, plus one secretary; me. Each time a distributor wants his product tested by us, five of our testers get a copy of it and review it according to a few criteria that we've chosen. This is all volunteer work, so now and then someone drops out and doesn't deliver his report. Therefore, I cannot guarantee that all the products we've looked at have been tested by as many as five people.

MCC Interim 1.0+

This is a free distribution, which we got by ftp from **ftp.mcc.ac.uk** in **/pub/linux/mcc-interim/1.0+**. It's available at **nic.funet.fi**, **tsx-11.mit.edu**, and **sunsite.unc.edu**

The best thing about MCC is the documentation: a 60+ page dvi document guides the user through the installation. There are also ASCII excerpts from this file.

The installation is menu-driven. The menus are not fancy (no color, scroll bars, etc.), but error conditions (i.e., no disk in drive) are caught.

There are couple of problems. The descriptions of the packages are a tad too short, and don't say whether each package is recommended or optional. Also, it displays the disk space use of each package after it has been installed. It would be nice to get that information before choosing whether to install a package.

Compared to Slackware, this is a pretty slim package. No soundcard support, no X-Windows, no TeX, etc. One tester complained that some of the “standard”

utilities that he was used to from Slackware were missing. How-ever, instructions for getting these parts are included. MCC only takes about 30MB of disk space; ideal if you're a newcomer to Linux and would like to try it out. There is no UMSDOS support, however, so you'll have to repartition your disk.

Linux Quarterly—Spring 1994

This is a commercial CD-ROM distribution from Morse Telecommunications.

A very polished package, Linux Quarterly comes with an MS Windows installation program. One of our testers wasn't able to get this installation program to run, but that might have been due to some local misconfiguration. The documentation is well-written and fairly complete. Of course, like most Linux material, it's not intended for the computer novice, but if you have some previous DOS or Unix experience, it'll do fine.

The CD contains about everything you'll want or need, including MCC, Slackware, an image of the Linux area on tsx-11, etc. It can provide hours of fun for the inquisitive. Free tech support is included in the package.

Magnus Y Alvestad (magnus@ii.uib.no) is a student of computer science who is the current keeper of the world record for single-machine factoring (113 digits). He is also the secretary of the Linux Review Group.

Archive Index Issue Table of Contents

Advanced search

Linux Journal Demographics

Laurie Tucker

Issue #6, October 1994

We know that at least a few of you are curious about those other folks.

Do you ever wonder who else reads *Linux Journal*? “No” you say? All your best friends and buddies already have their two-year subscriptions? All those friends live not necessarily in your neighborhood, but all over out there in Internet-land?

Well, we know that at least a few of you are curious about those other folks. (And besides, it's a chance for us to put some colorful pie charts on this page and dazzle you with statistics!)

The distribution of *Linux Journal* as of the October Buyer's Guide issue was 30,000. Our readers include subscription-holders as well as those who buy the magazine in bookstores and other retail outlets.

Of our subscription-holders, 70 percent are in the United States, and 30 percent are non-US. Inside the U.S., most magazines get sent to the famous area of the country known as “other”. Our next-largest subscribership is the state of California with 17 percent. Washington state has a large percentage because we threaten all of our friends to help keep us employed! We had to put Washington D.C. on the chart because people are always getting us mixed up with them.

The non-North America pie shows Great Britian, Germany, and Asia each at approximately 17%. Australia and New Zealand have a surprising 7 percent. Miscellaneous Europe lets us know that there are plenty of people in the Netherlands to go to the International Linux Symposium which will be held in Amsterdam in December. (See related story in the Linux Events section, page 10.) Their 14 percent is second only to France's 22 percent.

So, we hope that this gives you a little better picture of the demographics of *Linux Journal* subscribers. Oh, and don't forget to get an extra subscription for your Mom for her next birthday.

LJ Total Subcriptions

LJ U.S. Subcriptions

LJ Non-North America Subcriptions

LJ Misc. Europe Subcriptions

Laurie Tucker is the assistant editor of Linux Journal, cover designer of the September issue, and sysadmin of ssc.com; a Linux system. She hides out at info@linuxjournal.com.

Laurie Tucker (info@linuxjournal.com) is the assistant editor of *Linux Journal*, cover designer of the September issue, and sysadmin of linuxjournal.com; a Linux system

Archive Index Issue Table of Contents

Advanced search

Code Freeze

Linus Torvalds

Issue #6, October 1994

Lots of other stuff has also changed in the 1.1.x releases; sorry for not doing release-notes, but I'm too lazy.

Linus Announces Code Freeze in Preparation for 1.2.x

```
From: Linus Torvalds torvalds@cc.helsinki.fi
Newsgroups: comp.os.linux.announce
Subject: Approaching 1.2.x, I hope
Date: 30 Jul 1994 02:05:45 +0300
```

I'm slowly making ready for something looking like a code-freeze for 1.2.x, and that means you can all start doing your favorite pre-release stuff: doing weird things to the latest kernels and seeing how they break. And maybe even sending me in a report (or patches if you feel like it).

The latest kernel right now is 1.1.36 (but they have changed daily) and contains the “mprotect()” system call that some people have been asking for. The last kernels have gone through major re-organizations in the memory manager, so we'll see how well it works out. Also, I wrote the mprotect stuff from scratch instead of using any of the old patches, so that's rather untested. If you have something depending on mprotect, do give this one a try.

(Aside: The mmap() interface still doesn't allow shared writeable mappings, but now you can do a shared read-only map and then “upgrade” it with mprotect(). That's not supposed to work, but I didn't bother to put in the extra checks, as I hope to have real write-mappings working some day. Going through mprotect is likely to give bogus results, etc; don't even try it as the kernel may do strange things.)

Lots of other stuff has also changed in the 1.1.x releases; sorry for not doing release-notes, but I'm too lazy. Essentially everything is faster, bigger and better, but it may be a bit unstable which is why I'd like people to test it out. The credit goes to everybody who has written code and tested so far (including, but

not limited to Alan Cox, Eric Youngdale, Mark Lord, Jacques Gelinas, Hannu Savolainen, Frank Lofaro, Rik Faith, Bjvrn Ekwall, Remy Card, Dmitry Gorodchanin...,the list goes on forever).

Anyway, I hope 1.1.40 (or 1.1.50 or whatever) will turn out stable enough to be called 1.2.0 so that people who want to use mainly stable kernels know which version to get. Sadly, everything always works perfectly for me, so in order to find the problems some outside help is needed.

—Linus

Archive Index Issue Table of Contents

Advanced search

Harbor

Michael K. Johnson

Issue #6, October 1994

For Internauts, finding a port of call can be a trying experience. I recently ran the gauntlet of choosing a commercial Internet access provider (do I have enough mixed metaphors yet?) and would like to share my experiences, both good and bad.

The other day, I started shopping, both for a Unix shell account, and for a SLIP or PPP connection that allows my home Linux network to become a real part of the Internet. Because I almost always use the Internet after 5:00pm, I made my calls in the evening, in order to find out whether I could expect to find anyone available to fix problems that occur when I am logged on. I've gone through this exercise before, and found myself frustrated. This time, it ended up better.

Since I'm going to divulge conversations that I had with providers, and since I didn't record the conversations to have as proof if I were to be sued, I'm not going to divulge the names of the providers I didn't choose. But never fear; by calling major Internet service providers and asking the right questions, you can get the same answers I did (more or less) and play “match the provider”. And if they have come up with better answers in the meantime, then my comments no longer apply to them; it's up to you to decide what you want.

When I called service provider “A”, I was given a low price for a standard Unix shell account with unlimited access time and a 5MB limit. Everything sounded quite good until I said the magic word “SLIP.” “A” said that it provides SLIP connections only with MS-Windows software. I explained that I use Linux, and that their Windows software was useless to me: I wanted to be a part of the Internet. They assured me in no uncertain terms that Linux support was coming someday, but declined to say when. I explained that Linux comes with SLIP and PPP support, and asked what they intended to do differently. The answer? “Oh, that's a technical question! I can't answer that! Call our technical support at 123-555-5555 and ask them.”

I was not interested in paying a toll call to hear someone explain why “A” couldn't support my operating system of choice, so I did not call back. On the other hand, they probably weren't interested in paying to explain to me why they wouldn't support my operating system of choice right now, and why they have to write their own SLIP for Linux, so the feeling is most likely mutual. I was angry enough to choose never to buy any Internet services from them. Many customers couldn't care less.

It was still late afternoon or early evening; before supper, anyway; when I called “B”, with high hopes. Inexpensive, unlimited SLIP and shell access in one package, for a rather low fee. The person who answered the phone was not able to answer any questions, and did not even have any local modem numbers. She was able to take a message for someone to call me back the next day. Well, she said someone would call back. I haven't received a return call yet. Furthermore, the modem number advertised in “B”'s advertisement (one could, in theory, log in and try out their service before making a purchase) didn't even ring. Still hoping, I called the help desk number that they advertised, thinking that they could help me make a connection. I let the phone ring for a minute or two, until the phone company cut off the connection, but they never answered the phone.

Another service provider, “C”, had very good service: they were available in the evening, have V.Fast class modems, the sales people are well-versed in technical issues, and they don't mind users using Linux to connect. Unfortunately, unlimited SLIP would have cost me a $300 setup fee and $300/month. It was not clear that I could have Internet traffic for both of my home machines routed at that price; I might have had to buy a commercial connection to get that. Furthermore, my machine would have to be in their domain: **mymachine.foobar.c.*net*** instead of **mymachine.mydomain.*org***.

And so it goes, and so it went, and went, and went, until I called (and here I break the anonymity rule; I doubt they will sue me for a pleased report of their service) Vnet. It was a little later now, but the person I spoke to on the phone was technically good. And the service was inexpensive: $25/month for a full shell account (up to 80 hours/month), and $50/month (with a one-time $15 setup fee) for unlimited SLIP or PPP access. I said that I would like to start with a shell account and move to a SLIP or PPP account soon, and that was fine with them.

Now it really got good. When I mentioned that I was currently connected through sunsite.unc.edu, he said, “Are you a Linux user?” I responded that I am; I have two machines running Linux. “Are you?” I asked. “Installed Slackware a few weeks ago. I love it. It's the best way to connect to the net.” (As I said, I didn't record this call; I may not be quoting him perfectly. Bear with me.) “Oh,

and when you convert to SLIP, we will give you a dip script that you simply change a few words to configure for your machine, and it will dial in and connect you properly." We went on to chat about Linux and *Linux Journal* and get the account details set up, and 15 minutes later my account was ready.

When I logged on that night, I sent e-mail to their on-line help desk, and got fairly quick responses, even after midnight. I don't know that they always have someone on-line after midnight, but the replies I got were quick, helpful, and friendly.

There is hope. Just when you think that you have to hide the fact that your computer is running Linux in order to get a service provider to allow you to connect, your luck may turn. You can find a helpful provider. Spend the time asking potential service providers tough questions, and the good ones will shine through. I'm sure Vnet isn't the only good service provider, and they might not even fit your circumstances, but you can't know without asking: info@vnet.net or (voice numbers) (800)377-3282 or (704)334-3282. Check local computer papers for advertisements for local service providers that I can't direct you to. Several books are available in most bookstores that carry computer-oriented books that have much more exhaustive lists of local, national, and international service providers. Assume that any pricing data in the books is out of date; prices drop unevenly all over, regularly.

Grades of access

Michael K. Johnson is the editor of *Linux Journal*, but that doesn't keep him from hacking.

Archive Index Issue Table of Contents

Advanced search

Linux Events

LJ Staff

Issue #6, October 1994

Linux Symposium in Amsterdam and Linux Track at Open Systems World.

Linux Symposium in Amsterdam, December

The International Symposium on Linux will be held in Amsterdam on December 8 and 9, 1994. This event will take place in the RAI Congress Centre. Students get in cheap for about $50, others must pay about $75.

The event is sponsored by the ICCE, University of Groningen, and is organized by Frank B. Brokken, Karel Kubat, and Piet W. Plomp. This non-profit group is planning on lowering the entrance fees if enough attendees register.

The most current information about the symposium is available via anonymous ftp at beatrix.icce.rug.nl in the directory pub/symposium. It is refreshed daily, and contains a list of speakers, a list of interested attendees, and information about local hotels. The organizers of the symposium can be reached at linux@icce.rug.nl.

Some of the twenty-five speakers already scheduled include Bob Amstadt, Remy Card, Michael K. Johnson, Linus Torvalds, Theodore Ts'o, and Matt Welsh. Formal lecture topics include “Viability of Linux”, Ham radio and Linux, “Typesetting, X and MS-Windows”, “Linux and UnixWare; a comparison”, “Linux in Biostatistic Research”, “Development of Linux and the Role of the Expert Community”, “Onyx”, Wine, and “Programming in a Multi-Threaded Environment”.

People without Internet access can reach ICCE at:

ICCE, Univ. of GroningenP.O. Box 3359700 AH GroningenThe Netherlands(+31) 50 63 36 47

Linux Track at Open Systems World

The week before all the fun starts in Amsterdam, *Linux Journal* will be hosting the Linux Track at Open Systems World. This event, previously known as FedUNIX, is a week-long affair and will be held in Washington, D.C. from November 28 until December 2, 1994.

This is a chance to rub elbows with SCO, Solaris, Novell, and Microsoft, who will all be there hosting their own conferences, helping to bring in some of the 10,000+ expected attendees. This is a huge opportunity to present Linux to a large number of government and corporate IS managers, administrators and executives, systems integrators, hardware and software developers, industry analysts and journalists.

The Linux Track will include two days of tutorials and panel discussions presented by some well-known personalities in the Linux Community, including Bob Amstadt, Eric Youngdale, and Don Becker. Currently planned topics include commercial use and future of Linux, Wine, Linux and NASA, legal implications of using and developing tools and applications on Linux, iBCS2 compati-bility, X Windows System on Linux, a beginner's clinic, Linux and the Internet, relationship between Linux resellers and the Linux development community, Linux capabilities, and how to convince your boss/employer/ customer to use Linux.

For those of you who are not able to go to the International Symposium in Amsterdam, Open Systems World is a chance to participate in a conference on the other side of the Atlantic. And those of you who are going to the Netherlands can use the Open Systems World event as a spring-board before your journey.

Keep a watch out here in *Linux Journal* for more information in the November issue.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux

Matt Welsh

Issue #6, October 1994

This month, Cooking with Linux looks at a problem faced by most would-be Linux users—choosing the right distribution.

It's the time of month again when I manage to dig myself out of the piles of source code listings, overdue bills, and technical manuals that invade my desk, just long enough to take a deep breath and make a wry comment on the status of the computing community. Please excuse any spelling errors; my eyes are still adjusting to the light.

By now, you've (hopefully) gone over this Buyer's Guide issue with a toothpick and magnifying glass, noting every minute detail, pitting distribution against distribution, vendor against vendor, in a fierce battle of marketing wit and technical expertise. Unless, of course, you flipped immediately to this column, realizing; along with so many other readers; that this is the only part of *Linux Journal* that condenses any genuinely useful information.

At any rate, after digesting the product reviews and coverage of major Linux distributions and software, you've more than likely reached the stark conclusion that you're no better off than before you picked up the magazine. There are just *too many* distributors out there, and too many software distributions to choose from. How on earth can anyone decide amongst them all?

You may be tempted to employ the infamous Monte Carlo method. Blindfold yourself. Open the magazine to a random page. Put your finger down. Remove blindfold. Pick up the phone and order whatever your finger happens to be pointing at. (“Hello? Yes, I'd like to order... uh... *Ian Murdock* please. What? He's not for sale?”)

Certainly you can see the futility of this approach. A better method is the one that I suggest to true die-hard Unix hackers who have, oh, a few months to kill.

That is to forego this distribution nonsense and to put together your own system from the kernel up. Can't be done, you say? Impossible? Not so. The only requirements are: One (1) personal computer. One (1) small Linux system on a floppy; for example, the Slackware boot/root disk combination. And, one (1) Finnish computer science student named Linus Torvalds. If you happen to have all of the above ingredients, you should have no problem at all.

Needless to say, things were easier back in the Good Old Days, when the only “distribution” was a pair of floppy images made available by H.J. Lu. Back then, users had no choice but to build a system from scratch. Those rugged survivalists that made it through the ordeals of Linux prehistory are now known as “old hats”. Some of them live in caves. (Some of the caves have Ethernet drops, which is certainly convenient.) Others have moved onto bigger and brighter things, such as writing editorials for a certain Linux-related magazine. Still others are nowhere to be found.

All right, all right, but what about the rest of us; those who don't have the hard-headedness required to install Linux the old-fashioned way? After all, this magazine is literally teeming with Linux distributors, right? Can't the choice of a Linux vendor or distribution be boiled down to a simple, straightforward, no-frills answer? Why the runaround? Why don't I get to the point?

I seem to have forgotten the point.

But I do remember this: Selecting a Linux distribution is not unlike buying a new car. You are faced with many questions: Do you want a sporty and fun convertible (Slackware), or a family-sized minivan (Yggdrasil)? Or are you comfortable enough with your Unix hacking repertoire to test-drive an experimental new design (Debian)? Do you want to go for a newer, and perhaps more expensive model, with all of the options (anti-lock brakes, air conditioning, and the Linux Documentation Project manuals), or are you content with a rugged, do-it-yourself version (such as the InfoMagic CD-ROM set)?

The list is endless. The best advice that I can give is to talk with other Linux users, who have survived the adventure of selecting and installing a particular distribution, and hear about their experiences. Given the fact that Linux is free, you can always borrow or copy the software from a friend. Using the UMSDOS filesystem, you can even install Linux in a directory of an MS-DOS partition, saving yourself from the time-consuming and destructive repartitioning process; which usually requires you to backup the entire system.

So, what are you waiting for? Pick a distribution and give it a spin. Kick the tires. Find a country road and floor it. Crash it a few times, if you can. If it works, and if it feels like the system for you, take the plunge and buy the darn thing.

Fuzzy dice and vanity plates sold separately.

Matt Welsh (mdw@sunsite.unc.edu) is often seen standing by the roadside of the Information Superhighway, holding a cardboard sign, which reads: “Will Hack Linux for Food”.

Archive Index Issue Table of Contents

Advanced search

Linux Programing Hints

Michael K. Johnson

Issue #6, October 1994

Most Linux users have at least heard of Makefiles, but many do not know how powerful a program make is. It is thought of as a tool for maintaining other programs, but it is far more. It can make sense out of chaos in any project where some files are created from other files, whether the end product is a program or a book or an automated post to Usenet. Even if you have never written a makefile, this tutorial will set you on your way to using make effectively.

An Introduction to make

Most Linux users have at least heard of Makefiles, but many do not know how powerful a program make is. It is thought of as a tool for maintaining other programs, but it is far more. It can make sense out of chaos in any project where some files are created from other files, whether the end product is a program or a book or an automated post to Usenet. Even if you have never written a makefile, this tutorial will set you on your way to using **make** effectively.

by Michael K. Johnson

Many people are confused by **make**: maybe you are too. You know that it is hard to use, because it has a weird syntax unlike any other program you use. If you are lucky, you have been warned that it is important to have tab characters (not spaces) in certain places, and you know that if you mess up a makefile you won't be able to fix it.

Writing a makefile of your own is out of the question. It is difficult, and besides, you aren't really a programmer, anyway. What could this programmer's maintenance program do for you, and why should you learn its weird syntax? Or maybe you are a programmer, and you don't want to learn this tool called make, which has a syntax different from any language you have ever learned.

The reason for the weird syntax is that make does a job very different from normal programming tools, and it is well-suited to that very different job. Understanding the job is the first step to understanding make. Once you understand the job, and have learned a little bit about make, you will be able to write short, powerful makefiles.

We will pretend for the moment that you are writing a book, although the exact same ideas apply to writing a program. (I just want to emphasize that make isn't just for Real Programmers.) You are using LaTeX. Each chapter has several figures. These figures are done in xfig, and need to be converted to PostScript format with fig2dev before being included in your book.

Here comes the problem. You are occasionally editing the figures with xfig, and forgetting to make a PostScript copy of each figure when you are done, so you write a large shell script that converts xfig to encapsulated PostScript (EPS) for each figure. It is large, bulky, and inflexible, but you get the job done right each time you print. Unfortunately, it takes a while to convert all the files from xfig to encapsulated PostScript. Even if you have only made a minor change in one figure, it re-converts all of your figures. This is annoying, and takes a while for it to convert your 80 or so figures.

Welcome to the wonderful world of make

The intelligence of make is summarized by two concepts: dependencies and rules. You need to tell make that the dvi file (we'll call it book.dvi) needs to be created from the main LaTeX file (book.tex), and from the encapsulated PostScript files (*.eps). In make terminology, book.dvi depends on book.tex and *.eps. Also, each of the .eps files depends on the corresponding .fig file. You also need to give make rules for turning book.tex into book.dvi, and for turning the .fig files into .eps files.

Newbie Note:All lines containing shell commands in your makefile must start with the TAB character.

Here is an example of a makefile that will do everything that is necessary:

```
0:  # This is a makefile to create book.dvi
1:  EPSFIGS = fig1.eps fig2.eps fig3.eps fig4.eps \
2:  fig5.eps fig6.eps fig7.eps fig8.eps fig9.eps \
3:   <... more .eps files, too many to print ...> \
4:   fig78.eps fig79.eps fig80.eps fig81.eps
5:
6:  book.dvi: book.tex $(EPSFIGS)
7:          latex book.tex
8:
9:  fig1.eps: fig1.fig
10:         fig2dev -Lps fig1.fig fig1.eps
11:
12: fig2.eps: fig2.fig
13:         fig2dev -Lps fig2.fig fig1.eps
14:
```

```
<many more similar rules that you can imagine>
220:
```

This file can be saved as **makefile** or **Makefile**; either will work. If you have both files in the same directory, **makefile** will be used instead of **Makefile**. In addition, there are other names that can be used and rules to control that. See the *GNU Make* manual if you care (you usually won't). People almost always use **Makefile** because capital letters show up at the top of directory listings.

The first line, line 0, is a comment. Line 1 starts to define the variable EPSFIGS. The backslashes continue the line, so the EPSFIGS variable contains the names of all the EPS files from fig1.eps through fig81.eps, and logically all those lines are really one long line. Line 6 tells make that if book.tex or any of the .eps files are newer than book.dvi, then book.dvi has to be recreated. Line 7 explains how to do this. It is very important that this line start with a TAB character. This is how make knows that this is a shell command to be executed to update the dependency that preceeds it. Eight spaces will not work. Spaces can follow a tab, but the first character on that line must be a TAB. The rest of the lines work the same way: line 9 says that fig1.eps depends on fig1.fig, and line 10 tells make how to update fig1.eps from fig1.fig if fig1.fig has been updated since the last time fig1.eps was created.

Simply typing **make** will automatically make book.dvi, because the first target in the makefile (here consisting of lines 6 and 7) is the **default target**. You could conceivably type **"make fig1.eps"** to just update fig1.eps from fig1.fig, and that only if necessary. If it is not necessary, make will tell you **"fig1.eps is up to date."**

The basic syntax of a makefile can be simplified to this:

- Any line can be continued onto the next line by making the last character of the line be a backslash character.
- Variables are defined with lines containing an equal sign: **FOO=bar**.
- Variables are referenced by enclosing them in parentheses (or curly braces, but parentheses are preferred and are more portable) and prepending a dollar sign: **$(FOO)** (or **${FOO}**).
- Files are made to depend on others by putting the file that is created before a colon, and a list of files needed to create or update that file after the colon, on the same line.
- A list of shell commands for creating or updating the file follows that line directly on lines with a TAB character as the very first character. Each line is run by a separate invocation of the shell, so a cd command on one line will only have effect on that line. To make successive lines be part of the same shell invocation, append “ ;\” to the line to make the next line really be another part of the same line.

- Comments begin with the “#” character.

Knowing those six very simple rules will allow you to maintain most makefiles that you will find on the Internet, and will allow you to create almost any makefile you need. However--

Don't get satisfied yet

This makefile works until you define fig82.fig and forget to translate it into fig82.eps. It is also far longer than it needs to be. I will rewrite it to be much smaller, and at the same time work better. I'm going to assume that you are using GNU make, which is the standard make on Linux and some other systems. It is very easy to build on most other systems that you may have.

```
0: # This is an improved version of the makefile for
1: #  making book.dvi
2: # FIG is a list of all the native xfig drawings
3: FIG = $(wildcard *.fig)
4: # EPS is a list of EPS files to create from the xfig files
5: EPS = $(subst .fig,.eps,$(FIG))
6:
7: %.fig : %.eps
8:          fig2dev -Lps $< $@
9:
10: book.dvi: book.tex $(FIG)
11:         latex book.tex
12:
13: print: book.dvi
14:         dvips -r -Z -q book.dvi
```

This makefile does everything that the previous one did, and more. It introduces several new concepts along the way, some of which are unique to GNU make. The first, and perhaps the most obvious, is that I did not explicitly list all the .fig or .eps files. Listing all the files requires me to update the makefile each time I add a figure, which is rather error-prone. Using the GNU make wildcard function allows the makefile to automatically update the files correctly, no matter how many figures you add. If you choose to use it, be careful that you really mean to list all the files, and be aware that it will not work under most commercial vendors' versions of make.

GNU make has many functions like wildcard. Like variables, they are enclosed in **$(...)** (or **${...}**), but you can distinguish functions from variables because the functions have embedded spaces. The name of the function comes first, with the argument, or comma-seperated list of arguments, following. They are all documented in Chapter 9, “Functions for Transforming Text”, in the GNU Make manual. wildcard is equivalent to the shell “globbing” that you use every day on the command line (like **ls *.fig**) and subst works like a very simple version of sed (like **sed 's/\.fig/\.eps/g'**), replacing each instance of one substring in a string by a different substring.

Another GNU make feature used is called a “pattern rule”. This is like a “suffix rule” as used in all versions of make (suffix rules still work in GNU make, don't worry), but is more powerful. The example given here is equivalent to the suffix rule

```
.fig.eps:
        fig2dev -Lps $< $@
```

but the pattern rule is a more general construct. Pattern rules may by preferable if you are writing makefiles for yourself or for projects where you can guarantee that GNU make will be used, and suffix rules are preferable if you are trying to write portable makefiles for all platforms.

The reason pattern rules are more powerful than suffix rules is they can match any sort of pattern, not just suffixes. For example:

```
foo.% : bar.%
```

would be a “prefix rule”, which does not exist in other forms of make. Patterns are a more general concept than suffixes, and can be used in other ways in other kinds of rules. The GNU Make manual has examples of this.

The variables **$<** and **$@** are special variables that can be used in rules. The **$@** variable stands for the file that is being updated, the target which depends on the files that follow the “:” in the dependency. The **$<** variable stands for the files upon which the target depends. This is a standard make facility, and does not require GNU make.

Finally, this makefile uses a standard trick of defining a target that doesn't exist. Typing **make print** will print a copy of your book with everything up-to-date, even though there is no file called print. Similarily, since the first target is the default target, many makefiles have the first rule be a rule called “all:” which does whatever seems right when “everything” is to be built. In fact, the practice of typing “make all” instead of just “make” is so common that even when there is only one program to be built by default, some people will add an all target to their makefile even though it is not at all necessary:

```
0: all: foo
1:
2: foo: foo.o bar.o baz.o
       <and so on>
instead of
0: foo: foo.o bar.o baz.o
       <and so on>
```

This is only a taste of make. The GNU Make manual is a gentle, but much more thorough and correct introduction to GNU make, and is distributed with the GNU make source code. It does not assume that you already know how to use any version of make. On a properly installed system, it is available through the

info system. The info files can be read from within Emacs (type **C-h i** from within Emacs) or from a standalone info reader such as info or tkinfo. Alternately, the book can be printed with the TeX typesetting system or ordered from the Free Software Foundation.

Michael K. Johnson is the editor of *Linux Journal*, but that doesn't keep him from hacking.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

What's GNU: Texinfo

Arnold Robbins

Issue #6, October 1994

This month's column discusses the Texinfo document formatting language, which is used for all GNU documentation. Its main feature is that one source file can be used to prepare both printed text, and on-line, hypertext-style documentation.

We will just cover the high points of Texinfo. There is a complete manual (around 200 pages) that describes Texinfo in full detail. If you intend to actually write a manual, or do major surgery on an existing one, then it will be worth your while to read the Texinfo manual first. (Nicely printed and bound versions of the Texinfo manual can be purchased from the Free Software Foundation. Doing so not only saves wear and tear on your laser printer and gives you a nicely bound book, it also directly contributes to the production of more free software.)

Texinfo comes in the **texinfo-3.1.tar.gz** software distribution from the Free Software Foundation (**ftp://prep.ai.mit.edu/pub/gnu** is the site). This contains all the software (except TeX) needed to use Texinfo.

Processing Texinfo Files

Texinfo is designed to be processed by two (really three) different sets of programs. The first is TeX, the text processing system written by Donald Knuth. TeX is a freely available, fairly large software suite. If you acquired a CD-ROM Linux distribution, TeX was probably already set up for you. The file **texinfo.tex** is a series of TeX macros that tells TeX how to format Texinfo files, and must be installed in the TeX macros directory.

Texinfo allows you to produce output in several paper sizes: 6 by 9.25 inch (normal book), 8.5 by 11 inch (standard American sheet of paper) or A4 (standard European paper).

The second program is called **makeinfo**. This program reads your Texinfo files and generates a formatted Info file or files. These files can then be viewed with an Info viewer. If you are a GNU Emacs user, then you have a third option, **texinfo-format-buffer** within Emacs. This will also generate a formatted Info file. The **makeinfo** program is preferred; it is written in C and is faster than **texinfo-format-buffer**. It is also somewhat smarter, and it does not require Emacs underneath it.

To view an Info document, you need an Info viewer. There are two choices, Info mode in GNU Emacs, or the stand-alone **info** viewer that comes with the Texinfo distribution. We'll talk about viewing a little bit later.

An important point to remember when writing Texinfo documents is that you are writing for two audiences; those using info and those reading the printed book. This dual nature of the document has a strong influence on the design of the Texinfo language; there are a number of constructs that are used for one case and ignored for the other. There are Info-to-HTML converters in use at a number of WWW sites on the Internet, and **makeinfo** is being enhanced to generate HTML directly, although there is not yet a release date for this version of makeinfo.

What is Texinfo?

Texinfo documents are structured like conventional books, with chapters, sections, subsections, and subsubsections. (Four levels is as deep as you can go, but that is almost always more than you need.) Each chapter, section and so on is referred to as a “node”. In fact, this structure is actually what Computer Scientists like to refer to as a tree. The “root” of the document is a special node called the “top” node. Computer Scientists draw their trees starting at the top and growing downwards. **********Place figure here

Basic Structure Of A Texinfo Document

We'll take an abbreviated tour through a Texinfo document.

```
\input texinfo  @c -*-texinfo-*-
@c %**start of header (This is for running texinfo on a region.)
@setfilename Foogol.info
@settitle Foogol Language Programming
@c %**end of header (This is for running texinfo on a region.)
```

This is the front of the document. It reads in the Texinfo macros, and sets the Info file name and the title to be used in page headers. The @c starts a comment. Shown here are magic comments that make GNU Emacs happy (the **%**start of header**, and so on). These comments are boilerplate; just put them there and don't worry about what they mean (I don't; I'm not an Emacs user).

```
@ignore
@ifinfo
@format
START-INFO-DIR-ENTRY
* Foogol: (foogol.info).   A Wonderful Programming Language.
END-INFO-DIR-ENTRY
@end format
@end ifinfo
@end ignore
```

This is the entry for the **dir** file in the **/usr/local/info** directory. This is how info knows what Info files are available. Most of the text surrounding the middle line is so that separate shell scripts can find and extract this line and add it to the dir file. Unfortunately, the FSF is not yet distributing these scripts. In short, this is more boilerplate; the middle line is the important one. It describes your document in one line. Right now, you have to manually update the dir file when installing a new Info file in the **/usr/local/info** directory.

This shows a number of things about Texinfo. First, the @ is the command character. All special constructs in Texinfo start with @. (Use @@ to get a real "@".) Second, many constructs come in pairs, **@foo ... @end foo**. Things between **@ignore** and **@end** ignore are skipped by all the Texinfo processors.

The **@ifinfo ... @end ifinfo** lines bracket text that is meant only for an Info file, and not for printed TeX documentation. There is also a corresponding **@iftex ... @end iftex** construct for things that should only be in the printed document, and not in the Info file.

The **@format** and **@end** format bracket example text that is not indented. (In this case, since it's being ignored, it has no effect.)

(The whole business with the dir line is a bit unusual. It is a feature in transition, that will eventually become a mainstream part of the Texinfo language. Right now, it's done as shown, with everything ignored by the Texinfo processors. You don't have to do this in your Texinfo documents, but it doesn't hurt, either.)

Next comes information describing what is documented, the edition of the document, and the version of the program being documented. This is followed by the copyright information and permission notices.

```
@ifinfo
This file documents @code{foogool}, a programming language that does
really neat things.
This is Edition 1.23 of @cite{The Foogol Language}, for the 3.21 version
of the GNU implementation of FOOGOL.@refill
Copyright (C) ...
Permission is granted ...
@end ifinfo
```

The full permissions are documented in any FSF manual and in the Texinfo documentation. They are omitted here for brevity. In this example we also see how Texinfo does in-line font changes. **@code{xxx}** puts "xxx" into a fixed-width

font, like this **xxx**. In Info, the text is quoted. The **@cite{}** is used for citations of titles; it sets the enclosed text in italics.

Next comes the title page. Almost all of it is boilerplate; things that have to be there in a set order, where you fill in the values that apply to what you are writing.

```
@titlepage
@title The FOOGOL Language
@subtitle A User's Guide for GNU FOOGOL
@subtitle Edition 1.23
@subtitle July, 1994
@author Arnold D. Robbins
```

This starts the actual title page, listing the title, subtitle(s), and author(s) of the document. Title pages only apply to printed materials.

```
@c Include the Distribution inside the titlepage
@c environment so that headings are turned off.
@c Headings on and off do not work.
@page
@vskip 0pt plus 1filll
Copyright @copyright{} 1994 Free Software Foundation, Inc.
@sp 2
This is Edition 1.23 of @cite{The FOOGOL Language}, @* for the 3.21
version of the GNU implementation of FOOGOL.
@sp 2
Published by the Free Software Foundation @*
675 Massachusetts Avenue @*
Cambridge, MA 02139-3309 USA @*
Phone: +1-617-876-3296 @*
Printed copies are available for $20 each.
Permission is granted ...
@end titlepage
```

This does the back of the title page, called the copyright page. It lists the copyright information, edition information, information about the publisher, and copying permissions. Again, the Texinfo manual has the full details. This example assumes a GNU manual, but of course, if you are writing a manual, you or your company is the publisher. The @* forces a line break, so that each line in the Texinfo source file will be printed on a separate line in both the printed and Info versions.

```
@ifinfo
@node Top, Preface, (dir), (dir)
@top General Introduction
@c Preface or Licensing nodes should come right
@c after the Top node, in `unnumbered' sections,
@c then the chapter, `What is foogol'.
This file documents @code{foogol}, a programming language that does
really neat things.
This is Edition 1.23 of @cite{The Foogol Language}, for the 3.21 version
of the GNU implementation of FOOGOL.@refill
@end ifinfo
```

This is the special “top” node. It should appear only in the Info file, which is why it is bracketed in **@ifinfo** and **@end ifinfo**. It has a brief description of what the Info file documents.

Here, for the first time, we see an actual **@node** statement. The **@node** lists, in order, this node's name, it's successor, it's predecessor, and the “up” node. The successor and predecessor are usually nodes at the same level in the tree; the up node is the parent. In the case of the top node, it's a little bit different. The successor is the first child node (chapter) in the document, and both the predecessor and up node are the special (**dir**) node that represents the dir file.

Following any node with children is a menu. The menu lists all of the node's children (usually in order), with a brief description of what each node describes.

```
@menu
* Preface::        What this Info file is about; brief
                  history and acknowledgements.
* Copying::        Your right to copy and distribute @code{foogol}.
* What Is Foogol:: What is the @code{foogol} language;
                  using this Info file.
* Getting Started:: A basic introduction to using @code{foogol}.
                   How to run a @code{foogol} program.
                   Command line syntax.
.....
* Index::          Concept and Variable Index.
@end menu
```

The menu for the top node is called the “master menu”. It has entries for at least all the chapter level nodes in the file. More commonly, it has entries for every node in the file, almost in order. Usually, all the chapter level nodes are first, and then all the “interior” nodes come after that, in the order they would be in a book. After all the stuff at the front, you simply settle down and write your document. Each node has a unique name. In **makeinfo**, case in node names is ignored, while in TeX it is not, so for best results, keep the case in all instances of a node name the same.

Each **@node** statement is followed by a sectioning command.

```
@node Introduction, Syntax, Preface, Top
@chapter Introduction To Foogol
The @code{Foogol} language was invented in 1897. ...
```

There are corresponding **@section**, **@subsection** and **@subsubsection** commands. There are also commands for unnumbered chapters and sections, **@unnumbered**, **@unnumberedsec**, **@unnumberedsubsec**, **@unnumberedsubsubsec**, and similarly for appendices as well, **@appendix**, **@appendixsec** and so on. There are even commands for titles that won't appear in the table of contents: **@heading**, **@subheading**, and so on, although these still require separate node names and menu entries.

As you might imagine, keeping the “next”, “previous”, and “up” pointers in synch while writing a document is a bit painful. Fortunately, as long as you have the correct sectioning commands and each node with children has a correct menu, makeinfo will figure out the pointers for you. In practice, this means that you

can leave them off. TeX does not care a whole lot about nodes, although it does not ignore them entirely, as we shall see.

The Texinfo major mode in GNU Emacs will construct menus for you, and also automatically update the “next”, “previous”, and “up” pointers. If you don't use Emacs, you must do this manually, or write a separate program to do it for you (which is what I did, using gawk).

At the end of your document, you need to print the table of contents. Texinfo provides both a “summary” contents page, and a full table of contents page.

```
@summarycontents
@contents
@bye
```

The **@bye** ends processing. You can put notes to yourself in the file after the **@bye**; I use it in The GAWK Manual to keep my “to do” list.

Indices

Texinfo maintains six pre-defined indices; the concept, variable, function, keystroke, program, and data type indices. To add to the concept index, for instance:

```
@cindex History of @code{Foogol}
@cindex @code{Foogol}, History of
```

The variable index uses **@vindex**, the function index **@findex**, and so on.

You can create your own indices, and it is also possible to merge indices; for example to have variables, functions, and concepts all in the same index. These are typically printed at the end of a document, right before the table of contents, with the **@printindex** command.

In-Line Font Changes

We have already seen **@code**, that sets text in a constant-width or Courier-like font. There are several other commands that provide special output for different items.

```
@samp   - for in-line code fragments
@file   - for file names
@var    - things that can vary, parameters, values, and so on
@emph   - emphasizes something, using italics
@strong - makes a strong point, using bold
@r      - force roman (normal) text
@i      - force italic text
@b      - force bold text
```

The latter three are for occasional use, usually the other commands suffice.

Tables And Lists

Texinfo allows you to easily write lists and simple tables.

```
@enumerate A @c A list "numbered" A, B, C ...
@item
First item here
@item
Second item here
....
@end enumerate
```

Leaving off the “A” would have generated a numbered list, and using a small “a” would have used lower-case letters. A simple list with a minus or bullet would be written:

```
@itemize @bullet
@item
First item here
@item
Second item here
....
@end itemize
```

This generates a list with bullets for each item. Using **@minus** would generate a minus sign. (Minus signs and dashes are different characters when typesetting; minuses are longer.)

There are several kind of tables. Each one takes a formatting code that describes how to format the individual items. For instance, a list of command line options would use @samp.

```
The list of @code{foogol} command line arguments are:
@table @samp
@item -f @var{filename}
Use @var{filename} as the source file.
@item -o @var{prog}
Use @var{prog} as the executable program.
@item -d
Print the date, time of day, and current phase of the moon.
...
@end table
```

Two other similar commands are **@vtable … @end vtable**, and **@ftable … @end ftable**. These are just like **@table**, but they automatically make an entry for each item in the variable and function indices, respectively, which can save you some typing.

Examples

There are several commands for formatting text that is marked off in a special fashion, such as examples, quotations, and so on. The primary one is **@example … @end** example which is generally used for program or other computer input and output. The text will be printed indented, in a Courier-like font, and it will not be otherwise filled or adjusted.

The **@format ... @end** format is like **@example**, but does not do the indenting. You can quote extended pieces of text by enclosing the quote in **@quotation ... @end quotation**.

After an example, it is typical to continue a thought without starting a new paragraph. To keep the continuation text from being indented like a new paragraph, precede it with **@noindent**.

This example shows the results of running our program:

```
@example
$ make_money
Congratulations! You are a rich   Memory fault - core dumped
$
@end example
@noindent
As we can see, there is a small problem somewhere.
...
```

Cross-References

A distinguishing feature of Texinfo documents is that they are liberally filled with cross-references. There are several kinds of cross-reference statements, all of which take at least a node name as their argument.

```
@xref{Introduction}, for a description of the
history of @code{Foogol}.
```

There are longer forms that can take the full title of a node (from the **@chapter** command, for example). In TeX, the cross-reference will generate a typical cross-reference, including a page number.

```
See [Introduction], page 12, for a description of the history of Foogol.
```

In an Info file, the cross-reference commands put cross-references into the text. The cross reference would look like this:

```
*Note Introduction, for a description of the
history of `Foogol'.
```

When reading the Info file with a viewer, you can issue a simple command that will instruct the viewer to follow the cross-reference. It is the cross-references that provide the “hypertext” feel to the on-line Info reader. Cross-references are typically references to other nodes within the current document, but they don't have to be! A cross-reference to a node in another Texinfo document can be followed, and the Info reader will go to it just as easily as to a node in the current document.

Info Viewers

There are two ways to view Info files. (Remember, before you can view a texinfo file, you must create it from the **.texi** file using either **makeinfo** or **texinfo-format-buffer**.) The first is the stand-alone info viewer. This is a rather slick program, with menu completion features, the ability to split the screen to view multiple nodes, and much much more.

It requires a terminal or terminal emulator with cursor motion facilities. If you can run vi or Emacs in your window, or on your screen, then you can run info. The second is with the Info major mode in GNU Emacs. The Emacs Info viewer has almost all the features of the stand-alone info viewer, except the ability to split the screen and view multiple nodes.

As mentioned, **info** can also follow an infinite chain of cross references, allowing you to browse documentation to your heart's content.

Miscellaneous Commands

There are several miscellaneous features that should be mentioned. First, you can write footnotes, using **@footnote{...}**. This creates real footnotes with TeX and a collection of footnotes at the end of the node with Info. If you absolutely must use fancy TeX features, then you can drop into pure TeX mode by bracketing your text with **@tex** and **@end tex**. In between these two statements, you must write in pure TeX, with backslash as the escape character, and so on.

Note that **@tex** and **@end tex** are different from the **@iftex ... @end iftex** commands mentioned earlier, which conditionally include text into the printed document.

Finally, there is a simple macro facility in Texinfo. Macros can be set with **@set**.

```
@set EDITION 1.23
```

The value can be retrieved with **@value**

```
This is Edition @value{EDITION} of ...
```

Macros can also be used as flags, for example to indicate if a version is a draft or not. The flags can be tested with **@ifset ... @end ifset** and **@ifclear ... @end ifclear**, which test if the macro is set or not, respectively.

```
@set DRAFT
@ifset DRAFT
@b{This is just a draft, please mark it up and send it back.}
@end ifset
@ifclear DRAFT
```

```
Welcome to Edition @value{EDITION} of ....
@end ifclear
```

The macro facility is particularly useful for the multiple cases near the front of a document where the title, edition number, and program version number are repeated. By using **@value** in all those places, you can use **@set** for the title, edition, and version, and only have to update the numbers once. This is a true time saver.

Getting Printed Documentation

To generate printed documentation, you need to have TeX installed on your system. The **texinfo.tex** macros need to be in TeX's macro directory. You will also need to install a program named **texindex**, that comes with the Texinfo distribution, into a public **bin** directory, where everyone can get to it.

Texinfo files usually have a **.texi** extension. When TeX runs, it generates unsorted indices. The file name for the unsorted indices match the Texinfo file's name, with the **.texi** replaced with the two-letter name of the index. For example, the concept index for **foogol.texi** would be **foogol.cp**. The unsorted indices are not terribly useful. The texindex program sorts the indices into new files, whose names are the same as the index files, with an s on the end, e.g., **foogol.cps**. If these files exist, **texinfo.tex** will include them when TeX is run again.

The typical sequence is to run TeX three times, running texindex in between each run.

```
$ tex foogol.texi
$ texindex foogol.??
$ tex foogol.texi
$ texindex foogol.??
$ tex foogol.texi
```

The first run generates unsorted indices, and creates **foogol.aux**, which lists the page numbers of the nodes. This information is used to fill in the cross-references. The second run generates a complete DVI file, but unfortunately, the cross-references in it could be off by a page or so. The third run gets everything right, and at that point you can arrange to print the file. Typically, this is done with **dvips**, to generate PostScript.

```
$ dvips foogol.dvi -o foogol.ps
$ lpr foogol.ps       # or however you print
```

The Texinfo distribution comes with a shell script named **texi2dvi** that will figure out how many times TeX should be run. If it has been installed, it is probably the easiest thing to use.

If you don't have TeX, another option is to use a separate package called **texi2roff**. This reads Texinfo files and generates Troff input that can be processed with GNU Troff (**groff**). You have your choice of Troff macro packages, **-me**, **-ms**, and **-mm**.

TeX is preferred to **texi2roff**, but at least the latter is an option. The generated Troff may need some massaging by hand, but should otherwise be fairly usable. Unfortunately, the most recent version of **texi2roff** is considered obsolete, and is no longer being distributed by the FSF.

Also of note is a package called LaTexinfo, which is a set of LaTeX macros and a modified version of makeinfo for doing the same thing as **texinfo.tex** and the regular **makeinfo**. Texinfo files are not compatible with LaTexinfo, unfortunately. Use archie to find a recent copy of LaTexinfo if you prefer to use LaTeX.

Summary

Texinfo provides a clean input language with everything necessary for producing handsome printed documentation and highly usable on-line hypertext help. The info viewer provides a friendly interface for reading the on-line Info files.

The nicest thing I have found about Texinfo is that you don't need to know TeX to use it. I have been happily writing in Texinfo for around seven years, and have not really needed to learn TeX. Even though Texinfo has over 160 commands, what I've covered in this article is 95% of what most people would use on a day-to-day basis.

I also recommend buying and reading the Texinfo manual from the FSF. It is well-written and thorough. You will need to do this anyway if you plan to write a large Texinfo file, as this article has just scratched the surface. The Texinfo manual comes with the Texinfo distribution, and is of course written in Texinfo; this provides a nice example that uses all of Texinfo's features.

Acknowledgements

Thanks to Miriam Robbins for making me clarify a number of points in this article, and to Robert J. Chassell of the FSF (primary author of the Texinfo manual) for his comments.

Arnold Robbins is a professional programmer and semi-professional author. He has been doing volunteer work for the GNU project since 1987 and working with Unix and Unix-like systems since 1981.

Overview Of The Debian GNU/Linux System

Ian Murdock

Issue #6, October 1994

In previous columns Ian introduced the Debian system, explained the circumstances that led to its creation and detailed the motivations that keep the project alive. This month's column will tell the reader how and where to get Debian and what it has to offer.

Debian differs from other Linux distributions in many ways, a few of which are radical departures from the ways distributions of the past were assembled. These differences have attracted developers from around the world to work together toward the common goal of making Debian the best Linux distribution available. Indeed, one of the differences that has attracted them is the fact that they *can* work together.

Open Development

The most unique aspect of Debian development as compared to other Linux distributions is the fact that it has been and continues to be developed openly by a group of volunteers, and that it is open to other volunteers who wish to join the effort. Debian is not developed by one individual or a small, closed group. Instead, it follows in the tradition of the Linux kernel; it is developed by those who use it, and this makes for a higher quality, more dynamic, and truly modular system.

Does this sound to you like an invitation to chaos? Originally, many people claimed that the open development of the Linux kernel was an invitation to chaos and disaster, yet Linux is not a disaster. Neither is Debian, for a good reason.

Modularity Of Debian Components

As the Debian developers create their pieces, they follow strict guidelines for constructing and maintaining these pieces, called packages. Because these

guidelines are followed, each package can be dropped into the system independently without damaging or interfering with programs from other packages. By working with a set of consistent rules and with identical tools, the volunteers can and do create a truly modular system.

Modularity is extremely important to such a large collection of software as a distribution. New releases of the software that comprise the distribution are constantly being made, and it is the task of the distribution maintainer (or maintainers, in the case of Debian) to keep this software well integrated with the rest of the system and up to date. It is very difficult for the maintainer to do this successfully with dozens of megabytes of software, especially when the software is not written specifically for the system. When one person or small group attempts to do this, maintenance of the distribution soon becomes a nightmare.

A distribution with many different people responsible for the maintenance of its packages does not suffer from this overwhelming task; different people are able to devote more attention to the packages they maintain than would otherwise be possible, and it is possible for experts in a particular area to take responsibility for the packages involving their area of expertise. The result is a better, more timely set of packages, complete with up-to-date components, full documentation and solid examples. A collection of such independent but highly cooperative packages makes a high quality, consistent, modular distribution, which is exactly what Debian is.

FSSTND Compliance

Debian was the first Linux system to adopt, support and participate in the construction of the Linux Filesystem Standard (FSSTND); since that time, Debian has been joined by Linux/PRO, MCC, Slackware, TAMU, and other major distributions. FSSTND compliance means full compatibility with the distributions that follow it, easy integration of third-party packages and easy installation of the system into a network of FSSTND-compliant Linux machines.

Ease Of Installation And Configuration

Debian was designed to be simple enough for the novice to install and configure, yet not so simple-minded as to frustrate the advanced user. The installation process is as modular as the system itself; the base system, which requires less than 7MB of disk space, can be installed in less than ten minutes. All packages are installed independently of the base system with the Debian package maintenance utility, dpkg.

Intelligent And Powerful Package Maintenance

A new package maintenance system called dpkg has been developed specifically for the Debian system. With dpkg, the administrator of a Debian system can easily install, remove, upgrade and obtain information about both installed and not-yet-installed packages.

dpkg is being written to easily and extensibly support multiple package formats, and it is planned to eventually support (at least) Slackware and System V packages.

Built-In System Upgradability

Since the beginning of the project, Debian has been designed with upgradability in mind. Every component of the system can be easily upgraded with just a few simple tools. Packages installed with dpkg can be easily upgraded. All one needs to do is install any upgrade package normally, just as if installing it for the first time, and dpkg will automatically detect that an older version of the package is installed on the system and ask for confirmation that it should be upgraded. dpkg takes care of the rest, asking whether to replace configuration files, and notifying the user in the unlikely event that any manual steps need to be taken after the upgrade has been completed.

Similarly, the base system is upgradable, albeit by a slightly different method. Periodically, the Debian Linux Association will release upgrade packages to the latest release of the base system. Usually this will involve executing a script or a similar task. The upgrade process will always be simple and will usually be fully-automated.

The key point to make about upgradability in Debian is that the user need only install the base system once. Upgrade packages to the base system will be provided when a new version of the distribution is released, and packages in dpkg format will always be upgradable. In summary, the entire system will be upgradable with an absolute minimum amount of work on the user's part.

Availability

The current release of Debian may be found at the Internet FTP site sunsite.unc.edu in /pub/Linux/ distributions/debian. It is also available at a number of public “mirror” sites around the world; for details on how to obtain a complete and current list of mirror sites, please see below.

We have several e-mail lists set up for developers and the general public to use. Bruce Perens (bruce@pixar.com) is the manager of the lists and the moderator of debian-announce.

- debian-announce@pixar.com A moderated mailing list for important Debian announcements (new releases, bug fixes, etc.)
- debian-user@pixar.com An open forum for users of Debian (questions and intelligent discussion are welcome here)
- debian-devel@pixar.com A closed mailing list for use only by Debian developers and BETA testers

The first two lists are open admission. To join them, send mail to LISTSERV@pixar.com with the following information in the body of the message:

```
subscribe debian-announce YOUR-NAME-HERE
subscribe debian-user YOUR-NAME-HERE
```

Put your name where it says **YOUR-NAME-HERE**!

If you are actively developing software for the Debian distribution, you should subscribe to **debian-devel**. Send mail to **bruce@pixar.com** with a sentence explaining what software you are developing and a request to be added to the list.

For the latest Debian announcements and information, Debian users have one of three options:

- Join the **debian-announce** mailing list. Please note that you do not necessarily have to be on the Internet to do this; many of the popular on-line services (CompuServe, for example) have Internet mail gateways. You may also join any of the other mailing lists from such an on-line service, but as many on-line services surcharge incoming and outgoing mail to the Internet you may wish to think twice before doing so. **debian-announce** is a moderated mailing list; the others are not and can be quite active.
- Download the files in **/pub/Linux/distributions/ debian/info** on **sunsite.unc.edu**. These files contain detailed information about Debian and its motivations, the Debian Linux Association, where Debian can be obtained (FTP, BBS, etc.), where Debian and related materials can be ordered and so on.
- Request the information from the Debian Linux Association. If you do not have access to the Internet or FTP, hardcopy of recent announcements mailed to debian-announce and the contents of **/pub/Linux/ distributions/ debian/info** may be obtained by sending a self-addressed, stamped envelope to:

The Debian Linux AssociationStation 11P.O. Box 3121West Lafayette, IN47906 USA

By the time this article is published, the Debian system should be available on floppy diskette and CD-ROM from the Debian Linux Association and the Free Software Foundation. Please send mail to info@debian.org or the Debian Linux Association at the address above for the latest ordering information.

What is a Distribution?

Ian Murdock (imurdock@debian.org) is the founder and leader of the Debian project. He can be reached via c/o The Debian Linux Association at the address above.

Archive Index Issue Table of Contents

Advanced search

Motif 1.2.3 Runtime and Development System for Linux

Dale A. Lutz

Issue #6, October 1994

The Motif toolkit has come to Linux. Dale reviews Sequioa International's Motif Toolkit for Linux.

It is worth pointing out up front that for the most part, Motif is Motif regardless of who you buy it from. A vendor licenses the Motif source code from the Open Software Foundation (OSF), and then ports it to whatever platform they want so they can sell the resulting libraries and executables. Therefore, the main part of the product will not vary much from vendor to vendor, or platform to platform. However, where a vendor gets a chance to differentiate themselves is with the user's manuals and the installation procedure they provide. In my view, Sequoia has done an excellent job in bringing Motif to the Linux platform. Any of the complaints I raise in this article are really complaints about Motif and not about the job Sequoia has done.

What is Motif?

Motif is really three things. First, it is a set of guidelines which define the way an X-Window user interface should look and feel. To the untrained eye, Motif looks an awful lot like Microsoft Windows or OS/2's Presentation Manager. This should come as no surprise, when the history of Motif is considered. The Motif “shell” program provided with the libraries gives this history of Motif:

On December 30, 1988, OSF announced that the user environment component offering will be based on several leading technologies: Digital Equipment Corporation's toolkit technology (widgets) and the joint Hewlett-Packard/ Microsoft submission of H-P's 3-D appearance and Microsoft's Presentation Manager-compatible behavior (window manager). The hybrid offering, OSF/ Motif environment will provide users with a familiar way to access computer resources across a broad variety of computing hardware platforms, from personal computers to mainframes.

The driving force behind Motif was to unify the graphical user interface look and feel of Unix-based software so that it could compete for the desktop against other standards like the Macintosh or Microsoft Windows. By having a standard look and feel which was common across many hardware platforms, users would benefit by not having to relearn how to interact with software as they moved around and developers would benefit by having a standard to program against.

This brings us to the second part of what Motif is. Motif is a standard set of widgets with a documented interface which may be ported from platform to platform. It is safe to say that if a platform supports X-Window development, then it will not take much effort to make it support Motif. All that's needed are the Motif libraries, or failing that, a source code license from OSF for Motif and a minor bit of effort to port them.

To many people, Motif is just a window manager for X-Windows. This third aspect of Motif is what determines the way window borders are drawn, how windows are resized, iconified, and moved. Generally speaking, it defines how users and applications will interact with their windowing system.

Why Motif?

Why would you want to run Motif on your Linux system? There are three reasons, which parallel the three aspects of Motif discussed above. First, you may want the applications you build to follow the standards for GUI look-and-feel so that your users will feel comfortable sooner and will benefit from consistency across applications. Second, you may want to develop software on your Linux box that you would like to port to other X-Windows platforms. If you use the Motif toolkit to build your code, you will be able to quickly port the user interface to any other platform that has the Motif libraries available.

It is worth pointing out that the TK toolkit which comes with many distributions can be used to create Motif-compliant applications, though programming with it is radically different than programming with the Motif widgets. I will not betray my own bias on this issue; instead I defer you to the endless discussions on this topic which are found in the Usenet comp.lang.tcl and comp.windows.x newsgroups.

The last reason you may want to buy Motif for your Linux box is to run the Motif Window Manager (mwm). You or your clients may wish to have exactly the same look and feel on your Linux box as you enjoy on other machines. While fvwm can closely emulate a Motif look, it is not the same. The mwm window manager seems rock solid and gives your Linux box an exact Motif look. Note that I am not trying to knock fvwm; in fact, I prefer it to mwm because I like its virtual desktop and its low memory use.

Requirements

What kind of system resources does the Sequoia Motif consume? The manual indicated that 12MB of free disk space were needed under /usr, and 3MB were needed in /tmp. It turned out that 3.6MB of space was needed in the /tmp directory, and only 9.9MB of space were used in the /usr directory.

A Linux kernel newer than 0.99pl13 is required, and XFree86 2.0 must be installed in the standard way (under /usr/X386). The documentation describes how to check this (and in fact the installation script checks it as well). My Slackware 1.1.1 had it in the right spot.

Sequoia also recommends 12MB of RAM to do development (i.e., compiling and linking Motif programs). I have only 8MB and I was able to do it with no more swapping than I normally get when I do compiling. It is usable with 8MB, but certainly it's better with 12MB.

Installation

Sequoia has done a great job with their installation instructions. They make it very clear what to do. Motif comes on three 3.5" diskettes. You first install copy these into /tmp using cpio and then you run an installation script. One small complaint I have is that cpio tells you to load a ”tape“, which could confuse some people, but that is cpio's problem and certainly not Sequoia's or Motif's. The installation script verifies that you have X386 installed and won't continue without it. This is nice.

I was done installing in about 10 minutes. The script made all the necessary links and put all the executables and libraries into the right spots. I literally had nothing more to set up or do to run or develop motif programs.

Firing up MWM

I had a ”start MWM“ option on my “fvwm” root menu, so I tried it. Bingo; just like that I was running mwm! To make mwm my default window manager I just had to edit my .xinitrc and replace the last fvwm in the last line with mwm.

The documentation did not describe this, though anyone who is advanced enough to want to use Motif would likely be able to figure it out. This is a minor instance where the product could be improved.

The main area that more instructions are needed is in the customization of the window manager. I had a whale of a time configuring mwm. I thought that all I'd need to do is copy the sample mwmrc file they provide into ~/.mwmrc, make

a few changes and I'd be off. Well, an hour later I finally figured it out (after signing onto a friend's machine to check out his .mwmrc).

It turns out that for your own window manager menus to be activated, you must have a set of “Button Bindings” named “DefaultButtonBindings” to tie the buttons to the menus. Now the example mwmrc file does not have a set button bindings named “DefaultButtonBindings” anywhere. Instead, it has “MyExplicitButtonBindings”. So no matter what I did I wasn't picking up my own menu buttons. Finally I read through the manual enough to finally understand, changed the “ExplicitButtonBindings” to “DefaultButtonBindings” and I was in business. This is not a fault of the Linux product, but more generally with the Motif documentation. Most often, users just copy an .mwmrc from someone else they know. However, for Linux people who are often working in isolated environments, it might be a nice touch to supply a more ready-to-go sample .mwmrc or better documentation on setting one up.

The other common thing you may want to change is the keyboard focus policy. This determines how you choose the window into which you will type. I prefer “focus follows mouse”, which means that all I need to do is put the pointer into a window and I can start typing. The default Motif behaviour is to have “click to focus” where a window keeps focus until you click in another window. You can also set whether or not you want the window with focus to be raised automatically to the top of the stack. The two X resources you alter in your .Xdefaults file are:

```
Mwm*keyboardFocusPolicy: pointer
Mwm*focusAutoRaise: false
```

This is the type of behaviour I prefer, which is the focus follows the mouse and it does not automatically raise the window with focus. If you prefer the Microsoft windows behaviour, choose the opposite settings (“explicit” for keyboardFocusPolicy and “true” for focusAutoRaise) in your .Xdefaults file, type:

```
xrdb < .Xdefaults
```

to update the settings, and finally restart mwm to apply the changes.

What is Included?

The Sequoia Motif comes with UIL (the user interface language of Motif), static and shared Motif libraries, the full set of on-line Motif manual pages, a number of demo programs with source code, and the OSF/Motif user's guide.

If you want to distribute Motif programs, you need to link statically by changing the definitions of the Motif libraries in a makefile to include the static (.a) libraries. You are not allowed to distribute the shared Motif libraries. This

makes executables a lot bigger; for a program I wrote, the executable was 130KB when the shared libraries were used and 1.2MB when linked statically. If you are going to be running a lot of Motif programs, it is better to have the shared libraries around and have executables not linked statically. It is possible to distribute executables linked with the shared Motif libraries, but then only people with copies of the shared Motif libraries will be able to run your binary.

The demo programs all make without a hitch on my 8MB system. I believe that development can be done with 8MB, but of course if you have more memory, things will work much faster, since there will be much less swapping.

Performance of mwm

I found mwm to perform very well on my system.

I did check its memory usage, and found that it used 1460KB of memory (according to TOP), while fvwm used only 296KB. Clearly, if you are trying to conserve memory, fvwm is the better way to go. As mentioned earlier, there is no virtual mwm, so if you like having a larger than screen desktop, you'll want to stick with fvwm. However, in terms of a solid consistent sharp look-and-feel, mwm has a slight edge.

fvwm in top:

```
  PID USER PRI NI SIZE RES SHRD STAT %CPU %MEM
  TIME COMMAND
  67 dal   2  0  160  296 296  S  0.8  4.1
  0:25 fvwm
```

mwm in top:

```
  980 dal  3  0 344  1460 608  S  1.7 20.3
  0:01 mwm
```

Conclusion

I was impressed by Sequoia's port of Motif for the Linux platform. They have done a fine job bringing the industry standard to Linux. If portability is a concern for you or you want to use the same tools as the rest of the industry, then I recommend this product.

[Ed: Since this review was written, GUI, Inc. has taken over responsibility for the SWiM Motif distribution, and has released a new version of SWiM based on Motif 1.2.4. GUI is responsible for support, including upgrades, for SWiM. More details will be available in the review of GUI's SWiM Motif 1.2.4 which will be published in a future issue of *Linux Journal*.

Dale A. Lutz can be reached at (dale@wimsey.com)

LINUX JOURNAL

Unix Interactive Tools

Clarence Smith,, Jr.

Issue #6, October 1994

Being a new administrator for your Linux system can be very trying at times. When maintaining your system, you want to be able to manage it in an organized fashion. Unix Interactive Tools makes the task of running a smooth system easy.

Unix Interactive Tools enables users to view and handle their file-system with ease. The software package itself is made up of three main components: a file browser, process viewer/killer, and a Hex/ASCII viewer. With these tools, new system administrators can complete the necessary duties of overseeing their new kingdom more quickly and efficiently.

The file system browser is made up of two side-by-side panels. Each displays the current working directory of the user who initiated the UIT session. Inside the panels, along with the directory listing, is info about the files themselves (permissions, size, etc). The windows can be easily navigated using the cursor keys and the page-up and page-down keys. These keys move a highlighted bar which shows the current file/directory selected. The ENTER key is used as well, to move in/out of the highlighted directories. Lastly, the file browser contains a shell-like command line, for typing in normal shell commands, and also has a status bar which contains optional “hot-keys”. These keys contain such commands as move, copy, and mkdir.

The UIT package also includes a process viewer/killer. Like the file browser, this process utility can be navigated by the user via the cursor keys. Similar to the file browser, the viewer has a status line with certain hot-keys that can be used. The process viewer/killer lists all the current processes the user is running, or if you are running UIT as root, it shows all the processes that are running. Other things listed are the PID (Process ID) number and the TTY (terminal used by the process). The viewer's ability to examine/manipulate a process quickly and efficiently serves a functional purpose in the lives of new (and experienced) system administrators. This tool wouldn't be used for a “quick glance” of the

processes running, such that **ps -aux** would provide. It is more of a detailed outlook at the processes.

The last component of the UIT package is a Hex/ASCII file viewer. It can be used to view and edit executables or data files. This is a great tool for those who have done assembly in DOS, for they can recognize the structure that the Hex/ASCII viewer resembles.

I highly recommend the UIT package for new administrators because of its usefulness in providing a global view of any particular filesystem. This package enables users to maintain their system efficiently, without having to worry about which shell command to use, how to pipe to an output file, etc. It's all in there, and the UIT package makes it simple. I don't recommend using the UIT package without first learning some of the basic Unix commands, though, because there are times when you may want to use that nifty shell command line that comes in the file browser!

Unix Interactive Tools was written by Tudor Hulubei and Andrei Pitis. The current version for UIT is 4.3a. The complete source to the UIT package can be found at ulise.cs.pub.ro, in the /pub/public/linux/uit directory. Check it out, and tell me what you think!

Clarence Smith, Jr., is a pre-communications student at the University of Washington, studying Public Relations and Sociology. He is a Linux Hobbyist, and hopes one day to become a developer of software packages for Linux.

Archive Index Issue Table of Contents

Advanced search

Crisp Text Editor

Robert Broughton

Issue #6, October 1994

How can a commercial editor compete with all the free products that the Linux Community has access to? In this article, Robert Broughton helps us decide whether to spend our money on this Brief-turned-shareware-turned-commercial editor.

First, a history lesson. Crisp was originally a shareware product, written by Paul Fox, of London, UK. It is a clone of Brief, an MS-DOS product marketed by Borland, and was originally released in 1988. The last shareware release, 2.2e, was in 1991. Fox, who is also known to the Linux community as the initiator of the NOTIF project, was disappointed with the low shareware income. He was also aware that he had an excellent product on his hands. So, version 3 of Crisp was released in 1992 as a commercial product; the initial Linux version is 4.1.9. It is available on a number of platforms, including Sun, AIX, SCO, HP, Univel, Windows 3.1, and Windows NT. Both character and X-Windows versions are available for Linux.

The original designer of Brief paid a great deal of attention to ergonomics. Crisp is easy to learn: the only thing you really need to know to get started is that <alt>x saves files and exits. Once you've got that down, <alt>l is used to block lines, and <alt>c is used to block columns. The + key on the numeric keypad is used to copy a block, and the - key on the numeric keypad is used to cut a block. If no block is defined, the current line is copied or cut. Blocks are then inserted using the <insert> key. The <home> key takes the cursor to the beginning of the current line, and the <end> key takes the cursor to the end of the current line. <PageUp>, <PageDown>, and the “arrow” keys perform their obvious functions. <alt>h gives you help.

To list all of the features of Crisp would make this a very long article. It has just about all of the capabilities of Emacs, but it is, in my opinion, much more user-friendly than Emacs. It's easier to customize, and the documentation and on-line help are much easier to deal with. It is **much** easier to install. Both Crisp

and Emacs have macro languages, but Crisp's is C-like, as opposed to the lisp language for Emacs. From my personal point of view, Crisp has one key feature that Emacs lacks: when you exit Crisp, it remembers all of the files that you were editing, the position of the cursor in each file, and, for the X-Windows version, the size of the window. Not only that, but it keeps a separate “record” for each working directory that you edit files in. You can edit one set of files in directory /a, save and exit, cd to directory /b, edit another set of files, save and exit, and cd back to /a, and edit the first set of files without losing your place. Even search strings are carried over.

Version 2.2e (the final shareware version) has been ported to Linux. It can be found on **sunsite.unc.edu**, in **/pub/Linux/apps/editors**.

The Character Version

At first glance, the look and feel of the current version of Crisp is not very different from the old shareware version. The first thing you'll notice that's different is that if you're editing a C program, the comments will be in a different color from the rest of the text. The big difference comes when you press <F5> to do a search: you can recall previous search strings by pressing the <up-arrow> key. Also, searches can “wrap” as they do in vi. The “context memory” works better: the shareware version remembered only the last file edited, but the commercial version remembers all files. The commercial version has optional pull-down menus.

There are other improvements that are less obvious, but very important. The on-line help has a professional look, rather than a shareware look, and the product is generally tighter.

The X-Windows Version

The shareware version of Crisp includes an X-Windows version, but Paul Fox describes it as “How I Learned to do X-Windows Programming”. It works OK, but it's very rudimentary: no menus or icons, and the mouse support is incomplete.

The Linux commercial version uses XView. Crisp has been implemented with Motif on some other platforms. This was not done for Linux for practical business reasons: the number of Linux users that have purchased Motif libraries is small, and of these, the number who would shell out money to purchase Crisp would be even smaller.

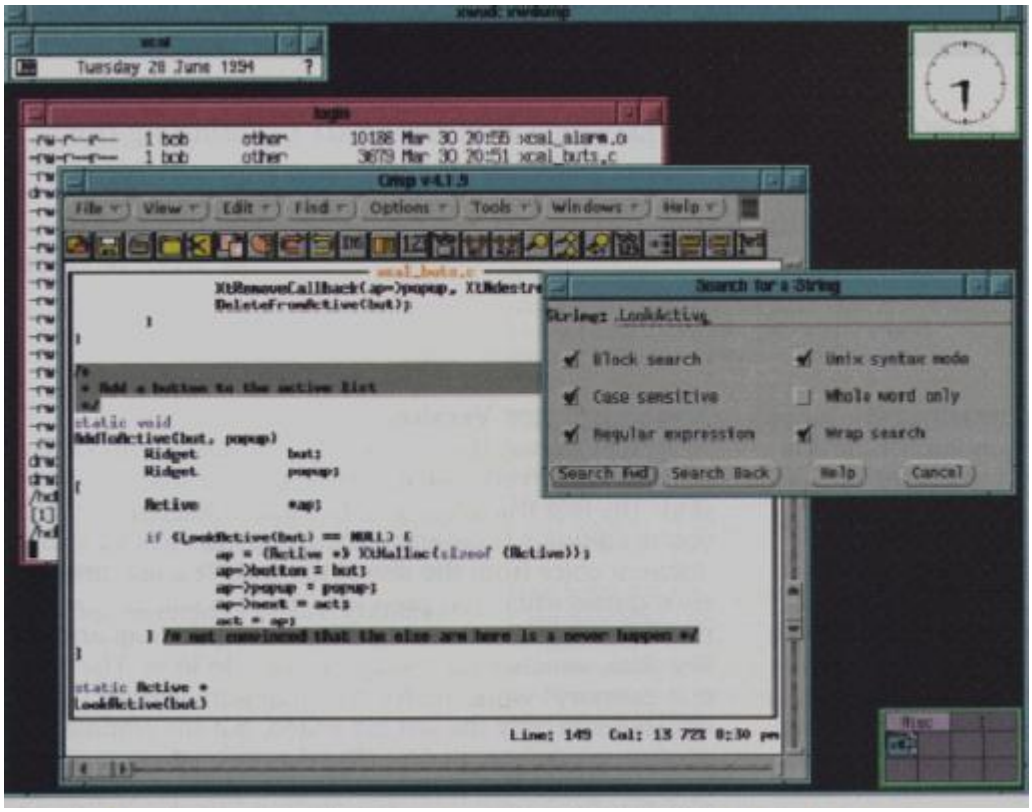
I regard this as a minor drawback. I have an SCO system at work, and use the Motif version of Crisp on it eight or nine hours a day. I prefer the Motif look-and-feel over XView, although people with Sun backgrounds may feel differently.

Anyway, I said “minor” drawback. The XView version certainly works, and Crisp has been designed to take maximum advantage of this environment. The mouse can be used to move the cursor, and cut and paste text. Above the text window is a row of color icons that can be clicked to activate the most commonly used functions: search/replace, switching files, recording and playing macros, etc. At the top of the Crisp window is a set of pull-down menus that can be used to do anything (as far as I know) that Crisp is capable of doing, including make, grep, and reading e-mail. There are vertical and horizontal scroll-bars for text editing and on-line help windows. There's an option for a separate “toolkit” window with a more extensive set of mouse-activated commands. It is possible to peel off additional editing windows.

There are also dialog boxes for functions like opening files and search/replace. The dialog boxes, fortunately, are configurable. I suspect that most users will want the dialog boxes turned on when they are learning their way around Crisp, and they will gradually turn them off. One reason is that you can get more work done without the dialog boxes. Another reason is that if you have a window manager that always makes the window under the cursor the active window (I use fvwm), you may type something like a file name into a dialog box, and discover that you have accidentally edited your text instead, because the mouse was pointing in the wrong place.

The major “ooh, ahh” feature for the X-Windows version of Crisp is colorization. As with the character version, comments are shaded. Language keywords, such as if, case, and int, appear in blue. Strings appear in red. These color choices are customizable, and there are, of course, templates for languages other than C, including Ada, COBOL, and SQL.

Crisp has vi emulation. With the X-Windows version, the vi commands and the GUI features are interchangeable.



Some Additional Features

As I said, discussing all of the features of Crisp would make this a very long article. I will mention a few that I don't have any personal experience with, but the developers are very proud of.

- PostScript program listings: If you have a PostScript printer, or ghostscript, Crisp will attempt to simulate the language colorization by using shading, boldface, etc. If you have a color PostScript printer, you'll get a really fancy listing.
- Filter mechanism: Clicking an icon will execute a predefined command, such as a compiler, and the contents of the current file become the input to that command. Crisp opens another window, and the output of the command appears in that window.
- Asynchronous mechanism: External programs can write commands that Crisp can understand, such as “edit__file” or “goto_line” to a file named cr.async. The external program can then send signal -16 or -17 to the Crisp process, and Crisp will execute the commands. (If your external program cannot send signals, it's possible to configure Crisp to look for the cr.async file every few seconds.)

The Bottom Line

Crisp is sold with a variety of license arrangements. The one that will be of interest to most Linux users will be the single-user, single-execution license. The price for this will probably be $99US for either the character or

X-Windows version, and the manual (useful, but not essential) will be another $40. The international agent

for Crisp is Vital, and they can be reached by e-mail at owner-crisp-list@uunet.uu.net.

Robert Broughton (a1040@mindlink.bc.ca) has been developing software for 22 years, and has been using Linux since February, 1993. He is employed by Zadall Systems Group, in Burnaby, BC, Canada.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #6, October 1994

Readers sound off.

Yet another fan…

I just discovered your *Linux Journal*. I purchased and enjoyed issue #3, and I just wanted to write to say thank you for this terrific publication. It's terrific to have a legitimate source for Linux info, and to find so many resources for Linux.

Thanks again; I really like *Linux Journal*.

—Peter Horadan peteho@halcyon.com

A Port in a Storm

I've read an issue of *Linux Journal* and I think it's great. I will probably be sending in a subscription soon.

In the issue I read (Vol. 1, Ed. 2), there was a request for questions. One thing I would like to see is a definitive review of multi-port serial boards. I'm sure that there are a lot of people like me who want to find a good board that is supported by Linux (without having to make several patches to the kernel).

If you plan to do such an article or have already done one, please let me know.

—Jason Funk jasonfunk@aol.com

LJ replies:

We had one article about the newly-supported Cyclades board in issue #5. It is currently the only smart serial board supported by the kernel. Digiboard will probably be supported later. All dumb serial boards that use a UART with a

FIFO like a 16450 or 16550 (almost all of them) are pretty much equivalent, and most are supported in the standard kernel.

Virtual Reality??

I just read your post to c.o.l.announce; part of which was a preview of the contents of the September issue. I'm particularly interested in the article "Writing an Intelligent Serial Driver", by Randolph Bentson because that is what I am doing! Is this an introductory article or does it document a real project to implement intelligent serial I/O on Linux?

BTW, I subscribe to *LJ* and look forward to reading it each month. I was particularly impressed with the recent article about the history of Unix. I'd also like to see more "real world" examples of using Linux like the comic example this month.

—Simon Allen simonallen@cix.compulink.co.uk

LJ replies:

The article (which was printed in *Linux Journal* issue #5) does indeed document a real project: implementing a driver for the Cyclades 8Ys intelligent serial board.

We have a policy of publishing articles about Linux being used in the real world. However, they are sometimes hard to find out about. Therefore, if any readers know of Linux being used in the "real world" in an interesting way, please send us a letter at ljeditor@ sunsite.unc.edu and tell us about it.

Shopping with *Linux Journal*

I saw where you requested feedback on page 19 of the June issue. I subscribe and am shopping for a Linux machine, which I'll probably buy preloaded from an ad in *LJ*. I was already a customer of SSC, which helped me make the journey from DOS/Mac to Unix. I'm delighted that SSC got into Linux; it's a perfect match image-wise. The only thing better would be you guys cooperating with O'Reilly or something. There's a nice feel to the Journal and a nice mix of hard stuff (over my head but challenging) and beginner stuff.

—SEAKAYAKER@delphi.com

LJ replies:

Thanks.

We do have a good working relationship with O'Reilly. We have subscribers there, and we read and recommend their books. O'Reilly is also helping to work on Linux documentation, working with the Linux Documentation Project. We mention their new books in the New Products section when they send us press releases which relate to Linux; see the New Products section in this issue.

All letters to the editor are subject to editing for clarity, length, grammar, and spelling. Send letters to info@linuxjournal.com and use a subject like LTE. Alternately, send paper mail to *Linux Journal*, P.O. Box 85867, Seattle, WA 98145-1867. E-mail is preferred, because we can print the results of a discussion, which can be more useful than just your letter and our response.

Archive Index Issue Table of Contents

Advanced search

Stop the Presses

LJ Staff

Issue #6, October 1994

Announcement of version 0.9.2 of Linux/68k and Change of Maintainer of LSM.

Announcement of Linux/MIPS—Linux on Mips R4x00-based computers

The port of the Linux Operating System to Mips R4x00 based computers has been announced on the Internet. A complete Linux/MIPS distribution is expected in 1995.

Announcement of version 0.9.2 of Linux/68k

This message announces the availability of version 0.9.2 of Linux/68k. It can be ftped from directory /pub/linux/680x0 at tsx-11.mit.edu. A precompiled kernel executable can be found in vmlinux-0.09pl2.gz in the “kernel” subdirectory.

The corresponding “bootstrap” program for the Amiga is available in file amiboot-1.8.gz in the “kernel” subdirectory. If available, the “bootstrap” program for the Atari will be found in file “ataboot-0.2.gz” in the “kernel” subdirectory (the author of this ANNOUNCE-* file cannot compile such a “bootstrap” program and thus is waiting for one from someone else).

The source patch for the this version of the kernel can be found in linux-0.9.pl2.diff.gz in the “src” subdirectory.

Change of Maintainer of LSM

Lars Wirzenius posted in comp.os.linux.announce: ..I'm taking over maintenance of the Linux Software Map, or LSM for short. I'll start by making a number of changes; see the new LSM template below. In addition to changes to the template, you might want to prepare yourselves for some initial confusion on my part while I make myself comfortable with the new job. :-)

Hopefully everything will go well. Don't wish me luck, send me constructive criticism.

I am uploading to sunsite a database in the new form and a tool for browsing it. The tool, called lsmtool, is not all that well written, and is somewhat prone to crash. You have been warned. It also requires Publib, a library of C functions I've written, which you will have to get separately.

...

I would be grateful if anyone who keeps the LSM on WWW or makes it otherwise available would contact me. There might be something that I can do to make your life simpler, and at the very least, I can list URL's in the README.

—Lars Wirzenius Lars.Wirzenius@cs.Helsinki.FI

Archive Index Issue Table of Contents

Advanced search

New Products

LJ Staff

Issue #6, October 1994

ESQLFLEX, INFOFLEX-4GL, ACCOUNTFLEX and more.

Infoflex announces Linux database support

Infoflex has announced Linux ports of its database products ESQLFLEX, INFOFLEX, and ACCOUNTFLEX.

- ESQLFLEX is a low-cost clone of the Informix-ESQL/C product. It allows software developers to build, modify, and query databases using standard SQL calls from a C program, and allows developers to completely replace Informix-ESQL/C without modifying the application. ESQLFLEX is available for $695.
- INFOFLEX-4GL is similar in syntax to Informix and is 100% compatible with the Informix-SE database. INFOFLEX is easy to modify through its consistent WYSIWYG approach to programming menus, screens, and reports and through its many powerful built-in features designed to reduce coding. INFOFLEX is available for $995.
- ACCOUNTFLEX includes modules for Sales, Purchasing, Inventory, Jobcost, Payroll, Accounts Receivable, Accounts Payable, and General Ledger. Each module costs $795 with source. Executable versions may be distributed for $150 per module under an OEM license.

Infoflex Incorporated can be reached at (415)697-6045 or at 840 Hinckley Road, Suite 107, Burlingame, CA 94010.

Quadralay announces Linux support for UDT for C/C++ v1.3

Quadralay Corporation has announced the release of the Linux version of its development environment UDT for C/C++, a software tool that greatly simplifies the design, implementation, maintenance, and management of C and C++ source code and documentation.

“We believe that Linux will bring the Unix operating system into the mainstream,” said Jeffrey Stockett, Quadralay's Chief Technical Officer. “For the first time, an individual can run a complete Unix implementation in his or her own home... ...Linux is fast, reliable, and best of all, free. Therefore, we are offering UDT for C/C++ on Linux at a special price, one which will make it available to anyone who wants it.”

UDT for C/C++ consists of five components:

1. A drag-and-drop tool manager that allows developers to integrate existing tools into the UDT environment easily
2. A powerful C++ class browser with a rapid prototyping capability
3. A fully-integrated object-oriented editor
4. A complete C source code browser
5. A hypermedia-based source code librarian and authoring system (for $50 additional)

UDT for C/C++ is available for $99 without the hypermedia GWHIS Viewer, and $149 with GWHIS Viewer.

For more information, contact Brian Combs, Director of Technical Marketing, 8920 Business Park Drive, Austin, Texas 78759, (515)346-9199, combs@quadralay.com.

Interactive Software Engineering announces ISE EIFFEL 3 for Linux

ISE Eiffel 3 is a powerful and user-friendly object- oriented programming environment designed for large, complex systems. ISE Eiffel 3 is an integrated GUI workbench consisting of a variety of Eiffel-based components: the EiffelBench melting-ice (fast incremental compilation and portable C code generation) workbench, the EiffelBuild interface builder and application generator, the EiffelVision graphics and GUI library, and the EiffelBase basic libraries. The EiffelCase analysis and design workbench will be available by the time this is published. The whole suite, with documentation, is available on 1/4" cartridge tape or DAT tape for $295 +S&H.

For more information, contact Interactive Software Engineering, Inc, 270 Storke Road, Suite 7, Goleta, CA 93117, phone (805)685-1006, info@eiffel.com. Linux from Nascent CD-ROM, version 2.0

Nascent Technology has released Linux from Nascent 2.0. It is now FSSTND-compliant, and features electronic design tools for behavioral synthesis and sea-of-gates place and route, Andrew, Tcl/Tk, full source with a hierarchical source build, laptop support, and incremental package installation. Free kernel

upgrades specifically for Nascent will be made available via ftp. Linux from Nascent costs $59.95 +S&H with a 30-day guarantee, or $119.95 with six months of e-mail support.

For more information, contact Nascent Technology, P.O. Box 60669, Sunnyvale CA 94088-0669, phone (408) 737-9500, nascent@netcom.com.

FlagShip from WorkGroup Solutions

WorkGroup Solutions (WGS) has released FlagShip, their CA-Clipper compatible database development environment and xBASE porting system, for Linux. FlagShip has full xBASE binary file support, and translates Clipper code into portable C, which makes applications developed under FlagShip several times faster, and requires no royalties or run-time licensing. A demo is available for a nominal handling fee, or can be downloaded freely from ftp.wgs.com in /pub2/wgs/fsdlx.gz. For a limited time, Linux single-user licenses are $199, and unlimited-user licenses are $499.

For more information, contact WorkGroup Solutions, Inc., P.O. Box 460190, Aurora, CO 80046-0190, phone (303)699-7470, fax (303)699-2793, info@wgs.com.

Database Illustrator for Linux

Ray Ontko & Co. has released their Database Illustrator product for Linux. This program is a documentation tool for Oracle 6 and 7 databases, which utilizes information stored in your database to produce entity relationship diagrams which show the structure of your database application. Versions of Database Illustrator are available for DOS, VMS, and several versions of Unix, as well as Linux. The price for the DOS and Linux versions is $249 per license. Maintenance is $49, and an extended support plan is available for $199.

For more information, contact Ray Ontko & Co., P.O. Box 9, Richmond, Indiana 47375, phone (317)935-4283, fax (317)962-9788, dbi@ontko.com. BSD4.4 Lite documentation set released by O'Reilly and USENIX

The BSD documentation set has always been a valuable reference for Unix programmers and administrators. The new 4.4 release will be of even more use, both because the entire set has been updated and because documentation for many free programs not orignally part of the BSD distribution, including many of the GNU utilities, has been included. For this reason, and also because Linux distributions include some BSD utilities, these manuals may be useful to Linux programmers and administrators. The full set of five manuals is available for $120 (ISBN: 1-56592-077-5). In addition, the

entire 4.4BSD-Lite distribution is available on CD for $40, and the complete manual set with the CD for $150.

For more information, contact O'Reilly & Associates, 103 Morris Street, Suite A, Sebastopol, CA 95472, phone (800)998-9938, (707)829-0515, fax (707)826-0104, order@ora.com.

Archive Index Issue Table of Contents

Advanced search