Linux Journal Issue #29/September 1996

Features

News and Articles

LJ Interviews Linus Torvalds

Phil Hughes

Gena Shurtleff

Issue #29, September 1996

With a new major release of Linux it was time to see what the father of the OS had to say.

Phil:

Why is 2.0 so wonderful?

Linus:

If you mean “why do we call it 2.0 instead of 1.4”, then the answer is that there are a lot of major conceptual jumps in it, notably the multi-architecture support and the SMP [symmetrical multi-processor] support. Even with nothing else new, those two features certainly are enough to warrant a major number bump.

Note that this doesn't mean that the multi-architecture and SMP work is finished—the only really supported platforms in 2.0 will be alpha and the x86, and the SMP stuff will need to be worked on too in order to scale better. The important thing about 2.0 is that the support is there, even though we'll obviously continue working on it (the same way 1.0 had networking support, even though we obviously had to work on that too after releasing 1.0).

Now, if the “why is 2.0 so wonderful” question is more “why should I upgrade”, i.e., you have a machine that is already supported by 1.2, then the answer is that no, you don't have to upgrade, but there are lots of things that makes an upgrade to 2.0 a good idea.

For example, performance is much better, especially in the networking and NFS client department. If you have your home directory NFS-mounted, you

definitely want to upgrade. There are other areas too where 2.0 is noticeably faster, like process handling and filesystem throughput.

Also, 2.0 finally supports read-write shared memory mappings of files, along with file descriptor passing, using Unix domain sockets. Those are the two major things that 1.2 doesn't do and that most other modern Unices support.

Phil:

What comes next?

Linus:

The SPARC and the PPC port are very close to being integrated into the normal kernel—most of it is already there, just a few things missing that haven't been applied due to the code-freeze. And MIPS support will also probably go in early in the 2.1 development kernels. So, during 2.1 we'll get a lot of new interesting architectures supported—essentially 2.0 lays the ground work for multi-architecture, and so it becomes a lot easier to do ports now.

Also, SMP needs to do finer-granularity locking for good performance on more CPU's, and while SMP currently works on the x86 and Sparc, we'll probably work on making the other architectures SMP-aware too.

There are obviously other things too: the normal kernel update stuff, more drivers, and more performance improvements. IPv6, DECnet etc.

Phil:

Anything about yourself that you want to share? Lots of people ask us to cover more about who the developers are. They can read the code or the documentation but not everyone can meet the developers.

Linus:

Ehh, umm, aahh... Nothing much has happened, I finally got my BSc but I'm still working on my masters degree. It's slow, as Linux obviously takes a lot of my time. Hopefully I'll have it done by the end of the year, I mainly just have the thesis to write. It's not as if I didn't have anything to write about, it's just hard to get started with so much else going on.

Tove [Linus' “significant other”] and the cats are doing fine...

Phil:

When you get your masters degree do you plan on “getting a real job” or do you intend to stay the highest-profile unemployed person in the computer field?

Linus:

Hey, I resent that remark. I'm not unemployed, I'm just selective about what I do...

Actually, I've been employed by the University of Helsinki for the last few years, and that has been paying my bills. While I'm still studying, I also work in a research and teaching position. And the CS department is flexible enough that I can do Linux on work hours, and they encourage this by trying to keep my other work at a minimum.

Obviously a “real job” pays better than most universities will pay, but I've been very happy with this arrangement—I get to do whatever I want, and I have no commercial pressures whatsoever doing this. Getting a masters won't change things radically, although it will obviously make it easier for me to accept other work as I don't need to worry about wanting to graduate some day.

As I've been able to live happily on university pay, the deciding factor is not so much the money as the interest level of any “real job” (but lest somebody gets the idea that money doesn't matter at all, I'll just mention that yes, it does).

Phil:

How much of the code in the kernel is still yours?

Linus:

Umm.... Very little when it comes to the number of lines. What's still “mine” is the mm/*.c, kernel/*.c, fs/*.c (only the BVFS code, not the specific filesystem stuff) and parts of the x86 and alpha-specific low-level architecture files.

Even those parts have much of the code contributed by others, but the basic stuff is still pretty closely under my control. It's essentially all of the really basic stuff—things that everything else depends on.

There are lots of things I haven't really even touched: most of the device drivers are totally written by others, and while they sometimes are based on stuff I have written they really aren't mine any more. Same goes for a lot of the filesystem code.

The networking has been completely written by others, although I've touched some of it.

In lines of code, I probably am responsible for about 10% these days. That's just a rough guess, I haven't really taken a look.

Phil:

How (or maybe why) does project management work? That is, Linux is a huge effort and it continues to progress very well. How is this cooperation possible?

Linus:

Most of it happens automatically—people who are doing things for fun do things the right way by themselves. That said, I do work 8 hours a day (and that's just about minimum) on Linux, and most of the time goes to administrative things, mostly email. And it's not as if I'm the only “manager”--there are others who manage their own subsystems and then send me already cleaned-up patches (notably when it comes to networking).

Phil:

Is there any expertise lacking? Is there something or someone that, if available, would make development go better?

Linus:

I think we're doing pretty well. I need longer days (and nights!), but there isn't anything specific we really need. Lots of areas needing work, and lots of developers that don't have enough time, but we can't really complain.

Phil:

We asked this before, but the answer may have changed. What, if anything, did you do wrong in Linux development. (When Ken Thompson was asked this about Unix he said he left the “e” off the “creat” system call. Is there something that you would do different?

Linus:

I'll be arrogant and say “nothing”. I think that's the same answer as last time. I've made lots of mistakes, but that's okay and normal, and the kernel is the better for it—it tends to only help make the corrected version more robust. And I've obviously conned a lot of people to work for free on this project!

Phil:

Are there any shows you will be attending in the U.S. later this year or in early 1997?

Linus:

I hope to attend at least the USELINUX conference (or whatever it's officially called) in January. No firm plans...

Phil:

I hope to see you there and let you trade some of the virtual beer I promised for the real stuff.

Gena:

How do you feel about commercialization of Linux?

Linus:

On the whole, it's a very positive thing for me. I'm not worried about the kernel itself or the basic system—all the commercialization is about the distributions and the applications. As such, it only brings value-added things to Linux, and it doesn't take anything away from the Linux scene.

However, I don't think commercialization is the *answer* to anything. It's just one more facet of Linux, and not the deciding one by any means. Let me mention Wabi as an example—a commercial windows emulator (that may actually be out by the time this article hits print; it's in beta testing now).

Wabi is a nice program (I've been using it to make slides with PowerPoint under Linux) and a lot of people will find the reason to pay for it. However, it won't *revolutionalize* Linux the way Wine may do—a freely available windows emulation package will make a lot of difference for the whole market, while Wabi makes a lot of difference to only a subset of the market. Obviously, right now Wabi is a lot more advanced than Wine, but we'll see what happens in a year or two.

Gena:

What challenges do you see the Linux Community facing in the next 10 years?

How do you feel about commercialization of Linux?

Linus:

I'm having trouble planning two weeks ahead, much less 10 years... It's too hard to say what will happen. It all depends on availability of good applications, and the first steps are being taken with both commercial and free end-user applications starting to appear. Linux already has a lot of traditional UNIX applications, what I'm really looking forward to is the desktop personal and business stuff...

Gena:

What advice/pearls o' wisdom would you share with new members of the Linux Community?

Linus:

Umm.... “Be excellent to each other”? No, wrong movie.. Ahh.. “Multiply and populate the earth”? Naah, that's been done too..

Gena:

What keeps you motivated (i.e., why do you keep on doin' what you do)?

Linus:

It's a very interesting project, and I get to sit there like a spider in its web, looking at the poor new users struggling with it. Mwbhahahhaaahaaa..

No, seriously, what kept me going initially after I had “completed” my first test-versions of Linux back in '91 was the enthusiasm of people, and knowing people find my work interesting and fun, and that there are people out there depending on me. That's still true today.

And it really is technically interesting too—still, after these five years. New challenges, new things people need or find interesting. But the community is really what keeps me going.

Phil:

Thanks. A case of virtual beer will be on the way.

Linus:

Virtual, smirtual.. Where's the *real* stuff?

Archive Index Issue Table of Contents

Advanced search

Linux Distributions Compared

Bryan Phillippe

LJ Staff

Issue #29, September 1996

Back in March, we compared the available Linux distributions. This review draws heavily from that work and offers updated infromations on what is available on PC platforms.

A Linux distribution is a package that contains the Linux kernel, application and utility programs and an installation method. While all distributions are based on the same or similar kernels and many of the same utilities, each distribution has something different to offer the consumer. Thus, while there really isn't a *best* distribution, there are distributions that are better for different users. In this article, we attempt to offer you enough information so you can select the distribution best for you.

Is this a cop-out? Are we shirking our duty? It has become clear to us that it would be hard for *Linux Journal* to rank the distributions; our staff members have different personal favorite distributions and defend their choices rationally—occasionally even argumentatively. At *Linux Journal*, we use Linux extensively—nearly exclusively—and we still don't agree which distribution is the best. We have different priorities, skills, and expectations, and we believe this is true of our readers as well.

Before we get too deep into this review, note that as this is being written, the Linux 2.0 kernel is about to be released (the scheduled release date is June 9, 1996). This new kernel offers some major enhancements. (See the interview with Linus in this issue for details.) The distributions will surely be updated to take advantage of this new kernel. Look for an updated review in the special Buyer's Guide issue of *Linux Journal*, which will be out in February 1997.

Some of the descriptive text that originally appeared in the March 1996 review article has been removed. If you want to look at all the information presented in that issue, it is available on our web site, /issue/0023.

Version Numbers

Many new Linux users confuse the version of the **distribution** they are using with the version number of the **kernel**. As mentioned above, the Linux kernel is just one of the many pieces of software needed to create an entire distribution. Each distribution uses version numbers of its own to keep track of the state of the entire distribution, which has more to do with the collection of programs than with the particular kernel involved. Indeed, many distributions have included two or more different kernel versions in one version of the distribution.

However, it's worth understanding the version numbers used for the Linux kernel itself, since the kernel is a key part of any Linux distribution. Kernel version numbers come in three parts: the **major** version number, the **minor** version number, and the **patch-level**. The Linux kernel is being constantly developed by a large team of developers, and while they add new features, they occasionally introduce new bugs. To keep this from causing a problem for Linux users, the developers periodically dedicate several months to fixing bugs and creating a very robust, stable kernel. When this is done, a **stable version** is released with an **even**-numbered minor version number. The developers then begin adding features (and temporarily breaking things sometimes) to **development versions** with **odd**-numbered minor version numbers.

Unless you want to live on the “bleeding edge” of Linux development, you will probably want to stick with the latest stable kernel version. As of this writing, the latest stable kernel version is 2.0, but most distributions are based on the previous production kernel, which is 1.2.13.

Value Added

Several vendors are adding value to existing distributions in various ways. For example, Caldera is adding commercial components to Red Hat Commercial Linux to create their Caldera Network Desktop. Trans-Ameritech bases their value-added work on Slackware, trying to make it easier to install.

Binary File Formats

One of the most common sources of confusion in the Linux world today involves binary file formats (See What is a binary file format?). Most vendors have now switched from the older “a.out” binary file format to the new “ELF” binary file format.

ELF is the binary file format used by Unix System V Release 4, but that doesn't mean a Linux binary in the ELF file format is compatible with SVR4, nor does it mean that SVR4 binaries can run on Linux. The capability for Linux to run **some** SVR3 and SVR4 binaries is provided by the **BCS2** compatibility package, which most distributions include.

One of ELF's features is extensibility; with ELF, it is possible for developers to add features that weren't thought of when the format was first designed. For instance, one Linux developer has noted that he could add icons to ELF executables without breaking any software. Icons weren't considered when ELF was developed, but the format is extensible enough that they can be easily added.

But perhaps you don't care if you can add an icon to your ELF binary, or even if anyone else can. What does ELF do for you? Fundamentally, it makes life much easier for Linux developers. It also has a few esoteric features which make it practical to support some software under Linux that was previously impractical to support. So it gives you more and better software available for Linux.

What do we mean by **based**?

ELF-based means the entire distribution or, at least, almost the entire distribution, consists of ELF binaries. Binaries in the a.out format are not provided with the system, or if they are provided, they either aren't part of the “core” of the system or they are not available in ELF format.

By contrast, “supports ELF” means that, while the distribution is partially or completely built of binaries in the a.out format, the ELF **programming libraries** are included so that ELF binaries will also run.

Media

Many distributions are available solely on CD-ROM. There are several reasons for this:

- So much software is available for Linux that it is impractical to provide it all on floppies.
- It is much easier to install software from CD-ROM than to change floppies once a minute.
- An inexpensive CD-ROM drive costs the same or less than the stack of floppies needed to install a complete Linux distribution.

However, some distributions (including Debian, Red Hat, and Slackware) are available via FTP over the Internet in a form designed to fit on floppies.

Copyright Considerations

Many distributions are freely available over the Internet, although only some are actively distributed in a way that makes it feasible to install them directly from the Internet. You should be aware that while distributors can restrict you from running commercial software components on more than one machine, they cannot restrict you from installing the base Linux software on multiple machines. A Linux distribution packaged in a form which cannot be installed without installing proprietary software under copyright licensing terms more strict than the GNU General Public License is probably in violation of copyright law.

Missing Bugs

One thing you may notice missing from this overview is a list of bugs in each distribution. In general, we assume some noticeable bugs are inevitable in software that is evolving as quickly as Linux, and vendors will be responsive in tracking and eradicating bugs. If they aren't, word of mouth between users will be more effective than any sort of complaining on our part. We also want to avoid showing any bias, and if we start listing bugs we find, we will either show bias or appear to show bias simply by the “choice” of bugs we publish.

Since we can't do a fair comparison of the number and severity of bugs between distributions, we have tried to note only bugs that involved completely missing functionality or which were difficult to work around and hard for the user to avoid.

If working around simple bugs is a problem for you, we urge you to purchase a distribution from a vendor who offers technical support for the installation process. By providing installation technical support, these vendors not only help you with the problems, but force on themselves a vested interest in fixing those bugs. Providing support is expensive, and solving the same problem over and over again is a waste of their time and money.

At the same time, if you buy installation technical support, understand that you aren't buying a lifetime warranty for every part of the system. None of the vendors here provide that as part of their base product. You can buy very comprehensive support packages from several vendors if you need them. Happy shopping!

Caldera Network Desktop 1.0

The Caldera Network Desktop, by Caldera, Inc., comes as a boxed set of 1 CD-ROM, two floppy disks and a user's guide. Caldera Linux is based on Red Hat

Linux 2.1 but uses its own installation system and desktop environment, and it includes many applications which can be found nowhere else.

Installation is very simple. The two disks included are a standard boot disk and a PCMCIA boot disk; you'll only need one of them. Simply boot your machine with the boot disk, insert the CD-ROM, and follow the messages which prompt you along the way. The initial package selection is somewhat granular, providing three predefined choices: Minimal Installation, Default Caldera, and Maximal Installation. But you can always fine- tune package selection later from within the Desktop. The installation process covers all the bases—from network configuration to X setup to LILO—and then requires you to make a rescue disk for safety. Overall, installing Caldera is very easy, and I encountered no unexpected surprises.

Once the system is installed, you can reboot into Caldera Linux, which will prompt you to set a root password and create a user account. After this, you can log in as your new user and start the desktop by typing **startx**. The Caldera desktop looks somewhat MS Windows-like. It has a toolbar at the top which includes options such as File, Window and Help; it supports quick-keys, like ALT+F3 and ALT+F4; has “program groups” and makes extensive use of icons. It has several additional features not found in Windows, however. It supports virtual screens; uses an additional, customizable icon toolbar (similar to fvwm's GoodStuff module); and of course, since it's on top of X, has all the power and configuration capabilities (not to mention stability) of X.

Since Caldera is based on Red Hat Linux, all the standard Red Hat system administration tools are available within Caldera, and all packages are stored in RPM format.

Caldera includes Motif, Accelerated-X (by X-Inside), Netscape, the CRiSP Lite editor, Caldera's Font Server and a full NetWare client (Netware 3.x and 4). Caldera has also ported to Linux WordPerfect 6.0, among other applications. These applications are generally available separately as an “add-on” package, to Caldera users.

Caldera is a very professional Linux distribution which I would recommend to anyone who wants a user-friendly, commercial-grade Linux system or to anyone who wishes to integrate a full-featured Linux network server into a NetWare environment.

Debian .93R6

The only distribution in this lineup which is not developed on a commercial basis, Debian has been a long time in the making. Unlike all the other Linux distributions, Debian is put together by a team of volunteers. While a few

people are in charge of a very small core of the distribution, almost all the decisions are made by consensus, and nearly all the packages are developed independently by members of a large development team.

By following a strict set of rules and using the most powerful packaging technology currently available for Linux, this team has achieved a remarkably coherent Linux package. Debian has complete *dependencies*: when you install one package, it checks to see if other needed packages are also installed. It also checks version numbers if requested; package **A** can insist that package **B** be installed and be version *x* or greater. This makes upgrading any package nearly foolproof. The installation program will also suggest installing other packages the user will likely want if he or she installs package **A**.

Debian installs the entire “base” system from floppy (three floppies) and then you reboot into the base system with enough software to install any other packages you like. This way, all the base system has to support is floppies and hard drives. During the initial “base” installation, you can choose which kernel modules to load, determining what hardware is supported. Debian is highly modularized.

All of Debian's packaging tools run in text mode, so X is not required to use any of Debian's sophisticated upgrade capabilities. The package tools are also very fast.

Debian is currently being transformed from an a.out-based distribution with optional ELF support to a fully ELF-based distribution. [At print time, Debian 1.1 beta was stable; this version will be reviewed in a future issue—Ed.]

Debian's home page can be found on the Web at www.debian.org.

Craftworks Linux 2.0

Craftworks Linux, by Craftwork Solutions, Ltd., comes as a boxed set of one CD-ROM, one floppy disk and a 70 page manual. The single floppy disk is both a boot and a root disk, alleviating the need to make any floppies. Craftworks will install from FTP, CD-ROM, NFS, locally mounted media, tape and PPP. It supports PCMCIA Ethernet and CD-ROM for installation, as well. It has no MS Windows or UMSDOS installation capability and requires 8MB of RAM to install. The distribution is all ELF.

Craftworks' installation utility prompts you for your desired partitioning scheme (and shows you a recommendation), the installation method you want, and what kind of system you want to install. It has three precompiled kernels to choose from: Cyclom, SoundBlaster and Generic. All are ELF 1.2.8 with AIC7xxx and iBCS patches and have support for the ftape loadable module (QIC-40,

QIC-80, QIC-3010 and QIC-3020). The SoundBlaster kernel supports the SB-Pro and SB-16. The Cyclom kernel supports the Cyclades Cyclom-Y driver.

Craftworks Linux uses a unique system for installing packages, called Component Replacement And Fabrication Technology (CRAFT). This system does extensive package dependency checking to assure that all required packages for a particular utility are installed and that packages are intact according to files present, size and checksum. The software itself comes bundled in something called Component Catalogs, grouping together related packages. Installing additional software is as easy as mounting the media containing the catalog and running sysadm (see below) to select and install it.

There are three predefined choices for package installation: Minimal, Developer and X Workstation. The fourth choice is Custom, and it allows you to pick which packages you want to install ahead of time. Selecting any of the first three choices will begin installation without further delay. Package selection comes into play only if you choose Custom. The package selection is considerably more granular than many other big Linux distributions, giving you somewhat less control over exactly what is installed and what is not. Once the installation process has begun, it requires no further attendance. Take a little break, and when you return, simply reboot and your system will come up ready to run. If you chose to use LILO, it will have checked your partition table, and any partitions labeled DOS will have been automatically added to the LILO configuration for you to choose from at boot time.

Craftworks Linux comes with sysadm, a graphical utility that, from a single screen, lets you control many aspects of your system, including modification of user/group information, security settings, tcp wrappers, IP firewalling, X Windows setup, system backup, and much more. The fact that sysadm is a text-based application is another useful characteristic, as it will run in low memory and on inferior or unconfigured display cards, in addition to X Windows.

I was impressed to find that Craftworks comes with shadow password support, IP firewalling, FlexFax and APSfilter. I was even more impressed to find that it also comes with DOSemu, the MS-DOS emulator, and BRU, a high-quality commercial backup and restore utility, correctly configured.

Craftworks Linux is FSSTND-compliant and everything is laid out logically and sensibly. I could easily find all of the configuration files I needed, which were all stubbed out to provide consistent and useful examples.

I think Craftwork Solutions' distribution would be an excellent choice for serious Linux users who want a complete distribution that comes mostly preconfigured and is easy to use and maintain.

Linux-FT 1.2

Linux-FT, by LaserMoon, Ltd., comes as a boxed set of two manuals (a 64 page installation and configuration guide and a Motif 2.0 user's guide) and 6 CD-ROMs. It is based on kernel 1.2.13 and uses the ELF file format. It is the only Linux distribution that is POSIX.1 certified.

Installation of FT is fairly easy. There are far fewer questions to answer compared to many other distributions. X configuration (which is an optional part of the installation process) is also very simple. Linux-FT does not use LILO but, instead, uses its own multi-OS boot loader. One of the most interesting and different things about Linux-FT's installation process is that hardly anything gets copied to your hard drive. Instead, the bare minimal operating system is installed, and from that point, the system reboots with the runtime CD mounted and uses what LaserMoon calls *copy-cache* technology. With copy-cache, whenever you attempt to run an application or use a package that does not exist on your hard drive, the software is transparently installed to it (from the mounted run-time CD-ROM) for later use.

Of course, once you have Linux-FT installed and X up and running, you can begin tweaking your system with “xadmin”, a very powerful, extremely thorough X-based configuration and maintenance utility, to explicitly install packages or remove them permanently, just as with any other distribution.

I was highly impressed to find that xadmin can do anything from configuring user and group settings to formatting a floppy disk. Some of its maintenance capabilities include NFS, lpd, PPP/SLIP, file system and hardware setup, package maintenance, process table and much more. It makes administration a whole lot easier and less time consuming.

Linux-FT includes mwm, the Motif Window Manager, and has shadow passwords, IP firewalling, iBCS and quota support in the kernel. There is a massive amount of software included with FT: three archive CD-ROMs which contain a lot of Linux-related material from all over the Internet, and also a separate web server CD, which contains both the Apache and NCSA web servers, as well as a good deal of HTML- and CGI-related material and web browsers. Additionally, there is the run-time CD, which has the Linux-FT distribution on it, and a source code CD which contains—you guessed it—source for many of the applications and tools on the run-time CD. If there's anything in the Linux-FT distribution that isn't in abundance, it would have to be printed documentation. There's not a lot of “how-to” information.

In my opinion, Linux-FT would be a good choice for any hard core professional Unix/Linux user who wants a powerful, flexible and high-quality, professional, POSIX.1-certified, UNIX-like work environment.

Linux Universe

Linux Universe is a book, with simple installation and configuration instructions and a small reference section, which includes a Linux distribution on CD-ROM. It's translated from German, and the distribution on it is also apparently translated from German, since some of the comments in the scripts are still in German.

Like Linux-FT, Linux Universe has its own full-screen boot loader (instead of using LILO) and is intended to be run with a minimum of software loaded onto the hard disk, caching other programs to the disk as they are needed.

Linux Universe is intended to be a companion to *Linux—Unleashing the Workstation in your PC,* by the same authors. You can purchase Linux Universe alone or in a kit with its companion volume. If you are not already familiar with Linux (or at least Unix), you will want to purchase the whole kit, not just Linux Universe.

The graphical configuration utility is simple to use and seems to work well. It works quickly and intelligently. When filling out the networking configuration, for example, it guesses most of the information once you type in the IP address.

Red Hat Linux 3.0.3

Red Hat Linux 3.0.3 comes as a boxed set of two CD-ROMs and a 190 page user's guide. There is no included boot disk, so you must make your own using another operating system, such as DOS or Unix. Red Hat comes with over 70 possible boot disk images to choose from, depending on your hardware configuration. Red Hat can be installed from floppy, FTP, CD-ROM, NFS, or pre-mounted local file system and will install from PCMCIA ethernet and CD-ROM. There is no support for a UMSDOS installation, although you can install from within DOS. Installing from DOS does not require making a boot floppy.

This latest version of Red Hat can be run live from CD-ROM (disc 2) and can also be installed in 4MB of RAM, provided you can perform a CD-ROM installation. All other types of installation require 8MB of RAM and two additional floppies, which are used to load the RAM disk.

Red Hat 3.0.3 is completely ELF and uses Linux kernel 1.2.13, libc-5.2.18, gcc-2.7.2 and XFree86-3.1.2. It also comes with the Metro Link X-server. Red Hat uses its own package format, Red Hat Package Manager (RPM). RPM packages are easy to install, upgrade and remove, and have built-in dependency checking. The RPM package format is primarily used for Red Hat Linux, but the

software itself has the Gnu Public License and is and is gaining acceptance among commercial Linux developers.

The installation procedure is pretty well polished and feels nearly foolproof. You may choose to do either a text-based or an X-based installation. For both options, the script is intelligent and will correctly identify most of your hardware, providing you with sane defaults for most of the options, including what packages to install, LILO configuration and network setup. The X configuration is very simple and reliable; any novice with a video card manual should be able to get X Windows fired up in no time. This version of Red Hat by far surpasses the previous in package selection. There is simply a staggering number of packages available to install from the Red Hat CD. There are now almost 30 main groups of packages, each subdivided into several smaller groups, which are in turn subdivided into individual packages. This makes for an extremely fine degree of control over exactly what is and is not installed on your system. I noticed one possibly big drawback to such a high degree of control: there does not seem to be a foolproof package dependency system to ensure that required packages are installed, e.g., runtime libraries or the Tcl/TK utilities needed to run Red Hat's own setup and configuration tools. The installation program does have all required packages highlighted by default, so, as a word of advice to the novice, don't unselect anything unless you're sure about it.

Once the package installation is complete, you may reboot into your freshly installed Linux system. All system configuration and setup is done through the Control Panel. The Control Panel is an X Windows-based program that makes configuration of functions like network setup, account administration, and package management a simple process. Red Hat discourages you from indulging in the time-honored trade of editing configuration files by hand, and, instead, urges you to use Control Panel. The drawback to such an easy-to-use, point-and-click interface is if you can't run X, you can't run Control Panel.

I think Red Hat would be a good choice for anyone who is looking for a big distribution that is both fun to use and easy to maintain and install. Red Hat would be good, in particular, for a person new to Linux who may want to dive in head first after testing the water.

Slackware 3.0

Slackware has evolved from a distribution that was available only on the Internet (and on archive CDs provided by various vendors) into an *official* distribution from Walnut Creek CD-ROM. Much like Red Hat, Slackware is still available on those archive CDs and can be used as the base of other Linux distributions, but again, Walnut Creek offers the official version.

Slackware Linux is commonly downloaded from ftp sites around the globe, but it is also frequently mastered on CD and bundled with other Linux material to be resold. It is organized into several “disk sets” which make up the main categories of packages to install, such as “networking” or “X Windows”. Each category is broken down into individual packages for you to choose from at installation time. Due to Slackware's highly categorized organization and its history of being available on the Net, it is one of the easiest distributions to install from floppy (provided you're patient enough to download it). Its package format is a plain Unix .tgz file, gzip-ed tar file. Each disk set also includes a file that describes the contents of the files that make up the set. This format, while highly portable, offers little in the way of dependency checking or upgradability. This will barely slow down a Unix Wizard, but it can be a moderate hang-up for a newbie. Fortunately, Slackware's text-mode installation program can be very verbose and helpful and should suffice for all but the more advanced users, anyway.

Slackware requires a single boot and root disk and will install in 4MB of RAM. It can install from floppy, local partition, CD-ROM and NFS, and has a mature range of non-commercial packages to choose from. Slackware 3 is an all-ELF distribution based on kernel 1.2.13.

Once Slackware is installed, you're pretty much ready to rock 'n' roll. Slackware does provide some configuration tools for networking and basic package maintenance, but it does not really posses any kind of “desktop administration” utility. However, one of Slackware's greatest strengths is its age. Many software packages out there on the Net have been modeled after Slackware, so you should be able to install pretty much anything without a hitch.

Slackware is a good distribution for beginners because it is available freely on most archive distributions and from ftp sites. It's also a great distribution for people who like to really get their hands under the hood and tweak their system (custom configure, edit rc files, etc.).

WGS Linux Pro 3.0

WorkGroup Solutions' WGS Linux Pro 3.0, or Linux Encyclopedia, is a large Linux encyclopedia (approximately 1200 pages) containing all the freely redistributable documentation from the Linux Documentation Project and 4 CD-ROMs of Linux-related material and distributions. There are more FAQs, docs and HOWTOs here than I could list, examples of which include the “Network Administrator's Guide”, “Firewalling and Proxy Server HOWTO”, “Linux SCSI HOWTO”, “Kernel Hacker's Guide” and many, many more.

Disc one contains the main distribution, which is Red Hat Mother's Day Release. Mother's Day Release is based on Linux kernel 1.2.11 and uses the a.out binary

format as its primary file format. Note that WGS Linux Pro version 4 will be released shortly and will be based on Red Hat 3.0.3, described above. The other three discs are supplemental and contain all sorts of software, which is provided for the user who requires later releases of software(and possibly developmental material), which may not be stable.

The installation of Linux Pro could not have been simpler. Although it does not contain any boot or root floppy disks, if you're using DOS, you simply insert the first CD-ROM into your drive and type “install” (or double click the icon from Windows). An installation program will prompt you for your CD-ROM drive type, Ethernet card type and SCSI type, and build boot and root floppies for you. Then, simply reboot and the installation begins. If you don't have DOS handy, you can build boot and root floppies the same way you normally would for this version of Red Hat, using rawrite, dd, cat or a mkfloppy script. Since Linux Pro 3 is based on Red Hat, the section on Red Hat mostly applies, with a few exceptions. The version of Red Hat used for Linux Pro is a.out and requires 4MB of RAM to install. Its packages are in a format called “.rpp” instead of Red Hat's current package format, which is “.rpm”.

If you're looking for an easy-to-install Linux distribution with a lot of hard-copy documentation, this might be the one for you.

Yggdrasil Plug and Play Linux, Fall '95

The first Linux distribution designed for CD, Yggdrasil has had continued popularity due, in part, to graphical configuration, bootable floppies included with the distribution, and multimedia support. Yggdrasil also has a history of including kernels with patches they have tested but aren't included in the standard Linux kernel. Fall '95 continues this tradition.

After installing the “Suggested” configuration—at 250 MB, the only choice smaller than “Everything”, which takes 600MB—it is impossible to compile the kernel without either mounting the CD or installing some packages that aren't installed as a part of the “Suggested” configuration. The manual doesn't say what packages need to be installed to build the kernel with the CD removed, and since the control panel doesn't show which packages are already installed, it's not clear what packages need to be installed. Even after installing all the needed software components, we still had to figure out we had to remove /usr/src/linux/include/linux/version.h in order to successfully build the kernel; we were not able to find that in the documentation. This would not have been an important issue except for the fact that the standard kernel didn't support the 3C509 Ethernet card, and we needed to build a kernel with 3C509 support in order to test the networking setup. Further testing showed several functions of the basic control panel, such as printing, also did not function with the “Suggested” configuration unless the CD-ROM was mounted.

Although running Yggdrasil without the system CD mounted can take some work, Yggdrasil is one of the few distributions able to run entirely from the CD. This capability has been included with Yggdrasil for some time and is a great way to demonstrate Linux to skeptical friends. Not surprisingly, running multimedia applications under X entirely from the CD-ROM takes considerable memory; it doesn't work well with less than 16 MB of RAM.

Yggdrasil doesn't set up networking as part of the installation, but it does provide a graphical tool for setting up basic networking once you get X set up. It also has a reasonably easy X setup program that gets invoked automatically when you start X, if you haven't already configured X.

Yggdrasil doesn't come with some programs that are now considered standard with Linux, such as

rxvt

. On the other hand, it does seem to include a fairly full range of multimedia tools. The PC Speaker sound patches, which provide sound capabilities through the standard PC speaker, are already applied to the supplied kernel source tree and are one of the configuration options. Yggdrasil also comes with XEmacs, a version of Emacs with a good graphical user interface.

Reading the Feature Chart

First, a word of warning: while the feature chart is intended to help you compare the features offered by different distributions, it doesn't say anything about the quality of the features. The chart doesn't tell the whole story, nor does this article, because it is impossible for a chart or article to do so. The chart *can* give you an idea of what the vendors are trying to provide with their distributions.

A “no” isn't necessarily a strike against a distribution. Don't sit down and count up the number of boxes with “yes” in them and consider that a reasonable ranking of the distributions. Instead, consider the features that you particularly want and need. If you have further questions, contact the vendors and ask. Buy from vendors with a money-back guarantee if you are worried you'll get stuck with a distribution you don't want.

The labels on the feature chart (page FIXME) aren't entirely self-explanatory, as you may have already noticed. An explanation of the entire chart is in order here. If there is any part that you don't understand or are not interested in, feel free to skip it. Starting from the top...

Like everything else, the **vendor's list price** isn't necessarily the price you will pay. Nearly all these distributions are re-sold. Also, like Linux itself, Debian has no official vendor and no official vendor price. The Debian distribution is made available for ftp from ftp.debian.org, and several vendors provide it, either as the sole distribution on a CD or as part of a collection.

Packaging is technology for easily and correctly installing, removing, and upgrading parts of the system. Each part would include several inter-related files. For example, all the files typically installed together will probably be included in the same part. The first line of the Packaging section indicates whether you can at least add and remove packages with the provided package management tools.

Package upgradability involves preserving (as much as possible) the configuration of the package correctly on the hard drive while installing a new version. This involves, at the very least, some method of distinguishing configuration files from other files, so that when you upgrade, a package's configuration files are not changed, and you do not have to re-configure the package. If the package management tools are unable to preserve the configuration untouched (for example, if the format of a configuration file changes with a new version), the package management tools need to notify the administrator of that fact.

Dependencies are a capability which allows one package to require that another package also be installed. For instance, LaTeX requires that TeX also be installed, so if LaTeX and TeX are provided in separate packages, the package containing LaTeX may require that the TeX package be installed first, since the functionality of the former depends on the latter.

Dependencies are usually absolute—a package either requires or does not require another package in order to function. Debian allows fine shades of distinction: A package can absolutely depend on another package being installed, it can recommend that any sane user would want the additional package installed, or it can suggest that you will probably want to install the other package.

Format refers to the way packages are maintained on the installation media. Debian and Red Hat each use special archives of their own design which contain the extra information needed to implement their special installation features. Slackware installs standard “.tar.gz” or “.tgz” archives created by using tar to archive a set of files and gzip to compress the archive; those archives do not, therefore, have the extra information needed to remove or upgrade archives. Yggdrasil and Linux Universe install by copying files directly from a file

system on the CD-ROM to the hard drive, so they can only use their package management tools to install from the installation CD.

Third-party packages are closely related to format; those distributions which provide single-file packages are capable of installing packages built by third parties, downloaded from the Internet or found on other CDs. As an example of what can be done with third-party packages, Caldera built the Caldera Network Desktop upon Red Hat Commercial Linux (RHCL) and added new packages in Red Hat's RPM format to create a new distribution that is essentially a superset of RHCL.

Source packages means the source code is also installable with the package management tools.

Buildable Source means the package has the capability to build entire packages with one command (which may or may not be integrated with the package management tools).

Multi-architecture means support for more than one binary architecture. The Intel 80386 and above (denoted here as i86) is certainly the most common CPU architecture supported, but Linux also runs on other platforms, including some Amigas and Ataris, DEC's Alpha, the Acorn Archimedes, and Sparc. Some of the distributions are beginning to include support for more than one architecture. We expect, by the time you read this chart, support for more architectures will be, at least, announced by more distributions.

Graphical and text-based package management refer to the tools available for managing packages. **Graphical** refers to native X-based tools, and **text-based** refers to those that run in character mode (even in an xterm or rxvt session).

Series sub-selection refers to logically grouping packages together into intelligent groups while also making the selection of individual components possible. For instance, making a group or series of everything related to the TeX typesetting system, but also allowing the user to choose not to install all the fonts or to leave out LaTeX.

Over the past year, the Linux community has been migrating from the a.out to the ELF **binary file format**. ELF has many advantages, but it has been necessary to move slowly to avoid causing unnecessary pain. However, ELF is now the standard, and we have purposely not included any distributions that do not at least support ELF binaries.

ELF-based distributions can choose to include a.out libraries, in order to provide support for legacy applications.

Most distributions (all in this lineup) provide iBCS2 support for binaries from many i86 Unix platforms.

Supported architectures are the types of machines supported by the distribution.

The standard PC architecture doesn't provide a standard way to boot from CD-ROMs, so in order to install Linux, it is not possible to simply boot a Linux CD-ROM. [No letters to the editor telling us that the Adaptec 2940 provides support for booting from a CD-ROM; that's not "support by the standard PC architecture"—ED] This means the distribution needs to provide some other way of booting. There are essentially two ways of doing this. One is to provide a bootable floppy disk containing the Linux kernel, and the other is to rely on already having DOS installed. Several distributions provide both options. In order to boot from a floppy, either a bootable floppy needs to be included with the distribution, or a way needs to be provided to make a floppy from files on the CD-ROM.

Unfortunately, it's hard to make a single floppy that is sufficient for booting on all platforms—but failing to supply a floppy with a kernel makes the user rely on having another operating system (DOS, Linux, or any version of Unix will do) with which to create the floppy. Fortunately, this is rarely a problem, but if you have no other operating system available (even on a friend's computer) to write the necessary floppies, you will probably want to purchase a distribution which provides the necessary floppies as part of the package.

It is possible to boot Linux directly from DOS, and some distributions provide an option to use no boot floppies at all by first booting DOS and then booting Linux from the CD-ROM.

Some distributions provide a wide variety of boot images, whereas others use one or only a few. If you are able to make floppies yourself, those with more boot floppy choices *may* prove easier to configure for your hardware. The **Boot Images** line doesn't count duplicates for different size drives, it only counts the choices for one size of floppy drive.

All the distributions provide at least the option of booting from floppy, and all of them provide a way to make your own boot floppies from the CD-ROM under DOS and/or Linux. Some also provide some assistance (under DOS, Linux, or both) to make choosing or creating the floppy or floppy set easier.

The **number of floppies required** to install the system varies widely. Debian installs its entire "base system" from a set of five floppies and then requires one blank floppy after the installation to create an appropriate boot disk. If you

boot the Linux Universe CD-ROM from DOS, it can install without any floppies at all.

While most of this article has assumed you are installing from CD-ROM (after all, that's what most of the vendors provide), that's not necessarily true. You may wish to copy packages to a subdirectory of a local hard drive and install from there, or you may wish to install from floppy, via NFS from a local server, via anonymous FTP, or from a tape.

In order to install on most notebooks, support for PCMCIA (also known as PC Card) is required, unless you are want to install one of the distributions available on floppies and are interested in shuffling floppies for hours. If you have a PCMCIA Ethernet or SCSI adaptor supported by the Linux PCMCIA card services package, some distributions make it possible for you to install from a network location over Ethernet or from a SCSI CD-ROM connected to the SCSI adaptor. This can be a real time saver.

If you don't want to partition your disk, some distributions provide the option of installing in a **\LINUX** subdirectory of a DOS file system using a Linux file system called UMSDOS. While this imposes on Linux all the inefficiencies of the DOS file system, it is an easy way to get started using Linux without committing a partition to do so. Yggdrasil goes one step farther, allowing you to install the base Linux system without booting Linux, using a DOS program to do the installation.

Most distributions now have customized documentation written about them, which is either included with the distribution by default or is available as an option. Some distributions have been covered in other books available in bookstores, and some books include distributions.

One of the most difficult parts of configuring a Linux system is configuring the X Window System. The XFree86 Project, Inc. provides a configuration program called xf86config, which has a fairly basic user interface and requires you to know a lot about your configuration. It is, however, quite complete. Some distributions provide tools to make X configuration easier.

Some vendors provide a support program, and some provide several different support programs. This chart only mentions whether at least one program is available—check with the vendors if you have specific requirements that you need met.

Some of the distributions include configuration tools, mostly graphical ones. If you are not familiar with administering a Linux or Unix system, you are likely to find most common tasks simplified by a user interface that handles common

tasks easily. None of the configuration tools provided with any of the distributions prevents you from doing your configuration manually, so there is no reason to complain if you prefer to configure your system by hand.

In particular, we looked for configuration tools for setting up and maintaining a network, adding and deleting users and groups, mounting and un mounting file systems, setting up print services, and manipulating boot scripts (generally kept in /etc/init.d). Again, we did not try to rank them in usefulness, and some of the scripts were definitely more functional than others for our purposes. On the other hand, a script with more functionality than one user needs could include needless, confusing detail for another user. Configuration scripts should have a review of their own some day, but mentioning whether or not they exist is a first step.

Summary

Linux distributions have changed a lot in the few years they have been around. The Linux File System Standard has improved their ability to inter-operate; binaries compiled for one distribution are now more likely to run on another distribution, because standard files are more likely to be in the same place from one distribution to another. As distributions competed, bugs have been worked out of all of them, thus, users have benefited from distributions they haven't even used.

Since no single distribution provides an optimal environment for everyone, each distribution contributes something to the pot. Furthermore, having distributions targeted for different kinds of users means no distribution is pressured into complete mediocrity by an attempt to be all things to all people. It's a well-worn cliché that to do all things equally well is to do nothing at all well...

Bryan Phillippe is a 21-year-old Linux enthusiast who also enjoys the company of his fiancee, rollerblading and street-style snowboarding. He can be reached at bryan@terran.org.

Archive Index Issue Table of Contents

Advanced search

Licenses and Copyright

Michael K. Johnson

Issue #29, September 1996

If you program for Linux, you do need to understand licensing, no matter if you are writing free software or commercial software. Here's a road map.

How often do you see on Usenet a sentence like “DISCLAIMER: I am not a lawyer, and I am not qualified to give legal advice ?” Perhaps one day tort reform will be passed and non-lawyers won't have to make a disclaimer like this in order to avoid being sued. Until then, I need to make the same disclaimer: I am not a lawyer. I don't even play one on T.V. What you read here is my interpretation of these licenses, and while I have done my best not to err, to err is human. Before betting your life, your fortune, or your sacred honor on my interpretation you should read them yourselves, come to your own understanding of them, and if you are not comfortable **ask a lawyer**.

Licensing versus Copyrighting

The first thing you need to understand is the difference between copyright and license. A copyright safeguards the ownership of an intellectual property. If you hold *copyright* to some intellectual property, you have several rights regarding that property, and you can assign (sell or give) some or all of those rights to others. A *License*, on the other hand, is a document lets someone use your intellectual property.

For example, the GNU General Public License (GPL), which is often incorrectly called a “copyright,” is a license. Applying the GPL to code to which *you* own the copyright does not assign the copyright to the Free Software Foundation (FSF, the authors of the GPL). You still own the copyright—applying the GPL to your code merely lets other people modify and redistribute your code in accordance with the GPL's terms. The GPL is too long to include in this article; you can use FTP to retrieve a copy from the canonical site, prep.ai.mit.edu, in the /pub/gnu directory.

The GPL is one of the most popular licenses in the Linux world; the MIT X and BSD licenses are also popular. The MIT license is very permissive: it says (like all software licenses of which I am aware) that no warranty is provided, that the copyright notices must be maintained, and that MIT's name (or with similar licenses, the name of the copyright holder) cannot be used in advertising or publicity without written permission. The BSD license, by contrast, *requires* that all advertising materials display an acknowledgment of "the University of California, Berkeley and its contributors," while prohibiting using either name as an *endorsement*. In other respects, it is much like the MIT license. If you can't figure out how to acknowledge MIT and BSD without using them, talk to a lawyer. The MIT and BSD licenses are short enough to reprint in this article—see the sidebars.

Let's say you wish to develop *free* software derived from software that is licensed under the terms of any one of these agreements. All you have to do is maintain the current copyrights and insert your copyright notice for the code you add, thereby licensing your additions under the same license as the rest of the code.

However, if you wish to develop *commercial* software derived from free software, there are thornier issues. The MIT and BSD licenses do not require you to release your source code. You must simply follow the terms of the license, which mostly have to do with maintaining copyright notices and following limitations about advertising and promotion. If the original software is licensed under the GPL, however, you *must* release your source code in compliance with the GPL in order for you to distribute the derived work.

Licensing when Developing under Linux

The biggest single concern shared by software developers new to Linux is that because Linux is subject to the GPL, any software written or compiled under Linux is also subject to the GPL. Fortunately, this concern is groundless. Plenty of commercial software is available for Linux, and it does not violate the GPL in any way. Developing and compiling your software on a Linux system does not cause your software to be *derived* from the GPL-licensed Linux source code.

Some people assume that because they use **#include** to include system header files in their application, their application is suddenly considered to be "derived from" those header files. This is not the case; the declarations in header files are legally considered "public interfaces" and cannot be copyrighted. This is the same as any other development platform: the header files on every commercial platform sport copyright notices that assert ownership over the header files, but that doesn't mean your application was derived from those header files.

Some people also assume that if they compile their program with the GNU C compiler (gcc), their program must be licensed under the terms of the GPL. This is not true. The GPL states that “the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program).” Many commercial software vendors compile their products exclusively with GCC; some Unix systems are shipped from the vendor with GCC as the standard system compiler. They are not in violation of the GPL.

You do need to be careful, however, when linking with libraries that are licensed under a slightly relaxed version of the GPL called the GNU Library General Public License, or LGPL. Linux's standard C library, and most of its other libraries, are licensed under these terms, which require you (read the license for further details) to do **one** of the following:

- Link dynamically.
- Provide object files that can be linked to static libraries.
- Provide source code.

The second option isn't very easy to implement, and the third option, while great for freely redistributable software, is less than ideal for more restricted commercial software vendors. The right answer for *everyone*, whether the software is free or not, is to link dynamically. This provides several advantages:

- Your program uses less system memory when running.
- Allows your program to be upgraded automatically when bugs are found in the library.
- Creates smaller binaries.

Since dynamic linking is the default on all normal Unix platforms, using it on Linux platforms is not likely to be a hardship.

Note that this restriction applies only to libraries licensed under the LGPL. Commercial libraries such as the Motif libraries are not affected by this condition, nor are libraries that are licensed under the BSD or MIT terms. Since there is rarely any reason to link statically, you can almost always ignore this issue. You need only worry about it if you want to distribute a program that is staticly linked against an LGPL-licensed library *and* you do not wish to distribute the source code.

Caution

There are a few things that you can do that are incompatible with using the GPL. First of all, you can't derive a work from code licensed both under the GPL

and the BSD license. This is because the GPL forbids imposing any restrictions beyond those imposed by the GPL, and the BSD license imposes the additional restriction that UCB and its contributors be acknowledged in advertisement.

Also, non-disclosure agreements (NDA) that prohibit you from releasing source code developed under the terms of the agreement are blatantly incompatible with using the GPL. If you sign an NDA that doesn't allow you to distribute source code, and then write code for a GPL application under the terms of the NDA, you can *use* your modified version of the GPL application (remember that the GPL doesn't restrict use in any way), but you cannot redistribute it.

Similarly, if your code uses a software patent, the patent must be licensed for everyone's free use, or you cannot redistribute your modified version of the application.

Licensing and the Linux Kernel

The Linux kernel is licensed under the GPL. This means that if you write code linked into the Linux kernel, and want to distribute it, the code *must* be licensed under the GPL. For example, you are not allowed to distribute the driver for a new hardware device as an object file that needs to be linked into the kernel at compile time. To do so would beg a lawsuit brought by everyone who holds copyright to some part of the Linux kernel.

It *is* possible, however, to distribute a driver in binary form only, under any licensing terms you wish, as a kernel loadable module. The kernel provides a public interface to which object modules can be bound at run time. The key words here are *public interface*. A public interface cannot be copyrighted. The declarations in a C header file and the system call interface are also examples of public interfaces, that are also not copyright-able. You can safely think of the kernel-loadable module interface as “system calls for code running with kernel privileges in kernel space”.

However, it is usually not advisable to distribute binary-only kernel loadable modules, for a variety of reasons:

- The public interface is subject to change at any time and is frequently changed.
- It violates the spirit of the GPL that has made Linux a success by taking away users' freedom to support themselves, and some users or potential users may become vocal about losing that freedom.
- Linux developers will not notify you of changes in the interface; you need to be proactive in discovering changes by “joining the kernel-of-the-day club” (downloading, compiling, and running the latest kernel every time a

new development kernel is released) or you will get complaints from users that your software suddenly broke.

Maintaining a binary-only product based on a kernel-loadable module is definitely possible but doing this does take extra work, and if you don't account for the work beforehand, you may be unhappy when you need to take the time. To say this another way: most commercial software development on Linux raises no more issues than are raised on any other platform, but binary-only kernel modules *do* raise other issues that you would need to consider.

Licensing Tricks

Sometimes people want their code to be licensed under terms that are compatible with both the GPL and the BSD licenses. There are three ways to do this.

The first involves the *copyright* instead of the license. You can donate the copyright to the public domain which gives intellectual ownership to everyone in the world, individually and collectively. Anyone can do *anything* at all with the code, with no restrictions whatsoever. Since they *own* the code, they don't need a license to use it.

The second involves writing a very permissive license that allows anyone to use the code for almost any purpose. As long as you don't add any restrictions that aren't in the GPL, and as long as you don't prevent other people from adding restrictions, code protected by this form of licensing can be used for projects that use the GPL or the BSD license, without legal problems. If you choose to write your own license, be *sure* to get legal help in doing so. There are many licenses written by laymen that are legal nonsense, and if you want to have any chance of being able to enforce your license, get a legal opinion. Also, get a legal opinion if you do want other people to be able to use your code, since without a license, it is illegal to use code you don't own. anyone concerned about lawsuits won't be able to use your code if it is licensed under non-enforceable terms, even if you have personally decided you would never sue anyone over use of the code.

Perhaps the best option, however, is this: as the copyright owner, you can license your code to anyone you like under any license you like. You don't have to offer only one set of license terms. You can offer users the choice between, say, the GPL license terms and the BSD license terms. For example, a set of libraries called PAM is being developed which will need to be linked both with GPL applications and BSD applications. Its license first gives the BSD license terms (modified not to mention UCB, since UCB isn't involved in its development), and then says, “ALTERNATIVELY, this product may be distributed

under the terms of the GNU General Public License, in which case the provisions of the GPL are required INSTEAD OF the above restrictions." Some code in the Linux kernel has been licensed in this way.

Shareware Licensing

Shareware hasn't really caught on in the Linux world. Perhaps because so much quality software is available without any sort of payment, and perhaps because Linux grows out of the movement to write free software for Unix and predates the shareware movement, there are only a few popular shareware packages in the Linux world. Most of these provide for *optional* payment for personal use, and required payment only for business or commercial use. There are no *legal* restrictions that keep you from releasing shareware for Linux, but be aware that you are entering relatively uncharted waters if you choose to license a Linux application as shareware.

It's Not That Hard

The most important thing to remember is that licensing isn't a particularly complicated issue in most cases. This entire article has been concerned mostly with exceptional cases. Most normal application vendors and most free software authors will have no licensing issues to resolve. Just be aware of the differences between licensing techniques, so that you will be aware of the issues involved. if you ever do have a problem to resolve.

Michael K. Johnson (johnsonm@redhat.com) is the outgoing editor of *Linux Journal*, and as a programmer has worked on both free and commercial software for Linux. He is now a programmer at Red Hat Software, creators and maintainers of the Red Hat Linux distribution.

Archive Index Issue Table of Contents

Advanced search

Using TeX with xtem

Evelyn Mitchell

Issue #29, September 1996

If the process of creating documents with TeX or LaTeX has confused you, Evelyn may have the answer you have been searching for.

One of the first things you notice about TeX or LaTeX is that neither have a good user interface. In order to run either program, you must type in commands at the prompt line. While running, error and status messages are written to your terminal. The process is more like compiling a program than using a word processor. Just as you would compile, link and then test with a C compiler, so in TeX you compile and preview.

Running TeX or LaTeX is only part of what you need to do to create a good-looking document. TeX compiles your source into a dvi file (DeVice Independent), which is conceptually similar to an object file produced from a C source file. In order to view or print a dvi file, you must use a viewer like xdvi or convert the dvi to a PostScript or a Portable BitMap (pbm) file. This is a completely different process than when a word processor is used—there you get to see how your words will look when they are printed as you type and format them. With a word processor, you can usually print without the extra step of converting between print formats.

Writing documents in a way similar to writing programs is not what most writers are familiar with today. The advantage of this style of production cycle is that it is easier for the writer to focus on the task at hand. That is, while writing, you don't have to worry about hyphenation or page breaks. Also, while formatting, you can concentrate on applying a consistent structure to your document without worrying about how that structure will show up on the printed page. Finally, when you are working with how the structure is displayed, you can be more sure that your formatting will be applied consistently throughout the document, than when you are using Words for Windows, for example.

I use a menu interface to TeX/LaTeX and associated programs called xtem that I found on the Internet at the Comprehensive TeX Archive Network (CTAN). CTAN has a WWW page at O'Reilly and Associates (jasper.ora.com/ctan.html), and is the place to go for fonts, filters, style sheets, back issues of the TeX Users Group newsletter, TUGboat and all the other things you may want in order to use TeX more effectively.

Without xtem, I'd still be trying to get all the pieces together. With xtem, I am creating complex documents in just a few hours. It helps me manage all the tasks I need to accomplish to write and format letters, articles and longer documents. For example, I can manage my bibliography database with BibTeX, check spelling with ispell and even check my TeX syntax with texchk.

The name xtem comes from X11-TeX-menu. It is a Tcl/Tk program written by G. Lamprecht, W. Lotz and R. Weibezahn of Bremen University in Germany (weibezahn@lrw.uni-bremen.de). It requires version 7.4 of Tcl and version 4.0 of Tk. The latest version of xtem is version 4, and I'll be describing this version.

xtem provides on-line help for LaTeX syntax. The LaTeX commands are illustrated with sample output, which is handy for things like demonstrating LaTeX commands with sample output for both LaTeX 2.09 and LaTeX2e. The on-line help very useful when I was first learning LaTeX, but finding it was a little confusing for me. The help window is supposed to come up automatically when you open a .tex file to edit or when you select the “help” button with the middle mouse button. I suspect there was some problem with my installation, because I couldn't get the “right mouse button for help on a menu item” to work.

xtem was easy to install. I picked up the file xtem_texmenu.tar.gz off the Internet via ftp from ftp.lrw.uni-bremen.de in the directory /pub/tex/xtem/. The file is about 1.2MB compressed. It decompresses to about 3MB.

I first decompressed it:

```
$ gunzip xtem_texmenu.tar.gz
```

Then I checked whether I would have to create a directory for the archive with:

```
$ tar tf xtem_texmenu.tar
```

All the files in the archive are under /xtem, so I de-archived them with:

```
$ tar xf xtem_texmenu.tar
```

To use xtem, according to the xtem/README file, I needed Unix, Tcl/Tk, XWindows (X11R4 or R5 or R6) and a three button mouse, as well as TeX/LaTeX,

an editor and a viewer. You'll also need a spell checker, BibTeX for handling bibliographies, makeindex for creating an index, etc.

Most of these programs, e.g., TeX/LaTeX and Tcl/Tk, were already installed on my system as they come with the Slackware distribution of Linux, so I didn't have to install them. [They come with all major Linux distributions as well—ED] The instructions to install Tcl/Tk, if necessary, are included in the xtem/README file.

xtem comes with an English and a German version, and it is easy to add support for other languages. I installed only the English version. Deleting the German version files in xtem/help_german and xtem/locals_german saved about 1.2MB. It also comes with an HTML page that describes all the included files, and their links.

The README instructions were thorough, and the descriptions of the steps to install xtem were in the order I needed them.

The most complex step, and it wasn't very complex, was modifying the /xtem/locals_english/mkcommand.vst file. I deleted where indicated, and chose the high performance option, (mkcommand.3_tcl7.3) because it will work under Linux and Tk7.3. I discovered that I could choose another option while actually running the program, so I didn't worry about picking the wrong one while doing the installation.

I then had to modify the shell script xtem/xtem to tell it where the programs that it manages are located. To discover where they are on your system use the command:

```
$ which programname
```

if you use a C shell, and the command:

```
$ type -path programname
```

if you use the bash shell.

For example, the wishx Tk Shell is installed in /usr/bin/wishx on my system, so I changed the wishcall line in the shell script. I also had to add a line to set the environment variable **XTEMPATH** which contains the directory path where I installed xtem. This will vary depending on your system. On my system the line I added was:

```
$ XTEMPATH=/tmp/xtem
```

Then I uncommented the line which set the default language to English:

```
defaultlanguage="english"
```

by removing the pound sign (#) at the beginning of the line.

The rest of the xtem configuration options can be set while the program is running, just by selecting any of the menu buttons with the middle mouse button.

Once I had checked to see that it worked, I moved the xtem application to /users/local/bin, and the rest of the xtem directory to /users/local/lib, changing the **XTEMPATH** when I did so.

About the only thing I don't use xtem for is printing, even though there is a print option available. Once I have formatted a file in Tex/LaTeX, I take a look at it with either xdvi or Ghostview, depending on whether I'm going to be printing it or not. xdvi looks closer to the printed output than Ghostview, but viewing in Ghostview automatically invokes the dvi to PostScript filter, and saves the PostScript file. My printer is connected to a different computer and I use Ghostscript to print, so when everything looks perfect in xdvi, I view the file in Ghostview and print from the .ps file.

xtem is a program that makes it easier for someone new to TeX to manage all of the different programs. It provides a complete on-line help system with formatted examples of the different LaTeX commands. However, xtem is more than training wheels. It has a couple of essential features I use regularly, since they make TeX file management much easier. The first is cleanup, which allowed me to delete all of the .log, .aux and other files created in a TeX session from one window. The second is that xtem tries to catch “conflict” situations. This means it warns me if I'm spell-checking a file I am already using in an open edit session. stem doesn't prevent me from getting into trouble; it just lets me know that I may be making changes that won't be saved.

I recommend xtem because of the detailed help files for LaTeX syntax, and because it makes managing all the different programs I use with LaTeX easy. It is also easy to customize for different installations, even for someone without a lot of Unix experience. Essentially, xtem has made using LaTeX clearer to me.

Evelyn Mitchell (efm@tummy.com) is a partner in tummy.com, ltd., an Omaha, Nebraska-based Linux consulting and development company. See: http://www.tummy.com.

Archive Index Issue Table of Contents

Advanced search

Getting to Know gdb

Michael Loukides

Andy Oram

Issue #29, September 1996

It's worth making friends with a good C debugger.

There are many reasons you might need a debugger—the most obvious being that you're a programmer and you've written an application that doesn't work right. Beyond this, Linux depends heavily both on sharing source code and on porting code from other Unix systems. For both types of code, you may turn up problems that the original authors didn't have on their platform. So it's worth making friends with a good C debugger.

Luckily, the Free Software Foundation has come through with an excellent debugger named **gdb**, which works with both C and C++ code. gdb lets you stop execution within the program, examine and change variables during execution, and trace how the program executes. It also has command line editing and history features similar to those used with **bash** (the GNU shell) and Emacs. In fact, it now has a graphical interface. But since we've grown up using the command-line interface (and it's easier to show in print) we'll stick to that in this article.

To get full documentation on all gdb commands, read the *Debugging with gdb* manual on-line or order it from the Free Software Foundation.

Compilation for gdb

Before you can use gdb to debug a program, compile and link your code with the **-g** option. This causes the compiler to generate an augmented symbol table. For example, the command:

```
$ gcc -g file1.c file2.c file3.o
```

compiles the C source files file1.c and file2.c, generating an expanded symbol table for use with gdb. These files are linked with file3.o, an object file that has already been compiled.

The compiler's **-g** and **-O** are not incompatible; you can optimize and compile for debugging at the same time. Furthermore, unlike many other debuggers, gdb will even give you somewhat intelligible results. However, debugging optimized code is difficult since, by nature, optimization makes the machine code diverge from what the source code says to do.

Starting gdb

To debug a compiled program with gdb, use the command:

```
$ gdb program [ core-dump ]
```

where ***program*** is the filename of the executable file you want to debug, and ***core-dump*** is the name of a core dump file left from an earlier attempt to run your program. By examining the core dump with gdb, you can discover where the program failed and the reason for its failure. For example, the following command tells gdb to read the executable file qsort2 and the core dump core.2957:

```
$ gdb qsort2 core.2957
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 4.13 (sparc-sun-sunos4.1.3),
Copyright 1993 Free Software Foundation, Inc...
Core was generated by `qsort2'.
Program terminated with signal 7, Emulator trap.
#0  0x2734 in qsort2 (l=93643, u=93864, strat=1)
at qsort2.c:118
118           do i++; while (i <= u && x[i] < t);
(gdb) quit
$
```

The startup is fairly verbose; it tells you which version of gdb you're using. Then it tells you how the core file was generated (by the program **qsort2**, which received signal 7, an “emulator trap”), and what the program was doing (executing line 118). The prompt “(gdb)” tells you that gdb is ready for a command. In this case, we'll just quit.

Both the executable file and the core file arguments are optional. You can supply a core file at a later date with the **core** command.

Basic gdb Commands

With just a few commands, you can get most of your work done in gdb. The basic things you have to do are: look at your source code, set breakpoints, run programs, and check variables.

If you forget which command to use (or want to check for obscure features) use the built-in help facility. You can request a particular command (like **help print**) or on a number of special topics.

Listing a File

To see the contents of the source file from which the executable program was compiled, use the command **list**:

```
$ gdb qsort2
(gdb) list
13     void qsort2();
14     void swap();
15     void gen_and_sort();
16     void init_organ();
17     void init_random();
18     void print_array();
19
20     main()
21     {
22            int power=1;
(gdb)
```

To print specific lines from the file you are currently debugging, use a list command:

```
(gdb) list line1,line2
```

To list the first 10 lines from a particular function, use a list command:

```
(gdb) list routine-name
```

Executing a Program

To run the program you are debugging, use the **run** command. This may be followed by any arguments you want to pass to the program, including the standard input and output specifiers < and >, and shell wildcards (*****, **?**, **[**, **]**). You can't use C-shell history (**!**) or pipes (**|**).

For example, consider running the program **exp** through gdb. The following gdb command runs exp with the argument **-b**, taking the standard input to exp from *invalues* and redirecting standard output to the file *outtable*:

```
$ gdb exp
(gdb) run -b < invalues > outtable
```

That is, this command runs **exp -b < *invalues* > *outtable***. If you have not set any breakpoints or used any other gdb debugging features, exp will run until it terminates, either correctly or incorrectly.

If the program you're debugging terminates abnormally, control returns to gdb. You can then use gdb commands to find out why the program terminated. The **backtrace** command gives a stack backtrace showing exactly what the program was doing when it bombed out:

```
$ gdb badref
(gdb) run
Starting program: /home/los/mikel/cuser/badref
0x22c8 in march_to_infinity () at badref.c:16
16              h |= *p;
(gdb) backtrace
#0  0x22c8 in march_to_infinity () at badref.c:16
#1  0x2324 in setup () at badref.c:25
#2  0x2340 in main () at badref.c:30
(gdb)
```

backtrace (which is usually abbreviated **back**) produces a list of all active procedures and the arguments with which they were called, starting with the most recent. So this display shows that the program died in a function named **march_to_infinity()**; this function was called by the function **setup()**, which in turn was called by the function **main()**. The only thing left is to figure out exactly what wrong in **march_to_infinity()**.

Printing Data

You can inspect the variable values by using the **print** command. Let's use it to see exactly what happened in the previous program. First, we'll list some code to see what we're dealing with:

```
(gdb) list
8
9            p=&j;
10           /* march off the end of the world*/
11           for ( i = 0; i < VERYBIG; i++)
12           {
13                h |= *p;
14                p++;
15           }
16      printf("h: %d\en",h);
17
```

It should already be pretty clear what's happening. **p** is some kind of a pointer; we can test that by using the **whatis** command, which shows us its declaration:

```
(gdb) whatis p
type = int *
(gdb) print p
$1 = (int *) 0xf8000000
(gdb) print *p
$2 = Cannot access memory at address 0xf8000000.
(gdb) print h
$3 = -1
(gdb)
```

When we look at **p**, we see that it's pointing somewhere up in the stratosphere. Of course, there's no *ad hoc* way to know whether this value for **p** is legitimate or not. But we can see if we can read the the data **p** points to, just as our program did—and when we give the command **print *p**, we see that it's pointing to inaccessible data.

print is one of gdb's true power features. You can use it to print the value of any expression that's valid in the language you're debugging. In additions to variables from your program, expressions may include:

- Calls to functions within your program; these function calls may have "side-effects" (i.e., they can do things like modify global variables that will be visible when you continue program execution).

```
(gdb) print find_entry(1.0)
$1 = 3
```

- Data structures and other complex objects.

```
(gdb) print *table_start
$8 = {e_reference = '\e000' <repeats 79 times>,
location = 0x0, next = 0x0}
```

Breakpoints

Breakpoints let you stop a program temporarily while it is executing. While the program is stopped at a breakpoint, you can examine or modify variables, execute functions, or execute any other gdb command. This lets you examine the program's state to determine whether execution is proceeding correctly. You can then resume program execution at the point where it left off.

The **break** command (which you can abbreviate to **b**) sets breakpoints in the program you are debugging. This command has the following forms:

break line-number
: Stop the program just before executing the given line.

break function-name
: Stop the program just before entering the named function.

break line-or-function if condition
: Stop the program if the following ***condition*** is true when the program reaches the given line or function.

The command **break *function-name*** sets a breakpoint at the entrance to the specified function. When the program is executing, gdb will temporarily halt the program at the first executable line of the given function. For example, the **break** command below sets a breakpoint at the entrance to the function **init_random()**. The **run** command then executes the program until it reaches

the beginning of this function. Execution stops at the first executable line within **init_random()**, which is a *for* loop beginning on line 155 of the source file:

```
$ gdb qsort2
(gdb) break init_random
Breakpoint 1 at 0x28bc: file qsort2.c, line 155.
(gdb) run
Starting program: /home/los/mikel/cuser/qsort2
Tests with RANDOM inputs and FIXED pivot
Breakpoint 1, init_random (number=10) at
qsort2.c:155
155             for (i = 0; i < number; i++) {
(gdb)
```

When you set the breakpoint, gdb assigns a unique identification number (in this case, 1) and prints some essential information about the breakpoint. Whenever it reaches a breakpoint, gdb prints the breakpoint's identification number, the description, and the current line number. If you have several breakpoints set in the program, the identification number tells you which one caused the program to stop. gdb then shows you the line at which at which the program has stopped.

To stop execution when the program reaches a particular source line, use the **break *line-number*** command. For example, the following **break** command sets a breakpoint at line 155 of the program:

```
(gdb) break 155
Note: breakpoint 1 also set at pc 0x28bc.
Breakpoint 2 at 0x28bc: file qsort2.c, line 155.
(gdb)
```

When stopped at a breakpoint, you can continue execution with the **continue** command (which you can abbreviate as **c**):

```
$ gdb qsort2
(gdb) break init_random
Breakpoint 1 at 0x28bc: file qsort2.c, line 155.
(gdb) run
Starting program: /home/los/mikel/cuser/qsort2
Tests with RANDOM inputs and FIXED pivot
Breakpoint 1, init_random (number=10) at
qsort2.c:155
155             for (i = 0; i < number; i++) {
(gdb) continue
Continuing.
test of 10 elements: user + sys time, ticks: 0
Breakpoint 1, init_random (number=100) at
qsort2.c:155
155             for (i = 0; i < number; i++) {
(gdb)
```

Execution will continue until the program ends, you reach another breakpoint, or an error occurs.

gdb supports another kind of breakpoint, called a “watchpoint”. Watchpoints are sort of like the “break-if” breakpoints we just discussed, except they aren't attached to a particular line or function entry. A watchpoint stops the program

whenever an expression is true: for example, the command below stops the program whenever the variable **testsize** is greater than 100000.

```
(gdb) watch testsize > 100000
```

Watchpoints are a great idea, but they're hard to use effectively. They're exactly what you want if something is randomly trashing an important variable, and you can't figure out what: the program bombs out, you discover that ***mungus*** is set to some screwy value, but you know that the code that's supposed to set ***mungus*** works; it's clearly being corrupted by something else. The problem is that without special hardware support (which exists on only a few workstations), setting a watchpoint slows your program down by a factor of 100 or so. Therefore, if you're really desperate, you can use regular breakpoints to get your program as close as possible to the point of failure; set a watchpoint; let the program continue execution with the **continue** command; and let your program cook overnight.

Single-step Execution

gdb provides two forms of single-step execution. The **next** command executes an entire function when it encounters a call, while the **step** command enters the function and keeps going one statement at a time. To understand the difference between these two commands, look at their behavior in the context of debugging a simple program. Consider the following example:

```
$ gdb qsort2
(gdb) break main
Breakpoint 6 at 0x235c: file qsort2.c, line 40.
(gdb) run
Breakpoint 6, main () at qsort2.c:40
40      int power=1;
(gdb) step
43      printf("Tests with RANDOM inputs
and FIXED pivot\n");
(gdb) step
Tests with RANDOM inputs and FIXED pivot
45      for (testsize = 10; testsize <=
MAXSIZE; testsize *= 10){
(gdb) step
46            gen_and_sort(testsize,RANDOM,FIXED);
(gdb) step
gen_and_sort (numels=10, genstyle=0, strat=1) at
qsort2.c:79
79      s = &start_time;
(gdb)
```

We set a breakpoint at the entry to the **main()** function, and started single-stepping. After a few steps, we reach the call to **gen_and_sort()**. At this point, the **step** command takes us into the function **gen_and_sort()**; all of a sudden, we're executing at line 79, rather than 46. Rather than executing **gen_and_sort()** in its entirety, it stepped “into” the function. In contrast, **next** would execute line 46 entirely, including the call to **gen_and_sort()**.

Moving Up and Down the Call Stack

A number of informational commands vary according to where you are in the program; their arguments and output depend on the current frame. Usually, the current frame is the function where you are stopped. Occasionally, however, you want to change this default so you can do something like display a number of variables from another function.

The commands **up** and **down** move you up and down one level in the current call stack. The commands **up** *n* and **down** *n* move you up or down *n* levels in the stack. Down the stack means farther away from the program's **main()** function; up means closer to **main()**. By using **up** and **down**, you can investigate local variables in any function that's on the stack, including recursive invocations. Naturally, you can't move down until you've moved up first—by default you're in the currently executing function, which is as far down in the stack as you can go.

For example, in **qsort2()**, **main()** calls **gen_and_sort()**, which calls **qsort2()**, which calls **swap()**. If you're stopped at a breakpoint in **swap()**, a **where** command gives you a report like this:

```
(gdb) where
#0  swap (i=3, j=7) at qsort2.c:134
#1  0x278c in qsort2 (l=0, u=9, strat=1) at
qsort2.c:121
#2  0x25a8 in gen_and_sort (numels=10, genstyle=0,
strat=1) at qsort2.c:90
#3  0x23a8 in main () at qsort2.c:46
(gdb)
```

The **up** command directs gdb's attention at the stack frame for **qsort2()**, meaning that you can now examine **qsort2**'s local variables; previously, they were out of context. Another **up** gets you to the stack frame for **gen_and_sort()**; the command **down** moves you back towards **swap()**. If you forget where you are, the command **frame** summarizes the current stack frame:

```
(gdb) frame
#1  0x278c in qsort2 (l=0, u=9, strat=1) at
qsort2.c:121
121                     swap(i,j);
```

In this case, it shows that we're looking at the stack frame for **qsort2()**, and currently executing the call to the function **swap()**. This should be no surprise, since we already now that we're stopped at a breakpoint in **swap**.

Machine Language Facilities

gdb provides a few special commands for working with machine language. First, the **info line** command is used to tell you where the object code for a specific line of source code begins and ends. For example:

```
(gdb) info line 121
Line 121 of "qsort2.c" starts at pc 0x277c and
ends at 0x278c.
```

You can then use the **disassemble** command to discover the machine code for this line:

```
(gdb) disassemble 0x260c 0x261c
Dump of assembler code from 0x260c to 0x261c:
0x260c <qsort2>:        save  %sp, -120, %sp
0x2610 <qsort2+4>:      st  %i0, [ %fp + 0x44 ]
0x2614 <qsort2+8>:      st  %i1, [ %fp + 0x48 ]
0x2618 <qsort2+12>:     st  %i2, [ %fp + 0x4c ]
End of assembler dump.
```

The commands **stepi** and **nexti** are equivalent to **step** and **next** but work on the level of machine language instructions rather than source statements. The **stepi** command executes the next machine language instruction. The **nexti** command executes the next instruction, unless that instruction calls a function, in which case **nexti** executes the entire function.

The memory inspection command **x** (for “examine”) prints the contents of memory. It can be used in two ways:

```
(gdb) x/nfu addr
(gdb) x addr
```

The first form provides explicit formatting information; the second form accepts the default (which is, generally, whatever format was used for the previous **x** or **print** command—or hexadecimal, if there hasn't been a previous command). ***addr*** is the address whose contents you want to display.

Formatting information is given by ***nfu***, which is a sequence of three items:

- ***n*** is a repeat count that specifies how many data items to print;
- ***f*** specifies what format to use for the output;
- ***u*** specifies the size of the data unit (e.g., byte, word, etc.).

For example, let's investigate **s** in line 79 of our program. **print** shows that it's pointer to a **struct tms**:

```
79          s = &start_time;
(gdb) print s
$1 = (struct tms *) 0xf7fffae8
```

The easy way to investigate further would be to use the command **print *s**, which displays the individual fields of the data structure.

```
(gdb) print *s
$2 = {tms_utime = 9, tms_stime = 14,
tms_cutime = 0, tms_cstime = 0}
```

For the sake of argument, let's use **x** to examine the data here. The **struct tms** (which is defined in the header file time.h) consists of four **int** fields; so we need to print four decimal words. We can do that with the command **x/4dw**, starting at location **s**:

```
(gdb) x/4dw s
0xf7fffae8 <_end+-138321592>:  9  14  0  0
```

The four words starting at location **s** are 9, 14, 0, and 0—which agrees with what **print** shows.

Signals

gdb normally traps most signals sent to it. By trapping signals, gdb gets to decide what to do with the process you are running. For example, pressing CTRL-C sends the interrupt signal to gdb, which would normally terminate it. But you probably don't want to interrupt gdb; you really want to interrupt the program that gdb is running. Therefore, gdb catches the signal and stops the program it is running; this lets you do some debugging.

The command **handle** controls signal handling. It takes two arguments: a signal name, and what should be done when the signal arrives. For example, let's say that you want to intercept the signal **SIGPIPE**, preventing the program you're debugging from seeing it. Whenever it arrives, though, you want the program to stop, and you want some notification. To accomplish this, give the command:

```
(gdb) handle SIGPIPE stop print
```

Note that signal names are always capital letters! You may use signal numbers instead of signal names.

C++ Programs

If you write in C++ and compile with **g++**, you'll find gdb to be a wonderful environment. It completely understands the syntax of the language and how classes extend the concept of C structures. Let's look at a trivial program to see how gdb treats classes and constructors. Listing 1 contains a listing produced in gdb.

In order to see the program in action, we'll set a breakpoint at the **entry** statement on line 24. This declaration invokes a function, of course—the **entry** constructor.

```
(gdb) b 24
Breakpoint 1 at 0x23e4: file ref.C, line 24.
(gdb) run
Starting program: /home/los/mikel/crossref/ref
Breakpoint 1, main (argc=1, argv=0xefffd8c) at
ref.C:24
```

```
24          entry entry_1(text_1, strlen(text_1),
ref_1);
```

Now we'll enter the function. We do this through the **step** command, just as when entering a function in C.

```
(gdb) step
entry::entry (this=0xeffffcb8, text=0x2390
"Finding errors in C++ programs",
    length=30, ref=0x23b0 "errc++") at ref.C:14
14              e_text = new char(length+1);
```

gdb has moved to the first line of the **entry** constructor, showing us the arguments with which the function was invoked. When we return to the main program, we can print the variable **entry_1** just like any other data structure.

```
(gdb) print entry_1
$1 = {e_text = 0x6128 "Finding errors in C++
programs",
  e_reference = "errc++",
  '\e000' <repeats 73 times>}
```

So C++ debugging is just as straightforward as C debugging.

Command Editing

Another useful feature is the ability to edit your commands in order to correct errors in typing. gdb provides a subset of the editing commands available in Emacs, letting you move back and forth along the line you're typing. For example, consider the command below:

```
(gdb) stop in gen_and_sort
```

If this doesn't look familiar to you, it shouldn't; it's a **dbx** command. We really meant to type **break gen_and_sort**. To fix this, we can type **ESC b** three times, to move back over the three words in **gen_and_sort** (spaces, underscores, and other punctuation define what's meant by a “word”). Then we type **ESC DEL** twice, to delete the erroneous command **stop in**. Finally, we type the correct command, **break**, followed by RETURN to execute it:

```
(gdb) break gen_and_sort
Breakpoint 1 at 0x2544: file qsort2.c, line 79.
(gdb)
```

Emacs has a special mode that makes it particularly easy to use gdb. To start it, give the command **ESC x gdb**. Emacs prompts you for a filename in the minibuffer:

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

Add the executable's name and press RETURN; Emacs then starts a special window for running gdb, where you can give all regular gdb commands. When

you stop at a breakpoint, gdb automatically creates a window displaying your source code, and marking the point where you have stopped, like this:

```
          struct tms end_time, *e;
          int begin, end;
  =>      s = &start_time;
          e = &end_time;
```

The mark **=>** shows the next line to be executed. The position is updated whenever gdb stops execution—that is, after every single-step, after every **continue**, etc. You may never need to use the built-in **list** command again!

This article was adapted from the book “Programming with GNU Software”, published by O'Reilly & Associates.

Archive Index Issue Table of Contents

Advanced search

ILUG Shows Off

Shay Rojansky

Issue #29, September 1996

Linux User Groups all over the world are becoming large and more active. Here's the story of the Israeli Linux Users' Group.

Working and playing with Linux is fun, but it's more fun to do it with others. It was for this reason that I was thrilled to hear that a group of people calling themselves a “Linux User's Group” was going to meet the following Friday morning. Until then, I had heard stories about the amazing cooperation in development and support between Linux users, but it was distant. All I knew was that my home 386 could run Unix, and for the moment that was enough.

The meeting turned out to consist of about 20 people in a room that the Hebrew University's Computer Sciences Institute donated to the cause (well, with a bit of influence from a group member). The idea had been born when Harvey J. Stein (abel@netvision.net.il) posted a message on a local newsgroup wondering if there were any other Linux users out there. At the meeting, we discussed general issues about Linux, and some specific ideas such as Hebrew support for Linux and more meetings in the future. Being a newbie, I was silent throughout the meeting.

The Israeli Linux Users' Group (ILUG) has changed since then. In our third meeting, which took place a year ago, there were over 50 people. It might be misleading to think of us as people who get together once in a while to talk about Linux; almost all communication is conducted via our mailing list—which, in fact, existed before the first physical meeting. Our list currently has over 160 members, and is usually intensive, especially around group events. We discuss issues such as giving help for newbies, installation tips, how to get that SCSI adapter working and installing Hebrew fonts under X.

We have had five physical meetings to date. Most were organized into two parts—in the first part, members would lecture about cool things in Linux, and

in the second part we would discuss Linux “current events”, including shows, meetings, ideas—just about anything.

So far, we have exhibited at two shows--Computax95 and Computax96 (despite the name, this annual event fortunately has nothing to do with taxes). The Computax event is mainly aimed at the general crowd, and does not specialize in any specific field. This meant we had to concentrate more on Doom, DosEMU, X and the idea of a free OS, rather than firewalls, kernels, and the rest of the more “advanced” technical Linux topics.

Our bold group learned of the show about two weeks before its opening date. This called for a rush meeting (plasma-gate.weizmann.ac.il/Linux/meeting-2.html), which rated a very high attendance. After several talks that night , the group enthusiastically attacked the problems of being in the show. The main barriers were financial and organizational. People graciously donated money, equipment and time. Of course, the people who staffed the show were group members who enthusiastically offered their time and effort in shifts over the three days.

Computax95 turned out to be a success. We received a corner booth in the multimedia building, but we had Doom and several *large* speakers to fire back with. At least we weren't in the same building as Microsoft's altar to itself. The big words “Free Unix Operating System” attracted many users, and while most were normal everyday people (that is, MS Windows users), we got the attention of several key industry people walking by.

We had a few problems—due to the lack of time, we were unable to get any CDs, so it turned out we had nothing to sell. That surprised people a bit, and I think we might have covered booth expenses had we sold, but we were content at handing out flyers, slapping “Linux Inside” stickers on people and generally infecting them with the fever. Soon after the show many users joined our list, and brought the hardships of newbies with them—questions asked before reading proper documentation. But it was definitely worth it.

Computax96 was a success as well. Although we still didn't manage to get organized in time, we did have the experience of Computax95 to help us along. This time we turned to Digital for support, and got it. About half of the cost of exhibiting was donated by Digital, and they also donated two Alphas and several Pentiums. In return, the booth was designed by them, which naturally displayed the big Digital logo and advertisements for the Alpha. The other half of the money came, as usual, from group members (about $1000). While we after much effort succeeded in running Linux on the Alphas, the machines were supplied with 24-bitplane TGA cards, and the TGA X server supports only 8-

bitplane cards. However, they were the first Alphastation 255s ever to run Linux.

A member, Ira Abramov (ira@scso.com) offered to order CDs from abroad, and after some looking around, we found one we liked—CD Slackware 3.0.0 from Cheapbytes (www.cheapbytes.com). The CDs arrived in time, and were popular—about 90 were sold. However, due to an overestimate of sales, quite a few CDs were left over, and Ira lost money. Unfortunately, we didn't have enough time to order other CD packages, such as Infomagic's.

We had a multitude of problems when organizing both shows, but they were essentially successful. Linux-ILUG is not an organization in the legal sense of the term. This meant that no businesses were willing to agree to a special deal such as taking back any goods not sold in the show. As we funded most of the show from our own meager pockets, this meant that many Linux-related items could not be sold (books, high-priced CD distributions, etc).

Overall, our activities have been fruitful. Linux is widely known and used in Israel, and not just in educational institutes—Linux is penetrating the commercial sector as well. I think it is very important for each sector to have a LUG in order to keep up the Linux spirit. We get more and more recruits all the time, and provide valuable support (and sometimes even manpower) for those who need it. I think I can safely boast that our group is continually fulfilling its goals, and that a bright future is unfolding with Linux Users' Groups.

Shay Rojansky (roji@cs.huji.ac.il) is a 17-year-old high school student and Computer Science student at the Hebrew University of Jerusalem. He works in his high school as a system administrator (mainly Linux) and in the CS institute at the Hebrew University as a lab assistant.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Bochs: A Portable PC Emulator For Unix/X

Kevin P. Lawton

Issue #29, September 1996

A software solution that allows you to run PC programs ona SPARC station.

By far, the majority of my time involving a computer was spent working with my Sun SPARCstation. I could do almost everything computer related, including e-mail interaction, system administration, net surfing, network Doom (I confess—it was me who circulated that hack so you could run Doom on Solaris 2.3), all on the same machine I used for software development. Occasionally, though, I found myself firing up the ol' PC on my desk, using MS Word to crank out documentation or a memo, or to run one of a handful of smaller utility programs I purchased years ago. Well, that got old quick! Having two keyboards, two monitors, and two computers on one's desk consumes more than desktop real estate—it takes much more time to maintain and introduces frustrations from the disparate OS designs. So, I began looking for a software solution which would allow me to run on my SPARCstation the handful of PC programs I used.

The idea of paying a lot of money to run something I'd already purchased didn't sit well with me, and I was interested in finding something I could extend if necessary. A little searching on the Internet didn't turn up anything low- or no-budget that would run MS Windows 3.1 on a SPARC and for which source code was accessible. The Wine and DOSEMU projects were making great progress, but they would never run on non-x86 architectures. There was an emulator program called pcemu, which would run on a SPARC. It did what it was intended to very well—run DOS. Unfortunately, it was engineered to emulate an 8086, and didn't lend itself to be extended to support 80286 and 80386 features. My search also turned up another 8086 emulator, one which went along with an older version of MINIX, allowing MINIX to run on non-x86 platforms. Since it was geared toward allowing a real mode version of MINIX to run within the emulator, this didn't seem like a good candidate for enhancing either.

There just wasn't anything out there that would do what I wanted it to do or which would be a good starting point if I was willing to put the effort into enhancing it. At the same time, I noticed significant similar interest on the net, specifically in the comp.emulators.* newsgroups. At that point, I began working on Bochs (pronounced “box”), a portable software PC emulator project, with the goal of making it possible to run PC software on a Unix workstation.

Bochs' Journey

Looking back, it's still hard to believe it all came together. The biggest hurdle often was documentation. Here we have a somewhat undocumented DOS (and later Windows) trying to use parts of a multitude of non-standardized and sometimes poorly documented BIOSs, using hardware devices which aren't always documented accurately or fully, and running on a non-open Intel x86 architecture, which has undocumented instructions and features (e.g., LOADALL). If it weren't for the likes of *The Undocumented PC* and *Undocumented DOS*, I would have quit a long time ago. The other large hurdle was (and will be) tracking down and fixing the bugs which seem nearly impossible to find in such a complex system. One wrong bit flipped here, and sometimes the ill effect shows up 10 million instructions later, when the wrong character is displayed on the screen! Bugs which depend on timings proved to be elusive, since they don't always show up, and timing is drastically affected by the debug print statements inserted in the code. I've had occasion to sift through debug files of some 50+ megabytes (which I refer to as stealth bug contrails), only to find I didn't print out the one piece of info I needed!

In the two and a half years I've worked on this software, Bochs has made it past some very important landmarks. The first was booting up MS DOS 5.0 to the A prompt! At that point, I had only very rudimentary text HGA (a monochrome video card) emulation and no keyboard support, so I had to hardwire keystrokes into the keyboard BIOS to get past the time and date prompts during boot and to run DOS commands afterwards. It wasn't pretty, but it showed there was potential.

I was then on my way to implementing the graphics mode of the HGA and basic keyboard support, in an attempt to get Windows 3.0 running in real mode. After many red-eye nights, I managed to get Windows 3.0 to boot up past the initial graphics banner screen. What a sight that was! I remember staring at the screen, soaking it all in, not daring to type any keys, fearing it would hang Windows. Indeed it did, and that's when I knew I was right on track...

Though I had designed the emulator to be 32-bit right from the start, at this point I had only 8086 emulation. It was time to take the plunge and implement the protection and memory management models of the 80286. Surprisingly,

after only a few months, I had coded a large part of 80286 support and was able to get MS Windows 3.1 (and 3.0) running in standard mode (286 protected mode).

Though MS Windows and MS-DOS have been largely the software I focus on running within the emulator, I've had some other interesting experiences. Recently, I've been working with the MINIX project (Andrew S. Tanenbaum and friends) to get it to boot/work under the Bochs emulator. The idea is to allow MINIX (a Unix clone suited for learning how internals work) to run on platforms to which it has not been ported. Recently, I was able to perform a full MINIX installation within Bochs and to boot up MINIX in 286 protected mode (although more work needs to be done).

I worked with and integrated my source code with the DOSEMU project's source, for a while. However, they were so productive with their code changes, it was hard to keep up, especially before things began to solidify with Bochs, and I had soon written my own keyboard and video support.

When I get to 80386 emulation (I've recently began this trek), I look forward to talking with the Wine team to look at the potential of a compile-time, optional merge of Wine and Bochs, allowing Wine to run on non-x86 platforms. Wine contains an MS Windows-to-X Windows/Unix translation technology, but it lacks x86 emulation which would allow it to run on other platforms. A merger would mean a significant performance gain could be achieved while running Windows within Bochs, since the Windows GUI and OS calls would be mapped to native Unix/X functions by Wine, instead of Bochs emulating the whole process entailed by the the Windows call. This is the same basic concept behind Sun's Wabi product.

A Technical Look at Bochs

Since the term “emulator” has been tossed around quite a bit in recent times, let me clarify the different types of emulation. Software such as DOSEMU and Wine are, in my terminology, environmental or OS emulators. That is, they run the executables natively (x86 code running on an x86 processor) and translate or “emulate” the interaction between the OS the executable expects (like MS Windows) and the OS that is currently running (like Linux). This kind of “emulation” is bound to an x86 platform, since no instruction set emulation is provided, but it is very efficient and an excellent alternative to having a dual-boot setup on your x86 machine.

Bochs is a pure emulator in that it emulates each x86 instruction in software, along with the necessary BIOS and hardware you expect to find on a PC. There's one giant decode loop, which closely models the fetch-decode-execute actions of the CPU. Components of the CPU—namely the registers—are

modeled by fields in a large structure representing the CPU. Main memory is represented by a large array of memory within the C program. Input and output devices, such as the keyboard, timers, PIC, etc., are of a very modular design, “plugging into” the rest of the code via the registration of their corresponding IRQ, I/O address space usage, and interrupt service routines.

Bochs needed to emulate a graphics card supported by Windows, and I chose the monochrome Hercules Graphics Adapter (HGA), since it was the simplest to implement. Using an X window to represent the PC monitor and video frame buffer was a portable and natural choice, and it provided a keyboard input mechanism, as well.

Since all instructions are emulated in the C language, Bochs is not bound to any particular processor and is very portable to many other Unix platforms. I want to extend this theme as far as possible, and would like to see Bochs ported to other non Unix-like OSs, like Mac OS.

As with any emulation software, performance is always an issue, as running programs under an emulator is inherently slower than running native software. I've tried to keep performance in mind while coding Bochs, though I am postponing some optimizations until later, so that they don't interfere with the development path towards 386 emulation.

A few people have asked me if I had any plans for adding dynamic compilation techniques to Bochs to enhance the performance. I strongly subscribe to the K.I.S.S. philosophy, and I'd like to keep the source simple, understandable, and manageable, so I doubt I will add it any time soon. There is room in the immediate future for some more traditional performance enhancements, however. I think flags processing is an area with potential performance improvements. Though, one of the blessings of using such a simplistic (and brute-force) method of emulation, devoid of many fancy optimizations, is there are no problems handling self-modifying code, since the instruction stream is decoded and executed on-the-fly.

From the beginning, I coded Bochs to be a 32-bit emulator (386 and up). Internally, registers and other features have always been represented by 32-bit quantities. This has been an immense help, as I continue to add 32-bit processor features to the emulation support.

Compiling and Installing Bochs

Since I began use of the GNU “autoconf” utility, the configuration and compilation process has become much easier and more portable. However, the process of setting up the environment (hard disk and floppy image files, VGA font, etc.) and the installation of DOS/Windows 3.1 within Bochs is a little

lengthier. Consult the INSTALL file within the Bochs distribution for details. However, to give you a feel for the compilation and configuration process, I'm including the commands I use to compile Bochs and install Windows 3.1 on my Linux machine. Note, I'm using tcsh, and commands may vary on different platforms.

First, unpack the distribution:

```
$ cd parent-dir
$ gzip -dc bochs-YYMMDD.tgz | tar xvf -
```

Then configure the Makefile and source code, and compile.

```
$ cd bochs-YYMMDD
$ setenv CFLAGS "-O3 -Wall -m486"
$ ./configure --enable-80286 \\
  --enable-native-floppy
$ make
```

If you are using a Bourne shell, such as bash, setenv will not work. Use

```
$ CFLAGS="-O3 -Wall -m486"; export CFLAGS
```

instead.

You may need to install the “VGA” font. Read the INSTALL file within the bochs distribution for this procedure. Your font directory may be /usr/lib/X11/fonts/misc, /usr/openwin/lib/X11/fonts/misc, or somewhere else.

```
$ cp fonts/vga.pcf font-dir
$ mkfontdir font-dir
```

Create a **bootable** 1.44M floppy containing the DOS FORMAT and FDISK executables, **using a real DOS machine**. Assuming the 1.44MB floppy is drive A:

```
C:> FORMAT /s /u A:
C:> COPY FORMAT.COM A:
C:> COPY FDISK.EXE A:
```

Now insert this floppy into your workstation, and create an image of it as a Unix file. Assuming your 1.44MB floppy drive is /dev/fd0, do this:

```
$ dd if=/dev/fd0 ibs=512 of=1.44
```

Create an empty 1.2M floppy image file for B: (unused in this example):

```
$ dd if=/dev/zero of=1.2 bs=512 count=2400
```

Now create a 20MB hard disk image file (other sizes are possible, but we'll stick with a simple example for now):

```
$ dd if=/dev/zero of=20M bs=512 count=41820
```

Edit the .bochsrc file included at the top level of the source. Change the floppya line to **floppya: file=./1.44**, change the floppyb line to **floppyb: file=./1.2**, and change the diskc line to **diskc: file=./20M**.

Boot Bochs with the 1.44M floppy image file:

```
$ bochs -bootA
```

Bochs should now be running. Press the Return key a couple of times and accept the date and time.

Use FDISK to partition the C: drive (the 20M file). Add the whole drive as a PRIMARY partition. That is, run FDISK, and type Return three times; once to “Create DOS partition or Logical DOS Drive”, once to “Create Primary DOS Partition”, and once to chose the maximum size.

At this point, just click a mouse button on the window to exit bochs. Since Bochs doesn't support rebooting yet, you must exit and restart Bochs, then format the C: drive and add the DOS system files to it:

```
$ bochs -bootA
A> FORMAT /u /s c:
```

Again, quit Bochs by clicking any mouse button in the window.

Now change the **floppya** line in .bochsrc to **floppya: 1_44=/dev/fd0**. Put Disk 1 of the Windows 3.1 installation set in the floppy drive. Then restart Bochs, booting from the hard drive image file, and fire up the Windows 3.1 installation:

```
$ bochs -bootC
C> a:setup
```

Press the Enter key to install Windows now, and press. “c” to do a custom installation. Press Enter again to install Windows in the default C:\WINDOWS location.

In the Windows Setup screen, everything should be recognized fine, except the keyboard type, which you will need to change. Use the arrow keys to select “All AT keyboards”. Continue installing Windows, using keystrokes only (remember, clicking the mouse in the window exits Bochs). There is no point in setting up printers and applications. Allow Windows to create (or change) your AUTOEXEC.BAT and CONFIG.SYS files, and watch Windows boot. Then return to DOS, click in the window to exit Bochs, and restart Bochs and Windows:

```
$ bochs -bootC
C> WIN
```

Figure 1 shows a sample screen. Note that there are some artifacts of bugs in the display code showing.

Current "Hardware" Support

The set of hardware, CPU, and BIOS features supported by Bochs has been largely determined by the suite of software which I've run within the emulator environment. A lot of my effort has been focused on supporting enough features to run MS DOS 5.0 and Windows 3.1. Recently, I've added Minix 1.7.2 to my list of targeted software. Given this, Bochs currently supports the following items: (Keep in mind that new features are always being added, and some of these may only be available in the next release):

8086/80286 instruction set including protection and virtual memory models. An option to the configuration script allows for compiling for either architecture.

Monochrome Hercules Graphics Adapter (HGA) This is a fairly simplistic graphics adapter with character and pixel mapped graphics capabilities. Text and pixel graphics are mapped to an X window.

Floppy drives: 1.44M 3.5", 1.2M 5.25", and 720K 3.5" These are implemented by mapping them to either Unix files of the corresponding size or to the floppy drive on your workstation. Originally, only BIOS software interrupts (like a subroutine call) were supported, but recently I added a minimal amount of Direct Memory Access (DMA) and floppy I/O support for use with Minix.

Hard drives: 10, 20, or 30-Megabyte drives Hard drives are also implemented as a Unix file. Read and write requests to the hard drive are directed to read() and write() Unix system calls, operating on the hard disk image file. Access to the hard drive is currently limited to BIOS software interrupts. It's likely that use of DMA and hard drive I/O will be added.

Master/slave Programmable Interrupt Controller (PIC) Hardware devices which need to communicate their need to interrupt the CPU to be serviced are attached to either PIC. These include the keyboard, system timer, disk controller, etc. A fairly complete emulation of both master and slave PIC is provided.

Non-enhanced keyboard Keystroke input from the X window used for the video display, provides input for the keyboard emulation. Both BIOS keyboard software interrupt (int 16H) and hardware I/O access are provided. The keyboard code still needs a little work.

CMOS functions I've implemented only a partial set of CMOS functions, although it allows both BIOS software interrupt and hardware I/O access. Primarily, functions which deal with CMOS time, date, and shutdown status are provided. A CMOS real-time clock (RTC) has not yet been implemented.

A fair amount of typical BIOS interrupts and data area values These functions include sensing features and capabilities of your PC, memory size information, reading/writing floppy and hard disk sectors, an interval timer, video functions, a bootstrap loader, etc.

Makefile configuration via GNU 'autoconf'

Runs DOS 5.0, Windows 3.0, Windows 3.1 Additionally, I was able to install and boot up Minix 1.7.2, but more work needs to be done.

Why and What for Bochs' Future

There are many more features to add to Bochs to truly make it a useful tool. In the near future, the plan is to focus on adding the 80386 features set, since this will allow Bochs to run up-to-date operating systems, such as Windows 95 and 32-bit code within Windows 3.1. I have other improvements in mind, and I get more ideas from suggestions and input I receive via the Net. Here's a list of additional features I have planned:

386, 486, and other x86 generation features There's a good start on 32-bit emulation already. Coding should progress nicely; testing and debugging will be slower and more arduous. This should allow Bochs to run Minix for 386, Windows 3.1 in enhanced mode, Windows 95, and Linux.

Networking

Floating point coprocessor emulation

Mouse emulation I've already got a good start on emulating a PS/2 mouse, but since it's so intimately integrated with the keyboard hardware, it makes debugging the keyboard/mouse code difficult. Mouse code is currently disabled until I revisit it. I'd really like to get good (low-level) documentation on a Logitech or other bus mouse, in which case I'd happily implement it.

File system redirector This would allow DOS or Windows to map a drive, such as E:, to your Unix directory (e.g., /home/johndoe/windows-drive). You'd also be able to read CDs mounted on your Unix file system.

More performance enhancements I'm hesitant to do much in the way of performance enhancements until 386 features are coded, but afterwards, a lot

of elements of the code need to be examined for potential performance enhancements. This will be a good time to elicit ideas from the performance wizards of the world.

And more...

Resources

For further information about Bochs, the developer's e-mail list, FTP site, etc., please visit world.std.com/~bochs/.

Kevin P. Lawton (bochs@world.std.com) is an '88 Computer Science graduate from the State University of New York at Oswego. He worked for MIT Lincoln Laboratory for 6 and 1/2 years as a software engineer and systems administrator. His off-line interests include mountain biking, skiing, hiking, Ultimate Frisbee, and just about anything else involving adrenaline and the outdoors. find more info about bochs at world.std.com/~bochs/.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #29, September 1996

Our experts answer your technical questions.

Several readers have requested a forum where short, specific questions—especially those regarding installation and setting up a Linux system—could be answered. Others have expressed a desire for better, more specific, information on how to use each of the various Linux distributions.

In an effort to fill these needs we will feature a new column: *Best of Technical Support*.

The column will be in question-and-answer format. We hope to answer both “generic” Linux questions and vendor-specific questions.

The generic Linux questions will be answered by the team of Linux gurus we have assembled. The expert support staffs of the vendors themselves will answer the questions aimed at one vendor or distribution.

Our new column should provide an opportunity to learn more about the features and quirks of the various distributions, as well as be a great learning tool for less experienced Linux users.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Owning the Code

I am considering porting an application I have been using and developing since 1984 to Linux, for commercial distribution.

I e-mailed rms@gnu.ai.mit.edu and asked if, in their opinion. I could own my program copyrights if I wrote in C using the gcc that comes with Linux. They said I could practically own my program, except the gpl for libc requires that I make arrangements for my user to recompile my program for use with another version of gpl. I guess this makes it so that I would have to distribute the source code. Not something I would be willing to do (I have to feed my kids).

Do you concur, that it is basically impossible for me to wholly own a program that I might write for Linux? How do the commercial applications sellers who advertise in your magazine avoid that requirement?

Your consideration is appreciated. Thank you.

p.s. For Linux to get REALLY popular, I would think that the writers of the libc and other items would allow free development rights...maybe this is why there are so few popular apps for Linux.
—johng@frugal.com

Not a Problem with gcc

While rms is always on the side of “source code should be free”, this is a non-problem with programs compiled with gcc. All you have to do is dynamically link your program (the default for Linux) and you are fine. No requirement for source to be included. See the article on Software Licenses in this issue for the details.

Using Progress at SSC?

On page 25 [of the June 1996 issue of *LJ*] you write “here at SSC we use the Progress database (now running on Linux) and....” But Progress doesn't run on Linux, Progress Software says and will never run!!! So what do you use? Postgres95, some other tools???

More generally I think it's time to have some article about database tools available for Linux (*LJ* already did it, but for DB clones only). Now that we have Postgres95, MiniSQL, Yard-SQL, Empress, the choice is hard.
—Vladimir Novikov
vnovikov@starnet.fr

Yes! Progress at SSC

We are using the SCO Unix version of Progress. There is an article on this in the ***Linux Means Business*** *column in this very issue.*

As to your comments about an article on databases for Linux, I think this is a good idea. I will have it added to the "we want an article on ..." list.

A Hex on X?

My friends and I have all found that XF86 is hard to configure. If one doesn't have a well-supported video card, it is confusing because X wants to know silly things like clock cycles, vertical refresh rate, sync frequency, etc. Why does it need to know these things? These specifications are not widely available, and there is no singular authority (this all being free and sensible) to contact to find them, save the original designers. My question is this: is there a non-destructive probing program anywhere?

We ran xf86config several times. First, it asks us about our mouse, which was pretty simple. Then it asks us what video card we have, so we look at the list, but we don't know the type so it ends up being generic. Then it asks for the monitor type. It says that if the sync rates that it will use to probe are wrong (too high) it can fry your monitor. When it probes, it says that the system might hang, and if it does, reboot and don't try again. It hung and we didn't try again.

When we got to this point again, we passed it and it showed me some video modes that might work, and we accepted them and saved it in the configfile. I ran it and it goes through lines of weird stuff, at which point the monitor starts squealing uncontrollably. I hit Ctrl+Alt+Backspace to exit. However, when it goes back to text mode, the text is all fuzzy and not synced, so I shut down ad reboot. We don't think this is a hardware problem. We have a generic video card, generic monitor, no phone number for the vendor, no manual, etc. We have used graphical programs before; most games, svgalib, and even MS-Windows ran the first time through. Granted, if we had an accelerated video card, we couldn't use the features with these apps, but at least it worked. With X, we could probably use these features, but we don't feel that it is an even trade-off of added functionality for those with accelerated hardware, and the loss of functionality for those with generic hardware. We think that the developers should concentrate on making the configuration of X easier, and, if possible, more universal and cohesive for all types of hardware.

Thank you for your time.

Sincerely,
Ethan Wellman Albuquerque, NM

X Un-hexed

While I don't want to justify the current situation or discourage development, I do want to explain why configuration tends to be non-

trivial. First, you should always be able to get the VGA X server running. It should come up with any video board and monitor and there should be no potential for frying your monitor.

The problems come about when you are looking for more capabilities: either higher performance or higher resolution. While video cards tend to come with drivers for MS-Windows and such, they do not come with Linux drivers (yet). This means that you have to do the research work.

Monitors that support standard VESA timings can use one of these standard configurations from the X config files. The reason for being able to specify clock frequencies, sync frequencies and such is so that you can *tune* your configuration for the maximum capabilities of the monitor. For example, a monitor that works with generic 1024x768 resolution might be capable of something more like 1100x900.

Hang in there and, if necessary, settle for less than the ultimate for your hardware. As Linux grows and Linux vendors grow, X configuration will continue to get easier.

Archive Index Issue Table of Contents

Advanced search

Goodbye

Michael K. Johnson

Roger Scrafford

Issue #29, September 1996

Michael says goodbye to *LJ* and hello to Red Hat.

It is always hard to say goodbye.

This is the last issue of *Linux Journal* that I will be editing. I am moving on to do programming for Red Hat Software.

I have enjoyed editing *Linux Journal*.

Linux Journal has been and will continue to be an important part of the Linux community. And I am not really saying goodbye to *LJ* in any real sense—I intend to continue to write *LJ* articles. I will be busy doing things, rather than talking about them, and I enjoy occasionally writing about the things I am doing.

—MKJ

Hello It's me, your new editor. My qualifications?

- I've been a musician since the age of 16—guitars and (now) mandolin;
- my first computer, a Kaypro, ran CP/M;
- my first Linux was 1.1.54;
- I survived nearly fifteen years as a sysadmin in a CAD design environment, and have the nervous tics to prove it;
- and I've been the editor of a weekly magazine.

Phil and the rest of the *Linux Journal* staff have taken me in despite this checkered past. For this I thank them a lot.

I'd like to assure them, and you, that I won't be trying to fill Michael's shoes. That's too big of a task. Besides, it's too early for that kind of hubris; maybe after you get to know me better. Nor do I think that you'll see any big changes... for a while.

The wizards at SSC will help keep me from crashing and burning. What I want most of all is for us to keep doing what we do best—serving the worldwide Linux community. That's you.

I'm looking forward to working with this crew and meeting as many of you as I can. To use one of the most over-used words in what passes for discourse these days, I'm *excited* about being able to give something back to the Linux community through this magazine; this is going to be *fun*.

I look forward to seeing you next issue.

—Scrafford

Archive Index Issue Table of Contents

Advanced search

From the Publisher

Phil Hughes

Issue #29, September 1996

Michael leaves, and where *LJ* goes from here.

Here comes one of those good news/bad news stories. To get the bad news over with, this is Michael K. Johnson's last issue as editor. To quote Michael, “I want to be a geek.” To explain, for the past two years Michael has been working at home as Editor of *LJ* and he wants to work with other “geeks” (I used to call them computer professionals, but I don't want to detract from what he said). He was offered an opportunity at Red Hat Software that features the environment he wants without having to relocate.

He will be missed. When Michael climbed on board, *LJ* was in transition from being run out of my basement to becoming a product of SSC. In the past two years *Linux Journal* has become the single largest product within SSC. We now have two full-time staff members in publishing, one and one-half advertising representatives, and over a dozen other employees who are shared with *WEBsmith*, our sister publication, and the other SSC publishing work.

Michael gets much of the credit for acquiring quality technical material for *LJ* and turning us into a respected publication in the Linux community as a whole. And Michael has always been willing to write something to “fill in the gaps” when an author hasn't come through. We wish him well in his work with the other geeks and encourage him to send an article our way whenever he has a chance.

Now, the good news. In these two and one-half years *Linux Journal* has existed, Linux itself has grown up. There are vendors around the world offering products for Linux. There are Unix software vendors porting their applications to Linux. There are hardware vendors seriously addressing the Linux market.

You don't have to take my word for it—just look at our advertisers. And watch for the “Cool, it works with Linux” logo in ads in other publications. Linux is being taken seriously in lots of places. Two years ago, would you have expected:

- Apple to offer a version of Linux
- Digital to support Linux on the Alpha
- WordPerfect to be available for Linux

This means Linux has a life of its own. While I feel *Linux Journal* gets some credit for spreading the word, Linux has taken off because it is a viable alternative to other operating systems.

Who Is the Staff?

When Michael made his decision to leave *LJ* in April, we started looking for a replacement. In the interim, we pooled our internal expertise. Unlike most magazines, we actually use what we publish a magazine about. On our staff we have well over 50 years of combined time working with Unix or Linux.

There are two recent changes that bolster the *LJ* staff. First, Gena Shurtleff moved into the position of Assistant to the Publisher. Gena was previously in charge of Accounts Payable for SSC. The transition brings an experienced SSC employee onto the *LJ* staff.

SSC also recently hired Marjorie Richardson. She is in the position of Technical Editor and will be shared between *LJ* and other SSC projects. Margie's 6 years of experience with Unix, both as a programmer and technical writer, gives our internal experience a huge boost.

Enter Roger Scrafford

As well as our staff, much of our copy editing is done by consultants. Gary Moore and Roger Scrafford have been working with us for some time and bring their experience, as well as their Linux expertise, to the table.

In our shopping for someone to fill the Managing Editor position we looked at all the people who had been working with us. Of those people, Roger Scrafford looked like he had the necessary skills and, as it turned out, the interest in this position. I have been talking to Roger about the position for the last month and, as this issue goes to press, he has accepted the job. You will see a quick introduction written by Roger in these pages and you will be hearing a lot more from him starting with the next issue.

Where Do We Go from Here?

Linux Journal has always tried to address the needs of the newcomer as well as the experienced user. We recently added our *Linux Means Business* column to show where Linux is being used to offer real commercial solutions.

But we need to do more. We need to do a better job of helping the newcomer get seriously involved, we need to show more of the business solutions that are springing up, and we need to help Linux move into new areas (like embedded applications).

To do this, we need a combination of a good staff, interested readers, committed advertisers and authors excited by the topics about which they write. It is my job to come up with the good staff and, if I do say so myself, I think I have. We see more and more committed advertisers every day, many of whom have the same sort of enthusiasm the developers have. New advertisers mean we can expand the size of *Linux Journal*, and thus, offer more editorial information. Support those guys—they are who make all this possible.

To have interested readers we need to have interesting things to say. That is where you come in. If you are just a reader, tell us what you want to see in *LJ*. We listen. Tell us about your interests.

If you have an interesting story to tell, want to describe a program you wrote or just think you might want to write something at some time in the future, let us know you are around and what you are thinking about. You can request an Author's Guide and “hot list” or send in a query by sending e-mail to info@linuxjournal.com .

Archive Index Issue Table of Contents

Advanced search

Sticking with Progress...

Peter Struijk

Lydia McIntosh Kinata

Issue #29, September 1996

As the publishers of *LJ*, SSC is committed to the use of the Linux operating system throughout our offices. Many of SSC's operations revolve around our Progress database server which still ran, until recently, on a difficult-to-maintain At&T SVR4 486 tied into our Linux network. In this column we outline how we made the conversion to an ELF-based Linux system running the SCO Unix version of Progress 6.3F08.

At SSC, we've been using Progress for years (since 1985), and a Linux port of Progress is not, and, as far as we know, will not, be available from Progress Software Corporation. We decided to investigate the feasibility of using the iBCS emulation package together with the well-supported SCO Unix version of Progress.

SSC's database is used throughout the office for order processing, subscriptions, sales reports and plots, marketing, advertising, billing statements, label printing and shipping. Integrated with the database is the authorization and settlement of credit card orders by means of a certified external program developed in-house. Future plans for the database include automatic ordering from our web site and address matching software with CASS (Coding Accuracy Support System) certification. SSC prefers to stay with Progress, rather than change to another database specifically ported to Linux, such as Postgress, since we have invested considerable time customizing the database for SSC's growing needs. Progress also has a proven track record of being very stable and reliable. That is all very well, but what is iBCS? iBCS stands for the Intel Binary Compatibility Specification, which specifies the interface between application programs and the surrounding operating system environment for i386-based systems. This allows a number of binaries developed and compiled for other Unix OSs to run on Linux. The freely available package is still being developed but a recent version of iBCS compiled

and installed without a hitch under Linux kernel 1.2.13. The demo version of SCO Progress we ordered arrived in a black box with manuals and a DAT tape. Installation did not go flawlessly; following the prescribed procedure worked only up to the point where the remaining archives were supposed to be read from the tape. I ended up reading them “by hand” using cpio and tweaking the installation scripts to look on the disk instead.

After that it turned out the file permissions and group/user IDs were set incorrectly and the standard shell scripts had the wrong directory paths in them. It helps to have an existing Progress v6 setup, although it's certainly possible to fix this without one. At this point, I was able to run the db server in single- and multi-user mode—time to start looking for other, more sneaky problems. It turned out that setting the TZ environment variable and using a local password file is required for SCO Progress; otherwise, you get a lot of memory violations. An invaluable tool to help you solve this kind of problem is **itrace**, and it is included in the iBCS package. It took some preparation to adapt our custom Progress code for the move, but after that the transition was mostly just a matter of copying all the SSC database-related files over to a new system. I didn't even have to do a time-consuming dump and reload of the database, because both architectures were x86-based. After several more days of testing, we were confident enough with the setup to go ahead and order the “real” version of Progress for SCO Unix. For reasons not entirely clear, the program arrived on floppy disks, which seemed more of a pain than the tape at first but it was actually easier to adapt the install scripts to work correctly, requiring only patience and lots of disk swapping. The server wouldn't run in “raw” mode, a major problem with the standard SCO version. That is, database consistency couldn't be guaranteed. This turned out to be a bug in the SCO version and after requesting and applying the latest patch (6.3F08), this was fixed.

Switching from the old server to the new was a pleasant surprise for everyone (except me) and consisted simply of logging on to a different computer—everything else looked and worked the same. We all were prepared for some multi-user glitches, but none occurred.

There are few disadvantages to running SCO Progress under Linux, other than some minor installation incompatibilities and no “official” Linux support (although Progress SCO technical support is quite helpful). I can't think of any others, aside from the fact that Progress v6 under Linux shows the same bugs as under SCO!

On the contrary, there are quite a number of advantages for SSC or any office environment already using Linux. Firstly, over the last few months, the new system proved to be just as stable. Most of SSC's operations revolve around the

Progress database server, and without it, business grinds to a costly halt. It now fits well with the rest of the network; there are no more problems with shared directories or slightly different commands and environments. It's faster and more responsive; record searching and scanning is much faster. One reason for that is Linux has dynamic buffering (using free memory), which speeds up the overall system I/O. And of course, it's cheaper—Linux is free—SCO is not.

But what better proof than to let one of SSC's employees and a long-time user of Progress, Lydia Kinata, rave about the virtues of the new system in her own words.

Lydia Speaks

I've been using the Progress database since April 1994. SSC was a much smaller company when I started, and my job description included a wide range of responsibilities from shipping and order entry to accounts receivable and marketing. I have been in a position to regularly use most, if not all, of the end user capacities of the database. As a user, I waited with bated breath for the transition of the database from AT&T SVR4 Unix to Linux. I was involved in the testing phases of the new database, but I've been around long enough to know that a transition this massive usually does not go smoothly. Phil Hughes, publisher of *Linux Journal* and chief pesterment of SSC, prepared us all for a bumpy ride by announcing repeatedly that Peter and our systems administrator, Liem Banneman, would be working all night on the transition. I was prepared to believe him.

By 5 pm Wednesday night, we had all cleared out our home directories on the computer that was home to the SVR4 version of Progress. I came in the next morning ready to find Peter and Liem bleary-eyed and cranky. I telnet-ed to the computer with the SCO version of Progress installed and settled down to what I thought would be a long day of bugs. I was overjoyed to discover that everything worked! In fact, the database functioned almost exactly the same as before, with a few pleasant exceptions. Under SVR4, vi worked very poorly. New users often were left with the impression that they were hitting capital letters accidentally at the command line or that vi was just “weird”. Whole lines would repeat erratically, or the cursor would not really be where it appeared to be. Under Linux, this problem went away. Certain time-consuming database operations, like global key searches, are faster under Linux; having the same home directory files accessible on all computers is also an advantage. Formerly, we would have to rcp files to have them on the SVR4 system and our personal workstations, which was a mild annoyance.

For most purposes, the database looked and acted at 8 am on Thursday just as it had at 5 pm on Wednesday. To date, we have not experienced any database

crashes, neither have we discovered any bugs that could be traced to Linux. In all cases, bugs could be traced to SCO Progress itself or to temporary bugs at the programmer's level.

SCO Progress running under Linux works.

Peter Struijk (info@linuxjournal.com) has been SSC's programmer-analyst since August 1995, and knows more about SSC's database than could possible be good for anyone. He's originally from the Netherlands, and studied computer science at Delft University, but was lured away by a smart and gorgeous woman from seattle, where they are now happily married.

Lydia McIntosh Kinata A secret Mac user and graphic artist, Lydia McIntosh Kinata (yup, that's my middle name) is a die-hard DOS hater, but had not used Linux or Unix until she came to work for SSC. She started as customer service and shipping goddess, and has moved on to a position as Director of Marketing's personal slave. Lydia has used the Progress database for over two years.

Archive Index Issue Table of Contents

Advanced search

dtree

Phil Hughes

Issue #29, September 1996

dtree is a utility that will display a directory hierarchy or tree.

While Linux comes with hundreds of utilities, something you got used to on another system always seems to be missing. One program in this category is something that will display a directory hierarchy or tree.

While some file managers that run under X-Windows will do this sort of task, it is sometimes very handy to have a command-line version. While not Linux-specific, the dtree utility is such a program.

I will first explain how to use dtree, then explain how it works. If you invoke it by just entering its name it will display the directory hierarchy starting at the current directory. If you invoke it with an argument, that argument is used as the starting directory. For example, if you enter **dtree /home/fyl/Cool**, a tree of directories under /home/fyl/Cool will be displayed.

dtree is written in the finest old-time Unix tradition using common utilities with a short shell script to glue them together. Here is the program:

```
: dtree
: dtree
# print a hierarchy tree starting at
# specified directory (. default)
(cd ${1-.}; pwd)
find ${1-.} -type d -print | sort -f |
sed -e "s,^${1-.},," -e "/^$/d" -e \
"s,[^/]*/\([^/]*\)$,\`-----\1," -e "s,[^/]*/,|      ,g"
```

Before you panic, remember, it is only four lines plus comments. It can't be that hard to figure out. The first step is to run the program and produce some sample output. If you don't have a computer at hand-or want to see what it does before you become a believer-I have included what I get by running it in my current directory (/home/fyl/*LJ*).

```
/home/fyl/LJ
`-----Advertising
`-----Ams
`-----Ams.old
`-----Angoss
|     `-----Orig
|     `-----Shots
`-----Artsys
`-----Buyer
`-----Caldera
`-----Employee
`-----Fms
`-----Jobs
`-----ljml.d
|      `-----src.d
|      `-----tst.d
`-----OSW
`-----Posts
`-----Presentations
|       `-----Amus
|       `-----Boston
|       `-----Cjk
|       `-----Decus
|       |        `-----Old
|       |        `-----Vancouver.96
|       `-----UW
`-----Progs
`-----Related
`-----Work
`-----Write
```

The first line in the output is the name of the directory dtree was run on. This line was produced by the line that begins with **(cd**. Breaking this line down:

- **${1-.}** means use the first argument from the command line **($1)** if it is available, otherwise use **.** which is a synonym for the current directory. Thus, the cd command either changes to the directory specified on the line that invoked dtree or to the current directory (a virtual no-op).
- **pwd** then displays the path name of the current directory.
- The parentheses around the whole line force the command to be run in a subshell. This means the cd command is local to this line and subsequent commands will be executed from what was the current directory when dtree was initially invoked.
- The **find** command prints out all files whose type is d (for directory). The same directory reference is used as in cd.
- The output of find is piped into **find** and the **-f** option tells sort to fold upper and lower case names together.
- The tricky formatting of the tree is done by **sed** in four steps. Each step is set off by **-e**. This is how you tell sed a program follows.
- The first expression, **s,^${1-.},,"** is a substitute command which tells sed to replace everything between the first two delimiters (a comma is used as the delimiter) with everything between the second. The initial **^** causes the match to be performed only at the beginning of the line. The expression that follows is, again, the starting directory reference, and the string between the second pair of delimiters is null. Thus, the requested directory name from the beginning of the output of sort is trimmed.

- The second expression, **/^$/d** tells sed to delete all blank lines (lines with nothing between the beginning and the end).
- The third expression is probably the trickiest. It used the ability to remember a string within a regular expression and then use it later. The expression **s,[^/]*/\([^/]*\)$,\`-----\1,** tells sed to replace the last two strings separated by a slash (/) with a backquote, five dashes and the last string (following the final slash).
- Lastly, the final expression, **-e "s,[^/]*/,| ,g"** tells sed to replace every occurrence of strings that do not contain a slash but are followed by a slash, with a pipe (|) and six spaces.

Unless you are familiar with regular expressions you probably didn't follow all that. But you probably learned something and you can easily use dtree without having to understand how it works.

Archive Index Issue Table of Contents

Advanced search

New Products

Margie Richardson

Issue #29, September 1996

Research Systems ENVI Version 2.5, FairCom Developer CD and more.

Research Systems ENVI Version 2.5

Research Systems announced Version 2.5 of its ENVI image processing software. ENVI (Environment for Visualizing Images) is an image processing application for analyzing remote sensing data (i.e. any type of satellite or aircraft digital imagery). New features include state plane projections to support GIS users, customized annotation, and new file formats. ENVI 2.5 is available for Linux, UNIX and other operating systems. Personal computer pricing starts at $3,350 and workstation pricing starts at $5750 (North America).

Contact:Research Systems Inc., 2995 Wilderness Place, Boulder, CO 80301, Phone: 303-786-9900, Fax: 303-786-9909 E-mail: info@rsinc.com, URL: www.rsinc.com.

FairCom Developer CD

FairCom Corporation announced the release of its new Developer CD, which contains the new c-tree Plus version 6.6A and FairCom Server release 6.06.26B. The CD contains fully functioning FairCom database servers for over 25 supported environments including Linux. It is priced at $895 with full C source code, no royalties. FairCom also announced it has licensed its technology to Computer Associates International, Inc. for use within the CA-Realia product line.

Contact: FairCom Corporation, 4006 W. Broadway, Columbia, MO 65203, Phone: 800-234-8180 (US), 039-35-773-464 (Europe), 0592-29-7504 (Japan), Fax: 314-445-9698, URL: www.faircom.com/.

Applixware for Red Hat Linux

Applix Inc. and Red Hat Software, Inc. announced the availability of Applix's Applixware suite of office automation products for the Red Hat Linux operating system. This product features graphical fully integrated spreadsheet, word processing, graphics, presentation, electronic mail, and HTML authoring software. Red Hat Software bundles Applixware version 4.2 with Red Hat Linux 3.0.3 (already available) at a price of $495.

Contact: Red Hat Software, Inc., 25 Sylvan Rd South, Suite F1, Westport, CT 06880, Phone: 800-454-5502, Fax: 203-454-2582 E-mail: info@redhat.com, URL: www.redhat.com.

Unix Sound System for Linux by 4Front Technologies

4Front Technologies announced the availability of UNIX Sound System (USS) for Linux (version 1.3.81 or higher). USS provides device drivers for a wide variety of sound cards and audio hardware for Linux. Functionally, USS provides support for 8bit and 16bit stereo digital audio playback and sampling, FM sound synthesis, MIDI input and output, and Audio Mixer for setting input and output levels. Retail price for USS/Linux is $50.00.

Contact: 4Front Technologies, 11698 Montana Ave., Suite 12, Los Angeles, CA 90049, Phone: 310-820-7365, Fax: 310-826-2465, E-mail: info.4front-tech.com, URL: www.4front-tech.com.

TEAMate On-line Publishing for Linux

MMB Development announced the availability of TEAMate on-line Web & BBS server software for Linux. TEAMate supports five basic interface protocols including access via a character based terminal, a customizable TEAMate GUI client, a WEB Browser, Query-by-Mail, and a Java client. TEAMate for Linux is available for $495.

Contact: MMB Development Corporation, 904 Manhattan Ave., Manhattan Beach, CA 90266, Phone: 800-832-6022, Fax: 310-318-2162, E-mail: query@mmb.com, URL: teamate.mmb.com.

MkLinux from Apple

Apple announced the first developer release of MkLinux, a port of the Linux operating system for the Power Macintosh based on the Mach microkernel. Right now MkLinux only runs on NuBus-based Power Macs (the 6100, 7100 and 8100). Apple plans to make MkLinux available for the PCI Power Macs and on future PowerPC Platform machines. Purchase price of CD-ROM version of

MkLinux developer release is $10 from Prime Time Freeware (includes complete source code). If you have a fast connection, the entire release is available on-line at: www.mklinux.apple.com. The reference release, MkLinux: Mach/Linux for the Power Macintosh, will retail for $50.

Contact: Julia Carey, Apple Computer Inc., Phone: 408-974-4455, E-mail: julia.c@applelink.apple.com, URL: www.apple.com.

NetSmiths Toolkit from UniPress Software, Inc.

UniPress Software announced the release of the NetSmiths Toolkit. The toolkit contains the most current versions of software needed to set-up and maintain a web site on a Unix platform, including browsers, servers, graphics tools, animation programs and Perl. It is available for $69 on CD or from WWW at www.toolkit.netsmiths.com. Some of the programs are available free at archive.unipress.com.

Contact: UniPress, 2025 Lincoln Hwy, Edison, MJ 08817, Phone: 908-287-2100, Fax: 908-287-4929, Email: info@unipress.com, URL: 5://www.unipress.com.

Empress DataWEB from Empress Software, Inc.

Empress Software announced the release of Empress DataWEB, an application development environment designed for Hypermedia Web Data Management. It is available on CD for $997. The CD includes Hypermedia, RDBMS Lite, SQL server, Database Explorer, Database Administrator Tools and Apache HTTP Server.

Contact: Empress Software, Phone: 301-220-1919, Email sales@empress.com URL: www.empress.com.

Archive Index Issue Table of Contents

Advanced search

Consultants Directory

> This is a collection of all the consultant listings printed in *LJ* 1996. For listings which changed during that period, we used the version most recently printed. The contact information is left as it was printed, and may be out of date.

ACAY Network Computing Pty Ltd
Australian-based consulting firm specializing in: Turnkey Internet solutions, firewall configuration and administration, Internet connectivity, installation and support for CISCO routers and Linux.

Address:
Suite 4/77 Albert Avenue, Chatswood, NSW, 2067, Australia
+61-2-411-7340, FAX: +61-2-411-7325
sales@acay.com.au
http://www.acay.com.au

Aegis Information Systems, Inc.
Specializing in: System Integration, Installation, Administration, Programming, and Networking on multiple Operating System platforms.

Address:
PO Box 730, Hicksville, New York 11802-0730
800-AEGIS-00, FAX: 800-AIS-1216
info@aegisinfosys.com
http://www.aegisinfosys.com/

American Group Workflow Automation
Certified Microsoft Professional, LanServer, Netware and UnixWare Engineer on staff. Caldera Business Partner, firewalls, pre-configured systems, world-wide travel and/or consulting. MS-Windows with Linux.

Address:
West Coast: PO Box 77551, Seattle, WA 98177-0551
206-363-0459
East Coast: 3422 Old Capitol Trail, Suite 1068, Wilmington, DE 19808-6192
302-996-3204
amergrp@amer-grp.com
http://www.amer-grp.com

Bitbybit Information Systems
Development, consulting, installation, scheduling systems, database interoperability.

Address:
Radex Complex, Kluyverweg 2A, 2629 HT Delft, The Netherlands
+31-(0)-15-2682569, FAX: +31-(0)-15-2682530
info@bitbybit-is.nl

Celestial Systems Design
General Unix consulting, Internet connectivity, Linux, and Caldera Network Desktop sales, installation and support.

Address:
60 Pine Ave W #407, Montréal, Quebec, Canada H2W 1R2
514-282-1218, FAX 514-282-1218
cdsi@consultan.com

CIBER*NET
General Unix/Linux consulting, network connectivity, support, porting and web development.

Address:
Derqui 47, 5501 Godoy Cruz, Mendoza, Argentina
22-2492
afernand@planet.losandes.com.ar

Cosmos Engineering
Linux consulting, installation and system administration. Internet connectivity and WWW programming. Netware and Windows NT integration.

Address:
213-930-2540, FAX: 213-930-1393
76244.2406@compuserv.com

Ian T. Zimmerman
Linux consulting.

Address:
PO Box 13445, Berkeley, CA 94712
510-528-0800-x19
itz@rahul.net

InfoMagic, Inc.
Technical Support; Installation & Setup; Network Configuration; Remote System Administration; Internet Connectivity.

Address:
PO Box 30370, Flagstaff, AZ 86003-0370

602-526-9852, FAX: 602-526-9573
support@infomagic.com

Insync Design
Software engineering in C/C++, project management, scientific programming, virtual teamwork.

Address:
10131 S East Torch Lake Dr, Alden MI 49612
616-331-6688, FAX: 616-331-6608
insync@ix.netcom.com

Internet Systems and Services, Inc.
Linux/Unix large system integration & design, TCP/IP network management, global routing & Internet information services.

Address:
Washington, DC-NY area,
703-222-4243
bass@silkroad.com
http://www.silkroad.com/

Kimbrell Consulting
Product/Project Manager specializing in Unix/Linux/SunOS/Solaris/AIX/ HPUX installation, management, porting/software development including: graphics adaptor device drivers, web server configuration, web page development.

Address:
321 Regatta Ct, Austin, TX 78734
kimbrell@bga.com

Linux Consulting / Lu & Lu
Linux installation, administration, programming, and networking with IBM RS/6000, HP-UX, SunOS, and Linux.

Address:
Houston, TX and Baltimore, MD
713-466-3696, FAX: 713-466-3654
fanlu@informix.com
plu@condor.cs.jhu.edu

Linux Consulting / Scott Barker
Linux installation, system administration, network administration, internet connectivity and technical support.

Address:
Calgary, AB, Canada
403-285-0696, 403-285-1399
sbarker@galileo.cuug.ab.ca

LOD Communications, Inc
Linux, SunOS, Solaris technical support/troubleshooting. System installation, configuration. Internet consulting: installation, configuration for networking hardware/software. WWW server, virtual domain configuration. Unix Security consulting.

Address:
1095 Ocala Road, Tallahassee, FL 32304
800-446-7420
support@lod.com
http://www.lod.com/

Media Consultores
Linux Intranet and Internet solutions, including Web page design and database integration.

Address:
Rua Jose Regio 176-Mindelo, 4480 Cila do Conde, Portugal
351-52-671-591, FAX: 351-52-672-431
http://www.clubenet.com/media/index.html/

Perlin & Associates
General Unix consulting, Internet connectivity, Linux installation, support, porting.

Address:
1902 N 44th St, Seattle, WA 98103
206-634-0186
davep@nanosoft.com

R.J. Matter & Associates
Barcode printing solutions for Linux/UNIX. Royalty-free C source code and binaries for Epson and HP Series II compatible printers.

Address:
PO Box 9042, Highland, IN 46322-9042
219-845-5247
71021.2654@compuserve.com

RTX Services/William Wallace
Tcl/Tk GUI development, real-time, C/C++ software development.

Address:
101 Longmeadow Dr, Coppell, TX 75109
214-462-7237
rtxserv@metronet.com
http://www.metronet.com/~rtserv/

Spano Net Solutions
Network solutions including configuration, WWW, security, remote

system administration, upkeep, planning and general Unix consulting. Reasonable rates, high quality customer service. Free estimates.

Address:
846 E Walnut #268, Grapevine, TX 76051
817-421-4649
jeff@dfw.net

Systems Enhancements Consulting
Free technical support on most Operating Systems; Linux installation; system administration, network administration, remote system administration, internet connectivity, web server configuration and integration solutions.

Address:
PO Box 298, 3128 Walton Blvd, Rochester Hills, MI 48309
810-373-7518, FAX: 818-617-9818
mlhendri@oakland.edu

tummy.com, ltd.
Linux consulting and software development.

Address:
Suite 807, 300 South 16th Street, Omaha NE 68102
402-344-4426, FAX: 402-341-7119
xvscan@tummy.com
http://www.tummy.com/

VirtuMall, Inc.
Full-service interactive and WWW Programming, Consulting, and Development firm. Develops high-end CGI Scripting, Graphic Design, and Interactive features for WWW sites of all needs.

Address:
930 Massachusetts Ave, Cambridge, MA 02139
800-862-5596, 617-497-8006, FAX: 617-492-0486
comments@virtumall.com

William F. Rousseau
Unix/Linux and TCP/IP network consulting, C/C++ programming, web pages, and CGI scripts.

Address:
San Francisco Bay Area
510-455-8008, FAX: 510-455-8008
rousseau@aimnet.com

Zei Software
Experienced senior project managers. Linux/Unix/Critical business software development; C, C++, Motif, Sybase, Internet connectivity.

Address:
2713 Route 23, Newfoundland, NJ 07435
201-208-8800, FAX: 201-208-1888
art@zei.com

Archive Index Issue Table of Contents

Advanced search