Linux Journal Issue #1/March 1994


Linux Journal

The Monthly Magazine Of The Linux Community

Linux Code Freeze

Linux vs. Windows NT and OS/2

Inside This Issue:

Interview with Linus Torvalds, Author of the Linux Kernel

Introductory Letter from the Editor

Linux and Hams: an existing Link

Consultants Directory

Features

Linux Vs. Windows NT and OS/2 *by Bernie Thompson*
We continue to see media blurbs and ads for both Microsoft's Windows NT and IBM's OS/2. Both promise to be the oprating system that we need and to take advantage of the Intel 386 and beyond.

Interview With Linus, The Author of Linux *by Robert Young*
The interview with Linux Torvalds, the author of the system kernel of Linux. His thoughts and ideas of Linux past, present and future are truly far reaching.

Onyx *by Michael Kraehe*
We continue to see database applications being developed in fourth generation languages (4GLs), and we continue to seemore and more sophisticated (and expensive) 4GLs.

News & Articles

Linux Code Freeze *by Linus Torvalds*
IC MAKE Part 1 *by Frank B. Brokken and K. Kubat*
Linux and Hams *by Phil Hughes*
Linux Programming Tips *by Michael K. Johnson*
The DF Command *by Phil Hughes*
What's GNU? *by Arnold Robbins*
Cooking with Linux *by Matt Welsh*
The Debian Distribution *by Jon A. Murdock*

Reviews

Book Review Linux Installation and Getting Started *by Phil Hughes*

Columns

Letters to the Editor
From the Editor *by Phil Hughes*

Archive Index

Advanced search

LINUX JOURNAL

Linux vs. Windows NT and OS/2

Bernie Thompson

Issue #1, March 1994

We continue to see media blurbs and ads for both Microsoft's Windows NT and IBM's OS/2. Both promise to be the operating system that we need and to take advantages of the capabilities of the Intel 386 and beyond. In the mean time, development and use of Linux, another system that takes advantage of these capabilities, lumbers along. In this article, Bernie Thompson explores these as three alternatives to just staying with what you have.

Picking an operating system is a dangerous business. You're committing yourself to a couple hours, certainly, or maybe a couple days of manual-reading, file-editing, and hassles. If your real goal was just to get some work done, maybe it would have been simpler to stay with Windows 3.1 and never embark on an adventure in computing.

But, then again, there seems to be a substantial body of computer users who are dissatisfied with DOS and Windows. Some are moving to OS/2, Windows NT, or some other Comdex wonder. Some are even daring enough to wipe out DOS in favor of an anti-establishment system like Linux.

Before you take the plunge, you should know up front what you stand to gain. More importantly, too, what you stand to lose. Here's what lies ahead for you if you want OS/2, Windows NT, or Linux to be part of your future.

Hardware is the First Issue

Don't even think about switching systems until you know what your hardware supports. The wonderful features of a new system won't be compelling if your system doesn't work.

You must have a Intel 386 or better to have any 32-bit choices. Then you need memory. Linux needs 2MB RAM to try out, OS/2 needs 4MB, and NT needs 12 MB. And you need disk space. You need to set aside at least 15MB for Linux,

32MB for OS/2, and 70MB for NT for a good trial run. A full working system will require even more resources.

If these requirements are satisfied, you still have to determine if all the pieces of your machine are compatible. If your machine uses the Microchannel bus (all IBM PS/2s), Linux doesn't support you. If you have a Compaq QVision video board, OS/2 won't use it. If you have a network card with a 3Com 3c501 chip, NT can't talk to it. And these are just samples of some possible compatibility problems. The full list changes often. Incompatibilities constantly recede as better hardware support is added. But a constant stream of new, incompatible hardware is always hitting the market.

Why are computer users put through this ringer? Well, The PC hardware market has few solid standards. IBM-compatible hasn't really meant anything since IBM stopped leading the industry. These nightmares can be avoided by getting your 32-bit operating system the same way you got DOS and Windows—buy a complete computer system with Linux, OS/2 or NT pre-installed. Companies which do this are rare, but you'll save trouble by seeking one out. Let them find the best hardware to fit the operating system you want.

If buying a whole new system isn't an option, you'll have to take the path most Linux, OS/2, and NT users have taken. Just start installing. If you have trouble, be prepared to find out more than you ever wanted to know about the pieces of your system.

Why Operating Systems Matter

Operating systems determine which applications will work, what those applications will look like, and how they will work together.

For example, if you want to run Microsoft's application suite (Word, Excel, Access, and PowerPoint), you're out of luck with Linux. They won't work. With OS/2, they work for now, but the burden is on IBM to keep up since Microsoft abandoned OS/2 in 1991. In the end, Windows 3.1 and Windows NT are the only safe choices for using Microsoft applications.

How applications look and how they work together are determined by the operating system, too.

Table 1. Disk Space Required

Windows NT uses the same program manager—file manager—print manager interface as Windows 3.1. This interface is not elegant, but it has one very significant advantage—it is simple. And because it's not very configurable, users can't do much damage by moving icons around and changing settings.

OS/2 takes the more radical route of a completely object oriented interface. Data and programs are objects which can be arranged in any manner. Clicking on a data object starts the associated application. Dragging data to the printer object prints it. Although OS/2 has a notoriously bland color scheme and layout when first installed, every detail can be re-configured.

With OS/2's flexibility comes a daunting depth of detail for first-time users. It is too easy to get lost. With dozens of windows open, it's a pain to locate and manipulate things. However, these disadvantages fade when the system is used for a while. The detail, power, and regularity of the interface become persuasive.

Linux uses the X/Windows system. X/Windows is a graphical chameleon, able to look and act many ways. The advantage is flexibility and choice. The disadvantage is complexity. Applications may not look and act alike. Many different interfaces are available. This makes user instruction and support more difficult.

Linux is primarily a command-line system where programs are typed in by name, although program managers and file managers are available to ease the transition of a novice user. The same tasks done with Windows and OS/2 are possible under Linux, but they generally require more knowledge and skills. If one knowledgeable user configures the Linux system, most novice users will be comfortable starting and running applications.

Table 2. Features Required

All three systems have a wide variety of books and tutorials available which can help novice users. Although Linux is a free system, it still has a library of books written about it—any book about Unix will apply to Linux. So finding assistance on the use of these systems should not be difficult.

If the issues of interface are surmountable, Linux has many positive characteristics that are not shared by OS/2 and NT. Linux enjoys the advantage of having no guarded secrets, no technology owned by a single company. The source code is freely available, which means it can be inspected and improved upon by any corporate or individual user. And, surprisingly, this common knowledge was used to build a system which is more miserly with memory and disk space than either OS/2 or NT. IBM and Microsoft would actually have much to learn from Linux if they cared to look.

The Foundations

When it comes down to it, an operating system is just a foundation. Choose the foundation that supports the features you need and will need in the future. But

be aware of the high price in memory, storage, and performance that these features exact.

Linux, like OS/2, is designed and optimized to run on Intel 386 and compatible CPUs. By contrast, Windows NT is designed to be ported to many different CPUs. NT is currently available for MIPS, DEC Alpha, and Intel 386. This independence from Intel is an important advantage for NT, because users have more hardware choices.

All three systems support multitasking, which is the ability to have many programs running simultaneously. For example, it is possible to format a disk, download a file from a BBS, and edit in a word processor, all simultaneously. You can't do this using a system like MS DOS, which doesn't support multitasking.

NT supports multiprocessing, which means using more than one CPU in a single machine. An NT PC could have 2 or more processors, all working together. Again, this means more hardware possibilities for the NT user.

NT and Linux both support dynamic caching. Caching stores recently used information in memory, so it is readily available if needed again. OS/2 sets aside a pre-determined chunk of memory to do this (typically 512K to 2MB), whereas Linux and NT will dynamically use as much spare memory as possible. The result is much faster disk access for Linux and NT, because the information is often already in the cache. OS/2's inflexibility causes memory to be wasted when not used, and memory to be used poorly when it is scarce.

Linux, unlike OS/2 and NT, has full multiuser support. Local users, modem users, and network users can all simultaneously run text and graphics programs. This is a powerful feature for business environments that is unmatched by OS/2 or NT.

Table 3. Memory Required

Linux has security systems to prevent normal users from misconfiguring the system. Although Windows NT isn't multiuser, it has security checks for the individual using the machine. It is safe to have a Linux or NT machine available for use by many people, whereas an OS/2 user could (mis)configure the system software.

Linux's security and multiuser features are so well developed because they are traditional features for Unix. Since Linux is “Unix-compatible,” it supports these same powerful features.

Table 4. Applications Supported

The Costs

Every feature supported will tend to make an operating system larger, consuming more memory and storage. Larger systems are also slower than smaller systems when memory is scarce. So the size of a system is an important issue.

NT is the largest of the three systems. NT's support for portability, multiprocessing, and many other features is the cause of its large size. Given a powerful enough machine, NT offers a set of features that is very compelling.

Linux with X/Windows is the next smaller system. Linux itself is very miserly, but X/Windows puts a burden on the system. For most the graphical interface will be worth the cost in resources.

OS/2 is smallest of the three when using a graphical interface. This is the attraction of OS/2. A user need only upgrade to 8MB of RAM to use an object-oriented interface and have a good platform for multitasking DOS, Windows, and OS/2 programs. OS/2 is the strongest of the three for backward compatability with DOS and Windows. OS/2 has sold several million copies in the last two years, primarily because of these strengths.

Linux without X/Windows is the smallest of the three. This is a great sacrifice for many, running without graphical windows. But by jettisoning expensive graphics, the system is smaller and faster than OS/2 or NT will ever be. 4MB RAM, the standard configuration for a DOS/Windows PC, is plenty for most tasks. So Linux can make good use of a low-end 386 PC with little memory, where OS/2 or NT either would not run, or not run well. Systems with lots of memory will be able to use Linux's dynamic caching to achieve unusually high performance. With 16MB RAM, almost 12MB remains to be used for caching and running applications.

In general, the issue of size is a great strength for Linux. Linux was designed to be as small and efficient as possible. NT's most important criterion was portability, and OS/2's was backward compatibility. The result is Linux is the most efficient of the three. And because a company or individual has access to the Linux code, it can be optimized and scaled to suit the hardware and needs of the user. OS/2 and NT do not have this flexibility.

The Practical Results

Windows NT is compelling because it is a solid system that offers freedom from the single CPU Intel world.

OS/2 is compelling because it offers the best system for running 16-bit DOS and Windows applications while moving into the more flexible and powerful 32-bit world.

Table 5. Cost

But both systems still end up locking users into proprietary technology—applications that will only work on either OS/2 or NT. Linux does not pose this danger. Applications written for Linux can be ported to any of the dozens of other Unix systems available. Betting on an “open” technology from IBM or Microsoft is still a risky game. Linux offers freedom from this kind of entrapment.

The greatest difficulty in realizing this freedom is finding high quality applications. To keep from getting locked into a proprietary system, you have to choose applications with support for multiple platforms. If your spreadsheet supports Windows, OS/2, Unix, and Mac, you can be confident that support for additional platforms would also be possible. The trade-off is fewer features and higher prices.

Linux has an interface to run commercial applications designed for other Intel Unix systems like SCO Unix. But the quality of applications is still a problem. For example, there is no commercial word processor for Linux which matches the quality of ones for Windows and OS/2. This kind of glaring inadequacy alone can preclude the use of Linux.

Which System to Use

For the corporate user, Linux will fit in well with a TCP/IP based client-server strategy. Linux can turn low-end hardware into a solid fileserver or PostScript print server. Linux works better than many commercial Unix systems on common Intel hardware. Linux is small and fast. Linux can be completely inspected and customized by anyone. Linux has built-in mail and internet tools. Phone support and documentation for Linux are available.

But there are three disadvantages. One, there are few commercial applications. Two, if something goes wrong, there is no one organization to blame as with OS/2 or NT. Three, Linux's foundations are strong, but Microsoft and IBM are constantly developing new technologies that may leave Linux behind. In general, Linux has the features to make it a better choice than NT or OS/2 in some situations. As Linux gains exposure, more businesses are likely to take advantage of this potential.

For the technical user, Linux offers the exciting chance to tinker with an operating system. All of the system's source code is available. It is a great

learning tool and motivator. And since most current Linux users are technical hobbyists, a wealth of applications are available to suit these tastes. Ray tracers, morphing programs, graphics viewers, compilers, games, and more are all available. Linux does lack full-motion video, speech recognition, and some other cutting-edge technologies. These features, along with OS/2 and NT application development, may be compelling enough to draw the technical user towards OS/2 or NT.

For the novice user, OS/2 or NT is the best 32-bit option. OS/2's object-oriented interface and free technical support are compelling factors. NT's power to sway commercial developers is reassuring. But the safest and most likely choice for the novice user is to stick with the operating system that came with their computer, typically DOS and Windows 3.1. Tackling installation, configuration, and new applications is still not trivial for these three 32-bit systems.

Overall, Linux stacks up surprisingly well for a free system developed by a horde of volunteer programmers. It's foundations are solid. The quantity and quality of many free applications are stunning. If Windows-class applications and an OS/2-class interface are developed for Linux, it will have the compelling features to tackle commercial systems. While many computer users now know only OS/2 and NT, thousands of others have discovered Linux. As all three of these systems quickly improve and evolve, Linux is likely to gain an expanding base of users. Free software has a powerful new platform to build on.

Archive Index Issue Table of Contents

Advanced search

Interview with Linus, the Author of Linux

Robert Young

Issue #1, March 1994

Linus (rhymes with shyness) Torvalds (author of the Linux kernel) traded email with us for several days in January giving us his views on the future direction of Linux (rhymes with clinics) and his ongoing role in its development.

***Linux Journal*:** Ken Thompson was once asked, if he had the chance to do it all again, what changes would he make in Unix. He said he would add an e to the creat system call.

How about you and Linux?

Linus: Well, Considering how well it has turned out, I really can't say something went wrong: I have done a few design mistakes, and most often those have required re-writing code (sometimes only a bit, sometimes large chunks) to correct for them, but that can't be avoided when you don't really know all the problems

If it's something I have problems with, it's usually the interface between user-level programs and the kernel: kernel-kernel relations I can fix easily in one place, but when I notice that the design of a system call is bad, changing that is rather harder, and mostly involves adding a new system call which has semantics that are the superset of the old and then leaving in a compatibility-hack so that the old calls still work. Ugly, and I avoid it unless it really has to be done.

Right now I'd actually prefer to change the semantics of the and write() system calls subtly, but the gains aren't really worth the trouble.

***Linux Journal*:** The most consistent compliment that Linux receives is its stability on Intel PC computers. This is particularly true compared to “real Unices” that have been ported to the Intel platform.

What do you see that was done right in Linux that is causing problems for these other PC Unices?

Linus: There are probably a couple of reasons. One is simply the design, which is rather simple, and naturally suits the PC architecture rather well. That makes many things easier. I'd suspect that the other reason is due to rather stable drivers: PC hardware is truly horrendous in that there are lots of different manufacturers, and not all of them do things the same (or even according to specs).

That results in major problems for anybody who needs to write a driver that works on different systems, but in the case of linux this is at least partially solved by reasonably direct access to a large number of different machines. The development cycle of linux helps find these hardware problems: with many small incremental releases, it's much easier to find out exactly what piece of code breaks/fixes some hardware. Other distributions (commercial or the BSD 386-project which uses a different release schedule) have more problems in finding out why something doesn't work on a few machines even though it seems to work on all the others.

Linux Journal: Have you heard of any problems running Linux on the Pentium chip? Do you expect any?

Linus: I know from a number of reports that it works, and that the boot-up detection routines even identify the chip as a Pentium (“uname -a” will give “i586” with reasonably new kls, as I ignore Intel guidelines about the name). The problems are not likely to occur due to the actual processor itself, as much as with the surrounding hardware: with a Pentium chip, manufacturers are much more likely to use more exotic hardware controllers for better performance, and the drivers for them all won't necessarily exist for linux yet. So I've had a few reports of a Pentium PCI machine working fine, but that the kernel then doesn't recognize the SCSI hard disk, for example.

From a performance viewpoint, the current gcc compiler isn't able to do Pentium-specific optimizations, so sadly linux won't be able to take full advantage of the processor right now. I don't know when gcc will have Pentium-optimization support, but I expect it will come eventually (most of the logic for it should already be there, as gcc can already handle similar optimization problems for other complex processors).

One interesting thing is that the “bogo-mips” loop I use to calibrate a kernel timing loop seems to actually be slower on a Pentium than on an i486 at the same clock frequency. The real-world performance is probably better despite that (the timing loop is just a decrement operation followed by a conditional

jump: the Pentium won't be able to do any super scalar execution optimizations).

***Linux Journal*:** With the end of the road for Intel's 80XXX series chips in sight (although at least a few years away), what chip or hardware platform would you like to see Linux ported to?

Linus: The Amiga 680x0 (x>=3, MMU required) port is already underway and reportedly mostly functional already. I haven't been in any close contact with the developers, as they seem to know what they are doing, but I understand they track the PC versions rather closely, and have most of the features working. I'd expect something truly functional by the end of this year, even though the installed machine base is much smaller.

As to other ports: I'd really enjoy some port to newer and more exotic hardware like the DEC Alpha chips or the PowerPC, but as far as I know nobody is really working on it. The main problem with non-i386 ports is simply lack of momentum: in order to get this kind of port going, you'd need hacker-type people with access to such hardware with “nothing better” to do on it. DEC or IBM has yet to show enough interest that they'd donate hardware and documentation to this worthwhile cause.

***Linux Journal*:** What aspects of Linux are you taking responsibility for on an on-going basis?

Linus: Everything that directly concerns the kernel: some of it I can't actually fix myself (mostly drivers for hardware I don't own and have no idea about), but in that case I still want to know about the problems and try to act as a “router” to the person who actually handles that piece of code. The areas I consider especially “mine” are memory management, the VFS layer and the “kernel proper” (scheduling, interrupt handling etc). Generally things that make up the very heart of the kernel, and on top of which all the other stuff has to go.

***Linux Journal*:** Do you see yourself earning a living from your work in Linux in future?

Linus: Well, I do hope and expect to be able to find a job much more easily due to linux, so yes, indirectly at least I hope to be able to make a living off this, even though the work itself might be completely unrelated. As to whether it would actually concern linux itself in some way, I don't know

***Linux Journal*:** The use of Linux is growing exponentially around the world. However, unlike commercial products, there is no central registry for Linux users.

What is your “best guess” of the number of machines ruing Linux worldwide today and what would you base an estimate on.

Linus: I actually have no good idea at all: I haven't really followed either the CD-ROM sales or any ftp statistics, so it's rather hard to say. I guesstimate a user base of about 50,000 active users: that may be way off-base, but it doesn't sound too unlikely. The c.o.l. newsgroup had about 80,000 readers according to the network statistics back before the split (and I haven't looked at the statistics since), and I saw a number like 10,000 CD-ROMs sold somewhere. Not all of those are active users, I'm sue, but that would put some kind of lower limit on the number.

***Linux Journal*:** Hindsight being 20/20, do you occasionally wish you had patented, or otherwise retained rights to Linux?

Linus: Definitely not. Even with 20/20 hindsight, I consider the linux copyright to be one of the very best design decisions I ever did, along with accepting code that was copyrighted by other holders (under the same copyright conditions, of course). I'm not fanatic about the GPL, but in the case of linux it has certainly worked out well enough. As to patents, I consider software patents a patently bad idea in the first place, and even if I didn't, I would abhor the paperwork needed. Getting a trade-mark on the name “linux” might be a good idea, and there was some talk about that, but nobody really found the thing important enough to bother about (especially as it does require both some funds and work).

***Linux Journal*:** What is your field of study, and what do you plan to specialize in upon graduation?

Linus: I'm studying mostly operating systems (surprise, surprise), and compiler design: rather low-level stuff mostly. I expect I'll expand that to communications and distributed systems for obvious reasons, but I haven't really decided on anything yet. So far, my “field” has been any courses that I find interesting, and I hope I won't have to specialize any more than that in the future either.

***Linux Journal*:** Linux is benefiting from a worldwide development effort. The number and frequency of new releases of Linux, and drivers and utilities are amazing to anyone familiar with traditional UNIX development cycles. This seems to be giving Linux a huge “competitive advantage” over alternate UNIX-on-the-pc products.

What do you see as the future of Linux?

Linus: I rather expect it to remain reasonably close to what it looks like now: the releases may become a bit less frequent as it all stabilizes, but that might just mean that I'll make my snapshots weekly instead of daily as I do now during intense development, and that the “real” releases will happen a couple of times a year instead of monthly or bi-monthly as now.

Similarly, there will probably remain several different “package releases”: some of them will be more or less commercial (currently the Yggdrasil CD-ROM, for example, or the various disk copying services), while others will continue to be mostly electronically distributed by ftp.

***Linux Journal*:** What would you LIKE to see for the future of Linux?

Linus: Related to the question above, I do hope to see one change: support and documentation. Some of this has actually already happened or is happening now, but there is still room for growth. I know of a few book projects (one of which went into print a couple of days ago), and a few support companies, and I hope that will still grow.

Then there are various interesting projects going on that I'd be very interested to see:

Windows emulation (being worked on, and the kernel support is already there);i386 SysV binary compatibility (already in early stages of testing) etc.;As well as the porting projects to various different hardware platforms, of course.

I also have various general (and vague) plans about actual kernel development, and some specifics I want to have implemented in the reasonably near future (I think I'll work mostly on memory management and related areas this spring, for example). Mostly, I just hope to have a stable and enjoyable platform

***Linux Journal*:** Also, would you have a photo of yourself we could use to accompany the article? This is by no means required, but a huge number of Linux users are very curious about who you are, why you did Linux, etc... you know, all the human interest side to the Linux story.

Linus: I'm “ camera-shy”, so I have no good photos for this purpose, which has resulted in some rather weird photos being used in some places. A magazine in Holland used one of the gifs that were put out long ago (bad quality, and very much done in jest: I drink beer in most of them, including the one they used), and one Finnish magazine used a photo from a party I was at which also had lots of beer-cans in it.. I guess I should find some rather more presentable photos somewhere. I'll see.

***Linux Journal*:** We saw a photo that was distributed over the net. One that has you smiling, with a beer bottle in front subtitled `Linus Torvalds - creator of Linux'—In fact, for all the `official' format for photos requires a tie and at least a semi-serious pose, I think that this was a VERY good photo, as it showed you as a happy, friendly human being.

Linus: It's another of the `party photos', although the party was a much smaller and more informal one. I don't know who has the originals anymore, so I'm unlikely to find it in time with most of the concerned people still being somewhere else as teaching at the university hasn't started yet. What the magazine from Holland did was actually to have a screen-shot of linux running X, and have the gif-picture in an xv window (with a few other windows like xload to give it some more lf); that way the quality of the picture didn't matter much, and it also looked like a clever idea. You could use some similar trick. I don't mind looking like a human being instead of a tie+shirt robot, so I don't mind the picture even though it was all done mostly in jest.

***Linux Journal*:** We'd like to send you a complimentary subscription(s) to *Linux Journal*.

Linus: I'd like a copy, please.

***Linux Journal*:** Also, re your response on the `other platforms' question, if you can find someone willing to do the work, we should be able to help find someone at IBM or HP (maybe even DEC, but I doubt SUN) who would be able to donate/loan some hardware.

Linus: It would be fun, but as I can't make any promises and would need lots of technical documentation as well (and not under any non-disclosure), this is probably not really something companies like to do.

***Linux Journal*:** Where did you learn to write English this well?

Linus: I read more English than either Swedish (my mother tongue) or Finnish (which is the majority language in Finland, of course), so I while I'm not completely comfortable actually speaking the language (partly due to pronunciation), I don't have any problems reading, writing or indeed thinking in English.

The reason for reading English is simply that there are more interesting books available in English, and that they are usually cheaper even over here (larger printings, no translation costs, etc.). Besides, it's often the original language, so even when the book is available as a translation, I usually prefer to read it in English.

This interview was conducted by Robert Young, Publisher of the *Linux Journal*, NY Unix.

Archive Index Issue Table of Contents

Advanced search

Onyx

Michael Kraehe

Issue #1, March 1994

We continue to see database applications being developed in fourth generation languages (4GLs), and we continue to see more and more sophisticated (and expensive) 4GLs. Onyx is on the other end of the spectrum, offering a 4GL that takes advantage of the tools built into the operating environment under which it runs. This effort also shows that it is possible to earn a living working with free software.

Onyx is a tool for rapid prototyping of database applications. The theory behind Onyx is that any application consists of the following:

- Internal tables to store temporary data
- Masks to view and edit the date
- Transactions between the internal tables and the rest of the world

Transactions can be bound to a menu, an input field, a function key or to the change of the current row of the cursor. So if the user selects a menu item or leaves an input field by pressing return the transaction will automatically be started.

Unlike other 4GLs, Onyx doesn't hide your environment (UNIX/Linux) but gives you the full functionality of the shell. For example, this is how we could print letters using data in an Onyx table combined with the capabilities of shell commands.

```
export table to pipe {
        awk | mailmerge | groff -Tps -mm | lpr -Pps; }
```

It is also possible to import a table using shell commands as in the example below:

```
import table from pipe {
        tar xvf /dev/rmt0 | sed "s/ *-> */->/" | sed "s/  */     /g"
        | sed "s/[^]*$/&  &/"; }
```

And, of course, it is possible to export or import to an SQL statement.

Onyx needs a SQL database which is doing the real work. So its possible to use a SQL anywhere in your Net. I'm normally using Informix on a Sparc II or Ingres at home under Linux.

Databases are named as **database%engine@host.domain**. Supported engines are currently GNU-Awk, Informix, Ingres and SHQL.

The last one (SHQL) is a genius shell-hack which is parsing the SQL by the shell and doing the job by awk. Of course that's really slow. It is included because that is where my idea for the Onyx project came from. Also, you need SHQL because it is used to manage the schema definition (create table, etc) for gawk.

Current System

Why did I write Onyx?

I started looking at Linux in late 91 and since MCC 95c+ I've thown away Xenix and am using Linux at home. As TCP/IP got into 98p1 kernel I thought about using Linux as a client for databases.

In winter 92 I began a project for the ”Wasserwirtschaftsamt Bremen“ called ”Verwaltung von Anlagen mit Wassergefaehrdenden Stoffen“. They had a Siemens MX 500 which was so slow that it was possible to roll a cigarette and smoke it to the end when somebody wants to do a query.

I told them that ”Client/Server computing is the only solution" and they installed my suggested hardware consisting of a Sparc II and 8 PCs. Having Linux in mind I gave the PCs 8 MB of RAM and a Tseng video card. This was the right idea as I saw later.

Informix SE and NET for the Sparc were delivered together with Informix for MS-DOS. The port of the old database and program was only a job of character conversation between DIN to MS-DOS. (In Germany we always have a umlaute problem.)

The hardware was installed as described above. It was fast enough to do the querys in just a second. Unfortunately, the MS-DOS systems were crashing nearly every hour! We didn't know if it's MS-DOS, PC-NFS, or Informix that was causing the problem. But we knew it was the time to look for a stable operating system. Of course Linux was my choice!

As an interim measure, the MS-DOS systems were used as terminals to telnet into the Sparc and jobs where run on the Sparc. The query times grew to up to half a minute and and I started to hack a better solution using Linux.

Two month later I installed the first version with Linux 99p6. It's now running 99p9. And I now have earned enough money (6,000 DM) to live for the next two years and I'm now rewriting this quick hack that I did in the first month. As there are some more Wasserwirtschaftsaemter (administration of installations with water endangering substances) for the Department of the Environment of the State of Bremen having interest in the program, I will continue the project earning more money and giving the Onyx-4GL to the Internet community. Onyx a killer application for Linux? Large databases are a place where the money is made. So let's talk about money. As I saw it, MS-DOS is bad choice as a possible platform for a database clients. I see two possible solutions:

- Use a Mainframe for doing the job
- Use Sparc or $CO Unix systems as clients

Both are really expensive choices. With Linux you only need a cheap PC to do the job in client/server computing and you don't have to pay a lot of money to a Unix vendor for every client.

Another possible scenario is needing to upsize an entry-level database from an MS-DOS platform. Let's think about the following configuration

- First a 486-bases system with Linux and [university] Ingres as a server
- If there are old 286 systems they can be used as terminals
- Newer 86 can be upgraded to run the client locally
- As the database is growing up you can use a faster (commercial) SQL on a commercial Unix as a server without trouble
- DOS applications can be run by QuarterDeck DeskView-X without the posibilty of crashing the database in an xterm on any Linux box, if
- DOS applications are really necessary

Such a configuration can be sold to any company and it cost less than a Novell Netware. And if the company has several locations in perhaps different cities its possible to mirror the transactions on normal modem lines which is impossible to do with a traditional database over a remote Novell FS.

Future Plans

As I've written above I'm now rewriting the whole program, and I hope to finish this in December. Look at the Usenet group comp.os.linux.announce for my posting.

The new version will include the following features :

- Rewritten by using an Object Oriented Parser Generator
- Include awk & ingres as local servers for Linux
- Backup store and forward as a model for distributed transactions
- Some example applications
- Hopefully some documentation (Real programmers don't write docs, they write programs. The code was hard to write it should be hard to read.) But, I've expanded the parser generator to produce manual pages for each parser it produces. That's a start.
- I'm considering adding a RFC on my transaction monitor so I can find out what features should be added to the rd version

Where to get it?

If you are in Germany, Onyx can be found on the ftp sites wowbagger.pc-labor.uni-bremen.de or olis.north.de. Onyx has also been posted to alt.sources on Usenet. Otherwise, your best bet is to use archive to locate a copy.

Michael Kraehe lives in the back woods of Germany and names the machines in his network as would an old anarchist so will find machines with names like durruti and kropotkin. His ideal computer would be a palmtop with a 17" display, Ethernet and SCSI so he could pack it on his 12 year old SR500 motorbike along with his dog.

Archive Index Issue Table of Contents

Advanced search

Linux Code Freeze

Linus Torvalds

Issue #1, March 1994

This is a general announcement of the imminent code-freeze that will hopefully make Linux 1.0 a reality. The plan has been discussed a bit with various developers already, and is already late, but is still in effect otherwise.

In short, the next version of Linux (0.99.15) will be a “full-featured” release, and only obvious bug-fixes to existing features will be applied before calling it 1.0. If this means that your favourite feature or networking version won't make it, don't despair: there is life even after beta (and it's probably not worth mailing me about it any more: I've seen quite a few favourite features already ;-).

In fact, 1.0 has little “real meaning”, as far as development goes, but should be taken as an indication that it can be used for real work (which has been true for some time, depending on your definition of “real work”). Development won't stop or even slow down: some of it has even been shelved pending a 1.0 already.

Calling it 1.0 will not necessarily make all bugs go away (quite the opposite, judging by some other programs), but I hope it will be a reasonably stable release. In order to accomplish this, the code-freeze after 0.99.15 will be about a month, and I hope people will test out that kernel heavily, instead of waiting for “the real release” so that any potential bugs can be found and fixed.

As to where we are now: as of this moment, the latest release is the `r' version of pl14 (aka “ALPHA-pl14r”). I've made ALPHA releases available on ftp.funet.fi almost daily, and expect a final pl15 within a few more days. Testing out the ALPHA releases is not discouraged either if you like recompiling kernels every day or two..

And finally: we also try to create a “credits” file that mentions the developers of the kernel and essential Linux utilities. The credit file compilator is

jmartin@opus.starlab.csc.com (John A. Martin), and if you feel you have cause to be mentioned in it, please contact him.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

ICMAKE Part 1

Frank B. Brokken

K. Kubat

Issue #1, March 1994

In this first part of a three part article, Frank and Karel explore their motivation for writing a new programming “make tool”, the organization of the system itself and where you can get a copy. Part 2 of the article will cover the icmake grammer and the final part will show examples of its use.

Introduction

Icmake was developed initially as a make-tool to be used under MS-DOS. Although make-utilities under MS-DOS exist, we tended not to use these utilities for a two main reasons:

1. Since our primary language is C (C++), we considered the grammar of the make utilities awkward.
2. In a program development cycle in which sources are compiled, included in a library, which is then linked to a main object module we experienced problems in setting up the correct dependencies between files.

Once icmake was developed, it was soon thereafter ported to UNIX, where it now not only serves as our make-utility of choice. However, by now it also serves as a utility with which we find it easy to write shell-scripts.

Currently icmake has been ported to several UNIX platforms, such as LINUX, SCO-UNIX, HP-UX, and SunOS. Icmake, therefore, reflects almost perfectly the transition from DOS to UNIX we experienced ourselves: developed initially under MS-DOS, it has now become primarily a UNIX tool. But then, we used the `tools of the trade', already under MS-DOS: both bison and flex were used for the construction of the icmake compiler icm-comp. The first version of icmake was available after some nine days, including the decisions we had to take about what form it should take. Since we were also working with some UNIX platforms (SunOS and AIX) by that time, the `porting daemon' started to

influence our implementation descisions as well. The first port of icmake to a unix operatingsystem took only a few days, which time was mainly invested in porting some specific MS-DOS functions to an acceptable UNIX form.

In this article we describe icmake from the point of view of its users. The organization of the software, the grammar of icmake files, icmake-scripts and - to start with-the way to obtain icmake is described in the remainder of this article. At the end some illustrative examples of icmake-scripts are given.

2. Obtaining Icmake

Icmake can be obtained by anonymous ftp from the site beatrix.icce.rug.nl, where it is found in the directory **pub/unix**.

The package consists of an archive, usually in the form icmake-X.XX.tar.gz, where the current version of icmake is denoted by X.XX. This archive contains a tarred and gzipped directory structure, in which the source files for icmake (and the executable programs for MS-DOS) can be found. Also at beatrix.icce.rug.nl the file icmake.doc can be be found, which is a guide to the installation of icmake. This latter file is especially useful for UNIX installations.

Alternatively, icmake can also be found at tsx-11.mit.edu, where it is part of the Linux Operating System distribution. At tsx-11.mit.edu the icmake-files are usually located in pub/linux/sources/usr.bin.

Apart from the source-distribution and the installation guide there is a icmake.dvi file available at beatrix.icce.rug.nl describing icmake in somewhat more detail than is possible in this article. Also, a postscript version (icmake.ps) of the documentation is available.

The Organization of the Software.

Icmake consists of five programs. One program is a nitor program, monitoring the construction and execution of an icmake file. The monitor program may call an icmake preprocessor, an icmake compiler and an icmake executor. A fifth program is the icmake unassembler, showing the code generated by the icmake compiler in a human-readable form.

The programs can be divided in two categories: programs of which the ordinary user of icmake must be aware (the top-level programs) and programs which are called internally by icmake (the nested-level programs). The top-level programs are the icmake-monitor program and the icmake unassembler. Normally, only the monitor program is started at the command line, but programs of the nested-level are sometimes called explicitly. All icmake programs can be started as stand-alone programs.

The normal processing of an icmake script runs as follows:

1. An icmake source script is written, containing a description of the tasks to be performed by icmake. This script is very C-like.
2. The monitor is started, receiving the name of the file containing the icmake source script as its argument.
3. The monitor program may call the icmake preprocessor and the icmake compiler to convert the source file in a binary file to be processed by the icmake executor.
4. The icmake executor is started by the monitor. The executor reads the compiled icmake script, executing the instructions found in the compiled script.

Once an icmake-script has beenompiled, the compilation and preprocessing phases are skipped by the monitor program, and the icmake executor is called immediately. In this way icmake monitors its own dependency between the icmake source script and the compiled icmake script.

Each of the programs of the icmake family has a version number. The version numbers consist of a major and a minor number. E.g., in the version number 6.03, the major version is 6 and the minor version is 03. The programs can only communicate when the major version numbers of all icmake programs are equal. So a working set of programs must all have the same major version number. The minor number is used to indicate small changes in the separate programs.

3.1. The top-level programs.

At the top-level of icmake two programs are found. icmake itself and icmun. The program `icmake' is the monitor program, monitoring the execution of anicmake-script. The program `icmun' is an unassembler, showing the contents of the compiled icmake script in a human-readable form. Normally, icmun is not used: it serves mainly as a debugging tool (although we've used icmun also for instructional purposes, showing people some basic notions of assembly-language programming).

Icmake

The icmake program monitors the execution of an icmake script, and acts therefore as an interface between the user and nested-level icmake programs. Normally, the nested-level icmake programs are not started explicitly at the command-line, but are called by icmake itself.

Icmake may be invoked unsing one of the two following command lines:

icmake *[flags] asciifile [binaryfile] [arguments]*

or

icmake *[flags] -i asciifile [arguments]*

The two invocation modes differ in the fact that the first invocation (without the **i** flag) makes it possible to specify explicitly the name of the compiled icmake file (the “binaryfile”). The second mode uses a predefined name for the compiled icmake file. In this form the name of the asciifile is used, having the extension “.bim”.

With both invocation modes the “flags”, the two consecutive hyphens and the arguments are optional. The arguments are passed to the makefile and can be inspected there. The flags are used for requesting specific actions of icmake. The specification of the asciifile is obligatory in both invocations: this file is the make script, which is interpreted by icmake and defines the actions to be performed.

Whenever program from the icmake-family is started without arguments, a help summary is displayed.

When icmake is activated without any arguments, the help shows among other things the flags which are recognized by icmake. In that case, something like the following table below will appear on the screen:

The following flags are recognized:

-b. When this flag is used, no recency test is performed. Rather, the binary make file is executed immediately `as is'. When this flag is absent, icmake compiles the input file (.im file) into a binary file (.bim file) if the input file is more recent.

-c. When this flag is used, the ascii input file is compiled to a binary makefile, but is not executed. The compilation occurs irrespective to the recency of the input file and the binary file.

-i file. This option specifies file as the input file for icmake and stops the further processing of the arguments. In this case the nam of the binary makefile (.bim file) is equal to the name of the input file receiving the extension .bim. The default input file extension, .im, is not supplied by icmake when this flag is given: the specified name is taken literally.

-p. When this option is used, only the icmake preprocessor is activated. The output of the preprocessor is sent to a file having the same name as the input file, but having the extension .pim. When this flag is absent, icmake's preprocessor still generates a .pim file. However, this file is deleted after the compilation-phase.

-q. When this option is used, icmake operates quietly. I.e., the copyright banner is not displayed.

--. By default, the first non-flag argument on the command line is the input file, for which the extension .im is assumed. When a second non-flag argument is given, then this name is used for the binary file (.bim file). The processing of arguments stops only when two consecutive hyphens are encountered. All arguments which follow the—flag can be inspected by the icmake-file.

Note that the—flag is not required when icmake is activated with the -i flag.

The asciifile specification is obligatory. This is the icmake source file which will be compiled and tested by icmake. Icmake assumes a default extension **.im**. E.g., the command line:

icmake test will activate icmake to process the icmake sourcefile test.im. Note however that the command line:

icmake -i test will start icmake to process the makefile test. The presence of the -i flag prevents icmake from supplying an extension.

The binaryfile specification is optional. When given, icmake uses this file as the binary intermediate file in the process of making. Extension **.bim** is the default. By default icmake uses the basename of the asciifile and extension **.bim**.

Following the binaryfile specification, several arguments to the makefile itself may be given. However, before any extra arguments are specified two consecutive hyphens are needed. Following these hyphens extra arguments may follow which will be passed to the icmake dependent programs. As described above, the delimiting two hyphens are not necessary when the **-i** flag is used.

The icmake specification file is written as a C program, and contains a *main()* function which receives some of the arguments specified on the command line. The first argument is always the name of the binairy make file (normally having the .bim extension). Remaining arguments are the arguments that follow the two hyphens. The hyphens themselves are not included in the series of

arguments which are passed to *main()* (The user-defined function *main()* is described below).

Icmun

Icmun is mainly used in developing icmake. The icmun program is an unassembler for the compiled icmake file created by icmake. Icmun produces an assembly-like listing of the instructions contained in the compiled makefile. Normally it is hardly ever used. More information about icmun is beyond the scope of this article, but can be found in the icmake documentation, which may be obtained from, e.g., beatrix.icce.rug.nl.

The nested-level programs.

The following programs are icmake's `nested-level' programs. Normally these programs are not started at the command-line, but are started as child- or overlay-processes of icmake itself.

The preprocessor icm-pp.

Icm-pp is the preprocessor of the icmake compiler (described in the next section). This program scans the icmake source file for preprocessor directives (e.g., **#include**, see the section on the preprocessor directives below). and takes appropriate actions when they are encountered. Icm-pp writes an output file in which the preprocessor directives have been `expanded'. This (temporary) file is used thereupon by the icmake compiler.

The compiler icm-comp.

The stage following the preprocessing is the compilation. The icmake compiler is called icm-comp, and this program translates an icmake source file, generated by icm-pp, into a binary format and performs error checking. The resulting binary file contains opcodes, much like the output file of a compiler of a programming language. When a binary makefile is generated, the intermediate output file of the preprocessor is no longer needed and is removed.

The executor icm-exec.

Icm-exec is the executor of the binary makefile generated by icm-comp. In this phase the compiled icmake file is read. It contains opcodes which are interpreted, thus executing the commands defined in the icmake source file: lists of files are built, files are compared by dates, and other actions may be taken.

Executable makefiles under Linux (Unix)

Under the Linux (UNIX) operating system it is possible to create makefiles for icmake which are themselves executable. As an example, this could result in a setup in which an icmakefile “backup” exists, which can simply be started as:

backup

In such a setup the same effect may be achieved by creating an icmakefile backup.im, which is then started with the command:

icmake backup

The former form is certainly simpler to use, and requires an executable icmake script called `backup'. To create an executable makefile, the following steps should be taken:

1. The makefile can be given a suitable name. This name is later used to invoke the process which executes the makefile. For this reason, the name backup may be preferred to backup.im.
2. The makefile is made executable using the UNIX chmod program: **chmod +x backup**

   This labels the file backup as an executable program. Furthermore, the makefile may be placed in a directory pointed to by the PATH environment. E.g., the makefile may be placed in a user's private bin directory.
3. The following line is added as the first line in the icmake source file:

   ```
   #!/usr/local/bin/icmake -qi
   ```

This line informs the operating system that, when executing this file, the program **/usr/local/bin/icmake** should be started, using the flags **-qi** and then appending the name of the icmake file. As discussed before, the flag **-q** suppresses icmake's copyright banner, while the flag **-i** causes icmake to take the following file argument as the literal input file instead of supplying the default extension **.im**.

Note that this line must contain a full reference, including the path, to the program icmake. In this example this path is taken to be **/usr/local/bin**.

The compiled icmake file (**.bim** file) is placed by icmake in the same directory as the icmake source This directory must therefore be accessible to the user invoking the makefile. If the icmake script and the associated .bim file is to be accessible system-wide, the icmake script could be installed by root. Using the

example of the backup script again, root could install the “backup” script using the following command in the directory where the “backup” script is stored:

icmake -c backup

This will merely result in a compilation of “backup”, thus generating **backup.bim**.

Frank B. Brokken, ICCE, Department of Education Universtiy of Groningen. Groningen, the Netherlands.

K. Kubat, ICCE, Department of Education Universtiy of Groningen. Groningen, the Netherlands.

Archive Index Issue Table of Contents

Advanced search

LJ 1: Linux and Hams

Phil Hughes

Issue #1, March 1994

A couple of weeks ago I posted the following to the Usenet newsgroup comp.os.linux.misc. I am the editor of *Linux Journal*, a paper magazine that will be covering the Linux scene. In my correspondence with people about writing articles for *LJ* I have seen an amazing number of ham calls. Being a ham myself (WA6SWR) I was just wondering how many of "us" are hams.

If you are a ham, let me know and maybe include a short blurb about how/why you got involved in Linux. I think it could be interesting. I will post a summary and might even include the info in an upcoming issue of *Linux Journal*.

Rather than try to draw any conclusions I have decided to just post what people said. I think hams will find it interesting. Note that in the ham tradition I edited the signatures so they consist of the person's first name and ham call unless they didn't give me a call. Then I include the full name.

Assuming there are no objections (and enough space) I plan to reprint this in the March issue of *Linux Journal*. I think it helps tie Linux into another community. One who has traditionally been on the leading edge of both technology and spending as little money as possible. :-)

Any hams out there doing "ham stuff" with Linux and want to write an article about it for *Linux Journal*?

The SLS/slackware distributions have basically given me a UNIX/X workstation at home...wish my 486 was as fast as the Indigo on campus though.

I've been using Linux for over a year on a Gateway. I just now have some time to try hacking on some programs for the radio.

Most of the time I'm on 80/20 meter PACTOR. I almost have a PACSAT station working (still need a preamp and a few other odds-and-ends).

I think Linux would work great for automating satellite uploads and downloads, tracking, etc.**-Phil AD4FH**

Well, this comes to you from host w4bzl.concert.net at the end of a quite productive SLIP link. Of course this host at home runs linux. AT CONCERT we have at least two other hams that run linux with their calls as their hostname. I'll leave it to them to respond.**-Joe W4BZL**

I'm a ham too, N5SNN down in Austin, TX. How I got involved? Well that's because our lab has too many junks (or goodies in ham word), that made me just buy a used 486 motherboard and a new 486sx/25 to start a unix-pc using Linux. Why? To replace the NOS-box that I use for ausgw (austin's packet-internet gateway) with a better system.

My linux is now 100% on the Internet (it's under my lab desk at campus), runs gopher server, anon ftp, and uucp-internet gateway (private).**-Paulus N5SNN**

Add me in there too. BTW, I started using Linux about 6 months before becoming a HAM. Part of the reason for becoming a HAM was the large number of them involved with Linux.**-Jim KD4PPG**

I am VK4BSB. Packet mail to VK4BSB@VK4ZGQ.BNE.QLD.AUS.OC**-Serge VK4BSB**

Who, ME? I've been a netnews reader since the Great Renaming. I decided it was time to start working with the `blood & guts' of a Unix.....**-David WB8FOZ**

I got interested in Linux last year when Algorithm magazine published a quick blurb about the freely distributed Unix clone and how great it was. I wasn't able to start playing with it until this past August when I upgraded my old XT to a 486. Unfortunately, that also corresponded to my starting a PhD program, so I haven't had much time to play with Linux, outside of installing SLS 1.03 and wondering how to patch what doesn't work!**-Chris N0OQT**

I am: Jim Graham, N5IAL /4. I've been a general-class since `86. I live on Okaloosa Island, in Ft. Walton Beach, FL (NW FL). I started looking at Linux in mid-1992 as part of my (then) ongoing search for a UNIX that would run (read, would actually work) on my 386 at home. I finally got a tape drive, which made the installation much simpler, and a memory upgrade to make Linux happier, and installed 0.99 PL6 very early this year. Needless to say, my search for a UNIX system for my machine has ended.

Oh, I'm also the author of KAMterm, which is a shareware host mode terminal program for Kantronics TNCs. It currently runs only under dos (and some dos emulators, I'm told), but I'm working on porting it over to Linux (don't hold your breath—due to all of the things that are going on right now, and the fact that

very few people have contacted me about KAMterm in the last couple of months, I'm not spending a whole lot of time on it at the moment...and the port wouldn't be a quick thing to do even if I were). Of course, there are other issues involved with a Linux port, such as the fact that KAMterm is shareware, not freeware, and so on (I need the KAMterm money to pay bills).**-Jim N5IAL/4**

I use Linux because I want to run a Unix system at home. I'm one of those `sick' people who like the punishment that Unix systems hurl onto their operators. I hope to working with packet radio, as I am a avid packeteer. I understand that there is a version of NOS out for Linux, but have not found it yet. Know where? I also know KB9CTJ here running it too. Were working together on this. I hate MSDOS too.**- Chuck WO9K**

I use SunOS 4.1.3 at work and don't particularly fancy DOS or DOS/Windows. Linux is an excellent OS for learning how to admin Unix, as the SLS distribution comes complete with lots of configuration problems. That is the configuration for lots of the programs provided with SLS are incomplete or wrong, but close enough that it is better, and more realistic, than course examples. I will be almost upset if the next distribution has all of the problems fixed, as requested by so many flames, because I would have chosen a fixed version had I the opportunity and not learned nearly as much.

All the applications that I need for home are available on Linux and the VC's are great. 386sx/16 with wait states *really* bogs running X. I am waiting for something in c.o.l.a to brag about an excellent TNC interface prog/doc set before I buy one and learn about the real-ether network.**-Alan VE3ALO**

FWIW, I'm not a ham, although I am studying to become one next year. At this moment I'm using the CB for packet radio. [Don't panic - Peter is in the Netherlands where apparently you can do anything with CB as long as the radio is type approved.]

At this moment Linux is one of the most promising systems for hams. At this moment several versions of NOS have been ported to Linux, like Wampes and JNOS.

The big advantage is that these versions don't suffer from severe lack of memory like most of the MS-DOS versions. And for the future, Linux will have the AX.25 protocol in the kernel. The normal UNIX socket interface will allow easy program development. It is also rumored that FBB will be ported to Linux when the AX.25 support is built in.**-Groetjes, Peter Busser**

I am WA8USA. I use LINUX on a laptop I have for work (image and digital signal processing). I went to LINUX because of the large number of public domain

programs that are included in distributions (or are easily built) and because the availability of source is helpful in solving problems.**-Bob WA8USA**

I am a ham (ve3ich) and am active in Linux, but the reasons have little to do with amateur radio. I suspect that the type of person who is interested in amateur radio would likely be interested in Linux: doing something technical that is new and state-of-the-art at a low-cost.

I've used Unix at work for a number of years, and had various microcomputers at home. It was a dream of mine to be able to afford a machine at home that had the hardware and software capabilities of a Unix workstation. Just over a year ago I bought a PC specifically for the purpose of running Linux. I learned more about Unix in the first 3 months of using Linux that in 5 years of experience with commercial Unix workstations. I now find myself wishing that the $10,000 Sun workstation that I use at work had more of the capabilities of the $1,800 Linux PC at home.**-Jeff VE3ICH**

Well, there's my wife N2VIS (Carolyn Carrock) and me N2RDI... I'm more a Unix admin geek type that got into Amateur Radio than a ham that got into Unix... My wife dared me to take the test at the Trenton Computer Festival after hearing me say I could pass it... If you're looking for writers—I was a Unix instructor for Pyramid Technology and a former newspaper editor and reporter... I'm looking forward to subscribing... and reading every article in *LJ*. My last project—idasendmail on Linux with pathalias routing. My next project - 99.14 kernel and making a linux SVR4 lookalike with all new scripts and the /sbin directory structure.**-Bill N2RDI**

My call is KF6VB, I've been a ham since 1970. Ham radio got me into electronics, electronics got me into software, software got me into a nice career . But it all started with ham radio.

I got into Linux about a year ago, around version 95pl7, I think. Linux offered me a way to learn about unix without spending big $$$. Also, it allowed me to upgrade my home UUCP node to a much more powerful environment than what DOS supported.

Four years ago, I got my jollies on the radio, talking to people all over the world. Now propagation is in the toilet, and I get the same jollies with uucp, email, and News.

I've also been a student of the Russian language most of my life, and in the past few years I took two trips to Moscow. The first one was pre-Linux , in 1988. Then I visited radio amateurs. I have pictures of Yours Truly at the operating console of the Moscow Radio Club.

The second trip was last December; no radio buddies this time, just people I met through the email. I took the complete latest SLS distribution with me, and seeded Moscow with Linux! At that time, people there had heard of it, but never seen it. Even while I was there, I could see it seading exponentially: people borrowed my diskettes, copied them and gave them to other people, who copied them....

Right now, I am trying to get the Ka9q networking program running on my 99pl9 system. If I am successful, I will be able to integrate the amateur and landline email worlds at one console. So if I say “mail wa6xxx@ampr.org” the system would automatically squirt it out the TNC instead of the phone line. Also, since the Linux box runs 24 hours a day, I would be able to offer routing to my friends, run an on-the-air ftp site, etc, etc....

I too, have noticed the multitudes of hams on the linux groups; what's more, I have seen more and more mention of Linux on the ham groups. In my mind, hams have two main characteristics:

- They like to communicate
- They like to play with gadgets.

A Linux box is both a powerful communications platform and a really neat gadget! What ham could resist?**-Jerry KF6VB**

1 more. And soon to be a real US HAM (N5something)--I passed the Tech exam some time ago. : If you are a ham, let me know and maybe include a short blurb : about how/why you got involved in Linux. I think it could be : interesting. I will post a summary and might even include the : info in an upcoming issue of *Linux Journal*. I am also doing an AX.25/NETROM layer in NET-2.**-Fred N. van Kempen**

I'm a radio amateur, and a LINUX user. My computer is not (yet) connected to packet radio, but once I get around to buy my own rig, instead of just using the radio club's (SK5EU) gear, it sure will be. Sadly, TCP/IP on packet seems to be virtually unknown here in Sweden..**-Ture SM5UUO**

Phil, I'm N4HHE and saw your request to hear from Linux Hams in the mail-archive on tsx-11.mit.edu. Don't know what kind of information you are looking for. I use Linux for “recreational computing.” When I work (as an electrical engineer) my choice is Macintosh. Been Unix'ing for over 5 years, usually as the system administrator, and always made sure I had a Macintosh that I could use for login. A couple of years ago I “ran” a 12 machine Silicon Graphics installation from a 512k Mac. Lots of fun.

Currently experimenting with TCP/IP over amateur radio. WAMPES has been running at my home for the last couple of weeks. See some people gripe about the lack of Linux documentation, they need to try WAMPES and really learn about lack of documentation.**-David N4HHE**

I'm a ham and I have used Linux since May 1992 I think. I just can't remember where I heard of Linux first, but I followed the development of Linux from November 1991 (or maybe December). First by listening to reports by others, from 23. January I started to `finger' Linus' account to get the latest news about the state of Linux. As I said above, in May 1992 (or was it April? Hmm, not easy to remember.. :-) I started to actually use it, after having followed the mailing list for some weeks, enough to feel confident about how to install it and so on. I'm not using Linux for any ham-related stuff, one of the reasons is that I don't have a PC at home (I'm using Linux in my job only).**-Tor LA1RHA**

N3PFP. I'm essentially inactive, I only bothered to get license because I was planning on trying out packet radio. I passed the written tests up to advanced before I ran out of testing/checking time that day, but I've never done the code tests (and probably couldn't break 10wpm at my peak). ... still looking for a cheap, short-range 56k-1Mbs packet radio setup.**-Donald N3PFP**

I am a ham, WB4ARV, but I only use linux at work. I at least have been working on it when I have time. I have been pulled to another project and have been unable to work on it for the last couple of months. I also have been working on the depca cards for the ethernet so that has slowed me down alot. If I ever get it working good I think I will try to use it at home. I am not into packet at the moment but I may if I switch to Linux.**-Joe WB4ARV**

Alright, I'll bite! ;) While in college, I started to dream of having my very own “unix box” to hack on for endless hours. The thought of being able to do this *and* get to participate in the evolution of a new OS was really exciting to me, and remains so to this day, a couple years later!

There is absolutely no doubt in my mind that I have been able to learn *so* much more about unix and PCs in general because of the decision to run Linux on my own computer. I have been able to contribute to portions of the OS in 1992 and early 1993, in the form of initial ports of network clients, a BBS package, and several X Windows games. I also established a free Linux-oriented BBS in late 1992 which continues to flourish and present new learning opportunities.

Ham Radio entered the equation this year, when I decided to once again obtain a license, this time Technician. I operated CW as a novice at the tende r age of nine years - fifth grade, and now enjoy 2 meter rag-chewing, packet radio and

maybe, CW. Someday. When I get around to it again. ;) A small group of experimenters in the Bay Area are architecting a high speed packet network. It uses 256 Kbps modems designed by a fellow at Stanford, and protocols by a friend (Cliff Skolnick N1DPH) and myself. Mike Cheponis (K3MC) is the mastermind behind the project. A mailing list is set up on my machine (list is called “speed-freaks” - send mail to listserv@hip-hop.sbay.org with “help” in the body for info).

It's coming along, albeit slowly, and we're experimenting with some low-speed stuff right now. My Linux system talks to a 286 DOS box (running KA9Q) over ethernet, and from there out onto the local TCP/IP network; the 286 functions as a router, and allows access to network services on my Linux box from the ampr-net. Cliff and I are eagerly awaiting the release of the Linux AX25 drivers to try out and possibly integrate into our network software.

Hope you enjoyed reading this response to your post. I look forward tohearing other Linux-using ham operators' stories as well!**-Dave KE6AJC**

Actually, the number of technical “netters” in *any* field who are hams is rather amazing.

++Brandon (JNOS Linux port, and if I ever get time I'll finish up some enhancements to XSat that make it almost worth using)

--... ...-- -.. . -.- ..-. ---.. -. ....-.-

[in English, 73 de KF8NH—boy is morse code hard to read like that] 73 de KF8NH

Hi. I am also a ham (WQ3S). I recently started running Linux because I wanted a REAL multitasking OS for my computer. I was already familiar with UNIX, so I found Linux to be an ideal choice for my needs.**-Andrew WQ3S**

I got into Linux because I wanted a real Unix and it was free, and nowadays its free and better. I'm currently running a public access Linux box via AX.25 and WAMPES (much hacked).**- Alan Cox**

Well, count me in. I've been keen on Unix from the git-go, but it always seemed to be out of my reach for hobby use until Coherent. When Linux reached ~pl12, I jumped in; never looked back.

One of these days RSN, I hope to start using Linux for some ham applications, but that's somewhere down in the queue. OTOH, it seems that I'm having so much fun dinking with Linux that the ham gear is gathering quite a coat of dust.**-John N4VU**

I am N2VKD, just got my technician's license in June. Have been playing with Linux on my 386/20 for about a year now. Running packet radio here in New York City under Linux is one of the reasons I got my ticket. So, sometime in the coming year, I hope to get the radio gear and tnc that will best work with Linux. I joined AMSAT too, that big bird they are going to send up sounds exciting. Overall, I think Linux and Ham Radio will be great together, with both being the better from the merge.**-Jim N2VKD**

I've been involved with Linux since 0.12. I work in data networking, and am an active packet radio operator. I am co-administrator of the Sydney amprnet tcp/ip Wormhole gateway. I have a strong interest in computing too, and for me, Linux is ideal, allowing me to meld my radio and computing hobbies into a single series of projects.**-Terry VK2KTJ**

I am KD4UBM. I am eventually going to have a login: over the air via packet. It'll be a while till I get the money to buy the equipment, but I am setting up my Linux PC for it now.**-John KD4UBM**

My call is DL4YBG, name is Mark and QTH is Berlin/Germany. It was much fun working on a homebrew CP/M-system, because you could create your own system and built your own hardware. All this fun was gone when using MSDOS... LINUX with its complete sourcecode and kernel-hacking brought back the fun to computing...**-Mark DL4YBG**

Finally, my turn. I have been a ham for over 30 years and have always been on the experimenter end. Even build a remote base station when I was still in high school. I see Linux as another chance to experiment and hope to get Linux and ham packet plugged in together.**-Phil WA6SWR**

Archive Index Issue Table of Contents

Advanced search

Programming Tips...

Michael K. Johnson

Issue #1, March 1994

In this initial column, I'll explore porting programs from other Un*x versions to Linux. Porting Un*x applications to Linux is best done, as a general rule, by porting the application to some standard which Linux follows. This way, not only will Linux users benefit from your port, but so will users of other operating systems which follow the same standards. As always, there are some exceptions.

Broad Hints

When you write a new application, trying to make as much of the application as possible follow the POSIX standard will generally provide the highest level of portability. If an application "tracks," or follows the rules of, POSIX, it will compile without changes on almost every known version and clone of Un*x, and on Windows NT, OS/2 2.x, and recent versions of VMS, as well. If some system-dependent code must be written, it can be isolated and clearly marked, making it possible for the next person who ports the application to a new platform to port it much more easily. And programmers on non-POSIX-compliant platforms will probably know their platforms limitations, and know what to do to port your application to their platform.

Unfortunately, the current version of the POSIX standard is showing its age a little. To make a sufficiently powerful application, you generally need to use a more powerful standard, on which defines facilities missing in POSIX. In general, if your program compiles under System V (often referred to as SYSV) or BSD systems, it will probably compile without change under Linux. System V compatibility is enabled default in Linux, and I will show later in this article how to enable BSD compatibility. In general, the Linux libraries, combined with GCC, provide one of the easiest platforms available to port to. In fact, the greatest challenge in writing applications on Linux is to not use all the facilities available on Linux. To do so would make it difficult to port the application to other platforms, even though it would make the application easier to write originally

for Linux, because many of these rich facilities are not available on other platforms. Some say that Linux is a better platform to port to than to port from.

For the future, a specification called “Spec 1170” is being written, and all operating systems will have to follow it to be called “UNIXtm”. It is more powerful that most previous un*x specifications—powerful enough that common applications will not have to violate it to get real work done and will be widely followed, as most Un*x vendors have already pledged to support it. Spec1170 is being developed by X/Open, developers of the current XPG3 specification. Linux will eventually comply with most, if not all, of Spec 1170, but the developers will not commit to it without seeing the final standard.

Specific Issues

There are essentially three areas in Linux which cause problems porting, two of which are not avoidable, and the other of which it would be undesirable to avoid.

Conflicting standards

In most cases, it has been possible to make library functions compatible with both SYSV and BSD, but in some cases, standards simply conflict. When this happens, the POSIX behavior is chosen, if POSIX defines it. (If POSIX does not define it, the most reasonable behavior is chosen, or some other standard behavior is used.) For instance, the behavior of signals has changed from one version of unix to another.

Signals in early versions of un*x were unreliable. Whenever a signal was called, the signal handler was uninstalled, and so the first thing that most signal handlers did was reinstall themselves, leading to code like:

```
void sigfoo_handler(int foo) {
    signal(SIGFOO, sigfoo_handler); /* Rest of signal handler */
}
```

However, if another **SIGFOO** signal was received after the signal handler had been scheduled to run, and before **signal()** had been called, the default behavior for the signal would happen (perhaps it was just ignored, or perhaps the program was terminated, depending on the signal), or if the new signal was received before the signal handler has returned, the signal could be lost completely. Later, reliable signals were introduced to solve these problems, but SYSV and BSD took different routes.

BSD introduced a **sigaction()** function to replace the old **signal()** function, and then re-implemented **signal()** in terms of **sigaction()**, but changed the semantics of **signal()** so that a signal handler installed with the **signal()** function stays

installed. SYSV kept the old version of signal() which uninstalls itself, and introduced a whole mess of different functions for dealing with reliable signals. POSIX uses sigaction(). Linux follows POSIX, but can provide BSD signals if the BSD environment is selected, as described below.

Terminal handling also varies, and BSD terminal handling is significantly different from SYSV and POSIX, although SYSV and POSIX terminal handling are similar enough that both interfaces are easily provided by Linux by using the same mechanism in the kernel. The SYSV scheme is known as **termio**, and the POSIX scheme is known as **termio**. The older BSD scheme is known as **sgtty**. When porting a BSD application, you may notice compiler warnings and errors referring to sgtty.h, **TIOCGETP**, **TIOCSETP**, **TIOCFLUSH**, **RAW**, and other similar things.

To compile applications written specifically for the BSD platform, simply compile with the **-I/usr/include/bsd** flag, and link the application with the **-lbsd** flag. This makes BSD terminal handling **ioctl()**'s work, and makes **signal()** install the reliable signals that “recent” BSD code expects.

System Dependencies

Some code simply requires functionality that isn't defined by any standard, or at least any sufficiently popular one, and is therefore written separately for each operating system. An example of this is finding the current load average or other indication of system load. For most implementations of Un*x, this requires that a process have superuser powers, and that it go looking in special places in kernel memory for magic numbers. Exactly where to look in the kernel memory (usually **/dev/kmem**) is determined by the derivation of the source code—SYSV code looks in one place, and BSD code looks in another. On other operating systems, many various methods are used.

In Linux, the load average can be obtained by any process by simply reading the file **/proc/loadavg**--you can see it yourself at the command line by typing **cat /proc/loadavg**. Fortunately, this and many other special features like it are standard across all installations of Linux and fairly easy to use.

Unfortunately, there are a few problems you may run into that are not quite as tractable. The formats of some files, such as /etc/utmp, are not all well-defined under Linux. That is to say, standards do exist, but they appear to be interpreted differently on different platforms. This situation will hopefully improve over time, but right now, it appears to be difficult to write code which works on all Linux installations.

For now, I recommend having a reasonably standard Linux distribution yourself, and making your software work on your own computer, and then ask

that users who have problems using your software contact you. Work it out for each individual case when they do, perhaps using **#ifdef**'s, so that when you release a new copy of your software, it will work on more Linux installations. It will also allow your software to work on more non-Linux platforms. I also suggest getting the Linux Documentation Project man pages, which include man pages defining these formats if they did not come with your Linux distribution. Follow them in the best way you can, pointing out to the authors where and why they are not clear, and help re-write them if necessary to clarify them. With your help, these small problems can be resolved, in time.

Non-Broken Behavior

In some cases, source code written on other operating systems makes assumptions that are true on the original operating system, but which are not true under Linux because some functionality has been fixed. An example is **select()**. When **select()** was first introduced, it was pointed out in the man page that the *time* parameter, which was passed to select by reference (& *time*), was not currently modified, but that in the future it might be modified to return the time remaining in the select. Programmers either never noticed or completely ignored this advice, and now that several operating systems, including Linux, do modify the time parameter, programs which depended on it not being modified are breaking.

As a result, at regular intervals (about once a month, I think), someone posts to the newsgroup **comp.os.linux.misc**, saying: "The Linux **select()** call is broken!" even though the Linux manual pages and the Linux GCC FAQ carefully explain the situation. Operating systems keep being improved, but one thing never changes: some people will alway refuse to read the documentation.....

Some code blindly assumes that signals like SIGBUS and SIGEMT are defined, even though there is no guarantee (under POSIX) that these signals will exist on a platform. This type of code is easily fixed—simply wrap code referring to the offending signal in **#ifdef SIGNAME**, so that it is only included on operating systems that define that signal. Alternately, you can re-define the signals to **SIGUNUSED**, by adding **-DSIGFOO=SIGUNUSED** to the CFLAGS line in the Makefile.

Many programmers have ignored warnings not to make any assumptions about the FILE structure, and have dereferenced members of the FILE structure at will. Ignoring for the moment the argument that the original standard I/O should have included macros or functions to access those members, this causes problems under Linux, because standard I/O under Linux is based on C++ iostream library, and therefore has completely different structure members. As an example, while porting GNU Emacs 19.xx to Linux, I found that emacs

wants to know how many characters are left in the stdio buffering, and the default behavior was to look at **(FILE)->_ptr - (FILE)-<_base**, which don't exist under Linux. Fortunately, this was done with a macro which is defined in the Emacs source, called **PENDING_OUTPUT_COUNT**, which I was able to define in the Linux-specific **file linux.h** as **(FILE)->_pptr - (FILE)->_pbase**.

Also, it appears that the standard libraries of many operating systems define a symbol called sys_siglist, but do not declare it. Since it is declared in the Linux header files, you simply need to comment out any extra declaration of **sys_siglist** in the source for the program. This is true of several other features as well: **sys_siglistis** only an example.

A Case Study

I thought that I would find a sample program to port as an example for this column, but over a weekend, as I downloaded program after program, I was just more and more impressed with the Linux C library as program after program compiled with simple tweaks like changing the Makefile to use gcc instead of cc and changing paths to executables. I finally found a program that might give some people problems porting, an editor called Freyja.

I copied the supplied makefile.unx to Makefile, and edited the Makefile. I changed CFLAGS to -O2 to use the highest level of optimization from GCC, and typed make at the command line. GCC complained that **TIOCGETP**, **TIOCSETP**, and **RAW** were undefined. This means that Freyja is written with BSD in mind. There did not appear to be any #define's that I could make to change Freyja's behavior to SYSV or POSIX, either.

So, following the steps in the GCC-FAQ, I added **-I/usr/include/bsd** to the CFLAGS line, and **-lbsd** to the link line (called FXLINK in Freyja for some strange reason; it's usually called LDFLAGS), and ran make again.

That was **all** that was required to “port” this bsd-oriented program. I had to read the documentation to find out that I needed to call it with the arguments “-kT -s29” to tell it how to write to the screen and read from the keyboard, or that I needed to compile an equivalent change into the resource file that Freyja uses, but it was very simple.

Freyja is written by Craig A. Finseth, and is available via ftp from **mail.unet.umn.edu**, or if you don't have ftp access, via U.S. mail. Quoting the README:

Diskette: Send the author blank diskettes:

- 3 1/2" (1.44 MB), or

- 3 1/2" (720 KB)

and a SASE or enough stamps to cover return postage plus a dollar or so (so that I can buy a diskette mailer). Or you can just send me about US\$5.00 in check, stamp, whatever and I will furnish the diskette(s) and mailer. Non-US people can send me four 1.44 MB 3 1/2" diskettes in lieu of money. (More money is always nice, but please don't feel obligated in any way.)

The address is: Craig Finseth1343 LafondSt. Paul, MN55104, USA

Help!

Here's your chance to contribute! If you have difficulties porting a general Un*x application to Linux, please either send email to johnsonm@redhat.com or send snail mail to **Programming Tips, *Linux Journal*, P.O. Box 85867, Seattle, WA 98145-1867**, with a description of how to get the program via the internet, or with a copy of the application enclosed on floppy, 150MB QIC tape, or standard DAT, and with a detailed explanation of what you have tried in your attempt to port it, and I may try it myself, especially if it looks like it will make worthwhile material for this column.

Archive Index Issue Table of Contents

Advanced search

The DF Command

Phil Hughes

Issue #1, March 1994

In this series we look at a command. We don't necessarily describe everything it can do but rather some useful capabilities.

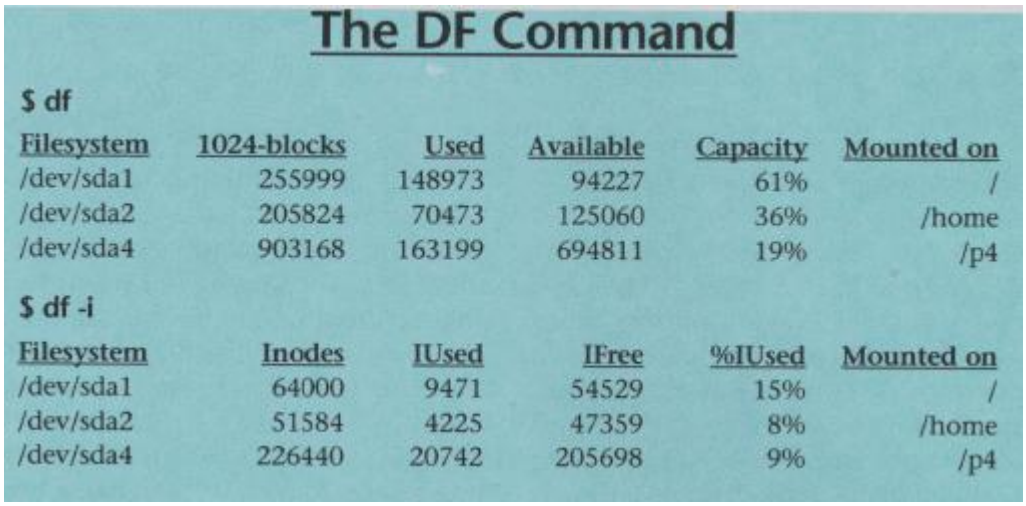
The df command is used to show the amount of disk space that is free on file systems. In the examples, df is first called with no arguments. This default action is to display used and free file space in blocks. In this particular case, th block size is 1024 bytes as is indicated in the output.

The first column show the name of the disk partition as it appears in the /dev directory. Subsequent columns show total space, blocks allocated and blocks available. The capacity column indicates the amount used as a percentage of total file system capacity.

The final column show the *mount point* of the file system. This is the directory where the file system is mounted within the file system tree. Note that the root partition will always show a mount point of /. Other file systems can be mounted in any directory of a previously mounted file system. In the example, there are two other file systems, the first in mounted as /home and the second is mounted as /p4.

In the second example, df is invoked with the **-i** option. This option instructs df to display information about *inodes* rather that file blocks. Even though you think of directory entries as pointers to files, they are just a convenience for humans. An inode is what the Linux file system uses to identify each file. When a file system is created (using the mkfs command), the file system is created with a fixed number of inodes. If all these inodes become used, a file system cannot store any more files even though there may be free disk space. The **df -i** command can be used to check for such a problem.

The df command allows you to select which file systems to display. See the man page for details on this capability.



The DF Command

$ df

Filesystem	1024-blocks	Used	Available	Capacity	Mounted on
/dev/sda1	255999	148973	94227	61%	/
/dev/sda2	205824	70473	125060	36%	/home
/dev/sda4	903168	163199	694811	19%	/p4

$ df -i

Filesystem	Inodes	IUsed	IFree	%IUsed	Mounted on
/dev/sda1	64000	9471	54529	15%	/
/dev/sda2	51584	4225	47359	8%	/home
/dev/sda4	226440	20742	205698	9%	/p4

Figure 1. The DF Command

email: info@linuxjournal.com

Archive Index Issue Table of Contents

Advanced search

What's GNU?

Arnold Robbins

Issue #1, March 1994

Welcome to the inaugural edition of What's GNU?, a semi-regular column on the GNU project. The “semi” in semi-regular means that we expect this column to appear in every issue of *Linux Journal*, but it may not happen occasionally.

The content will be a mixture of “What is the GNU project”---history, motivation, status of various parts, in other words general things, and more in-depth looks at the various “major” pieces of GNU software, e.g. columns on **gawk**, GNU **make** , Emacs, etc. If possible, I will solicit articles from the primary authors of the various programs, in which case I will serve more as an editor. Occasionally, I will devote a column to some other piece of free software that may not be part of the GNU project, but which nonetheless is likely to be of interest to the readers of *Linux Journal*. Many of these programs are distributed under the same terms as GNU programs.

While I will always strive to present accurate, up to date information, this column in no way represents the official statements and/or policies of the Free Software Foundation.

Here are some questions, and the answers that go with them.

Q. What is the GNU Project?

A. The GNU project is an ongoing effort on the part of the Free Software Foundation (FSF) to create a complete, usable, *freely redistributable* software development environment, including both operating system and utilities. In particular, the FSF has chosen to create a clone of Unix. GNU source code is copyrighted, using a license that requires you to distribute or make source code available when you distribute binaries.

As of this writing, essentially everything but the kernel has been completed. A future column will list everything that has been done; hopefully another future column will discuss the status of the GNU kernel, called the “Hurd”.

By the way, GNU stands for “GNU's Not Unix”. The `G' is pronounced, it is not silent.

Q. What is the FSF?

A. The Free Software Foundation is a not-for-profit corporation whose goal is spread the use of free software. To help this goal, it started the GNU project, described above.

The FSF was founded by Richard Stallman and several others in approximately 1983. It has a few full-time employees, and a large number of volunteers working on the GNU project. Contributions to the FSF are tax-deductible in the US.

The FSF can be reached at:

Free Software Foundation675 Massachusetts AvenueCambridge, MA 02139-3309USAVoice: +1-617-876-3296Fax: +1-617-492-9057Email, general info: gnu@prep.ai.mit.edu Email, to order doc and/or media: fsf-order@gnu.ai.mit.edu

To order software and/or documentation in Japan: japan-fsf-orders@prep.ai.mit.edu FAX (in Japan):0031-13-2473 (KDD)0066-3382-0158 (IDC)

Q. What is the meaning of the word “free”?

A. As used by Richard Stallman and the FSF, the term “free” means that source code for software is freely available. It does *not* mean “no monetary cost”. This has often been a source of confusion among people who are not familiar with the FSF and its goals.

The FSF makes sure that its source code is available by licensing it under the GNU General Public License, or GPL. FSF source code is *not* public domain. It is copyrighted, and distributed with a license that allows you to modify the code. If you distribute modified versions of FSF programs (e.g., in binary), the GPL requires you to distribute your modifications to the code under the same terms as the original source code. If you never distribute your modifications, then there is nothing that requires you to distribute your source code, either.

The GPL will hopefully be the topic of a future column. In the meantime, if you have a Linux system, you undoubtedly have a copy of the GPL somewhere, probably in one or more files named **COPYING**.

Q. Why is GNU relevant to Linux?

A. It is fair to say that without the GNU project, your Linux system would not be the usable, fairly complete environment that it is today. Essentially all your utilities are from the GNU project—the C compiler, **make** , **awk**, the shell, the editors, almost all of the utilities are the GNU versions.

As the GNU developers update the programs, you should be interested in acquiring and installing these new versions, since they very likely contain bug fixes, performance enhancements, and/or new features.

Q. Who is Arnold Robbins and why is he qualified to write this column?

A. I am a professional programmer who has been working with various Unix systems since 1981. I have been involved with the GNU project as a volunteer since 1988. The main program I have worked with is **gawk**--GNU Awk. I am both a co-author of the program and the primary author of the accompanying manual that documents the Awk language and the **gawk** implementation. I also wrote some of the smaller miscellanious programs in the Shell Utilities package.

I am also a user (and debugger) of many of the major GNU programs, such as the C and C++ compilers ([all of these in cw], gcc, g++), the debugger gdb, and the grep and diff suites.

As a developer and user, I interact with many of the other developers of GNU tools, and in general I attempt to “keep my finger on the pulse” of the GNU project.

But, as I'm neither a paid employee nor an official of the FSF, anything I write in this column is my own interpretation of things. If you are in doubt about something, always contact the FSF directly; never just take my word for it.

Q. What is the history of the GNU Project?

A. Richard Stallman started the GNU Project in 1983. The first programs released were Emacs and **gdb**, the debugger. Since then, the project has grown. As of this writing, essentially everything but the middle third of the kernel exists and is fairly stable. Much documentation needs to be written, and work is proceeding on the kernel. Contact gnu@prep.ai.mit.edu for a list of tasks that remain to be initiated.

Q. Why did Richard Stallman start the GNU project?

A. We'll let RMS (as he's known) answer that one himself. Included below is a document known as the “GNU Manifesto”. It was last modified in 1985. This version comes from the Emacs 19.22 distribution.

What's GNU? Gnu's Not Unix!

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have an Emacs text editor with Lisp for writing editor commands, a source level debugger, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. A new portable optimizing C compiler has compiled itself and may be released this year. An initial kernel exists but many more features are needed to emulate Unix. When the kernel and compiler are finished, it will be possible to distribute a GNU system suitable for program development. We will use TeX as our text formatter, but an nroff is being worked on. We will use the free, portable X window system as well. After this we will add a portable Common Lisp, an Empire game, a spreadsheet, and hundreds of other things, plus on-line documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer filenames, file version numbers, a crashproof file system, filename completion perhaps, terminal-independent display support, and perhaps eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaosnet, and Internet protocols for communication.

GNU is aimed initially at machines in the 68000/16000 class with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on smaller machines will be left to someone who wants to use it on them.

To avoid horrible confusion, please pronounce the `G' in the word `GNU' when it is the name of this project.

Who Am I?

I am Richard Stallman, inventor of the original much-imitated EMACS editor, formerly at the Artificial Intelligence Lab at MIT. I have worked extensively on compilers, editors, debuggers, command interpreters, the Incompatible Timesharing System and the Lisp Machine operating system. I pioneered terminal-independent display support in ITS. Since then I have implemented one crashproof file system and two window systems for Lisp machines, and designed a third window system now being implemented; this one will be ported to many systems including use in GNU. [Historical note: The window system project was not completed; GNU now plans to use the X window system.]

Why I Must Write GNU

I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI lab to deny MIT any legal excuse to prevent me from giving GNU away.

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to Help

I have found many other programmers who are excited about GNU and want to help.

Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.

One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machines should be complete, ready to use systems, approved for use in a residential area, and not in need of sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work for GNU. For most projects, such part-time distributed work would be very hard to coordinate; the independently-written parts would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, th these utilities will work right when put together. Even allowing for Murphy to create a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high by programmers' standards, but I'm looking for people for whom building community spirit is as important as making money. I view

this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air.

This means much more than just saving everyone the price of a Unix license. It means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company which owns the sources and is in sole position to make changes.

Schools will be able to provide a much more educational environment by encouraging all students to study and improve the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do with it will be lifted.

Arrangements to make people pay for using a program, including licensing of copies, always incur a tremendous cost to society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay for. And only a police state can force everyone to obey them. Consider a space station where air must be manufactured at great cost: charging each breather per liter of air may be fair, but wearing the metered gas mask all day and all night is intolerable even if everyone can afford to pay the air bill. And the TV cameras everywhere to see if you ever take the mask off are outrageous. It's better to support the air plant with a head tax and chuck the masks.

Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free.

Some Easily Rebutted Objections to GNU's Goals

“Nobody will use it if it is free, because that means they can't rely on any support.”

“You have to charge for the program to pay for providing the support.”

If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable.

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost.

If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem; you are not at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible for there to be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them.

Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just hand-holding and repair service. If it is true that users would rather spend money and get a product with service, they will also be willing to buy the service having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

“You cannot reach many people without advertising, and you must charge for the program to support that.”

“It's no use advertising a program people can get free.”

There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it.

On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really

necessary to spread GNU. Why is it that free market advocates don't want to let the free market decide this?

“My company needs a proprietary operating system to get a competitive edge.”

GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems.

I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each.

“Don't programmers deserve a reward for their creativity?”

If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

“Shouldn't a programmer be able to ask for a reward for his creativity?”

There is nothing wrong with wanting pay for work, or seeking to maximize one's income, as long as one does not use means that are destructive. But the means customary in the field of software today are based on destruction.

Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction.

The reason a good citizen does not use such destructive means to become wealthier is that, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics; or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one to do so. Specifically, the desire to be rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

“Won't programmers starve?”

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else.

But that is the wrong answer because it accepts the questioner's implicit assumption: that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing.

The real reason programmers will not starve is that it will still be possible for them to get paid for programming; just not paid as much as now.

Restricting copying is not the only basis for business in software. It is the most common basis because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other bases of organization which are now used less often. There are always numerous ways to organize any kind of business.

Probably programming will not be as lucrative on the new basis as it is now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

“Don't people have a right to control how their creativity is used?”

“Control over the use of one's ideas” really constitutes control over other people's lives; and it is usually used to make their lives more difficult.

People who have studied the issue of intellectual property rights carefully (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes.

For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for whom the cost and effort of a license agreement are small compared with setting up production, the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of non-fiction. This practice was useful,

and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: are we really better off granting such license? What kind of act are we licensing a person to do?

The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object code which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to.

“Competition makes things get done better.”

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it des a good job; but its defenders are wrong in assuming it always works this way. If the runners forget why the reward is offed and become intent on winning, no matter how, they may find other strategies—such as, attacking other runners. If the runners get into a fist fight, they will all finish late.

Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them (“For every ten yards you run, you are allowed one kick.”). He really ought to break them up, and penalize runners for even trying to fight.

“Won't everyone stop programming without a monetary incentive?”

Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way.

But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will not disappear, only become less. So the

right question is, will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of non-monetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself.

Then most of them left when offered a chance to do the same interesting work for a lot of money.

What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are banned.

“We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey.”

You're never so desperate that you have to obey this sort of demand.

Remember: millions for defense, but not a cent for tribute!

“Programmers need to make a living somehow.”

In the short run, this is true. However, there are plenty of ways that programmers could make a living without selling the right to use a program. This way is customary now because it brings programmers and businessmen the most money, not because it is the only way to make a living. It is easy to find other ways if you want to find them. Here are a number of examples.

A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.

The sale of teaching, hand-holding and maintenance services could also employ programmers.

People with new ideas could distribute programs as freeware, asking for donations from satisfied users, or selling hand-holding services. I have met people who are already working this way successfully.

Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a Software Tax:

Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.

But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.

The consequences:

- the computer-using community supports software development.
- this community decides what level of support is needed.
- users who care which projects their share is spent on can choose this for themselves.

Endnotes:

The author would like to thank Len Tower of the GNU project for his comments on this column.

Questions and/or comments about this column can be addressed to the author via postal mail C/O *Linux Journal*, or via email to arnold@gnu.ai.mit.edu .

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux…

Matt Welsh

Issue #1, March 1994

“Cooking With Linux” is a monthly feature, intended primarily to give the Linux bourgeois an inside look at some of the issues, both humorous and profound, pertinent to the Linux development community. Interested in recipes for virtual memory or virtual beer? Here's the place. Interested in the politics and structure of the free software world? Here's the place. Interested in serious, unbiased discussion of Linux development? Better check somewhere else.

Guerrilla UNIX Development

We live in a society of hierarchies, organizations, and structures. Virtually every aspect of our lives is structured in some way, be it socially, politically, religiously —you name it. Rules are hard and fast. Everything has a proper place.

This is especially true of the computing community, no doubt because computers themselves are highly structured, deterministic beasts. Large software development corporations employ scores of programmers into hierarchies so deep that the lowest-level employees are often lost in the carpeting. Every position has one or more obscure titles ranging from “Manager” to “Submanager” to “Peon”, often including secret code-names for the particular project at hand, to thwart the copycat tendencies of competing corporations.

This organization extends well beyond the physical and social world, as well— electronic files, networking protocols, and even USENET newsgroups are arranged into hierarchies. Hierarchies can be real, virtual, or purely imaginary. They're everywhere. And this is all intended, I'm sure, to teach us a very good lesson: Nothing in the Real World gets done without a strong sense of teamwork and strict organization. Nothing!

Well, almost nothing. The Linux development community is a seemingly unstructured, amorphous conglomerate. There is no single leader, or nucleus, for the organization. No single entity can claim responsibility for Linux. Each of

the many parts of the Linux system, from individual kernel source files to entire distributions, are copyrighted by different individuals. In fact, it looks like a big mess. Or is the Linux development cabal as disorganized as it seems?

The Linux development community grew, like crystals, around a solitary kernel (no pun intended). Linus Torvalds was, in the beginning, singly responsible for the kernel development, back when the kernel itself was thought of as the entirety of Linux. In more recent times, however, Linus claims he authored less than 50% of the current kernel code. So, while Linus may be the “father” of this rapidly growing amoeba, very few people would assign him the title of “leader”—all due respect, of course.

But that's ancient history. It's more instructive, I think, to look at how the Linux community *appears* to work now. The community seems to be essentially a huge network of individuals, all working towards a single goal (or many goals, as the case may be). In fact, Linux development is not unlike guerrilla warfare, wherein every member of a small, seemingly disorganized political faction has more or less the same idea of what the overall goal is, but each individual may have his or her own ideas about how to go about it.

Within this gargantuan bee colony, it is difficult to make out any direct or intentional hierarchy among the developers, but there does seem to be a metaphysical hierarchy of sorts—a “caste” system which isn't enforced externally, but instead stems from within the community itself. This hierarchy is more a handy convention than a definite rule. Guerrilla troops are tied together by a similarly complex system of blood ties, tenure, and the occasional vendetta. But the driving force behind it all is a shared vision of the world (or the operating system) to become. Basic human natures fills in the rest.

What is this hierarchy? Most members of the Linux community observe a straightforward split between “developers” and “non-developers”. It's not always clear who is a developer and who isn't, and what gives someone the right to call themselves a developer is equally as muddy. But it seems obvious that some kind of ladder does exist. Finding the rungs is another matter altogether.

On the lowest level, it would seem, is the myriad of Linux users. These range from the extremely intelligent to the utterly clueless, but on the whole, the “users” don't contribute to the growth of Linux, beyond the occasional bug report or helping hand to another of their kind. Of course, such a claim is purely bogus—in fact, Linux users contribute to the expansion of Linux just by using it. The more popular and widespread the system is, the more flexible it has to be. Nevertheless this is an indirect effect of propagation, and while individual users may not be able to directly shape the system, the user

community as a whole contributes an essential aspect of Linux just through its existence. (Waxing philosophical again, am I? Don't say we didn't warn you.)

On the highest level of this quasi-taxonomy are the “developers” who actually program and support the system. They make the most noticeable direct influence on the furtherance of Linux itself, but as is the case with the user community described above, the wide disarray of differing development goals lends itself to another form of “meta-development”. The seemingly uncontrollable patchwork of many independent development efforts, all reduced and squeezed into a single package, gives Linux a kind of spirit and—I'm not afraid to use the word—*charm* not found in other operating systems.

Whereas commercial UNIX systems smack of being “developed by committee”, Linux actually resembles software that has been developed by a madhouse. UNIX kernel development is deluged with black magic, obfuscation, obsolescence, and such sheer complexity that it's a wonder that programmers who, for the most part, have never met face-to-face can get anything done at all. Not to mention the obvious troubles arising from the language barrier and the inherent limitations of electronic mail.

Yet, at the core (no pun intended) of this crazy mishmash of conflicting programming models and conventions a pearl has emerged. Not only has Linux managed to survive the rough seas of early development, but it has snowballed into a colossal collaborative development effort by people from all walks of life. Everyone has the chance to contribute, no matter how insignificantly. Linux is developed by and for its users, not for some arbitrary third-party target market that commercial UNIX vendors attempt to cater to. If Linux reeks of hackishness that is because its users (and thus its developers) are hackers. Of course, non-hackers are eager to jump onto the bandwagon as well, and the first to do so have the opportunity to steer it in that direction for others to follow.

So it doesn't seem to make much sense to classify the Linux community into some kind of ladder or scale, even with Linus at the top and J. Random User at the bottom. Each member of the community has his or her own personal niche and potential for contribution. The only difference is the amount of effort that each person puts in. (For some, it takes a great deal of effort *not* to hack Linux, but that's another story.) Age, race, sex, and planetary origin are completely irrelevant. It's attitude that matters—and everyone has their own attitude.

What this all comes down to is this: You can look at the Linux community in any way that you please. Some prefer to see it as a ladder by ranking the members based on knowledge, aptitude, contribution, or other such intangible qualities for which there are more exceptions than rules. Others envision a flat hierarchy

in which everyone is equal in a utopian technocratic society. Others, like myself, visualize a giant cocktail party gone madly awry. Who are you going to believe?

One thing is for certain, however. The Linux development guerrilla troops are effecting their programming and political aims no matter how disorganized they may appear to be. Linux is spreading like wildfire in the dry, ready-to-burn forests of the commercial UNIX market, determined to devastate anything in its path. §

We came. We saw. We hacked.

Matt Welsh is a systems programmer, student, and extreme Linux geek at Cornell University. He coordinates the Linux Documentation Project, an effort to produce a canonical set of manuals for Linux.

Archive Index Issue Table of Contents

Advanced search

The Debian Distribution

Ian A. Murdock

Issue #1, March 1994

The distribution of Linux as a commercial product is unique in the world of computer software in that most commercial Linux enterprises have not developed the systems that they market and sell.

Linux is freely-available and freely-redistributable, so anyone who wishes to do so may obtain a Linux distribution and resell it for profit without the need to obtain licenses or pay royalties. The GNU General Public License, which covers most system-level components and distributions of Linux software, legally allows the occurrence of this “unusual” practice. There is little chance that Linux could ever have reached the level of quality that it has today without the freedom provided to its users by the GNU General Public License. However, the freedom that has made Linux possible also poses an interesting problem to the legitimacy of it in the eyes of the large commercial organizations in which it must gain acceptance to succeed. “What exactly is `Linux'? Is a particular distribution `Linux'? What is the difference between `Linux' and a `Linux distribution'? Why hasn't this particular commercial vendor developed their own distribution of Linux? Who else will stand behind and support the Linux product if its vendor has not developed it? Is Linux for hackers and hobbyists, or is it really a suitable replacement for the commercial operating systems in use at `real-life' businesses?”

There has been much talk lately of consortiums, foundations and official Linux organizations to ensure that the pursuit of profit by commercial Linux distributors will never take precedence over the quality of the products and services that they sell. Others have suggested that the developers of the system-level software or end-user distributions should undertake the responsibility; after all, the developers spend a great deal of time and energy constructing the software and systems for free and are not likely to “sell out” for profit.

Whether we like it or not, Linux is rapidly becoming big business and individuals are profiting from it. While it does not bother me personally that people are making a living from distributing and selling Linux, we, the developers and users of it, must ensure that providing a first-class product remains the highest priority of such businesses and that the future of the operating system is never compromised in the battle for competitive advantage and higher profit. Only by doing so can we assure the future success of Linux in the commercial market, a success that will benefit everyone.

Before proceeding further, an introduction is probably in order. My name is Ian A. Murdock, and in mid-August of 1993 I began working on what would eventually become the Debian Linux distribution. Over the past four months and with the assistance and support of hundreds of users around the world, Debian has evolved into a commercial-quality system that will soon be able to compete successfully alongside commercial UNIX implementations and non-UNIX operating systems alike. By the time this article is published, Debian will be available to the public via anonymous FTP.

We are also currently in the process of forming the Debian Linux Association, an organization that will serve as the official maintainer of Debian and the backbone and “watchdog” of commercial distribution of it. The Free Software Foundation is involved and will soon be distributing Debian on CD-ROM. There is much work to be done yet, but just as much progress has already been made. The future looks very bright, and by the beginning of March 1994 we hope to be well on our way toward our goal of making Linux a viable alternative to commercial operating systems.

My goal from the beginning has been to create a commercial-quality distribution of the Linux operating system, a product which has not existed until now but which is absolutely essential to the success of Linux in the commercial market. In this month's column I am including the Debian Linux Manifesto as a means of introducing what I have done and what I plan to do with Debian Linux. In future columns I will discuss such topics as the commercial potential of Linux, the problems with the current commercial distribution of the operating system and the progress of the Debian Linux Association toward solving these problems.

What is Debian Linux?

Why is Debian being constructed?

Distributions are essential to the future of Linux. Essentially, they eliminate the need for the user to locate, download, compile, install and integrate a fairly large number of essential tools to assemble a working Linux system. Instead, the burden of system construction is placed on the distribution creator, whose

work can be shared with thousands of other users. Almost all users of Linux will get their first taste of it through a distribution, and most users will continue to use a distribution for the sake of convenience even after they are familiar with the operating system. Thus, distributions play a very important role indeed.

Despite their obvious importance, distributions have attracted little attention from developers. There is a simple reason for this: they are neither easy nor glamorous to construct and require a great deal of ongoing effort from the creator to keep the distribution bug-free and up-to-date. It is one thing to put together a system from scratch; it is quite another to ensure that the system is easy for others to install, is installable and usable under a wide variety of hardware configurations, contains software that others will find useful, and is updated when the components themselves are improved.

Many distributions have started out as fairly good systems, but as time passes attention to maintaining the distribution becomes a secondary concern. A case-in-point is the Softlanding Linux System (better known as SLS). It is quite possibly the most bug-ridden and badly maintained Linux distribution available; unfortunately, it is also quite possibly the most popular. It is, without question, the distribution that attracts the most attention from the many commercial "distributors" of Linux that have surfaced to capitalize on the growing popularity of the operating system.

This is a bad combination indeed, as most people who obtain Linux from these "distributors" receive a bug-ridden and badly maintained Linux distribution. As if this wasn't bad enough, these "distributors" have a disturbing tendency to misleadingly advertise non-functional or extremely unstable "features" of their product. Combine this with the fact that the buyers will, of course, expect the product to live up to its advertisement and the fact that many may believe it to be a commercial operating system as there is also a tendency not to mention that Linux is free nor that it is distributed under the GNU General Public License. To top it all off, these "distributors" are actually making enough money from their effort to justify buying larger advertisements in more magazines; it is the classic example of unacceptable behavior being rewarded by those who simply do not know any better. Clearly something needs to be done to remedy the situation.

How will Debian attempt to put an end to these problems?

The Debian design process is open to ensure that the system is of the highest quality and that it reflects the needs of the user community. By involving others with a wide range of abilities and backgrounds, Debian is able to be developed in a modular fashion. Its components are of high quality because those with expertise in a certain area are given the opportunity to construct or maintain the individual components of Debian involving that area. Involving others also

ensures that valuable suggestions for improvement can be incorporated into the distribution during its development; thus, a distribution is created based on the needs and wants of the users rather than the needs and wants of the constructor. It is very difficult for one individual or small group to anticipate these needs and wants in advance without direct input from others.

Debian Linux will also be distributed on physical media by the Free Software Foundation and the Debian Linux Association. This provides Debian to users without access to the Internet or FTP and additionally makes products and services such as printed manuals and technical support available to all users of the system. In this way, Debian may be used by many more individuals and organizations than is otherwise possible, the focus will be on providing a first-class product and not on profits or returns, and the margin from the products and services provided may be used to improve the software itself for all users whether they paid to obtain it or not.

The Free Software Foundation plays an extremely important role in the future of Debian. By the simple fact that they will be distributing it, a message is sent to the world that Linux is not a commercial product and that it never should be, but that this does not mean that Linux will never be able to compete commercially. For those of you who disagree, I challenge you to rationalize the success of GNU Emacs and GCC, which are not commercial software but which have had quite an impact on the commercial market regardless of that fact.

The time has come to concentrate on the future of Linux rather than on the destructive goal of enriching oneself at the expense of the entire Linux community and its future. The development and distribution of Debian may not be the answer to the problems that I have outlined in the Manifesto, but I hope that it will at least attract enough attention to these problems to allow them to be solved.

Archive Index Issue Table of Contents

Advanced search

Linux Installation and Getting Started

Phil Hughes

Issue #1, March 1994

The quick review is “if you plan to run Linux you must have this book”. This book is a product of the Linux Documentation Project and Matt Welsh is the coordinator of that project. What may sound strange is that this book is free but, once you understand the spirit of the Linux community, it makes sense.

Matt released version 1 of this book in August, 1993. Version 2 was just made available on January 14, 1994. The major revision was to make it generic for most any Linux distribution. The previous edition was somewhat specific to the SLS distribution.

The following appears on the cover of the book and is a reasonable explanation of why the book exists and who should read it.

> “This book is an installation and new-user guide for the Linux system, meant for UNIX novices and gurus alike. Contained herein is information on how to obtain Linux, installation of the software, a beginning tutorial for new UNIX users, and an introduction to system administration. It is meant to be general enough to be applicable to any distribution of the Linux software. Anyone with interest in installing and running Linux should read this book first.”

The book starts with an introduction to Linux including history, features, design and hardware requirements. The next chapter covers obtaining and installing Linux. How to get Linux from the Internet as well as other sources is covered. The section on installation problems covers many if not most of the questions that are regularly asked on the Linux newsgroups on Usenet.

The third chapter is a Linux tutorial. While not differing much from a standard Unix tutorial it covers the basics that a newcomer to Linux (or Unix) will need to know. It starts very basic but covers pipes, wildcards, file permissions, job control and the vi editor.

Systems administration is covered which includes routine stuff as well as disaster recovery. The final chapter is an introduction to the advanced features of Linux including X Windows, TCP/IP networking, E-mail and Usenet news.

Appendices cover other documentation, linux distributions, a Linux BBS list where you can get Linux files and even a tutorial on using ftp to get Linux off the Internet.

All in all, a thorough job of writing what needs to be said to get people started. If I had to find shortcomings it would be in the appendices talking about other sources of information and Linux distributions. The other books list mentions quite a few of the O'Reilly books (which are Unix-specific) but leave out what I would consider important books such as *The UNIX Programming Environment* by Kernighan and Pike (Prentice-Hall), *The UNIX System* by Stephen Bourne (Addison-Wesley), *Introducing the UNIX System* by Henry McGilton and Rachael Morgan (McGraw-Hill) and SSC's series of Unix pocket references and tutorials. The list of distributions covers MCC, TAMU, Nascent, Slackware and Trans-Ameritech but leave out major players such as Yggdrasil. Also, the information on Slackware is out of date but, as the change only happened 30 days ago, I can't be overly critical.

The final word: well worth the price which, as I mentioned is free if you have Internet access. It can be found on SunSite (**sunsite.unc.edu:/pub/Linux/docs/LDP/install-guide**) as well as all the other standard mirror sites. It is available as LaTeX source, a dvi file and in PostScript. If you don't have Internet access (or just would rather buy paper) SSC has agreed to make a comb bound copy available for $15 plus shipping ($3 in the U.S.).E-mail info@linuxjournal.com or call (206)527-3385 for details.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #1, March 1994

All of the following letters are in response to the questionnaire we sent out to Usenet in December, 1993. The comments below have been used with permission of the authors. Some have been edited because of space considerations.

In the area of molecular biology, it appears many of the preferred programs work optimally under a Unix environment. Since most scientists lack a Sun SPARCstation at home (some may), or even at work, it seems that Linux offers the ability to easily bring that sort of computer power to the home. I've seen that some of the molecular biology utilities have been ported to Linux. Some of the examples are:

ACaDB (A Caenorhabditis elegans Data Base) -nematode database for this evolved:

AAtDB (An Arabidopsis thaliana Data Base) -Plant database and other similar databases are evelving to serve as databases to accumulate information on these model biological systems.

GDE (Genetic Design Environment) An X-windows based working environment to run molecular-biology based applications. It allows for the easy entry and visualization of nucleic acid (DNA, RNA) or amino acids (protein) sequences. It further allows analysis of these sequences by programs such as:

- FASTA - sequence alignment algorithm to compare sequences to a database
- BLASTA - similar sequence analysis but by a different algorithm

If a local database (like GenBank) is not available on a Linux system, GDE is designed to format an e-mail submission in the proper format to one of the national databanks available on the Internet.

PHYLIP - phylogeny analysis programs and others wihch may easily be linked to GDE.

I'm not sure if many others have been ported to Linux, but it seems quite feasible. I see new Unix versions of useful programs being offered and often wonder if anyone has ported them to Linux. The *Linux Journal* should be able to facilitate learning of this information. I feel Linux brings the most in easily obtainable computing power to the scientific workbench.Gerard R. Lazo,Southern Crops Research Laboratory, College Station, TX

Asked about other topics that should be covered in *LJ*, Kenneth listed

- Reviews of different distributions
- Comparisons between Linux and other operating systems
- Tips and tricks on Unix in general, and Linux in particular
- Interviews and/or articles by known freeware authors (not necessarily Linux specific software), e.g. FSF etc.
- Articles on emerging standards or trends in the (open systems) industry, e.g., COSE, POSIX, Usenix, WABI, X11R6...

Kenneth Osterberg,Helsinki, FINLAND

We should have had you design our magazine. See our Debian column, comparision article on Linux, DOS/Windows and OS/2, our programming tips column, our FSF column and even our interview with Linus. Guess we are on the right track.Editor

Good day!

I'm highly enthusiastic about the existence of *Linux Journal*. Since I started using Linux in February of 1993, I find my current magazine subscription is becoming totally irrelevant, since the editors are violent MS-DOS/Windows bigots.

About the software wanted column

This would be useful. What do people want for linux, in the way of software, additional drivers, and so on. It should be possible for a person's e-mail or actual address to be printed with their suggestions so people can either point out possible software, or ask about details so they can implement it.

About the New Linux-related products column

Definitely. For me, the major point of subscribing to a magazine is for the reviews, and for tutorials. I also love opinion pieces.

Andrew suggests the following additions to *LJ*

Alpha and beta software: Not reviews, but at least mentions of it. “Andrew Kuchling's FOOGOL compiler is currently in early beta. Word-of-mouth says it's not bad, although some of FOOGOL's more complex constructs produce buggy code at this point.” This is so people know what's on the way, and so corporate types can see that Linux is alive and well.

Also, keep us up to date on what's going on in other software areas, like the DOS/Windows and commercial Unix worlds. We don't want a 6-page review of Windows 4.0, but simply mention new programs that we should know about, and the reaction to them. (Was it buggy? Is Windows finally usable? Can Wine be rewritten to run 4.0's programs?) The same applies to new Unix versions; have they got any new ideas we can apply to Linux? Did they encounter any pitfalls?

Another feature might be “The Cutting Edge”, on using Linux in new and experimental ways. For example, to simulate a parallel processing system in software; as a server for a database; running it on a Pentium; using Linux as a cheap firewall system; object-oriented databases for Linux; using a Linux network to test networking hardware (according to Paul Tomblin, this is what Gandalf does). This is to show that Linux makes inexpensive PC-compatibles competitive with pricy workstations for demanding Unix applications. *LJ* should also show how Linux/X is better than DOS/Windows, capable of running everyday applications like word processors & spreadsheets.

LJ should not exclusively discuss things from the viewpoint of a beginning user. We also need more advanced articles on system administration, and programming topics (not necessarily Linux-specific ones). Some possibilities I'd personally like to see are:

Optimizing: What does gcc do when optimization is turned on? How can I write code that will optimize well?

Using PC-specific features in Unix: How can I play a digitized sound in my own programs? Display VGA graphics? Play an MPEG? Read a mouse?

Tweaking XF86 for better performance: How can I make it use less memory? What graphics cards are really fast?

Backing up your system: How to do it onto floppies? What tape devices are good?

Kernel patches: What's available? How well does it work?

SCSI: How do I go about moving from IDE to SCSI? Which cards and peripherals are good?

Adding your own system call to the kernel: What do you do?

Texture mapping in graphics programming: This is not a Linux-specific topic, but could be given with Linux-specific examples.

Most of these topics are too big for one article, and some partially duplicate FAQs. That's to be expected; articles should never (well, almost never) say “See the XYZ FAQ for the details”. If they're not giving the details, what's the point of the article? (On the other hand, articles should be written for various levels of expertise; some obscure details should be referred to FAQs, then.)

A longer-term idea: It should be possible for writers to include a demo program, possibly a complex one. For example, the texture mapping author might provide an svgalib program that lets the users change colours, and experiment with different noise functions on the fly. The software could be available via mail server or FTP; people without Internet access could order a diskette with the software from *LJ*. (This would be a large effort, though, and so should be postponed; maybe you could politely tell non-Internet readers that they're out of luck. :)) This software may not be significant enough for it to be put on tsx-11 or sunsite, so *LJ* should handle this itself. (Or convince tsx-11 to add a *LJ*/software directory.)

Also, it's _very_ important to have book reviews. A possible yearly feature might be “The Linux Bookshelf”, a listing and discussion of the best books for using and programming Linux (and Unix, more generally); both reference books and tutorial books should be covered, and both Linux Documentation Project efforts and third-party publishers. It would also be valuable to review Linux Doc. Project works as they're released (with allowances, when they're in alpha).

It's very important to also print negative reviews (possibly unsigned, to avoid chilling friendships). Some magazines (such as Compute!) didn't print negative reviews, saying they'd rather use the space to discuss a good program or book, but I think that's wrong. Otherwise, if product X isn't reviewed, is it because the reviewers hated it, or because the magazine has never seen it? There's no need to be cruel, especially in reviewing free products. (Our expectations are higher for commercial software, and people often write free software in their spare

time.) But we should know that foo0.99 compiles out of the box and works wonderfully, while bar0.99 requires lots of config hacking and seems a bit unstable once it's running. Reviews are really the most important factor to me; if *LJ*'s reviews section is perfunctory, then the magazine's usefulness is greatly impaired.

Another point: There's so much free software around, and it's updated so quickly that reviewing it all would be an impossible task. But a half page of 1-5 line mentions of especially notable or dangerously unstable software would be nice...

admscr0.01.tgz: A system administration script with menus. Be careful; the “Clean Man Pages” option may destroy man pages for which you have no source. admscr0.50.tgz: A system administration script with menus. Small, but it works nicely.

In short, be negative about things from time to time. Don't hesitate to say “This software is bad.”, or “386BSD is better than Linux in the following areas...” If we don't talk about what's wrong, things will never get fixed.Andrew Kuchling Hemmingford,Quebec CANADA

About additional topics that *LJ* should cover, Steve suggests

1. How to load Linux (SLS vs Slackware)
2. A tutorial on lilo and all the things you can do with it.
3. an analysis of how Linux stacks up to the POSIX standards (.1 and .2), BSD interface and SVID (2 or 3, or whatever the latest version is)
4. Updates on WIN3 emulation and threads support

Steve Zanoni,BrookField, WI

Archive Index Issue Table of Contents

Advanced search

What Is Linux

Phil Hughes

Issue #1, March 1994

In this first editorial I will attempt to give you an idea of what Linux is and what it will be. I also want to give you an idea of what (and who) *Linux Journal* is and what we see as our mission.

Linux is an independent implementation of the POSIX operating system specification (basically a public specification of much of the Unix operating system) that has been written entirely from scratch. Linux currently works on IBM PC compatibles with an ISA or EISA bus and a 386 or higher processor. The Linux kernel was written by Linus Torvalds from Finland, and by other volunteers. Many of the utilities are from the Free Software Foundation's GNU project. Add to this basic definition the fact that everything is essentially free (more on this later) and you have yourself a complete Unix-like operating system that can run on the average personal computer.

Although Linux can run on a 386SX based system with 2MB of RAM and 40MB of disk storage this won't show you the power of the system. The average system consists of 8MB of RAM and 300-1000MB of disk storage along with the usual array of peripheral equipment such as VGA monitors, keyboards, floppy disks and interface boards. Where do you get it? If you are connected to the Internet you can download it for free. With its current size, however, purchasing a packaged version may be a better alternative. Packaged versions of Linux are available on floppy disks, cartridge tape and CD-ROM and range in price from $30 upward depending on what is included.

Although the initial Linux effort was started as a more or less hobby project, commercial distributions and commercial use of Linux systems are becoming more common. It is expected that Linux will play an important role in the current computer industry trend of downsizing as a single Linux system running on standard PC hardware can act as a server for multiple systems as well as offering a low-cost alternative to proprietary workstations running the Unix operating system.

Most of the development of the Linux system has taken place in semi-public forums on the Internet where current readership of the Linux newsgroups is around 100,000. This effort has grown to this substantial size in only about 2 years. Over the next two years, we expect to see an amazing growth in both the number of Internet users and the number of Linux users. In fact, Linux will be the tool that many people choose in order to get themselves connected to the Internet.

Today Linux is primarily a Unix-like system. But an emulator exists that allows it to run MS-DOS based applications and an interface is currently being developed that will allow Linux to run Microsoft Windows applications directly using the capabilities of the X-windows system which comes with many Linux distributions. Thus, in the near future, Linux will offer both native applications and the capability to run many of the existing PC-based applications all on one platform.

Where does *Linux Journal* fit in?

Up to this point, most of the work on Linux has been done in a reasonably public fashion. Usenet newsgroups and various electronic mailing lists have been the place where design decisions are made, bugs are reported and new releases are documented. In fact, it is this open forum that has made if possible to develop such an amazing product by a team of volunteers spread around the globe in such a short time frame.

As the Linux user community continues to grow we see an amazing number of newcomers entering the Usenet discussions each month. Many of these newcomers are seeking answers the the eternal questions: what hardware do I need, where can I get a copy of Linux and why doesn't something work. I expect for every question asked on one of these newsgroups there are at least ten others that go unasked and probably unanswered.

This is one of the places where *LJ* fits in. We plan to become a resource where people can find the answers to questions. And being a magazine instead of a discussion group we can assemble the needed answers andmake them available in a form that can be accessed by more people in more locations. Yes, some people have laptops and cellular phones that they can take on the bus but it still remains easier to hand a friend a magazine than help him or her get up to speed on using Usenet news readers.

We see this part of our audience as being two groups. Lots of the current Linux users have worked professionally with Unix. The other segmentis the DOS user who wants to upgrade to a multi-user system. With a combination of tutorials and technical articles we hope to satisfy the needs of both these groups. The

second audience we want to address is the commercial user. As I have been talking to people about *LJ* I find that I need to be an evangalist. For example, I have talked to hardware vendors and manufacturers of diagnostic tools. Only one had heard of Linux but all of them are somewhere between interested and excited when I told them the Linux story.

Many people question how something can be “free” and still be able to make money change hands. The vendors understand. Getting more Linux systems out there means selling more computers, more hard disks, more ethernet cards, more communications boards and more consulting. For comparison, look at your CD collection. For most of us, the cost of the CD player is insignificant in comparison to the cost of the CDs we have purchased. Linux is like the CD player and will encourage people to buy what they need to support it.

Besides making money, Linux can offer better alternatives for the end user. For example, I did some consulting a few years ago for a small business. They needed a multi-user system with a database, word processing and some document layout software. And they had a small budget. The solution they got was a single 386-based system with a fewterminals running SCO Xenix, troff and a database. Today they need to add more horsepower and would like to network an MS-DOS based system to what they have.

The upgrade path for them is to purchase more computers, more copies of Xenix (or Unix), TCP/IP communications software, Network File System software and some Ethernet boards to connect everything together. In other words, spend more money than they did initially to add the functions they need.

If their system was based on Linux they would already have TCP/IP and NFS as it comes with Linux. And Linux is not licensed on a per system basisso they wouldn't need to purchase more licenses. Thus, for the cost of the computers and Ethernet boards they could expand their network. They could probably get the system they need for not much more than the cost of the system they had to settle for.

Where did *Linux Journal* come from?

About a year ago I was talking to some friends about the ideaof starting a free software magazine. After a few meetings we realized that “free” was a very arbitrary term and what was really needed was a magazine that looks for the best value in software for the consumer. We quickly realized this would be a huge effort and how the subcription cost would be high because it couldn't carry advertising if it was to be totally objective. After all, companies with the biggest advertising budget seldom have the lowest prices.

At that time I had started playing with Linux. Having run Unixon PCs for about 8 years I was interested to see what Linux had to offer. I quickly came to the conclusion that Linux was betterthan many commercial Unix systems and was rapidly growing intoan amazingly nice product. I mentioned the idea of a Linux magazine to the team I had been working with on the free software magazine expecting them to discount it. Instead virtually all of themthought it was a good idea.

I distributed a questionnaire at a local Linux meeting and alsoon the Internet. Again, feedback was very positive so we decided that *LJ* was the way to go. After a few disasters we managed to hook up with Bob Young, publisher of New York Unix, and you see the result. Where we go from here is up to you. Initial articles and columnsare based on the interests expressed in the questionnaires that were returned. I encourage you, the reader to write or e-mail me with what you want to see in *LJ*. Our goal is to further theinterests of the Linux community. You are that community so let us know what you want.

Linux Journal Review Team

Archive Index Issue Table of Contents

Advanced search