Advanced search

Linux Journal Issue #3/June-July 1994

Archive Index

Advanced search

The World Wide Web

Bernie Thompson

Issue #3, June-July 1994

Geneticists share genome data with colleagues. Fans of Anne Rice talk about her latest books. Hundreds of programmers team up to create a free Unix system called Linux.

By enabling individuals around the globe to communicate and cooperate, the Internet has sped up the pace of scientific innovation. With 20 million users and growing, it has created a culture based on instant information and hyperkinetic communication.

The challenge now is to expand the power of the Internet to a wider audience and make it more convenient for all. Rising to this challenge is a system called the World Wide Web. By bringing all of the Internet's old and new resources together, the Web stands to become the one simple, standard way to access all of the Internet's riches.

It stands to revolutionize the revolution.

What is the Web?

To picture the World Wide Web, imagine a page from a book. By pressing your finger on any of the words, you receive a new page with more detailed information about eh subject selected. The Web is like a huge book being constructed on the Internet.

Tens of thousands of these pages are already scattered around the world. They are created by experts and novices of all disciplines who use the Internet. The amount of information available is stunning: an encyclopedia, a dictionary, world maps, complete information on US government agencies, extensive Linux documentation, and much much more. It's an amazing source of information.

How does it Work?

The Web is really a collection of computers, hooked together over the Internet, that pass pages of information back and forth. A program called a Web browser is used to find and view information. When you select a word in your Web browser, a message goes out to another computer somewhere in the world. That computer will respond by sending back the page you requested.

The messages between computers are encoded in the form of a Universal Resource Locator. The URL describes what kind of information you want, and where to look for it. It's like a mail address for the Web.

The pages can contain text, images. sounds and more. A page can even contain fill-in-the-blank forms which you complete to send information off to another computer. A protocol called HTML describes, in general terms, how the text, images, and forms should be positioned on the page. Web browsers can use this general description to lay out pages in different ways. For example, a browser that works in text mode only can ignore all the images. Or an X Windows browser can shift the text and images of a page when the window is resized.

URLs describe the locations of a page, and HTML provides an adaptable description of the information. Together, they make the Web an extremely flexible system.

Why Use the Web?

To everyone familiar with Internet newsgroups, FTP sites, and gopher servers, the Web may not seem groundbreaking. But the impact is, in fact, dramatic.

The most important advantage is simplicity. To maneuver around libraries of information, you need only click the mouse to learn or key combinations to memorize.

The Universal Resource Locator also provides an easy way to access other forms of Internet information such as gopher and archie. This way, there is only one, simple interface used to access a multitude of resources. New users benefit most from this simplicity.

Unlike newsgroups, Web information stays around as long as the author, or anyone else, wants to keep it available. Unlike information available vie FTP, the information is cataloged and categorized, so it is much easier to find and access. Unlike gopher, the information is laid out like one gigantic book. You can read what you like, and select more detail only if you're interested.

Many have predicted that cyberspace would become a chaotic mess where information is impossible to find. The World Wide Web is a system designed to bring order to the chaos.

Linux and the Web

Linux is one of the best systems, commercial or otherwise, for access to the Web. Linux is a version of Unix, the native platform of the Internet. So the best Web tools are often available on Linux before DOS, Windows, or Macintosh.

Two of the most popular tools for the Web are Mosaic and lynx. They are Web browsers with different goals. Lynx is small and text-only. Mosaic is large and full-featured. To browse the wealth of information on the Web, just pick one, install, and wander off to explore the wonders of the Web.

Lynx will display the text of a page without any images or fancy fonts. It only requires the keyboard arrow keys for navigation. This simplicity makes it fast and flexible. Lynx uses the VT100 protocol, so people without access to X Windows can still use lynx, including users logging in to a Linux system remotely.

Lynx is miserly with disk space and memory. It will only take up 320K of disk and consume about 620K of RAM while running. This makes lynx perfect for 4MB Linux workstations.

Lynx is not as common as Mosaic, however. This is primarily because lynx doesn't support the colorful images and different fonts that make the Web so expressive. For these, Mosaic or some other graphical browser is required to experience the full richness of the Web.

Lynx is freely available. To get lynx for Linux, check the Linux Software Map. For lynx source code, FTP to ftp2.cc.ukans.edu.

In contrast to lynx stands Mosaic. Mosaic is probably the best Web browsers available, and it is certainly the most famous. It takes full advantage of the Web.

Mosaic is a graphical browser, so it will only run on systems with X Windows (or MS Windows or Macintosh). A number of companies such as Quarterdeck have licensed the free Mosaic for commercial use rather than write their own. Mosaic is truly a high=quality application. But the price for beauty is more memory and disk space. Mosaic consumes over 1.3MB of disk space, and will use about 2MB of RAM while running.

Linux lynx, Mosaic is freely available. This freedom is slightly restricted, however, because Mosaic uses Motif programming libraries. This means you cannot recompile Mosaic yourself without a license for the Motif libraries.

To get Mosaic, search for it in the Linux Software Map. A copy of Mosaic can be found at sunsite.unc.edu and most other Linux sites. Assuming you have X Windows, no special installation procedures are needed. Just prepare to be impressed by a very professional program.

Using Linux to Contribute to the Web

DOS, Windows, and Mac users can happily access the Web with a browser. But the extra dimension Linux adds is the ability to become part of the Web—the ability to make information available to others via a Web server.

Setting up a Web server is certainly more difficult than using a browser, but is still surprisingly easy. This ease of use allows a large number of Internet users to contribute to the Web project. And Linux makes a great platform for a Web server because of its small size and speed. With the growing body of Internet Linux users, the potential is enormous.

Httpd is a popular Web server available for Linux. Its creator, NCSA, is the same group that developed Mosaic.Httpd will consume about 200K RAM while running. The program itself uses very little disk space. Just allocate enough disk space to hold all the pages of Web information.

The burden of the Web on a Linux server was tested by creating a working Web site on a i386-33 Linux machine. During a 40 day span, the PSU Linux WWW received 5375 requests from 1328 different sites around the world. This is an average of 6.9 requests per hour and 165 per day. The Web requests never interfered with any work being done by other users at the machine. A Linux system can easily provide Web services and have horsepower to spare.

To get httpd and set up your own Web server, look for httpd in the Linux Software Map. For full documentation on httpd, look at http://hoohoo.ncsa.uiuc.edu/. This site has all the documentation needed for installation.

The Web Means Business

The Web has attracted a surprising amount of attention from the commercial world. This is a testament to its effectiveness. No other existing system has the clean design, flexibility and momentum that the Web enjoys. “It's the killer application of the Internet,” says Eamonn Sullivan of PC Week. “I know everyone says that now, but that doesn't make it any less true.”

PC Week Labs discovered Linux during the course of setting up a Web server. The result is they continue to use Linux on their server, and Linux won PCX Week's Product of the Week for April 18, 1994. Press exposure of this sort will inspire new and exciting business applications for Linux.

The number one buzzword in business today is client/server. The Web and Linux fit perfectly into such a system. A documentation solution for a large organization can be quickly and effectively developed with the Web.

For example, a business might be grappling with the documentation requirements of ISO 9000, the process quality standard of the International Standards Organization. Using the Web with Linux servers and clients running Mosaic, documents related to ISO 9000 can be sorted in one location, updated only by the group responsible for that process. But anyone inside the organization can view the documents. The Web is an easy, open solution to a thorny problem.

Exciting Possibilities for the Linux Community

Still, the greatest potential of the Web lies with the Internet-connected Linux community. As Linux continues to prove itself to Internet users, the audience for a Linux Web will grow. By distributing data around the world, enormous amounts of information can be conveniently accessed from any Linux machine. There are already a number of Web servers which have manual pages and info files on-line. Hook into the Web and save space.

The first possibility is for manual pages and info pages. This infrequently accessed information can be viewed via the Web instead of storing copies on every machine. There are already a number of Web servers which have manual pages and info files on-line. Hook into the Web and save space.

Using FTP to access files can be difficult for novice users. A Web browser provides a friendly interface for getting files from FTP sites. Common FTP sites can appear as links on a Web page. Newsgroups, also, can be accessed via the Web. Hypertext links between followups are automatically created.

Subscribing to mailing lists can be automated. Using fill-in forms, users could select links to subscribe and unsubscribe. In the same way, they could register with Linux User Counter.

To better bring the Linux community together, a Web server can be configured so that each user has a home page of information about themselves. A tree of Linux information can be developed, right down to individuals. New Linux users would register themselves, and their home page, with some local server. Every new local server would register itself with a central server. Now the location and interests of each Linux user are easily available.

Lastly, there is potential for information beyond mere man pages. Linux is extraordinary in the quantity and quality of online information available. Because of the contributions of groups like the Linux Documentation Project, information about nearly every aspect of Linux use are available. All of these manuals, information sheets, and FAQs can be made available on the Web.

What makes this possibility so exciting is that each manual can be stored in one place—the author's home site. So when a document is updated, the author has only one central location the change it. Although the documents would actually be scattered around the world, to the Web user they would all appear on one easy-to-locate Web page. The Web provides an optimal system for both authors and users.

The World Wide Web is still in its infancy. The first half of 1994 saw triple-digit growth in Web traffic. By encompassing older systems of information access, like gopher, the Web guaranteed instant compatibility.

Native Web information is exploding. Through the Internet and through CD-ROM distribution, the Linux community is finding many new and creative uses for this flexible technology. No doubt more and better uses will be forthcoming. It is certain that the phenomenal growth of the Web will continue.

Bernie Thompson ran the PSU Linux WWW during its 3-month life span. He can be reached at bjt105@psu.edu.

Bernie Thompson (bjt105@psu.edu) Bernie Thompson ran the PSU Linux WWW during its 3-month life span.

Tips for Optimizing Linux Memory Usage

Jeff Tranter

Issue #3, June-July 1994

In a previous issue, Jeff discussed ways to reduce disk space usage under Linux. In this sequel article, he shows some useful techniques for making the best use of another valuable resource—memory.

Introduction

Like most Unix-compatible operating systems, the single most important factor in determining the performance you get out of Linux is often the amount of physical memory available. This is often a source of confusion to users accustomed to other systems such as MS-DOS. Since many Linux users are on a tight budget, the option of simply purchasing more memory is not always feasible. This article presents some ways in which you can make better use of the memory you already have.

Background

Linux implements a demand-paged virtual memory system. Processes have a large (4 gigabyte) virtual memory space. As virtual memory is referenced, the appropriate pages are transferred between disk and physical memory.

When there are no more physical memory pages available, the kernel swaps some older pages back to disk. (If they are code pages that have not been changed, then they are just discarded; otherwise they are written to the swap areas.)

Disk drives are mechanical devices; reading and writing to disk is several orders of magnitude slower than accessing physical memory. If the total memory pages required significantly exceed the physical memory available, the kernel starts spending more time swapping pages than executing code. The system begins thrashing, and slows down to a crawl. If this increases to a point where the swap device becomes fully utilized, the system can virtually come to a standstill. This is definitely a situation we want to avoid.

When extra physical memory is not in use, the kernel attempts to put it to work as a disk buffer cache. The disk buffer stores recently accessed disk data in memory; if the same data is needed again it can be quickly retrieved from the cache, improving performance. The buffer grows and shrinks dynamically to use the memory available, although priority is given to using the memory for paging. Thus, all the memory you have is put to good use.

Tools for Measuring Memory Utilization

In order to know what your memory situation is and whether any changes you make are resulting in improvement, you need to have some way of measuring memory usage. What tools do we have at our disposal?

When the system first boots, the ROM BIOS typically performs a memory test. You can use this to identify how much physical memory is installed (and working) in your system, if you don't know already. On my system, it looks something like this:

```
ROM BIOS (C) 1990
008192 KB OK WAIT......
```

The next piece of useful information is displayed during the Linux boot process. Output such as the following should be displayed:

```
Memory: 7100k/8192k available (464k
kernel code, 384k reserved, 244k data) ...
Adding Swap: 19464k swap-space
```

This shows the amount of RAM available after the kernel has been loaded into memory (in this case 7100K out of the original 8192K). You can also see if the swap space has been properly enabled. If the kernel bootup messages scroll by too quickly to read, on many systems you can recall them at a later time using the “dmesg” command.

Once Linux is running, the “free” command is useful for showing the total memory available (which should match that shown during boot-up), as well as a breakdown showing the amount of memory being used, and the amount free. (If you don't have a “free” command, you can use “cat /proc/meminfo”.) Both physical memory and swap space is shown. Here is a typical output on my system:

Here is a typical output on my system:

total	used	free	shared	buffers
Mem:	7096	5216	1880	2328

Swap:	19464	0	19464	2800

The information is shown in kilobytes (1024 bytes). The “total” memory is the amount available after loading the kernel. Any memory being used for processes or disk buffering is listed as “used.” Memory that is currently unused is listed in the “free” column. Note that the total memory is equal to the sum of the “used” and “free” columns.

The memory indicated as “shared” is an indication of how much memory is common to more than one process. A program such as the shell typically has more than one instance running. The executable code is read-only and can be shared by all processes running the shell.

The “buffers” entry indicates how much of the memory in use is currently being used for disk buffering.

The “free” command also shows very clearly whether the swap space is enabled, and how much swapping is going on.

To better understand how the kernel uses memory, it is instructive to watch the output of the “free” command as the system is used. I'll show some examples taken from my own system; I suggest you try similar experimentation yourself.

On bootup, with one user logged in, my system reports the following:

```
        total   used    free    shared  buffers
Mem:    7096    2672    4424    1388    1136
Swap:   19464   0       19464
```

Note that we have considerable free memory (4.4MB) and a relatively small disk buffer (1.1MB). Now watch how the situation changes after running a command that reads data from the disk. (In this case I typed **ls -lR /**.)

```
        total   used    free    shared  buffers
Mem:    7096    5104    1992    1396    3460
Swap:   19464   0      19464
```

We see that the disk buffer has grown by over 2 MB. This brings the “used” memory up correspondingly, and the free memory down. Next, I start up the X Window system and examine the results:

```
        total   used    free    shared  buffers
Mem:    7096    7016    80      3112    3792
Swap:   19464   8       19456
```

This has caused the memory used to increase to 7MB, leaving only 80K free. The increase is to support the additional processes running (the X server, window manager, xterm, etc...). Note that the disk buffer didn't shrink, because there is still free memory. Remember: “free” memory means memory that is being wasted.

Now I start up the GNU chess program, having it play against itself. This starts two instances of a rather large program:

```
        total   used    free    shared   buffers
Mem:    7096    7016    80      1080    860
Swap:   19464   5028    14436
```

We see now that the disk buffer has shrunk down to less than 1MB and we are 5MB into swap to accommodate the large processes. Because of the swapping, the system has slowed down, and heavy disk drive activity can be heard. There is still a small amount of free memory. (The kernel tries to prevent user processes from taking all of the available memory; it reserves some for the “root” user only.)

The next step is to exit the X Window system and the applications running under it; here is the result.

```
        total   used    free    shared  buffers
Mem:    7096    2444    4652    412     1480
Swap:   19464   728     18736
```

We now have lots of free memory, the swap usage is almost gone (some idle programs are still presumably swapped out), and the disk buffer is starting to grow again.

The “top” and “ps” commands are also very useful for showing how memory usage changes dynamically, and how individual processes are using memory. For the scenario described earlier, we can see from the output of “ps” that each of the two chess processes was taking almost 8MB of virtual memory, obviously more than could fit in physical memory, causing the system to thrash.

```
USER  PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND ...
tranter   282  4.1 34.4 7859 2448 v01 D  14:08 0:11 gnuchessx 40 5
tranter   285  7.9 30.7 7859 2180 v01 D  14:09 0:21 gnuchessx 40 5
...
```

Another facility for getting system status information is built into the virtual console driver. This depends on your keyboard mapping, but the default for the US keyboard is to use the Scroll-Lock key. Pressing <Alt><Scroll Lock> shows the current value of the CPU registers. The <Shift><Scroll Lock> combination shows memory information, similar to the “free” command, but more detailed. Finally, <Ctrl><Scroll Lock> will give information on individual processes, much like the “ps” command.

These keys can be particularly handy if your system is slow, or appears to have crashed. Note that if you are running the syslog daemon, this information will probably be logged to a file instead of being displayed on the console. On my Slackware system for example, it is logged to the file /var/adm/syslog.

Increasing Available Memory

Now that we have some measurement tools at our disposal, its time to try to improve the memory situation. The first line of attack is before Linux boots—your ROM BIOS setup program has some options that may increase the amount of memory available. Many systems can shadow the ROM address ranges in RAM, because it is faster than ROM. Unlike MS-DOS, however, Linux doesn't use the ROM BIOS routines, so disabling this can free close to 200K of memory (if you still run MS-DOS occasionally then you may not want to do this).

Incidently, now is also a good time to look at your other setup options and do some experimentation. You may be able to improve CPU performance with the options to enable caching and setting the CPU clock speed. One way to measure this is to use the BogoMIPs rating displayed when Linux boots as an indicator of CPU speed (this is not always accurate though, because as everyone knows, BogoMIPs are “bogus”). If you boot Linux from a hard disk, you may also be able to speed up reboot times by disabling the floppy disk drive seek at bootup. Don't change too many settings at once, or you may not know which changes are having a positive effect. Be sure to write down your original settings in case you put your system in a state where it will no longer boot.

Recompiling the Kernel

Are you still using the default kernel that came when you installed Linux? If so, shame on you! Kernel memory is special—unlike the memory pages used by processes, the kernel is never swapped out. If you can reduce the size of the kernel, you free up memory that can be be used for executing user programs (not to mention reducing kernel compile times and disk storage).

The idea here is to recompile the kernel with only the options and device drivers you need. The kernels shipped with Linux distributions typically have every possible driver and file system compiled in so that any system can boot from it. If you don't have network cards, CD-ROM, SCSI, and so on, you can save considerable memory by removing them from the kernel. Besides, you can't really consider yourself a Linux hacker if you've never recompiled a customized kernel yourself.

If there are drivers that you only need occasionally, consider building several kernels, and set up LILO to let you choose an alternate kernel when booting. If you have a math coprocessor, you can consider taking out the FPU emulation routines as well. You can also remove any of the Linux file systems that you do not require.

More advanced Linux hackers might want to look at the “modules” facility which allows for loadable device drivers. With this you can dynamically add and remove drivers without rebooting. This facility has been available for some time to kernel hackers, and it has now become a part of the standard kernel. This facility is particularly useful for rarely used devices such as tape drives that are only occasionally used for backup purposes.

Finally, make sure you are running a recent kernel. Newer kernels, as well as (in most cases) being more stable, also have improvements in memory usage.

Compiling Applications

If you develop your own applications, or compile code you obtain from the Internet or bulletin board systems, then using the right compile options can reduce the memory used. Turning on optimization will generally produce code which is smaller and executes faster, as well as requiring less memory. A few optimizations, such as in-line functions, can make the code larger. You should also check that your executables are dynamically linked and stripped of debug information.

Which optimizations are best depend on the specific application and even on the version of compiler used; you may wish to experiment.

Reducing Memory Usage Further

Once Linux is up and running your new kernel, it's time to look at where the memory is going. Before you even log on, how many processes are running?

The bare minimum for a Linux system would typically be:

- init (this starts all other processes)
- update (this periodically writes the disk buffers to disk)
- a single getty (which becomes your shell when logged in)

Run “top” and see what is running on your system. How many getty processes do you need? Do you really need all those other processes such as lpd, syslogk, syslogd, crond, and selection? On a standalone system, you don't need to run full networking software.

If you are using an init package that supports multiple run levels, you might want to consider defining several different run levels. This way you could, for example, switch your system between full networking and running standalone, allowing you to free up resources when you don't need them.

You can also examine some of your larger executables to see if they were built with the appropriate compiler and linker options. To identify the largest programs, try using a command such as this:

```
  ls -s1 /bin /usr/bin /usr/bin/X11 |  sort -n | tail
```

Strictly speaking this only finds the largest files, but file size is usually a good indication of the memory requirements of a program.

The most common shell under Linux is GNU BASH. While very functional, it is also quite large. You can save memory by using a smaller shell such as the Korn shell (usually called ksh or pdksh).

The emacs editor is also big; you could use a smaller editor such as vi, jove, or even ed instead.

The X Window System

If you ran the command line described earlier, one of your largest binaries was probably the X server. The X Window system takes a lot of memory resources.

The first question to consider is, do you really need to run X? Using the virtual consoles and selection service you can have multiple windows supporting cut & paste of text using a mouse. Particularly while performing large compiles (such as the kernel), you should consider the option of simply not running X.

There is also a windowing system called “mgr” than can be used as an alternative to X, but requires less memory.

If you decide to use X, then you can obtain replacements for some of the standard tools that require less resources. “Rxvt” is similar to xterm, but requires significantly less memory. The window manager “fvwm” will also use less resources than others, and “rclock” is a small X-based clock program. These three tools, written by Robert Nation, can make running X feasible on a machine that constantly swapped before.

How many programs do you run on the X desktop? Run “top” to see how much memory is being taken by xclock, xeyes, xload, and all those other goodies you think you need.

The “Tiny X” package, put together by Craig I. Hagan, contains the Korn shell, fvwm window manager, rxvt, rclock, X server, and the minimum of other files needed to run X. The package is small enough to fit on one 3.5" floppy disk. Also included are some useful notes on saving memory under X.

With the techniques described here, you can run small X applications reasonably well on a machine with only 4 megabytes of memory. On machines with more memory, the same methods will allow you to run larger applications and free up memory to use for disk buffering.

Conclusions

By combining the techniques I've described, the net effect on system performance can be well worth the effort. I encourage you to experiment, and along the way you'll almost certainly learn something new.

For More Information

The software mentioned in this article is available on a number of Internet archive sites, including sunsite.unc.edu and tsx-11.mit.edu. I suggest getting a copy of the Linux Software Map to help track down the software you need.

If you want to learn more about how the Linux kernel implements memory management, check out ”The Linux Kernel Hackers' Guide“, by Michael K. Johnson, part of the Linux documentation project. Appendix A of that document includes an extensive bibliography of books covering operating system concepts in general.

”How to Maximize the Performance of X" is periodically posted to the Usenet newsgroup news.answers, and contains more ideas for improving X performance on small systems.

Sendmail+IDA

Vince Skahan

Issue #3, June-July 1994

A mail transport agent that combines advanced capabilities with easy configuration.

As a Linux (or Unix) user you are certainly familiar with the program you use to read your electronic mail. It may be mail, mailx, elm, mush or pine but it preforms the function of allowing you to access your mailbox in an orderly fashion. This program is called a mail user agent or MUA.

But, how does all that mail get in the mailbox? And when you send mail, how does it get routed properly? That is the job of the mail transport agent or MTA.

Introduction to Sendmail+IDA

It's been said that you aren't a REAL Unix system administrator until you've edited a sendmail.cf file. It's also been said that you're a crazy person if you've attempted to do so twice :-)

Sendmail is an incredibly powerful program. It's also incredibly difficult to learn and understand for most people. Any program whose definitive reference (Sendmail, published by O'Reilly and Associates) is 792 pages long quite justifiably scares most people off.

Sendmail+IDA is different. It removes the need to edit the always-cryptic sendmail.cf file and permits the administrator to define the site-specific routing and addressing configuration through relatively easy-to-understand 'tables'. Switching to sendmail+IDA can save you many hours of work and stress.

Compared to the other major mail transport agents, I've yet to find anything that you can't do faster and simpler with sendmail+IDA. Typical things needed to run a normal UUCP or Internet site are absolutely trivial to accomplish. Normally difficult configurations are simple to create and maintain.

At this writing, the current version of sendmail5.67b+IDA1.5 is available via anonymous ftp from ftp.uiuc.edu. It compiles without any patching required under Linux.

All the configuration files required to get sendmail+IDA sources to compile, install, and run under Linux are included in newspak-2.0.tar.gz which is available via anonymous ftp on sunsite.unc.edu in the directory /pub/Linux/system/Mail.

Sendmail+IDA Configuration Files - Overview

Traditional sendmail is set up through a system configuration file, typically /etc/sendmail.cf or /usr/lib/sendmail.cf, that is not anything close to any language you've seen before. Editing the sendmail.cf file to provide customized behavior can be a humbling experience.

Sendmail+IDA makes such pain essentially a thing of the past by having all configuration options table-driven with rather easy-to-understand syntax. These options are configured by running m4 (a simple macro processor) or dbm (a simple database processor) on a number of data files via Makefiles supplied with the sources.

The sendmail.cf file defines only the default behavior of the system. Virtually all special customization is done through a number of optional 'tables' rather than by directly editing the sendmail.cf file.

- mailertable - define special behavior for a remote host or domain
- uucpxtable - force UUCP delivery of mail to a domainized host
- pathtable - define the pathalias-style UUCP path to a remote host or domain
- uucprelays - short-circuit the pathalias path to well-known remote hosts
- genericfrom - convert internal addresses into generic ones visible to the outside world
- xaliases - convert generic addresses to/from valid internal ones
- decnetxtable - convert SMTP addresses to decnet-style addresses

The sendmail.cf file

Description

The sendmail.cf file for sendmail+IDA is not edited directly, but is generated based on an administrator-specified m4 configuration file. This file creates a few definitions and otherwise points to the tables where the 'real work' gets done. In general, it is only necessary to specify the paths used on the local system, the name(s) the site is known by for e-mail purposes, and which default mailer (and perhaps smart relay host) is desired.

There are a large variety of parameters that can be defined to establish the behavior of the local site or to override compiled-in configuration items. These configuration options are identified in detail in the documentation that comes with the sources in the file <IDA_SOURCE_DIR>/ida/cf/OPTIONS.

Example

A m4 file for a minimal configuration (UUCP or SMTP with all non-local mail being relayed to a directly connected smart-host) can be as short as 10 or 15 lines excluding comments.

A typical sendmail.m4 file for a UUCP-only site who talks to an Internet relay host is shown below.

Virtually all systems should set the DEFAULT_HOST, DEFAULT_MAILER, and PSEUDONYMS.

UUCP hosts will probably also need to define the UUCPNAME, RELAY_MAILER, and RELAY_HOST parameters.

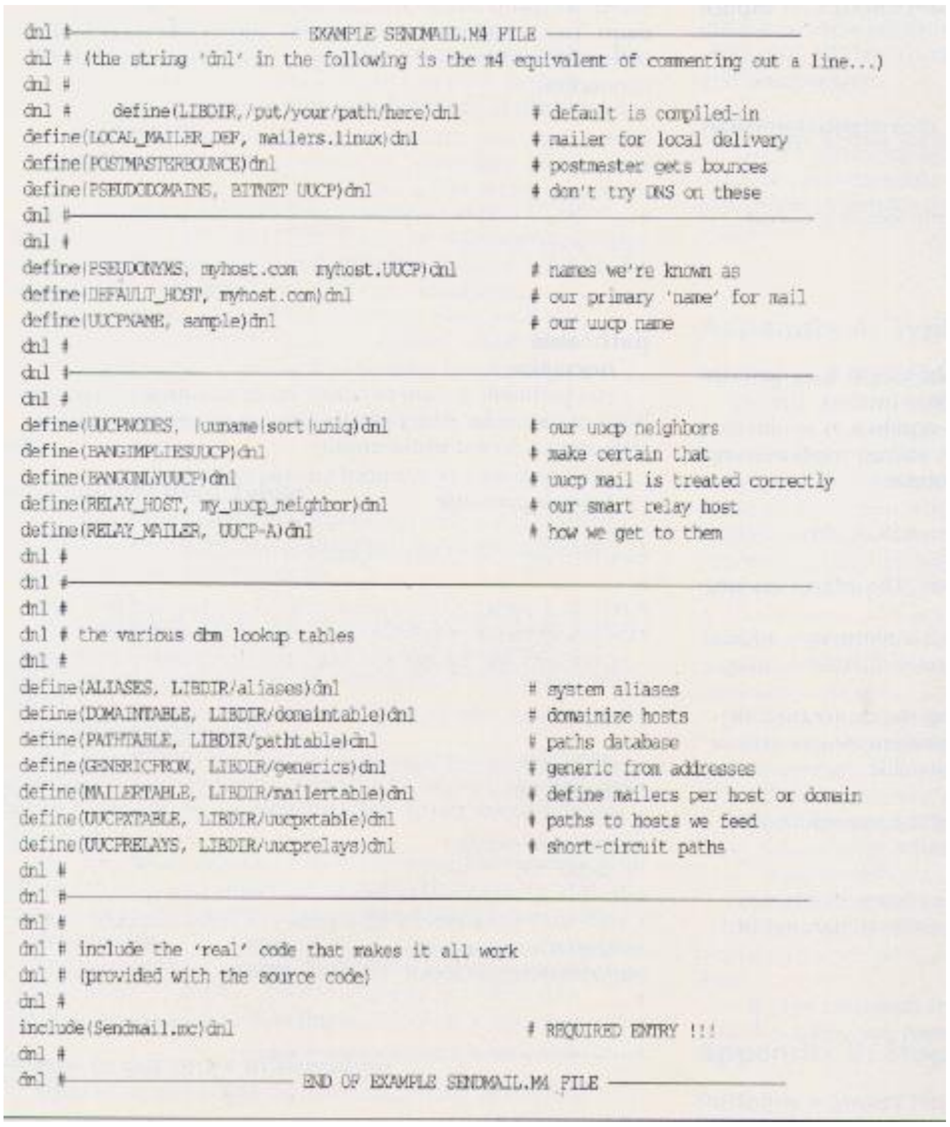
If your site is SMTP-only and talks 'Domain Name Service', you would change the DEFAULT_MAILER to TCP-A and probably delete the RELAY_MAILER and RELAY_HOST lines.

See Figure 1

The mailers.linux LOCAL_MAILER

Most operating systems provide a program to handle local delivery of mail. Typical programs for the major variants of Unix have defaults in the sendmail+IDA binary. In Linux, it is necessary to explicitly define the appropriate local mailer since a local delivery program is not necessarily present in the distribution you've installed. This is done by specifying the LOCAL_MAILER_DEF in the sendmail.m4 file.

The following example is how you would set the local mail delivery program to be the commonly available (and ported to Linux) program 'deliver' to provide this function.



```
dnl #------------------- EXAMPLE SENDMAIL.M4 FILE -------------------
dnl # (the string 'dnl' in the following is the m4 equivalent of commenting out a line...)
dnl #
dnl #    define(LIBDIR,/put/your/path/here)dnl         # default is compiled-in
define(LOCAL_MAILER_DEF, mailers.linux)dnl               # mailer for local delivery
define(POSTMASTERBOUNCE)dnl                              # postmaster gets bounces
define(PSEUDODOMAINS, BITNET UUCP)dnl                    # don't try DNS on these
dnl #---------------------------------------------------------------------
dnl #
define(PSEUDONYMS, myhost.com  myhost.UUCP)dnl           # names we're known as
define(DEFAULT_HOST, myhost.com)dnl                      # our primary 'name' for mail
define(UUCPNAME, sample)dnl                              # our uucp name
dnl #
dnl #---------------------------------------------------------------------
dnl #
define(UUCPNODES, |uuname|sort|uniq)dnl                  # our uucp neighbors
define(BANGIMPLIESUUCP)dnl                               # make certain that
define(BANGONLYUUCP)dnl                                  # uucp mail is treated correctly
define(RELAY_HOST, my_uucp_neighbor)dnl                  # our smart relay host
define(RELAY_MAILER, UUCP-A)dnl                          # how we get to them
dnl #
dnl #---------------------------------------------------------------------
dnl #
dnl # the various dbm lookup tables
dnl #
define(ALIASES, LIBDIR/aliases)dnl                       # system aliases
define(DOMAINTABLE, LIBDIR/domaintable)dnl               # domainize hosts
define(PATHTABLE, LIBDIR/pathtable)dnl                   # paths database
define(GENERICFROM, LIBDIR/generics)dnl                  # generic from addresses
define(MAILERTABLE, LIBDIR/mailertable)dnl               # define mailers per host or domain
define(UUCPXTABLE, LIBDIR/uucpxtable)dnl                 # paths to hosts we feed
define(UUCPRELAYS, LIBDIR/uucprelays)dnl                 # short-circuit paths
dnl #
dnl #---------------------------------------------------------------------
dnl #
dnl # include the 'real' code that makes it all work
dnl # (provided with the source code)
dnl #
include(Sendmail.mc)dnl                                  # REQUIRED ENTRY !!!
dnl #
dnl #------------------- END OF EXAMPLE SENDMAIL.M4 FILE -------------------
```

EXAMPLE

There is a also built-in default for 'deliver' in the sendmail+IDA Sendmail.mc file that gets included into the sendmail.cf file. To specify it, you would not need a mailers.linux file and would instead define the following in your sendmail.m4 file:

```
dnl -- (in sendmail.m4) --
define(LOCAL_MAILER_DEF, DELIVER)dnl
# mailer for local delivery
```

Unfortunately, Sendmail.mc assumes deliver is installed in /bin, which is not the case with Slackware1.1.1 (which installs it in /usr/bin). In that case you'd need to either fake it with a link or rebuild deliver from sources so that it resides in /bin.

Sendmail+IDA dbm Tables

Sendmail+IDA provides a number of tables to provide the ability to override the default behavior of sendmail (specified in the sendmail.m4 file) and define special behavior for unique situations, remote systems, and networks. These tables are post-processed with dbm using the provided Makefile.

Most sites will need few, if any, of these tables. If your site does not require these tables, the easiest thing is probably to make them zero length files (with 'touch') and use the default Makefile in $LIBDIR rather than editing the provided Makefile.

A generic site that is on Internet and speaks Domain Name Service (or one that is UUCP-only and forwards all mail via UUCP through a smart RELAY_HOST) probably does not need any specific table entries at all.

mailertable

Description

The mailertable defines special treatment for specific hosts or domains based on the remote host or network name.

It is frequently used on Internet sites to use a particular protocol (uucp/smtp) to forward to an intermediate mail relay host or gateway in order to reach a remote network.

UUCP sites will generally not need to use this file.

Order is important. Entries match based on a top-down interpretation of the rulesets so it is generally wise to place the most explicit rules at the top of the file and the more generic rules below.

Example

Suppose you want to forward all mail for a mythical JoeUniversity via UUCP to a relay host 'sysA'. To do so, you would have a mailertable entry that looked like the following:

```
# (in mailertable)
#
# forward all mail for the domain .joe-u.edu via uucp to sysA

UUCP-A,sysA .joe-u.edu
```

Suppose you want all mail to the larger .edu domain to go to a different relayhost 'sysB' for address resolution and delivery. The expanded mailertable entries would look quite similar.

```
# (in mailertable)
#
# forward all mail for the domain .joe-u.edu via uucp to sysA
```

```
UUCP-A,sysA .joe-u.edu
#
# forward all mail for the domain .edu via uucp to sysB
UUCP-A,sysB .edu
```

As mentioned above, order is important. Reversing the order of the two rules shown above will result in all mail to joe-u.edu going through the more generic 'sysB' path through the explicit 'sysA' path that is really desired.

```
# (in mailertable)
#
# forward all mail for the domain .edu via uucp to sysB

UUCP-A,sysB .edu

#
# no mail for joe-u will go through sysA because the above
# rule was matched and used by sendmail

UUCP-A,sysA .joe-u.edu

#
```

Mailertable Format

In the mailertable examples above, the UUCP-A mailer means use UUCP delivery with domainized headers. The comma between the mailer and remote system tells sendmail to merely forward the message to 'sysA' for address resolution and delivery. Mailertable entries are of the format:

```
MAILER DELIMITER RELAYHOST HOST_OR_DOMAIN
```

There are a number or possible mailers. The differences are generally in how they treat addresses.

Typical mailers are TCP-A (tcp/ip with Internet-style addresses), TCP-U (tcp/ip with UUCP-style addresses), UUCP-A (uucp with Internet-style addresses).

The choice of the character separating the mailer from the host portion on the left-hand-side of a mailertable line defines how the address is modified by the mailertable.

! - (exclamation point) means strip off the recipient hostname before forwarding to the mailer

, - (comma) means do not change the address in any way. Merely forward via the specified mailer to the specified relay host

: - (colon) means remove the recipient hostname only if there are intermediate hosts between you and the destination

uucpxtable

Usually, mail to hosts with fully-qualified-domain-names is delivered via Internet-style (SMTP) delivery based on the domain name server (DNS) configuration. The uucpxtable forces delivery via UUCP routing by converting the domainized name into a UUCP-style un-domainized remote hostname.

It is frequently used when you're a MX forwarder for a site or (sub)domain or when you wish to send mail via a direct and reliable UUCP link rather than potentially multiple hops through the default mailer and any intermediate systems and networks.

UUCP sites that talk to UUCP neighbors who use domainized mail headers would use this file to force delivery of the mail through the direct UUCP point-to-point link between the two systems rather than using the less direct route of through the RELAY_MAILER and RELAY_HOST or through the DEFAULT_MAILER.

Internet sites who do not talk UUCP probably would not use the uucpxtable.

Example

Suppose you provide MX forwarding service to a system called 'foo.bar.com' in DNS and 'foobarcom' in the UUCP maps. You would need the following uucpxtable entry to force incoming mail for their host to go through your direct UUCP connection.

```
#======= /usr/local/lib/mail/uucpxtable  ==========
#Mail sent to joe@foo.bar.com is rewritten to foobarcom!joe and
#therefore delivered via UUCP
#
foobarcom foo.bar.com
#
#-------------------------
```

pathtable

Description

The pathtable is used to define explicit routing to remote hosts or networks. The pathtable file should be in pathalias-style syntax, sorted alphabetically.

Most systems will not need any pathtable entries.

Example pathtable

```
#======== /usr/local/lib/mail/pathtable ==========
#
# this is a pathalias-style paths file to let you kick mail to
# uucp neighbors to the direct uucp path so you don't have to
# go the long way through your smart host that takes other traffic
#
```

```
# you want real tabs on each line or m4 might complain...
#
# pathalias-style routing through a system
foo!bar!%s bar
#
# mixed mode address
foo!%s@bar.com foo
#
#
# all mail for a network to a gateway (see the leading '.' ?)

%s@gateway.host.name.domain .UUCP
relayhost!%s@othergate.domain .BITNET
#
#
#============ end of pathtable ===============
```

domaintable

Description

The domaintable is generally used to force certain behavior to occur after a DNS lookup has occurred. It permits the administrator to make shorthand names available for commonly referenced systems or domains by replacing the shorthand name with the proper one automatically. It can also be used to replace incorrect host.domain information with 'correct' information.

Most sites will not need any domaintable entries.

Example

```
#========= /usr/local/lib/mail/domaintable =======
#
#replace a wrong domain people are mailing to with the correct one
#
brokenhost.correct.domain brokenhost.wrong.domain
#
#
#============ end of domaintable =============
```

aliases

Aliases permit a number of things to happen:

- provide a shorthand or well-known name for mail to be addressed to in order to go to one or more persons
- invoke a program with the mail message as the input to the program
- send mail to a file

All systems require aliases for Postmaster and MAILER- DAEMON to be RFC-compliant. Always be extremely aware of security when defining aliases that invoke programs or write to programs since sendmail generally runs setuid-root.

Changes to the aliases file do not take effect until the '/usr/lib/sendmail -bi' command is executed to build the required dbm tables. This can also be done by executing the 'newaliases' command, usually from cron.

Details concerning mail aliases may be found in the aliases(5) manual page.

Infrequently Used Tables

The following tables are available, but rather infrequently used. Consult with the documentation that comes with the sendmail+IDA sources for details.

uucprelays

The uucprelays file is used to 'short-circuit' the uucp path to especially well-known sites rather than using a multi-hop or unreliable path generated by processing the UUCP maps with pathalias.

genericfrom and xaliases

The genericfrom file hides local usernames and addresses from the outside world by automatically converting inside usernames to generic 'From' addresses that do not match internal usernames.

The associated 'xalparse' utility automates the generation of the genericfrom and aliases file so that both incoming and outgoing username translations occur from a master xaliases file.

decnetxtable

The decnetxtable rewrites domainized addresses into decnet-style addresses much like the domaintable rewrites undomainized addresses into domainized SMTP-style addresses.

Where to get more information

There are many places (see the Linux MAIL HOWTO in comp.answers or on rtfm.mit.edu for a list), but the definitive place is in the sendmail+IDA sources. Look in the directory <IDA_SRC_PATH>/ida/cf for the files DBM-GUIDE, OPTIONS, and Sendmail.mc.

aliases example

Acknowledgements

Thanks go to Neil Rickert and Paul Pomes for lots of help over the years regarding the care and feeding of sendmail+IDA and to Rich Braun for doing the initial port of sendmail+IDA to Linux.

Appendix A: Typical Problems

Mixing and Matching Binary Distributions

There is no “true standard configuration” of electronic mail transport and delivery agents and there is no “one true directory structure”.

Accordingly, it is necessary to ensure that all the various pieces of the system (Usenet news, mail, tcp/ip) agree on the location of the local mail delivery program (lmail, deliver, etc.), remote mail delivery program (rmail), and the mail transport program (sendmail or smail). Such assumptions are not generally documented, although use of the 'strings' command can help determine what files and directories are expected. The following are some problems we've seen in the past with some of the commonly available Linux binary distributions and sources.

- Some versions of the NET-2 distribution of tcp/ip have services defined for a program called 'umail' rather than sendmail.
- There are various ports of Elm and mailx that look for a delivery agent of /usr/bin/smail rather than sendmail.
- Sendmail+IDA has a built-in local mailer for 'deliver', but expects it to be located in /bin rather than the more typical Linux location of /usr/bin.

Rather than go to the trouble of building all the mail clients from sources, we generally fake it with the appropriate soft links...

Appendix B: Stupid Mail Tricks

Defining a Smart Host and Mailer

You set the default smart host via the RELAY_HOST and RELAY_MAILER parameters in the sendmail.m4 file that is processed into sendmail.cf.

To forward mail to a particular host or domain to a designated relay system, generally use the mailertable.

For example, to forward mail for relayhost.com to their uucp gateway system 'uucpgate'.

(in mailertable)UUCP-A,uucpgate relayhost.com

Forcing Mail into Misconfigured Remote Sites

Frequently, Internet hosts will have trouble getting mail into misconfigured remote sites. There are several variants of this problem, but the general symptom is that mail is bounced by the remote system or never gets there at all.

These problems can put the local system administrator in a bad position because your users generally don't care that you don't personally administer every system worldwide (or know how to get the remote administrator to fix the problem). They just know that their mail didn't get through to the desired recipient on the other end and that you're a likely person to compain to.

A remote site's configuration is their problem, not yours. In all cases, be certain to NOT break your site in order to communicate with a misconfigured remote site. If you can't get in touch with the Postmaster at the remote site to get them to fix their configuration in a timely manner, you have two options.

It is generally possible to force mail into the remote system successfully, although since the remote system is misconfigured, replies on the remote end might not work...but then that's the remote administrator's problem.

You can fix the bad headers in the envelope on your outgoing messages only by putting a domaintable entry for their host/domain that results in the invalid information being corrected in mail originating from your site:

```
   # (in domaintable)
   braindead_site.correct.domain.com braindead_site.wrong.domain.com
```

or.....

Treat them as totally brain-dead and strip off all hostname.domain information in the envelope of messages to them from your site.

The '!' in the following results in mail being delivered to their remote site and appearing to be locally originated (for sendmail purposes). The return address for your site will not be changed, so the proper return address will still show up in the message.

```
   # (in mailertable)
   TCP!braindead_site.correct.domain.com braindead_site.wrong.domain.com
```

Regardless, even if you get mail into their system, there is no guarantee that they can reply to your message (they're broken, remember...) but then their users are yelling at their administrators rather than your users yelling at you.

Forcing Delivery to a Remote System via UUCP or SMTP

Without use of any of the optional DBM tables, sendmail+IDA delivers mail via the DEFAULT_MAILER (and possibly RELAY_HOST and RELAY_MAILER) in the m4 file used to generate sendmail.cf. It is easily possible to override this behavior through entries in the domaintable or uucpxtable.

Forcing mail to be transferred via UUCP

In an ideal world (from the Internet perspective), all hosts have records in the Domain Name Service (DNS) and will send mail with fully qualified domain names.

If you happen to talk via UUCP to such a site, you can force mail to go through the point-to-point direct UUCP connection rather than through your default mailer by essentially 'undomainizing' their hostname through the uucpxtable. The result is that sendmail will then determine (via UUCPNODES in the sendmail.cf m4 file) that you are directly connected to the remote system and will queue the mail for delivery with UUCP.

```
   # (in the uucpxtable)
   # un-domainize sys2.com to force UUCP delivery
   sys2 sys2.com
```

Preventing mail from being delivered via UUCP

The opposite condition also occurs. Frequently, systems may have a number of direct UUCP connections that are used infrequently or that are not as reliable and always available as the default mailer or relay host.

For example, in the Seattle area there are a number of systems that exchange the various Linux distributions via anonymous uucp when the distributions are released. These systems talk UUCP only when necessary, so it is generally faster and more reliable to send mail through multiple very reliable hops and common (and always available) relay hosts rather than through unreliable direct point-to-point UUCP links.

It is easily possible to prevent UUCP delivery of mail to a host that you are directly connected to. If the remote system is domainized, you can add an entry in the domaintable that will fully domainize the hostname, and prevent a match by the UUCPNODES line in the sendmail.cf m4 file. The result is generally that mail will go via the relay_mailer and relay_host (or default_mailer).

```
   # (in domaintable)
   # prevent mail delivery via uucp to a neighbor
   uucp_neighbor.domain.com uucp_neighbor
```

Running the Sendmail Queue on Demand

To process queued messages immediately, merely type '/usr/lib/runq' which will invoke sendmail with the appropriate options to cause sendmail to run through the queue of pending jobs immediately rather than waiting for the next scheduled run.

Building and testing sendmail.cf

- cd to $LIBDIR/CF (generally /usr/local/lib/mail/CF)
- copy the example.m4 file to yourhostname.m4
- edit it to do the right thing (set your relay, hostname,pseudonymns, etc.)
- 'make yourhostname.cf'
- test that dude...

  ```
  /usr/lib/sendmail -bt -Cyourhostname.cf
  ```

  at the '>' prompt, try:

  ```
  "3,0 me"
  "3,0 my-uucp-neighbor!foo"
  "3,0 me@mynode.mydomain"
  "3,0 mynode!me"
  "3,0 me@somenode.com"
  ```

  (all should "do the right thing" hopefully)
- put it into place as /etc/sendmail.cf

start up sendmail as a daemon

```
/usr/lib/sendmail -bd -q1h
```

put the above line in the /etc/rc.local file so sendmailstarts up routinely when the system boots.

Reporting Mail Statistics

Sendmail comes with a utility called 'mailstats' that reads a file called /usr/local/lib/mail/sendmail.st file and reports the number of messages and number of bytes transferred by each of the mailers used in the senmail.cf file. This file must be created by the local administrator manually for sendmail logging to occur. The running totals are cleared by removing and recreating the sendmail.st file. One way is to do the following:

```
cp /dev/null /usr/local/lib/mail/sendmail.st
```

Probably the best way to do quality reporting regarding who uses mail and how much volume passes to, from, and through the local system is to turn on mail debugging with syslogd. Generally, this means running the /etc/syslogd daemon from your system startup file with a line in /etc/syslog.conf that looks something like the following:

```
mail.debug /usr/adm/syslog.mail
```

The sendmail.st file does not grow enough to be a concern. If you use mail.debug and get any medium to high mail volume, the syslog output can get quite large. Output files from syslogd generally need to be rotated or purged on a routine basis from cron.

There are a number of commonly available utilities that can summarize the output of mail logging from syslog. One of the more well known utilities is the syslog-stat.pl file that is distributed with the sendmail+IDA sources.

Archive Index Issue Table of Contents

Advanced search

Interview with Fred van Kempen

Phil Hughes

Issue #3, June-July 1994

Fred van Kempen is tha author of the “net2” networking code and is currently co-owner of a startup called ARIS that is putting together a commercial-quality Linux-based system. The following interview was conducted via e-mail in late April.

Linux Journal: Lots of people know you for your code but there must be another side to you. What do you do for fun?

Fred: Apart from catching up my reading and sleeping every now and then (I tend to forget doing both of these...) hacking, listening to music a lot, and attending to family matters pretty much fill up my days. Hmm, perhaps people will now understand why my computers are named “metallica”, “bonjovi” and such? (smile)

Linux Journal: When did you get interested in computers? What inspired it?

Fred: In 1984 or so. Our school has this huge IBM (The Computer), which had all of our grades and such stuffed away. I then decided I'd be a computer technician one day...

Linux Journal: When did you first start working with Linux?

Fred: Uhm, in October 1991. I just switched from a 20MHz 286 to a 25MHz 386DX, and a giant hard disk crash terminated the Minix system that was running on it. I then tried a number of UNIXes on that machine, and ultimately a friend dropped some disks with the MCC 0.96c release. It ran pretty well, and was quite fast, even on the slow 25MHz CPU.

I had seen systems like 386BSD crawl on it, so...

Then, after installing stuff like news and such (all ports from the Minix system, as I got to like that stuff for its being small and simple) I started reading news again, and found that someone was about to release TCP/IP for Linux. This was just what I was doing with Minix at the time, and I decided to stick with Linux at least until after I checked the TCP/IP stuff. That, by the way, was Linux 0.98.

***Linux Journal*:** For those unfamiliar with your contribution to the Linux effort, describe what you have done.

Fred: Well, mostly system cleanups (the Linux kernel structure has come a long way since 0.95 or so..) and of course the networking code. After some ugly flamish messages on the network, the original Linux NET author decided to quit the project, and I took over. Since then, the Linux NET code has been vastly improved, and it made Linux a real operating system, capable of competing with other UNIX-like systems like 386BSD and such.

***Linux Journal*:** Why did you decide to do it?

Fred: I have always been doing network-related things, even in the era in which I was programming PDP-11 computers in assembly. Taking over the NET effort was natural, as I was already rewriting major parts of it, with in mind a system, compatible with “existing standards” as much as possible.

***Linux Journal*:** Did you do the work alone or did others get involved?

Fred: I started out on my own, but later on several people joined the team. Of course, having multiple minds work on code is usually good for the code, because each person has a different view. I think the current NET development team is one of the better network programming teams around.

***Linux Journal*:** Who are the others? Does each person have a specific task or area of expertise?

Fred: The others are:

Laurence Culhane: SLIP driver author, pre-alpha tester (mycroft@holmes.demon.co.uk)

Mattew Dillon: Tester, “skb” code debugger, general race-condition tracer, tidbits-tracer (dillon@apollo.west.oic.com)

Johannes Stille: TCP-hacker. Johannes spent a great deal of time on improving the TCP module, and the results are very good. Also, a lot of tracing, testing, etc. (stille@titan.westfalen.de)

Terry Dawson: Testing, especially on slow (HAM) links which can cause very funny (ahem) situations. (terryd@extro.ucc.su.oz.au)

Fred Baumgarten: Testing, race-conditions, Netstat system, HAM link testing. (dc6iq@insu1.etec.uni-karlsruhe.de)

Danny ter Haar: Testing the code, often with fatal results for his disks. We had some fun during the first set of “code cleanups” ... ;-)

***Linux Journal*:** You are a ham, right? Does your Linux work and being a ham fit together?

Fred: Yes, I recently passed my US HAM test. Well, being someone highly interested in using and improving the art of digital communications over radio links, doing this on a system for which networking is natural is only a plus. The latest-and-greatest release of Linux Networking was designed to fit all kinds of networking experiments in an easy-to-implement way. As such, using Linux for HAM-like activities (meaning sending and receiving mail, talking to other people in general, exchanging knowledge and such) feels good. It's what made Linux in the first place...

***Linux Journal*:** Does Linux work with AX.25 (amateur packet) code?

Fred: Yes. Right now, there are three ways to connect HAM equipment to a Linux machine:

1. Use the Linux version of the NOS software, either the WAMPES release, or the JNOS/Linux release. Personally, I prefer the JNOS code. This means running a user-level program, which handles all the ugly details. No kernel code is required.
2. Use the AX.25 kernel code currently developed by Alan Cox. This code works fine, but needs a modified kernel to run.
3. Use the new code I am writing for Linux. This is a long-term project, and right now, is only of interest for people who are willing to spend a lot of time re-implementing a set of horrible protocols. <laugh>

***Linux Journal*:** Do you see a future for you and Linux?

Fred: Well, I certainly would hope so. I am co-owner of a US company which is trying to introduce Linux in the commercial market.

ARIS Technology, Inc (we usually call it ARIS, or even just ATI) is a small company located at the Technology Park of the Mississippi State University in Starkville, MS. It was founded by Dr. Michael L. Stokes, computer programmer and researcher at MSU's “Engineering for Research” (ERC) facility, and later I joined him.

ATI's short term plans are to consult with medium to large companies interested in bringing Internet into their business. We use Linux to do this when PC's will do the job because it will save money. As part of this plan, we will offer turn-key Linux systems through a local (Starkville) computer dealer. Longer term plans are to offer a commercial version of Linux based on Linux/PRO in CDROM, tape and diskette distributions.

This OS will offer a good stable platform for other commercial software vendors to support, and will support licensed software such as Motif 2.0, WordPerfect, etc. The distribution will be mostly licensed under GPL, though some custom desgined software will carry a license similar to SUN's (source code available free to licensed users for personal non-commercial use).

(ATI is reachable through the Internet at ARIS.COM. My address there is waltje@ARIS.COM, Dr. Stokes' address there is stokes@ARIS.COM.)

One of the advantages to this method of Linux availability could be that the commercial software developers, with all their great but usually DOS or “commercial UNIX”-based applications, may notice our efforts, and get interested in Linux as a platform to which they can port their products. Also, the fact that Linux is cheap has pushed many organizations and companies into installing it for business, fun or semi-personal use. After doing that, they usually find that a UNIX-like system, however cheap, needs a system administrator to keep it running smoothly. I think Linux created a lot of jobs in these fields, which is good...

***Linux Journal*:** Do you have a vision of what the Linux community will look like in a few years?

Fred: Depending on what some of the recent Linux projects produce (I am talking about ELF/COFF, iBCS and Wine here), Linux users will either continue to be semi-hackers like now, who use Linux for fun (as there won't be any real applications) or standard things (mail, news etc.) or there may come some sort of shift towards the “business” and “semi-business” class of users, who use Linux systems for applications, like they would use a “any-commercial-UNIX-here” now. Right now, it is hard to say what will happen.

Personally, I think the Linux community will have to get used to (a) paying some money for the software they use (for example, shareware and commercial applications), and (b) a somewhat more closed development environment of the system itself. Many, many people will disagree, and exactly this program is what is keeping Linux from a major breakthrough in The Real World.

This really is a shame, because Linux has proved the fact that expensive, commercial software doesn't always do a job better than a free system. The only thing keeping the commercial systems ahead is the fact that they have the support of the third-party application developers, which Linux does not have.

***Linux Journal*:** Thanks for taking the time to do this interview.

Archive Index Issue Table of Contents

Advanced search

UniForum 1994

Phil Hughes

Issue #3, June-July 1994

The week of March 21-25 I attended the UniForum show in San Francisco. At that show I played the multiple roles of press person, *Linux Journal* booth staffer, Linux evangelist and general-purpose nerd. It was an exciting show for me and for Linux.

For those unfamiliar with UniForum, it is an annual Unix extravaganza with hundreds of vendor exhibits, a host of tutorials and other seminars and tens of thousands of attendees. While some attendees are casually dressed, it is becoming more the “suit and tie” Unix show.

The show is put on by the UniForum Association, a nonprofit, vendor-independent association of developers, vendors and end users. The association has been around since 1981 and is in the business of promoting open systems as well as providing leadership in cooperation and standards and education.

Our Booth

The *Linux Journal*/New York Unix booth was busy all three days. In fact, the people in the next booth came over and mentioned how our booth had much more traffic than the other booths in the area. Then they started asking questions about porting their product (CD-ROM-mastering software) to Linux.

Much as in SSC's booth at the University of Washington computer fair where I spent a day two weeks earlier, most people came over to talk about Linux rather than ask what it was. Some Linux-related vendors dropped by including James Vera of Fintronic and Adam Richter of Yggdrasil.

Lots of Linux users dropped by and told us about their Linux systems and experiences. Many non-Linux users, after talking to us about Linux, headed upstairs to Computer Literacy's bookstore to purchase a Linux CD-ROM.

Of the vendors that stopped by, a group of people from Compaq was the most interesting. They showed up with one of their laptops with Linux on it. Certainly not difficult to do but it was real nice to see that a hardware manufacturer had recognized Linux as something real. When they offer their computers with Linux as an alternative we will know they are serious.

Dion Johnson of developer relations for SCO also dropped by. He told me he was glad we were here, as not everyone was going to run SCO Unix. Gee, what could I say. :-)

The Show

In my evangelist role I walked around the show talking to vendors about Linux. My primary target was communications board manufacturers. Virtually all of them had heard of Linux and I did get contact names at a couple of the companies so I could talk further about Linux and intelligent comm boards. One vendor, Cyclades, gave one of their semi-intelligent comm boards to a Linux user on the condition that he write a Linux driver for it. I arranged the deal, adding the condition that he write an article about it for *Linux Journal*.

Grove's Keynote

I attended Andrew Grove's keynote address on Wednesday morning. Andy is the CEO of Intel. I found it interesting that the CEO of a company that grew on the marketing of MS-DOS was giving a keynote at a Unix-based convention. It certainly reinforced the respectability of Unix as a real operating system with a real market.

Grove said “Unix has gone from challenger to defender without having ever been declared the winner”. This made me think about my 15 years working with Unix. My perception was that Unix was pretty mainstream but I live in Seattle which makes me think that jeans and birkenstocks are mainstream dress. But, if the CEO of Intel also feels that Unix won the OS battle maybe it really did.

He went on to say that Windows NT is the big challenger and that we need to get our collective acts together if we want to be successful in maintaining our position. Most of this responsibility is, of course, with the Suns and HPs and IBMs to cooperate on presenting a unified Unix. But, the Linux community can play an important part. If we can offer a reliable low-cost solution we can head off the move to another operating system.

Dennis Ritchie

On Friday, Dennis Ritchie gave a keynote address. The title was “Building the Operating System of the Future: How the Lessons of Unix are Shaping Plan 9”.

Ritchie, along with Ken Thompson, developed the Unix operating system 25 years ago. Ritchie is now head of the computing techniques research department at AT&T Bell Laboratories. Plan 9 is the current research project much like Unix was 25 years ago.

The first half of Ritchie's talk covered the history of Unix development. When talking about the AT&T/Sun alliance he pointed out that the creation of OSF, the consortium formed by other vendors to defend against AT&T and Sun setting the direction of Unix without other vendor input, was a direct result of their work. At the time of the creation of OSF, Ken Thompson was on sabbatical in Australia. Dennis sent e-mail to Ken about it and Ken's response was, “Geez, Dennis, DEC and IBM in the same bed and WE did it.”

I talked to Ritchie at the press conference after his talk. I asked him if he saw parallels between development work on Unix 20 years ago and Linux today. He admitted that he wasn't familiar enough with current Linux development to be able to make the comparison but he did expect that it was the case.

I then offered him a copy of *Linux Journal* and received a surprising answer. He said, “Oh, I've seen copies [of *Linux Journal*] around the terminal room at The Labs.” Made me feel like Linux had made it into the big leagues.

Block of Booths?

UniForum will happen again next year. And while we wait for UniForum there are other Unix shows such as Unix Expo. We had a booth at UniForum, are being represented by Telemax at the Dayton Hamvention and are planning for a booth at Unix Expo.

In order to make Unix users aware of the Linux movement, the Linux community needs to make a show of force at these events. A block of booths would certainly help. We may be able to have a shared booth for smaller Linux vendors as part of this block. If you are interested in this idea, contact Joanne Wagner, our advertising manager. If we get enough interest she will coordinate this.

LJ Booth

Dennis Ritchie

ICMake Part 3

Frank Brokken

K. Kubat

Issue #3, June-July 1994

Part 2 covered some of the grammar of icmake source files. This part completes the task. The final part of this article will appear next month and will show examples of the use of icmake.

4.4.4.2. Special operators

In addition to the operators of the C programming language, icmake recognizes some `special' operators. These are:

1. The `younger' operator is used to compare two strings which represent filenames. An expression using younger is evaluated to non-zero or zero and may be used in a condition. The operator `newer' is an alias for younger.

   The expression using the younger operator yields non-zero if a file with the name represented by the left operand is more recent than the file represented by the right operand.

   E.g., the following code prints a message if file main.c is more recent than main:

   ```
   if ("main.c" newer "main") printf ("main.c is more recent than
   main\n");
   ```
2. The `older' operand compares two files and yields non-zero if the file represented by the left operand is older than the file represented by the right operand. When the date of a file is compared using older or younger and when no file with such a name is present, then the age of the file is assumed to be infinite. A consequence of this implementation is that as in the following code example, a message is displayed if “try” does not exist:

   ```
   if ("try.c" younger "try") printf ("try.c should be
   compiled!!\n");
   ```

4.4.4.3. Type casts

Though icmake does not allow the use of operators on different types, it is possible to convert one type into another. The conversion of a type into another type is referred to as a `type cast'.

Type casts are denoted by a type name in parentheses before the operand which should be converted, e.g., (int)x converts the operand x to integer representation. Type casts are not allowed on all types. For example, a list variable cannot be converted to int.

The following type casts are permitted:

1. An integer may be cast to string. For example:

   ```
         string
                      stringvar;
              stringvar = (string) 14; // now, stringvar is "14"
   ```

2. A string may be cast to int. This is the reverse action of the type cast shown in the listing above.
3. A string may be cast to a list. This may be particularly useful when filenames should be added to or removed from a list, e.g., in the listing below, the filename “main.c” (a string) is removed from the list cfiles:

   ```
      list
              cfiles;
                              // cfiles is set to hold a list of cfiles =
                              makelist ("*.c");   // all filenames with
                              extension .c
      cfiles -= (list) "main.c";      // filename main.c is removed from
                              // the list
   ```

Note that the string “main.c” must be converted to a list type to allow the subtraction from the list.

Other typecasts, specifically from a string to an ascii-representation, can be realized through specialized functions (see, e.g., the function ascii() in the next section).

Built-in functions

Built into icmake is a number of functions which may be used to perform special operations, such as scanning a directory for files, displaying information, etc.. Here, all built-in functions are described.

1. **arghead**: int arghead(string): This function sets the argument head to string. The `argument head' is used with the functions exec() and execute(), described below.

2. **argtail**: int argtail(string): This function sets the argument tail to string. The `argument tail' is used with the functions exec() and execute(), described below.
3. **ascii**: int ascii(string): This function returns the ascii-number of the first character in the string, supplied as argument, e.g., ascii("A") returns 65.
4. **ascii**: string ascii(int): The overloaded function, which expects an int argument, returns a string representation of a numeric ascii number. e.g., ascii(65) returns "A".
5. **change_base**: string change_base (string, string): This function changes the basename in the string which is supplied as its first argument to the basename which is supplied as second argument. The string with the changed basename is returned.

   Example:

   ```
   string
           name;
   name = change_base ("main.c", "test");  // name now is "test.c"
   ```
6. **change_ext**: string change_ext(string, string): This function changes the extension in the string which is supplied as its first argument to the extension which is supplied as second argument. The modified string is returned.

   The extension (the second argument) may be specified as an empty string (""); in this case change_ext() removes the extension. Also, the extension may be specified as one dot ("."); in this case change_ext() removes the extension but leaves the dot.

   Example:

   ```
   char
           name;
   name = change_ext ("main.c", "o");      // name now is "main.o" name =
   change_ext (name, "");       // name now is "main" name = change_ext
   (name, ".");      // name now is "main."
   ```
7. **change_path**: string change_path(string, string): This function changes the path in the string which is supplied as its first argument to the path which is supplied as second argument.

   Example:

   ```
   string
           name;
                                   // name is now "/bin/prog"
   name = change_path ("c:/usr/local/bin/prog", "/bin"); name = change_path
   (name, "");  // name is now "prog"
   ```
8. **chdir**: string chdir(int, string): This function changes the current working directory to the supplied name. The first int argument may be **P_CHECK** or **P_NOCHECK**. This argument is optional. When absent, **P_CHECK** is assumed. Failure to change the working directory with the presence of **P_CHECK** leads to the termination of icmake process.

   However, on completion of the icmake process the original directory is always restored.

A string containing the new working directory, always ending in a directory separator, is returned. The string argument may terminate in a final slash. The returned string can be used to inspect whether the requested directory is reached, given that the modifier **P_NOCHECK** is supplied as first argument.

Two special string arguments are recognized by **chdir()**:

a. A directory argument which consists of one dot (i.e., the string ".") realizes a `change' to the current directory. The return value is then a string holding the current working directory.

b. A directory argument which is an empty string (i.e., the string "") will not produce a directory change. Instead, the directory from which icmake was started originally is returned.

Example:

```
                      // print the current working directory
                           printf("Current dir: ", chdir ("."), "\n");
   chdir ("/usr/bin");      // change to directory /usr/bin
                           // print startup directory
   printf ("Startup dir: ", chdir (""), "\n");
```

9. **cmdhead**: int cmdhead(string): This function sets the command head to string. The `command head' is used with the functions exec() and execute(), described below.
10. **cmdtail**: int cmdtail(string): This function sets the command tail to string. The `command tail' is used with the functions exec() and execute(), described below.
11. **echo**: int echo(int): This function determines whether, before the execution of a command, the command will be displayed. The argument of the function determines the displaying mode: when zero, displaying is suppressed; else, commands are displayed before execution. Two predefined constants are available for use as an argument to echo(): the constants ON and OFF. The values of these constants are, respectively, 1 and 0. Initially, echoing is on.

    Example:

    ```
       echo (ON);      // commands will be displayed echo (OFF);     //
            commands will not be displayed
    ```

12. **element**: string element(int, list): This function retrieves a string from a list. The order number of the name in the list is given by the first argument. Note that this index is zero-based, i.e., the first element in the list has index 0. The last element in the list has index **sizeoflist(list) - 1**.

    Example:

    ```
                  list
                              l;
                       string
                              n;
                       int
                              i;
                       l = makelist ("*.c"); for (i = 0; i < sizeoflist (l);
    ```

```
                  i++) if (element (i, l) newer "main") printf ("Source file
                  ", element (i, l), " is more recent than main\n");
```

13. **element**: string element(int, string): This function retrieves a substring of one character from the string given as its second argument.

    The character which is returned is found in the second (string) argument at the offset position specified in the first (int) argument. This index is zero-based: the first character of the string has index 0.

    Example:

    ```
                 string
                           s;
                      int
                           count; i;
                      count = 0; s = "Hello world"; for (i = 0; element(i,
                      s); i++)
                           count++;
                      printf("String '", s, "' contains ", count, "
                      characters.\n");
    ```

14. **exec**: int exec(int, string, ...): This function executes a command by spawning a child process. The arguments are:

    a. The first argument is an optional mode (an int). It may be P_CHECK (0) or P_NOCHECK (2). These predefined constants determine whether the exit status of the command should be checked or not. If the exit status should be checked, and a non-zero value is returned by the called program, the processing of the icmake file is aborted. If the first argument is omitted (i.e., if the first argument is not an int), P_CHECK is assumed.

    b. The second argument is the command to run (a string). This is the name of the program to be activated.

    c. The following arguments are the arguments which should be passed to the called program. These arguments may be ints, lists or strings.

    Each command is composed of the program name (the second argument), followed by the current setting of the command head (see cmdhead(), followed by all arguments and terminated with the command tail (see cmdtail()). Each argument to the command is prefixed with the argument head (see arghead()) and postfixed by the argument tail (see argtail()).

15. **execute**: int execute(int, string, string, string, ..., string, string): This function executes a command by spawning a child process. The arguments are:

    a. The first argument is an optional mode (an int). It may be P_CHECK (0) or P_NOCHECK (2). These predefined constants determine whether the exit status of the command should be checked or not. If the exit status should be checked, and a non-zero value is returned by the called program, the processing of the icmake file is aborted. If the first argument is omitted (i.e., if the first argument is not an int), P_CHECK is assumed.

    b. The second argument is the command to run (a string). This is the name of the program to be activated.

c. The third argument is the command head (a string). This string is used as first argument to the program name. The string may be empty (i.e., ""), in which case no command head is used.

d. The fourth argument is the argument head (a string). This string is prefixed to all following arguments. The string may be empty, in which case no argument head is used.

e. The following arguments are the arguments which should be passed to the called program. These arguments may be ints, lists or strings.

f. The next to the last argument is the argument tail (a string). This string is postfixed to each argument passed to the called program. The string may be empty, in which case no argument tail is used.

g. The last argument is the command tail (a string). The command run by the execute() function is postfixed with this string. The string may be empty, in which case no command tail is used.

After execution, execute() resets the command head, command tail, argument head and argument tail to empty strings. Both the exec() and execute() functions terminate the making process if error checking is turned on (mode flag P_CHECK) and if the run command exits with a non-zero exit value. If error checking is off, the exit status of the child process is returned.

16. **exists**: int exists(string): This function tests if a file exists. The file name is supplied as argument. A non-zero value is returned when the file exists; else, zero is returned.

```
      if (exists ("main.c"))
                        printf ("file main.c found\n");
                   else
                        printf ("file main.c not found\n");
```

17. **fgets**: list fgets(string, int): This function reads a line of text from the file whose name is given as its first (string) argument. Reading starts at the offsetposition specified in the second (int) argument. A list is returned, containing as its first element the string which was read, including the final newline character (as it is returned by the C++ function fgets()). The second element of the returned list contains the string representation of the offset of the file after the line was read. This string can be cast to an int.

    Example:

```
      // showing the file info.doc on the screen: int
                           offset;
                   list
                           l;
                   for (
                           offset = 0;
                                    l = fgets("info.doc", offset);
                                   offset = (int)element(1, l)
                   )
                           printf(element(0, l));
```

18. **fprintf**: int fprintf(string, ...): This function appends information to the file whose name is given as its first string argument. The remaining arguments define the information which is written to the file. The information is always appended to an existing file, which is opened in textmode.

    fprintf() acts analogously to printf() (see below), but the information is written to file rather than to screen.

    The arguments beyond the first argument of fprintf() define the information to print and may be ints, lists or strings. Example: **fprintf ("file.txt", 1, " line written to file.txt\n");**

19. **get_base**: string get_base(string): This function returns the basename of the filename stored in the string argument. The empty string is returned if the argument contains no basename. This happens when a disk or a root directory is specified in string. It may also happen if the syntax rules for a filename specification are violated. Example:

    ```
                                // prints 'main'
            printf(get_base ("/path/main.c"));
                                // prints 'No basename: '
            printf("No basename: ", get_base ("/"));
    ```

20. **getch**: string getch(): This function returns one character as a ministring. The character is read from the standard input stream (usually the keyboard).

    Under Unix, this function waits until a key and the enter key are pressed. Example:

    ```
            printf(getch());        // prints a character
                                    // (or an empty string)
    ```

21. **get_ext**: string get_ext(string): This function returns the extension in the string argument of the function. The empty string is returned if the argument does not contain an extension. Example:

    ```
            printf(get_ext ("/path/main.c"));   // prints 'c' printf(get_ext
            ("main"));             // returns empty string
    ```

22. **get_path**: string get_path(string): This function returns the path stored in the string argument of the function. An empty string is returned if string does not contain a disk specifier. The function returns the longest possible pathname which can be derived from the string argument. Example:

    ```
      printf(get_path ("/path/main.c"));      // prints '/path/' get_path
      ("main.c");            // returns an empty string
    ```

23. getpid: int getpid(): This function returns the process number of the currently executing icmake program. Example:

    ```
            // this function kills the current process..
            // analogous to exit()
            void harakiri()
            {
            exec ("kill", "-9", getpid ());
            }
            // this function returns a name for a temporary file
            // based on the process ID number file names are,
            // e.g., "/tmp/_TMPFILE.1256"
            string tempfilename ()
    ```

```
          {
               return ("/tmp/" + "_TMPFILE." + (string)getpid());
          }
```

24. **gets**: string gets(): This function returns the return value of the C function gets() as a string. The function accepts character, including backspaces allowing corrections, until the enter-key is pressed. The entered characters are returned in a string. Example:

```
          printf(gets()); // prints a string
                          // (or an empty string)
```

25. **makelist**: list makelist(int, string): This function makes a list of strings, representing filenames which match the expanded form of its argument. The arguments are an optional int, specifying the type of directory entries to search for, and a string specifying the file mask. The returned list may hold zero or more names.

    The first int argument specifies the type of entries to search for. It may be O_FILE (when searching for files), O_DIR (when searching for directories) or O_SUBDIR (when searching for subdirectories). The difference between the searching for directories and the searching for subdirectories lies in the fact that the current directory, denoted by ".“, and the parent directory, denoted by "..“, are not considered subdirectories but are considered directories. This argument may be absent, in which case O_FILE is assumed.

    A fourth type is O_ALL. When this type is given, makelist() searches for all directory entries irrespective of their type; e.g., under DOS, hidden and system files are matched as well as normal files or (sub)directories.

    The behavior of makelist() is dependent on the used platform, e.g., to search for all files or (sub)directories under DOS, the file mask "*.*“ must be given. The file mask "*“ will fail to find files or (sub)directories with an extension. Furthermore, makelist() behaves under DOS similar to the C run-time functions **_dos_findfirst()** and **_dos_findnext()**; e.g., **makelist(O_DIR, ".")** returns a list containing the name of the current directory.

    In a similar vein, the filemask "*“ will, under Unix, fail to find files or (sub)directories starting with a dot. Example:

```
          list
                  l;
          l = makelist ("*.c"); printf ("All found *.c files are: ",
          l, "\n"); l = makelist (O_SUBDIR, "*"); printf ("All found
          subdirectories are: ", l, "\n");
```

26. **makelist**: The function makelist(), furthermore, has overloaded versions. These, apart from a first optional int indicating the type of entries to search for, expect three arguments. The arguments of these versions are a (string) mask; a comparison operator which must be younger, newer or older; and a (string) referencefile. The function returns a list of files matching the mask which are older or newer than the referencefile.

An optional first int argument, which specifies the type of directory entry (O_FILE, O_DIR or O_SUBDIR) may be present. Example:

```
list
                l;
        l = makelist ("*.c", newer, "libprog.a"); printf ("All .c files
        newer than libprog.a are: ", l, "\n");
```

27. **printf**: int printf(int, ...): This function displays information. The arguments define the information to print and may be ints, lists or strings. A list is printed as a series of all its elements with spaces in between.
28. **putenv**: void putenv(string): This function may be used to set environment variables during the execution of icmake programs. The environment variables remain active during the complete icmake run. Example:

```
        main() {
                putenv("term=vt320");      // set variable system("set");
                // show settings
        }
```

29. **sizeof**: int sizeof(list): This function performs the same action as sizeoflist().
30. **sizeoflist**: int sizeoflist(list): This function determines the number of names held in a list. Example:

```
        list
                l;
        int
                i;
        list = makelist ("*.c"); i = sizeoflist (l); printf ("There are
        ", i, " names in the list.\n");
```

31. **stat**: list stat(int, string): This function attempts to retrieve file attributes of the file specified by the second string argument. The first int argument may be P_CHECK or P_NOCHECK. When absent, P_CHECK is assumed. The making process is aborted when stat() fails to retrieve file attributes and when P_CHECK is given for the first argument. The returned list holds the following information:

    a. The first element is a string representation of the mode of the file or directory. This string can be converted to an int where the following bits represent the modes:

    - The bit S_IFDIR is set when the entry is a directory.
    - The bit S_IFCHR is set when the entry is a character-special.
    - The bit S_IFREG is set when the entry is a regular file.
    - The bit S_IREAD is set when the entry is readable.
    - The bit S_IWRITE is set when the entry is writeable.
    - The bit S_IEXEC is set when the entry is executable.

    The second element is the file size, also represented as a string.
32. **strlen**: int strlen(string): This function returns the number of characters in the string which is supplied as its argument. Its working is analogous to the function strlen() in C.

33. **strlwr**: string strlwr(string): This function returns a copy of the string which is supplied as argument in lower case.
34. **strupr**: string strupr(string): This function returns a copy of the string which is supplied as argument in upper case.
35. **strtok**: list strtok (string, string): This function parses the first string in substrings, separated from each other by the separators specified in the second string. Each substring is an element of the returned list. If no separators are specified (i.e., the second string is empty), the individual characters of the first string become the elements of the returned list. Example:

    ```
    list
            l;
    int
            i;
    l = strtok("Hello-world\n", "-");
    printf("Two elements: ", l, "\n");
    l = strtok("Hello-world\n", "");
    printf("A string of ",
    sizeof(l), " characters\n");
    ```

36. **substr**: int substr(string, string) This function searches for the string which is given as the second argument in the string given as first argument. The position where the second string occurs in the first string is returned. The value -1 is returned when the second string does not occur in the first string.
37. **system**: int system(int, string): This function activates the operating system's command interpreter to run the command defined by the argument. The string holds the command to execute and, if needed, its arguments.

    The first argument specifies whether a failure of the system() function should terminate the making process. The value of this int may be P_CHECK or P_NOCHECK. This argument may be absent, in which case P_CHECK is assumed. The return value of system() is zero when the command could successfully be executed. A return value which is not zero can be received by the makefile only when P_NOCHECK is given as first argument.

    system() succeeds if the command could be executed and if the return value of the command itself is zero.

The Open Development of Debian

Ian Murdock

Issue #3, June-July 1994

The Debian project has always been open to anyone wanting to participate in discussion of its development, but it was only recently that the actual task of package maintenance was allocated to members of the project. This was an extremely important step in the development of Debian, and in this month's column I will explain why it was done, why it could not have een done any sooner, and why this step will only make the Debian system better.

Since the early days of Debian, I have described it as the first distribution of Linux to be developed openly in the spirit of the operating system and the free software movement. My justification was the success of the kernel; after all, open and distributed development had worked so well for the kernel, why couldn't it work just as well for a distribution?

The justification made sense, of course, but the reality was not quite that simple. Before any open development could be done, a cohesive development group had to be organized, a solid foundation upon which to develop had to be made available, and most importantly, it had to be proven that open and distributed development could actually work for a distribution. At the beginning, there were many strong supporters of Debian, but there were just as many people who said that it simply could not be done.

For the first months of the project, although many volunteers helped prepare Debian for distribution, I was the one who actually assembled the version to be released, since the guidelines for creating all the pieces of the distribution changed from day to day as we improved the system. Also, the most recent versions of most of the guidelines were often only to be found in my head.

Because one person has limited time, I could only do this job by myself for so long, while the system was still small, even with volunteers helping get many of the pieces ready. As the guidelines began to stabilize, and as the system became large enough that I could not put the system together single-handedly (as I found out when I tried), it became clear that the time had come to take the next step.

With an organized system for creating the Debian distribution in hand, the volunteers who had already been helping to create Debian packages were able to step forward and take complete responsibility for them, without fear of anarchy. They can package their own software, knowing that it will fit smoothly into the rest of the Debian system.

It has been a rough ride, but the Debian project survives and is stronger now than it has ever been. We are organized, we have a solid foundation coming in the release of Debian 0.93 BETA, and, as proven by the support that Debian has received and the enthusiasm of the Debian project volunteers, we have proven that the idea of an openly-developed distribution can work. I remain the maintainer of the base system and several development packages, and I serve as the leader of the project as a whole. But I am now only a part of a larger movement, a movement that is gaining momentum. So, what does all of this mean to the users of Debian?

It means high quality.

We have many well-respected members of the Linux community directly involved in the development of Debian. We have experts working directly on the parts of Debian involving their area of expertise. We have package maintainers that closely follow new releases of the software that they are packaging, and in many cases, participate directly in the development of the software itself.

It means consistency.

We now have drafted guidelines which define the construction of Debian binary and source packages. With the help of these guidelines, package maintainers may put together packages that are consistent with those put together by other package maintainers. The Debian leadership checks to ensure that each package is assembled correctly and that the system as a whole is solid; the end result is a set of packages that, though developed and maintained by many different individuals, are as consistent and as professionally constructed as if they were developed by a single person or closely-knit group, but without the limitations imposed by centralized development.

It means modularity.

No longer does one person have to shoulder the burden of constructing every package. As mentioned earlier, I maintain the base system and many of the development packages; others maintain the networking packages, the XFree86 packages and so on. When a new component is available, the package maintainer may simply update that package and make the newest version available, without a complete update of Debian. The days of massive, comprehensive re-releases of Debian are over; each component of Debian will be fairly independent of all others, and it will be easy for the end-user to upgrade his system or any component of it.

And, most importantly, need I mention that you, too, can become involved!

Linux Programming Hints

Michael K. Johnson

Issue #3, June-July 1994

This article will explain how to program the VT interface to do things which can't easily be done with “escape sequences” on a Linux console, giving a reference for the ioctl()'s needed to do this. Much of this column is derived from a document written by Orest, as he is interested in disseminating this information further.

Around Linux versions .12 and .95 (Which were consecutive, for those of you who aren't up on some of the weirdness of Linux history ...), Orest Zborowski1 undertook the task of getting the X Windowing System to run on Linux. Instead of taking the short-sighted methods and spending his time porting X to Linux, Orest ported Linux to X. To do this, he wrote the orginal Unix-domain sockets for Linux and the VT interface, which is a subset of the VT interface under SVR4. Later, Andries Brouwer2, who did most of the work for loadable keymaps, added more keyboard handling functionality.

This article will explain how to program the VT interface to do things which can't easily be done with “escape sequences” on a Linux console, giving a reference for the ioctl()'s needed to do this. Much of this column is derived from a document written by Orest, as he is interested in disseminating this information further.

The VT interface

The VT interface is a set of ioctl()'s that can be performed on any console device. VT's are tightly bound to VC's, or virtual consoles. They are named differently because they are called VT's in SVR4, and also because in the source there is some differentiation between VT operations and VC operations. VT numbering is the same as VC numbering: 0 is a synonym for the “current” VT, and all real VTs start from 1. In all the ioctls below, it is legal to use VT 0 as the target of the ioctl—it simply affects the VT that's currently active.3

This is different from SVR4, where 0 is the first VT, and /dev/console is the current VT. This difference is due to the fact that the orignal VC's in Linux used VC 0 as /dev/console, whereas SVR4 makes /dev/console a seperate device. This does not cause problems in practice, fortunately.

The header files sys/vt.h and sys/kd.h are pretty much complete, according to SVR4 rules, but much of their content is unsupported by Linux. The header file

The linux/keyboard.h file maintains more information dealing with keyboard mapping, and contains the parts written by Brouwer.

VT Reference

ioctl(int ttyfd, KIOCSOUND, unsigned int count)

KIOCSOUND will turn on a sound using the relation

```
                    hz = 1193180
                  ---------------
                       count
```

If count **= 0**, then sound is turned off. The sound will continue until it is explicitly turned off.

```
    ioctl(int ttyfd, KDMKTONE, unsigned int count_ticks)
```

KDMKTONE will turn on a sound for a specific number of ticks. count_ticks is composed of two pieces: the upper 16 bits hold the number of ticks (hundredths of a second under Linux/86, at least; see the HZ define in linux/sched.h) that you want the sound to last, while the lower 16 bits hold the count, which is the same as the count argument to KIOCSOUND. The call returns immediately.

ioctl(int ttyfd, KDGKBTYPE, unsigned char *kb)

KDGKBTYPE returns the keyboard type in kb. This can be:

```
           KB_84           84 key keyboard
           KB_101          101 key keyboard
           KB_OTHER         other keyboard
```

Bugs:

Currently, **KB_101** is always returned.

```
   ioctl(int ttyfd, KDADDIO, int port)
```

KDADDIO will enable access to the specified port. The port must be in the range **0x3b4** to **0x3df** (which covers the common graphics ports). For access to ports outside this range, use the **ioperm(2)** system call.

```
  ioctl(int ttyfd, KDDISABIO, int port)
```

KDDISABIO will disable access to the specified ports. See **KDADDIO** for further details.

```
  ioctl(int ttyfd, KDSETMODE, int mode)
```

KDSETMODE changes the mode of the VT to be either text or graphics:

```
          KD_GRAPHICS                  graphics mode KD_TEXT
          text mode KD_TEXT0             same as KD_TEXT KD_TEXT1
          same as KD_TEXT
```

ttyfd must be the current console. If the mode specified is already in place, nothing is done. When going to text mode, the screen is unblanked and the blanking timer is enabled (as in normal operation). When going to graphics mode, the screen is blanked and will remain so until switched back into text mode.

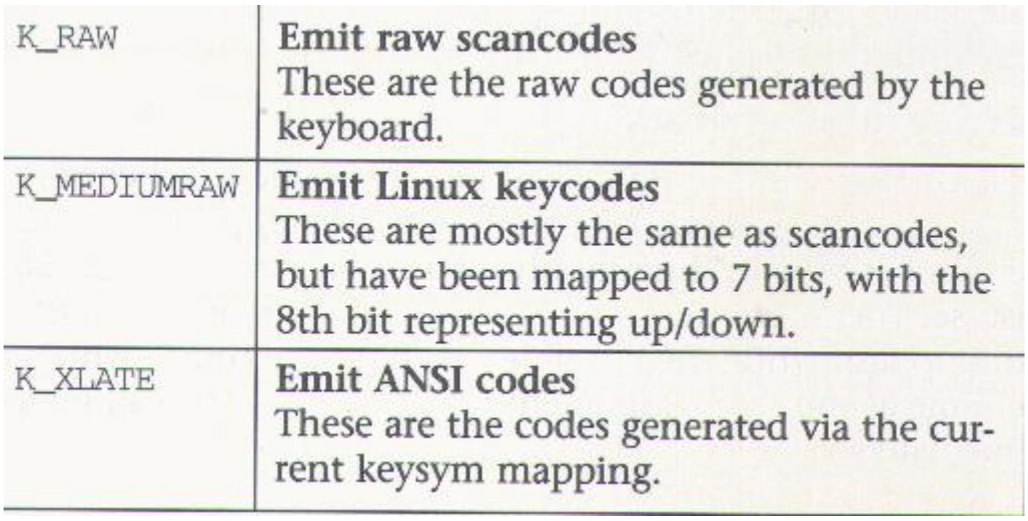
Bugs: No special provisions are made to save or restore the contents of the VT during this call. It is up to the application to save any necessary information for later restoration. This is because chipset-specific information is required to properly save or restore the contents of the VT.

```
  ioctl(int ttyfd, KDGETMODE, int mode)
```

KDGETMODE returns the current mode of the specified VT. See **KDSETMODE** for further details.

```
  ioctl(int ttyfd, KDSKBMODE, int kbmode)
```

KDSKBMODE sets the translation mode on the keyboard. The options are:



K_RAW	**Emit raw scancodes** These are the raw codes generated by the keyboard.
K_MEDIUMRAW	**Emit Linux keycodes** These are mostly the same as scancodes, but have been mapped to 7 bits, with the 8th bit representing up/down.
K_XLATE	**Emit ANSI codes** These are the codes generated via the current keysym mapping.

Switching from one to the other also flushes the input queue to avoid confusion. The kernel maintains correct state information for shift, lock, etc. keys regardless of the current mode.

```
ioctl(int ttyfd, KDGKBMODE, unsigned long *mode)
```

KDGKBMODE returns the keyboard mode associated with the particular tty.

```
ioctl(int ttyfd, KDGETLED, unsigned char *leds)
```

KDGETLED returns the state of the LED's as the flags:

```
        LED_SCR scroll lock is down
        LED_NUM num lock is down
        LED_CAP caps lock is down
```

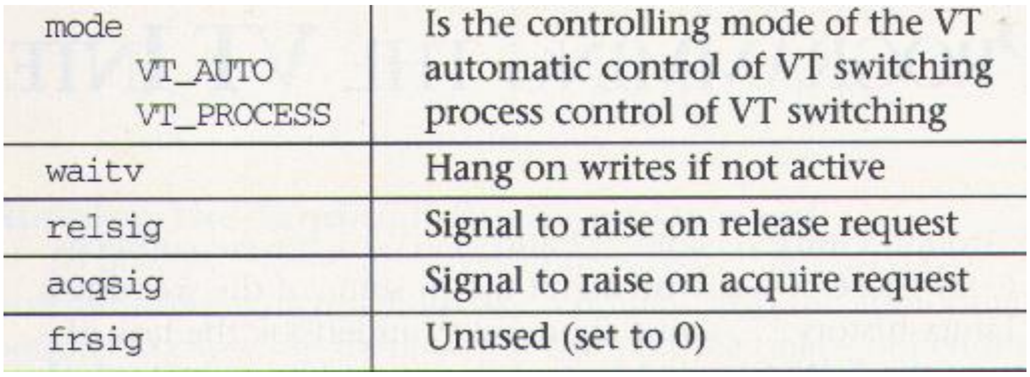
```
ioctl(int ttyfd, KDSETLED, unsigned char leds)
```

KDSETLED sets the LED's according to the flags passed in. The correct use is to use **KDGETLED**, then make changes to those flags, then use **KDSETLED** to change the flags.

```
ioctl(int ttyfd, VT_SETMODE, struct vt_mode *vtm)
```

VT_SETMODE sets the control mode of the VT according to the structure:

```
struct vt_mode {
        char mode;
        char waitv;
        short relsig;
        short acqsig;
        short frsig;
};
```



mode VT_AUTO VT_PROCESS	Is the controlling mode of the VT automatic control of VT switching process control of VT switching
waitv	Hang on writes if not active
relsig	Signal to raise on release request
acqsig	Signal to raise on acquire request
frsig	Unused (set to 0)

In **VT_AUTO** mode, the kernel takes care of VT switches, etc. This is the default mode. In VT_PROCESS mode, one process takes control over a VT. It is responsible for acknowledging switch requests and performing any duties required. For example, a graphics program may want to run in VT_PROCESS mode, so if the user wants to switch to another VT and back, the graphics mode is properly saved and restored.

A full description of the switching semantics is described in a section below.

Bugs: The waitv mode of writes is not supported.

```
ioctl(int ttyfd, VT_GETMODE, struct vt_mode *vtm)
```

VT_GETMODE returns the current control state of the VT. See VT_SETMODE, above, for further details.

```
ioctl(int ttyfd, VT_GETSTATE, struct vt_stat *vts)
```

VT_GETSTATE returns the state of all VT's in the kernel in the structure:

```
struct vt_stat {
        ushort v_active;
        ushort v_signal;
        ushort v_state;
};
```

```
v_active        the currently active VT
v_state         mask of all the opened VT's
```

v_active holds the number of the active VT (starting from 1), while **v_state** holds a mask where there is a 1 for each VT that has been opened by some process. Note that VT 0 is always opened in this scenario, since it refers to the current VT.

Bugs:

The **v_signal** member is unsupported.

```
ioctl(int ttyfd, VT_OPENQRY, long *num)
```

VT_OPENQRY returns the number of the first available VT, one that hasn't been opened by any process. If there is no free VT, -1 is returned in num.

```
ioctl(int ttyfd, VT_ACTIVATE, int num)
```

VT_ACTIVATE will cause a switch to VT number num, as if caused from the keyboard. In particular, if VT number num is in **VT_PROCESS** mode, then negotiation with the process in charge is begun. The call may return before the switch is complete. Use **VT_WAITACTIVE** to wait until the switch is completed.

```
ioctl(int ttyfd, VT_WAITACTIVE, int num)
```

VT_WAITACTIVE will wait until the specified VT has been activated (the switch to it has been completed).

Bugs:

This call does not actually do the switch, but it may need to do the switch as well, as SVR4 does, to be compatible with some applications.

```
ioctl(int ttyfd, VT_RELDISP, int val)
```

VT_RELDISP is used to signal the kernel about the switch in progress. If **ttyfd** is the current console, then it must be in **VT_PROCESS** mode.

If switching from one VT to another VT, the “from” VT is signalled about the switch-from request. The reply is through the VT_RELDISP ioctl with the following values:

```
0       switch is disallowed, and the kernel aborts the attempt
1       switch is allowed, and the kernel continues with the switch
2       switch has been completed
```

If switching to a VT from another VT, the kernel will signal about the switch-to request. The reply is through the **VT_RELDISP** ioctl with the following value:

```
VT_ACKACQ               switch-to is allowed
```

Bugs:

The switch-to response is a nonstandard behavior in SVR4. Currently, Linux doesn't require the switch-to **VT_RELDISP** ioctl, but if made, it must have the argument **VT_ACKACQ**.

```
ioctl(int fd, KDSKBMETA, int flags)
```

KDSKBMETA specifies if pressing the meta (alt) key generates an ESC (**\033**) prefix followed by the keysym, or the keysym marked with the high bit set.

```
K_METABIT       generate an ESC  prefix
K_ESCPREFIX     same as K_METABIT
0               generates a high-bit marked keysym
```

```
ioctl(int fd, KDGKBMETA, unsigned long *flags)
```

KDGKBMETA returns the state of the META prefix, as described in **KDSKBMETA** above.

```
ioctl(int fd, KDGKBENT, struct kbentry *kbe)
```

KDGKBENT returns the keysym mapping for a particular key and modifier.

```
struct kbentry {
        u_char kb_table;
        u_char kb_index;
        u_short kb_value;
        };
```

The user sets the **kb_table** to the modifier table requested, and **kb_index** to the keycode requested. **KDGKBENT** returns the keysym in **kb_value**. The modifier table is generated by the logical “or” of the following values:

```
        K_NORMTAB       normal table
        K_SHIFTTAB      shift
        K_ALTTAB        alt (meta)
        K_SRQTAB        right alt (altgr)
```

```
ioctl(int fd, KDSKBENT, struct kbentry *kbe)
```

KDSKBENT sets the keysym mapping for a particular keycode and modifier combination. See **KDGKBENT** above for more information.

```
ioctl(int fd, KDGKBSENT, struct kbsentry *kbs)
```

KDGKBSENT returns the string bound to a particular function key:

```
struct kbsentry {
        u_char kb_func;
        u_char kb_string[512];
        };
```

kb_func is the index of the function key (**0 - NR_FUNC**), and **KDGKBSENT** will return the currently mapped string in **kb_string**.

```
ioctl(int fd, KDSKBSENT, struct kbsentry *kbs)
```

KDSKBSENT sets the string mapped to a function key. When this function key is depressed, the string is emitted. See **KDGDBSENT** above for an explanation of **struct kbsentry**.

```
ioctl(int fd, KDGKBDIACR, struct kbdiacrs *kbds)
```

KDGKBDIACR returns the kernel diacritical mapping table:

```
    struct kbdiacr {
            u_char diacr,
            base, result;
            }; struct kbdiacrs {
            unsigned int kb_cnt;
            struct kbdiacr kbdiacr[256];
            };
```

See the keymap package for details.

```
    ioctl(int fd, KDSKBDIACR, struct kbdiacrs *kbds)
```

KDSKBDIACR sets the diacritical table. See **KDGKBDIACR** above for details. See the keymap package for details.

```
    ioctl(int fd, PIO_FONT, unsigned char font[8192])
```

PIO_FONT sets the console video font. The font is 8192 bytes long and specific to the particular mode one is using. See the keymap package for details.

```
    ioctl(int fd, GIO_FONT, unsigned char font[8192])
```

GIO_FONT gets the console video font. Returns 8192 bytes of font information. See the keymap package for details.

```
    ioctl(int fd, PIO_SCRNMAP, unsigned char trans[256])
```

PIO_SCRNMAP sets the user console translation table. This maps an 8 bit code to a video font code. The user table is selectable by sending ESC**(K** to the console. See the keymap package for details.

```
    ioctl(int fd, GIO_SCRNMAP, unsigned char trans[256])
```

GIO_SCRNMAP returns the console translation table. See the keymap package for details. VT switching When the user types <Alt>-<Fn>, where n is the number of a VT, the kernel will switch to that VT. The same sequence happens if some process performs an **ioctl(fd, VT_ACTIVATE, n);**

First, if the “switch-to” VT is in **VT_AUTO** mode, then the kernel will ignore the switch request if it's also in **KD_GRAPHICS** mode, else it will continue the switch.

If the “switch-to” VT is in **VT_PROCESS** mode, the relsig signal is sent to the “switch-from” process so it can release the VT. If the process accepts the signal, the kernel will await the **VT_RELDISP ioctl** from it. If the process has died, the VT is forcib-ly reset to **KD_TEXT** and VT_AUTO mode. This can cause great confusion and unhappiness, but the kernel can't do anything better.

The “switch-from” process will need to perform any cleanup, and issue the **VT_RELDISP ioctl**, telling the kernel that it is OK to continue the switch. It is also possible for it to deny the switch, in which case the kernel discontinues the switch.

If the “switch-from” process has consented to the switch, the kernel will change to the new VT, changing the keyboard mode and LED's as well. Then, if the new VT is under **VT_PROCESS** control, the “switch-to” process is sent the acqsig signal. If this process is missing, the new VT is reset to **KD_TEXT** and **VT_AUTO** mode. In this fashion, there is a certain amount of auto-resetting going on during normal use. Of course, if a process makes graphics changes in **KD_GRAPHICS** mode, these will not be undone by the kernel.

At this point the switch is complete. The “switch-to” process may call **VT_RELDISP VT_ACKACQ**, but this is not required by the kernel. If there were any processes waiting for this new VT to become active, they are woken at this point.

Examples

For most people, the X source code is far too large to easily download, and far too large to easily study. There are, however, other examples available. **svgalib** provides an easy-to-use interface to these functions, as well as providing a consistent interface to VGA and some SVGA video boards. It also serves as example code for those who want to roll their own code, as it includes examples of using **mmap()** and **ioperm()** to directly access video memory, once it has used the **ioctl()**'s described above so that it is allowed to. The following code fragments descibe one way to access ports and memory, without using svgalib. **PAGE_SIZE** is defined in **<linux/page.h>**, and **GRAPH_SIZE** and **GRAPH_BASE** may differ from video card to video card. This code is based on code in vgalib version 1.2.

```
FILE *mem_fd;
char *graph_get, *graph_mem;
if (ioperm(port, 1, 1)) {
   fprintf(stderr, "Can't access port %x\n", port); exit(1);
} if ((mem_fd = open("/dev/mem", O_RDWR)) < 0) {
   fprintf(stderr, "Can't open /dev/mem\n"); exit(1);
} if ((graph_get = malloc(GRAPH_SIZE + (PAGE_SIZE-1))) === NULL) {
fprintf(stderr, "Insufficient memory\n");
   exit(1);
}
graph_mem = graph_get; if ((unsigned long)graph_mem % PAGE_SIZE)
   graph_mem += PAGE_SIZE - ((unsigned long)graph_mem % PAGE_SIZE);
graph_mem = (unsigned char *)
                mmap((caddr_t)graph_mem,
                        GRAPH_SIZE, PROT_READ|PROT_WRITE,
                        MAP_SHARED|MAP_FIXED, mem_fd, GRAPH_BASE);
if ((long)graph_mem < 0) {
   fprintf(stderr, "mmap error\n"); exit(1);
}
```

At this point, writing to **graph_mem** is actually writing to screen memory. **iopl()** and **ioperm()** can also be used to get permission to write to ports, as can the

KDADDIO ioctl(), described above. The Linux Documentation Project man pages include man pages on **iopl()** and **ioperm()**, so I will not document them here, as those man pages should have come with your Linux distribution. They are accessible on sunsite.unc.edu as /pub/Linux/docs/LDP/man-pages/* if you do not have them.

The DOS emulator also uses some of these calls to provide the interface that DOS is used to having to real DOS programs, and to allow DOS sessions to use the video bios provided with the video card.

The definitive example code for most of the **KD*i ioctl()**'s is the keymap package, distributed with the Linux kernel.

For next month, I plan to explain how these calls can be used to write a screen-lock package for Linux, as my space and time are up for this month.

Archive Index Issue Table of Contents

Advanced search

What's GNU: Bash - The GNU Shell

Chet Ramey

Issue #3, June-July 1994

While originally written by Brian Fox of the Free Software Foundation, bash is now maintained by Chet Ramey. In this article, Chet explains the nhistory of shells and then goes on to explore features specific to bash.

Bash is the shell, or command language interpreter, that will appear in the GNU operating system. The name is an acronym for the “Bourne-Again SHell”, a pun on Steve Bourne, the author of the direct ancestor of the current Unix shell /bin/sh, which appeared in the Seventh Edition Bell Labs Research version of Unix.

Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and the C shell (csh), described later in this article. It is ultimately intended to be a faithful implementation of the IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2). It offers functional improvements over sh for both interactive and programming use.

While the GNU operating system will most likely include a version of the Berkeley shell csh, Bash will be the default shell. Like other GNU software, Bash is quite portable. It currently runs on nearly every version of Unix and a few other operating systems-an independently-supported port exists for OS/2, and there are rumors of ports to DOS and Windows NT. Ports to Unix- like systems such as QNX and Minix are part of the distribution.

What's POSIX, anyway?

POSIX is a name originally coined by Richard Stallman for a family of open system standards based on Unix. There are a number of aspects of Unix under consideration for standardization, from the basic system services at the system call and C library level to applications and tools to system administration and management. Each area of standardization is assigned to a working group in the 1003 series.

The POSIX Shell and Tools standard has been developed by IEEE Working Group 1003.2 (POSIX.2). It concentrates on the command interpreter interface and utility programs commonly executed from the command line or by other programs. An initial version of the standard has been approved and published by the IEEE, and work is currently underway to update it. There are four primary areas of work in the 1003.2 standard:

- Aspects of the shell's syntax and command language. A number of special builtins such as cd and exec are being specified as part of the shell, since their functionality usually cannot be implemented by a separate executable;
- A set of utilities to be called by shell scripts and applications. Examples are programs like sed, tr and awk. Utilities commonly implemented as shell builtins are described in this section, such as test and kill. An expansion of this section's scope, termed the User Portability Extension, or UPE, has standardized interactive programs such as vi and mailx;
- A group of functional interfaces to services provided by the shell, such as the traditional system() C library function. There are functions to perform shell word expansions, perform filename expansion (globbing), obtain values of POSIX.2 system configuration variables, retrieve values of environment variables (getenv()), and other services;
- A suite of “development” utilities such as c89 (the POSIX.2 version of cc), and yacc.

Bash is concerned with the aspects of the shell's behavior defined by POSIX.2. The shell command language has of course been standardized, including the basic flow control and program execution constructs, I/O redirection and pipelining, argument handling, variable expansion, and quoting. The special builtins, which must be implemented as part of the shell to provide the desired functionality, are specified as being part of the shell; examples of these are eval and export. Other utilities appear in the sections of POSIX.2 not devoted to the shell which are commonly (and in some cases must be) implemented as builtin commands, such as read and test. POSIX.2 also specifies aspects of the shell's interactive behavior as part of the UPE, including job control and command line editing. Interestingly enough, only vi-style line editing commands have been standardized; emacs editing commands were left out due to objections.

While POSIX.2 includes much of what the shell has traditionally provided, some important things have been omitted as being “beyond its scope”. There is, for instance, no mention of a difference between a login shell and any other interactive shell (since POSIX.2 does not specify a login program). No fixed startup files are defined, either-the standard does not mention .profile.

Basic Bash features

Since the Bourne shell provides Bash with most of its philosophical underpinnings, Bash inherits most of its features and functionality from sh. Bash implements all of the traditional sh flow control constructs (for, if, while, etc.). All of the Bourne shell builtins, including those not specified in the POSIX.2 standard, appear in Bash. Shell functions, introduced in the SVR2 version of the Bourne shell, are similar to shell scripts, but are defined using a special syntax and are executed in the same process as the calling shell. Bash has shell functions which behave in a fashion upward-compatible with sh functions. There are certain shell variables that Bash interprets in the same way as sh, such as PS1, IFS and PATH. Bash implements essentially the same grammar, parameter and variable expansion semantics, redirection, and quoting as the Bourne shell. Where differences appear between the POSIX.2 standard and traditional sh behavior, Bash follows POSIX.

The Korn Shell (ksh) is a descendent of the Bourne shell written at AT&T Bell Laboratories by David Korn. It provides a number of useful features that POSIX and Bash have adopted. Many of the interactive facilities in POSIX.2 have their roots in the ksh. For example, the POSIX and ksh job control facilities are nearly identical. Bash includes features from the Korn Shell for both interactive use and shell programming.

For programming, Bash provides variables such as RANDOM and REPLY, the typeset builtin, the ability to remove substrings from variables based on patterns, and shell arithmetic.

- RANDOM expands to a random number each time it is referenced. Assigning a value to RANDOM seeds the random number generator.
- REPLY is the default variable used by the read builtin when no variable names are supplied as arguments.
- The typeset builtin is used to define variables and give them attributes such as readonly.

Bash arithmetic allows the evaluation of an expression and the substitution of the result. Shell variables may be used as operands, and the result of an expression may be assigned to a variable. Nearly all of the operators from the C language are available, with the same precedence rules:

```
    $ echo $((3 + 5 * 32)) 163
```

For interactive use, Bash implements ksh-style aliases and builtins such as fc (discussed below) and jobs. Bash aliases allow a string to be substituted for a command name. They can be used to create a mnemonic for a Unix command name (e.g., **alias del=rm**), to expand a single word to a complex command (e.g.,

alias news='xterm -g 80x45 -title trn -e trn -e -S1 -N &), or to ensure that a command is invoked with a basic set of options (e.g., **alias ls="/bin/ls -F"**).

The C shell (csh) was originally written by Bill Joy while at the University of California at Berkeley. It is widely used and quite popular for its interactive facilities. Bash includes a csh-compatible history expansion mechanism ("! history"), brace expansion, access to a stack of directories via the pushd, popd and dirs builtins, and tilde expansion, to generate users' home directories. Tilde expansion has also been adopted by both the Korn Shell and POSIX.2.

There were certain areas in which POSIX.2 felt standardization was necessary, but no existing implementation provided the proper behavior. The working group invented and standardized functionality in these areas, which Bash implements. The command builtin was invented so that shell functions could be written to replace builtins; it makes the capabilities of the builtin available to the function. The reserved word "!" was added to negate the return value of a command or pipeline; it was nearly impossible to express "if not x" cleanly using the sh language.

There exist multiple incompatible implementations of the test builtin, which tests files for type and other attributes and performs arithmetic and string comparisons. POSIX considered none of these correct, so the standard behavior was specified in terms of the number of arguments to the command. POSIX.2 dictates exactly what will happen when four or fewer arguments are given to test, and leaves

the behavior undefined when more arguments are supplied. Bash uses the POSIX.2 algorithm, which was conceived by David Korn.

Features not in the Bourne Shell

There are a number of minor differences between Bash and the version of sh present on most other versions of Unix. The majority of these are due to the POSIX standard, but some are the result of Bash adopting features from other shells. For instance, Bash includes the new "!" reserved word, the command builtin, the ability of the read builtin to correctly return a line ending with a backslash, symbolic arguments to the umask builtin, variable substring removal, a way to get the length of a variable, and the new algorithm for the test builtin from the POSIX.2 standard, none of which appear in sh.

Bash also implements the "$(...)" command substitution syntax, which replaces the sh `...` construct. The "$(...)" construct expands to the output of the command contained within the parentheses, with trailing newlines removed. The sh syntax is accepted for backwards compatibility, but the "$(...)" form is preferred because its quoting rules are much simpler and it is easier to nest.

The Bourne shell does not have such features as brace expansion, the ability to have a variable and a function with the same name, local variables in shell functions, the ability to enable and disable individual builtins or write a function to replace a builtin, or a means to export a shell function to a child process.

Bash has closed a long-standing shell security hole by not using the $IFS variable to split each word read by the shell, but splitting only the results of expansion (ksh and the 4.4 BSD sh have fixed this as well). Useful behavior such as a means to abort execution of a script read with the "." command or automatically exporting variables in the shell's environment to children is also not present in the Bourne shell. Bash provides a much more powerful environment for both interactive use and programming.

Bash-specific Features

This section details a few of the features which make Bash unique. Most of them provide improved interactive use, but a few programming improvements are present as well. Full descriptions of these features can be found in the Bash documentation.

Startup Files

Bash executes startup files differently than other shells. The Bash behavior is a compromise between the csh principle of startup files with fixed names executed for each shell and the sh "minimalist" behavior. An interactive instance of Bash started as a login shell reads and executes ~/.bash_profile (the file .bash_profile in the user's home directory), if it exists. An interactive non-login shell reads and executes ~/.bashrc. A non-interactive shell (one begun to execute a shell script, for example) reads no fixed startup file, but uses the value of the variable $ENV, if set, as the name of a startup file. The ksh practice of reading $ENV for every shell, with the accompanying difficulty of defining the proper variables and functions for interactive and non-interactive shells or having the file read only for interactive shells, was considered too complex. Ease of use won out here.

New Builtin Commands

There are a few builtins which are new or have been extended in Bash.

- The enable builtin allows builtin commands to be turned on and off arbitrarily. To use the version of echo found in a user's search path rather than the Bash builtin, "enable -n echo" suffices.
- The help builtin provides quick synopses of the shell facilities without requiring access to a manual page.

- Builtin is similar to command in that it bypasses shell functions and directly executes builtin commands. Access to a csh-style stack of directories is provided via the pushd, popd and dirs builtins.
- Pushd and popd insert and remove directories from the stack, respectively, and dirs lists the stack contents.
- On systems that allow fine-grained control of resources, the ulimit builtin can be used to tune these settings. Ulimit allows a user to control, among other things, whether core dumps are to be generated, how much memory the shell or a child process is allowed to allocate, and how large a file created by a child process can grow.
- The suspend command will stop the shell process when job control is active; most other shells do not allow themselves to be stopped like that.
- Type, the Bash answer to which and whence, shows what will happen when a word is typed as a command:

```
$ type export export is a shell builtin $ type -t export builtin $
type bash bash is /bin/bash $ type cd cd is a function cd () {
    builtin cd ${1+"$@"} && xtitle $HOST: $PWD
}
```

Various modes tell what a command word is (reserved word, alias, function, builtin, or file) or which version of a command will be executed based on a user's search path. Some of this functionality has been adopted by POSIX.2 and folded into the command utility.

Editing and Completion

One area in which Bash shines is command line editing. Bash uses the readline library to read and edit lines when interactive. Readline is a powerful and flexible input facility that a user can configure to their own tastes. It allows lines to be edited using either emacs or vi commands, where those commands are appropriate. The full capability of emacs is not present-there is no way to execute a named command with M-x, for instance-but the existing commands are more than adequate. The vi mode is compliant with the command line editing standardized by POSIX.2.

Readline is fully customizable. In addition to the basic commands and key bindings, the library allows users to define additional key bindings using a startup file. The inputrc file, which defaults to the file ~/.inputrc , is read each time readline initializes, permitting users to maintain a consistent interface across a set of programs. Readline includes an extensible interface, so each program using the library can add its own bindable commands and program-specific key bindings. Bash uses this facility to add bindings that perform history expansion or shell word expansions on the current input line.

Readline interprets a number of variables which further tune its behavior. Variables exist to control whether or not eight-bit characters are directly read as input or converted to meta- prefixed key sequences (a meta-prefixed key sequence consists of the character with the eighth bit zeroed, preceded by the meta-prefix character, usually escape, which selects an alternate keymap), to decide whether to output characters with the eighth bit set directly or as a meta-prefixed key sequence, whether or not to wrap to a new screen line when a line being edited is longer than the screen width, the keymap to which subsequent key bindings should apply, or even what happens when readline wants to ring the terminal's bell. All of these variables can be set in the inputrc file.

The startup file understands a set of C preprocessor-like conditional constructs which allow variables or key bindings to be assigned based on the application using readline, the terminal currently being used, or the editing mode. Users can add program-specific bindings to make their lives easier: I have bindings that let me edit the value of $PATH and double-quote the current or previous word:

```
# Macros that are convenient for shell interaction $if Bash # edit the
path "\C-xp": "PATH=${PATH}\\C-e\C-a\f\C-f" # prepare to type a quoted
word-insert open and close double quotes # and move to just after the
open quote "\C-x\"": "\"\"\C-b" # Quote the current or previous word
"\C-xq": "\b\"\f\"" $endif
```

There is a readline command to re-read the file, so users can edit the file, change some bindings, and begin to use them almost immediately.

Bash implements the bind builtin for more dyamic control of readline than the startup file permits. Bind is used in several ways. In list mode, it can display the current key bindings, list all the readline editing directives available for binding, list which keys invoke a given directive, or output the current set of key bindings in a format that can be incorporated directly into an inputrc file. In batch mode, it reads a series of key bindings directly from a file and passes them to readline. In its most common usage, bind takes a single string and passes it directly to readline, which interprets the line as if it had just been read from the inputrc file. Both key bindings and variable assignments can appear in the string given to bind.

The readline library also provides an interface for word completion. When the completion character (usually TAB) is typed, readline looks at the word currently being entered and computes the set of filenames of which the current word is a valid prefix. If there is only one possible completion, the rest of the characters are inserted directly, otherwise the common prefix of the set of filenames is added to the current word. A second TAB character entered immediately after a non-unique completion causes readline to list the possible completions; there is an option to have the list displayed immediately. Readline provides hooks so that applications can provide specific types of completion before the default filename completion is attempted. This is quite flexible, though it is not completely user-programmable. Bash, for example, can complete filenames, command names (including aliases, builtins, shell reserved words, shell functions, and executables found in the file system), shell variables, usernames, and hostnames. It uses a set of heuristics that, while not perfect, is generally quite good at determining what type of completion to attempt.

This article will be continued next month.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux: Virtual Dramamine

Matt Welsh

Issue #3, June-July 1994

In this editorial, Matt takes a look at what might happen when large companies get involved in the Linux community.

I'm sure that I am not the only Linux enthusiast who is kept awake at night by the following terrible vision: One day, while flipping through one of my favorite computer rags, I come upon a full-color two-page advertisement. The large caption reads, “New Technology for a New Generation”, and above it is a glossy photo of Bill Gates, proudly holding forth a box emblazened with the logo: “Microsoft Linux NT”. It gives me chills just thinking about it. Excuse me while I get another cup of coffee and a dramamine.

Are you plagued by nightmares of the commercialization of Linux? Frightened, as so many are, that multibillion-dollar corporations are going to snatch Linux away from the hands of the grunt hackers, who shed so much blood, sweat, and nutritional health to develop this system from the ground up? Does every line of code that you write, every command to you type, resound with one single purpose—“Keep Linux Safe and Legal”?

If so, there's help for you. Here at the *Linux Journal* Paranoia Support Center we specialize in calming jittery nerves and putting those fears to rest. To prove to you that our service works, this article contains the complete text for your first counseling session—free of charge. If, after reading this article, the image of a gold-toothed Bill Gates still lurks in the corners of your most frightening dreams, call us. We're here to help.

First, the problem: As everyone is well aware, the GNU General Public License has provisions which allow vendors to sell free software, even for profit. In fact, a number of companies are currently doing so: Yggdrasil, Trans-AmeriTech, and Softlanding—just to name a few—sell Linux on various media formats, generally via mail order. And, I think we would all agree, this is a Good Thing. It allows those not fortunate enough to have Internet access to still get their hands on the Linux software, at a modest fee. In fact, the prices charged for Linux CD-ROMs are becoming increasingly competitive—ranging anywhere from $199 to under $30.

Some of these CD-ROMs are simply mirrors of Linux FTP archive sites, where your mileage may vary and some complete distributions, maintained and supported by the distributor, including consulting. The Yggdrasil Fall 1993 CD-ROM manual even claims that for a flat fee of $500, they will fix the problem of your choice and send you a shiny gold CD with the problem fixed. Wonderful! You know, I've been having problems running Microsoft Windows applications under Linux—I wonder if they can do anything about that.

You may be aware that Linus didn't originally intend to license Linux under the GPL. At first, he wanted to impose the additional restriction that nobody could sell Linux for profit—a reasonable expectation, back when the only working device drivers were the console and the serial electric cheese grater. Who'd want to make money off of Linux, anyway? All it could do then was run gcc and make nachos (but, at the same time!).

Eventually, however, Linus gave into political extremism and decided to make Linux a textbook case with which to test the validity and extent of the GPL. As far as I know, Linux is the only (non-mythical) operating system licensed under the GPL, and most of the software used by Linux is covered by the GPL as well. A tasty morsel has been thrown into the sharkpool of the free software world, and it remains to be seen how well the GPL will protect the system from legal turmoil.

Thus far, the fish to bite the Linux bait have been somewhat small-mostly startup companies that are able to market the software on a small scale. Without large-scale marketing, Linux is still controlled and developed by the volunteers who brought it this far. We are still free to implement new features —or break old ones—on a whim, without pressure from any ferocious marketing megalomaniacs.

However, larger companies have started to turn a hungry eye towards Linux. Here, before them, is a complete 32-bit UNIX implementation for the PC—and it's free! The word “free” causes the heads of marketing vice-presidents everywhere to ring with alarms and buzzers. In some of them, it initiates a salivation response not unlike that in Pavlov's dogs.

You see, large companies like Microsoft and Novell have the resources and programmer-power to take Linux, squandering on the Net amongst a group of loosely-knit volunteers, and turn it into something robust, marketable, and probably a bit frightening. A recent issue of PC Week contained an announcement that Novell has plans to base a new graphical environment on Linux. With news such as this, less than a few months after the release of kernel version 1.0, others are sure to follow. Hence, the vivid nightmares of Microsoft Linux NT.

The GPL allows anyone to take Linux, modify it, and market it in whatever way that they please—as long as the modifications are also covered by the GPL. This means that Novell's modifications to Linux must be made freely distributable—perhaps by being available via anonymous FTP. There is an exception to this: If Novell's system were not to involve any changes to the Linux kernel itself, but acted as a completely separate entity (for example, a graphical system running on top of the Linux kernel), Novell would not be required to license their system under the GPL. Of course, the Linux kernel, and any modifications to it, will always be free under the GPL. But independent software which runs with Linux as a base may not be.

Like me, your head is probably buzzing with legalese.

If you're confused, grab a copy of the GNU GPL from prep.ai.mit.edu, in the file pub/gnu/COPYING. Or, if you

have a Linux system, chances are the GPL can be found in the directory /usr/src/linux, along with the kernel sources. The idea is that nobody can take Linux and do anything to it that's not freely distributable.

What does this all mean? It means that a large company, such as Microsoft, could take Linux and market it. In some sense, that's good. Never mind that hard-working volunteers like Linus may never see a penny of Microsoft's funny money—most of the Linux developers don't expect to make anything from the commercialization of their software, and that's fine.

If Linus truly felt that he was getting ripped off, he would distribute Linux under a different license. Making money isn't the important thing—hacking the system is.

This brings us to the other edge of the sword—and what a sharp one it is. In addition to marketing Linux, a large corporation could throw a team of experienced programmers at it, and pay them to develop Linux full-time. This well-paid commercial cadre, given nothing better to do, could possibly implement features in the Linux system that the disorganized network of volunteers would agonize over for months. Don't underestimate the power of cooperation. Given enough cash and a cluster of offices not ten feet away from each other—as opposed to thousands of miles, which is the current maxim—a team of programmers putting their heads together could accomplish things that those of us who only hack Linux during our spare cycles could never do in any reasonable amount of time. Of course, these modifications would be available freely, and could perhaps be incorporated into the standard kernel for all to use.

While some would welcome the commercial development of Linux, I think that it could take away the single most important aspect of the system: That it was developed by volunteers and hackers. Yes, some of those hackers are professionals in the computing industry, yet they act as hackers—not as representatives of a corporation with a vested interest in seeing Linux develop. Not in order to promote their own financial gain, but in order to promote the cause of Linux itself.

If nothing else, Linux should strive to retain its heritage, inasmuch as it has one, as a system developed by and for hackers, and volunteers in particular. (I'd just like to see how far we can get without the help of Big Business.) It's clear, however, that there's no stopping the eventual commercial development of Linux. So, what can we do? The best course of action would be to establish a healthy, working relationship between the current hacker community and the commercial development community. Because all code will be covered by the GPL, the hackers and professionals can share the fruits of each other's efforts as much as they like. The commercial expansion of Linux isn't something to fear, except for purists who may not wish to use any professionally-developed code.

Whatever happens, Linux can still remain a hackers' operating system. The do-it-yourself attitude can survive, even if commercial development plows forward. If the point behind Linux was just to produce a complete, working UNIX system, nobody would bother—there's already NetBSD. Instead, the thrust is to do it by hand, to implement a UNIX system more or less from scratch. As long as there is Linux, there will be hackers.

Ostensibly I have no problem with Linux being marketed well. Although it does put some degree of pressure on the developers. It can't do any harm, and as we have seen, commercialization is good for Linux. One thing to watch out for with respect to marketing is who claims ownership for the Linux software. It wouldn't be right for any company to make Linux appear to be their own product. Although the proper copyright notices may be buried within the kernel source, the major Linux developers deserve credit for their work, at the least.

What have we learned? Well, first of all, that there's no reason to fear the floating head of Bill. A commercialized product based on Linux would have to be freely distributable, and we can all benefit from that. However, it's very important that Linux itself remains a hackers' operating system. No problem.

Even if Microsoft develops and markets Linux NT for us, I imagine that folks like Linus and Ted will still spend night after night, staring at the console, wondering why the hell the serial cheese grater device stopped working.

Archive Index Issue Table of Contents

Advanced search

Newton's Telecom Dictionary

Phil Hughes

Issue #3, June-July 1994

Although you probably won't be looking for everything in the book you can be fairly sure that what you are looking for is there.

- **Author**: Harry Newton
- **Publisher**: Telecom Library, Inc. ISBN: 0-936648-42-2
- **Price**: $24.95
- **Reviewer**: Phil Hughes

Newton bills his book as the “world's number 1 selling telecommunications dictionary”. This new edition weighs in at 1110 pages and is totally filled with computer and telecommunications terms. Although you probably won't be looking for everything in the book you can be fairly sure that what you are looking for is there.

For example, look under “error control protocols” and you will find a very good explanation of V.42, MNP and LAP-M stretching to about a page. Or look up “group 1, 2, 3, 3bis & 4” and get another one-page explanation of various FAX standards. I even found “pentium” in the book. I had to look for PEP, Telebit's proprietary modem protocol, to stump the book.

If it has anything mildly to do with telecommunications it seems to be in the book. That's the good news. But there's some bad news as well. First, the whole book is set in Helvetica type. This works ok for short definitions but it becomes just plain hard to read for longer explanations. The type face is fairly large so that isn't a serious problem, but Harry could have made a better book and saved a lot of paper by doing the work in a serifed face and smaller point size.

The second problem with the book is an assortment of small errors. For example, the 8088 is described as the 8-bit version of the 16-bit 8080. Most anyone knows the 16-bit cousin is the 8086, not the 8080, but such an error makes me nervous that there may be some real hidden “gotchas” hidden in the more technical information. The book even defines the 80387SX as the coprocessor for the 80385SX microprocessor. Yeah, we all know that this was supposed to be the 80386SX as the book has already said but, once again, simple typos like this scare me.

The bottom line? I think Newton's book is worth purchasing, but I hope the 7th edition addresses these concerns.

Internet Public Access Guide

Morgan Hall

Issue #3, June-July 1994

Information is the basic reason for the existence of any book of this sort. Accurate, concise, and relevant information.

- **Author**: Phil Hughes
- **Publisher**: SSC ISBN: 0-916151-70-0
- **Price**: $2.95
- **Reviewer**: Morgan Hall

My daughter just celebrated her 14th birthday. With some trepidation, I asked the fatal question: “What would you like for your birthday?”

The answer was something I've been dreading for some time. “Dad, I'd like an Internet account.”

What do you say or do when the tiny waif that met you with smiles and hugs when you came home from work is turning into a beautiful and self-assured young lady? How can you guide and support a young person taking their first steps into the world without smothering them and insulating them from the experiences that they need to grow and prosper?

Then, by lucky coincidence, I was offered the chance to review The Internet Public Access Guide by Phil Hughes, a 64-page booklet containing what I hoped would be the basics she would need to know. This, then, is the basic viewpoint of this review: a concerned dad hoping to pass essential information to his child.

Information is the basic reason for the existence of any book of this sort. Accurate, concise, and relevant information. An author faces the opposing tasks of packing the greatest possible amount of information into a page of text and, at the same time, making it interesting and readable. Phil Hughes succeeds remarkably well at this difficult task. Don't expect in-depth discussions, careful examinations of every possible switch and combination of parameters to commands, or discussions of efficiency. You won't find them here. Instead, you'll find the basics that a beginner needs in order to start exploring the Internet. In short, you'll find the very information that I was looking for.

The first four chapters of this booklet are used to establish a foundation. They explain how the booklet, itself, is organized and what it contains, explain what the Internet is, define how the booklet uses terms and typography, and finally give a very rudimentary survival guide to Unix. I'm impressed. I've seen many complete books on beginning Unix that are less useful. It's not complete, and not meant to be. As the absolute minimum for setting a beginner down at a Unix system it's very sparse, but it is the absolute basics.

Chapter 5 covers the basics of electronic mail and the ELM mailer. Here the author takes what may be the wisest course and avoids the arguments over what editor to use. Unfortunately, this also leaves perhaps the largest information hole in the entire booklet (I'd be happy with a minimum vi survival guide as it's almost always available). However, criticism aside, this chapter alone may be worth the purchase of the booklet.

Chapter 6 discusses usenet news and news readers. The information on the TIN and TRN news readers is well organized, concise, and well thought-out. Here's a chapter that the novice will thumb through time after time.

Chapter 7 talks about remote system access with emphasis on telnet and ftp. The discussion of telnet is extremely informative (and after 15 years as a net junkie, I picked up a few points!). This chapter shows the reader how to use anonymous ftp, explains how to move around the local directory hierarchy, the remote hierarchy, and the difference between text and binary file transfers. Again—it's the basics. It's the stuff that the beginner absolutely must learn. The `r' programs are conspicuous by their absence (rlogin is mentioned in passing). The author showed considerable wisdom here, as it's very easy to tell just a bit more about more, and more, and more...

Chapter 8 introduces the network-wide services that are available. The reader is shown how gopher, archie, veronica, www, and wais can help him or her. Other services such as MUDs and IRC are alluded to.

Chapter 9 discusses downloading from a host computer to a home machine (a PC is assumed). It's short, sweet, and perhaps unnecessary for some users while too sparse for others. I can't argue with the author's judgment call that it must be mentioned, but here is also an area that can expand out of control.

The entire booklet is well-written and not intimidating. For a reader who is intelligent, curious, and experimental, it's just about the right amount of information to get them going on their own. The major deficiency is the lack of some sort of text editing survival guide.

My review copy now has a pink bow around it. Along with a text file that explains a little vi, it will be presented to my daughter at her birthday party tonight. I've made the arrangements for her account, and resigned myself to the fact that it will be harder to get time on my own system in the future...

Archive Index Issue Table of Contents

Advanced search

Metamorphoses

Michael K. Johnson

Issue #3, June-July 1994

Linux Journal has gone through some major changes during its first few months of existence, and Linux itslef has gone through a few. However, some things have not changed.

A New Editor

That's me. One of the changes here at *Linux Journal* is the new editor. My name is Michael K. Johnson, and I have used and hacked Linux since it was first able to boot itself late in 1991. I have made some small kernel modifications (“hacks” in Linux-speak) and maintained several programs for Linux. I started the original “man project” in early 1992, and helped form the “Linux Documentation Project” which was formed later to write a complete documentation set for Linux.

A New Publisher

SSC, publishers of UNIX reference guides and books for over 10 years, is now the publisher of *Linux Journal*. By consolidating most of the work on *Linux Journal* in one place (Seattle, Washington), we hope to avoid the early problems that *Linux Journal* encountered when it was produced cross-country between Washington and New York.

A New Linux

While *Linux Journal* was experiencing its birth pangs, Linux version 1.0 was released. Version 1.0 was not quite bug-free, and in fact was barely different from the versions that preceeded it. The biggest difference was one of philsophy.

Previous to version 1.0, a new version of Linux would come out every week or two, and many people would upgrade right away. No one would know whether or not some new feature in the new version had a bug, and some people were bitten by new bugs. This caused many others to avoid upgrading to the most recent kernel for fear of encountering a bug.

Eventually, Linux versions were discussed as if they were vintages of wine. “I have never trusted any of the patch level 13 kernels.” “Version 0.99 patch level 12 was a nice, stable kernel.” “Ah, I remember version 0.95. It was so simple, clean, elegant...” A folklore sprang up about which versions should be run by which people.

A New Philosophy

With Linux 1.0, Linus Torvalds (the main author of Linux) devised a new scheme to avoid this confusion. Linux 1.0 was the start of a line of Linux versions with no new features, and hopefully no new bugs. Linux 1.0.1 fixed a few bugs, Linux 1.0.2 fixed more, and so on. However, no Linux version 1.0.<anything> contains any new features that Linux version 1.0 did not have, and therefore hopefully will contain no new bugs. These versions are intended to be safe.

New features are now being introduced in a new set of different versions: Linux versions 1.1.x. Linux 1.1 contained many new features that were not in 1.0. Each new version 1.1.x may or may not contain new features, and is likely to contain new bugs. These new versions are intended for people who like to play with new features and are willing to risk bugs. These versions are intended to be fun.

In general, any “even” version of Linux, like 1.0.x or 1.2.x, will be intended to be safe, and any “odd” version, like 1.1.x or 1.3.x, will be intended to be fun.

An Old Philosophy

Linux 1.0 is still free; still licensed under the GNU public license. It is perhaps worth explaining what “free” means here. It does not mean free of monetary cost, although you do not have to pay to get a copy of Linux unless you want to. It means that the source code is available to anyone, and that anyone who makes changes to Linux must release the source code for their changes if they release their changes at all. This will ensure that Linux remains a completely open system.

A New Option

More and more commercial software is being made available for Linux. As Linux users keep asking software companies when they are going to release a Linux version of their product, more and more companies have decided that they need to port their product to Linux. *Linux Journal*'s presence at trade shows has increased interest among some vendors to port their applications to Linux.

Another New Option

However, other companies are scared away by the “counter-culture” appearance of Linux, confused about whether or not it is legal for them to release commercial software that runs on Linux, or simply not interested in porting their software.

Eric Youngdale, Al Longyear, and several other hard-working Linux hackers have been working on “iBCS2 compliance” for Linux. This weird-looking acronym (It stands for intel Binary Compatibility Standard) means that it is now possible to run at least some binaries intended for SCO Unix or for any of the many versions of SVR4, including UnixWare.

The SCO version of WordPerfect, for instance, runs, and even works under X. The SCO version of Xess, an X-based spreadsheet, is also known to work, and there have been reports of databases running as well. The iBCS2 team is interested in hearing success reports, which they will post in a list so that users can know whether software works under Linux before puchasing the software.

The iBCS2 patches have now been publicly released. The product isn't perfect yet-in fact, it is still in alpha testing as I write-but if you want to, you can play with it. Statically linked binaries seem to work fairly well, but the shared libraries for SCO and SVR4 are still being written, and do not work as well.

The IBCS2 patches are available from tsx-11.mit.edu in /pub/linux/ALPHA/ibcs2/ as I write, but may well have moved to /pub/linux/BETA/ibcs2/ by the time you read this. There is a README file in that directory which explains the rest of the files. The more people that try these patches, the better this iBCS2 support can get.

An Old Commitment

Linux Journal is still committed to giving its readers what they want and need. We really want your feedback. Please tell us what you like and dislike about *Linux Journal*, and what you think we should do to improve. You can send e-mail to info@linuxjournal.com, or send paper mail to *Linux Journal*, P.O. Box 85867, Seattle, WA, 98145-1867, USA. You can phone us at (206)524-8338 or fax us at (206)782-7191, if you like.

Archive Index Issue Table of Contents

Advanced search

Let's Take Linux Seriously

Phil Hughes

Issue #3, June-July 1994

I am sure some of you out there want to figure out how to start making money as a software developer or computer consultant. The general answer is “solve a problem”.

Back in the early days of inexpensive Unix systems it was a serious uphill battle to sell Unix-based computer systems to small businesses. Why? Because MS-DOS did everything they wanted. By the time these businesspeople realized they had a problem-they needed a computer system that could be accessed by more than one person at a time- Novell was telling them they needed Netware to hook together their MS-DOS systems.

Well, Novell managed to sell these MS-DOS people another operating system. Most probably didn't realize it but they were Netware users with MS-DOS “terminals”. It sold because they could keep on thinking they were running MS-DOS and it solved the problem.

I am sure some of you out there want to figure out how to start making money as a software developer or computer consultant. The general answer is “solve a problem”. In the rest of this column I am going to show how you can use Linux to solve problems and make money along the way. Get rich quick? No. Guaranteed? No. But, if we all play it right it has a good chance of success.

In the past the hurdles to selling a complete system were:

- Getting the hardware
- Getting the software
- Putting together something that was needed
- Finding customers and selling it
- Supporting what you sell

Has something changed? Not in this list, just in what you need to do to jump these hurdles. Let's look at them one at a time.

In Seattle there are almost as many places that sell computers as sell gasoline. At one time I thought there would be more computer stores than expresso stands but now that dry cleaners, insurance agents and car dealers are selling espresso I would guess not. This means that it is pretty easy to buy a computer- even one that can run Linux. Add to this the fact that computer hardware is just plain inexpensive these days and you have another advantage over the old days when you had to sell your firstborn in order to afford to buy a computer.

If you are not a hardware whiz (or don't want to be) your best bet is to talk to the owner of a computer store. Not a big chain, just a local businessperson who would be interested in your story. Tell him about Linux, about yourself and about why he will benefit. What you want is for him to supply the hardware to your customer and be responsible for it when it breaks. He can make the profit on the hardware and you don't get any of the hassles including financing and inventory.

This deal helps both of you. You send hardware customers to him. He sends people looking for multi-user solutions to you.

The second item on the list is software. This is still a problem with Linux. In fact, Linux is probably not the right answer for a general-purpose business solution yet. But there are lots of things that can be done, and in many cases, Linux is a better answer for the customer.

But first, here is why it is a better answer for you:

- Minimal costs for software-Linux is (almost) free and 100 copies of Linux cost the same as one copy.
- You can get the source code for Linux and other programs such as Ingres or Postgres that you need to work with. You may not want to hack source code but it means it can be done and, as it is available without a $100,000 licensing fee, you can probably find someone to do the hacking for you if it becomes necessary.
- It costs your customer less. If the customer has $20,000 to spend to solve their problem and you can reduce the software cost by a few thousand by using Linux on the systems instead of a commercial OS, you get to put those few thousand in your pocket, or split the difference with the customer.

What should you do?

The best place to start is with something you need or want. If, for example, you currently work for a small company that sees the Internet as a way to expand their business, offer to help them get on the Internet. Get a Linux system, an Internet connection, and get it up and running. Get a news server and an ftp server running for their announcements and bulletins. Develop programs or scripts to translate internal files into announcements suitable for electronic retrieval.

You will have solved a problem, probably got a few gold stars (and a black eye or two) and have learned how to put together something that is needed by lots of businesses. Now you have a product: your skill and expertise in getting a small business electronically connected to the Internet. Start marketing but also start thinking of what else you can do; what would be a logical additon. For me, voice mail and FAX retrieval seems like the logical complement: If the customer doesn't have e-mail they can request the same information via FAX or, if all else fails, voice mail.

Building this system is not for everyone but it is something that is useful and needed. If, instead, you are an accountant with C programming experience, you might want to consider some accounting software. Or start with the Ingres or Postgres database and build an application that solves a problem for your business.

As Linux grows it will start to appeal more to commercial software vendors. This means that you can either get in on the ground floor building applications you need (and making money with them) or wait for whoever the new “Lotus” is to appear as the application provider for the Linux market.

For those of you who just want to use applications, start writing letters to *Linux Journal*. Tell us what you want and why. There is a lot of talent out there ready to write code. If you can identify what is needed, someone may provide it.

Stop the Presses

Michael K. Johnson

Issue #3, June-July 1994

Some breaking news, some old news in a new light, and some shameless preaching...

Have you ever wanted to send e-mail to a Linux developer, but not known what e-mail address to send your mail to? Mike McLagan, with help from Liem Bahneman, has created a new Internet domain called linux.org. If you want to send e-mail to Alan Cox (who is in charge of Linux networking development), but can't remember his e-mail address, you can send it to Alan.Cox@linux.org and it will be correctly forwarded to him. You can send mail for me to johnsonm@redhat.com, and for Linus himself to torvalds@transmeta.com. Pat.Volkerding@linux.org is the maintainer of Slackware, and Mike.McLagan@linux.org maintains this service. If you need to find out what aliases exist, you can get a list of the aliases from ftp.linux.org in the file /pub/info/alias-list. Linux developers only may request an alias from alias-request@linux.org.

Other Services

World Wide Web (WWW or W3), a distributed hypertext application which organizes information across the entire Internet, (see the article by Bernie Thompson on page 9 for an introduction to WWW which explains of all the terms in this paragraph) includes quite a bit of information about Linux. linux.org provides a set of Linux documents all together in one place. The URL is www.linux.org. [The site can now be found at www.linuxresources.com] I was very suprised as I browsed the WWW pages on www.linux.org to find connections to a HTML version of my own book, the *Linux Kernel Hackers' Guide*, available. There is a lot out there, and it is worth exploring if you have a connection to the Internet.

Another service that linux.org will provide is ftp service from ftp.linux.org. As I write this, the ftp directory on ftp.linux.org is mostly empty, but Mike is asking for help from Linux developers to change this. Mike is considering several other services, which will be announced in forthcoming issues of *Linux Journal*, and is taking suggestions at Suggestions@linux.org.

Non-Intel Linux

When Linux first came out, Linus said that Linux would be very difficult to port to other architectures. Everyone paid attention when he said this, but no one appeared to notice when he later said that Linux had become much more portable; as portable as most versions or clones of Unix.

For over a year and a half, work has been under way on a port of Linux to the Motorola 680x0, specifically the Amiga. Work went in fits and starts until Hamish MacDonald announced his first port, but since then progress has been rapid. It is now starting to work well, at least for hackers, and supports many common devices. The X Window System has not yet been ported, but Linux/68 compiles itself, has shared libraries, and runs emacs. Join the **680X0** channel of the mailing list if the Linux/68 port interests you. Send an empty mail message to linux-activists-request@niksula.hut.fi for instructions on how to do this.

In the last few weeks, a version for the Atari TT and Falcon has been begun, and now is booting itself. Soon all of these 680x0 platforms should be supported by a single kernel, or at least be built from the same source. Linux/68 has most of the features of Linux/i86 (standard Linux), and is still under development. It requires a 68020 with a seperate MMU or a 68030 or 68040. 68000 and 68010 are not and will not be supported.

Recently, Jim Paradis of DEC announced that he has started work on a port of Linux to the DEC ALPHA chip, and two college students have announced that they have begun work on a port to the PowerMac. DEC has offered to loan Linus an ALPHA to work on an ALPHA port, and IBM has offered to loan Linus a PowerPC to work on a PowerPC port. There is a lot of interest in ports of Linux to new architectures; not only on the part of hackers and even general consumers, but now also by some of the largest companies in the industry.

Brave New Linux

The Linux groups (especially comp.os.linux.misc) have been aflame in the last few weeks with rumors about Novell developing a complete Unix-like environment based on the Linux kernel. These reports, mostly fueled by reports in PC Week, have not been confirmed, but they do demonstrate that interest in Linux is spreading in the “conventional” market and that the “major players” the the computer industry have taken notice.

Even if the rumors are not true (although they sound convincing), the fact that PC Week named Linux its “Product of the Week” is major recognition, since PC Week is heavily oriented towards DOS and Windows, and has historically shown little interest in Unix or unix-like systems.

What can you do to encourage this recognition? I suggest that whenever you order hardware for your Linux box, that no matter who the vendor is, you explain that you will use the hardware with Linux, and ask if the hardware is supported under Linux. Be willing to explain what Linux is if they don't know. Several times, I have had an experience something like this: “Does that work with Linux?” “What's Linux?” “A free Unix-like operating system.” “Free?” “Yes, free.” “How can I get a copy?”...

If you are interested in commercial software applications being made available for Linux, call the company and ask if a Linux version is available. If enough people ask, the company will get the message—software companies do not make a living by ignoring large groups of potential customers. On the other hand, remember that they do not make a profit by making a port that only one person will buy.

Michael K. Johnson is a linux activist, programmer, minor systems administrator, and author who lives with his wife in Chapel Hill, North Carolina. Michael is the new editor of *Linux Journal*.

New Products

LJ Staff

Issue #3, June-July 1994

Complete Linux Package from SSC and FinSim for the Linux platform available.

Complete Linux Package from SSC

SSC is offering a complete package to get you into “the Linux experience”. The package includes a Linux Distribution on CD-ROM, a certificate for a 1-year subscription to *Linux Journal*, printed copies of Matt Welsh's Linux Installation & Getting Started, 3-hole punched copies of the Linux How-Tos and four of SSC's own publications (Internet Public Access Guide, Beginning Unix command Summary, RS-232 Reference and Bourne Shell Tutorial).

The package sells for $88.95. SSC, P.O. Box 55549, Seattle, WA 98155, (206) 527-3385 or e-mail to info@linuxjournal.com for more information or a complete SSC catalog.

FinSim for the Linux platform available

Fintronic USA, Inc. is pleased to announce the availability of FinSim for the Linux platform. FinSim: A high performance, compiled and interpreted simulation environment that supports both UDL/I and Verilog HDL. The FinSim UDL/I simulator features full language implementation. The FinSim Verilog simulator features full Verilog HDL implementation including gate, switch-level, user defined primitives, behavior, specify blocks, path delays, system tasks and functions, PLI, and VCD.

Currently FinSim runs on Sun Sparc, Digital MIPS, IBM RS/6000 and SGI MIPS workstations. It is also available on Intel x86 platforms running Windows NT, SVR4.2, interactive Unix 3.2 or Linux. It will soon be available on Digital Alpha, Pentium and Hewlett-Packard platforms.

FinSim prices for a Unix workstation range from $15,000 to $25,000 including free upgrades to SDF. The list price for FinSim for a PC platform is two thirds of the price of FinSim for a Unix workstation. Contact Fintronic directly for available discounts and terms.

E-mail: info@fintronic.com fax: +1.415.325.4908 tel: +1.415.325.4474 mail: Fintronic USA, Inc. 1360 Willow Rd., Suite 205 Menlo Park, CA 94025 USA

Fintronic also sells pre-installed, fully configured Linux workstations and notebooks. For information on this service please finger or send mail to linux-sales@fintronic.com.

Innovative Uses of the World Wide Web

Archive Index Issue Table of Contents

Advanced search