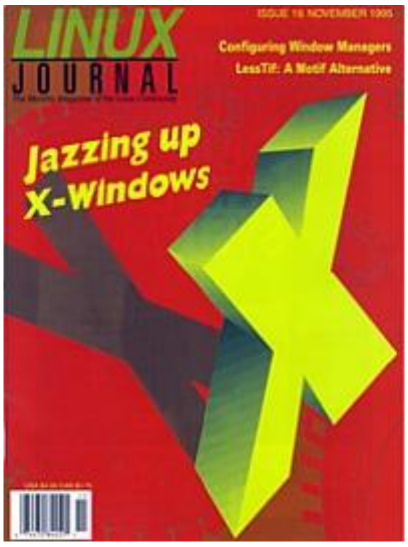
Advanced search

Linux Journal Issue #19/November 1995



Optimizing Linux's User Interface

Jeff Arnholt

Issue #19, November 1995

Jeff provides a handy guide to X-Windows window managers and tells you how to customize them.

One of Linux's most impressive but least recognized features is its flexible and powerful desktop environment. While great efforts are underway to develop the next generation of desktop interfaces (such as OSF's Common Desktop Environment or Microsoft's Windows 95 and Bob), Linux enthusiasts have for years had a choice of several stable, powerful, and customizable window managers and shells for program management.

Unlike many other operating systems, Linux runs extremely well on standard VGA 80x24 terminals. While unglamorous and less intuitive, Linux's terminal interface provides speed and ease-of-use still unmatched by GUIs. Linux uses mature, powerful shells such as tcsh and bash, which elegantly display and manipulate text at blazing speeds on any PC. Linux's text mode support is particularly appropriate for portables with small screens, slower processors, and little memory. [See “The Best without X” in this issue, page 22—Ed]

The reduced requirements for character-mode displays also enables Linux to run very well off of floppy drives, which has great utility for creating bootable emergency repair disks or running Linux on non-Linux PCs. Linux's text mode support is inappropriate for graphical applications like Netscape or WYSIWYG word processors, but is superior for system administration and text editing tasks due to its versatility, speed, and uniform support.

At the other end, Linux supports a wide number of X-Windows servers on large monitors at resolutions up to 1600x1200. Linux's X-Windows client-server technology offers the superb capability of displaying graphical applications locally which run remotely, a feature unmatched by other PC operating systems. X-Windows is also notable for the extreme amount of customization available through resource files. For example, as a left-handed mouse user I commonly switch the scrollbars of my X-Windows applications to the left side, which cannot be done with other operating systems. With native support for three-button mice, multiple displays, and a myriad of free window managers, Linux provides more powerful and flexible desktop interface options than most other operating systems.

Unfortunately, the Linux interface suffers from the inevitable tradeoff of flexibility and power for ease-of-use. Many Linux users continue to use default X-Windows or shell configuration files, unaware of alternative interface tools or configuration options which can simplify or enhance their current setup. I have encountered many Linux users who are unwilling to modify their installation because of arcane configuration file syntax and lack of knowledge on this subject. This problem continues to increase as more people without Unix training migrate to Linux.

This article describes strategies for desktop interface configuration and utilization under Linux. These strategies are based on a simple premise: the optimal desktop environment is one which maximizes data visualization and screen utilization (output) while minimizing the amount of time, interaction, and complexity required to perform a given task (input). Utilities which are frequently used must be immediately accessible to users with a few keystrokes or button clicks. Less commonly used utilities need not be as accessible but should be well-organized such that they are easily located and started. Utilities should optimize screen usage to permit monitoring and interaction with multiple, concurrent processes. Finally, all interface tools must make minimal usage of memory and CPU resources.

Having tested the majority of Linux window managers and shells over the past several years, I am most impressed by rxvt and fvwm, John Bovey's and Robert Nation's (fvwm@wonderland.org) xterm clone and window manager, as tools to manage my desktop environment. I use Tcsh, written by multiple authors, within rxvt, although Bash, GNU's “Bourne again shell”, is a fine alternative.

The Command Line

While I prefer most graphical programs to their command line alternatives (ftptool instead of ftp, for example), terminal windows continue to provide three important uses. First, most of my routine system administration tasks are performed with homegrown perl and shell scripts running in a term window. While many fine GUI program builders are available (such as Tcl/Tk), it is still simpler to program shell scripts for command line input and output. Complex networking, file management, searches, or other system activities are most easily performed in this manner. Second, even as SLIP and PPP become more widespread, wider support and better responsiveness exists for terminal-based serial line communications.

Finally, terminal windows usually provide much faster interaction than GUI alternatives. It is much easier and faster to type “date” at the command line than to open an Xclock (which requires more keystrokes), move it and/or minimize it, read it, and then close it. In addition, programs which do not require much user interaction are best run as command-line applications to avoid the additional resources and programming complexity required by graphical applications. Linux's terminal windows serve as master interfaces for all programs without sophisticated input/output requirements and eliminate the need to open multiple windows for different tasks.

Rxvt

I prefer rxvt to xterm as an X-Windows terminal emulator. Rxvt provides a subset of common xterm features with substantial memory savings. The amounts of physical memory required by xterm and rxvt, as reported by **top** (a program that displays CPU activity), are 2120KB and 560KB, respectively. Since I commonly run three to four X-Windows terminal sessions simultaneously, I save approximately 6MB of memory by using rxvt. Rxvt does not support Tektronix 4014 emulation or session logging, but this is seldom a concern.

Tcsh

Tcsh has enough features to merit another article, so I will concentrate on just a few aspects with respect to its user interface functions. Most notable is its excellent support of Emacs-style command line editing. Typographical errors are easily corrected by moving the cursor to the error with the arrow keys and using editing commands such as **ctrl-D** or **ctrl-K** to make the modification. Previous commands can be edited in the same fashion by first scrolling back through the history list using the up arrow. Filename completion is enabled by putting **set filec** in .cshrc (the tcsh configuration file located in the user's home directory), which enables the user to type in just a few characters of a long filename and press **tab** to complete the remainder (assuming those characters are unique). This feature has become so useful and natural to me that I find myself constantly pressing **tab** in situations which do not support it, such as remote ftp session or when I need to work under MS-DOS. As an example, If I want to view the contents of text file OptimizingLinuxUserInterface, I simply type **cat Opttabreturn**, and the filename is completed for me—unless there is another file with those first three characters. In that case, I need to supply additional characters.

Tcsh also has advanced ways to set the prompt, which I use to keep track of the present working directory (indentation is important). From my .cshrc:

```
:set prompt = '\n%B%n@%m%b:%/ %h %# '
```

gives: **root@regal:/home/root 27 #**

when working as the user root on the machine regal in the directory /home/ root with a current history command number of 27.

I also like the **ctrl-alt-Z** shortcut tcsh provides to allow me to jump from the shell to a suspended version of Emacs and **ctrl-Z** to bring me back to the shell. It has eliminated my dependence on Emacs's internal shell, which displays some non-standard behavior regarding key assignments.

Having been accustomed to command.com for many years, my favorite feature of tcsh and other Unix shells are aliases. As have most Linux users, I've aliased extremely common commands such as **ls, ls -l, rlogin**, and **less** to **l. ll, r**, and **t** (short for type). This is easily done in .cshrc with lines like

alias rlogin 'lr'

Here are a few useful and representative selections from my .cshrc for file viewing, process logging, telnet access, floppy formatting, and remounting the hard drive if it boots read-only (note: the last one comes up all the time frantically in Usenet's comp.os.linux hierarchy).

```
  :
  alias t             'expand -5 \!* | less'
  alias pss           'ps -au | grep $user | less'
  # Syntax: pss <user>
  alias archie        'telnet archie.internic.net'
  alias formatfd0     'echo "Using ext2 filesystem."
                       fdformat /dev/fd0H1440;
                       mkfs -t ext2 /dev/fd0H14440'
  alias remountroot   'set temp = $PWD; cd /;
                       mount -w -n -o remount /;
                       cd $temp'
```

Excellent shell tools such as tcsh and rxvt are one reason why I consider Linux's interface to be superior to most non-Unix operating systems. DOS, Windows, NT, and even the Mac do not offer this powerful and fast means for program and system interaction.

Fvwm

Fvwm is an excellent choice for a window manager, providing most or all of the functionality of Motif in far less memory (like rxvt compared with xterm). Fvwm was derived from twm code, but designed to use fewer system resources. Informal tests with top show that fvwm uses 700KB of physical memory (RSS) compared to OpenLook's 900KB, twm's 1700KB, and mvwms's (Motif) 1900KB. Not even the author still remembers what fvwm stands for, although I've often heard “feeble virtual window manager”. This is an unfortunate name, since fvwm is far from feeble. [I suggest that this is why the author chooses not to remember what it stands for. —Ed]

Fvwm employs a virtual desktop like olvwm with a maximum size of 32KB pixels squared (I doubt anyone would ever take advantage of such gargantuan proportions), and has a look and feel very similar to Motif. Sticky windows can be assigned which stay on the screen regardless of which virtual desktop is currently being viewed. As with Motif, resizing windows invokes a helpful grid showing the changing dimensions of the window. Shaped windows are supported, but increase memory utilization by 60KB (I don't use these).

Fvwm utilizes “modules”, a concept not shared by other window managers. Modules are separate programs which may be independently developed, yet which are integrated into fvwm. Modules run as separate Linux processes, spawned by fvwm such that a pair of pipes are used to transmit commands back and forth for execution. While this design may be superior for window manager programming, it is particularly important to the average Linux enthusiast because it provides more functionality than most window managers.

Fvwm's man page is very long (36 pages) and complete. More important, the sample configuration file which comes bundled with fvwm is crammed with comments and useful defaults. Fvwm is like other window managers or X-Windows applications in that great control may be exercised over almost every aspect of display. Window color, border size, and window behavior are easily specified in the .fvwmrc or .Xdefaults file. Specific directions for such modifications are available in the default .fvwmrc file. Configuring fvwm should not be difficult.

I find that the most useful desktop interface feature offered by fvwm (and most other window managers) is its root menu. Customization of the root menu provides easy access over almost every aspect of the system. On my Linux workstation, the first five root entries (Apps, Docs, Desktop, Network, and System) are pointers to submenus.

Apps contains all productivity X-Windows applications, like xemacs, xv, etc.

Docs are common ASCII documents which I constantly need to access, such as my to-do list. I also use this submenu to organize the many, many ASCII configuration files used by Linux. Since I have Emacs running at all times, the .fvwmrc statement controlling this entry runs emacsserver:

```
Exec "To do list" exec emacsclient /home/root/todo &
```

Emacsserver then loads the running Emacs with my document and switches buffers to display the new document. This method saves memory and minimizes the number of windows on my screen.

Desktop modifies the appearance of the screen and is used to invoke screen savers and change backgrounds. For example, I can call each of the different xlock screen savers in my .fvwmrc with:

Popup "DesktopScreensavers" Exec "Fractal Flame" exec xlock -nolock -nice 0 -mode flame & Exec "Game of life" exec xlock -nolock -nice 0 -mode life & ... EndPopup

I can load and automatically expand any graphic to serve as background wallpaper with:

Exec "Starry skies" xv -root -max -quit /data/wallpaper/VanGogh1.jpg

or create a texture (a small graphic which may be tiled to give an interesting background) with:

Exec "Red brick wall" xv -root -quit /data/wallpaper/redbrick.jpg

Network runs any of the common browsers or utilities, such as FTP, Archie, Gopher, etc. I also use this menu to access the different network daemons currently running.

System lists a wide variety of utilities involved with system administration. For those of you familiar with Windows 3.11, this entry is similar to, but substantially more powerful and customizaable than, the control panel applet. One entry in the submenu runs my perl backup script in a new rxvt window; another opens the excellent keyboard mapping utility xkeycaps:

Exec "Keyboard mapping" exec xkeycaps -keyboard DELL &

Other entries in my root menu start the screen blanker (I have a Nokia monitor which powers down upon receiving a black screen), refresh the window, and exit fvwm.

For an excellent (albeit expensive) reference to extensive root menu customization, see *The Shell Hacker's Guide to X and Motif* by Alan Southerton.

GoodStuff

The **GoodStuff** iconbar is truly one of the most useful Linux utilities I have ever found. GoodStuff is a fvwm module and only runs under fvwm. GoodStuff is an iconbar extremely similar to that found on the NeXT and to a lesser extent like Window's Dashboard or RipBAR. GoodStuff uses approximately 500KB of memory and very little CPU resources. The configuration file is the same as fvwm, .fvwmrc.

GoodStuff's primary use is simply to assign iconic buttons to commonly-used applications so that they may be started with a single mouse click. For example, I have rxvt terminals assigned to each machine in my home network which I can immediately log into. This is substantially faster than my old method of typing **xterm**, moving the mouse into the window's field of view, clicking for focus, typing **rlogin** ***`machine name`***, and then entering my password.

I also have common utilities, such as my mail utility, ftptool, Emacs editor, netscape, file manager, and so forth attached to individual buttons. I limit the GoodStuff iconbar to a dozen buttons (laid out in a 2'6 matrix), because each one takes valuable screen real estate, especially since I have it set permanently in the foreground. Rxvt, my most commonly started application, is not included in GoodStuff but instead is mapped to the middle mouse button on the root window. I can immediately pop up a new rxvt window by simply clicking the second button on the background wallpaper. Less commonly used programs are attached to the root menu (as described in the previous section).

Button bars are hardly novel. What **is** special about GoodStuff is that one may assign running X-Windows applications to each button and use the button as the display. For example, I have xload running as a 2x1 button at the top of the GoodStuff menubar, and it displays just as it would in a small window. I have xbiff in another window, which alerts me if mail has arrived (the button's color becomes inverted, and it is very noticable). I even have a less well-known but equally useful X-Windows app called xosview which monitors instantaneous CPU, memory, disk, and network usage as a small colored bar graph. It is very helpful for me to watch this program running in the GoodStuff button bar to see when I'm taxing the network or CPU or running out of memory or disk space. All I needed to do to incorporate xosview into GoodStuff was the following line in the .fvwmrc:

***GoodStuff - whatever Swallow "xosview" xosview -bg grey -geometry 210x96-1500-1500 &**

I also have fvwm's 2x2 virtual desktop at the bottom of the GoodStuff menu bar. I don't use it all that often, but it is a handy feature when needed.

Other fvwm modules exist, including Pager, Banner, WinList, Clean, Ident, Save, Scroll, Debug, and Sound. I don't use them as much as GoodStuff, but they are all useful utilities.

Conclusions

The strategy described above uses fvwm, tcsh, and other utilities to generate an effective desktop interface to manage programs, data, and system resources. While lacking in certain features, such as drag-and-drop desktop tools and object-oriented metaphors, the combination of these tools creates a desktop which is more flexible, customizable, and powerful than competing paradigms. Current versions of these tools are freely available at many Internet sites including ftp://sunsite.unc.edu/pub/Linux.

Jeff Arnholt is currently developing X-based biomedical imaging packages at the Mayo Clinic in Rochester, Minnesota. He is a medical and graduate student who hopes to earn his MD/PhD degrees by 1997. You may contact him at arnholt@mayo.edu.

Archive Index Issue Table of Contents

Advanced search

LessTif and the Hungry Viewkit

Malcolm Murphy

Issue #19, November 1995

The efforts of the Hungry Programmers are making the Motif widget set available for Linux users.

One of the nice things about running Microsoft Windows is that all your applications look and feel the same. You know that there will be a bar at the top of the window with some pull-down menus, that Alt-F4 will quit, Ctrl-X will cut, Ctrl-V will paste, and so on. Even with a brand new piece of software, everything looks familiar, and you don't have to spend time getting used to a new user interface before you can start to use the application.

Compare this with the X world, where it seems that no two applications look the same, and you often have to spend a few minutes familiarizing yourself with the controls, even with programs you have used before. Why is there such a difference in the appearance of X applications, when it could be argued that the Graphical User Interface (GUI) of a program is one of its most important parts?

The most basic tool for X programmers is Xlib, the X programmer library. It effectively hides the details of the X protocol from the programmer in a library of C subroutines. However, Xlib routines are very low level. The idea behind Xlib is to give the programmer a convenient environment while offering the full flexibility of the X window system. In particular, the programmer has to implement his/her own user interface. There are two important consequences of this. Firstly, there is a lot of work for the programmer, since common components such as scrollbars, buttons, etc. have to be written from scratch. Secondly, it means that different applications look and feel very different, since programmers implement their own GUIs according to their own personal taste.

The designers of X addressed the first problem by providing Xt, the X Window toolkit. This provides a set of functions that handle the user interface and other X-related sections of an application program, such as window creation and event handling. Xt provides routines at a higher level than Xlib, making life easier for the programmer. Although Xt is written in C, it has an object-orientated approach to the problem. The functions in Xt appear as self contained program units called widgets. Widgets are arranged in classes, and each widget class performs one particular type of task. Some widget classes provide the on-screen GUI components (scroll bars, buttons etc.), while others assist with managing the overall layout of the GUI. For example, the HTML interpreter of NCSA Mosaic is implemented as a (very complicated) widget.

Xt consists of two layers, the Intrinsics layer which provides an environment in which widgets can be created and managed, and the widgets themselves, which are usually packaged as a widget library. Several widget libraries exist, the most commonplace being the Athena widget library, which is provided as a part of X as an example widget set. The Athena widget library has been widely used, especially in public domain software, because it is widely available and free. Unfortunately, it consists of only a few basic widgets, which are often considered to be quite ugly.

A well developed and popular widget set is provided by the Open Systems Foundation as part of OSF/Motif. Motif provides literally hundreds of widgets, and is used in a lot of X application software. The recent cross-vendor COSE agreement describes a GUI style which is heavily based on Motif. One possible consequence of this agreement is that we could start to see X applications with a consistent look and feel, even though they are running on different hardware with different operating systems. This would help shake off some of the antipathy that is directed at the X window system.

There is a danger that all this could pass the Linux community by. Motif is typically bundled with commercial Unix/X systems, but since it is a commercial product, it does not come as part of XFree86. Several versions of Motif are available for Linux users, retailing in the region of $100—$200, but it is probably fair to say that most Linux users are quite happy to do without Motif, rather than pay for it. After all, we have a free operating system, a free C compiler, a free DOS emulator, a free windowing system, and so on. $100 for Motif? No, thanks.

However, Motif is already the de facto standard in the rest of the Unix world, and the reliance on Motif is likely to increase with the COSE agreement. While it is possible (depending on their licensing arrangements) for vendors to distribute statically compiled binaries of applications that use Motif, it would be preferable for everyone to have their own copy of the library. It is highly unlikely that the OSF is going to make Motif available free to Linux users, but this is where LessTif comes in.

LessTif is an active project of a group called the Hungry Programmers. It is intended to be a free widget set with exactly the same look and feel as the Motif library. More importantly, it will be source code compatible with Motif, so that the same source will compile with both libraries and work exactly the same. At the time of writing, the current focus is on getting the functionality of the Motif 1.2 widgets. When that is done, the intention is to add some of the features of Motif 2.0, and possibly other extensions. The speed of development of the code is quite astonishing. New releases are made weekly, and there is an active mailing list where patches and improvements are sent in the meantime. The main developer of LessTif is Chris Toshok of the Hungry Programmers, but many others are contributing to the project, and new contributors and bug reports are welcome.

The developers of LessTif are working only from descriptions of Motif 1.2 available in books and header files—they have no access to the “real” Motif source. Some of the developers have Motif libraries, so they cay compile applications against both libraries and compare the results. At the time of writing, LessTif still is at alpha status—we are still a long way from being able to compile and run most Motif applications using LessTif. There is a list of programs known to compile, link, and run with LessTif, but it is quite short at the moment. Other programs compile and link, but are still a long way from being usable. It is unrealistic to expect the developers to be able to give firm release dates, since for most (all?) of them, LessTif is a spare time project, but the developers hope that a usable release will be available in the reasonably near future; perhaps even this year. It looks very likely that the LessTif project will eventually succeed in its aims, and that Linux users will have access to a free Motif-like library.

The combination of the Xt toolkit with a suitable widget set still only addresses part of the problem. It provides the programmer with the building blocks for a user interface, but the responsibility for actually constructing the interface still lies with the developers of each individual application. In recent years, users have come to expect many features from their programs, such as context-sensitive on-line help, drag-and-drop facilities, and inter-application communication—and they expect to see them work the same way between different applications.

While there are guidelines for constructing such high-level elements, such as the OSF/Motif style guide, these guidelines are often written without consideration of the tools available to the programmer. Without a suitable set of tools with which to work, each developer will inevitably interpret the guidelines differently as they implement a user interface from the lower level building blocks provided by Xt and a widget library. Another factor is that the widget set being used may pre-date the style guidelines the programmer is attempting to follow, so that the widgets available may not be suited to the tasks at hand. What is needed, then, is a still higher-level toolkit which is tailored to the specific task of providing a GUI, and which enables the programmer to provide the features the user expects in a consistent fashion. These higher level objects are known as application frameworks.

Application frameworks have been used successfully on Macintosh and Windows systems, which is why there is such similarity between applications in each of those environments. Until recently, application frameworks have not been common in Unix environments, but that is beginning to change.

Silicon Graphics provides an application framework called the IRIS ViewKit (TM) with their workstations. The ViewKit is not intended to be a stand-alone library, but instead is meant to be used in combination with the Motif library. This gives software developers the ease, power, and consistency of the higher level objects, but allows them the full low-level flexibilty of the Motif widgets should they need it. This approach helps avoid the main danger of application frameworks—that the desired behaviour of the application has to be compromised in order to work within the framework.

As workstation vendors provide these application frameworks, the gap between the Unix environments and the personal computer environments should begin to narrow. Again, there is a danger that Linux will become the poor relation of the Unix world.

The Hungry Programmers are also busy attempting to ameliorate this problem by developing a free application framework which can be used on Linux. Called the Hungry ViewKit, it is a C++ class library for developing Motif applications which follows the API of the IRIS ViewKit. It is intended to be a superset of the Silicon Graphics kit, so that all code developed for the IRIS version will work will the Hungry kit, but not necessarily vice versa.

The LessTif project is in fact a spin-off from the development of the Hungry ViewKit. While working on the Hungry ViewKit and XWord (a word processor being developed for the X window system) the Hungry Programmers felt that there was too much reliance on the Motif widget set, and decided that they should implement a look- and feel-alike.

At the moment, there is no documentation available for the Hungry ViewKit, but the XWord source is available as an example of its use. Work on XWord has been given a higher priority than fixing bugs on the ViewKit. Thus, while a release of the Hungry ViewKit is available, it is not yet for the faint-hearted.

[Author's Note: As is the nature of the Web, the URLs I gave in the original article have changed. I believe the revised URLs below are correct.]

The Hungry Programmers have a home page, at http://www.hungry.com:8000/ or can be emailed as hungry@uidaho.edu. The LessTif home page is at http://www.hungry.com:8000/products/lesstif/ and you can subscribe to the LessTif mailing list by e-mailing majordomo@hungry.com with the request **subscribe lesstif** in the body of the message. Alternatively, the request **subscribe lesstif-digest** to the same address gets you the list in digest form. There is also a LessTif documentation project at the URL http://www.hungry.com:8000/products/lesstif/Lessdox/ The Hungry ViewKit home page is at www.hungry.com:8000/products/viewkit

Malcolm Murphy still wishes that he had discovered jazz before he gave up clarinet lessons at an early age. He considers himself too old (or too lazy) to start again now, so he plays the guitar instead. If you have an uncontrollable urge to send him some e-mail, his address is Malcolm.Murphy@bristol.ac.uk.

Archive Index Issue Table of Contents

Advanced search

Getting the Most Out of X Resources

Preston Brown

Issue #19, November 1995

Always wanted to change the look of X Windows? Here are the tools to do it easily.

Do you ever wonder how some peoples' xclocks and xterms always start up with different colors than boring old black and white? Do you wish those Athena 3-D widgets (discussed in *Linux Journal*, issue 15) looked a bit more like the Motif ones they are supposed to emulate? Through the magic of something called X Resources, you can make all this happen—and a lot more.

Resources may look confusing and complex at first glance, like something only a programmer with a mean streak would foist upon the unwitting user. It's true that resources are very closely tied to X programming, which is why they can appear so arcane. However, with a little practice, you can customize applications to suit your own personal preferences, and kiss “vanilla X” goodbye forever.

What are X Resources Useful For?

Color has already been mentioned, but to dismiss resources as a simple color control mechanism would be a mistake. Don't like an application's choice of fonts? Change them. Want to change the way the cursor appears? You can do that too.

Resources can even be used to modify the entire behaviour of an application, from the appearance and labels of buttons and pull-down menus, to the actual functions that these items call in the program. For instance, if you think that the “Quit” label on a button is too bland, you can easily substitute “Kill”, or something even more imaginative.

Resource Files

X Resources are stored in several locations. Applications often have a default set of resources that are stored in a file in the /usr/lib/X11/app-defaults directory. For example, you will find resource files for both xterm and xload in that directory. Notice the files usually have the first two letters capitalized, i.e. XTerm, and XLoad. While this is only a de-facto standard, it is related to the applications class name, which we'll get to later.

Besides these application-specific files, there are two other files in which you can store resources. If you have a file in your home directory called .Xdefaults, it will be loaded when your X session starts (either through xdm or startx), replacing any system-wide resources that might have been defined (stored in /usr/lib/X11/Xdefaults). A second file, .Xresources, doesn't replace the system defaults but instead is merged with them. For that reason, you probably want to use this file for your own resources instead of .Xdefaults.

Specifying a Resource

Resources are specified in one of the following formats:

```
name*variable: valuename.variable.variable: value
```

Resources are case sensitive, and you should be sure that there is at least a space or a tab after the colon, before the value specified. The first format listed, which utilizes the **`*`** separator (called loose binding), is used to indicate that all resources of a program with name ***`name`*** and variable **`variable`** are to acquire value ***`value`***. For instance, the resource

```
Xedit*font: 7x14
```

will cause xedit to use a 7x14 fixed font for everything, including the main window and the menus. (We will see how to find resource names later in the article—take them on faith for now.) But what if you only want to change the font in a particular area of a program? In that case, the **`.`** notation (called tight binding) would be used. This notation allows you to be more specific than loose binding allows:

```
Xedit.Paned.Label.font: fixed
```

will make two labels in xedit use the fixed font. However, the default font (7x14, as set by the first resource) will be used everywhere else. Note that although the first resource applies to **every** font resource in the Xedit program, including those two labels, the second resource is the one used for those labels because it is more **specific**. More specific resources are always used in preference to less specific resources.

Resources, and the **widgets** they modify, have a hierarchical nature. “What's a widget?” you ask. Widgets are the basic building blocks, or objects, of X programs. Some example widgets with which everyone is familiar: scrollbars, text-entry fields, buttons, and checkboxes. In the above resource, the Label widget is a “child” of the Paned widget, which is a “child” of Xedit. This will become more clear when the **editres** program is introduced below.

One other note before we continue: a distinction must be made between classes and **instances** of a particular class. In the above resource, **Label** specifies any instance of the widget class **Label**. There are two that match the specification, whose names are **bc_label** and **labelWindow**. All **classes** begin with a capital letter by definition; **Label** is the class, and **bc_label** and **labelWindow**, which start with lower case letters, are the instances of that class.

You can specify instances as well as classes, so that only one particular widget is affected; you can add the following resource:

```
   Xedit.paned.bc_label.font: 7x14
```

which will set the font for one of the two **Label** widgets back to 7x14—it is more specific than the previous resource.

It is usually more convenient to set the resources for an entire class of widgets than for an individual instance, as you typically want to make the entire application look consistent.

Viewing, Changing, and Loading Resources

Several times, you've been told to add or modify a resource. How is this accomplished? The X distribution provides several programs you can use to add, modify, and view resources. The most simple method (but perhaps the most difficult) is to specify the resource for a program as an argument when the program is started. Most well-behaved X programs accept the **xrm** command line option for adding or modifying a single resource. The format works like this:

```
   -xrm "resource"
```

Specifying resources on the command line can become tedious, to say the least. X provides considerably more sophisticated mechanisms to modify and examine resources. The most simple of these is **xrdb**. Xrdb is a command line utility that can load, query, and merge resources. Here are some of the common command line options:

- **-load** ***`filename`*** Load the resources contained in ***`filename`*** into the resource database. Replaces any resources currently in the database with new values if they are specified in the file.
- **-merge** ***`filename`*** Performs much like the load option, but only loads those resources which are not already modified. Nothing currently in the resource database that is encountered in the file will be loaded.
- **-query** Show the resource database that is currently in use. Only those resources that have been modified are displayed. If all resources were displayed (including defaults), then you would probably have much more information than you expected.

Another very useful program that comes with the X distribution is **editres**. Editres can be used to interactively modify the resources of a particular program. After starting editres, pull down the “Commands” menu, select “Get Widget Tree”, and then click on the application you want to examine or modify. Now you should see something like what is pictured in Figure 1 (the actual hierarchy pictured is for **xedit**). Note that it won't look exactly like Figure 1, because I've modified my resources for editres.

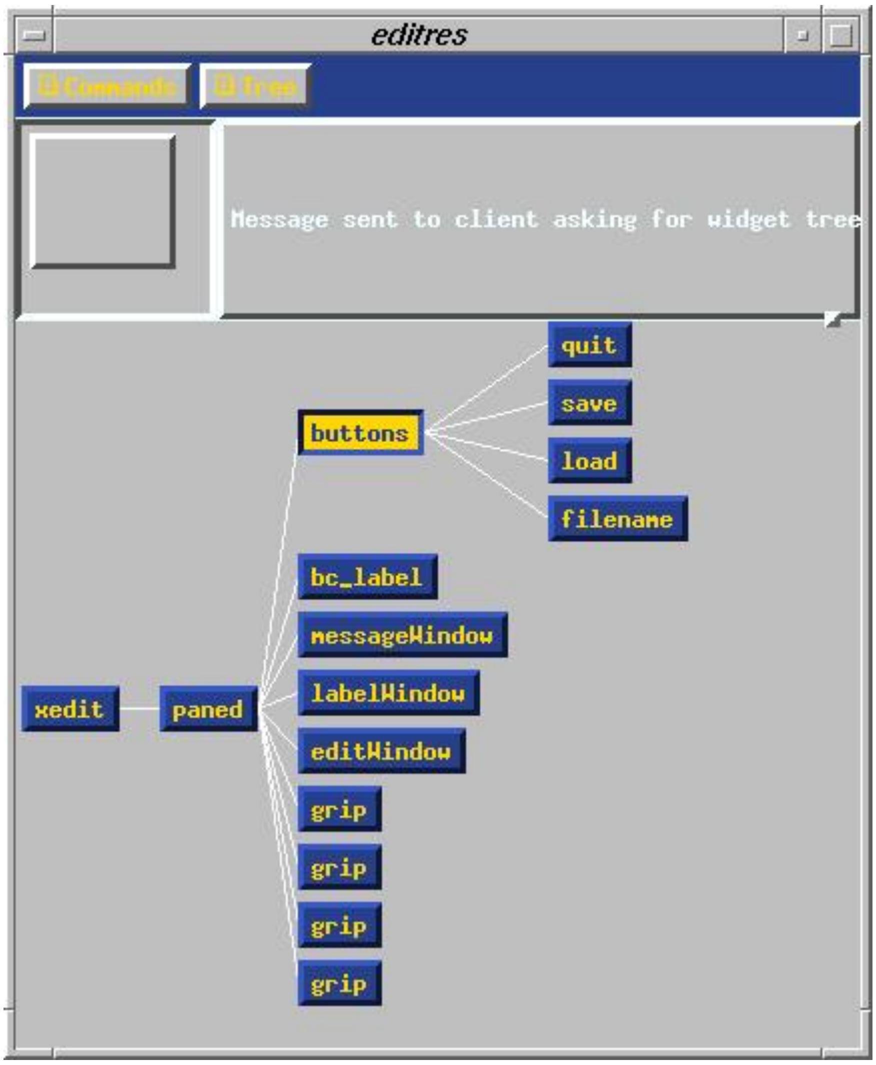
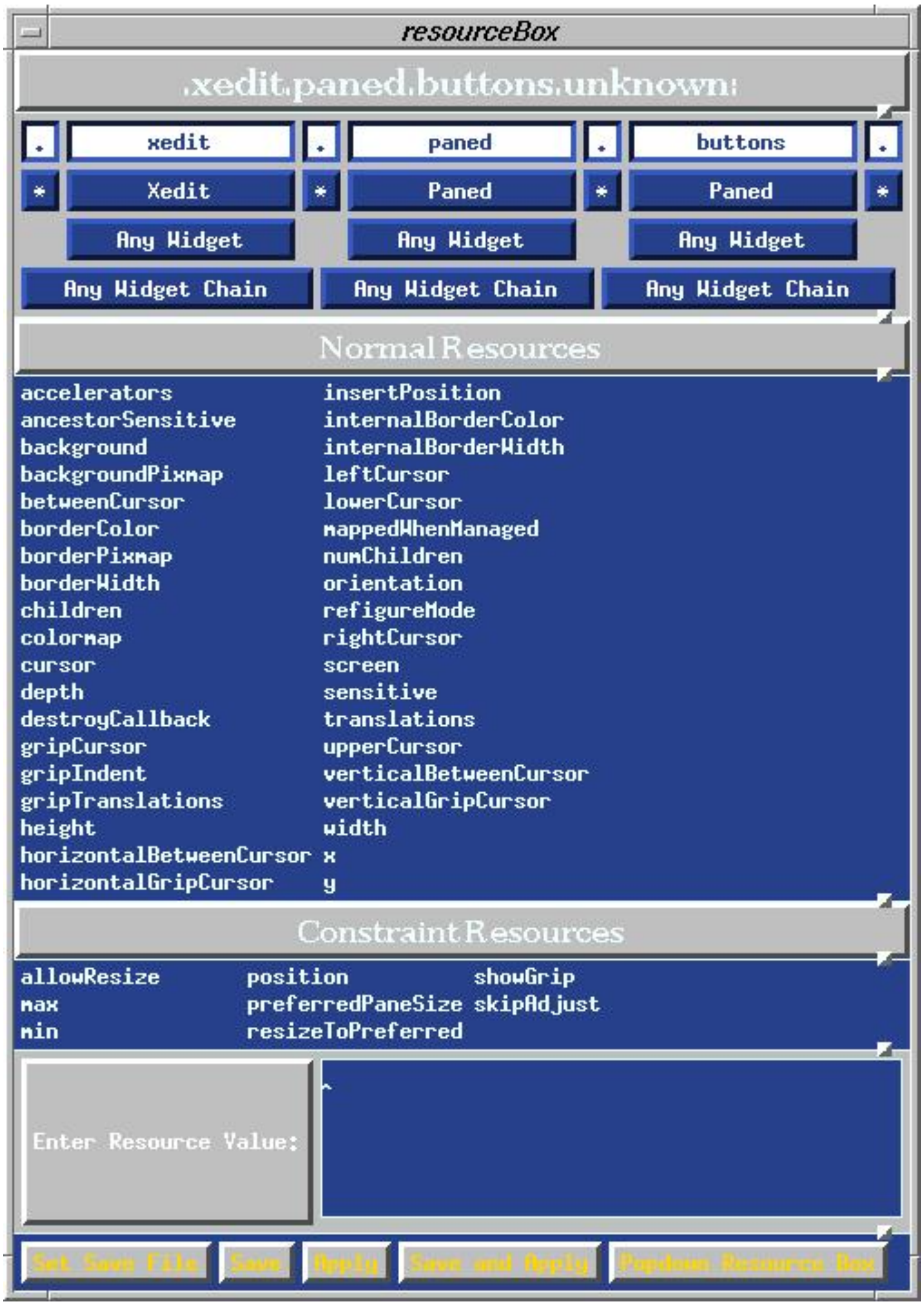


Figure 1. editres, an interactive X resource editor

All widgets are laid out in a tree-like fashion, with the parent widgets on the left, and their children progressing to the right. Lines connect children to parents. You can use the box in the upper-left corner of editres to move the display around if the widget tree is larger than the window itself.

Using the “Tree” menu, you can switch between class and instance names and select certain widgets in particular. If you want to modify the resources of one particular widget, select it by clicking on it once with the left mouse button, and then, from the “Commands” menu, choose “Show Resource Box”. A popup box with all the available resources for the widget selected will be displayed, as shown in Figure 2. In the Resource Box, select a resource with the mouse, and then enter a value for the resource in the text field below. This is perhaps the easiest way to both find the names of the resources and experiment with setting them to different values.



```
accelerators               insertPosition
ancestorSensitive          internalBorderColor
background                 internalBorderWidth
backgroundPixmap           leftCursor
betweenCursor              lowerCursor
borderColor                mappedWhenManaged
borderPixmap               numChildren
borderWidth                orientation
children                   refigureMode
colormap                   rightCursor
cursor                     screen
depth                      sensitive
destroyCallback            translations
gripCursor                 upperCursor
gripIndent                 verticalBetweenCursor
gripTranslations           verticalGripCursor
height                     width
horizontalBetweenCursor    x
horizontalGripCursor       y
```

```
allowResize      position          showGrip
max              preferredPaneSize skipAdjust
min              resizeToPreferred
```

Figure 2. Ascreen from editres showing the resource display screen

You can also make the resource string more loosely or tightly bound by adjusting which fields are highlighted at the top of the popup window. When you are ready to see the results of your changes, push the “Apply” button at the bottom of the window.

More detailed help with editres can be found on the editres manual page.

Some Examples

No article on X resources would be complete without detailed examples of how they can be used. To do this, we'll take a look at one simple X client—xclock—and then at a whole widget set—the Athena widgets (which are a stock part of any X11 distribution).

Let's try a couple of things with xclock. You can either make these changes with editres, so you can view them interactively, or you can add them to your .Xresources file, which you'll need to merge with xrdb. Remember, whenever you make changes to the resource database, the clients affected need to be restarted in order for the changes to take effect. If you think the normal black and white scheme of the clock is too dull, consider the following:

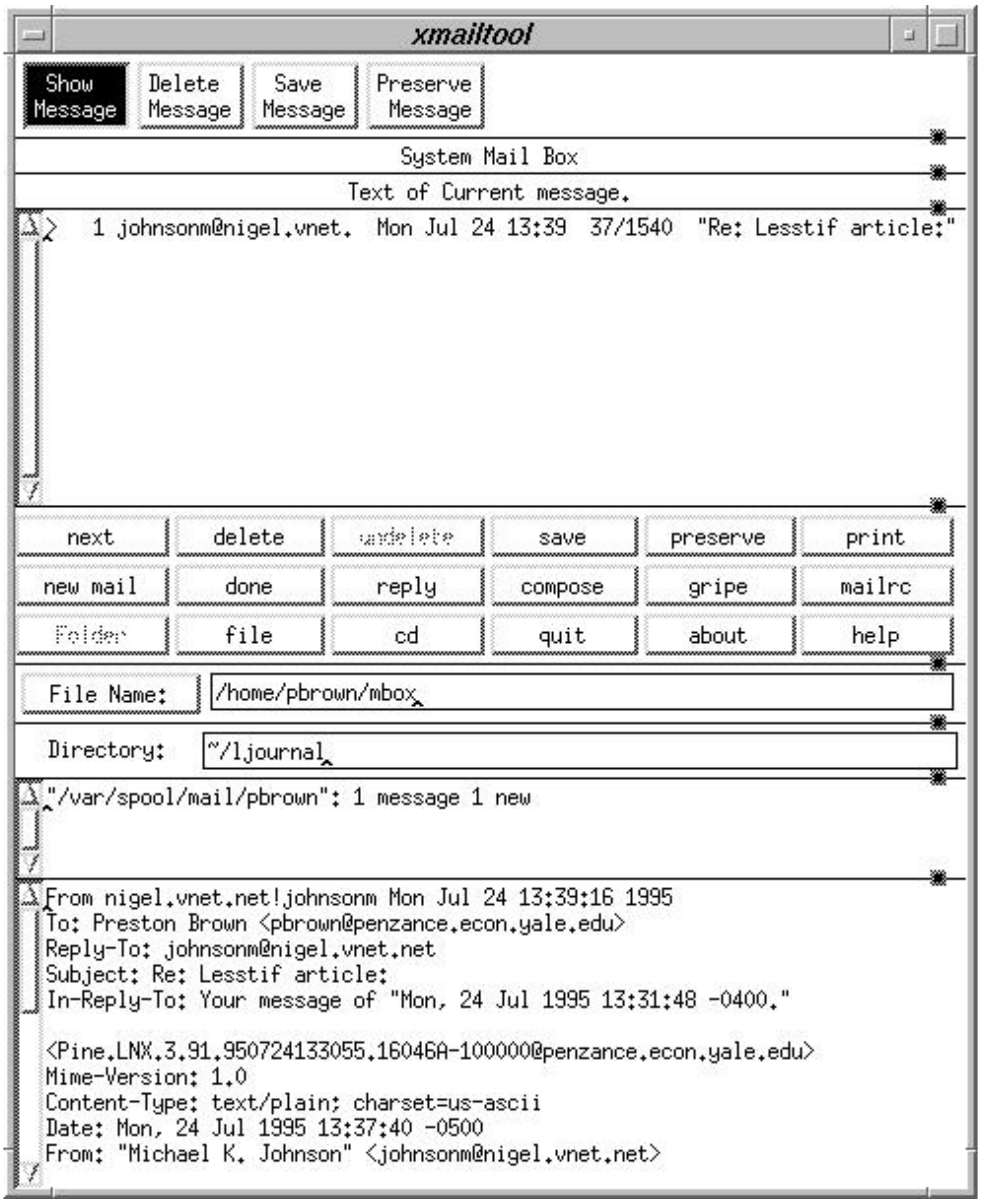
```
   *xclock.foreground: steelblue*xclock.hands:      steelblue
   *xclock.background: ivory
```

That should give you some ideas for starters. For a more extensive change in appearance, try:

```
   *xclock.analog: 0
```

This will make the clock display in a digital fashion. Specifying 1 as the value for this resource will reset it to the normal analog display.

Let's go a step further. In the July, 1995 issue of *Linux Journal*, the **Xaw3D** widgets were introduced. The purpose of these widgets is to give the default Athena widgets a more three-dimensional look and feel. However, with no default resources, programs still can look washed out and dull. This is because no default colors have been specified for the widgets, so programs that don't change these resources explicitly will display them in black and white with unattractive, dithered shadowing. See Figure 3 for an example of this.



```
> 1 johnsonm@nigel.vnet.  Mon Jul 24 13:39  37/1540  "Re: Lesstif article:"
```

```
"/var/spool/mail/pbrown": 1 message 1 new
```

```
From nigel.vnet.net!johnsonm Mon Jul 24 13:39:16 1995
To: Preston Brown <pbrown@penzance.econ.yale.edu>
Reply-To: johnsonm@nigel.vnet.net
Subject: Re: Lesstif article:
In-Reply-To: Your message of "Mon, 24 Jul 1995 13:31:48 -0400."

<Pine.LNX.3.91.950724133055.16046A-100000@penzance.econ.yale.edu>
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Date: Mon, 24 Jul 1995 13:37:40 -0500
From: "Michael K. Johnson" <johnsonm@nigel.vnet.net>
```

Figure 3. A “vanilla” xmailtool

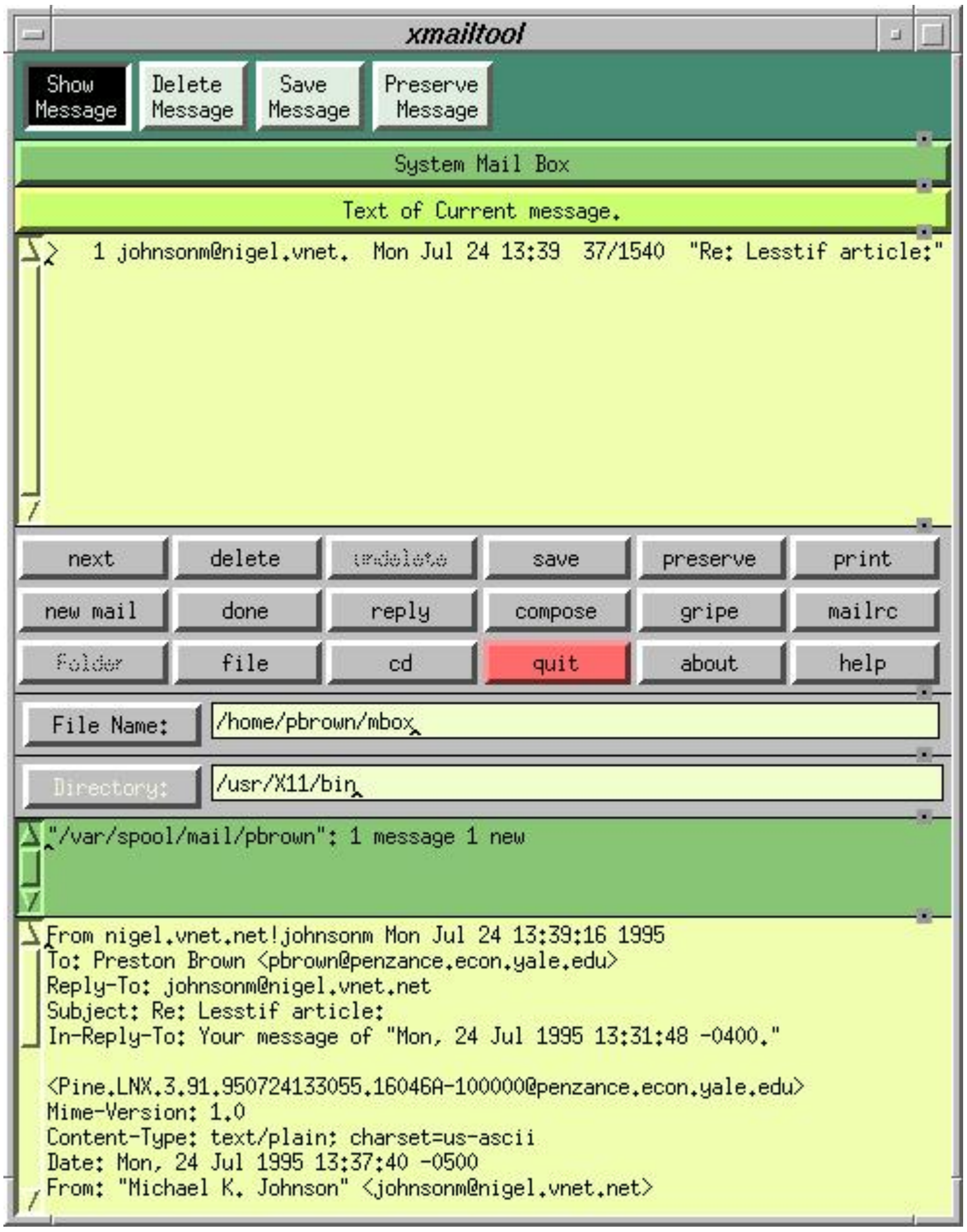
A set of resources which make the Xaw3D widgets appear more like the popular Motif widgets appear in Listing 1.

Listing 1

```
! Good Xaw3d Defaults*customization:           -color
*shadowWidth:             3
*Form.background:         gray75
*MenuButton.background:   gray75
*SimpleMenu.background:   gray70
*TransientShell*Dialog.background: gray70
*Command.background:      gray75
*Label.background:        gray75
*ScrollbarBackground:     grey39
*Scrollbar*background:    gray75
*Scrollbar*width:         15
*Scrollbar*height:        15
*Scrollbar*shadowWidth:   2
*Scrollbar*cursorName:    top_left_arrow
*Scrollbar*pushThumb:     false
*shapeStyle:              Rectangle
*beNiceToColormap:        False
```

```
  *SmeBSB*shadowWidth:      3
  *highlightThickness:      0
  *topShadowContrast:       40
  *bottomShadowContrast:    60
  ! fix up a few of the default X clients who
  ! now look silly
  *xclock*shadowWidth:      0
  *xload*shadowWidth:       0
  *xcalc*shadowWidth:       0
```

The first resource tells all programs that color is available and should be used. Then, the shadow width for all Athena widgets is set to 3 (which looks like Motif). Default colors are then selected for most of the common widgets (buttons, scrollbars, and the like), and shadow contrast levels are set. Finally, shadows on a few X clients is a bit overkill, so **shadowWidth** is reduced to 0 for these. Figure 4 shows how the program now looks more aesthetically pleasing as a result of these modifications.



```
> 1 johnsonm@nigel.vnet.  Mon Jul 24 13:39  37/1540  "Re: Lesstif article:"
```

/home/pbrown/mbox

/usr/X11/bin

"/var/spool/mail/pbrown": 1 message 1 new

```
From nigel.vnet.net!johnsonm Mon Jul 24 13:39:16 1995
To: Preston Brown <pbrown@penzance.econ.yale.edu>
Reply-To: johnsonm@nigel.vnet.net
Subject: Re: Lesstif article:
In-Reply-To: Your message of "Mon, 24 Jul 1995 13:31:48 -0400."

<Pine.LNX.3.91.950724133055.16046A-100000@penzance.econ.yale.edu>
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Date: Mon, 24 Jul 1995 13:37:40 -0500
From: "Michael K. Johnson" <johnsonm@nigel.vnet.net>
```

Figure 4. xmailtool after resource changes

This should get you started. X resources are one of the most important aspects of X programs in general, so a basic understanding of them is essential—not only for using and customizing X programs, but also writing them. You should now be able to discover the resources hiding behind your own favorite programs, and it would be good practice for you to apply the techniques in this article to a different program of your own choosing right now. Remember this while you are tinkering with X resources: X may not be as simple to configure as MS Windows, but it is much more powerful.

Enjoy!

Preston Brown is a sophomore Computer Science student at Yale University in New Haven, CT. He discovered Linux with the earliest TAMU release in late 1992. You can reach him by e-mail at preston.brown@yale.edu.

How to Build a Mac

Andreas Schiffler

David Moody

Issue #19, November 1995

Have a Linux box and need to run Mac applications? Andreas and David show you how it can be done.

If you have been browsing through Linux newsgroups, you may have heard some talk about a new Macintosh emulator called **Executor** (pronounced *ig-zek'-yu-tor*). When we (Andreas Schiffler, a long time Linux user, and David Moody, a die-hard Mac fan) first heard about this program, our interest was sparked: finally, some common ground on which to relate! We investigated and wrote this review for *Linux Journal* to give readers some idea of whether this commercial product will bridge the gap between these two completely unique operating systems, and also to give the Mac-illiterate Linux user a head-start on using the program.

Executor, a product of Abacus Research and Development, Inc. (ARDI), is “a commercial emulator that allows non-Macintosh hardware to run some applications originally written on a Macintosh”. Recently, a version for Linux has become available. This review is based on a pre-beta version, 1.99o, which was current at the time of evaluation. It should be noted that this pre-beta version has many known bugs which ARDI is addressing for its official 2.0 release of Executor.

Executor is available as a demo for evaluation purposes. The demo is a fully functional version which is limited to 10 minutes of use, a limitation which can be removed by obtaining a serial number and registration code, which you are granted when you purchase it.

The demo can be obtained from various Linux FTP sites, including ftp://sunsite.unc.edu/pub/Linux/system/Emulators/, or from the official (but slower) ARDI FTP site at ftp.ardi.com. For installation, we obtained the 3 MB archive, unarchived it with **tar xzf executor199o.tar.gz** changed directories with **cd executorlinux199o** and ran **make** as root. This copied the executable into /usr/local/bin/executor and created a Macintosh “volume” in /usr/local/lib/executor/. A quick look into its sub-directories reveals that Executor uses the native file system with additional emulator-specific files (prefixed by a %); the file system is seamlessly integrated into your Linux system. The current version is only supported under X-windows, but an SVGALIB version for those without X is in the works. Typing **executor** in an X-terminal brings your new Mac to life. (See Figure 1.)

Figure 1. Startup Screen

The first thing David (the Mac expert) noticed is the slightly non-standard setup. Instead of the “Finder” desktop (which Macs use to find and run programs on the hard drive), Executor runs its own “Browser” program. The main difference is the presence of a “hot band”, located just below the menu bar. This band contains a help button, which gives help on using the hot band, and 6 application grouping buttons for making quick reference icons. Basically, the Browser is a slightly watered-down version of the Finder; we found it fairly easy to use, but lacking a bit in on-line documentation and color.

The Mac system is not completely emulated, so there are some features which are handled differently in, or are absent from, Executor. For starters, the keyboard is slightly different. Macs have two keys which are absent from the PC keyboard, the option key and the

⌘
key, and the Mac has no Alt keys. Executor uses the left Alt key as the

⌘
key and the right Alt key as the option key. Since the

⌘
key is often used as an accelerator key, the documentation could have been a bit more specific about this. In future releases, a README.FIRST summarizing these topics, for people who don't want to read the fairly extensive FAQ before running executor, might be a useful addition.

Floppy disks are also handled differently from a real Mac, and some insight in Executor floppy disk handling is required to avoid hangups. On a Mac, inserting a disk immediately causes an icon to appear on the desktop; in Executor, you have to press

⌘
-shift-2 once the disk is in the drive and choose Eject before a disk is taken out.

This is how disk swapping encountered during the installation of program is handled as well. This is very important to note, as clicking the “OK” button during a disk swap before pressing

⌘
-shift-2 will result in a segmentation fault, and it's “game over” for Executor. Note, however, that this is explicitly an experimental release of Executor, and faults that we encountered may well be corrected in version 2.0, which will probably be released by the time you read this. As mentioned in the FAQ, Executor cannot read 800K Mac disks, because PC floppy drives are physically incapable of doing so. The current version hangs for a long time when a 800K disk is inserted and the disk-check option is selected. This is annoying but should be no problem for most people—if you are aware of it.

Executor has only a little System 7.x support, which is the current incarnation of the Mac operating system. This will only affect programs which use the special features of System 7.x. Also, Executor won't load CDEV's (Macintosh control panels) or INIT's (modular extensions to the system)---that means no QuickTime, After Dark, etc.

Now we decided to get down to work and install WordPerfect 3.0. Programs like WordPerfect have a particular use for Executor/Linux users who want to use commercial productivity applications that are not yet available under Linux or who simply want to make use of their old documents without having to switch operating systems. The WordPerfect installation requires knowing how to handle floppies under Executor, as mentioned above, but completes speedily and without problems. A new folder is created, and a double click starts WordPerfect.

Figure 2. Wordperfect Screen

Now that was easy! In Figure 2, you can see the demo document with inline graphics and tables. According to the Readme file, Executor can send output to a PostScript-compatible filter, so we printed our document. As promised, out came our formatted document. (Our printer prints PostScript by default). So far, so good. Now the bad news. Although the basic editing and formatting appeared to work well, we were unable to get the graphic editor or the equation editor to work.

PageMaker 4.2 installation was straightforward, and we had our program up and running in a few minutes. Everything appeared to be where it should be, and all documents were displayed correctly. The functions we tried worked well, although PageMaker is mentioned in the Executor 1.99o bug list as having trouble with some file functions.

Enough real work—what did we buy a computer for anyway? On to the games.

We started with the included shareware games. Lemmings was our first choice. It worked, but it seemed quite sluggish. This is because of the way Executor handles graphics. There are two methods to choose from for color mapping: either the standard color map allocation under X-windows (dynamic) or a private color map controlled by Executor. The first option preserves the colors of the X-windows desktop, but is slow. The second option, invoked with the command line **executor -privatecmap**, creates some bizarre colors for the rest of X-windows, but is very fast. When we re-ran Executor with the **-privatecmap** option, Lemmings blazed.

We ran the other two included games, Solitaire and Risk, without incident. On to the private stock. The FAQ sheet claimed that Wolfenstein 3D worked under Executor, so we loaded it up. Technically, it did work. However, even with the fast color option, it was unbearably slow. Perhaps we were pushing the limit a bit—or did we have to fiddle with more preferences? We tried installing Populous, but no amount of prodding would get it working.

Figure 3. Prince of Persia

We had more luck with Prince of Persia (Figure 3). It ran right off the floppy disk—and was it ever **fast**! Our jaws dropped as we watched the emulated game outspeed the version running on our Mac. Needless to say, we were very impressed. At this point, however, it would be worth discussing the kind of machines we were using and doing some real benchmark comparisons.

ARDI includes Speedometer (Ver. 3.23) with its emulator. Speedometer performs a variety of CPU, graphics and other benchmark tests, thus providing a convenient means of comparing machines, so we used this program to perform our tests. Our main test systems were a 25 Mz, 68030LC-based LCIII Macintosh and an AMD 486 DX4-100. ARDI claims that Executor runs at 1/3 the speed of a similar-speed Macintosh, so these systems should be roughly comparable in speed.

For our final software test, we decided to stack our emulators. With some trepidation, we booted up Executor, and loaded a program called SoftAT which is—you guessed it—a DOS emulator for Macintosh. Did it work? You bet. We even managed to mount an E: drive under SoftAT to the directory /dos, which was a DOS partition mounted under Linux, provided by Executor. Got your head around that? The short version is, it was really cool. Take a look at MSD.EXE running under SoftAT (see Figure 4).

Figure 4. MSD

According to the FAQ and WWW site, the following programs have been tested and work properly under Executor: Entertainment: Beam Wars, Glypha III, Shatterball, Civilization, Spaceward Ho! Productivity: Claris Works 2.0.1, Microsoft Word, Microsoft Excel Utility: DropStuff Enhancer, Stuffit Expander, NIH Image, AddressBook We are sure there are numerous other programs that will run.

So what's the bottom line? If there is even a remote chance that you will ever want to run a Mac program on your Linux box, you should definitely get yourself a copy of the Executor demo. That way, you can try out any programs you have to see if they work. At the very least, you'll impress anyone looking over your shoulder!

Although we were very impressed with the experimental version 1.99x, we recommend saving your work regularly, because of the slightly unstable nature of these pre-beta releases. ARDI promises that Version 2.0, due out in October, 1995, will fix many bugs, making Executor into a very attractive product. Missing from 2.0 will be AppleTalk, sound, serial port access, and support for INITs and CDEVs.

After version 2.0 has been released, ARDI will begin working on System 7.5 support, sound and serial port access, and better documentation—much needed additions. Current licensing fees are $49 for educational use, $99 for commercial use, both of which include updates up to version 2.0---which we find very reasonable.

Get the Executor demo, but be sure to read the accompanying documentation in the form of a FAQ. Take a look at the unofficial WWW pages at vorlon.mit.edu/arditop.html for more information and tips, and look around the Internet at Macintosh shareware sites, including ftp://wuarchive.wustl.edu/systems/mac/info-mac/.

Happy Mac-ing!

Disclaimer: We are in no way related to ARDI.

> Andreas Schiffler is a graduate student at the University of Saskatchewan and has been a full-time Linux'er since kernel version 1.0.9. He is the co-founder of the Saskatoon Linux Group, is working on a version of the DOS classic “Scorched Earth” for Linux, and has recently developed a taste for Macintosh software. He can be reached at andreas@karlsberg.usask.ca

> David Moody is also a graduate student at the University of Saskatchewan, but spends much of his time developing software through his company, Palindrome computing. David enjoys music and rabbit breeding.

Contact and Pricing Infomation

Andreas Schiffler (andreas@karlsberg.usask.ca) is a graduate student at the University of Saskatchewan and has been a full-time Linux'er since kernel version 1.0.9. He is the co-founder of the Saskatoon Linux Group, is working on a version of the DOS classic “Scorched Earth” for Linux, and has recently developed a taste for Macintosh software.

David Moody is also a graduate student at the University of Saskatchewan, but spends much of his time developing software through his company, Palindrome computing. David enjoys music and rabbit breeding.

Archive Index Issue Table of Contents

Advanced search

The Best Without X

Alessandro Rubini

Issue #19, November 1995

Small computers, especially those with little memory, don't run the X Window System—or any other graphic environment—very smoothly. An intelligent keyboard configuration and use of the gpm mouse server will help you exploit your small Linux box to its fullest.

If your system doesn't run X-Windows, you may miss the mouse support that makes interactive programs so easy to use. **gpm**, the general purpose mouse server, is designed with you in mind. Instead of having a multitude of mouse drivers, several from each mouse vendor, some that work well, others that don't, you can run gpm, which can talk to all mice, and works quite well. This article explains how to set up gpm to work with your mouse and programs, and also explains how to set up your text console to work the best for you.

The gpm program is derived from the older **selection** program, which was solely for cut-and-paste on the Linux console. gpm acts like **selection** until a client requests mouse events. Because gpm manages each console as an independent entity, you can use your multi-console text screen like a multi-window graphic environment. This article refers to **gpm-1.0**.

Configuring the Mouse Device

One major problem with Linux is hardware compatibility, and the mouse is no exception. Companies are always releasing new mice, and each of them provides a different mouse driver for DOS. Linux users are left alone with their device and no driver. Fortunately, companies tend to converge on a few “standard” protocols, which are supported by both XFree86 and gpm. Moreover, the gpm package includes **gpm-test**, which can help in detecting your own mouse port and protocol, and which suggests which command-line options you should use to invoke the daemon.

You must provide the protocol name and options to gpm on the command line, together with your own preferences. These will affect all mouse response until the server dies. One preference allows button reordering: left-handed people can reorder the buttons by using the command line option **-B 321**, and owners of two-button devices can use **-B 132** to use the right button as if it were the middle one, a useful way to paste the cut-buffer in Emacs without modifying Emacs itself. The current version of the gpm server duplicates the functionality of both **mconv2** and **MultiMouse**, and can act as a “repeater”. You can merge the events from two different devices and pass them along to the X server. This is useful if you use a laptop with both an internal pointer and an external mouse. If you'd like to use one mouse in each hand but keep the internal trackball active, however, gpm can't help you—no more than two mouse devices can be read at a time.

The “repeater” option is automatically enabled if you read two mice, but can be triggered independently; if you use gpm as a repeater, the X server can be configured to read **/dev/gpmdata**, a fifo named pipe, where gpm puts mouse packets received while the console is in graphic mode. This option is meant to be used by owners of busmice who want to multiplex text-only and X operation without killing and restarting the daemon. Owners of new dual-mode mice, which run the three-button protocol only if the middle button is kept down at mouse initialization, will enjoy it as well, because the device is initialized only at boot time.

How Does gpm Work?

The core of the gpm daemon is currently built around the **select()**system call and the process runs in the user space of the systems memory. The main loop of the daemon listens to a Unix-domain socket and to the mouse, and uses them in conjunction to multiplex event retrieval and management of new clients. The main loop of gpm can be (and has been) used to build a concurrent daemon for network services by modifying just a few details.

The choice of a user-space server for the mouse was originally meant to help owners of low-end boxes—the process could be swapped out when not in use and thus save a little precious memory. Unfortunately, when you use Emacs, a perceptible delay in delivery of mouse events can severely degrade performance, and combined use of mouse and keyboard is completely unreasonable on a slightly loaded machine.

The swap-in delay can be removed by locking the process in memory, but in the case of Emacs two processes should be locked in memory. The goal for **gpm-2.0**, which will supersede the current version, is to provide the choice between a user process and a kernel module. The advantage of running a kernel module is mainly fewer context-switches (and no swap-in delay whatsoever), while the main disadvantage is the waste of memory. The module alternative will offer the same interface to client applications, but will use a device node instead of a socket.

Configuring the Keyboard

The Linux keyboard is fully customizable (could you doubt it?) and can be tailored for smart text-only usage. The idea is to reduce context-switch time to get more performance out of your multitasking brain. This is the basic idea behind virtual consoles.

Here are some suggestions for improving your keyboard. I will describe some of the useful changes to make, and then give the appropriate lines for the **loadkeys** program to effect the change.

- **Caps_Lock**: Why have a “Caps Lock” key near the “a” key? When caps lock was useful to write silly BASIC programs they put it far from alphabetic keys; now that it is not so useful, they turned it to a trap for your little finger. Just get rid of it and turn it into a **Control** key.

  ```
  alt-CapsLock will still yield CapsLock
  keycode 58 = Control
  control keycode 58 = Control
  ```

- **Control**: The **Control** key on the bottom-left corner is a duplicate of the one we put by the “a”. You can turn it into **Last_Console** and thus have a fast editor/compiler context switch. Moreover, this makes your wrist useful in typing. **Last_Console** switches to the previous console you visited, and is one of the several exotic capabilities of the Linux keyboard.

  ```
  keycode 29 = Last_Console
  ```

- The Numeric Keypad Unless you're used to desktop calculators, the numeric keypad is too far in the right to be useful in typing digits, and can be turned to a console-switch scratchpad: hitting Alt-F8 takes a whole hand, and Alt-F1 isn't easy, either, at least on keyboards with the function keys at the top. Similarly, X-Windows users can configure the keypad as a map to their virtual desktop, provided that the **Num_Lock** key is left alone: **xmodmap** can't differentiate a non-**Num_Lock**ed keypad from the arrow keys. The “0” key then is suitable to be another **Last_Console**, useful if you didn't get rid of **Caps_Lock**.

  ```
  keycode  29 = Last_Console # KP_0
  keycode  79 = Console_1    # KP_1
  keycode  80 = Console_2    # KP_2
  keycode  81 = Console_3    # KP_3
  keycode  75 = Console_4    # KP_4
  keycode  76 = Console_5    # KP_5
  keycode  77 = Console_6    # KP_6
  keycode  71 = Console_7    # KP_7
  keycode  72 = Console_8    # KP_8
  keycode  73 = Console_9    # KP_9
  keycode  98 = Console_10   # KP_Divide
  keycode  55 = Console_11   # KP_Multiply
  keycode  96 = Console_12   # KP_Enter
  ```

```
keycode  78 = Console_13   # KP_Add
keycode  74 = Console_14   # KP_Subtract
```

- Home and End It can be useful to configure **Home** as **Control -a** and **End** as **Control-e**. This works with **bash**, **tcsh** and **Emacs**, without any other fiddling.

```
keycode 102 = Control_a
keycode 107 = Control_e
```

- **Escape**: If you are used to Sparc2s, old PCs, or the old faithful Apple II you'll enjoy putting the **Esc** key near the "1". This change forces you to reposition the backtick/tilde pair as well. The exact change made here may not work on your keyboard, you'd better check your keycodes with **showkeys**.

```
keycode  41 = Escape  # Escape
alt     keycode   1 = Meta_Escape
# recycle grave/asciitilde near the Enter key
keycode  43 = grave   # asciitilde
control keycode  41 = nul
alt     keycode  41 = Meta_grave
```

- Backslash: The backslash/bar pair should be near "Z", where the default keyboard configuration puts a duplicate of less/greater.

```
keycode  86 = backslash        bar
control keycode  43 = Control_backslash
alt     keycode  43 = Meta_backslash
```

The modifications listed above work with my keyboard. Check your actual keycodes using **showkeys** before applying these changes. **showkeys** is part of the **kbd** package. If you're a real typist, you can make something really useful out of the twelve function keys. Read the **keytables**(5) man page to probe further. For more information on how to modify the keyboard, see Kernel Korner in *Linux Journal* #14.

Spawning New Consoles

"Why should I use the numeric keypad to switch between 15 consoles when I only have 6?" I hear you say. Linux can handle a s many as 63 virtual console, and 6 (or whatever else) is only the number of "login prompts" configured in your system. Actually, consoles are dynamically created and destroyed during your system's lifetime.

The different **login:** prompts are spawned by the **init** process, which knows what to do by reading the file /etc/inittab; this very file specifies where **getty** should be invoked. You can play with inittab even if you don't completely understand it: to open more (or fewer) than 6 consoles for login, you can simply duplicate (or remove) lines. You must be careful, however, about the first field in the line—it is a unique "key" for the line, and it must be exactly two letters long.

My choice for console login sessions above 9 is **cA**, **cB** and so on, with the first nine entries **c1** through **c9**.

A more interesting, and memory-saving, approach to your Linux session is to spawn only one or two gettys using /etc/inittab, and dynamically allocate other as you need them. There are a number of ways to spawn a new console:

- gpm-root: This tool can spawn a new getty on the lowest-numbered free console in your system: just press control-mouse to wake the program, then press the mouse button again on the correct menu entry, and you will soon be presented with a newly-created **login:** prompt—it's that easy. When you log out from that console, everything is cleaned up automatically. This way of spawning consoles has the advantage that the /etc/utmp file is kept up to date, and thus the **who** command tells you the truth.
- open: The tiny **open** utility spawns a new console and executes a program in it. You can use **loadkeys** to create a hot key which invokes open. Thus a single keypress (or meta-key) can log you in. This approach doesn't update the **utmp** database and works only when the hotkey is fed to a shell prompt.
- spawn_console: The daemon spawn_console is part of the kbd package. It creates a console in response to a signal sent by the kernel in response to a **Spawn_Console** keysym. This approach works even if there isn't a shell to get your key, and doesn't update the utmp database.

What's the Difference?

The first proposed approach requires no intervention on your side—you should invoke the gpm server and the gpm-root client only at bootup, which you're already supposed to do. The gpm-root client then takes care of it all. Actually, a console is created only by opening it, so little more than **fork()** and **exec()** is required. Cleaning up is performed when the child process dies.

The other approaches are explained in the documentation for kbd-0.90, and are slightly more difficult only in that you need to change your keyboard configuration again, run an extra daemon program, or retrieve an extra package—open isn't part of the kbd package. The extra effort is small, because all of the hard work is implemented in the kernel.

Changing the Text Mode

Older versions of Linux couldn't allow console resizing, and a single video mode should be used for the console from boot to shutdown, and it usually was the bare 80x25. Linux-95 (Linux-1.2) allows console resizing. The user program

SVGATextMode, despite its cumbersome name, is a nice utility to change the appearance of your text console on-the-fly.

The tool makes use of the **ioctl(VT_RESIZE)** system call to change the way the video buffer is managed in the kernel, and modifies the internal registers in your video board in order to send the right signals to your monitor. The program must run with root permissions because both tasks are privileged. SVGATextMode isn't alone in the field of console resizing, but it currently is the most flexible choice.

Installing the program is easy—just **make && make install**. Then configure the file /etc/TextConfig—you need to tell SVGATextMode which chipset is in your video board. The **TextConfig** file is full of helpful comments.

The single tricky task is resequencing running applications to the new tty size. The configuration file provides a **ResetProg** line, where you can put a pathname of an executable file that will handle this; it will generally consist of sending **SIGWINCH** to applications, as outlined in the sample **ResetProg**.

The definitions for the specific modes are modeled on the XF86Config lines. The X-Windows configuration documentation and any previous experience with with X-Windows configuration can help in playing with text modes. If you're going to fine-tune your X-Windows screen, you can easily run your tests with SVGATextMode. Its fast cycle time makes trial-and-error better because you needn't restart the X server for each trial. Fine-tuning screen timings for text modes can lead to a good configuration to be pasted in your XF86Config file. Alternately, if you have set up X-Windows already, you can use that knowledge to set up SVGATextMode.

Other facilities offered by SVGATextMode are automatic font loading and cursor reshaping. This last feature alone is a good reason to run SVGATextMode on your laptop—no more kernel patch to have a block cursor.

Problems Related to Console Resizing

If you use SVGATextMode, especially on small machines, you'll notice that sometimes console resizing will fail, even if you have plenty of swap available, and sometimes even with plenty of RAM. The problem is related to the *kind* of memory needed; the kernel needs to complete the system call (an **ioctl()**) atomically, and it needs to get a contiguous chunk of memory for each active console. There's no time to swap out some process or to shrink the buffer cache, and the kernel keeps only 1/64 of the available RAM for these “urgent” issues. As a result, the smaller the box, the more consoles you use, the more you're prone to fail resizing. Resizing to a smaller estate won't always help, because the kernel must be sure to have place for *all* the active consoles before it starts copying video data to the new area, and only at the end can the old buffer be released. If it fails, simply try again; it will probably succeed the second time.

Another issue is the role of the **ResetProg**. Why do some applications do resize well (like **jed**), others become completely stuck (like selection) and still others need to be sent the **SIGWINCH** signal? Because a resizing of the surrounding window is an asynchronous event, which doesn't fit the normal environment of the application.

Applications belong to three types: over-attentive ones look at the window size often, and perceive the new situation right when it happens; more conventional applications wait for an asynchronous notification of the event (a signal, namely **SIGWINCH**, for **WIN**dow **CH**ange), and respond to the notification in the right way; and some applications simply don't respond to changes in window size, and ignore **SIGWINCH**---the current version of selection was written before console resizing was available. Thus, while a resizing xterm sends **SIGWINCH** by itself, a resizing console doesn't send anything, and an external **ResetProg** is needed to fill the gap.

Tools for the Text Console

The following tools work particularly well on the text console, sometimes even better than in graphics mode.

- gpm-root: **gpm-root** is a root-window manager. Its role is to draw menus on the screen background, like you do in the X environment. By default it responds to control-mouse events, since mouse-only is left to the selection mechanism, a vital feature if you work on a text console. The menus drawn by gpm-root are read from a user-local configuration file, and can be tailored to your own preferences. gpm-root allows console-switching, console locking, opening a new console to create a new system login, retrieving system information and executing external commands, as well as recursive menus. The user configuration file is reparsed when needed, to ease trial-and-error menu writing.
- Emacs The Emacs editor is made mouse-sensitive by loading the t-mouse.el package, which comes in the gpm distribution. All the functionality available under the X Window System is duplicated on the text console, including the scrollbar. The scrollbar acts on the last column of the screen and smooth scroll is accomplished through a variable resolution widget—the more you move your mouse to the left, the less scrolling takes place in response to vertical motions. A meta- mouse button press triggers the scrollbar independently of the position of the mouse.

- Jed: The Jed editor is mouse-sensitive as well. Mouse support has been developed by Jed's author, and thus is perfectly integrated. Jed is a good alternative to Emacs if you own a small computer—it is considerably smaller, both in disk usage and memory occupation, but offers the same basic commands and interface, as well as its own extension language. Well, if you learned “elisp” to configure Emacs, won't you learn “slang” to configure Jed?
- dialog: The dialog program is nothing special, except that it runs definitely better on the text console than under an xterm. Managing a Slackware installation with the dialog menus on the console is a breeze, especially if you can interact with your mouse. Under xterm, on the contrary, a dialog menu looks ugly, and available mouse events are limited to button press, so you're almost forced to use the keyboard. Moreover, the curses libraries tend to use the alternate screen provided by xterm, and thus message boxes are simply invisible, and you wonder why the terminal is idling around without any message on it.
- mc: **mc** (*The Midnite Commander*) a powerful file manager, is modeled on the famous DOS command **nc**, though much more powerful than the original. **mc** is fully configurable and extendable, and does a good job of managing your file system status. You can use its menus with the mouse as well as the keyboard, while shift-mouse runs selection as usual.
- screen: The **screen** utility is a viable alternative to opening a lot of consoles. It manages up to ten terminal sessions running on a single physical connection. **screen** offers a lot of functionality, and is a must if you use a vt100 or an old PC running kermit to connect to your linux box. It is useful also if you're really console-hungry and you don't have enough consoles. The major drawback of screen is that it emulates a vt100, so you lose all the extra features offered by the Linux console. Specifically, you can't run gpm-aware programs under screen. One really nice feature of screen is the visual-bell facility. It offers a cut-and-paste facility, too, but mouse-based selection is easier to use.
- minicom: **minicom** is an easy-to-use communication package resembling DOS's **telix** with a menu-oriented setup. It offers a good scripting utility, which makes your programs talk directly with the remote end of your serial connections. I use minicom to remotely control a Nicolet oscilloscope, with no concern about communication parameters.
- gnuplot: Its name says a lot about it. A drawing program that can read external ASCII files, its major advantage is the ability to manage a many different output devices—including a bare terminal. This means you can look at your data graphs without starting X-Windows. The granularity of a tty plot is coarse, but gnuplot does its job well. It has a fairly complete internal help facility, and you can produce nice PostScript (or other graphic format) graphs without entering your graphic environment.

Further Readings

All the tools described above come with manual pages or info files. **mc** has a good internal help utility. **gpm-root** and the lisp library t-mouse.el are part of the gpm package.

Text-Only Resources

Alessandro Rubini is taking his PhD course in computer science and is breeding two small Linux boxes at home. Wild by his very nature, he loves trekking, canoeing and riding his bike. He wrote **gpm**, and can be reached as rubini@ipvvis.unipv.it.

Archive Index Issue Table of Contents

Advanced search

Linux on Low-End Hardware

Trenton B. Tuggle

Issue #19, November 1995

Trappen is a low-powered Linux box on the Internet. It serves uucp e-mail and news feeds to two home Linux machines. It also provides anonymous ftp and telnet terminal capabilities. This is its story.

I work in a research lab with many Unix boxes. My problem is that these boxes are dedicated to research almost all the time. Our system administrator doesn't really have time to support people like me, who want more than the traditional dial-in e-mail reading. Sure, *term* is quite useful, but I'm a traditionalist.

You see, I run Linux at home. I have the full power of a Unix machine in my house, and I am not going to be satisfied by using it as a terminal to dial into a server with it. I want my own e-mail!

In the old days of Unix, before Internet, machines transferred e-mail, news, and files flawlessly [Well, pretty well, anyway—Ed] by calling each other up over phone lines. A machine could have dozens of modems and would accept calls and periodically call up other machines using the system called uucp—Unix-to-Unix copy. Instead of getting files through anonymous ftp, people used “public uucp”. Instead of writing some nightmarish scripts to call in, transfer files, hang up, and process them, I thought—why not use uucp?

So I began looking for a machine to use. About that time, a friend of mine was hired, and we found an old computer in his office. It was an old 16 MHz 386SX. In its previous life, it ran Procomm and was a full-time *terminal*. What a change we had in mind—we transformed it into a multi-tasking Unix *server*! It had 4MB of RAM, which was enough to run Linux. But the hard disk wasn't much to gawk at—40MB. We named it “trappen”.

The purpose of trappen was two-fold. First, we wanted uucp connectivity to our home Linux boxes. Second, we wanted to provide anonymous ftp and http for ourselves; not that we badly needed it (though it would be handy) but more because it's fun to have your own personal Unix server.

We decided to split the disk in half—20MB for the system and swap space, and 20MB for file storage. Now, it needed about 5MB of swap space, so how was I supposed to squeeze a Linux system into 15MB and still have plenty of room for news, e-mail, and ftp-able files?

I've installed many Linux systems over the years, but I always simple used Ethernet to copy my setup onto the target computer. (I have almost created my own distribution.) But my system would take me days to prune down to 15MB. I had to resort to using someone else's distribution, instead.

In the beginning, Linux distributions were not really very fancy. They gave you the files, and you were on your own. If you weren't a Unix expert, you could get stuck in a swamp of daemons, config files, and strange programs. But I knew installations were getting better, so I decided it was worth a try.

I grabbed the latest Slackware distribution and looked at the disk contents; I could do it. I didn't need any programming libraries, and I didn't need X-Windows or anything fancy. Just the basics. About 20 minutes later, I had the tiniest Unix system I've ever seen! I went ahead and installed everything I needed, and it took 14MB! I had all the binaries I needed: news-readers, mail-readers, and uucp. The next step was to configure everything.

Now I expected to have a lot of things to configure. We had **smail**, anonymous ftp, and the rest, but the Slackware system did such a good job of setting up default configuration files that most things worked out of the box. What I *did* spend time on was smail and uucp. I set up uucp to route things to our two Linux boxes at home, and configured smail to use it. Then we went through and stripped out any functionality we didn't need, to make the system even smaller.

To support us, our network administrator here at the lab agreed to put in DNS MX records for our home computers, pointing to trappen. This would allow e-mail addressed to us at our home machines to be sent to trappen, which would process and spool it and forward it to us when we were connected.

Through trappen, we now had e-mail at home—not just for us, but for anyone who had accounts on our machines. My whole family has access, and they're learning to use e-mail and news. (It takes quite a bit of disk space to store news at home, but I only have a few groups, and I don't keep articles for very long.)

Trappen has plenty of power for all its tasks, though it is pretty slow interactively. But we have its console, as well as a Wyse terminal in its office. We decided to make the ultimate use of the console and terminal—as telnet terminals to other, more powerful machines. We decided to write a program which would ask for a host to telnet to, which would allow trappen to function as a terminal server. It turned out to be easy, thanks to the getty_ps package. **Getty_ps** is so versatile that we were able to set up configuration files such that instead of spawning login, it would spawn telnet. We changed the login prompt accordingly, and voila! Trappen's VGA console is the fastest terminal we have in the building, especially because it runs over Ethernet. The serial terminal attached to it is quite handy to check e-mail with.

Everything worked perfectly! We didn't have to touch trappen for a week. In fact, when we had time we decided to add a few enhancements. We added a uucp-ftp gateway. Someone can anonymously ftp and place files in a directory named to *computer* and trappen will automatically copy the files to a directory on the machine with that name.

Then we really went all-out. We didn't want to put an entire X-Windows environment on trappen (it was quite under powered), but it did have a VGA card. We got the X-Windows server program and all the X-Windows fonts and copied them over. Then we ran X-Windows with the -query option, which causes it to be an X-terminal to another machine, in this case to one of our high-powered machines in the lab. It worked! Trappen didn't have nearly enough power to run X-Windows applications locally but it wasn't bad as a graphical terminal. We eventually decided it was too slow, but it did work, and did not require much disk space.

Trappen is still going strong, over a year after it was “born”. It now allows slip access for our home machines. The CPU is severely limited in power, and during interactive use, it is quite slow, but transferring files and telnet sessions through trappen to other machines is almost instantaneous. Trappen now supports several users who use it to access the Internet.

UUCP is almost completely error-proof, so when I want to transfer a file between home and work, I uucp it. Trappen doesn't call out right away, but the machines connect every few hours. It doesn't matter if it can't connect right away; uucp always gets through eventually, and I know that when I get home, the file will be waiting in my incoming directory.

Trent Tuggle is an Engineering student at the University of Central Florida. His other job is programming virtual reality simulations at the Institute for Simulation and Training. Between the two of them he doesn't have any time, so consequently he's into water sports, model airplanes, and music synthesizers. His e-mail address is tuggle@vsl.ist.ucf.edu.

Linux In The Real World: Linux Serving IKEA

Anders Östling

Issue #19, November 1995

IKEA has discovered that Linux is a low-cost solution for its TCP/IP networking needs. Anders Östling tells us the story.

It started by coincidence early last year. We had repeated problems with a system that sent files to a business partner using a leased line and VMS-based Kermit. A consultant suggested that we installed a PC with UUCP as replacement. We agreed, the PC arrived, and guess what operating system I found on the disk. Right, it was Linux 1.0, our—and my—first experience with a PC-based Unix system.

Our Organization

For those that don't know IKEA, here is a overview of my company. IKEA is a retailer of furniture and home interior stuff. We have sites in nearly all European countries, the US and Canada, and the Far East, including Australia. These countries are sub-divided into organizational units. Our unit, Northern Europe (at the moment of this writing), is responsible for managing the stores, warehouses and offices in 7 European countries. Our headquarters is located in Helsingborg, in southern Sweden, and this is where I work.

From here, there are network connection to all “our” countries, as well as to the other IKEA branches. We also have a couple of DEC Alpha systems as central “mainframes” for common database applications. These systems are critical for our survival (and so are the network connections to these systems).

New Times, New Network Demands

Like many other companies, we are rapidly evolving our IT structure to more distributed systems. This means that communication lines that could be cut for a day or more before without any serious problems, are now becoming absolutely essential for business.

In late 1994 and early 1995, we at IKEA Northern Europe restructured our European network to have faster links, new routers, LANs, etc. From being a more or less pure DECNET network, we converted more and more to TCP/IP, since this is the common denominator with our sister organizations worldwide. While we planned for this big task, we also found that we had to change our IP addressing and naming in order to avoid havoc. Our decision was to have a tree-structured DNS [footnote:Domain Name Service] system with the root at our head office in Helsingborg, Sweden, and secondary DNS servers at each site. The DNS servers would serve approximately 75 VMS systems, 20 AIX systems, and hundreds of other PCs and and NT-servers.

Why Linux?

One option was to use an existing VMS or AIX system as primary server, but since we demand 100% uptime, this was not very attractive. Not that VMS or AIX systems are unreliable, but they are used for other tasks, which means that they could be down for many unrelated reasons. We saw that we needed a dedicated system to avoid problems.

This was one year after our first Linux system; since I had been running Linux both at home and at work for over a year and was convinced of its usefulness and stability, I suggested Linux. My boss listened to my arguments, and agreed to give it a try.

We bought two 486/33 machines with 8 MB ram each, and I started to install 1.1.91 on a rainy day in March. After two days, we had BIND 4.3.9 [footnote:The nameserver program] up and running, with a second PC as a backup system should the first PC fail. After running internally for a week, we decided to start the transformation. The whole tech department spent a weekend changing more than 300 TCP/IP systems all over Europe. On Monday morning, we had over 50 secondary DNS servers (mostly VMS systems running TCPware) getting their information from CYGNUS, the primary server PC. CYGNUS was also serving approximately 200 IP systems at our headquarters from day one.

For redundancy, all headquarter systems have both “primaries” in their resolver file. Empirical tests (pulling out the cable) have shown that this works flawlessly. Another thing is that if CYGNUS breaks completely, we can replace him with VIRGO, the second PC. In order to make this a bit easier, we **rcp** all DNS, RCS and passwd files via **cron** at regular intervals. All CYGNUS specific files (rc.inet and such) are also stored safely on VIRGO so it should be fairly easy to switch identities, if needed.

Hands-On DNS: How we did it

As I mentioned, we use GNU RCS to maintain our DNS files. We have set up one master file for all IP systems and one reverse-translation file for each country. All persons allowed to edit these files have their private accounts which gives us a good overview of who did what, and the ability to reverse their editing in case they have done something wrong.

IKEA is using the RFC-recommended address 10.x.x.x internally. Each country has its earlier DECNET area as the second number, so for example, Belgium has 10.14.x.x since they were in DECNET area 14. The third part represents the location, so Antwerp has 10.14.5.x. This leaves 253 possible hosts for each LAN or site, and enables us to use class C subnet masking for all countries. Again, the exception is our headquarters, which has more than 253 hosts. We have allocated a special "area", 62, and use a class B subnet mask at headquarters.

The master file is called named.neurope, and the reverse files are called named.*xx*, where *xx* is the ISO code for each country (be, gb, dk, se, nl, and no). We also have a reverse file for our headquarters, named.hbg, since this is the single largest "domain". As an example, Figure 1 contains an edited extract form this file.

The cache (named.ca) file has entries for our central DNS system (where the Internet connection is) and for our sister organizations' primary DNS servers.

The boot (named.boot) file has a "forwarders" record which routes all unknown lookups to our Internet-connected DNS server, as well as a record that states that we are primary server for our organization.

Our secondary servers (40 VMS hosts) have corresponding files with pointers to CYGNUS so they know where to "zone-transfer" files and updates from. These updates takes place at boot and every fourth hour.

Conclusion

Today, system size has increased to well over 700 IP hosts due to the fact that new LANs with networked NT servers and Windows PCs are popping up every day of the week. How have CYGNUS and his partner (yes, it's a he) coped with this? Until today, there has been no problem worth mentioning (aside from a total power outage which killed both systems). Oh yes: one **big** problem is to make people not used to Unix use RCS and **vi** to manage our DNS files.

Another problem with Linux is that it's too cheap. I'm serious, since many people still put an equal sign between Cheap/Free and Bad/Dangerous. In the case of Linux (and XFree86) this has proven to be pure nonsense.

There are some companies here in Sweden offering support for Linux. I think that this will help to make Linux more socially acceptable; if you find somebody who is willing to accept a check from you, then you can always shout and yell at him if there are unsolved problems. Personally, I prefer to have direct contact with the programmers and designers.

A year later, that system is still running Linux 1.0 and UUCP. There have been a few problems, all caused by the other UUCP partner (I won't mention any brands), but all-in-all, everybody was happy. So happy, in fact, that a few of our techies have also tried Linux out on their own PCs. Some, like me, have kept Linux for good.

Anders Östling is a die-hard VMS fan who, after spending 10 years in the Digital farm, has gotten more and more into Unix and networking. When not doing what he's paid for—managing computers and networks—he likes to cuddle with his kids, computers, pets and wife (in no particular order...). He lives in the countryside outside Helsingborg in an old miner's village called Gunnarstorp. Don't miss it when you are in Sweden! If you have any questions or comments (general or DNS), he can be reached at anos@ineab.ikea.se.

Linux at SCO Forum

Belinda Frazier

Issue #19, November 1995

SCO Forum, famous for its fun, casual environment, offered thought-provoking discussions by Scott Adams, Clifford Stoll and John Perry Barlow. Linux Torvalds spoke on the future of operating systems.

The ninth annual SCO Forum took place on the University of California Santa Cruz campus August 20th to 24th, 1995. SCO, Santa Cruz Operations, is the provider of SCO Unix, billed as “the leading PC-Unix”. SCO also provides other system software for businesses.

During the Forum, SCO unveiled a new corporate logo to signify its expansion into new markets, serving the “entire cross-platform world of Unix server and Microsoft Windows desktop integration.” Many SCO speakers reiterated SCO's “Windows Friendly” strategy.

Forum tutorials and conference sessions varied from software specific to SCO, Marketing on the Internet, to the Future of Operating Systems.

The highlight of the sessions included three speakers. Scott Adams, who writes the Dilbert cartoons, gave a humorous presentation, augmenting some of his cartoons with the story behind the cartoon. Some of Adams' cartoons had gotten him into trouble at his former full-time jobs in the computer industry.

Stoll worried about the “cult of computing” noting that schools have converted their music rooms or their art room into computer labs, noting that, “We are saying something about what's most important in our society. and it's computing over music and art... or even history and social interaction.”

Barlow commented about the use of the Internet, “I think we're in the business of creating what Teilhard de Chardin talked about, writing in the 30's, about the collective organism of mind, an entirely new layer of the evolutionary process—evolution that is self aware.”

As Barlow eloquently started a statement about how the Internet is "...an ecosystem for all the creatures of mind," he was interrupted by Cliff Stoll saying, "How can you compare the Internet to an ecology? the Internet is a telephone system!"

Barlow: "Clifford, it was a virtual thought."

Cliff: "The Internet is a telephone system that's gotten uppity."

The banter continued between the two about interactions on the Internet versus "meet (face-to-face interactions in the real world) space".

Barlow told the audience about a project to establish a virtual conference facility, with a room in Portland, a huge video screen, and 3-dimensional sound, so that people could see the body language of people in other places during the conference. Barlow asked the project coordinator, "Ranji, does it work?" Ranji said, "Oh, no." Barlow said. "What's missing? It looks like you have everything." Ranji said, "But the Prana, the Prana is missing." Barlow concluded the story for the SCO audience explaining "Prana is the Hindu term for breath and spirit. And I think that the real enterprise here is to find out whether Prana can ever be fit through a wire."

Over one-third of the audience raised their hands in response to Linus's question, "How many of you are actually using Linux or have used Linux before?"

Linus stated that "The most important point I want to say about the future is that I personally say that the future is the desktop—or not even maybe the desktop, but the personal computer."

Linus added, "if Unix decides to ignore the desktop market and tries to be a server, even if it's a server that tries to serve desktops, Unix is eventually going to die. And I think the future is acknowledging that the desktop market is where it's at."

"We all know who's the boss on desktop. Certainly today, Microsoft is spending probably in excess of 200 million dollars on making sure who is King of the hill. Right? Is that due to technical merit? No, no, people on the desktop have been used to really crappy operating systems. Unix people who're telling us that they have a technically superior product obviously aren't doing the right thing, because on the desktop, technology isn't what matters. What matters is availability and price. So what we like to have, and what people like to have, [is] cheap and available, and actual technical merit comes second—actually, it comes last."

He went on to mention Free Software, Software Piracy, and of course, he talked a little about Linux.

“What [Linux] has to give this community is a desktop that doesn't reboot twice a day—or more often, if you're doing something strange like writing a document.”

He took technical and other questions in person after the discussion, and the two SCO technicians sitting next to me were up there first to ask him a question. Had I not been late to catch a plane, I would have stayed a while longer and eavesdropped.

Belinda Frazier has been working with Unix for nine years and with publishing for even longer. She enjoys traveling to Unix and Internet conferences for SSC. She has recently given up attempting to engage the neighbors' cats and dogs in a psychological discussion about why they should want to stay out of her garden.

Archive Index Issue Table of Contents

Advanced search

IGEL Etherminal 3X

Michael K. Johnson

Issue #19, November 1995

In other words, it is just large enough to hold a monitor.

A low-cost Ethernet-based X terminal, the Etherminal is a box 2.25 inches high, 12.5 inches wide, and about 11 inches deep. In other words, it is just large enough to hold a monitor. It is powered by a 386 SX-40 processor, 4 (or 8) MB of RAM, and 2MB of ROM, with video provided by a standard Cirrus Logic video chip and Ethernet provided by a standard AT/LANTIC chip. Add your favorite mouse, keyboard, and monitor to the inexpensive base box and plug it into your Ethernet network (all three common interface types are included), and you are ready to work. This is standard ISA PC hardware done with an inexpensive everything-on-the-motherboard approach. Said another way: this is hardware that can run Linux.

And it does run Linux. While the Etherminal is sold as an inexpensive X terminal, it is a standalone diskless workstation configured as an X terminal. Since it is built on freely available software including Linux, XFree86, and fvwm, IGEL is able to sell it for less than if they licensed any commercial operating system. Also, since Linux is resource-frugal, they are able to take advantage of low-performance hardware to make an inexpensive X terminal with reasonable low-end performance.

This is not a speed demon. Graphics-intensive programs do not run quickly. This is not a detriment: the Etherminal is optimized for cost, not for speed. (However, note that it will become faster in the version that IGEL intends to introduce next year, which will include a 486SX CPU.) If you need a faster, more capable, and quite possibly less expensive replacement for character-based terminals, the Etherminal may well meet your requirements, for several reasons.

First, it is easy to set up, using a built-in graphical configuration utility that comes up automatically the first time the Etherminal is booted, and can be easily started at any other time. While it can be hosted by remote machines for centralized administration, it defaults to running on its own, which makes dropping it on someone's desk—as an easy and inexpensive way of providing X connectivity—a one or two-minute job. It also has sensible defaults for X-Windows setup; you will have no need to spend hours fiddling with an XF86Config file to get X to work right—just be careful not to tell it that you have a more capable monitor than you actually have, as doing so may be difficult to recover from quickly. However, the configuration program provides far more capability than this: multiple languages, keyboards, and boot methods are all available, among other things.

Second, it does not require any other X-capable machine on the network. Local terminal sessions to any TCP/IP-capable host are easy to establish.

Third, while the Etherterminal is easy to set up in a “standalone” mode, it is also just as easy to configure it to depend on XDMCP and remote font servers, to download its X server from a centralized server, to run a local or remote window manager, and to do complete centralized maintenance if you wish. Unlike some X terminals, the Etherminal gives you complete control over this—and comes set up to work out of the box. It is both “plug-and-play” and configurable, a useful combination.

Lastly, if you are competent with Linux, you can bring up a “local” shell session. You will be in a simple shell on a Linux system, able to investigate the file system, including the /proc file system, able to directly execute the executables there. This is not usually particularly useful, but IGEL hasn't tried to hide this from the curious user or administrator, to their credit.

Michael K. Johnson is the Editor of *Linux Journal* and wishes he had spare time to spend pretending to be a Linux guru. You can reach him via e-mail at info@linuxjournal.com.

Teach Yourself PERL in 21 Days

David Flood

Issue #19, November 1995

Don't let the size of the book intimidate you, though, it's due not to the complexity of the language, but to the easygoing writing style of the author.

- Author: David Till
- Publisher: Sams Publishing
- ISBN: 0-672-30586-0
- Price: 29.99 USA/39.99 CAN
- Reviewer: David Flood

The 846 pages of *Teach Yourself Perl In 21 Days* from Sams Publishing—two inches thick—are intimidating. Don't let the size of the book intimidate you, though, it's due not to the complexity of the language, but to the easygoing writing style of the author.

The book is divided into 21 days (chapters) of differing length, three review sections (at the end of each "week"), several appendixes, and a thorough index. Each "day" has a discussion of a part of the language, examples (either complete programs or code fragments), warnings, sections of "Do's and Don'ts," and questions or problems at the end. Suggested answers to each "day's" questions are supplied in an appendix.

I started reading this book with no previous exposure to PERL; by the end of the second week I was beginning to see places where the language might be used in real world applications. The examples and discussions were clear and easy to follow. After completing the book, I started writing a simple application; so far I've been able to do everything in PERL that I need to do.

Some of the functions of PERL are also directly translatable to other programs. The discussion of Pattern Matching (Day 7) should be (with author/publisher permission) incorporated into the **sed** manual page; I'd never been able to get sed to do what I wanted until I read this chapter.

However, the copy of the book I received had a few problems. It was from the first printing and had several errors that the proofreaders had missed. Some of those early in the book would trip up new users of computers. For example, in the section on ftping the PERL file to install PERL on the reader's machine, a sample ftp session is presented. In the sample dialogue, the GET command wasn't followed by a file name: **get** . When I tried this, ftp told me I needed to supply a file name. I knew enough to do a **ls p*** to find the current file and then type **get filename** to get it. (I only did the ftp session to test the directions, since I had already installed PERL from my Slackware CD-ROM.)

Another hazard lies in the discussion of the required first line of a PERL program **#!/usr/local/bin/perl**. There is a small discussion of the fact that it must be the correct path of the PERL interpreter, but the only advice given is to talk to your System Administrator. Since several folks are their own System Administrator, this might cause problems. Since there are already “Unix Only” discussions in some of the chapters, a small discussion of the **which** command would be a welcome addition.

I was able to detect at least one error for every two chapters. Upon requesting an errata sheet from the publisher via their area on CompuServe (which they advertise in the book), I was informed that they would get back to me. A week later, they still hadn't. Since files containing errata for some of their other books are available from the area, I presume that the book is still too new for a sheet to have been compiled.

The only other problem that I have with the book is its classification of “User Level: Beginning—Intermediate.” Since I have taken classes in both C and Ada, I was able to relate some of the concepts that I had learned to this language. However, as I pointed out before, some of the ideas presented are definitely not for beginning computer users.

Generally, I would recommend this book for any person who wants to learn PERL. But this is not the book for a person who is attempting to learn a first programming language.

David Flood is currently a student in the School of Drama at the University of Washington. He plans to graduate sometime in 1996 and to get a real job. He is reachable (for now) at dcflood@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

ELF is on the Way

Michael K. Johnson

Issue #19, November 1995

Debian has had ELF support for its standard distribution available to developers and all other interested parties for several months, and some of the debian developers are working on ELF issues.

Nearly all the major Linux distributions have announced some support for ELF, and some have beta versions available on the Internet. Red Hat, Slackware, and Yggdrasil have each announced that alpha or beta level ELF-based distributions are available from their standard FTP sites. By the time you read this, all three expect to be shipping production-quality ELF-based distributions.

Debian has had ELF support for its standard distribution available to developers and all other interested parties for several months, and some of the debian developers are working on ELF issues. The current release is a.out-based, but users will be able to upgrade to ELF without re-installing the distribution. This in-place upgradability has been included in Debian for a long time, and has been well tested. Debian can be retrieved via FTP from ftp.debian.com and mirrors including tsx-11.mit.edu and its mirrors worldwide.

Red Hat's beta is available via FTP from ftp.pht.com, ftp.caldera.com, and other mirrors, and is being tested as of late August and early September. Red Hat has committed to a production-quality release in September to support the second preview release of the Caldera Network Desktop, which is built on top of Red Hat's distribution.

For several months, Slackware has provided Slackware 2.3 in a.out format and a beta-quality ELF distribution as well. Slackware's ELF distribution is unusual in retaining an a.out boot disk for installation. Slackware, including the ELF beta, is always available from ftp.cdrom.com.

Yggdrasil avoided releasing a new release of their Plug and Play Linux this spring, and instead put their resources into developing an all-ELF distribution.

That beta-quality distribution was made available for FTP from ftp.yggdrasil.com in August.

Linux Journal is not officially recommending any one particular distribution. We will (finally!) be publishing a very comprehensive overview of the most common distributions early next year, which should allow readers to make their own more informed decisions, based partly on the information *LJ* provides.

In other news…

DEC has announced that they have ported the XFree86 XF86_SVGA server to Linux/Alpha. This alpha-quality release is not guaranteed to work on all S3 boards, but X support for Linux/Alpha does now exist.

Jim Freeman announced that he has written a pre-alpha Linux Frame Relay driver is available for the Sangoma S502 Frame Relay card (a Z80 co-processed ISA card) from ftp://ftp.sovereign.org/pub/wan/fr/s502fr.tgz and ftp://www.caldera.com/pub/wan/fr/s502fr.tgz (new versions may be available by the time you read this). See http://www.sovereign.org/ for current information.

Vipul Gupta and Ben Lancki introduced alpha support for Mobile-IP for Linux. Patches are available at the ftp site anchor.cs.binghamton.edu in /pub/Linux-MobileIP/Linux-MobileIP.tar.gz

What is ELF?

ELF, which stands for Executable and Linking Format, is the new binary file format which has been implemented for Linux. Previously, Linux has used a version of the old “a.out” format, but that format has many limitations that ELF corrects. ELF was originally designed for UNIX System V Release 4, and for various reasons (technical and political) is becoming the most popular binary file format for UNIX and related operating systems such as Linux.

The Linux ELF implementation has been approximately two years in the making, and is of very high quality. Most of the ELF support in the GNU utilties was done by Linux developers to support this implementation.

Linux distributors are working to make the change to ELF as painless as possible for Linux users.

For more information on ELF, see *Linux Journal* issue 16 (August 1995) *Stop The Presses* for a light explanation, and issues 12 and 13 (April and May 1995) for a technical overview of the ELF file format.

ELF-based or “Supports ELF”?

What“s the difference? ELF-ased means that all (or perhaps almost all) of the binaries and libraries in the system use the ELF file format. ”Supports ELF" merely means that ELF libraries are included, so Linux binaries created for ELF systems will run. An ELF-based Linux distribution may also support a.out binaries by including the a.out libraries.

You only need a distribution that supports ELF in order to run ELF binaries. However, if you are running both a.out and ELF binaries at the same time, it requires a bit more memory than if you are only running one or the other, so if you need to run some ELF binaries, you are probably better off switching to a completely ELF<\#45>based distribution if you have the time to do so.

Some distributions (particularily Debian) give you the option to upgrade from a.out to ELF with ease. Red Hat has promised a floppy disk that you can use to record the most important parts of your current configuration (such as your carefully<\#45>tuned X Windows setup and networking configuration), which will be used to automatically recreate your configuration when installing Red Hat. This should work for upgrading from any distribution.

New Products

LJ Staff

Issue #19, November 1995

Red Hat Linux Developers Package, CE Editor for Linux and more.

Red Hat Linux Developers Package

Red Hat Software announced the Red Hat Developers Package, a Linux product and support program for software developers. The program includes monthly CD-ROM updates of Red Hat Commercial Linux and e-mail technical support, documentation, and Linux news. Price: $399 per year.

Contact: Red Hat Software, PO Box 3364 Westport CT 06880, Phone: 203-454-5500, Fax: 203-454-2582 E-mail: bob@redhat.com Web:www.redhat.com.

CE Editor for Linux

Enabling Technologies Group announced the availability of their latest version of the ARPUS/ce editor FREE for the Linux Operating System. “ce” is a full-screen, X-Windows based editor that provides easy-to-use text editing across a variety of UNIX platforms. Developed originally for users migrating from Apollo's Domain environment, ce was modeled after the Display Manager editor. ETG has incorporated the features of the DM editor that Apollo users liked best with new features. Some features of ce include: ceterm, multiple edit sessions, rectangular cut & paste, global bounded search and replace, coordinated mouse and text cursor control, command macros, unlimited UNDO & REDO, customized keyboard mapping, vertical and horizontal scrolling, and automatic file backup and save.

To get your FREE copy of ce for Linux, do an anonymous FTP to: ftp.std.com; cd to /ftp/vendors/ETG; and get the README file for detailed instructions. In this location, you will find the FREE Linux copy and also evaluation copies of ce for other vendors' platforms.

Contact: Enabling Technologies Group, Inc. 8601 Dunwoody Place, Suite 300 Atlanta, Georgia 30350. Phone: 404-642-1500 FAX: 404-993-4667 E-mail: arpus@etginc.com

Mathematica for Linux

Wolfram Research is now shipping a new version of Mathematica for Linux. With Mathematica running under Linux, users can take advantage of Mathematica's extensive numeric and symbolic capabilities, 2D and 3D graphics, a high-level programming language, as well as the many Mathematica applications available. Mathematica for Linux includes support for MathLink via TCP/IP. MathLink, Wolfram Research's communication standard, lets users make seamless connections between Mathematica, their own applications, and other commercially available software. With MathLink it is possible to exchange information between Mathematica and other programs either on one machine or among several machines on a network.

Contact: Wolfram Research, Inc. Phone: 800-441-6284 or 217-398-0700 E-mail: info@wri.com WWW www.wri.com

Contact: Wolfram Research Europe Ltd. Phone: +44-(0)1993-883400. E-mail: info-euro@wri.com

Contact:Wolfram Research Asia Ltd. Phone: 81-3-526-0506 E-mail: info-asia@wri.com

Archive Index Issue Table of Contents

Advanced search

Linux System Administration: Using LILO, The Linux Loader

Æleen Frisch

Issue #19, November 1995

This excerpt from *Essential System Administration* describes useful details of booting Linux on a PC.

In general, the boot process on a microcomputer has three stages: the system's master boot record (MBR) contains the primary boot program which starts the boot process and loads a secondary boot program from the boot blocks of the active partition; this second boot program is what loads the actual kernel.

Linux provides LILO, the Linux Loader, which can function as either a master boot program or a secondary boot program. **lilo** is installed with a command like this one:

```
# lilo -C /etc/lilo.conf
```

The **-C** option specifies the location of LILO's configuration file. (The location in the preceding command is, in fact, the default location, and so the **-C** clause is redundant.)

The lilo.conf file specifies LILO's behavior for certain aspects of the boot process and also defines the kernels and operating systems that it can boot. The following excerpt from a lilo.config file lists the most important entries—and the ones that you are most likely to want or need to modify:

```
# Wait 10 seconds before autobooting 1st entry.timeout=100
# Allow user to enter a boot command.
prompt
# Where to install/configure LILO
# (no partition #=MBR) [see below]
boot=/dev/hda
...
# Text file displayed before boot prompt.
message = /boot/boot.message
#
# Default kernel is the first one listed.
```

```
image = /vmlinuz
# Boot prompt command to boot this kernel.
  label = linux
# Fix for Sony CD-ROM (post 1.1.72)
# Specifies parameters to pass to kernel,
# changing CD-ROM's compiled-in I/O address.
  append = "cdu31a=0x340,0,"
#
# An alternate Linux kernel
image = /safe
# Its boot command
  label = safe
  append = "cdu31a=0x340,0,"
...
# A different operating system (DOS)
other = /dev/hdb1
# Use the D: drive boot loader.
  loader = /boot/any_d.b
# Use this partition table.
  table = /dev/hdb
# This is the corresponding boot command.
  label = ddog
```

[I tend to install LILO both in the MBR and the Linux partition for maximum flexibility, by running a second LILO command using its **-b** option (which replaces the boot entry in the configuration file):**# lilo -b /dev/hda1 -C /etc/lilo.conf**This way, if I decide to remove LILO from the MBR, I'll be all set to switch over to the Linux partition version.]

The final section ("stanza") illustrates the format for booting a DOS partition on the second hard disk; it uses an alternate loader, any_d.b, which tricks DOS into thinking it's on the C: drive. There are also loaders provided for OS/2 on the D: drive and DOS on the B: drive (os2_d.b and any_b.d, respectively; chain.b is the default loader for other operating systems).

If the label for a stanza is omitted, it defaults to the final component of the image or other entry (for example, vmlinuz or hda1).

The entry for a DOS partition on the C: drive is simpler, looking something like this:

```
other = /dev/hda1  label = dos
  table = /dev/hda
```

This stanza is actually what you need for any foreign operating system on /dev/hda1. There is one additional trick needed if SCO Unix is the operating system you want to boot. In order for LILO to successfully boot a SCO Unix partition, that partition must be the only active partition on the C: drive. This means you will have to turn off the active (boot) flag on the Linux partition and turn it on for the SCO Unix partition, using the Linux fdisk or cfdisk, before trying to boot SCO Unix (in addition to running LILO to install the new configuration). Note that LILO must be the master boot loader in this case.

Note: You will need to rerun the LILO command to reinstall it every time you rebuild the kernel or change any relevant aspect of the disk partitioning scheme. If you forget to do this, the system will not boot and you'll have to boot from a floppy. You will also need to rerun LILO if you change the text of the boot.message file.

More Complex Booting Scenarios

Booting a Linux partition on the second hard drive is also possible. For this to work, LILO must be installed in the MBR of the system's boot disk, as well as the secondary boot program in the Linux partition itself—this is usually taken care of when you install Linux on the hard disk and will be assumed in what follows. In this case, the best way to proceed is in two stages:

- First, set up a lilo.conf like this one:

  ```
  boot=/dev/hdaroot=current
  image=/vmlinuz
    label=linux
  other=/dev/hda1
    unsafe
  other=/dev/hda2
    unsafe
  ...
  other=/dev/hdb4
    unsafe
  ```
- Define all of the partitions on both hard disks in the same way; the unsafe keyword tells LILO not to read the boot blocks or the disk's partition table for that entry—it basically says, “Trust me and do what I tell you.” Install this LILO configuration, and make sure that Linux is bootable.
- Then, modify the file, changing entries for any bootable partitions on /dev/hda to their correct form and removing ones you don't need, and rerun the LILO command.

It is also possible to boot a Linux partition on each of two disks. The procedure for doing so is the following:

- Decide which one will be the usual Linux boot partition and set up LILO to boot it and any other non-Linux operating systems on both disks. Create an entry like the following for the second Linux partition:

  ```
  other = /dev/hdb2  label=eviltwin
    unsafe
  ```
- Create a boot.message file which tells you which Linux will be booted when you select the default option. Install this configuration into the MBR on the C: drive.
- Create (or retain) another LILO configuration for the second Linux partition, this time including an unsafe entry for the first Linux partition if you want to (this again assumes that LILO is installed in that partition, which usually happens at upon installation of the OS). Make sure that this partition's boot.message file also lets you know where you are. Install this

configuration into the Linux partition only—make sure that the boot entry specifies the partition and not the disk as a whole.

The boot sequence will then go something like this:

```
Welcome to gallant.Boot choices: linux (default; on C:), dos,
 eviltwin (Linux on D:), sco
boot: eviltwin
Welcome to goofus.
Boot choices: test (default; on D:),  goodtwin (Linux on C:)
boot: [Return]
Loading test...
```

Given these selections, Linux will boot from the D: drive. What happens is the LILO from the MBR on drive C: runs first, and it then starts the boot program on the Linux partition on the D: drive—which is again LILO. That (second) LILO then loads the kernel from the D: drive. (Note that if you wanted to, you could just keep popping back and forth between the LILO programs on C: and D: ad infinitum.)

If you think this is pretty silly, then omit the prompt keyword from the LILO configuration file for the D: drive (as well as its image section for the Linux partition on the C: drive), resulting in a simple lilo.conf file on the D: drive:

```
install=/boot/boot.bboot=/dev/hdb2
root=/dev/hdb2
map=/boot/map
image=/vmlinuz
  label=linux
```

Once this is installed, selecting eviltwin at the initial boot prompt will immediately boot the Linux partition on the second hard disk.

LILO's -r option

Sometimes it is useful to be able to run LILO for a disk partition mounted somewhere other than /. For example, if you have another Linux root filesystem mounted at /mnt, you might want to run LILO to install the kernel (currently) at /mnt/vmlinuz using the configuration file /mnt/etc/lilo.conf. LILO's **-r** option is designed for such a purpose. It sets the root directory location for the LILO operation to the directory specified as its argument and looks for all files relative to that point. Thus, for the scenario we've been discussing, the correct command is:

```
# lilo -r /mnt
```

The boot.message File

The boot.message file is displayed before the boot prompt is issued. Here is an example boot.message file:

```
Welcome to JAG Property of the Linux Guerrilla Hackers
AssociationComputational science is not for the fainthearted!
Our current boot offerings include:
    * linux (smaller test kernel--1.3.10 currently)
    * safe (Yggdrasil distribution)
    * hacked (do you feel lucky?)
    * ddog - guess what ... (on D:)
```

An effective file will list all the defined labels (but it needn't be this eccentric).

Restoring the DOS Master Boot Program

Should you ever need to, here is the procedure for restoring the DOS master boot program:

- Boot from a bootable DOS floppy.
- Run the command **fdisk /MBR**

The Linux-FT Bootmanager

The Linux-FT distribution replaces the entire LILO apparatus with its Bootmanager facility, a graphical menu-based, user-configurable utility by which you can select which kernel to boot. Its display looks something like this:

```
                Bootmanager
  Name      Location
MULTIUSER   :3/vmlinuz ro root=/dev/sda3 2
SINGLE      :3/vmlinuz ro root=/dev/sda3 single
FLOPPY      A:
INSTALL     :3/vmlinuz root=/dev/sr0 ramdisk=300 5
DOS         C:1
LUCKY       :4/test_kern ro root=/dev/sda4
```

The Location field indicates the kernel to boot. For example, the MULTIUSER item will boot the file /vmlinuz on the third disk partition of the current disk, using the indicated device as the root filesystem, booting to run level 2. The DOS item will boot partition 1 on the first hard disk (C:). The final item is one added for this system.

Customizing the Bootmanager is easy. Press ESC to override the automatic boot timeout, and then use the cursor and function keys listed at the bottom of the screen to edit the relevant fields. Hardware options applicable to all boot sequences may be entered into the designated field below the boot choices menu.

Introducing Linux Loadable Modules

Very recently (since version 1.2), the Linux kernel has supported loadable modules. As of this writing, the Debian and Linux-FT distributions use this functionality by default.

In this scheme, you build a minimal kernel and dynamically load modules providing additional functionality as required. Such an approach has the advantage that many types of system changes will no longer require a kernel rebuild; it also has the potential to significantly decrease the size of the kernel executable.

The modules package provides utilities for building, installing and loading kernel modules (including ones needed to build a kernel with module support). Running **make modules** after building a kernel will create the loadable modules files, and **make modules_install** will install them into the /lib/modules/1.2.6 directory tree (where the final component denotes the kernel release level). The configuration file /etc/MODULES (or /etc/modules) lists modules to be loaded automatically at boot time:

```
sysviBCS
ppp
```

This file says to load the modules supporting the System V filesystem type, the Intel Binary Compatability facility, and the PPP facility.

The following utilities are used to manipulate modules:

- depmod: Determines dependencies among modules. For example, the ppp module I selected earlier requires the slhc module to function. depmod creates the file modules.dep in the relevant subdirectory of /lib/modules. This utility may be run automatically at boot time or you may need to execute it manually after building modules.
- modprobe: Loads a module as well as all of those that it depends on (usually used to load modules on boots).
- insmod: Loads a module interactively.
- rmmod: Unloads a loaded module from the kernel (provided it is not in use).
- lsmod: Lists currently-loaded modules:

  ```
  Module:        # pages:  Used by:iBCS               19
  ppp                 5
  slhc                2    [ppp]
  sysv                7
  ```

At this point, the loadable modules facility is in its infancy, and only a few modules are available, but this will undoubtedly be the way Linux kernels generally operate in the not-too-distant future (and it is the way many other Unix versions already work).

Æleen Frisch (aefrisch@lorentzian.com) manages a very heterogeneous network of Linux and other Unix systems and PCs. After finally finishing the second edition of *Essential System Administration*, she has gone back to her true calling in life, pulling the string for her cats, Daphne and Sarah.

Archive Index Issue Table of Contents

Advanced search