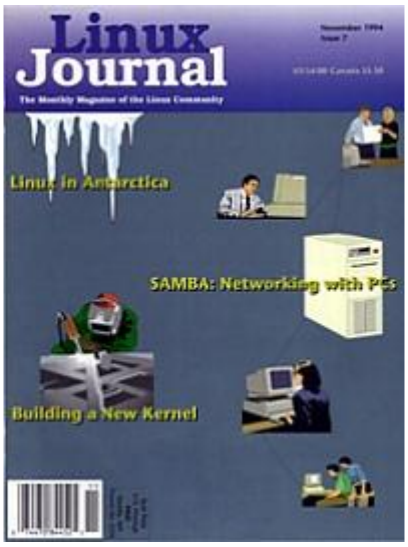
Advanced search

# *Linux Journal* Issue #7/November 1994



Archive Index

Advanced search

# Making the Most of Andrew

**Terry Gliedt**

Issue #7, November 1994

In this article, Terry Gliedt completes our tour through the Andrew project with details on using and customizing AUIS applications.

In previous articles I focused a great deal on the ez editor and on the multi-media user mail agent, messages, but these are only two of a rich set of applications. Here are a few other applications that you'll find useful:

**bush** (see Figure 1) provides a graphical interface to the file system. It shows the directory hierarchy, filenames and their attributes. It also provides a window where the ez editor will run as well as a means to invoke an arbitrary command on a file. With *bush* you can sort the list, rename, and delete files, too.

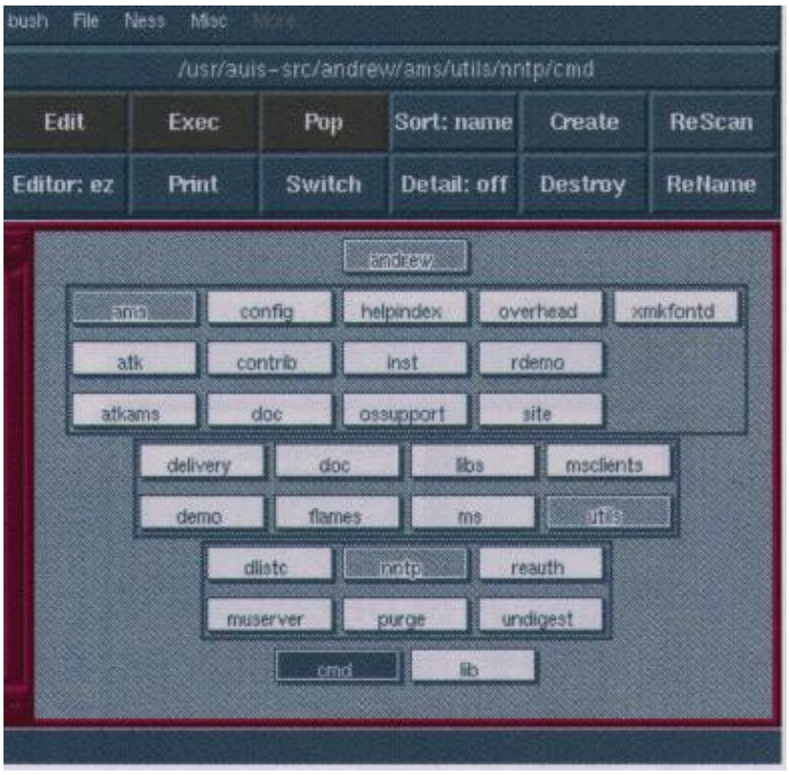


Figure 1. Bush Showing a Tree of Directories

**chart** allows you to create simple graphs from numeric data. The data can be presented as a histogram, pie chart, or line graph.

**ezdiff** is not actually an application, but rather a procedure you can call from all *ez* windows. This is used to run the *diff* command on the data in two windows and then interactively see the difference between the files.

To try *ezdiff*, edit two files (e.g., **ez/etc/hosts .deny/etc/hosts.equiv**). In each window, select the *Start* item on the *Ezdiff* menu card. As you select the *Next* item on the *Ezdiff* menu card, the differences in each file are shown as highlighted data. This makes it very easy to copy lines selected in one window and paste them in the other. And yes, *ez* correctly remembers the locations of all the other differences when you make changes to either document. This is a life saver when looking at various versions of a file.

**figure** (see Figure 2) is a fairly conventional drawing editor. The files usually have an extension of **.fi**. *Figure* can create a document with lines, circles, boxes, and other insets (e.g., rasters or text), and then move them around, reshape, etc. Since *figure* creates an ATK data-stream, you can easily insert figures in other AUIS documents.

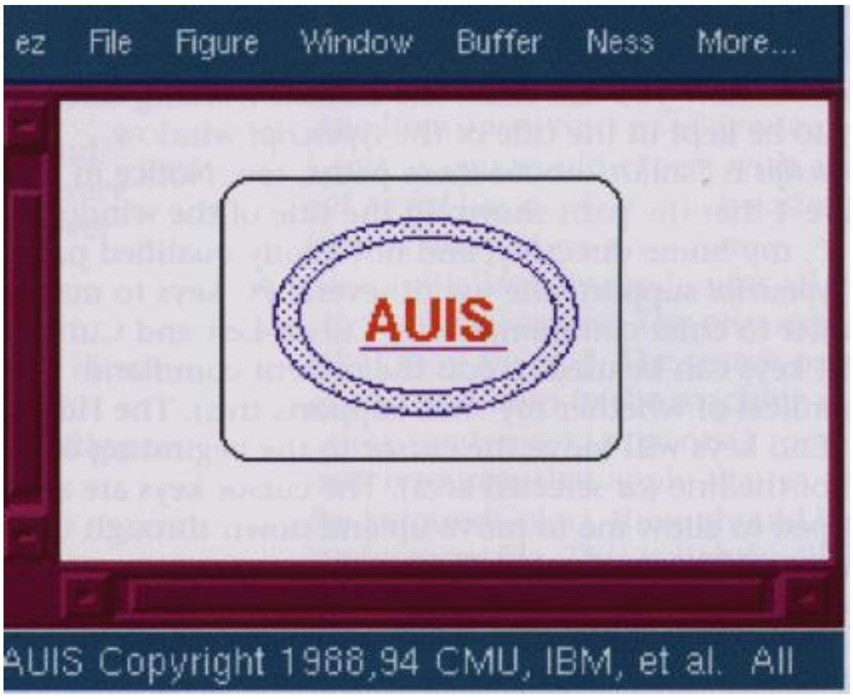


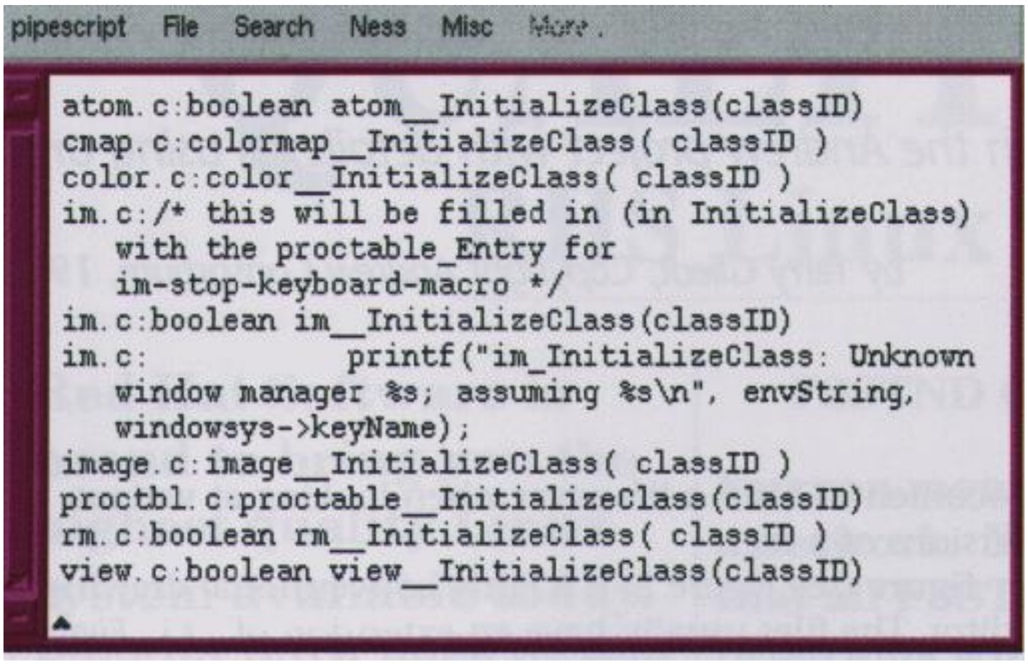
Figure 2. Sample AUIS Figure

**pipescript** (see Figure 3) reads data from stdin and displays it in a window. This is very convenient with pipes, because the output data does not disturb your *xterm* window and yet is not in a file you need to remove. The data can be conveniently scrolled, searched or saved. I use it all the time for commands like **tar tzvf auis63L1-wp.tgz | pipescript**.

**raster** provides a simple means to edit digitized pictures (called rasters). The data must be in AUIS data-stream format. The **pbmplus** package (available on sunsite) provides a wide set of filters to convert between various forms of digitized data. For example, to convert from a TIFF to an AUIS-raster, I'd use this command:

```
tifftopnm test.tif | ppmquant 256 | ppmtopgm\
| pgmtopbm | pbmtocmuwm > test.ras
```

**typescript** (see Figure 4) provides an alternative to *xterm*. You enter commands to your shell in the input window and the results are shown in the same window where they can be scrolled, searched or saved. *Typescript* does not support curses like xterm does, so you cannot run vi or bash in a typescript, for example. However, you can use the tcsh or pdksh (ksh) shells.

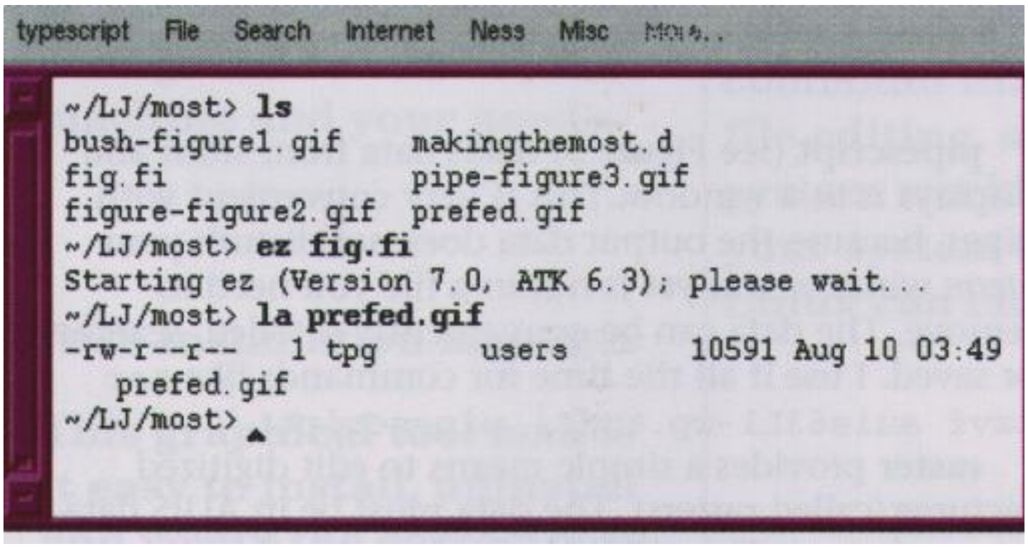


```
atom.c:boolean atom__InitializeClass(classID)
cmap.c:colormap__InitializeClass( classID )
color.c:color__InitializeClass( classID )
im.c:/* this will be filled in (in InitializeClass)
   with the proctable_Entry for
   im-stop-keyboard-macro */
im.c:boolean im__InitializeClass(classID)
im.c:           printf("im_InitializeClass: Unknown
   window manager %s; assuming %s\n", envString,
   windowsys->keyName);
image.c:image__InitializeClass( classID )
proctbl.c:proctable__InitializeClass(classID)
rm.c:boolean rm__InitializeClass( classID )
view.c:boolean view__InitializeClass(classID)
```

Figure 3. grep InitializeClass *.c | pipescript

*Typescript* has several other features that I find very useful. The **README** for the word processing package describes how you can cause the current working directory to be kept in the title of the *typescript* window. *Typescript* is "smart" about these paths, too. Notice in Figure 4 that the path shown in the title of the window is "~", my home directory, and not a fully qualified path.

*Typescript* supports the use of several PC keys to make it easier to enter commands. The Cursor-Left and Cursor-Right keys can be used to edit the current command (regardless of whether my shell supports this). The Home and End keys will move the cursor to the beginning or end of the line (or selected area). The cursor keys are also mapped to allow me to move up and down through the command history. Cursor-Up gets the previous command entered, Cursor-Down the next.



```
~/LJ/most> ls
bush-figure1.gif    makingthemost.d
fig.fi              pipe-figure3.gif
figure-figure2.gif  prefed.gif
~/LJ/most> ez fig.fi
Starting ez (Version 7.0, ATK 6.3); please wait...
~/LJ/most> la prefed.gif
-rw-r--r--    1 tpg     users      10591 Aug 10 03:49
   prefed.gif
~/LJ/most>
```

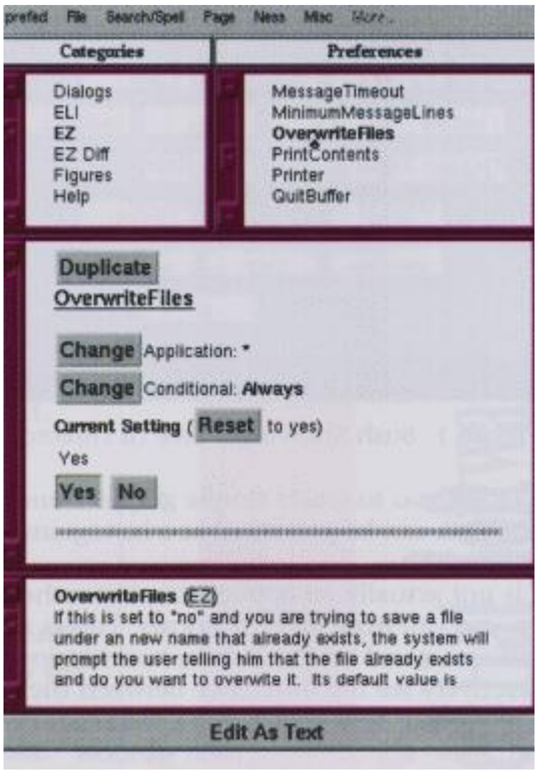
Figure 4. Sample Typescript Window

In my opinion an even more useful feature is command completion. If I type **ls** and then press Cursor-Up, I get the previous command that began with “ls”. File completion is supported with the tab key. If I enter **ls src/tp** and then press the tab key, *typescript* will complete the path “src/tp” to “src/tpg-config” (in my case). It makes entering paths to files ten times easier. I love it!

If I enter a command like **ls /etc**, the window will likely fill and scroll automatically. If I want to see the beginning of the scrolled data, I must use the mouse (or Page-Up/Down keys) to scroll. However, if I enter **ls /etc control-J** (where I enter the command with a control-J keystroke, rather than the normal Enter), the command is executed, but the output is pushed to the top of the current window so I do not need to scroll.

*Typescript* also allows me to create my own menus in the file **~/.shmenu**. In Figure 4, you can see I have a menu card *Internet* on the menubar. This is from my own **.shmenu** file which looks like this:

```
#  User's .shmenu file
#       Do help .shmenu for more information.
#       Establish the pop up menus for the
# typescript window.
Internet~20,FTP<->Sunsite export~15:ftp
sunsite.unc.edu
Internet~20,FTP<->CMU~16:ftp.andrew.cmu.edu
Internet~20,FTP_Anonymous~20:anonymous
Internet~20,FTP_Ident~21:tpg@mr.net
Internet~20,FTP_bin~22:bin
Internet~20,FTP_submissions~23:cd pub/next/submissions
Internet~20,Telnet->CMU~31:telnet ftp.andrew.cmu.edu
Internet~20,Bring Slip Up~40:/usr/local/bin/slipup
Internet~20,Take Slip Down~41:/usr/local/bin/slipdown
```



OverwriteFiles

Application: *

Conditional: Always

Current Setting ( Reset to yes)

Yes

OverwriteFiles (EZ)
If this is set to "no" and you are trying to save a file under an new name that already exists, the system will prompt the user telling him that the file already exists and do you want to overwite it. Its default value is

Figure 5. Modifying preferences with prefed

## Using Filters

All AUIS applications provide a means to use a filter to modify a selected area. The *Misc* menu card provides items to fold data to lower case, upper case or to “flow” the data together. To use filters, simply select an area (so it is highlighted) and select the correct item. There is also an item (*Filter Prompt*) which allows you to specify your own filter. For instance, specifying the filter sort would cause the selected text to be sorted—just like you would in a conventional Unix pipe.

Introduction to AUIS

## Personalizing AUIS

You can tailor the behavior and appearance of your AUIS applications with your own **~/preferences** file. Each time an application begins, it checks a set of resources you have specified in your **preferences** file. The resources in the **preferences** file are similar to other X-resources. A sample **preferences** file is provided in **/usr/andrew/sample.preference**. If you edit this file you will find lines like these:

```
Messages.Geometry: 750x600
Figure.BackgroundColor: White
*.BackgroundColor: LightSkyBlue4
```

The first field (e.g., Messages, Figure or “*”) is the application name. The asterisk means it applies to all AUIS applications. AUIS resource names are case insensitive, so “Messages” is the same as “messages”. The second field is specific to the application. Most of these, like “Geometry” or “BackgroundColor”, will be pretty obvious. Others, like “XStyleSelections”, will be a mystery unless you read the preferences help file (**auishelp preferences**). Some of these resources are specific to a single application and others apply to all AUIS applications.

The third field (after the colon) is the value of the resource and is tied directly to the second field. These resources can be rather confusing. To make it easier to understand what resources can be set for which applications, a specialized preferences editor was written, **prefed**. As you can see in Figure 5, the *prefed* window is divided into four areas. You select the application name in the upper left hand window (“EZ” in this case). The upper right hand window has the list of resource names (“OverwriteFiles”). The middle window shows you what the current setting is and allows you to change it. Finally, the bottom window provides a description of what the resource controls. When you select the item *Save*, your **preferences** file will be modified, just as you would expect with any editor. The addition of *prefed* has made it much easier to figure out this bewildering array of resource names. When an application initializes, it searches for all resources that apply to that application. The resources are applied in the order that they are read. The search order is first in *~/preferences* and then in the machine-wide “preferences” file in **/usr/andrew/lib/global.prf**. Thus, if you specify any resources that apply against all applications (e.g., *.BackgroundColor), these should be at the bottom of your preferences file. This is sightly different than resources for conventional X-applications.

One thing to keep in mind is that *preferences* are only read when an application initializes. If you change the *ez* BackgroundColor, for example, this will not take effect until you start a new *ez* from the command line (from an *xterm* or *typescript*).

*Messages*, the multi-media mail user agent, is unique with respect to its treatment of preferences. Since there are a great many preferences resources for messages, a long time ago the author of messages added his own specialized way to set preferences for messages by selecting the Set Options item on the Other menu card.

## Printing

Printing is an area where it seems everyone needs to “have their own way”. AUIS printing involves two steps. Applications generate *groff* data and the first step is to format this to generate the printer-specific datastream. The second step is to simply deliver the stream to the printer. These two steps can be overridden with your *preferences*. The default preferences settings are:

```
*.FormatCommand: /usr/andrew/etc/atkprint /tmp/%s.n |
*.PrintCommand:  lpr -P$PRINTER
```

These two commands are joined in a pipeline (hence the pipe character, "|", at the end of *FormatCommand*). When this command is built, the **%s** will be changed to the name of the *groff* input file.

Many AUIS applications actually generate PostScript data with just a simple *groff* "wrapper" around the PostScript. In the next major release, AUIS will generate PostScript directly and the need for a FormatCommand will be eliminated. But for now, if you are having difficulty you might want to copy **/usr/andrew/etc/atkprint** and make your own version which you can modify to meet your needs. You can then specify your own *FormatCommand* resource to invoke your own version of atkprint.

Sometimes I want to capture the groff input, so I simply make a quick *preferences* change

```
*.FormatCommand: cat /tmp/%s.n > /tmp/test.n
```

to copy in the input file to **/tmp/test.n**. I can then execute the print sequence manually with:

```
/tmp/atkprint /tmp/test.n | pipescript
```

so I can see exactly what's happening in the shell script. On the other end, if I want to simply capture the PostScript, I can change *preferences* to:

```
*.PrintCommand:  cat > /tmp/test.ps
```

There are obviously many ways to accomplish either of these steps. The point to remember is that AUIS printing is entirely externalized. You can control it to any degree you need.

## Initialization Files

At startup AUIS applications look for several initialization files. There are two classes of these, global or machine-wide files found in **/usr/andrew/lib** and personal files, found in your home directory. Each application can have its own initialization file. For example, ez would check for files in this order:

```
~/.ezinit
/usr/andrew/lib/global.ezinit
~/.atkinit
/usr/andrew/lib/global.atkinit
```

If the **.ezinit** file is found, the **global.ezinit** is not read. Similarly, if **.atkinit** is found, the **global.atkinit** file is not read. This ordering means that when you create an initialization file, you should make it refer to the files that it masks by using an “include” statement. So, if you are going to make a **.ezinit** file, you would want to include the **global.ezinit** by:

```
include /usr/andrew/lib/global.ezinit
include $HOME/.atkinit
```

and your **~/.atkinit** should probably include **/usr/andrew/lib/global.atkinit**.

If you don't include the global file, you will not get important default settings. Note, though, that not all programs have global initialization files. When something is defined more than once, the last definition stays in effect.

While I used *ez* in this example, this works exactly the same for *typescript* or any other AUIS application. *Typescript* would read **~/.typescriptinit**, or if that file did not exist, it would read **/usr/andrew/lib/global.typescriptinit** instead. All applications look for **~/.atkinit and /usr/andrew/lib/global.atkinit**. These files are described in more detail with the command auishelp initfiles.

## Controlling Menus

In AUIS applications menus are simply a means to call a procedure (method) for an object. AUIS has hundreds of these defined and they provide enormous functionality. To define a menu, edit one of the initialization files described previously (a personal initialization file is probably best) and add lines like these:

```
addmenu filter-filter-region-thru-command
"Misc,Flow~20"  textview \
        filter inherit "flowtogether"
addmenu textview-lowercase-word         "Misc,Lower~21"
textview
addmenu textview-uppercase-word         "Misc,Upper~22"
textview
addmenu filter-filter-region            "Misc,Filter
Prompt~25" textview
```

The entries shown add items like *Flow* and *Lower* to the *Misc* menu card. The order of these items is determined by the numbers (20, 21, etc.). When you select a item like *Lower*, the procedure *textview-lowercase-word* will be called and the selected area will be folded to lower case.

Knowing the procedure names is important to create menu cards. There is no fixed list of all the procedures that are in AUIS because the AUIS objects are dynamically loaded. To help you find out what is available, select *Describe Proc Table* on the *Misc* menu card. This will open a window which displays a list of all the procedures and a short description of purpose for each.

A more complex example is shown when the procedure *filter-filter-region-thru-command* is called when the *Flow* item is selected. In this case the procedure calls the filter *flowtogether*. Flowtogether is a simple filter which combines lines together to remove excess whitespace and create data in paragraphs. I use this filter in messages when I want to quote part of some mail and make it nicer looking.

Most of the menus you see in the Linux distribution were added using addmenu in the various initialization files. You can add your own menu cards by adding addmenu commands in your private initialization files. More details on adding menu cards can be found with the command **auishelp initfiles**.

## Filetypes

If I issue the command **ez test.d**, the data in this file will be an AUIS text document, but if I edit the file **test.document**, it will be just simply ASCII data. The difference obviously has something to do with the extension of the file. Linux, like all conventional Unix systems, really has no innate “knowledge” of what is in a file, but we all have expectations for what's in the file **test.c**. It's just a matter of convention. AUIS has its own conventions and these are controlled in the file **/usr/andrew/lib/global.filetypes** which has entries like these:

```
addfiletype .Xdefaults     rawtext   "template=rawtext"
addfiletype .c          ctext     "template=c"
addfiletype .h          ctext     "template=h"
addfiletype .d          text      "template=default"
addfiletype .doc        text      "template=default"
addfiletype .help       text      "template=help"
```

Addfiletype commands allow you to map extensions to inset types so that new documents you create with a certain extension will get the proper inset type specified by the “template=” keyword.

Editing **.Xdefaults** will have the template **rawtext** (which will insure that copying an AUIS datastream results in a simple ASCII string and not some bold text) and editing **.Xdefaults.old** results in the **default** template (which allows you to copy and retain **bold** text) which is probably not what you want. The current AUIS distribution does not allow you to use wildcards in addfiletype commands, so you must explicitly map .Xdefaults and **.Xdefaults.old** individually. More details on filetypes can be found with the command **auishelp initfiles**.

## Key Definitions

In AUIS applications, keystroke combinations are another means to get a procedure called for an object. The same procedures are available for keybinding as for menu items. To define a set of keybindings, edit one of the initialization files described previously and add lines like these:

```
addkey compile-build          ^X^E srctextview compile
addkey compile-next-error     ^X^N srctextview compile
addkey compile-previous-error ^X^P srctextview compile
addkey fcomp-complete-command-forward \eB typescript fcomp inherit
addkey fcomp-complete-filename ^I typescript
addkey fcomp-possible-completions \e^I typescript
```

The first line of the example says that when you are in a srctextview inset (e.g., editing C-source), the procedure name compile-build will be called when you press the keystrokes <control>X followed by <control>E. (The **\eB** in the fourth line means you should press the escape key and then <shift>b.) You can see a dynamically created list of all the keys that are bound to a procedure by selecting the *Describe Bound Keys* item on the *Misc* menu card. You can also query what procedure will be called for any keystroke by selecting the *Describe Key* item on the *Misc* menu card.

## Conclusion

AUIS is far more than just *ez* or *messages*--and yet in many ways it is no more. AUIS is built on a toolkit of objects which combine to provide a set of tools which are consistent in their look and feel and which can be extended or combined with new applications with remarkable ease. In these four articles on AUIS I have not attempted to show any of the underlying toolkit. The Andrew Consortium is dedicated to extending and disseminating this technology. If you think your organization could benefit, I'd encourage you to contact the consortium and talk with us about what else has been done and what's new.

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

# Linux in Antarctica

**Andrew Tridgell**

Issue #7, November 1994

Between GPS satellites and the Australian Surveying and Land Information Group are eight Linux systems, which have been collecting data for the last year. Find out how they're set up and how they keep everything running.

I was approached in September 1993 by Martin Hendy at AUSLIG (Australian Surveying and Land Information Group) to give him some information about Linux to see if it was suitable for a large project they had underway.

After I showed him a couple of Linux systems and gave him a rundown, he decided to go ahead, and the end result is that there are now eight Linux boxes scattered around Australia, gathering data on the Global Positioning System. The current systems are at Darwin, Ceduna, Alice Springs, Mawson Base (Antarctica), Davis Base (Antarctica), Casey Base (Antarctica), Macquarie Island, and Hobart. More are planned for other parts of Australia.

The application is a data gathering one. The Linux boxes are attached via serial ports to a “TurboRogue” satellite receiver system, which monitors the 32-satellite Global Positioning System. Data is downloaded from the satellites and stored in a 4MB flashcard in the back of the TurboRogue, and from there it is downloaded to the Linux systems. These systems store the data and forward it to a base system in Canberra.

The monitoring of the GPS system is essential for accurate surveying work. Some other countries (notably Canada) have set up similar networks for the same purpose.

Some of the Linux systems are connected directly to AARNet via Ethernet (yes, Antarctica is networked!) while others have 14.4 Kbps modems and download data via scheduled cron jobs, using the **term** package. Even the machines on the net have modems, as we can't absolutely rely on the network link.

## The Hardware

The Antarctic Linux systems are Digital 48633DX MT PCs with 8MB RAM. They have two 345MB IDE hard disks, with only one electrically connected, (the other is identically configured as a backup). They have one 1.44MB floppy drive and a spare floppy (again, not connected). They have WD8013 Ethernet cards and DataLink 14.4 Kbps modems.

The Linux systems for the other Australian sites are rack-mounted “clone” PCs, for various reasons. They are identically configured.

Each system has four serial ports, each on its own interrupt. We achieved this by making some minor modifications to a cheap 2-port serial board we bought in Canberra. The more recently installed systems have eight serial ports using two 4-port cards. The extra serial lines allow for control of more equipment.

The other custom hardware is a “rebooting module” designed and built by Anthony Wesley here in Canberra. It is a small box that attaches to a serial port, a power lead and the reset line of the computer. Basically, it expects a “healthy” signal from the computer every five minutes on the serial line, or else it “presses reset” by shorting the reset line for 30 seconds. These boxes are very rarely needed, but it's good to know that they are there.

A process runs a script to check half a dozen things every minute and outputs the “healthy” signal if they all pass.

## The Software

The systems originally ran 0.99pl12 with a few patches. I chose this because I have found it stable for my own use. We have upgraded (only rarely) when we needed particular features or bugs fixed. One of the features of having the PCs running Linux (they were considering DOS) is that we can completely replace the system remotely, either via modem or the net.

Each hard disk has four partitions: a DOS partition with their old software on it (just in case), two Linux root partitions (identically configured), and a data partition (the bulk of the disk).

The Linux root partitions are 20MB and are only half full! Most of them are, in fact, taken up with things that are only included “just in case”, like kernel sources, gcc, sources for the TurboRogue controlling software and even Emacs 19.16 (to make life a little easier).

I have designed the system to be very small. It is small enough, in fact, that the whole system can operate from a single 1.44MB floppy if I leave out the “optional extras”. This is achieved by having the whole of the /usr tree in a compressed tar file which uncompresses onto a ramdisk on bootup, and having next to nothing outside of /usr. The systems on the hard disk are already uncompressed.

This means that in case of emergency we can tell the on-site operators to insert one of the duplicate “Emergency root disks” and we can operate the complete system remotely just from the floppy, including disk-fixing tools like **fdisk** and **e2fsck**.

**cron** is used to control the regular downloading of data from the TurboRogue and transmitting the data to Canberra. Checksums are used to ensure the data is correctly transmitted, and it is re-transmitted if necessary.

Since October 1993, when the first systems in Antarctica came on-line, an enormous amount of data has flowed from the remote systems to the main system here in Canberra. It is a great credit to Linux that it has performed so well under difficult conditions. Congratulations to Linus and all the Linux developers!

**Andrew Tridgell** has been involved in the Linux community since December 1992. He has worked on a number of projects, including DOSEmu, the “Linux in Antarctica” project, some TCP hacks, and, more recently, Samba. He is currently the convener of the Canberra Linux Users Group. Andrew and his wife Susan live in Canberra and enjoy bushwalking, sailing, watching TV, speaking Swedish, and eating pizza.

Archive Index Issue Table of Contents

Advanced search

# Samba—Unix Talking with PCs

**Andrew Tridgell**

Issue #7, November 1994

*Linux Journal* mentioned to Andrew Tridgell that we wanted to interview him about his work with Samba, and he responded with this enlightening and entertaining account of the development of the Samba package, answering all the questions we had intended to ask before we could ask him.

The need to get the mainstream PC operating systems to talk amicably with Unix has been around for a long time. Recently, yet another option has emerged which takes a different tack from previous methods. I'm talking about SMB for Unix.

The dominant file-sharing protocol in the Unix world is NFS. The dominant protocols in the DOS/Windows/NT worlds are Novell and SMB. SMB is also known as LanManager, although LanManager is really only one implementation among many.

If you want Unix to talk to DOS, so that they can share file and print resources, then there are basically two choices. The first is to make the PC look like a Unix box by getting it to talk NFS. The second is to make the Unix box talk one of the PC networking protocols.

The best choice for getting NFS on a PC is to run a PC Unix such as Linux. This, however, is not a realistic option just yet for many PC users. The alternative is to run an NFS client on the PC. The problem with this approach is that the NFS protocol was never designed with PCs in mind. The security, administration, and general utility of PC-based NFS clients is far from ideal.

## Security ad nauseum

Just realising that providing NFS service from a Unix box to a PC allows the PC to handle the authentication should make any sane system administrator “go off his lunch”. The NFS protocol requires that the Unix box trust the PC completely. In some implementations this is hidden behind the appearance of a “login” procedure and password protection. Don't let it fool you. As I have demonstrated on several occasions, any half-wit with access to the PC's console can fool the Unix box into giving write access to just about anyone's files, without the need for any passwords.

If that isn't enough, just try to administer a large bunch of PCs running NFS clients. Nightmare!

So what about the other approach? Can you make a Unix box talk one of the PC networking protocols? The short answer is yes. Products have been around for some time for some Unix flavours to talk to PCs on their own terms. The problem is that software vendors realise that this is a very useful thing to be able to do, and charge accordingly. There must be a better way.

## Samba

There is a better way. Samba is a free implementation of the SMB protocol for Unix. The SMB protocol is the native file- and printer-sharing protocol for Windows for Workgroups, LanManager, Windows NT and OS/2. The SMB protocol is an X/Open standard and is in use on millions of PCs worldwide.

Before I tell you all about the current version of Samba and what it can do for you, I'm going to indulge in a little history. There is probably a similar history behind many other free software packages. They all start somewhere.

## Some History

The whole thing really started in December 1991. I was (and still am) a PhD student in the Computer Sciences Laboratory at the Australian National University, in Canberra, Australia. We had just acquired a beta copy of the PC X server *eXcursion* from Digital, and I was testing it on my PC. At this stage I was an MS-DOS user, dabbling in windows.

*eXcursion* ran (at the time) only with Dec's “Pathworks” network for DOS. I had until then been using PC-NFS to connect to our local Sun workstations, and was reasonably happy with it. In order to run Pathworks, I had to stop using PC-NFS and try using Pathworks to mount disk space. Unfortunately, Pathworks was only available for Digital workstations running VMS or Ultrix, so I couldn't mount disk space from the Suns anymore.

I had access to a DecStation 3100 running Ultrix that I used to administer, and I got the crazy notion that the protocol that Pathworks used to talk to Ultrix couldn't be that hard, and maybe I could work it out. I had never written a network program before, and certainly didn't know what a socket was.

In a few days, after looking at some example code for sockets, I discovered it was pretty easy to write a program to “spy” on the file sharing protocol. I wrote and installed this program (the **sockspy.c** program supplied with Samba) and captured everything that the Pathworks client said to the Pathworks server.

I then tried writing short C programs (using Turbo C under DOS) to do simple file operations on the network drive (open, read, cd, etc.) and looked at the packets that the server and client exchanged. From this I worked out what some of the bytes in the packets meant, and started to write my own program to do the same thing on a Sun.

After a day or so more I had my first successes and actually managed to get a connection and read a file. From there it was all downhill, and a week later I was happily (if a little unreliably) mounting disk space from a Sun to my PC running Pathworks. The server code had a lot of “magic” values in it, which seemed to be always present with the Ultrix server. It was not until two years later that I found out what all these values meant.

I thought other people might be interested in what

I had done, so I asked a few people at uni, and no one seemed much interested. I also spoke to a person at Digital in Canberra (the person who had organised a beta test of *eXcursion*) and asked if I could distribute what I'd done, or if it was illegal. It was then that I first heard the word “netbios”, when he told me that he thought it was all covered by a spec of some sort (the netbios spec), and thus what I'd done was not only legal, but silly.

I found the netbios spec after asking around a bit (the RFC1001 and RFC1002 specs) and found that they looked nothing like what I'd written, so I thought maybe the Digital person was mistaken. I didn't realise that the RFCs referred to the name negotiation and packet encapsulation over TCP/IP, and what I'd written was really an SMB implementation.

Anyway, he encouraged me to release it, so I put out “Server 0.1” in January 1992. I got quite a good response from people wanting to use Pathworks with non-Digital Unix workstations, and I soon fixed a few bugs, and released “Server 0.5”, closely followed by “Server 1.0”.

All three releases came out within about a month of each other.

At this point I got an X-terminal on my desk, no longer needed *eXcursion*, and promptly forgot about the whole project, apart from a few people who e-mailed me occasionally about it.

A year passed with just occasional e-mail asking about new versions and bugs. I even added a note to the ftp site asking for a volunteer to take over the code as I no longer used it. No one volunteered.

During this time I did hear from a couple of people who said it should be possible to use my code with LanManager, but I never got any definite confirmation.

One e-mail message I got about the code did, however, make an impression. It was from Dan Shearer at the University of South Australia, and he said this:

I heard a hint about a free Pathworks server for Unix in the Net channel of the Linux list. After quite a bit of chasing (and lots of interested followups from other Linux people) I got hold of a release news article from you, posted in Jan 92, from someone in the UK.

Can you tell me what the latest status is? I think you might suddenly find a whole lot of interested hackers in the Linux world at least, which is a place where things tend to happen fast (and even some reliable code gets written, BION!).

I asked him what Linux was, and he told me it was a free Unix for PCs. This was in November 1992, and a few months later I was a Linux convert! I still didn't need a Pathworks server, though, so I didn't do the port, but I think Dan did.

At about this time I got e-mail from Digital, from a person working on the DEC-Alpha software distribution. He asked if I would mind if they included my server with the “contributed” CD-ROM. This was a bit of a shock to me, as I never expected DEC to ask me if they could use my code! I wrote back saying it was OK, but never heard from him again. I don't know if it went on the CD-ROM.

Anyway, the next big event was in December 1993, when Dan again sent me e-mail saying my server had “raised its ugly head” on **comp.protocols.tcpip.ibmpc**. I had a quick look on the group, and was surprised to see that there were people interested in this thing.

At this time a person from our computer center offered me a couple of cheap ethernet cards (3c505s for $15 each) and coincidentally someone announced on one of the Linux channels that he had written a 3c505 driver for Linux. I bought the cards, hacked the driver a little, and setup a home network between my wife's PC and my Linux box. I then needed some way to connect the two, and I didn't own PC-NFS at home, so I thought maybe my server could be useful. On the newsgroup, among the discussions of my server, someone had mentioned that there was a free client that might work with my server that

Microsoft had put up for ftp. I downloaded it and found to my surprise that it worked first time with my Pathworks server!

Well, I then did a bit of hacking, asked around a bit and found (I think from Dan) that the spec I needed was for the “SMB” protocol, and that it was available via ftp. I grabbed it and started removing all those ugly constants from the code, now that all was explained. It was a shock seeing the real spec for SMB, and it made me realise how lucky I was that my original code worked at all.

<u>Samba Resources</u>

On December 1, 1993, I announced the start of the “Netbios for Unix” project, seeding the mailing list with all the people who had e-mailed me over the years asking about the server.

At this stage I called the package smb-server. This changed quickly one weekend when I got e-mail from a company that makes a commercial Unix SMB-based server. Apparently they have trademarked that name. I needed a new name quickly, and Samba was born.

This list has now grown to over 600 people and a newsgroup (**comp.protocols.smb**) has just started, primarily because of peoples' interest in Samba. I get approximately 100 connections to the Samba ftp site each day, and dozens of dedicated hackers have contributed to the code. Samba is now being used as a production PC file server at many sites worldwide.

Almost all of the development for Samba was done on my home Linux box. Linux has been a fantastic development plaform. Without Linux, Samba would certainly not be where it is today.

## What Can It Do?

Now that I've got that off my chest, I better tell you what Samba can do. Not that I expect anyone to still be reading after a tirade like that one.

Samba provides file and print services to SMB clients. These include LanManager, Windows for Workgroups, Windows NT and OS/2. There is also a free client for DOS put out by Microsoft, but it's a real memory hog.

Samba also provides a Netbios name server, so PCs can find the server, and a Unix SMB client program. The SMB client only has a primitive ftp-like interface, but a proper mountable SMB filesystem for Linux is in the works.

Samba uses quite a comprehensive configuration file mechanism written by Karl Auer. Karl also did all the documentation for Samba, which I think has been very important in its success.

Some features of the Samba server are:

- freely distributible source under GPL
- supports more than 20 flavours of Unix
- easy configuration
- supports mangled filenames with root name preservation
- much faster than NFS
- much more secure than NFS
- clients are pre-installed on many platforms
- most clients have auto-reconnect
- restrict access by username/password, by IP address or netgroup

There are a lot more bits and pieces. Samba has “suffered” from Karl's code that allows me to easily add new options. There are now more than 60 configurable options in the server, which can be applied in endless combinations for each exported file or print service. Thank god for Karl's man pages.

Samba is being improved all the time. It is now a distributed development effort with many active contributors. Upcoming versions are likely to include full long filename support for those clients that can handle it (such as Windows NT and Chicago), browsing support and a mountable SMB filesystem. Work is also proceeding on a more complete RFC1001/1002 netbios nameserver implementation.

Get it, use it. If it doesn't work for you, then remember how much it cost. Also remember to send me a bug report.

Now I think I'll go and have some lunch.

**Andrew Tridgell** is an associate lecturer in the department of computer science at the Australian National University in Canberra, Australia. He is also completing a PhD in automatic speech recognition in the computer sciences laboratory at the same university.

Archive Index Issue Table of Contents

Advanced search

# Linux Performance Tuning for the Faint of Heart

**Clarence Smith, Jr.**

Issue #7, November 1994

*Recompiling your Linux kernel might not be as scary as you think. Clarence Smith gives us a good step-by-step process for building your own customized kernel.*

Have you ever wondered why in the blazes someone would want to *recompile* a Linux system kernel? For some, that is a challenging question. Many new users to Linux are under the impression that when they install Linux, the installation itself is perfect. Don't assume that Linux is set up efficiently for your exact setup out of the box. Here's why (and how) to change it.

Let's say you have bought a new laptop, and you want to run Linux on it. You've got only 4MB of RAM, and you've got only 100MB of hard drive space. A finely tuned Linux kernel is key to having all possible memory space available on your system, and thus is the key to having a faster system for a small-time investment.

You are probably aware that Linux provides “virtual memory” (also known as swap space or paging space) that allows programs to use more memory than is really on the system by temporarily moving the contents of some of that memory to disk. You may not be aware, however, that no parts of the kernel can be paged out to disk. Every byte taken up by the kernel is a byte that can't be used by *anything* else.

If the kernel you are running has SCSI support, networking, and sound all compiled in, but you don't have or use SCSI, networking, or a sound card, you are wasting memory. This is probably the case if your kernel came with your Linux distribution, because those kernels are generally compiled to work with a very wide selection of devices, and are not tuned to your hardware.

If you only have 4MB of memory, you are also likely wasting a lot of time as other programs get swapped out to disk and then back into memory. By compiling a new kernel without the unnecessary pieces, you can make your Linux computer run a lot faster.

Fortunately, Linux makes this easy.

## Think First

First of all, you need to consider the hardware you have. What types of peripherals do you have? What type of mouse do you have? Do you have a sound card?

To build and compile the best kernel for your system, you must be aware of your hardware make-up. You may find it helpful to sit down and list all the parts of your computer. Not only will this help you now, but it will also help you post good problem reports if you encounter a problem or bug, since you need the same information about your configuration when posting problem reports.

To configure your system, you first need to have the Linux source code installed on your system. This is available via **ftp** from **nic.funet.fi** in **/pub/OS/Linux/PEOPLE/Linus**, **tsx-11.mit.edu** in the directory **/pub/linux/sources/system**, or **sunsite.unc.edu** in **/pub/Linux/kernel**, or other mirrors around the world. The directory name will depend on the kernel version: Linux versions 1.1.x are kept in the **v1.1** directory, and have names like **linux-1.1.45.tar.gz**, with patch named like **patch46.gz**. You only need to get patches with numbers later than the version of the tar file you get.

Alternately, if you bought Linux from someone, they are required by Linux's GPL copyright to have provided the source, or to provide it (possibly for a nominal fee) upon request.

The source code should be unpacked into **/usr/src/linux**, because it expects to be there. See the sidebar “Unpacking the Linux Source” if you do not know how to do this.

After unpacking the source, you should apply any **kernel patches** that are needed to get the version you want. You should insert your patches (while they are still compressed, or gzipped) into **/usr/src/linux**. Then from **/usr/src/linux**, type:

**gunzip -c patch?.gz | patch -s -p1gunzip -c patch??.gz | patch -s -p1**

The **?** gets patches 1-9 **in order**, and should be done only if you are using one of those patches. The **??** will get all patches (that exist, and that you have downloaded) between 10 and 99 **in order**. It is important that all the patches be applied **in order**. The **-s** argument to **patch** tells **patch** to work silently, and only complain about errors. The **-p1** tells **patch** that we are in the **/usr/src/linux** directory, so that it knows how to find files, and puts new files in the right place. The **-s** flag is optional; the **-p1** argument is mandatory.

Alternately, you can run each patch separately:

```
gunzip -c patch8.gz | patch -p1gunzip -c patch9.gz | patch -p1...gunzip -c
patch46.gz | patch -p1
```

and so on.

Once you've completed the patches, you may want to remove any unnecessary files created by the patches. These files are the original versions of the files that are changed in any way, so they can take up a substantial amount of space. You can find and remove them by typing:

```
find /usr/src/linux -name '*.orig' -o -name '*~'-exec rm -f {} ;
```

You can look for any files that did not patch correctly, and thus find out in advance that there has been a problem that will likely make it impossible to compile your kernel, by typing:

```
find /usr/src/linux -name '*.rej' -print
```

That will list any files for which there were “rejected hunks”; patches that could not be fitted into the source correctly. If you find any of these files, start over. If you still see these files, ask someone for help; there are too many things that could be wrong to cover in this article.

## Configuring the Kernel

Now, from within **/usr/src/linux**, type **make config**. You are now prompted to answer many questions about your system. If you say you don't have hardware that you do have, that particular hardware will not be supported by the new kernel. Likewise, if you say you have hardware that you don't have, you will waste memory. Also, you want to enable only the software features that you are going to use.

I will attempt to point out some of the most significant configuration questions. One of the first questions is:

```
'Kernel math emulation' CONFIG_MATH_EMULATION ?
```

If you don't have a math co-processor, you should answer this question YES. Those who do have a math co-processor chould answer NO. A kernel with math-emulation compiled in it will still use the processor if one is present. The math-emulation simply won't get used in that instance. This will result however in a somewhat larger kernel and a waste of memory.

There are a couple of questions about hard disk support. One in particular can cause a bit of confusion:

**'XT harddisk support' CONFIG_BLK_DEV_XD**

I answer this question NO. This really doesn't have to do with hard disks—it has to do with your **controller card**. **AT** controller cards are 16-bit cards supported by the standard hard disk driver. **XT** controller cards are 8-bit cards that are *very* rare in 386-class machines.

**'TCP/IP networking' CONFIG_INET :**

Answer YES if you plan to have your system interactive on the net. This includes SLIP and PPP connections. Answer NO if you aren't going to connect to the net right now; you can always compile another kernel later.

**'System V IPC' CONFIG_SYSVIPC :**

This isn't used for many things, but doesn't use much memory. YES is recommended.

**'Use -m486 flag for 486-specific optimizations' CONFIG_M486 :**

If you have an i386 system, answer NO. Otherwise you should select YES. This uses a little bit of memory. Adding this flag will not slow a 386 down, other than using extra memory, but will speed up a 486 quite a bit.

## Types of Devices

There are a series of questions which have to do with different types of SCSI drivers and interfaces. If you have a SCSI controller, then you would want to enable the drivers via the configuration process. For those who don't have a SCSI controller, select NO, and move on to the next step in the configuration. If you don't select SCSI support, you won't be asked whether or not to include SCSI devices.

Network device support mainly has to do with selecting the proper Ethernet device or other network connection. PPP and SLIP are used to connect to TCP/IP networks over the serial port; PLIP is used to connect TCP/IP networks over the parallel port, and the rest are ethernet controllers. **Do not** select drivers of devices you do not have. This can sometimes cause conflict later when you are booting.

Another important section in the kernel config process has to do with the different filesystems. There is an advantage to compiling the kernel to have only the filesystems you need. There are several different filesystems supported by Linux:

**'Standard (minix) fs support' CONFIG_MINIX_FS :**

This is the *original* Linux filesystem. It is considered to be one of the more stable filesystems, and is still widely used. You probably want this unless you are really desperate for space or will really never use it.

**'Extended fs support' CONFIG_EXT_FS :**

Only select this if you still have filesystems from the 'old days' that will use this precursor of the second extended filesystem. This filesystem is slow and no longer actively maintained. It is there only for backwards compatibility. NO.

**'Second extended fs support' CONFIG_EXT2_FS :**

The ext2 filesystem is the most 'full-featured' of the filesystems. It is a super rewrite of the original extended filesystem, with improved speed as well. This is by far the most popular filesystem. A filesystem debugging package is available for this filesystem that can help you recover from fairly bad filesystem crashes. YES.

**'xiafs filesystem support' CONFIG_XIA_FS :**

This is a modification of the Minix filesystem (allowing for such things as longer filenames). If you plan to use it (YES), otherwise (NO).

**'msdos fs support' CONFIG_MSDOS_FS :**

This filesystem allows for access to the FAT filesystem used by MS-DOS. It is a great advantage for those who need to access floppies or hard disk partitions that use that filesystem. In addition, if you use the UMSDOS filesystem (usually because you installed Linux on an MSDOS partition), you need to include MSDOS filesystem support.

**'/proc filesystem support' CONFIG_PROC_FS :**

The PROC filesystem does not touch your disk. It is used by the kernel to provide data kept within the kernel itself to system programs that need that information. Many standard utilities will not function without this filesystem. YES.

### 'NFS filesystem support' CONFIG_NFS_FS :

The NFS filesystem is necessary for those who have a physical network and need to mount NFS filesystems from other computers. If you are on a TCP/IP network, you probably want this option. Otherwise, you don't.

## Possible Confusion

### 'Kernel profiling support' CONFIG_PROFILE :

This is used only by seasoned kernel hackers. You don't need this if you need to read this article...

### 'Selection (cut and paste for virtual consoles)' :

Self explanatory. If you want to be able to use your mouse in any of your VC's, then you would select YES. Note that this requires a separate **selection** program to work. Your mouse will be inactive without this program. This has nothing to do with X-windows.

## Sound Tricks

The last section of the configuration process that needs insight has to do with sound card options. If you want sound card support, then you would definitely select YES. The confusing part are the next two questions; confusing because of the order they fall in.

**Full driver?** NO. I select NO because I want to configure my kernel to handle only my particular sound card. I don't want to enable drivers for cards I don't have. This simply wastes space.

Immediately following is: **Disable?** NO. It seemed strange to me that the driver question would come *before* a question about disabling the sound drivers altogether. Nonetheless, you should select NO, and answer the questions that follow accordingly, depending on what hardware you have.

That was the last part of the configuration process. All that remains is to make the system dependencies. After that, the actual kernel compile!

## Making System Dependencies

There isn't any confusing part about making the dependencies. All you do is type **make depend** (in the **/usr/src/linux** directory, which you should already be in); this sets up the system dependencies that all the files have. This allows **make** to intelligently recompile only the right files if you have to make changes to the source code.

## Compiling the Kernel

After the system dependencies have been created, you are ready to compile the newly configured kernel. At this point, you should type **make zImage** to create a *compressed* kernel. This helps in the process of keeping your kernel small. Depending on the speed of your machine, the amount of memory you have, and how many things you are compiling into your kernel, your compile could take anywhere from around 15 minutes on a fast 486 or Pentium to several hours on a slow 386sx with 4MB of memory.

When the compiling process is complete, you will find a newly created kernel, **zImage**, in the **/usr/src/linux** directory. The process of installing the new kernel however is not yet complete. If you reboot right now, even though you have compiled a new kernel, you will still boot your old kernel. You still need to *install* the new kernel.

## Installing the New Kernel

Before you install the new kernel, you should rename your old kernel, so that you can use it in the case of an emergency. In the **/** directory, there should be an image of your old kernel (**zImage**). Simply rename it to **zImage.old** using the **mv** command. This will be useful in the event that your new kernel will not boot up; you at least have a backup that still allows for a functional system.

Now, you must edit (if necessary) the LILO configuration file, so that it will accept your new kernel. It is most often found in the file **/etc/lilo/config**, but may be found as **/etc/lilo.conf**. Basically, editing the configuration file and running **lilo** tells the LILO which kernel to use upon bootup. The first image to be set up in the config file will be the default. By putting the new kernel image first, we will ensure that the new kernel boots by default; if we were to put the DOS entry first, DOS would boot by default. Here's an example:

**boot = /dev/hda**

#This is the path to the NEW kernel image to be bootedimage = /zImagelabel = linux

#This is the path to the OLD kernel image to be booted#if the other should fail to bootimage = /zImage.oldlabel = linux.old

#This is the path to the DOS operating systemother = /dev/hda1**table = /dev/hdalabel = dos**

Once you've edited the configuration file, copy or move the new kernel image to the **/** directory. Make sure that **/zImage** and **/zImage.old** are present before going any further. Next, you need to go back into the **/etc/lilo** directory and type **./lilo** to run **lilo**. Alternately (if **/etc/lilo** doesn't exit), you may need to **cd** to **/etc** and run **/sbin/lilo**. In either case, this is the “installation” step which makes it possible for you to boot your new kernel.

You are now ready to boot your new kernel. To do this, type **shutdown -r now**.

If you have not installed your new kernel as the default kernel by making it the first entry in **/etc/lilo/config**, you will need to manually select your new kernel as you boot up. You activate the LILO menu by pressing <shift>, <control>, or <alt keys> as LILO is starting, or by pressing the <CapsLock> or <ScrollLock> key before LILO starts.

At the LILO prompt, make sure you type the **label** of the image, and not the filename of the image. If LILO can't find the image you ask for, it will tell you. Typing **?** will give you a list of labels to choose from.

During the boot process, be sure to look at the information that relates the version of the kernel being booted, and the day that it was compiled (it's all on the same line). Mine looks like this:

**Linux version 1.0.9 (root@HoMiEz.ShOpPiN.NeT) Thu Jul 14 22:45:16 1994**

This particular line tells which version of the kernel is being booted, and the date and time that it was *last* compiled. The information shown here should coincide with the compile process you just recently completed. If there is an inconsistency, be sure to check your **/etc/lilo/config** file. Make sure that you put the proper label name for the particular table on the hard drive.

Once the new kernel is booted, you are free to hack away. Kernel compiling can be a confusing process. Some new administrators are intimidated by compiling a new kernel, but it is a prerequisite to running an optimized system. After a few kernel upgrades, you can become skilled in maintaining your system optimally. Then you can pass that knowledge to those who are as unsure as you once were. Remember, the more you know about your own system, the more optimized **you** become in using it.

Unpacking the Linux Source

Before you unpack the kernel source, it is a good idea first to backup you *old* kernel. You should do this, just for the moment, until you completely get the new kernel functioning properly. You can even shrink the size of the old kernel source by using **tar** and then compressing it. It is better to use both, as a double strength method:

```
cd /usr/srctar cvf linux-old.tar linuxgzip -9 linux-old.tar
```

The resulting file should be **linux-old.tar.gz**. After you've done this, it is safe to remove the old source directory by typing:

```
rm -rf linux
```

After you've done this, you can uncompress the new kernel. You must first move the new kernel source to **/usr/src**. Then type the following:

```
gunzip -c new-kernel-name.gz | tar xvof -
```

Once this process is complete, you are ready to move on. I'd also suggest keeping the new compressed kernel source in a backup directory. This way, if an error occurs in the process of upgrading and compiling your new kernel, you always have un-patched, “virgin” source to start over with. This eliminates the time it takes to download the kernel source again and also leaves you more time to concentrate on the task at hand.

## Kernel Resources

The kernel sources are available at a large number of Linux **ftp** sites, including **sunsite.unc.edu** in **/pub/Linux/kernel**, **tsx-11.mit.edu** in **/pub/linux/sources/system**, and **nic.funet.fi** in **/pub/OS/Linux/kernel/src**. Kernel sources stored at these sites are typically gzipped tar files with filenames of **linux-*version*.tar.gz**.

If you got Linux from a distributor, they may have already given you the kernel source. If they haven't, they are required by the GNU General Public License to make it available to you.

**Clarence Smith, Jr.** is a student at the University of Washington, working on a Public Relations and Sociology degree. He hopes to develop an increased knowledge of the Linux Operating System by the development of some useful software tools. Clarence lives by the hacker ethic of striving for perfection, based on a beutiful line of code and functionality.

# Selecting Hardware for a Linux System

**Phil Hughes**

Issue #7, November 1994

In this article Phil Hughes describes the basics of choosing a hardware platform for Linux.

Although Linux software is virtually free, the required hardware isn't. This makes some people hesitate to jump into the Linux movement. But it doesn't need to be complicated, scary or expensive. Linux runs on most common hardware. In this article I will point out how to make the right choices and hopefully keep you away from costly mistakes. If you currently have a system that adequately runs MS-Windows, you probably have all you need for a decent Linux system. Although Linux is more sophisticated than MS-Windows, it doesn't require more hardware. In fact, because Linux is a pre-emptive multi-tasking system (this means that programs can be interrupted to service other requests), you can get a lot more useful work from the same computer. Before I get into specifics, I would like to point out that there is a lot of documentation on Linux. This includes HOWTOs which detail many of the specifics. These HOWTOs are available at Linux archive sites on the Internet, on most CD-ROM Linux distributions and on paper from various sources. A short article is no substitute for the details presented in the HOWTOs. I highly recommend that you get a copy and that you read them.

## CPU and RAM

To run Linux you need a 386 or higher processor. For text-based applications even a slow 386SX system will perform very well. Although the Linux kernel is capable of emulating floating point arithmetic functions, it is significantly slower than a math co-processor; either in the form of a separate chip or a 486DX or better that has a built-in co-processor. The server system at the *Linux Journal* advertising/editorial offices is a 386DX40. It has proved to be a great performer with three high-speed modem lines connected to it, a local user and 2 or more active telnet sessions. It has 8MB of RAM but an upgrade to 16MB is planned. The system we use for running X-windows is a 486DX33 with 16MB of RAM. The built-in co-processor and additional RAM make this an excellent workstation for compute-intensive work. Linux needs a system that is based on the ISA (AT) or EISA bus. Linux does not support MicroChannel Architecture (MCA) machines such as the IBM PS/2. It does, however, support local bus systems such as VESA and PCI. What you need will depend on what you intend to do with the system. The minimum RAM is 4MB. If you currently have a 386SX with 4MB of RAM, give it a try. It may do all that you want with no investment. If, however, you need to purchase hardware to run Linux, look at higher performance systems such as something based on a 486DX chip and at least 8MB of RAM. The additional cost is not very great for a large performance increase.

### Disks

There are two considerations: type and size. Linux supports MFM, RLL, ESDI and IDE disks with virtually any controller board. It also supports a fairly wide assortment of SCSI controllers. If you already have a system, you can probably stick with what you have. But, if you are purchasing new hardware, SCSI disks are well worth considering. Here's why: Many of the supported SCSI disk controllers use direct memory access (DMA) to transfer information. Under MS-DOS this has little advantage because MS-DOS (and MS-Windows) waits for input/output operations. On Linux, however, processing is overlapped with I/O. This means that a DMA controller can be transferring data while the CPU is processing other tasks. One of the most popular controllers supported by Linux is the Adaptec AH-1542C. It is relatively inexpensive ($200 range), uses DMA and has reasonable performance. For EISA bus systems, the AH-1742 may be used. If your system has a VESA local bus, the Buslogic 445S controller seems to offer the best performance at a reasonable cost. Other supported SCSI controllers include the Adaptec 152x series, Allways IN2000, Adaptec 1542 clones (including Buslogic 445S and 447S and DTC 3290 and 3292), Seagate ST01 and ST02, Western Digital 7000, Trantor T128 and T130B, Ultrastor 14F and 24F. The SCSI HOWTO has many details that will help you weigh the costs and performance of the various SCSI controllers. Another consideration when choosing between IDE and SCSI disks is that you can have two disks with IDE controllers and with SCSI you can have up to seven. Selecting the disk size requires some forethought about what applications you'll be running. The price of disk storage is dropping, and you can add one or more disks later on (depending upon your disk type). Linux can run on as small a partition as 20MB, but 200MB is a common size. Some applications may require disk sizes of 1GB or more.

### CD-ROM

If you have a SCSI disk controller, a SCSI-based CD-ROM drive makes the most sense. It plugs into the same controller and works fine. With the advent of double and triple-speed CD-ROM drives, prices have dropped significantly on the single-speed drives which are adequate if all you intend to do is load files from a distribution CD-ROM. And prices for these devices on the surplus market seem to be in the $50-$80 range. If you are not using SCSI disks, using a non-SCSI CD-ROM is more cost-effective. Although others are supported, drives made by Mitsumi (and marketed under many names including BSR and Tandy) work well. They include their own controller card and can usually be found in the $150 price range.

## Printers

There is nothing special about Linux and printers. Standard parallel port connections are supported as well as serial port connections for printers. If you have a PostScript laser printer there are programs included with most Linux distributions that support these devices directly. If you don't have a PostScript printer, ghostscript, a program that comes with most Linux distributions, will translate PostScript into the necessary control codes for most printers. If you choose this options, be aware that ghostscript's default fonts are not as pretty as the fonts that come with a PostScript printer.

## Serial Communications

Serial devices which include mice, modems and terminals can be connected to the system. Because Linux is multi-user as well as multi-tasking, multiple people can be using the computer system at one time. Most MS-DOS systems come with two serial ports. Linux can use these ports but you may want additional ports to support multiple devices. Serial communications boards come in various flavors. The standard 2-port board that comes in most systems uses 16450 UARTs (Universal Asynchronous Receiver Transmitters). This UART has no buffering which means the CPU must stop what it is doing and grab each input character. For one line running at 38,400 bits per second, this means that the processor could have to get a character about every .00025 seconds. With multiple ports this could consume most of the CPU time and result in dropped characters. The 16550A UART is very similar to the 16450 except it has a built-in 16 character buffer. This means that the processor could fetch characters 1/16th as often and still get all the input data. The difference in price is not that big between boards based on the 16450 and 16550A so it is well worth considering. Also, it is possible to replace the 16450s with 16550As in boards where the UARTS are in sockets. If you are considering adding ports, there are various 4, 8 and 16-port boards that use 16550As and support interrupt sharing. This means that up to 16 ports could be configured all on one board using one interrupt line on the bus. There are other serial communications boards that Linux supports or will soon support. Intelligent serial boards are boards that contain an on-board CPU. This CPU handles specifics of the serial transfer, freeing up more of the main CPU. They vary from boards with “intelligent UARTS” to those that include a general- purpose CPU and DMA I/O. Many manufacturers including Spectrix, Stallion, Computone, Arnet and DigiBoard make these boards. They require special drivers and, at this time, the only one supported is the Cyclades 8Y, which uses Cirrus Logic RISC chips. Performance-wise, these boards should offer better performance than the 16550A-based boards. An alternative to serial communications boards is a terminal server. This is a device that connects to the host system via Ethernet and handles serial communications itself. Any terminal server that supports the TCP/IP protocol should work with a Linux system.

Getting HOWTOs via FTP

## Networking

Linux has built-in support for networking including TCP/IP and NFS. If you already have a network that supports TCP/IP protocol (used by Unix and other systems) you should be able to add your Linux box to the network. The one piece of hardware you will need is an Ethernet adapter. Ethernet comes in three flavors: thick-net, thin-net and UTP. Thick-net is thick coaxial cable where you connect to it via a tap and a transceiver with an AUI interface. Thick-net was the original but is no longer very popular. Most common today is thin-net, with UTP running second. Thin-net consists of machines connected on a single line of RG-58 coaxial cable. Each machine is connected using a BNC T-connector. Each end of the line is terminated with a 50-ohm terminator. Adding a new computer means either adding to the end of the line and moving the terminator along or splitting an existing cable and adding a new T. UTP (unshielded twisted pair) uses a different configuration. You have a hub and then the systems fan out from the hub. They are also called 10baseT where the T stands for twisted pair. Generally, thin-net is the least expensive, most practical approach. There are a large number of Ethernet boards out there. If you are looking for the quick answer; reasonable cost, reliability and reasonable performance, the WD8013 or SMC8013 card is a good choice. Other cards that generally work are NE2000 clones and the 3Com 3c503. Again, the Ethernet-HOWTO contains much more helpful information than I can include here.

## Video

Character-based Linux applications will work with any video board available for the PC. If, however, you want to run X-windows, you will need to select a supported video board. Although there is nothing difficult about supporting any boards, some manufacturers have refused to make the specifications available; therefore, Linux drivers could not be written for their hardware. One major vendor who has not released specifications (and is therefore not supported) is Diamond. The most cost-effective accelerated video boards supported by Linux are the low-end S3 boards. STB and Orchid are two vendors who have cards available for about $130-$150 that will be adequate for almost all Linux users. High-end S3 and ATI boards which are supported are available in the $350-$500, and will increase your video performance. You will want to check the XFree86-HOWTO for more current details when you are making your decision.

Low-End Configuration and High-Performance Systems

## Conclusion

Shopping for Linux hardware isn't harder than shopping for DOS/Windows hardware. If you are comfortable selecting video boards and talking about RAM chips, you should have no problem selecting what you need. On the other hand, if you don't want to know what a SCSI is, you may be better off to let someone else pick the hardware for you. Much like picking DOS/Windows hardware, define what you want to do; what sort of things you will be running on the computer system. Then give your requirements to a Linux-knowledgeable hardware vendor and see what they have to offer. Whether you do your own hardware selection or just get some ideas from this article and let someone else pick, expect the end result to be a real Unix-like com-puter system; for your work, for fun or for both.

Phil Hughes is the publisher of *Linux Journal*, and has put together a few Linux systems in his time.

**Phil Hughes** is the publisher of *Linux Journal*, and has put together a few Linux systems in his time.

# CD-ROM and Linux

**Jeff Tranter**

Issue #7, November 1994

A CD-ROM drive is one of the most popular hardware upgrades for personal computers and is becoming a standard peripheral for new systems. In the article, Jeff looks at support for CD-ROM under Linux.

CD-ROM stands for Compact Disc Read-Only Memory, a storage medium utilizing an optical laser to read microscopic pits on the aluminized layer of a polycarbonate disc. The same format is used for audio compact discs.

The storage capacity of a CD-ROM is approximately 650MB, equivalent to over 500 high density 3.5" floppy disks or roughly 250,000 typed pages.

First-generation drives (known as single speed), provide a transfer rate of approximately 150KB (kilobytes) per second. Double-speed drives are commonly available, and triple- and quad-speed drives have recently been introduced.

Most CD-ROM drives use either the Small Computer Systems Interface (SCSI) or a vendor-proprietary interface (which is often provided on a sound card). They also typically support playing audio CDs via an external headphone jack, line-level output, or speakers. A third type of CD-ROM interface, based on an extension of the IDE hard disk interface, has also recently become available.

CD-ROMs are usually formatted with an ISO-9660 (formerly called High Sierra) file system. This format restricts filenames to the MS-DOS style (8+3 characters). The Rock Ridge Extensions use undefined fields in the ISO-9660 standard to support longer filenames and additional Unix-style information (e.g., file ownership, device files, symbolic links, etc.).

Photo-CD is a standard developed by Kodak for storing photographic images as digital data on a CD-ROM. Photographic film can be transferred to a Photo-CD and, with appropriate software, you can view the images on a computer, perform further image processing, or send them to a printer. Up to 100 photographs can be stored on a CD with an image quality that is typically much higher than can be obtained using other methods, such as scanners.

CD recorders have recently become available. They use a different media and specialized equipment for recording, but the resulting disc can be read by any CD-ROM drive. (This is the same “write once” technology used for Photo-CD.)

## The Advantages of CD-ROM

The primary advantages of CD-ROM over other mass- storage media are its high storage capacity, high reliability, and low cost.

The drawbacks are that it is read-only, slower-than-hard-disk media, and the discs can be damaged if mishandled.

Linux provides good support for CD-ROM. The dynamic buffer cache used for the hard disk subsystem is also used for CD-ROM access, improving performance. Depending on the type, multiple drives can be supported. (The Panasonic driver, for example, supports up to 16 drives.)

Linux fully supports the Rock Ridge Extensions to the ISO-9660 file system, making all of the features of the hard disk file systems available, including long filenames, file permissions, links, and device files. PhotoCD is also supported by some of the CD-ROM drivers.

Many vendors (I know of at least ten) are now offering CDs of Linux source code, binaries, and documentation at reasonable prices. Many of these feature easy to use menu-driven installation programs. A single CD-ROM can hold a full Linux distribution, as well as all of the files from the two major Internet archive sites, with room to spare. By comparison, a recent Linux distribution can fill as many as 50-3.5 inch floppies.

Finally, most CD-ROM drives support playing audio CDs, so you can listen to music while waiting for the latest Linux kernel to compile.

## Supported Hardware

Linux supports virtually all SCSI CD-ROM drives, provided that a supported SCSI host adaptor is used.

Many of the popular proprietary drives are supported, including models produced by Sony, Mitsumi, and Panasonic/Matsushita.

CD-ROMs based on the enhanced IDE standard are not yet officially supported by Linux, although at the time of this writing, at least one driver is in alpha testing.

By the time you read this, more devices will likely be supported, either as part of the standard Linux kernel or as patches. See the CD-ROM HOWTO document for a detailed list of the latest supported hardware.

## Configuring Linux for CD-ROM

Setting up Linux to use a CD-ROM involves four steps:

1. Installing the hardware
2. Configuring the Linux kernel
3. Creating the necessary device files
4. Mounting the media

I will cover them briefly here; see the Linux CD-ROM HOWTO for more details.

The first step, installation, is dependent on the type of drive. Follow the instructions provided by the manufacturer or have the installation performed by your dealer. There are no special installation requirements for Linux.

Next, the Linux kernel must be configured. In some cases, you may be able to use a pre-compiled kernel that has the necessary drivers, but I recommend compiling it yourself; it will do you good! For SCSI drives you need to configure in SCSI CD-ROM support and the driver for the SCSI host adaptor being used.

For the proprietary CD-ROM interfaces, select the appropriate driver (e.g., Sony CDU31A).

In order to mount CD-ROMs, you must also configure in support for the ISO-9660 file system. If you have a sound card, now would be a good time to configure the kernel sound driver as well.

The third step is to create the appropriate device files. If you are running a standard Linux distribution you may have already done this during system installation. It's a good idea to verify these; the CD-ROM HOWTO lists the device file types, which are drive dependent.

You should now be ready to compile and boot the newly configured kernel. Verify that the CD-ROM was detected by looking at the kernel boot messages; here is the output on my system:

```
SBPCD version 2.5 Eberhard Moenkeberg <emoenke@gwdg.de>
 SBPCD: Looking for a SoundBlaster/Matsushita CD-ROM drive
 SBPCD: Trying to detect a SoundBlaster CD-ROM drive at 0x230.
 SBPCD: - Drive 0: CR-562-x (0.76)
 SBPCD: 1 SoundBlaster CD-ROM drive(s) at 0x0230.
 SBPCD: init done.
```

## Using a CD-ROM

To mount a CD-ROM, insert it in the drive and use the mount command (as root). A typical command line is the following:

```
% mount -t iso9660 -r /dev/cdrom /mnt
```

The example above assumes that the CD-ROM device file is **/dev/cdrom** and the disc is ISO-9660 formatted (this is almost always the case). The **-r** option indicates that the disc is to be mounted read-only. If successful, the CD can now be accessed under the directory **/mnt**.

For a more permanent setup, you may wish to mount the CD under a more meaningful name such as **/cdrom**. By adding an entry to the **/etc/fstab** file you can have a CD-ROM automatically mounted when Linux boots; see the fstab(5) man page for details.

When finished with the CD, it can be unmounted using the **umount** command (again, run this as root):

```
% umount /mnt
```

If you want to allow non-root users to mount and unmount CD-ROMs, you can use the “user” option provided by some mount commands. If you make an entry such as the following in **/etc/fstab**:

```
/dev/sbpcd  /cdrom   iso9660
user,noauto,ro
```

then an ordinary user will be permitted to mount and unmount the drive using these commands:

```
% mount /cdrom
% umount /cdrom
```

The disc will be mounted with some options that help ensure security (e.g., programs on the CD cannot be executed and device files are ignored). Another method is to obtain or write a program such as usermount which runs setuid to root and allows restricted mounting of specific devices (e.g., CD-ROM and floppies) for non-root users.

## PhotoCD

PhotoCDs use an ISO-9660 file system to store image files in a proprietary format, at several different resolutions. Not all CD-ROM drives support reading PhotoCDs. If yours does, you can mount it and use a program such as **hpcdtoppm** to convert the files to a format that can be displayed using graphics file viewers such as **xloadimage** or **xv**.

The **hpcdtoppm** program is part of the PBM (portable bit map) utilities, available on many Internet archive sites (look for **pbm** or **netpbm**).

The program **xpcd** is an X11-based utility for manipulating PhotoCD images. You can select the images with a mouse, preview them in a small window, and load the image with any of the five possible resolutions. You can also mark a part of the image and load only the selected part. This program can be found at **ftp.cs.tu-berlin.de** in the file **/pub/linux/Local/misc/ xpcd-0.2.tar.gz**.

## Playing Audio CDs

Several programs are available that allow playing audio CDs, either through a headphone jack or an attached sound card. workman, supplied with many Linux distributions, is one such program. It sports a graphical user interface that resembles the controls provided on audio CD players. Simple command-line CD player programs also exist. Note that to play an audio CD you should not try to mount it.

The CD player programs simply route the analog output of the drive to an external device. Some CD-ROM drives also support reading the digital sound data contained on audio CDs. Using a program such as cdda2wav you can save audio tracks from a CD-ROM as a sound file (e.g., in .wav format).

## Inheriting File System

The Inheriting File System (IFS) is a kernel driver that allows mounting multiple file systems at the same point. By mounting a hard disk directory over a CD-ROM file system, you can effectively obtain a writable CD-ROM file system.

At the time of this writing, an experimental version of IFS, written by Werner Almesberger for the 0.99 Linux kernel, is available as a kernel patch.

## Creating CD-ROMs

If you want to create your own CD-ROM, either by using a writable CD drive or sending a tape to a vendor to be mastered, there are some tools available under Linux that you can use.

The **mkisofs** package allows creating an ISO-9660 file system on a disk partition. This can be used to assist in creating and testing CD-ROM file systems before mastering discs.

There are also some utilities available for verifying the format of ISO-9660 file systems; these can be useful for checking suspect CD-ROMs.

Getting the CD-ROM HOWTO

## Common Problems

If you encounter problems setting up CD-ROM support under Linux, here is a list of things to check for. (See the CD-ROM HOWTO for more information.)

- Are the appropriate CD-ROM driver(s) compiled in the kernel? Try the command **cat/proc/devices** to see which drivers are installed.
- Are you running the newly configured kernel? Use **uname -a** and check the time-stamp to see.
- Is the drive recognized at boot time? The **dmesg** command should redisplay the boot messages if they scroll by too quickly to read.
- Did you create the proper device files and set protections? The **/dev/MAKEDEV** script is one way to do this.
- Is the ISO-9660 file system compiled in the kernel? Try **cat /proc/filesystems** and look for iso9660.
- Is there a known good CD-ROM (not audio CD) in the drive?
- Did you use the correct options to mount? You need to specify **-t iso9660, -r** (read-only), the CD-ROM device file, and an empty directory. You should run this as user “root”.
- Can you read data from the drive? Try using the dd command and checking for disk activity (e.g., **dd if=/dev/cdrom of=/dev/null bs=2048**).

For some drives, if they are located at a non-standard I/O address, you may need to edit the appropriate kernel driver header file.

Getting IFS

## For More Information

Here are a number of additional useful sources of information related to CD-ROM under Linux.

The Linux CD-ROM, SCSI, and Distribution HOWTO documents are freely available from major Linux archive sites, including sunsite.unc.edu in the directory **/pub/Linux/docs/HOWTO**. For those without network access, printed copies of the Linux HOWTOs are also published by a number of vendors, or you may be able to find them on a local computer bulletin board system.

The latest and most complete information on the Panasonic/SoundBlaster CD-ROM kernel driver can be found in the file **README.sbpcd**, usually found in the directory **/usr/src/linux/drivers/block**.

Additional information on commands such as mount and umount can be found in the corresponding Linux man pages.

For those with access to Usenet, the following news groups discuss information related to CD-ROM:

- comp.publish.cdrom.hardware
- comp.publish.cdrom.multimedia
- comp.publish.cdrom.software
- comp.sys.ibm.pc.hardware.cd-rom
- alt.cd-rom
- alt.cd-rom.reviews

A Frequently Asked Questions (FAQ) document for the **alt.cd-rom** newsgroup is also available; it is archived on many Internet sites including **rtfm.mit.edu**.

The Internet site ftp.cdrom.com has a large archive of CD-ROM information and software; look in the directory **/pub/cdrom**.

The Linux Documentation Project has produced several books on Linux; the most useful for new users is *Linux Installation and Getting Started*. These are freely available by anonymous FTP from major Linux archive sites or can be purchased in hard-copy format.

The Linux Software Map (LSM) is an invaluable reference for locating Linux software, including the programs mentioned in this article. The LSM can be found on various anonymous FTP sites, including **sunsite.unc.edu:/pub/Linux/docs/LSM.gz**.

(Jeff.Tranter@Software.Mitel.com) is a software designer for a telecommunications company in Ottawa, Canada. He has been using Linux for almost two years and is the author of the Linux Sound and CD-ROM HOWTO documents.

Archive Index Issue Table of Contents

Advanced search

# Linux User Group News

**LJ Staff**

Issue #7, November 1994

*Linux Journal* would like to promote and support user group meetings, and plans on having a column dedicated to LUGs.

Is there a Linux User Group meeting in your area that you attend? Would you like to find one? Would you like to start one? *Linux Journal* would like to promote and support user group meetings, and plans on having a column dedicated to LUGs. We'd like to include both announcements of meetings and perhaps brief summaries, too, as space permits. Also, since *Linux Journal* does have some readers who aren't yet connected to the Internet, please provide contact information other than e-mail addresses in your submissions. Thanks!

## TLUG

Toronto, Canada

The inaugural meeting of the Toronto Linux User's Group was held on August 31, at 7pm. The North York Public Library at 5120 Yonge Street was the location of the event. If you'd like more information about this LUG, please send e-mail to Laszlo Herczeg at las@light-house.gts.org.

## BUUG

Belgium

One hundred copies of *Linux Journal* were ordered for the September 10th meeting of the Belgian Unix systems User's Group. We didn't get any further information about what they did at this meeting, but you might be able to find out from Jan Vanhercke at java@java.be.

### PLUG

Phoenix, AZ

The PLUG (Phoenix Linux User's Group) had their August meeting at the Arizona Center Food Court at 11:45am on the eleventh of August. Highlights included an InfoMagic CD give-away and a report from “The Other Side” (i.e., NetBSD). For more information send e-mail to plug@tsiung.dist.maricopa .edu.

### SLLUG

Salt Lake City, UT

The August meeting of the Salt Lake Linux User's Group meeting was held on the 18th, at the Sandy Library (10010 S. Petunia Way, Sandy) at 7pm. Bryan Ford was the guest speaker and the focus of the meeting was on operating system features. For more information about the SLLUG, send e-mail to vir@xmission.com.

### Long Island Linux

Long Island, NY

The Board of Directors of the Suffolk County Computer Association (SCCA) has resolved to establish a Linux user group named Long Island Linux. Anyone interested is asked to send e-mail to Jim Edwardson at ceo@van.gapi.com.

### Youngstown State University, OH

A Linux User's Group at Youngstown State University for North-East Ohio is in the beginning organizational stages. Contact Steven A. DuChene at sduchene@cis.ysu.edu or s0017210@cc.ysu.edu if you would like more information.

### DC Linux User's Group

Washington, DC

The DC Linux User's Group meets at NIH Bethesda, Building 12A. Meetings are typically the first Wednesday of each month with presentations starting at 7pm. NIH is just inside the Beltway on Wisconsin Avenue. A WWW map is available. Meeting notices are posted to dc.general and dc.org.linux-users. Contact: Przemek Klosowski, przemek@rrdjazz.nist.gov or (301) 975-6249.

### LUNICS SIG

Scotch Plains, NJ

Since early spring, the The LUNICS SIG, an offshoot of the ACGNJ (Amateur Computer User Group of NJ), has been meeting on the second Friday of the month at the Scotch Plains Rescue Squad in Scotch Plains, NJ. The group is composed mainly of Linux users but welcomes Unix, Coherent, Free/Net-BSD freaks. Meetings have included everything from a demo of Linux projected from a MasterSport II to a screen, to a demo of “mouseless X-windows”, to extensive random access. For further information, e-mail to Peter Fillingham at pete@panix.com.

### Brookhaven National Lab, NY

This LUG is fairly active, with 30 members and meeting once a week. More information from Jim Quinn, JQUINN@bnlnr.hfbr.bnl.gov.

### PLUG

Portland, OR

The Portland Linux User's Group and Mutual Aid Society meets the third Saturday each month, usually at the Round Table Pizza at SE Foster and Holgate in Portland at 4pm. For more information contact Sean Utt at seanu@plaza.ds.adp.com, or use the PLUG mailing list plug@aseTT.com.

# What's GNU

**Arnold Robbins**

Issue #7, November 1994

This month's column discusses groff, the GNU version of troff.

## groff

*This month's column discusses **groff**, the GNU version of **troff**. Explaining **troff** in full detail can (and has!) taken more than one book. For now, we'll provide a little bit of history and an overview of what **groff** is, what the input tends to look like, and how you would use it.*

by Arnold Robbins

## What is troff?

While there are many WYSIWYG word processing programs out there, some of which are quite powerful, and others which are usable and freely available, many long time “power users” still prefer text formatters like **troff** and TeX for the control they give you. Another advantage that these programs have is that you can edit the input using any text editor, even **ed** or **vi** over a 2400 baud modem connection, or on a laptop system that can't support X windows.

**nroff** and **troff** are the Unix text formatters. They are essentially twins; each accepts the same input “language”. The difference is in the output they produce. **nroff** was designed to produce output for devices with fixed-width and fixed-size characters, such as terminals and line printers. **troff** was designed for photo-typesetters. **nroff** simply ignores requests that it cannot honor. From now on, we will follow the time-honored convention of referring to both programs as **troff**, to make things simpler.

**troff** was written at Bell Laboratories by the late Joseph Osanna. It was modeled after the text formatters of the time, notably one named **runoff**. (**runoff** was written by Jerry Saltzer for the CTSS system at MIT, running on a modified IBM 7094, in the middle 1960's time frame.) Interestingly enough, **nroff** was written first; the name stood for “new **runoff**”. Later, when the research group acquired a photo-typesetter, **nroff** was enhanced to deal with the newly acquired capabilities, and thus **troff** was born. In the early 1980's, after the death of Mr. Osanna, Brian Kernighan took over **troff**, cleaned it up and enhanced it. The **troff** language is now frozen. It will not evolve further.

### troff's Capabilities

Input to **troff** is a mixture of text and formatting commands. You might think of this as “what you want to say” and “how you want to say it.” Typically, commands are on separate lines by themselves. **troff** is able to distinguish commands from text, since command lines begin with a dot, or period. Special tricks have to be used to get **troff** to treat a line that begins with a dot as real text.

There are a large number of commands in **troff**. Some of the more important commands are for the following tasks:

- Filling. This means putting as many words of input text on one output line as possible.
- Adjusting. This means padding lines with blanks so that the margins on both sides are even. Book and magazine text is typically both filled and adjusted.
- Font changes. Printed text is often in multiple fonts. *Italics* are often used for emphasis. **Bold** text is used for strong emphasis and for headings. **troff** supports at least four fonts normally, with the ability to easily add others.
- Size changes. Photo-typesetters give you the ability to print characters at different sizes. Most text is set in 10-point type, where one point is 1/72 of an inch. Text can be made smaller or larger as needed. For example, footnotes are often set in a smaller point size than normal text.
- Margin control. **troff** gives you separate commands to control the size of all four margins on the piece of paper. This is typically done using a combination of the
    - - line length, how many characters or inches of text that can be in a line
    - - the page offset, how far to the right to shift the entire line, and
    - - the indentation, how far left or right from the beginning of the line to actually place text. E.g., in a book, the first line of a paragraph is often indented 1/2 an inch.
- Centering. Any number of input lines can be centered in the output text.
- Line drawing. **troff** can draw horizontal, and vertical lines, as well as arbitrary curves.

- Horizontal and Vertical Motions. You can move text up or down an arbitrary amount. Consider subscripts and superscripts in mathematical formulae, or footnotes indicators, which are often one half a line up and in a smaller point size.

You can add comments to your **troff** source. They begin with **\"** and continue to the end of the line. We will be using comments in our examples, to help explain what is going on.

Many of the facilities can be done both as standalone commands, and with in-line escape sequences. For example, to change to a bold font, one might have text like this:

```
Here is some regular text.
.ft B  \" now switch to bold
This is bold.
.ft    \" switch back to earlier font
This will be regular again.
```

You can do the same thing with in-line escape sequences. For font changes, you use **\f** and either a single letter font name, or a **(** and a two letter font name. A similar example would be:

```
Here is some regular text. \fBThis is bold.\fP This will
be regular again.
```

The letter **P** is special. It means to use the previous font.

**troff** provides two nice features, strings and number registers. A string is a shorthand for some text. For example, if you don't want to type **the Linux Operating System** over and over again, you could define a string **LX**, and then use the string in your text. This feature can save a lot of typing.

```
.ds LX the Linux Operating System \" ds means define string
If you are new to \*(LX, then you should subscribe to
\fILinux Journal\fP. It covers \*(LX in great detail,
month after month.
```

Number registers are like variables in programming languages. They can contain numeric values. They can also be used in an “auto-increment” and “auto-decrement” fashion. This means that with each use, the value goes up by one or down by one. Why would you need such a thing? Think about automatically numbering chapters, sections, and subsections, as well as figures and footnotes. You would have a register for the chapter number, another for the section number, and so on. With each new chapter, the section register is reset to one. With each new section, the subsection register is also reset to one, and so on.

## Macros

As you are hopefully beginning to see, **troff** provides you all the mechanisms you need for complete control over your document's format. Unfortunately, this is often more control than you need. Writing documents using bare **troff**, while possible, can be quite painful. It is very much like programming in assembly language: you have complete control, and when the result works it works really well, but there is an awful lot of detail to keep track of, and it can be tedious and difficult.

To make it easier for regular users to manage the detail, **troff** allows you to define macros. A macro is like a subroutine in a programming language. You can group commands together to perform a larger task, and use the macro name, instead of writing out the entire sequence of commands each time.

Consider starting a new paragraph. You have to do the following tasks:

- 1. See if there is room left on the page for at least two lines of text.
- 2. Skip a space after the last paragraph.
- 3. Indent the first line of text by 1/2 an inch.

You could write the commands to do this over and over again. But that is (a) tedious, (b) a hassle to update if you change how you do paragraphing. Instead, you can define a macro, say **.P**, to do this for you.

```
          .de P   \" DEfine paragraph macro
          .ne 3   \" we NEed at least three lines
          .sp     \" SPace down one line
          .ti .5i \" Temporary Indent .5 inches
          ..      \" end the macro definition
```

Then, in your text, you just put **.P** on a line by itself wherever you want a paragraph.

```
          .ds LX the Linux Operating System
          .ds LJ \fILinux Journal\fP
          If you are new to \*(LX, then you should subscribe to
          \*(LJ. It covers \*(LX in great detail,
          month after month.
          .P
          And even if you are an experienced user of \*(LX,
          \*(LJ will bring you valuable tips and tricks to keep
          your Linux system up and running.
```

One of the less attractive features of standard **troff** is that command, macro, string, and register names are limited to no more than two characters. Fortunately, **groff** allows longer names, with a different syntax for accessing them. In fact, **groff** has many nice extensions over **troff**, making the task of writing macro packages considerably easier. The extensions are documented in the **gtroff**(1) man page.

## Popular Macro Packages

There are a number of popular **troff** macro packages. Using them is like programming in FORTRAN; it beats the heck out of Assembly Language, but it's not as nice as C or Modula-3.

The common macro packages are:

- **-ms** - Manuscript macros. Originated in V7, popular on Berkeley Unix.
- **-man** - Manual Page macros.
- **-mm** - Memorandum Macros. Very powerful macros, popular on System V.
- **-me** - Berkeley Technical Paper macros. An ugly package.
- **-mdoc** - The new Document Macros from Berkeley.
- **-mandoc** - A package that figures out dynamically if you want **-man** or **-mdoc**.

**groff** will support these directly if you have them, particularly using the **-C** compatibility mode option. It also has its own version of many of these packages.

The **-ms** and **-mm** are the most portable packages to use. **-mm** has many more features than **-ms**, thus making it harder to learn. In the long run though, the effort is worth it, because you can do so much.

## Preprocessors For troff

Over the years, it was found that macros helped, but that there were some things that were just too difficult to do in bare **troff**, even with macro packages. The approach that was developed was to write a “little language” that solved a particular task, and to pre-process the language into raw **troff**. The common pre-processors are:

- **tbl** - formats tables
- **eqn** - formats equations
- **pic** - formats pictures (diagrams)
- **grap** - formats graphs

As an example, here is part of a table from a reference card I worked on:

```
          .TS
          tab(~);
          lfB l lfB l.
          abs~absolute value~int~integer part
          acos~arc cosine~log~natural logarithm
          asin~arc sine~sin~sine
          atan~arc tangent~sinh~hyperbolic sine
          cos~cosine~sqrt~square root
          cosh~hyperbolic cosine~tan~tangent
          .TE
```

We'll explain it line by line. First, **tbl** only looks at lines between **.TS** (table start) and **.TE** (table end). Everything else is left alone. This makes it easy to use **tbl** in a pipeline with the other preprocessors. The first line sets the tab character to **~**. Normally, tabs in the input separate each column of the table. For this table, a **~** is used to make it easier to mark off the columns.

Then, for each line of data in the table, you provide a line that describes the layout information. **l** means left justified, **r** means right justified, and **c** means centered. All the columns in this table are left justified. The first and third columns also use a different font (the **f**). Here, they are using the bold font.

In this example, there is only one control line, so it is applied to all the data lines. For more complicated tables, you have one control line per data line, with the last control line applying to any remaining data lines.

The other preprocessors are similar in functionality. **grap** is actually a preprocessor for **pic**.

Typically, the commands are used in a pipeline:

```
          grap doc.tr pic  tbl eqn  troff -mm -Tps > doc.ps
```

The actual usage will vary from machine to machine; we'll see below how to run **groff**.

## Output Devices

**nroff** was originally designed for terminals and line printers, which are devices with fixed width characters. **troff** was designed for the Wang CAT photo-typsetter, which could have up to four different fonts available, in many different point sizes.

Around 1980, Brian Kernighan revamped **troff** to create **ditroff**, the device-independent **troff**. This version accepted an enhanced language, and generated ASCII output that described the motions the output device should make around the page, the size and placement of characters, and so on. Then, to add a new output device (laser printer, photo-typesetter, or whatever), you would write a post-processor for the device independent output that would correctly drive your device. Recent version of **troff** are the device-independent version, usually with support for PostScript(TM) output.

The **ditroff** saga can be found in Bell Labs Computing Science Technical Report #97. You can get PostScript for this report via anonymous **ftp** to **netlib.att.com**. Change to **/netlib/att/cs/cstr**, use **binary** mode, and retrieve **97.ps.Z**.

## GNU troff

Now that you know what **troff** is, we'll discuss the specifics of the GNU version, **groff**. **groff** is written in C++. This is somewhat unusual; most GNU programs are written in C. To compile it, you need a C++ compiler. The GNU C++ compiler, **g++**, will usually do it with little problem. You will need both **g++** and the GNU C++ library, **libg++** to compile the **groff** suite of programs.

The programs in the suite are:

```
          gtroff - the actual troff clone
          gtbl - the tbl clone
          geqn - the eqn clone
          gpic - the pic clone
          groff - the driver for the other programs
```

There is no **grap** clone. Anyone who wishes to write one should contact **gnu@prep.ai.mit.edu**.

**groff** has a large number of extensions over Unix **troff**. It particular, **groff** supports long names for commands, strings, and registers, and has many additional commands. It also has a compatibility mode, where all the extensions are turned off. This is occasionally necessary when using macro packages meant for original **troff**.

The **groff** pre-processors described above cannot be used with original **troff**; they take advantage of **groff**'s extensions.

**groff** uses the **ditroff** model of post-processors for different devices, with the same intermediate format. By default, **groff** generates PostScript output. The other most useful output format is plain ASCII. This is in fact how **nroff** is provided; by a shell script that calls **groff -Tascii** (i.e., the output type [**-T**] is ASCII). An interesting output type is TeX DVI, which can be used on many older laser printers that do not support PostScript. **groff** comes with two previewers for X windows, using different density fonts (75 and 100 dots per inch).

**groff** comes with a number of macro packages. It has its own version of the **-man** macros. The **-mgs** package is the GNU version of **-ms**, and **-mgm** is the GNU version of **-mm**. These should be used in preference to the original packages, since they can also take advantage of the **groff** extensions. The Berkeley **-me**, **-mdoc**, and **-mandoc** packages are themselves freely distributable, and are included with the **groff** distribution.

What is really nice about **groff** is that it is like **lint** for your **troff** documents. The programs check *everything*. Many things that Unix **troff** silently ignores, **groff** will warn you about. Often there are subtle errors in your files, and **groff** will help you catch the problems. Although every once in a while, there really is no problem, and you need to use compatibility mode instead.

Unfortunately, the one major lack in the **groff** distribution is that there is no comprehensive manual. The Tenth Edition Research Unix Programmer's Manual describes **troff** and its friends in detail. **groff** is based on this specification. Additional information can be found in the man pages that come with **groff**.

Information about **pic** can be obtained via anonymous **ftp** from the same site and directory mentioned above, in the file **116.ps.Z**. A description of **grap** can be found in **114.ps.Z**.

## Summary

GNU **troff**, **groff** is a powerful, complete implementation of the **troff** software suite. If you will be doing anything with **troff**, it is definitely the version to get. It generates PostScript by default, will find bugs in your documents, and supports all popular macro packages. The source code is available on **prep.ai.mit.edu** in **/pub/gnu**, in the file **groff-1.09.tar.gz**. It should be found on all GNU mirror sites as well.

## Editorial

Every once in a while, it is a worthwhile exercise to step back and stop and think about the free software you use with Linux, day in and day out. The Linux kernel is only one part of it. There are literally hundreds of utility programs, the majority of which were produced by Free Software Foundation staff and volunteers. The GNU General Public License, whose terms cover the utilities and the Linux kernel, came from the FSF. Linux is testimony to the idea that freely distributable software can be usable, and of high quality. Linux would have never happened if it had it *not* been free, and had there not been the GNU utilities to complete the picture.

### <u>Free Software Foundation Information</u>

It is only good sportsmanship and fair play to "give something back" to the organization that has done so much for you: the FSF. You can help further the cause of the FSF in a number of ways, both directly and indirectly.

If you are a programmer or a writer, or both, the FSF has software *and* documentation that needs to be written. Serious volunteers are always welcome.

If you want to help support the FSF monetarily, you can do that too. You can buy software and/or documentation from them. The FSF sells tape and CD-ROMs with their software on it. You probably already have most of the software, but you may wish to have the printed documentation that goes with it. The GNU manuals are nicely printed and bound, and are not that expensive. Buying software and manuals directly contributes to the production of more, high quality, free software.

In the U.S., you can make tax-deductible donations to the FSF. It is considered a non-profit organization under U.S. law. This also helps.

Indirectly, you can choose to buy your Linux distributions from resellers who state that they give a percentage to the FSF. If your favorite distributor does not do this, then ask them *why* they don't, and encourage them to do so.

Consider what you can do to help the FSF, and then do it!

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

Archive Index Issue Table of Contents

Advanced search

# Linux Events

**LJ Staff**

Issue #7, November 1994

Open Systems World and Amsterdam.

## Open Systems World

*Linux Journal* will be hosting the Linux International Users Conference at the 6th Annual **Open Systems World/FedUNIX '94**. The event is being held at the Washington Conference Center, Washington, D.C, during the week of November 28, and the two-day Linux Confer-ence will be on Thursday and Friday, December 1-2.

Eight other conferences will be held during the week, including Federal Open Systems Conference, Motif/COSE International Users Conference, Novell AppWare Developers Conference, SCO Interoperability Confer-ence, Solaris Developers Conference, Windows NT Developers Conference, and the Word Wide Web/Mosaic Users Conference. The event is expected to attract over 10,000 attendees, so this is a great opportunity for Linux to show its stuff!

The Linux Track will include tutorials, panel discussions and presentations by some well-known personalities in the Linux Community, including Bob Amstadt, Eric Youngdale, Don Becker, Phil Hughes, Przemek Klosowski, Dirk Hohndel, Michael K. Johnson, David Wexelblat, and Matt Welsh. Sessions will cover the history of Linux, Linux and the Internet, Wine, the commercial future of Linux, Linux and NASA, legal implications of using and developing tools and applications on Linux, iBCS2 compatibility, X Windows System on Linux, a clinic for the novice user, and how to convince your boss/employer/customer to use Linux.

There will be Birds of a Feather sessions at the hotel on Thursday evening, with discussions on systems administration, Internet connectivity, and hackers; fun with Linux.

If you would like more information, contact us at *Linux Journal*, call Open Systems World at (301) 953-9600, or try URL:www.mcsp.com/OSW-FedUNIX.html.

## Amsterdam

One short week after the Open Systems World event in the United States, the **International Symposium on Linux** will be held in Amsterdam. The RAI Congress Centre is the place, December 8 and 9 are the dates, and the organizers are Frank B. Brokken, Karel Kubat, and Piet W. Plomp of the ICCE, University of Groningen.

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

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

People without Internet access can reach ICCE at:

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

# Andy

**Andy Tefft**

Issue #7, November 1994

Now and then, Andy Tefft will tell us what he ahs been doing with his Linux system, which may give you some ideas for your own system. This month, he installs Mosaic and pays with a clone of the classic board game, Risk.

I admit it. I'm hooked on Mosaic. You don't know what Mosaic is? Neither did I, until a few months ago. Mosaic is perhaps the most well-known WWW browser, at least for X and Microsoft Windows. (To find out more about WWW, see the article by Bernie Thompson in *Linux Journal* issue 3.)

After getting Mosaic at work (with no actual Web access, but to see how I could use it to display my own internal documentation), I decided I wanted to install it at home. I don't have Motif, which is required to compile Mosaic, but thankfully a few people have put pre-compiled binaries of different types on **sunsite.unc.edu** (**/pub/Linux/system/Network/info-systems**). Installation was easy; I just extracted the gzipped tar file, copied the binary to a suitable location, and copied the app-defaults file into a suitable location (**/usr/X386/lib/X11/app-defaults**). The **README** material from the source is also included. This is not a small package; over 1MB for the Mosaic binary on disk, and it uses nearly 2MB of RAM.

Now I can Mosaic over my PPP link from home. What fun! Interestingly, I have found that the traffic it generates is not too hard on the limited-bandwidth PPP link (except when loading large images). For the net-impaired, there is also a version of Mosaic which works over a term connection in the same directory on sunsite.

In between Mosaic-ing (OK, it does get a little boring when all you have to browse with your information browser is information you created), I had a little time to try out Frisk-0.99a, a pretty nice Risk clone by Elan Feingold (**elan@tasha.cheme.cornell.edu**). It is multi-player and networked, so you at least have to have loopback networking to be able to use it. Also, there is no computer opponent, although two players can share one window and one screen. It has a nice help facility, too. There is only one game style supported, but there is a lot of potential to this one.

Archive Index Issue Table of Contents

Advanced search

# Letters to the Editor

**Various**

Issue #7, November 1994

Readers sound off.

## Free Means You Pick the Price

I enjoyed [Matt Welsh's] article Thou Shalt Not Use MS-DOS, in the latest issue of *LJ*. I think what [he] said is true, but misses a point Phil Hughes has alluded to in an earlier editorial.

I think the reason that Linux; and free software in general; is so great is that it's an incredible consumer value. According to the dictum, “what do you put in to it, what do you get out of it, and what about support?”, most commercial software falls right down flat: it's expensive to buy, you often can't try it out in advance, and if it's broke you're at the mercy of the vendor, who is the only one with the source code.

If free software was free but you didn't get the source, it would be terrible; you could try it in advance, but when it finally did break, no doubt in the middle of an important project, you would be stuck. On the other hand, if payware came with source code, it would be a much better deal: when it broke, you could ask the manufacturer to fix it, or you could call your local hacker and negotiate a price based on how soon you need it fixed. Of course even payware with sources still doesn't let you test it all in advance of buying it, but it's a much better consumer value.

Consider the auto industry for a moment. It will cost you about as much to service a car over its life as it cost to buy in the first place. Would you pay double the price up front in order to avoid paying for service down the road? Yet this is just what software houses do, since the only way they can pay for customer support is by charging more for the product. It's no wonder software support is so lousy! This is why being able to hire your own hacker is so important.

And thus a common point of confusion about free software: it is free in the liberty sense, but it isn't free in the money sense, since programmers have to eat just like everybody else. What free software is is a much better deal. Even if you had to pay a little bit for it, it would be a great deal. Imagine that you had to pay Linus $1 every time you got a new major revision of Linux. Linux would still be the best consumer bargain going. And, indeed, many Linux users pay $40 and more for a convenient distribution.

And that, in my opinion, is what makes Linux great. It's cheap to test drive, it is high quality, it's cheap to buy, and when something breaks you can decide how much it's worth to you to get it fixed, how fast, and you can get it fixed. What do you put in to it? A few dollars. What do you get out of it? Professional-quality software. What about support? The best.- -David Keppelpardo@cs.washington.edu

## More About FSF?

1. Thanks for the *Introduction to the GNU C Library and Programming the VT Interface* columns in *LJ*. The notes on signal() were welcomed.

There have been a number of columns connected with the FSF+GNU. *LJ* should add a few notes to these columns on ways to support FSF and GNU. In your *GNU C Lib* article, you wrote “I do not know if [the reference manual] is being published on paper”. My copy of the GNU's Bulletin is at school, but I think it is listed. In any case, I'd like to see *LJ* more explicitly mention support of FSF (although you did mention FSF books in the sidebar).

Just to be completely clear: I'm not suggesting that *LJ* “beg” for money for FSF; only that *LJ* help new users become familiar with the project. (My only “official” connection with FSF is in support of GNU awk. I receive no money from FSF.)

2. There is a common thread in *LJ* and on newsgroups about the “free-ness” of Linux (Deutsch's letter in #2, Kempen interview in #3, Welsh's columns). Users forget (or are not aware of) the importance of various projects in the success of Linux. *LJ* could educate users in future columns on GNU software, C-Kermit, ghostscript, etc.—Darrel Hankersonhankedr@mail.auburn.edu

*LJ* Responds:

1. Read What's GNU for this month. While we won't run an editorial like that every month, we definitely agree that the FSF has played a key role in the development of Linux, and is worth supporting.

2. We have published several tutorials. It is true that quite a few have not been for FSF software, but that is probably because FSF is better known than some of the smaller new utilities that people express an interest in writing tutorials for. We encourage people who are excited about some package they work with to send e-mail to info@linuxjournal.com and discuss writing a tutorial for the package.

Archive Index Issue Table of Contents

Advanced search

# Stop the Presses

**Phil Hughes**

Issue #7, November 1994

While I expected development software there is a whole new area where Linux is becoming active: database software.

I was just flipping through back issues of *Linux Journal* looking for an idea for this column. What caught my eye was the change in profile of *Linux Journal* advertisers. CD-ROM distributors such as InfoMagic, Trans-Ameritech and Yggdrasil have been there from the beginning as have the systems distributors like Fintronic and Promox. The only Linux application that was advertised was Techplot from Amtec Engineering.

Things are changing. We have seen press releases and advertisements from hardware vendors (Cyclades communications board, for example), and software vendors. While I expected development software there is a whole new area where Linux is becoming active: database software.

New products from Revolutionary Software, Infoflex, WorkGroup Solutions, Poet Software and Ray Ontko & Co. offer commercial alternatives to Ingres and Postgres that come with many Linux distributions.

Finally, trade shows are starting to recognize the potential of Linux. *Linux Journal* is sponsoring a 2-day Linux track at Open Systems World. This puts Linux on an even keel with commercial tracks on NT, SCO and Solaris at the same show. Hopefully this will be a good chance for those of us in the Linux community to not just rub shoulders with “the commercial guys” but also to show them what we have to offer.

What does this mean? To me it means that Linux is well on its way to the commercial market. Six issues from now I expect that I will see ads and press releases for applications using the databases that have recently appeared. End users probably won't be asking “is it Linux” vs. “is it Unix” or “is it Netware” or “is it NT”. What they want is a solution and if we can offer the solution that works well we should get the market. And, the cost of Linux gives us a serious advantage.

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

Archive Index Issue Table of Contents

Advanced search

# New Products

**LJ Staff**

Issue #7, November 1994

Just Logic SQL Database and Yggdrasil Plug 'N Play.

## Just Logic SQL Database

Just Logic Technologies has introduced Just Logic/SQL Database Manager, an ANSI SQL relational database system for Linux. The database system includes a complete development system and utilities.

The system offers three choices of programming interfaces: a complete set of C++ Class definitions, a C application program interface, and a standard C precompiler for portability from or to other relational databases on other operating systems.

Contact: Just Logic Technologies, 40 Commerce Street, Nun's Logic, Quebec, H3E 1V6 Canada (514) 943-3749

## Yggdrasil Plug 'N Play

Yggdrasil's Fall 1994 Plug-and-Play Linux release and MoCheap 1.2.4 (a port of OSF/Motif to Linux) were released in September 1994.

The Fall 1994 release includes the new X11R6 XFree86 3.0 X-windows ported and integrated into Linux, an improved graphical user interface for Lucid Emacs and ImageMagick, and better DOS and Windows emulation.

To the 1.1.47 kernel used in the Fall 1994 release, have been added new device drivers that allow Linux to run under DOS and to transparently use DOS to access CD-ROM's and hard disks not directly supported by Linux.

Contact: Yggdrasil Computing, Inc. 4880 Stevens Creek Blvd., Suite 205 San Jose, CA 95129-1034 (408) 261-6630, FAX(408) 261-6631.

Archive Index Issue Table of Contents

Advanced search