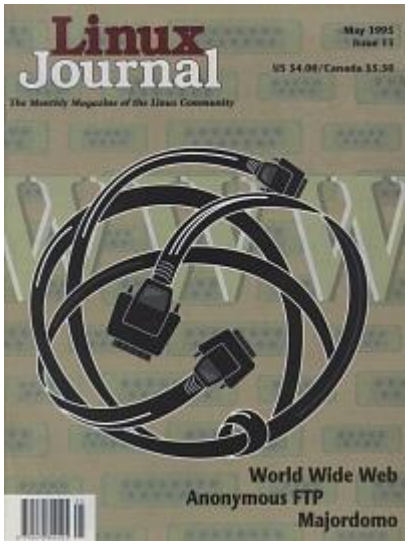
LINUX JOURNAL

Advanced search

Linux Journal Issue #13/May 1995



Features

Installing CERN's WWW Server *by Eric Kasten*
Tie your networked Linux machine to the World Wide Web.
Majordomo *by Piers Cawley*
Create your own internet mailing lists with the popular majordomo sofware.
The Pari Package on Linux *by Klaur-Peter Nischke*
Fast, efficient mathematical operations. Do arithmetic and symbolic math with Pari.

News & Articles

Netsurfing with Linux *by Arthur Bebak*
Linux for Public Service *by Dan Hollis*
Hamming it Up on Linux *by Brian A. Lantz*
A New Project or a GNU Project? *by Mark Bolzern*
Linus Torvalds Receives Award

Reviews

Book Review Linux Sampler *by Harvey Friedman*

Columns

Letters to the Editor
Stop the Presses Linus Torvalds Releases Linux 1.2.0 *by Phil Hughes*

Novice to Novice DOS Emulation with dosemu *by Dean Oisboid*
New Products
System Administration Anonymous ftp *by Mark Komarinski*
Kernel Korner The ELF Object File Format by Dissection *by Eric Youngdale*

Archive Index

Advanced search

Access Information Through World Wide Web

Eric Kasten

Issue #13, May 1995

The World Wide Web is designed to be easy to attach to. Not just by using Mosaic, Lynx, or Netscape to read other people's pages, but also by publishing your own information. Eric Kasten tells you how to start.

Access to networked multimedia data has become commonplace on the Internet. Chances are good that you've gone *surfing* in search of data for a project or technical paper or out of simple curiosity. You have probably passed through a large number of sites scattered across the globe and have seen what can be done. Now you want to set up your own collection of information, images, and sound so that the networked world can come and visit. One of the first steps is to set up a world wide web (WWW) server. The remainder of this article will endeavor to provide the necessary information to successfully install the WWW server provided by CERN.

Preparation and Compilation

The three basic parts of the CERN WWW source package are the line-mode client, the daemon, and the common library. Of these, the daemon and common library source are necessary to compile the server. You may also want to retrieve and compile the client code for future use. These packages may be retrieved by clicking on the appropriate links in the document found at URL www.w3.org/hypertext/WWW/Daemon/Status.html, or by anonymous ftp from gatekeeper.dec.com in directory /put/net/infosys/www/src. The file names are shown in the tar commands below. Once you have retrieved the tar archives, move them to the directory where you wish to compile the server, then extract the source. Using GNU's tar this can be done by executing the following commands:

```
tar xzvf WWWDaemon.tar.Z
tar xzvf WWWLibrary.tar.Z
tar xzvf WWWLineMode.tar.Z
```

The final file and tar command is optional since it extracts the line-mode client. Once extracted, the source will be in subdirectory WWW. To build the server under Linux, change to subdirectory WWW and execute BUILD. This is done as follows:

```
cd WWW
./BUILD
```

The build process should recognize that it is running on a Linux system, and proceed to compile the common library followed by the server. If you have not extracted the source for the line-mode client, the build process will exit with an error once the server has been built. This is normal and can be ignored. You can verify that you have successfully built the server by examining the files in subdirectory Daemon/linux. In addition to a number of .o files there should be the following six files:

- cgiparse: A tool for parsing CGI environment variables.
- cgiutils: A tool for generating replies from CGI scripts.
- htimage: Used to parse the input from a clickable image.
- httpd_3.0: The server or daemon.
- httpd: A symbolic link to httpd_3.0
- htadm: An administration tool for managing server access files.

CGI stands for Common Gateway Interface. CGI is a set of specifications for how scripts and other programs can interact with a WWW server.

Sidebar: The NCSA http Daemon

Basic Installation

You may install the server and associated tools in any directory you see fit. This article assumes that the installation directory is /usr/local/WWW. If you are installing in some other directory, be sure to make the necessary changes in the following references.

To install the server list, create a user id and a group for the server. This user id and group will help control what access rights the server possesses and allow selective access to the server files for administrative users. This article uses user **www** and group **wwwgroup**. They can be created using your favorite method for adding users and groups to the system.

Next, create the directory tree where the server and associated tools and files will reside. A typical server directory will have the following directories (and possibly others):

- config: Configuration files.
- cgi-bin: CGI scripts and programs.
- icons: Icons.
- htdocs: HTML documents.

To create these directories, execute the following:

```
mkdir /usr/local/WWW
cd /usr/local/WWW
mkdir config cgi-bin icons htdocs
```

Now these directories can be populated. Return to the directory where you compiled the WWW server, and execute the following:

```
cd Daemon/linux
tar cf - htadm httpd httpd_3.0 | \
  (cd /usr/local/WWW; tar xvf -)
tar cf - htimage cgiparse cgiutils | \
  (cd /usr/local/WWW/cgi-bin; tar xvf -)
cd ../../server_root/icons
tar cf - * | (cd /usr/local/WWW/icons; tar xvf -)
cd ../config
tar cf - * | (cd /usr/local/WWW/config; tar xvf -)
```

This sequence of commands copies the server binaries, CGI utility programs, supplied icons, and several example server configuration files to the proper subdirectories in /usr/local/WWW. Note how tar is used. This is an often-used method to successfully copy files, links, and subdirectories without accidentally modifying links and other characteristics during the copy operation due to improper flags passed to cp.

You should also set the ownership and permissions on this directory tree. Here is a possible set of permissions:

```
cd /usr/local
chown -R www.wwwgroup WWW
chmod a+rx WWW WWW/htdocs WWW/icons \
  WWW/cgi-bin WWW/config
chmod g+rx WWW/httpd_3.0
cd WWW/icons
chmod a+r *
cd ../cgi-bin
chmod a+rx *
```

These are typical permissions that allow access to the server when documents or icons are requested or the execution of a CGI program or script is required. However, it is possible to remove world permissions from most of these files as long as the server runs as user www and group wwwgroup (see UserId and GroupId below).

Basic Configuration

The /usr/local/WWW/config directory contains several example configuration files. The CERN server has a rich collection of options, including caching specifications, proxy support, and access control. This article will cover only a set of the basic options to get you started.

Listing 1 is a basic configuration file, which, with the following descriptions, you can modify to get your server up and running. This file should be created in /usr/local/WWW/config/ and can have any name you desire; however, here it will be called cern_httpd.conf. The default file name that the server will search for at startup is /etc/httpd.conf; however, this is easily overridden on the command line, or you can create a symbolic link between /usr/local/WWW/cern_httpd.conf and /etc/httpd.conf. I prefer to simply override the default, thus providing an obvious indication as to which configuration file is currently in use.

Upon examining listing 1 you will find a number of options set to various values. Note that a comment line can be added to a configuration file by using the shell script convention of placing a # in the first column.

```
# cern_httpd.conf
# An example httpd configuration file
ServerRoot /usr/local/WWW
HostName        wwwhost.my.domain.name
Port            80
PidFile         httpd-pid
UserId          www
GroupId         wwwgroup
AccessLog       /var/log/httpd.access
ErrorLog        /var/log/httpd.error
LogFormat       Common
LogTime         LocalTime
UserDir         public_html
Welcome         welcome.html
Welcome         index.html
AlwaysWelcome   On
#
# enable/disable methods
Enable          GET
Enable          HEAD
Enable          POST
Disable         DELETE
Disable         PUT
#
# Rules
Exec    /cgi-bin/*      /usr/local/WWW/cgi-bin/*
Pass    /*              /usr/local/WWW/htdocs/*
```

Listing 1. WWW Configuration File

One of the first configuration options is **ServerRoot**. This option determines the directory which the server will use as the default root directory. This may be prepended to other option settings (such as PidFile) in the case that an absolute path is not specified. In our case, ServerRoot should be set to /usr/local/WWW.

The HostName directive should be set to the fully qualified, dot-separated host and domain name of the host your server will run on. This is necessary so that the server can properly construct references to itself. This option may also be used to specify a hostname alias to be used in constructing URLs, as opposed to the hostname which is returned by the system.

Port specifies the port the server will accept connections on. When a client (such as Mosaic or Netscape) retrieves a document from your server, it will contact your host at this port to make a request. Ports provide a fixed location at which to access a particular service on a host. Many ports have been defined universally to be the access point for certain services. You may examine the /etc/services file to discover some of the ports that have been reserved on your system. If you are setting up a WWW server which you want accessible to the general public, you should probably use port 80. This port has been established as the default port for providing a hypertext transport protocol (http) service.

The **PidFile** directive specifies the file in which the server should log the process id of the principle httpd server. This process id can be used to help locate the server in the event that you want to send it a signal. The path specified can be either an absolute path or a path relative to the **ServerRoot**. For example, setting **PidFile** to httpd-pid would cause the server to log its process id in the file httpd-pid in the **ServerRoot** directory.

The next options are **UserId** and **GroupId**. These options specify the user and group ids under which the server will execute. In this article, as pointed out earlier, I will be using www as the user and wwwgroup as the group for the server.

Next are a set of options which control the kind of information the server will log about access activity and errors. **AccessLog** and **ErrorLog** should be set to valid file names. These logs may be useful for providing insight on tuning or security. You may want to keep these in /var/log or in a directory specially created for your WWW server logs. **LogFormat** should be set to **Common**, thus indicating a logging format that is likely to be recognized by many of the tools available to help you process the log information. LogTime can be set to either **GMT** or **LocalTime** depending on how you want the log records time stamped.

UserDir specifies the name of the public HTML document directory under each user's home directory. The example specifies public_html as the directory the server should support. This means that a universal resource locator (URL) of the form http://wwwhost.my.domain.name/~*username* is redirected to the directory ~*username*/public_html/. Each user on your system can then create a public_html directory where they can set up their own home pages or other publically available documentation.

There are several **Welcome** directives in the example. **Welcome** indicates which file should be presented when a URL is passed to the server where the path specifies only a directory. For instance, using the example configuration file, the URL http://wwwhost.my.domain.name/docs/ would result in either Welcome.html, welcome.html, or index.html being retrieved from directory /usr/local/WWW/htdocs/docs/ on the server. The order that the **Welcome** directives appear in the configuration file determines the search order which the server will use for finding the welcome document. Only the first document found will be displayed. **AlwaysWelcome** should normally be set on. If this option is off, the server will differentiate between URLs specifying directories with and without a trailing /. With this option off, a URL directory without a trailing / will result in a directory listing being displayed instead of the welcome document.

The **Enable** and **Disable** directives specify which methods are enabled on the server. Methods are actions that may be conducted during client-server sessions. For instance, the **GET** method allows documents to be retrieved from the server while **PUT** allows documents to be written to the server. By default, **GET**, **HEAD** and **POST** are enabled and **DELETE** and PUT are disabled. I prefer to explicitly define the methods so as to clearly control the accesses that I wish to allow. It is usually best not to allow destructive methods, such as **PUT** or **DELETE**, since it is possible to accidentally allow insecure accesses to the server which may provide a method for an intruder to enter your system.

The last section of our example configuration file deals with rules. Rules are used to control the processing of URLs which are passed to the server. These rules may map URL strings to specific files or operations. In the example file, two rules are included to simplify URL construction and to specify an executable path for the CGI programs and scripts. The first rule is an **Exec**. This rule tells the server that URLs which contain the string **/cgi-bin/** are to result in the execution of a CGI program or script in the physical directory /usr/local/WWW/cgi-bin/. **Pass** indicates a rule which causes all URLs starting with the string /to be mapped to the directory /usr/local/WWW/htdocs/. This is the directory where you should place your server's public HTML documents.

The Default Welcome Document

You will probably want to have a default document that will be displayed when a client accesses your server without specifying a particular document. For instance, a client might connect to your server using a URL of the form http://wwwhost.my.domain.name/. If there is no default welcome document, a file listing of the directory as specified by the **Pass** directive will be displayed. A client may then access any documents in this directory with the correct permissions. Also, keep in mind that if a document is in the htdocs directory, a

client can access it by explicitly naming that file in a URL. Only documents which you want made public should be here.

To create a default welcome page for your server, you need simply create a welcome document in the directory /usr/local/WWW/htdocs using one of the file names you specified in the server configuration file with a **Welcome** directive. To set up a simple default welcome document create the file /usr/local/WWW/htdocs/Welcome.html containing the following text and set the file permissions to be world readable (**chmod a+r**):

```
<TITLE>Welcome!</TITLE>
Welcome to my WWW server!
```

This will display a document with the title Welcome! and a simple welcome message displayed in the body. Later, you will probably want to add more interesting information to this file, along with links to other documents that your server provides.

Testing and Startup

Now the server is ready for an initial test run. You will first want to start up the server in verbose mode from a terminal connection so that you can check for any configuration errors. From a terminal, enter the following:

```
/usr/local/WWW/httpd -v -r /usr/local/WWW/config/cern_httpd.conf &
```

The **-v** flag indicates that server should run in verbose mode, and the **-r** flag specifies the server configuration file. The server will print a number of initialization messages and messages about any configuration errors it encounters as it starts. Check these messages for errors or discrepancies. Once the server has completed its start-up processing, you can use your favorite WWW browser to connect to it using an URL of the form:

```
http://wwwhost.my.domain.name/
```

If all goes well, you should see the message from the default welcome document. Be sure to examine the output of the server for any errors, and correct them as necessary.

The server can be started from one of your rc scripts (probably rc.local or rc.inet2) when the system is booted. It should be started after the network is initialized. A good place to start the daemon is in the same rc script used to start sendmail or inetd. The following entry can be used to start the httpd server daemon and to warn you if it cannot be started:

```
# Start the CERN httpd server
if [ -x /usr/local/WWW/httpd ]; then
          echo -n ", httpd"
```

```
            /usr/local/WWW/httpd -r /usr/local/WWW/config/cern_httpd.conf &
else
            echo
            echo "==================="
            echo " httpd not found. "
            echo "==================="
fi
```

This is a standalone startup of the daemon. You can also start the server using inetd, but you will get better response from a standalone startup. Proper configuration of inetd is outside of the scope of this article. If you wish to use inetd, please refer to the man pages and documentation on inetd and the relevant section of CERN's documentation available at www.w3.org/hypertext/WWW/Daemon/User/Installation/Inetd.html. **inetd** was also documented in the System Administration column in issue 12 of *Linux Journal*.

Controlling the Daemon

The httpd daemon accepts various types of signals. Two of these are most notable. A KILL signal will cause the daemon to terminate, and can be issued with the following command:

```
kill -KILL `cat /usr/local/WWW/httpd-pid`
```

or

```
killall -KILL httpd
```

Sometimes you may want to reconfigure the daemon without shutting it down. This can be done by changing the configuration files then issuing a HUP signal to the daemon as follows:

```
kill -HUP `cat /usr/local/WWW/httpd-pid`
```

or

```
killall -HUP httpd
```

As always, be sure to check the error log for any possible configuration errors.

User's Home Pages

For a user to create a home page, the subdirectory specified by the UserDir directive in the configuration file (public_html in the example) must be created in the user's home directory. The user should place all publically available documents under this directory. The user may want to create a welcome page for presentation when the server resolves URLs of the form http://wwwhost.my.domain.name/~joe. The server will attempt to resolve a URL of this form to reference a welcome document in the public_html directory. If a

welcome document is not found, a directory listing will be presented to the client instead.

For the server to properly access a user's home pages, the permissions on the user's home directory must be set to allow the server to *read through* the home directory into the public_html subdirectory. This can be done by setting the user's home directory so that it has world execute permissions (**chmod a+x**). The public_html directory must be set to allow world read and execute (**chmod a+rx**). Finally, the documents in the public_html directory should have permissions which allow world reading (**chmod a+r**).

An alternative to having users home pages in subdirectories of their home directories is to create a special directory tree under /usr/local/WWW/htdocs/. This directory tree will typically have a subdirectory for each user who will be creating public documents accessible via the httpd daemon. With this arrangement special permissions need not be set on users' home directories, maintaining some additional security. If this method is implemented, the **UserDir** directive shown in the example should probably be omitted. This will disable the support of URLs of the form http://wwwhost.my.domain.name/~*username*.

For example, you could create a /usr/local/WWW/htdocs/home/ directory. Under this directory you could create a subdirectory for each user who will be creating public documents. If the user is to be able to freely modify the directory contents, the ownership and permissions must be set accordingly. The permissions must also be set to allow the server to access this directory. This usually means making the directory world readable (**chmod a+r**). A user's home page can then be accessed using a URL such as http://wwwhost.my.domain.name/home/joe/.

You Have Just Begun...

What has been presented here has been designed to get a basic server up and running. Many topics are left out, including proxy support and special access control. The CERN httpd daemon is a very flexible and configurable WWW server. For further research you will probably want to explore the online documentation provided by CERN at www.w3.org/hypertext/WWW/Daemon/Status.html.

Eric Kasten has been a systems programmer since 1989. Presently he is pursuing his master's in computer science at Michigan State University, where his research focuses on networking and distributed systems. Well-thought-out comments and questions may be directed to him at

tigger@petroglyph.cl.msu.edu. You may also visit his home page at petroglyph.cl.msu.edu/~tigger/.

Archive Index Issue Table of Contents

Advanced search

Majordomo

Piers Cawley

Issue #13, May 1995

Want to set up your own mailing list? Piers Cawley takes you step by step through the process of creating a proper mailing list with Majordomo.

Majordomo is a nifty piece of perl that maintains mailing lists. The clever bit is that, once the system administrator at a site has completed the basic setup, the owner of the list, who doesn't actually have to have an account on the computer running Majordomo, can handle day-to-day maintenance via e-mail messages.

Majodomo was originally written by Brent Chapman after he got frustrated trying to compile and set up ListProc. The latest versions of Majordomo are maintained by John Rouillard.

Why Majordomo?

Majordomo and its associated programs were designed with the Unix guiding principle in mind; that of using suites of small, easily understood tools to do surprisingly complex tasks; and it does those tasks remarkably well. It also makes for an easily extensible system. There are some problems, but—and this brings us to the second big advantage of using Majordomo—you can fix them yourself.

Majordomo is free. This means that you get all the source code and you can modify it to your heart's content. Not only that, but that source is remarkably well commented and easy to understand, especially considering how easy it is to write obfuscated perl code.

Of course, there are things about Majordomo that things like LISTSERV and ListProc do better—it doesn't handle large, busy mailing lists as efficiently as they do. It is, however, easier to administer (once you've got the aliases set up) than either of these, you don't have to pay for it, and it works with Linux.

Fetching the Source

First make sure you have the most up-to-date copy you can lay your hands on. This discussion will cover Majordomo 1.93, but who knows: 2.0 might be out by the time you read this. Whatever you do, don't use 1.92, since it had a rather nasty security hole. Majordomo's home site is ftp://ftp.greatcircle.com/pub/majordomo/.

You'll also need perl, at least version 4.035, preferably 4.036. Majordomo does run under perl 5.000 but it was developed (and is being developed) under perl 4.036. The site here is running sendmail 8.6.91, but things should work okay with smail, although I haven't tried it. See footnote 1.

Compiling/Installing Majordomo

Compiling Majordomo is easy. Just a couple of caveats:

1. Majordomo expects perl to be in /usr/local/bin/perl
2. As it comes out of the box there is a bug which causes Majordomo to read in the config file twice, once from /etc/majordomo.cf and once from $homedir/majordomo.cf. This is easily fixed though; simply remove the line: **require `majordomo.cf';** from config_parse.pl.

For what follows, I'll assume you have set up a /etc/majordomo.cf which looks like this:

```
$whereami="yourhost.yourdomain";
$whoami="Majordomo@$whereami";
$whoami_owner="Owner-Majordomo@$whereami";
$homedir="/usr/lib/majordomo";
$listdir="/var/spool/majordomo/lists";
$digest_work_dir="/var/spool/majordomo/digest";
$log="$homedir/Log";
$mailer="/usr/lib/sendmail -f$sender";
$filedir="/var/spool/majordomo/archive";
$filedir_suffix="";
$index_command="/bin/ls -lRL";
$return_subject=1;
$majordomo_request=0;
umask(007);
@archive_dirs=();
```

and that the directories /var/spool/majordomo/lists, /var/spool/majordomo/digest and /var/spool/majordomo/archive exist and are owned by majordomo.majordomo with mode 775.

Configuring Basic Aliases

For this I'll assume you're using sendmail 8.6.9 (purely because that's the mailer that we use here at Frontier). If you're not using a version 8 sendmail you will need to add majordom to the list of “trusted” users that sendmail allows to

spoof the sender of a mail envelope (find the T line in sendmail.cf and add majordom to the list).

As root, add the following lines to your /etc/aliases file:

```
majordomo: "|/usr/lib/majordomo/wrapper majordomo"
owner-majordomo: postmaster
```

Now assume you want to set up a list “test” with all the bells, whistles, and gongs that Majordomo provides. You'll need to add the lines shown in Listing 1 to your /etc/aliases file.

Run newaliases to make the aliases visible.

Now, su to majordomo and run the commands in Listing 2.

Edit /var/spool/majordomo/lists/test.config, set a new list password, and add a moderator if necessary. Hopefully it's all pretty self-explanatory. Copy test.config to test-digest.config and edit things like **message_footer** and **message_fronter**.

For example:

```
message_fronter << END
  In test today:
_SUBJECTS_
END
message_footer << END
--
To switch to the undigested list,
send this to majordomo@host.domain
unsubscribe test-digest
subscribe test
end
END
```

Watch out for the config sections advertise and noadvertise. These expect a list of perl regular expressions, complete with /'s. These are used when someone queries your Majordomo server for a list of lists. So, say you had a list “local-jellygarglers” which you didn't want anybody off your site to know about (and who would), you could put the following in your local-jellygarglers.config file:

```
advertise << END /yourdomain$/
EOF
```

which means that Majordomo will only mention the list in its list of lists if the sender's address is in the domain yourdomain.

The default digesting provided by Majordomo is not bad for busy lists. It waits until the digest size reaches a certain specified size (use the maxlength setting in test-digest.config to set this) then sends the digest. However, this is not a

good solution if you're looking at a list which has periods of infrequent posting —especially if the majority of the lists subscribers go for the digestified version, there is the potential for a message to hang around unread in the digest queue for weeks. There are two solutions to this. The first method is to copy contrib/digest.send into /usr/lib/majordomo, editting the value of DIGESTDIR to correspond with the directory where you stash your digests, then add the line

```
0 2 * * * /usr/lib/majordomo/digest.send
```

to Majordomo's crontab file. The digest will be sent every morning at 2, no problems.

The second (and to my mind neater) method is to apply Paul Close's patch to digest and use a slightly different crontab line. The patch to digest does a couple of things. It adds the variables **digest_maxlines** and **digest_maxdays** to the list.config files. (If you do this after you've created a config file you'll need to mail the commands **writeconfig** test and **writeconfig** test-digest to majordomo so it can write out a new, valid config file again, so this is really best done before you make the config files.) These two parameters mean you can tune the digester so that a digest should never get sent with any messages older than (say) a week in it.

The patch also adds a **-p** switch to digest. When digest is invoked with this switch it checks to see if the digest should be sent (i.e., it checks to see if any articles in the digest are too old), and sends it out if necessary. To use this, simply change the **-m** to **-p** in the line that invokes the digest program from digest.send.

Setting Up Archiving

I'm assuming for this that you're going to be using rouilj's archive2.pl for your archiving work. This can be found in contrib/archive2.pl in the Majordomo source directory and it should also be linked to archive in the Majordomo home directory.

The only thing you need to do to get archive2 to work for the test mailing list is change the

```
@archive_dirs lines in /etc/majordomo.cf to
@archive_dirs=("/var/spool/majordomo/archive/test");
```

and, as they say, voilá!

Setting Up Files For Remote Retrieval

Since you've set up the rules for finding a given list's archive, Majordomo itself knows that a list's files are stashed in that directory (in our case /var/spool/majordomo/archive/test), and anything in there can be retrieved by members of that mailing list (or others depending on how you've configured the list) using the get command in mail messages to Majordomo.

Majordomo neither knows nor cares about the contents of this directory. It simply returns a list of files when asked for an index, or returns the file when asked for that. These files don't actually have to be mail archives, so you can put anything there that may be of interest to the list's users. For example, we have a client who is running a play-by-mail game and he puts the files that are freely available to all the game's players within the archive directory so that his players can retrieve them using mail.

However, there is a caveat associated with Majordomo's file handling. Since Majordomo doesn't know about the contents of a file, it doesn't know if that file is a binary file. (This is a conscious design decision—if you need to do heavy duty file work Majordomo can interface with ftpmail, but that's another story.) Thus, if you want to make binary files available you will need to uuencode and split them up by hand first.

Almost There…

The list is now virtually ready to hand over to the list administrator to operate via mail. The only thing left to do is to create a temporary info file (/var/spool/lists/test.info) for the mailing list—this should be a short file which gives a description of the list, what it's about, who's in charge, policy, that sort of thing. If you don't know all this, just put a dummy info file in place and let the list administrator worry about setting it correctly.

Now you are ready to set up the list's first subscriber. This is usually the list's owner (the chap that test-approval and owner-list point at), in this case pdcawley@ftech.co.uk. Send the following mail message to Majordomo:

```
To: majordomo
Subject: This bit is irrelevant
approve foo subscribe test pdcawley@ftech.co.uk
end
```

where *foo* is the list password and pdcawley@ftech.co.uk is the owner of the list.

You should also send this user a copy of the file Doc/list-owner-info from the Majordomo source directory, which you have editted to take account of the list

details (these are all set at the top of the file) so that she/he will be able to administer the list remotely without having to ask you too many questions.

Of course, if you are both the system administrator and the mailing list administrator you'll need to read this file yourself.

Support for Majordomo

What do you know—Majordomo has a bunch of assorted mailing lists. I strongly recommend that you at least join Majordomo-users. To do that, send the following in the body of a mail message to majordomo@greatcircle.com :

```
subscribe majordomo-users
lists
end
```

This will subscribe you to majordomo-users and also send you a list of all the mailing lists available.

Piers Cawley (pdcawley@ftech.co.uk) is the Systems Sheriff at Frontier Internet Services, a UK company providing a bewildering variety of services for getting connected. Piers spends most of his time administering the Linux boxes that these services run on. In his copious free time (hah!) Piers has a distressing tendency to sing folk songs. You can e-mail him or you can laugh at his woefully inadequate homepage at www.ftech.co.uk/~pdcawley.

Archive Index Issue Table of Contents

Advanced search

The Pari Package On Linux

Klaus-Peter Nischke

Issue #13, May 1995

Fast math is Pari's claim to fame. Klaus-Peter Nischke introduces us to a small, fast, flexible calculator with symbolic and numerical theoretic abilities.

The pari package (named after the French capital Paris, where the idea for this package originated) is a computer algebra system designed to work under several Unix derivatives, and of course Linux is one of them. It is well-known to a small group of mathematicians, and most probably useful for anyone who wants to perform symbolic or numerical computations or who just likes to have a powerful calculator. Its features include arbitrary-precision numerical computation, symbolic calculations, matrix/vector operations, plotting facilities (text mode or X11), and tons of number theoretic functions. Pari provides an interactive interface (the GP calculator) as well as its own programming facilities and a library for using the kernel within its own C/C++ programs. An emacs lisp file (pari.el) for using the GP calculator within an emacs buffer is included in the package. Pari is not so extensive as the commercial packages Maple, Mathematica, or Axiom are, but its major advantage is its speed. Pari claims to be 5 to 100 times faster than the commercial counterparts. I personally like its very economical use of memory. It performs really well on my “low end” 386/40 with 8 meg RAM.

Pari is available by anonymous ftp from megrez.math.u-bordeaux.fr as pari-1.39a.tar.gz, together with examples, benchmarks, and a manual (160 pgs.) which includes a function reference and a tutorial. The authors are C. Batut, D. Bernardi, H. Cohen, and M. Olivier, who are well-known number-theorists. You can contact them at pari@math.u-bordeaux.fr.

Unpacking and Compiling

On megrez.math.u-bordeaux.fr, you will find precompiled Linux binaries (gplinux.tar.gz) as well as the source package pari-1.39a.tar.gz. Because it contains the documentation and the examples, I recommend getting the

source package even if you get the binaries. pari-1.39a.tar.gz unpacks into three subdirectories: doc, examples, and src. If you have gcc installed, recompiling is quite straightforward. After running configure i386 and performing a minor hack in the Makefile (read the src/INSTALL file), you are prepared to run make. You can optionally compile the gp calculator with readline support, meaning you have a command history, programmable keystrokes, and other features as within GNU bash. The source to bash also contains the necessary readline library.

It's easy to install the pari library and the gp calculator by issuing **make install** as root. Installing emacs support is a little bit tricky and requires you to edit some pathnames and constants defined in pari.el to match your configuration. Once pari.el is installed, you can start gp by issuing **M-x gp** and get an overview via **M-x describe-mode**, like most emacs modes.

A First Session

After compiling and installing it successfully, let's start gp and try a few expressions at the "?" prompt:

```
? 2*3
%1 = 6
? 4/3*5/14
%2 = 10/21
? 4.0/3*5/14
%3 = 0.47619047619047619047619047761
```

As you can see, pari tries to use exact integer and rational numbers as long as possible. As soon as you introduce one real (floating point) number, the result will be real. You may request (almost) arbitrary precision:

```
? \precision=50
    precision = 50 significant digits
? pi
%4 = 3.1415926535897932384626433832795028841971693993751
```

You may enter expressions with indeterminates

```
? (x+2)*(x^2+1)
%5 = x^3 + 2*x^2 + x + 2
```

assign variables:

```
? x=2
%6 = 2
```

and evaluate, e.g., our (x+2)*(x2+1)

```
? eval(%5)
%7 = 20
```

or compute some factorial

```
? 1000!
%8 =
402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969404800479988610197196058631666872994808558901323829669944590...
```

within milliseconds.

Number Types

Of pari's basic types, until now you have seen integer, rational, and real numbers and rational expressions with indeterminates (polynomials/rational functions). Integers can store values with up to 315,623 decimal digits. The precision of reals is controlled by the

\precision=n setting, where n is restricted to be not greater than 315623. Further, you can work with complex numbers, power series, row or column vectors, matrices, and more. You can combine these types (i.e. vectors of matrices); pari handles these using a recursive technique.

Some Functions Within Pari

In addition to the standard mathematical operations +, -, *, and /, you find transcendental and number theoretical functions, functions dealing with elliptic curves, number fields, polynomials, power series, linear algebra, sums, and products, as well as functions for plotting.

For example, you can factor numbers and polynomials:

```
? factor(249458089531)
%9 =
[7 2]
[48611 1]
[104729 1]
```

meaning 249458089531=72*48611*104729, or

```
? factor(t^3+t^2-2*t-2)
%10 =
[t + 1 1]
[t^2 - 2 1]
```

meaning *t3+t2-2*t-2=(t+1)*(t2-2)*, where *t2-2* cannot be factored further using rational coefficients. It is only possible to factor polynomials in one indeterminate.

To solve a linear equation *x=3*y, y=2*x-1* (using the gauss method), you rewrite it as *x-3*y=0*, *-2*x+y=-1*, take the coefficient matrix A, the right side b and compute

```
? A=[1,-3;-2,1]
%11 =
[1 -3]
[-2 1]
? b=[0;-1]
%12 =
[0]
[-1]
? gauss(A,b)
%13 =
[3/5]
[1/5]
```

giving you the result x=3/5, y=1/5.

To determine the roots of a polynomial you may just enter roots:

```
? \precision=4
precision = 4 significant digits
:? roots(t^3+t^2-2*t-2)
%14 = [-1.414 + 0.0000*i, -1.000 + 0.0000*i,
1.414 + 0.0000*i]~
```

Plotting gives you a quick overview of a function even in text-mode; see Figure 1. For plotting to a separate X11-window, enter:

```
? ploth(x=-pi,pi,sin(x))
```

Figure 1. A Function Plot in Text Mode

Instead, to get the graph in Figure 2, enter:

```
? plot(x=-pi,pi,sin(x))
```

Figure 2. A Function Plot in X Window System Mode

The GP Calculator

The gp commands may be classified into expressions (which are evaluated immediately), function definitions, meta-commands, and help. Via the ? key, you obtain help for the meta-commands controlling gp as well as for each of the built-in functions. The meta-commands allow you to control the way of

printing pari results as well as reading and writing from or to a file. **\w <filename>** saves your complete session (from starting gp up to issuing this command) to a file, **\r <filename>** does the reverse job, reading the session, bringing you to (or returning you to) the exact state that you previously saved. Other useful features include the writing of expressions in TeX/LaTeX format (via **texprint**) and switching the printing of timing information by the # command. You may also of course run gp as a batch job using standard I/O redirection. You span input over several lines by using the \ continuation character.

Programming in GP

Defining your own functions in gp is quite simple. As an example, cube returns the third power of its argument:

```
? cube(x)=x*x*x
? cube(3)
%15 = 27
? cube(t+1)
%16 = t^3 + 3*t^2 + 3*t + 1
```

You can use control structures as **if, while, until**, for (there are some special variants), **goto** and **label** as well as functions for printing or clearing variables. Though pari already provides a function **fibo**, let us try to program a function for the Fibonacci sequence. This sequence is defined by *f0=1, f1=1, fn=fn-1+fn-2* for *n>=2*, yielding *f2=1+1=2, f3=2+1=3, f4=5,...* The (probably) shortest such function uses recursion. Here you need the **if** expression to test for the special cases *f0=1* and *f1=1*. **if(a,seq1,seq2)** evaluates **seq1** if a is nonzero and **seq2** otherwise:

```
?fib(n)=if(n==0,1,\
if(n==1,1,fib(n-1)+fib(n-2)))
? fib(5)
%17 = 8
```

For small *n* this is okay. A faster way is to compute the Fibonacci numbers by iteration. In each step the new value *h=fn* is computed as the sum of the last two values *g=fn-1* and *f=fn-2*, and afterwards these values are exchanged. For this you need variables **f, g, h**, and **m** (counter). To avoid conflicts with variables defined outside the function, these four are declared as local by writing them at the end of the parameter list. The **for(x=a,b,seq)** expression evaluates **seq** for each value of *x* running from **a** to **b**. Expressions separated by a semicolon ; form a sequence, and a sequence's value is always that of its last expression:

```
? fib2(n, m,f,g,h)= f=1; g=1; \
for(m=2, n, h=f+g; f=g; g=h); g
? fib2(5)
%18 = 8
```

Conclusion

This article can only cover some of pari's basic aspects and highlights. For a more complete description and for using the pari library within your own programs, the manual gives you an excellent reference.

```
? \q
Goodbye!
```

Klaus-Peter Nischke has been an enthusiastic Linux user since 0.99pl13 (1993). He works at a small software company, and has worked at the local university as a mathematician. He uses pari for his own use and for mathematical research. He can be reached at klaus@nischke.do.eunet.de.

Archive Index Issue Table of Contents

Advanced search

Netsurfing With Linux

Arthur Bebak

Issue #13, May 1995

To sail the cyber sea.

Purportedly, this article is about how an “obscure” operating system called Linux was used to launch a happy band of netsurfers into the wild ocean of the Internet. But it is really a rousing tale of adventure and discovery—with Linux playing the part of a trusty ship. That ship enabled us to chart a map of the vast Internet ocean, in the guise of *Netsurfer Digest*, a free, sponsor-supported e-zine (electronic magazine), serving as a gateway to on-line adventure for netsurfers all over the world.

It all started in the spring of 1994 when I gathered together a small band of netsurfers and, through the proper application of persuasion, hand waving, and free food, convinced them to put together an on-line publishing company. Following venerable startup tradition, we scribbled on numerous coffee house napkins, ate lots of pizza, exchanged tons of e-mail, and gave birth to *Netsurfer Communications*. Our goal was to publish interesting and user-friendly e-zines using the hot new technology of the World Wide Web.

Being a fairly on-line literate bunch, we decided that our products would be made available exclusively on-line, and that the company would be run as a virtual corporation. We intended to take every advantage of the vast leverage provided by modern communication technology. By existing entirely on-line, we would be able to effortlessly communicate with our consumers, tap a global pool of talented contributors, and keep our overhead to a minimum.

We Need a Ship

While prototypes of our flagship publication, *Netsurfer Digest*, were being prepared and the production process was being designed, a number of technical decisions had to be made. One of the most important was the choice

of operating system for our production facilities and for our Internet site. Whatever we chose had to meet a number of fairly stringent requirements.

First, we needed a very reliable e-mail platform. Our system had to be able to support mailing lists serving thousands and provide reliable e-mail storage for our internal editorial communications. Second, we needed a reliable World Wide Web and FTP site. Back issues of our e-zine as well as various background information had to be made available to people all over the world, at all hours of the day and night. We also needed the ability to easily change and update this information, sometimes by automated scripts. Finally, it was important that whatever environment we chose supported good development tools. We planned to create a number of custom programs to aid in production and distribution of our e-zines.

It didn't take a genius to figure out that some flavor of Unix was what we needed. After all, Unix is the native operating system of the Internet and may well be the best development environment ever designed. The only question was which brand of Unix to go with? There were a number of commercial versions available running on expensive workstations or feature-loaded PCs. The key word there was “expensive”. Now, if you've been involved with startups, then you know that next to the occasionally scrambled brains of the startup team, the most precious resource is cash. You only spend it on items absolutely essential for keeping the venture going, such as marketing, hardware, and pizza. This was definitely on our minds when the time came to choose our operating system.

It just so happened that at the time I had a copy of Linux available, which I had purchased on CD-ROM from Trans-Ameritech. I had played around with it at home and had also heard good things about it from my friends. It appeared to be a full-featured, relatively robust operating system which might be able to meet our needs through the early stages of the e-zine. The price was right, and what's more, we did not need some super-expensive machine to run it. We had little to lose by giving Linux a try. If it worked, we had a very inexpensive solution to our requirements. If not, well, we could always go with one of the more expensive commercial operating systems, something we figured we'd have to do anyway as our enterprise grew.

Trimming the Sails

It was clear that Linux was a nice stand-alone operating system, but we needed to find out if Linux could reliably support an Internet site. Early in May of 1994 the big day came. I had already installed Linux version 1.0.9 on our machine, a humble 486DX33 PC with 8 meg of RAM, a 245 meg drive, and an Ethernet card. The first priority was to see if we could hook it up to the Ethernet network at our provider site and, from there, work on getting it on the Internet. So I lugged

the machine across the San Francisco Bay to Berkeley and sat down with our netmaster, Bill Woodcock, to install it on his network. I was prepared to spend a few hours fiddling with the system to get it running. I even brought some snacks to munch on as we worked during the afternoon on getting the whole contraption to work.

First, we read the instructions in the Ethernet-HOWTO, which essentially said to make sure that the kernel had been compiled with support for our Ethernet card. No problem; I had already done this. Next we read the NET-2-HOWTO, which told us how to configure the TCP/IP network. This boiled down to either running a utility called ifconfig or changing a few well commented lines in the rc.inet1 and rc.inet2 files. It seemed deceptively simple, and after we made the changes, Bill and I looked at each other skeptically and rebooted the machine.

I've spent all my professional life working with complex hardware and software systems, first as a mainframe design engineer, and later, as a software manager. In my long experience, I've learned that the first time you test a new piece of software, turn on new hardware, or configure a network, it never works. Never. There is always some fiddling and adjusting, or even bug-fixing, which must be done before the whole thing works vaguely the way it was designed to. That's just the nature of the beast. Imagine my consternation when the machine came up, recognized the network, and responded to pings from the rest of the world. This just does not happen in the real world. I was, frankly, stunned and amazed. But in a good way.

In short order, we brought up the standard daemons and had telnet, FTP, and e-mail going between our machine and the rest of the network. What I thought would be a long afternoon of debugging and digging through obscure on-line documentation turned into a half-hour job. We took the rest of the afternoon off and went out to get some pizza. I even sprang for extra sauce.

Bill spent the next few days configuring the machine to our liking. He arranged domain name registration, set up secure FTP and WWW software (all freely available on the Net), wrestled with e-mail configuration, and set up the necessary user accounts. We were on the Internet and ready to support beta testing of our first e-zine, *Netsurfer Digest*.

The Sea Trials

Throughout the summer of 1994, the Linux box supported our beta effort, which eventually grew to over 1500 users getting copies of our e-zine twice per week. At the same time we obtained another PC to be used for development and production. This was outfitted with X-Windows as a user interface, Perl as the primary development language, and Seyon as the telecommunications program. Work proceeded on developing both our editorial process and on

various support tools. So far, Linux was proving to be a very cost-effective choice and quite flexible as a beta test platform. However, our voyage was not entirely smooth sailing.

Early during the Beta test period, we had problems handling the continually-growing mailing lists. Our problems were caused by an interaction between our mailing software (smail at the time), the relatively slow machine, and a slow Internet connection (28.8K modem). In short order, we located another mailer (sendmail) at a Linux FTP site, wrote some Perl scripts, and had a working mail configuration. Linux's flexibility as a development environment was starting to pay dividends.

Of greater concern to us was stability. One of the things we noticed was that the machine was not as stable as we would have liked. It appeared that, for one reason or another, we had to reboot two or three times per month, on average. Frequently, this was due to random segmentation exceptions, or other obscure errors, which would hang-up our machine for no discernible reason. We could live with these occasional glitches during our beta test period, but clearly, this would not do for a production machine which had to reliably serve thousands of readers. However, since these glitches did not seriously impact our beta testing effort, we could put up with them for a while, given the economical nature of the software in particular. We were a happy and cheap startup.

By the fall of 1994, we were on the eve of a formal launch of *Netsurfer Digest*. Our mailing lists were growing larger and our Internet site was being accessed more and more often. As part of our effort to prepare for our public debut, one of the things we did was upgrade our copies of Linux to the then current version of 1.1.61. We noticed immediately that the new version appeared much more stable than the earlier 1.0.9. This has, in fact, proven to be true over the long term, with the latter version of Linux taking us smoothly through the early days of our commercial existence.

Steady as She Goes: Hauling the Cargo

Throughout the winter of '94-'95, Linux performed flawlessly as our Internet and production platform. Our subscription rates, and therefore our weekly mailings, were growing at a furious pace. We reached the point where it took over 24 hours to mail out one issue to our thousands of direct subscribers. The response time on our system did slow during a mailing, but Linux took the load and completed each run.

The accesses to our WWW server also started taking off. It seemed that every week we were adding another thousand hits to our daily access statistics. The real test came in early January, when we created a special Macintosh issue to

coincide with Macworld Expo. Overnight, accesses to our poor 486DX33 PC topped 10 thousand per day as Macintosh aficionados overwhelmed us with accesses. Linux met the load with graceful performance degradation, doing just what a good operating system should do. There finally came a point where our faithful system could take no more and the machine hung. We had pushed our system to its limits.

By the end of January 1995, Linux was routinely handling over 16 thousand accesses to our Web site every day, with peak loads reaching almost 20 thousand hits per day. At the same time, we were handling mailing lists numbering over 15 thousand subscribers for two different e-zines. It was mind-boggling that this free system on a relatively puny machine could do all this. We definitely got value for our money.

Larger Oceans, Bigger Ships

We realized in late 1994 that there would come a time when we would have to upgrade our systems to handle the loads imposed by our dizzy growth rate. Simple math told us that, at some point, our weekly mailing would impose such a load on the system that people accessing our site would not be able to get through. Beyond a certain point, a 486 PC, or even a Pentium, would run out of horsepower. To handle tens of thousands of subscribers and tens of thousands of daily hits on our Net site, we would need to upgrade the speed of our feed, arrange for professional mirror sites, and get some serious hardware.

Fortunately, this does not mean that we will abandon Linux. All of our production and development still takes place on a Linux PC workstation. Our current machine, now a veteran of a long and eventful year, will remain hooked up to the Internet as a backup resource. Linux has proven itself in a very demanding environment. We have pushed it to its limits and occasionally beyond. We would not be where we are today without this wonderful operating system, and by extension, without all those who made it the great and powerful tool that it is today. We can only hope that by publishing our free e-zines, by making them entertaining and useful, we can return something to that vast Internet community without whom we could not have our dream jobs.

Arthur Bebak is a veteran of the Silicon Valley computer community, with a background in computer engineering and project management. As founder and publisher at Netsurfer Communications, Inc. he currently tries to stay out of the way of his talented staff as they create some of the best netsurfing e-zines currently available. Arthur can be reached at arthur@msm.com. To see what Linux has helped create you can visit the Netsurfer Digest home page: www.netsurf.com/nsd/.

Archive Index Issue Table of Contents

Advanced search

Linux for Public Service

Dan Hollis

Issue #13, May 1995

JCICNet, an Internet Services Provider, uses Linux to serve its customers. Here's why.

I'm the system administrator for **JCICNet**, an Internet Service Provider (ISP) that provides public access Internet services for Southern Oregon. This article explains why we use Linux and describes some of our experiences with it as an ISP.

These days, one of the hot topics seems to be using Linux as a base for public access Internet systems. There are a number of reasons why, for many ISPs, Linux is the operating system of choice.

Taking the Plunge

When our ISP first decided to provide Internet access to the local area, my first task was determining what kind of hardware was needed. The basic requirements for the system were:

1. It must handle multiple dial-up users smoothly,
2. It must be seamlessly integrated with the TCP/IP Internet protocols,
3. It must be easily expandable and easily modified for our various needs, and
4. It must be very inexpensive (we are a relatively small ISP).

For me, the answer to the first three was obvious—Unix. Requirement number four mandated the hardware be on a PC clone. But which version of Unix?

I had heard about Linux, but the biggest question was: if it's free, how reliable could it possibly be? There was one way to find out. I read the FAQs, the HOWTOs, and the comp.os.linux newsgroups. Although I didn't really have a clue what was being discussed in the newsgroups, I started asking questions. I

never received anything negative about its reliability, so I figured that even if it wasn't suitable, at least we wouldn't be out any money, so Linux was given the green light.

We eventually settled on the following configuration:

- DX4/100 with 16mb RAM
- Buslogic SCSI Adaptor
- Quantum EMPIRE-1080S 1 gig SCSI drive
- Toshiba XM-3401TA SCSI CD-ROM
- QIC-117-compatible tape drive
- BOCA Board 2016 (16 port serial card, cables not included!)
- Supra 28.8k and AT&T Paradyne 14.4k modems

Our network provider sold us the following equipment:

- Morningstar Express Router
- BAT Technologies 56k CSU/DSU

All told, the total hardware cost came to close to something like $10,000.

Setting It Up

Our Linux distribution was the Infomagic 2-CD set, from which we installed Slackware. The installation was relatively simple; everything was menu driven with very little guesswork. Within hours we were fully installed, up and running. Very impressive.

The network was all configured thanks to the menu driven installation. I never had to touch configuration files. We just configured our router, and we were on the net.

Next came the **BOCA** serial board. Using **setserial** and /etc/rc.d/rc.serial, configuration was painless. I just uncommented a few existing lines and the BOCA would auto-configure itself on bootup. I installed **mgetty**, and within an hour, we had working dialups.

Last came the front end that the users would have to tangle with. Since most of the users would not have any experience with the shell, I used the Dialog package (documented in *Linux Journal* issue #5) to whip up a spiffy menuing system using shell scripts.

All the while, I was amazed at how straightforward everything was, all the pieces to the Linux puzzle fit together neatly. I had experience with various

Unices in the past, and Linux was by far the easiest to come to grips with. Packages just plugged in and started working, with little or no configuration required. A lot of this I attribute to Linux's **FSSTND** (File System Standard)--a boon, especially to system administrators.

Bringing the System Up

With everything configured, we were confident enough to throw the switch and start letting users pour in. I watched the system closely for hours, all the while expecting this “free operating system” to explode into a million pieces, taking our hardware and subscribers with it into /dev/null.

Several weeks later, I stopped worrying. During our first few weeks the system ran smoother than I had ever expected. No kernel panics, no strange file system problems. It wasn't flawless by any means, but none of the problems were insurmountable or disastrous. For a free operating system, I was very happy with Linux.

The first real problem I ran into was with hanging processes. We were having a bizarre problem where, if someone dropped carrier without logging out cleanly, processes were left hanging on `con'. It can be very disconcerting to log in and find the system load at 26.0 or so! This took about a week to track down. I asked various questions on the newsgroups and IRC. I eventually ran across another system admin on IRC who had the exact same problems. We narrowed it down to the POSIX behavior of bash; replacing bash with zsh solved the problem.

The second problem we encountered was with our Supra modems crashing occasionally, which was (thankfully!) not a Linux problem. I would have to power cycle the modems to get them to wake up again. After several calls to Supra tech support, it was revealed that if the modem received characters while in the middle of a hard reset (via an `ATZ' or DTR drop) it could crash. A quick change to the initialization string and the modems cleared up.

Our third problem was a bug with mgetty. It turns out that mgetty will happily echo back `+' characters to your modem. If someone logs in and sends three `+' characters in a row, mgetty echoes those back to the local modem which, of course, puts it in command mode. Then mgetty will happily sit, blabbering away in a chat loop with the modem. A small one-line modification to mgetty (which would not have been possible without the freely available source code!) fixed this.

Our final problem was with **smail**. It simply would not send to hosts on the Internet that had an MX pointer in their domain name server (DNS) record. I finally figured out that the Slackware distribution of smail did not have the **bind**

driver compiled in. I downloaded a newer version of smail off of ftp.uu.net and it worked flawlessly.

Since then, the system has been relatively painless to administrate. When our net provider's DNS server started flaking out, I got my own DNS server up within an hour with the help of Douglas Rabe (darabe@templar.fgi.net), a friendly Linux admin from Illinois. (He also offered to act as our DNS secondary).

Linux Support

Support on the net has been great. The comp.os.linux.* newsgroups carry amazing amounts of information. IRC is good, except for a couple sour grapes: it can be very disappointing on IRC to see someone answer a newbie's question "how do I untar a file?" with **alias tar `rm -rf'** and then type **tar filename**. To me, that is one more person permanently turned off of Linux due to someone's lame joke. Nobody said newbies had to ask dumb questions, but then nobody said smarti-alecks had to answer them, either.

The software support for Linux has been outstanding. It seems like any new software package, no matter where I ftp it from, has a specific makefile configuration for Linux. And thanks to FSSTND, most of it plugs right in.

There are a number of reasons why I personally prefer Linux. Since the source code is available for everything, if something is wrong, I can fix it myself—instead of spending hours on hold with a software vendor who may or may not fix my problem (or even understand what the heck I'm talking about!). Have you ever spent hours on hold with a vendor only to get a support person who is perfectly ready to help you—with your VMS problems?

Another major point with Linux is that user licenses are a non-issue—something which has continually come up and bitten us with other commercial operating systems. The user licenses issue is something that should be of great importance to an Internet Service Provider.

From home, I administrate the main dial-in system with—what else—my own Linux box. Since I deleted DOS and Windows from my hard drive and installed Linux, I've never been happier with my PC.

Daniel Hollis is a systems programmer for Pharmacy Computer Services, where he is trying to find ways to slip some Linux boxes into the workplace. He is also the system administrator for the jcic.org domain and is currently involved with authoring the Linux-Public-Access-HOWTO. He can be contacted via e-mail at dhollis@hq.jcic.org.

Hamming It Up On Linux

Brian A. Lantz

Issue #13, May 1995

In an effort to corrupt the readers of *Linux Journal*, Brian asks you to take a pair of hypothetical binoculars (preferable of the “X-ray” genre so popular in old comic books) and become a “Peeping Tom” with him, looking into two very different homes.

The first house has something very different about it, seen even without the binoculars. You can see a 50-foot tower looming high into the sky. On the structure are scores of antennas in every size and shape imaginable. As you “peer” into the den of the house, you see more electronic equipment than you would expect to find at the local radio station. There are dials, lights, LEDs, and meters of every kind. There are the sounds of Morse code, static, and eerie-sounding human (human?) voices. The man at the equipment has a microphone in front of him, and in his right hand there is a telegraph operator's key.

Now, moving to the second house, we see a typical home, like you'd expect in Anytown, USA. Nothing unusual as we approach it. So get out the binoculars! (Come on, get into the shadows. Someone will see us!)

In the den of this home, we find an ordinary-looking man sitting at his personal computer. His eyes seem a bit bleary, but he seems to be enjoying whatever it is that he is doing. Looking closer, you can see that the operating system is obviously Linux, as he swaps virtual consoles at the speed of light. But then he stops on one VC which seems to be an IRC-type chat group, but instead of “handles”, each VC is identified by a cryptic series of letters and numbers...

Okay, put the binoculars down. What do these two men have in common? They are *both* amateur radio operators. And amateurs have been migrating to Linux in remarkable numbers. In this article, I'll try to give the non-amateur a feel for why this is occurring, from an insider's viewpoint (yes, I'm one of them: KO4KS is my amateur radio callsign).

What is the Amateur Radio Service?

Without getting into the nitty-gritty details of it, the countries of the world cooperate together to provide radio spectrum for use by licensed operators for experimentation, public service, and just plain fun!

Amateurs have a wide variety of radio-related ways to enjoy their hobby. There is the traditional Morse code, shortwave, as well as local voice communications. In addition, among others, there are satellite communications, participation with the space program, and something called packet radio.

What is Amateur Packet Radio?

Most of you have probably seen e-mail addresses in the “ampr.org” domain. These are addresses of amateur radio operators, using packet radio. But how did they get on the Internet? Don't get ahead of me....

Packet radio is described by some as Ethernet over radio (though Ethernet used to be described as packet radio over wire). Anything you can do over Ethernet can be done over packet. Packet operators and programmers have been highly involved in many Internet RFCs as they have found ways around the problems encountered in using radio in this manner.

Packet radio uses a variation of the X.25 protocol called AX.25 (Amateur X.25). Their radios are connected to their computers with a little packet modem.

Initially, the “killer app” of packet radio was the Packet BBS (PBBS). PBBSs are pretty crude by today's BBS standards, but they pass messages internationally by means of thousands of individually owned radio sites that relay the mail, hop by hop along the way. Some packet PBBSs have HF (“High Frequency”--which, paradoxically, is at the very low end of the frequencies used for packet radio) ports for passing the mail over longer hops, but most hops are only a few miles each.

Then, several years ago, Phil Karn/KA9Q wrote a program called NET, to incorporate AX25 and TCP/IP together on computers running MS-DOS. As time progressed, NET became NOS (Network Operating System), which was now a multi-threaded application allowing multiple incoming and outgoing sessions. I am told that some commercial manufacturers have licensed the KA9Q code as a base for their development, and many of the new and modified protocols have made their way into commercial wireless TCP/IP networks.

Phil made his code available freely to amateurs and for educational purposes and even has allowed variations to spring up. There are many “xNOSs”, such as

WNOS, JNOS, PMNOS, etc. and others that I'm sure I missed. I'm a little partial to Tampa NOS (TNOS), since I am the one who has developed this variation.

This interest in TCP/IP leads very naturally to Linux, but even more so when you consider the limitations of a program like xNOS under MSDOS. Good ol' DOS doesn't have a linear memory map, TCP/IP or multitasking built in. And it *does* have the 640K conventional memory barrier. It also requires dedicating a computer to xNOS under DOS.

Packet Radio Meets Linux

Linux is a good move for xNOS, since the DOS-related restrictions are gone. A couple of xNOSs have already been ported to Linux, including TNOS. Those interested can find it at ftp.lantz.com:/tnos/current/linux. For Web slingers, try www.lantz.com.

NOS under Linux is still a large program, with all of the same features as the DOS version, including its own TCP/IP stack. You can connect TNOS to the Linux TCP/IP stack by means of a ptty SLIP or PPP link.

Amateur Radio TCP/IP is passed over the *real* Internet by Amateur gateways. These gateways exist both on the Internet and on the local packet network. Other non-gateway Amateur packet stations can pass packets through the gateway to Amateur stations on the other side of distant gateways. This is done by encapsulating the 44.xx.xx.xx addresses (addresses assigned to the ampr.org domain) into “real” Internet addresses for passing through the backbones.

An Amateur TCP/IP Gateway Tour

In visiting a TNOS system, you will find many things that are familiar. Various core Internet protocols are supported, such as FTP, SMTP, POP, ICMP. Many familiar applications, like **finger**, can also be found.

You will find things that are new, too. A telnet connection, rather than a login to a shell program, will find you placed in a Packet Bulletin Board System. You will find specialized information servers and other goodies.

And then there are conference bridge applications. These are the amateur radio equivalant of IRC. You could be “chatting” with amateurs around the world.

I said “could” because most Amateur Radio gateways are secured very heavily: partly for the same reasons that *you* would secure your system on the Internet, and also because Amateur Radio operators have certain laws to which they are

held accountable regarding the content of the information transmitted over their radios.

What Next?

AX.25 support has been added to the Linux kernel, and it is maturing quite well, thanks to the efforts of Alan Cox and several others. As this becomes more and more usable, the large xNOS applications will probably become history, with their unique functions becoming stand-alone daemon programs which will run in the background. Several TNOS-specific features have already been made into stand-alone programs. Additional Internet services will become readily available via Amateur Radio, e.g. Gopher, World Wide Web, NNTP, etc.

So, expect to see increasing numbers of Amateur Radio operators coming to Linux. You can take this as a welcome addition, as they are some of the most innovative and technologically sound individuals to be found. Now, when you read magazine articles that say that both the Internet and wireless technologies are on the increase, maybe you will have a better understanding as to why.

Brian A. Lantz has been a software developer in embedded and multi-user systems for the last 15 years, specializing in device drivers, communications and networking. Aside from being employed by Utility Partners, Inc., in Tampa, FL, he spends most of his idle time hacking on Linux, helping with the local Amateur Radio network and running an amateur radio gateway to the Internet. Sleeping is attended to occasionally. He welcomes your comments, sent to brian@lantz.com.

Archive Index Issue Table of Contents

Advanced search

A New Project or a GNU Project?

Mark Bolzern

Issue #13, May 1995

How old is Linux? The Linux project was started about four years ago, but the Linux distributions are really far more than four years old. Linux is the product of many years of hard work.

Everyone knows that the Linux operating system was created from scratch about four years ago, right? Actually, while the Linux project is itself only four years old, it is important to remember that large parts of the Linux system on your desk are far older.

Linux is not an entire operating system—it is an operating system kernel. Any Unix or Unix-like operating system kernel requires a large suite of utility programs to complete the entire operating system. The Unix kernel has historically been distributed with many text manipulation tools, a C language compiler, programming libraries, and on-line documentation for all (or almost all) of the utilities and libraries. All of this together comprises the operating system.

Some History

Years before Linux was started, programmers who were fed up with bugs and limitations in the standard versions of Unix utilities started to write their own versions and contribute them to the Free Software Foundation's “GNU” project. Slowly, over the years, programmers noticed bugs and limitations in nearly all Unix utilities and wrote replacements for them, freeing the users from having to rely on software vendors to be consistently perfect.

These replacements were ported to most “flavors” of Unix that were available—but they all still required a commercial kernel, and the users were still dependent on their Unix software vendors to fix bugs and remove limitations in the kernel. The Free Software Foundation (known as the FSF) started work on a replacement kernel (now called “The Hurd”), but work was slow, original efforts

met with legal challenges, and although the legal questions have long been settled, The Hurd has still not been publicly released.

More Recent History

In the spring of 1991, Linus Torvalds, a college student in Finland, started writing a 32-bit terminal emulator for his 386. He wrote it in a combination of assembly language and C, and in order to learn more about the 386 chip, made the emulator completely independent of DOS or any other operating system. Over the next few months, he learned enough to turn his terminal emulator into the beginnings of a Unix-like kernel.

To compile his code, Linus used the FSF's gcc C compiler and assembler. As a shell, he used the FSF's bash ("bourne again shell"). For the **make** program, he used the FSF's GNU make. In short, he and the team that quickly assembled around him used nearly all of the FSF's GNU programs to complement the Linux kernel and assemble a complete operating system.

These pre-tested and refined utilities turned a Unix-like kernel into a Unix-like operating system overnight. Many users who are not very familiar with the Free Software Foundation's suite of utilities are under the impression that the entire Linux operating system has been developed completely from scratch in the last four years. In reality, Linux leveraged (to use a current buzzword) a huge amount of tested, quality code to turn a kernel into an operating system, virtually overnight—the GNU project has enabled the Linux operating system to take the world by storm. Most of what's really new is the name and the kernel —much of the rest is GNU.

This does *not* belittle the kernel. It is large and complex, and there are few people who have the combination of qualities required to write—and especially maintain—one. It merely points out that Linux is in reality not a four-year-old newcomer, but a respectable, mature option for Unix users.

This new, powerful combination of the FSF's GNU utilities and the Linux kernel has in turn contributed to the quality of the GNU utilities. Because Linux users rely exclusively on the GNU utilities, when they find a bug in a GNU utility, they cannot simply fall back and use the version of the utility that came with the system. This means that more people are working on fixing those bugs that still exist in the GNU utilities, making them even more stable than they used to be. Those bug fixes (and enhancements) are usually given back to the FSF, as is appropriate.

Other Contributors

Not all of the utilities that Linux uses come from the GNU project. The University of California at Berkeley had a long-running project which produced the Berkeley Software Distribution of Unix, known simply as BSD. Since much of the Unix networking software was developed by and for Bed's distribution of Unix, the BSD networking utilities still define the standards, and so many of the Linux networking utilities are time-tested utilities from BSD. As another example: thanks to the work done by MIT and the corporate members of the X Consortium, Linux had a well-tested, respected GUI less than a year after the project started.

True Innovation

Linux, as innovative as it is, has been created from “off-the-shelf” technology—it has gone fast and far on the shoulders of giants. It has employed (for the most part) proven technology to give users a truly open operating system.

The most important innovation Linux has given users is not the code, but the license. Because Linux is licensed under the FSF's GNU Public License, users have control, and are not tied to a vendor's apron strings. This creates a market for support, and users who are dissatisfied with the support they are receiving have the option to switch support providers. Not only is there a market for support, but users are also enabled to fix problems and enhance the software themselves, if they are so inclined.

Neither of those options truly exists for operating systems that do not include source; those options demonstrate the true innovation of Linux.

Mark Bolzern is the President of WorkGroup Solutions, Inc, and a Board Member of Linux International.

Archive Index Issue Table of Contents

Advanced search

Linus Torvalds Receives Award

LJ Staff

Issue #13, May 1995

The First Annual “Excellence in Programming” Awards Presented by *Dr. Dobb's Journal*

San Francisco, February 16, 1995—In conjuction with its 20th year of publication, *Dr. Dobb's Journal* magazine honored outstanding achievement in the field of computer programming by hosting the first annual “Excellence in Programming” awards at the Software Development `95 conference. Awards were presented to Alexander Stepanov, for his development of the C++ Standard Template Library (STL) and Linus Torvalds, creator of the Linux operating system. Richard Guggenheim, honorary council of Finland, was present to accept the award for Torvalds.

In his presentation, Phil Hughes, president of Specialized Systems Consultants and publisher of *Linux Journal*, discussed why Linus should receive this award for the most innovative software product of 1994. “Lots of software was designed in 1995. I see the key here as being innovative. Innovation can consist of development of a new type of product. For example, Multiplan, the first spreadsheet was innovative,” Hughes said.

He continued, “Linux is innovative in a very different way. While Linux is another Unix-like system, its development has been done in a totally unique fashion. Linus started the development effort but managed to continue his work in such an open manner that it made possible and even encouraged others to get involved in the effort.

Thousands of individuals have contributed to the effort to make Linux what it is today but Linus gets credit for the innovation. After all, he did manage to get these thousands of people to work on the development of Linux and the serious innovation is that he managed to get them to do all the work for free.”

“The recipients were chosen based on their contribution to an area of software development that has significant impact on programmers,” said Jonathan Erickson, Editor-in-Chief for *Dr. Dobb's Journal*. “The editorial committee was formed last year to look at and promote significant innovations in software development,” explained Erickson.

Mr. Torvalds almost single-handedly implemented true innovation in operation-system kernel design while achieving 100 percent UNIX System V compatibility —something even commercially-available systems have not accomplished. Torvalds is a student at the University of Helsinki in Finland.

Richard Guggenheim accepts the award for Linus Torvalds from Jonathan Erickson.


The whole gang: Alexander Stepanov, Peter Hutchinson, Jonathan Erickson, Richard Guggenheim, Phil Hughes, Bjarne Stroustrup.


Phil Hughes delivers another pithy speech.

Archive Index Issue Table of Contents

Advanced search

The Linux Sampler

Harvey Friedman

Issue #13, May 1995

Overall, this is a worthwhile book, even for those who have been subscribing to *LJ* since Issue 1.

- Editors: Belinda Frazier and Laurie Tucker
- Publisher: Specialized Systems Consultants, Inc. (SSC)
- ISBN: 0-916151-74-3
- Price: $14.95 (USD)
- Reviewer: Harvey Friedman

The Linux Sampler? “What can that be all about?” I asked. After reviewing a copy, I can say that SSC has produced a useful “Linux Resource Guide” of “Resources, Technical Information, and Articles from *Linux Journal*”, with an emphasis (as we might expect) on *Linux Journal* articles.

To help demonstrate the organization of the book, I will list the table of contents:

1. An Introduction to Linux
2. Linux in the Real World
3. Talking about Linux
4. Politics, Opinions and Future of Linux
5. Linux and Other Operating Systems
6. Technical Articles
7. Systems Administration

Also included are a Glossary and an Appendices. Each of these seven main sections has anywhere from three to six articles, selected mostly from *LJ*. A nice feature of this organization is that one doesn't have to leaf through seven or eight issues of *LJ* to read about one of the topics above. For example, two

interviews with Linus Torvalds that were published in issue 1 and issue 9 are placed back to back in chapter 3 and Chet Ramey's bash articles from issues 3 & 4 are a single seamless article in chapter 6.

Chapter 1 is comprised of two parts. The first three articles discuss the history —albeit in a broad overview—of Linux and will probably be used as background for many other publications. The next three articles are more of the what-is-available-for-me-now type and will be dated in a few years (I hope not sooner than that).

Chapter 2 has brief case studies of how Linux systems either replaced older mini-computers and improved upon their functionality and—user satisfaction— or how Linux systems solved new problems. These are well-written and should prove useful if one is trying to get a supervisor to authorize using Linux at work.

Chapter 3 has interviews with Linus Torvalds, Slackware's Pat Volkerding, and DOSEmu team leader James MacLean. I enjoyed the interviews, but I already enjoyed them in the various issues of *LJ* as well.

Chapter 4 seemed lacking to me. Although each of these articles was interesting in *LJ*, I feel that they lose something when brought together. Perhaps because the content of each article stimulates thinking, dreaming and imagining about the future, the effect is lost when they are read sequentially. I suggest reading them at least a sleep apart.

Chapter 5 is another useful one. An article comparing memory and disk requirements for Linux, Windows NT, and OS/2 appears to be a thorough evaluation. The remaining articles discuss emulators to run other code on Linux —iBCS2, which runs binaries for other Intel-based Unices; mtools, not really an emulator, but a set of tools for accessing MS-DOS file systems (particularly on floppy disk) from Linux; Wine, an MS-Windows emulator that is still in test mode; and SAMBA, an implementation for Linux of the SMB protocol (used by LanManager, Windows for Workgroups, Windows NT, and OS/2). SAMBA was the only one of these that I had not heard of before I read about it in *LJ*.

Chapter 6 contains articles on porting other Unix applications to Linux, the GNU C Library, and bash. These are all pithy but meaty articles, yet I do not like this section being called “Technical Articles”. Perhaps something like “Tips for Programmers” would have been a better title.

Chapter 7 barely scratches the surface of system administration, but what is there is well-written.

The Glossary is generally good, but I have a few quibbles: e.g.: “Cisco A brand of router”.

In the Appendix are found “Linux Products and Services Directory”, “Linux User Groups”, and “Linux Resources”. These are fine now, but some of the references have to be updated already. For example, in the list of Usenet usegroups, page 211 doesn't reflect the most recent division of the comp.os.linux newsgroups. Therefore, the newsgroup comp.os.linux.setup, one invaluable for those new to Linux and Unix, is not included.

Overall, this is a worthwhile book, even for those who have been subscribing to *LJ* since Issue 1. The collection of related topics, the good glossary, and the list of newsgroups, ftp sites, products and services, all together in one volume, make it a source that you want next to the Network Administrator's Guide, Linux Installation & Getting Started, and the HOWTOs. This reference (which could also be named The *Linux Journal* Sampler) does nothing to tarnish SSC's reputation.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #13, May 1995

Readers sound off.

Part-Time redefined

Although I'm convinced you selected BBSs for your list because they represented the best physical distribution, I am also honored and delighted in seeing the Part-Time BBS in the first column! Thank you!

The column also prompted me to update the files on the system—I figured if you were so kind with your words, I had better at least respond with some kind of positive action. So the Slackware 2.1.0 distribution is now on-line and available for download.

In spite of the name, Part-Time BBS is available 24 hours-a-day, 7 days-a-week, and since I've become interested in Linux, it has been extended to be a repository for Slackware. I am also (if I can ever find the time) exploring ways to actually convert what is already on the BBS (there's a lot more than just Linux) to a BBS running under Linux.

Thanks again!

—Tim Gales, tgales@empros.com

Bad review

Dear Sir,

Vince Skahan's review of the book Making TeX Work [December, 1994, Issue 8] was very unfair. He clearly does not like either TeX or emacs and so it is not surprising that most of his comments are negative. He seemed not to be sure for whom the book is intended. Well, I can tell you that; it is intended for people like me. I have The TeXbook and I have done quite a bit of document writing (I

have even set tables with \halign, wow!) but I have gotten to the stage where I need more information about the tools that surround TeX and how they fit together. This book is just what I need.

Vince was “hoping they'd pick an editor (even [ugh...] emacs) and give some real details regarding how to hook in and use [it] to efficiently write TeX documents” but the book explains about the tex and AUC-TEX modes in emacs and how to implement the edit-run-edit cycle within emacs. What more does he want? The font chapter contains “more than any sane person would want to know about font selection and generation”. Vince may not be interested in fonts but lots of people are. I have recently gotten involved with dvi drivers and I am very much interested in font selection and generation.

He ends by recommending to your readers that they save their pennies, learn SGML and get the rest of what they want off the net. I suppose you can get everything off the net if you spend long enough searching but, like most of your readers, I don't have time to waste and I am happy to spend $30 for Norman Walsh to put the hours in on my behalf and collect the results into a book. Why else would your reviewer have 19 O'Reilly books on his shelf? This was a shabby review and you chose the wrong person to do it.

Yours, etc.

—Tony Sumner, A.Sumner@reading.ac.uk

Mathematica & Linux

Dear Sirs,

As a user and fan of Linux, an admirer of Richard Stallman, and also as an employee of Wolfram Research, Inc., it distressed me greatly to read a comment about Mathematica that Richard Stallman wrote in a news group. A fellow employee responded to it and has given me permission to forward his response to you.

—Philip J. Wall, philw@wri.com

Wolfram Research, Inc.

The original comment and its response follows:

In article 199502112331.SAA24982@pogo.gnu.ai.mit.edu, rms@gnu.ai.mit.edu (Richard Stallman) wrote:

> Reportedly Wolfram Research has decided not to support Mathematica on GNU/Linux systems, because users would be able to change the kernel to work around certain deliberately inserted bugs designed to make Mathematica crash in some circumstances....

There is no truth to the suggestion that Wolfram Research has made any decision not to support Mathematica on GNU/Linux systems. In particular the idea that there are any "deliberately inserted bugs designed to make Mathematica crash in some circumstances" is complete nonsense. Mr. Stallman should probably check his information before posting.

Wolfram Research is in fact in the process of testing a version of Mathematica for Linux. If everything goes according to plan it will be shipping by the second quarter of this year.

For further information about availability of Mathematica for Linix, contact Wolfram Research at info@wri.com.

I hope this clarifies things.

—Ian Collier, ianc@wri.com Wolfram Research, Inc.

We goofed!

Just a note about a type-o in your March 1995 Kernel Kornel. On page 52, column one, paragraph 4.

You use request_irq() instead of request_dma() and you wrote "IRQ channel" instead of "DMA channel".

I don't think this caused any confusion because it is evident that you were talking about DMA.

Anyway, just thought I would bring this to your attention. Besides it gives me a chance to say keep up the good work.

—Don Hiatt, hiattd@mcs.com

Michael Responds: ACK! You are absolutely right. Mea culpa.

Logging more friends...

I've read the System Administration article ("How to log friends...") in the March edition of *Linux Journal* and found it quite useful. But I think the following

remark is useful, too: if the logging level is set to =something, i.e. =debug, then only messages with this level are logged.

By the way, the syslog configuration can be tested with “syslog_tst”, which I found in /usr/sbin on my Slackware 2.0.1-based system. I do not find a man page for it, and there is no mention of it in the syslog man page.

Kind regards,

—Joachim Schaaf, JS@Coopy.Fido.De

Linux Journal Escapes Disaster

I am writing to inform you that my address has changed.

Basically, my house burnt down and I was forced to move. Fortunately all of my back issues of *LJ* were on loan to my scungy mates, who had not returned them for 3 months or so, and thus were not burnt.

Regards,

—Leon Harris, harris@possum.murdoch.edu.au

Archive Index Issue Table of Contents

Advanced search

Linus Torvalds Releases Linux 1.2.0

Phil Hughes

Issue #13, May 1995

And other fast-breaking news stories...

Tuesday, March 7, 1995, Linus Torvalds released Linux 1.2.0, also called “Linux '95”, little less than a year after his release of the ground-breaking Linux 1.0. In his announcement, Linus pointed out an important difference—the license—between the two operating systems, with an elaborate spoof on Microsoft's license explanation. Here's a short sample from Linus's spoof:

Linux '95 has several types of licenses, including, but not limited to:

- End-User License Agreement—Applications:This is an application-specific license, which is intended for a single application running on your Linux '95-authorized computer. The license agreement rules differ depending on the application. See appendix `H'.
- End-User License Agreement—Systems:This agreement is intended for single system product use, such as the Linux '95 kernel license. It's important to note that the Systems product licenses do not permit concurrent, or second copies. There is a special Multi-License upgrade program for those that want to start out with a single license but later expand their setup.
- Multi-License Pak:This agreement is intended for sites with multiple systems, which want to run multiple copies of the Linux '95 system concurrently and/or on several machines. This license is available as a 10, 50 or 100-unit license depending on the size of your installation.
- [...]
- The “I've got too much money” License: Contact us for details on this exclusive licensing deal, we'll work something out. Please contact “ivemoney@linux.Helsinki.FI” directly.

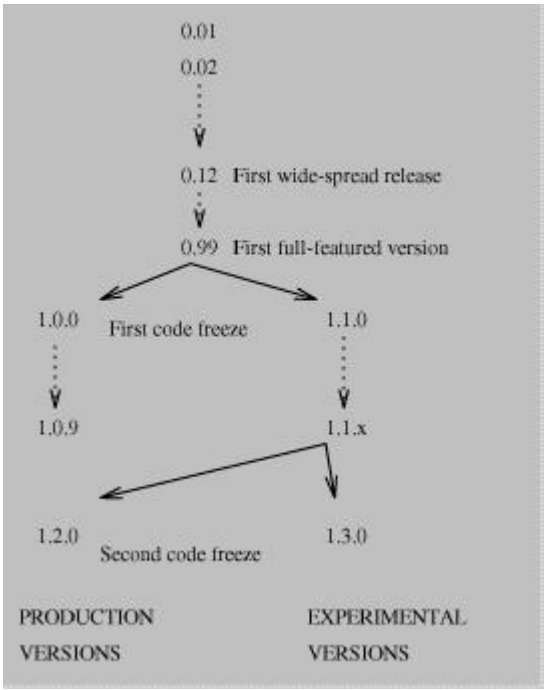
Large institutions that want to possibly combine several licenses can do so with a standard licensing fee reduction. Please contact our licensing department for further details.

[...]

We thank you for using Linux '95,

Linus

Of course, Linux 1.2 is still freely available under the GNU public license, just as Linux 1.0 was, and just as future versions will continue to be, and so all the licensing fees mentioned above are $0 per copy...except maybe the “I've got too much money” license.



Figure

Here is a list of a few of the new features that version 1.2 has, compared to version 1.0:

- Highly improved networking.
- Faster disk I/O.
- EIDE, multiple-IDE controllers, and ATA-CD-ROM support.
- Supports more kinds of floppies, including 2.88 MB.
- More CD-ROM devices supported.
- Improved SCSI support, support for new SCSI adapter drivers.
- Support for more network adapters.

- New, improved memory management.
- Multi-platform support underway.
- UMSDOS filesystem: install Linux on DOS filesystem.
- New sound driver supports Linux DOOM
- Much, much more.

Most Linux vendors are expected to release new versions of their distributions containing Linux 1.2 soon.

If you have Internet access, you can download the source code for the new kernel via ftp from ftp.funet.fi, in the directory /pub/OS/Linux/PEOPLE/Linus/v1.2/. Version 1.2.0 is in the file linux-1.2.0.tar.gz; by the time you read this, it is likely that a few bug fixes will have been released, so look for the newest version there. If you do not have internet access, a local BBS may have the source code, and any one of the vendors advertising in *Linux Journal* will be happy to sell you a CD-ROM with the source included as soon as they have them available.

Linus has indicated that further testing, and bug fixing as necessary, will be done for approximately another month, leading to an even more stable Linux 1.2.x release. He has indicated that he will probably begin work on a new development series, Linux 1.3.x, sometime in early April.

Some utilities will need to be upgraded if you upgrade from Linux 1.0 to Linux 1.2, including networking utilities like ifconfig. The update program and the module utilities will also need to be updated. An article in the next issue of *Linux Journal* will cover the process of updating from 1.0 to 1.2 in detail.

What Does 1.2.x Mean?

Since Linux version 1.0 was released, the version number has been used to distinguish between “production” or “stable” releases of Linux and “experimental” or “development” versions. Each version number has three parts: the major release number, the minor release number, and the patchlevel. Version 1.1.95, is major version 1, minor version 1, patchlevel 95. Version 1.2.0 is major version 1, minor version 2, patchlevel 0. All version numbers with *even* minor version numbers are considered production versions, and official patches to production versions only include bug fixes, so for instance, between version 1.0.0 and 1.0.9 there were only bug fixes. All version numbers with *odd* minor version numbers are development versions, which (although they usually work fine) are not expected to be stable—all sorts of changes might have happened during their development.

Linux 1.2 is the second production release of Linux, and follows the first production release, Linux 1.0, by just under a year.

What Is Left to Do?

Linus and others are working on support for multiple platforms, including DEC's Alpha, Sun's Sparc, MIPS' R3000 and R4000 series, and IBM and Motorola's PowerPC. Support for Amigas and Ataris with the Motorola 680x0 CPU has become relatively stable recently, and work is progressing on merging that work into the standard development kernel. DEC has funded a small team of programmers to work on porting Linux to the Alpha, and has loaned an Alpha machine to Linus. They are working in parallel. The DEC team has already released a 32-bit version of Linux that can run some programs, and Linus has released a 64-bit version that can run some Alpha OSF/1 binaries, and has successfully run GNU Bash.

Many Linux developers have been working behind the scenes on more performance enhancements and additional functionality that they would like to add to Linux 1.3.x. Linus has indicated that disk-space quotas, considered essential by many people who use Linux to run public-access systems, will probably be added during 1.3.x development. Further improvements to networking are being designed by Alan Cox, the main networking developer. Donald Becker, the developer of most of the Linux ethernet drivers, is working on some 100MB/s ethernet drivers that are likely to be added to 1.3.x. Work on wide SCSI support has already been started. More “intelligent” (or co-processing) serial adaptor drivers will be added. New hardware devices are being added all the time, and it is reasonable to expect that quite a few new supported devices will be added during 1.3.x development.

Not necessarily in version 1.3.x, but at some point in the not-yet-determined future, Linus has mentioned support for systems with multiple CPU's, such as the dual Pentiums being sold by many vendors now.

Of course, all the exact predictions here are the predictions of the author, and should be “taken with a grain of salt”. We can expect that the Linux developers have more nice surprises in store for us, some of which they haven't yet thought of.

In Mostly Unrelated News…

The Linux dosemu development team reported that their current development version of dosemu (runs DOS, and DOS programs, from within Linux) runs the 32-bit Borland C/C++ compilers. They also report being able to run several 32-bit DOS-only games that they had been told would be impossible to run in dosemu. The list of games posted included: DOOM, DOOM2, Mortal Kombat,

Mortal Kombat 2, OMF 2097, Raptor, Wacky Wheels, and Rise of the Triad. (Note that Id software has also made native versions of Doom available for Linux.) Of course, common DOS applications like WordPerfect still run, too.

The team intends to have a public release of dosemu ready within a month or so; hopefully dosemu 0.60 will be available for ftp from tsx-11.mit.edu in /pub/linux/ALPHA/dosemu/ by the time you read this. They are currently looking for volunteers to help them write documentation, and they will probably still be looking for volunteers to help with this large task by the time you read this; if you are interested, download dosemu and contact the team leader, James MacLean, jmaclean@fox.nstn.ns.ca.

—Phil Hughes, Publisher

Archive Index Issue Table of Contents

Advanced search

DOS Emulation with dosemu

Dean Oisboid

Issue #13, May 1995

Do you like Linux, but need to use DOS programs? Dean Oisboid recently braved the wilds of the Linux DOS emulator and has returned with the answer.

Welcome back. In the first Novice-to-Novice article, we installed Linux to an existing DOS setup using UMSDOS and without needing to repartition. This is one of the great beauties of Linux for PC users—being able to try it without losing their DOS setup.

If you read the first article, you know it took me a few tries to get a good installation, the problem primarily being not enough hard drive space despite reserving 150 megabytes for this purpose. Since then, I must confess, I reinstalled one more time, and by being very picky regarding which files to install, I managed to keep Linux down to about 80 megabytes of the 150 available.

Also I discovered that Slackware defaults to four virtual consoles set up for login, with X coming up on the fifth virtual console. While this was great for switching between screens using the Alt-F buttons, the resulting lack of RAM (I only have 8 MB) interfered with many programs, most notably **seyon** (a modem program) under X-Windows. The solution was to edit the /etc/inittab file and comment out (by putting a # character at the beginning of the lines mentioning tty3 and tty4) the third and fourth virtual consoles. Rebooting with this freed up a lot of memory, allowing X and seyon to run a bit smoother.

For this article, we'll explore a unique facet to Linux: running MS- or PC-DOS under Linux via **dosemu**. (Windows emulation using WINE is still in its infancy and was covered in the August 1994 issue of *Linux Journal*.)

Why try dosemu? Put simply, emulators allow you to use multiple operating systems at one time. The flexibility that this gives the user is something that is not available in Windows and DOS, but it is available under Linux. There are

also other emulators available for other operating systems; the brave can try an Apple II+ emulator or an “iBCS2” emulator that allows the user to run binaries intended for other Unix systems. You can see that Linux may well stand as the vanguard of a quiet yet extraordinarily powerful revolution—that of function and utility over sheer profit margin.

Enough of that, let's do it!

For the record, I tested version 0.53 patch level 28 of dosemu. This is the distribution I received on my Slackware Professional 2.1 disks. (As always with projects in development, these versions were obsolete even before the CDs were mastered.) The latest development version of dosemu is 0.53pl45 as of this writing, and the developers plan to have a stable, public release available soon after Linux 1.2.0 is released.

I copied /tsx-11/ALPHA/dosemu/Development

/pre0.53_28.tgz to /usr using **mc**, the Midnight Commander—the great Norton Commander clone—which also, via an option under F2, allowed me to unzip and untar the file once it was copied over.

If you have purchased any Linux documentation, you may have discovered that information on dosemu is skimpy. There is a HOWTO, but it doesn't cover using the emulator in conjunction with UMSDOS nor how to begin, except for a reference QuickStart to a file that comes with dosemu. Looking through this file, another thing you'll notice is that dosemu will need to be compiled. Time for a heart attack.

Two immediate caveats: One, you will need to have **bison** and **flex** (installed from Slackware's `D' set), otherwise the compilation process will crash. Second, you want to backup any important DOS files and prepare a DOS system boot disk. I insist!

If you want to compile this version of dosemu, the installation recommends having a sum total of 12-20MB of RAM and swap available. My computer has only 8MB of RAM and dosemu just wouldn't compile. Swap space was definitely needed. I had not ever tried setting up swap but decided to give it a try.

Novice Note: If you decide to create a swap space (while using UMSDOS) you **must—must, must**--repartition your hard disk. If you go through the creation process without repartitioning and referring to that partition, expect to lose something. My hard disk got slaughtered! Linux wouldn't recognize directories. A reboot wouldn't bring up DOS. Panic? Heart attack? Wet pants? Oh yes, and then some. Luckily I had a DOS boot disk, and Norton Disk Doctor was able to

put things right and restore the FAT (File Allocation Table) and system files. Sigh of relief! This is how religions get started.

What I really wanted, since it doesn't require repartitioning, was a *swapfile* for my swap. Peter Scott of the Jet Propulsion Laboratory pointed this out to me and gave me a clue as to the commands to do the job. In a normal Unix system, he also mentioned, you would want a large partitioned swap space, around 50MB or so. I shook my head; my hard drive was down to 30MB because of a sudden influx of vital games and work-related data. [Linux requires less swap space than normal Unix systems because it uses memory somewhat differently —ED]

According to the System Administration manual, you create a swap file called /swap with the following:

```
dd if=/dev/zero of=/swap bs=1024 count=8192
mkswap /swap 8192
sync
swapon /swap
```

This will create an 8MB swap file. Even if you decide to skip using dosemu, you could still benefit from using swap. For example, I've been playing with a copy of Netscape in X, but without swap, it wouldn't allow me to access the preferences menu. It would crash. But with the swap active, preferences and other menus became available. This was a real testament to the value of swap space.

In order to cause the swap file to be used each time you boot, you can add swapon /swap to your /etc/rc.d/rc.local file. However, a better way is to add the line:

```
/swap  none  swap  defaults  0  0
```

to your /etc/fstab file. This will cause the swap to be automatically added at the best possible time.

If you ever desperately need the disk space back, you can delete the swap file with:

```
swapoff /swap
rm /swap
```

If you have added a line to /etc/rc.d/rc.local or /etc/fstab, make sure that you remove that line.

With a swap file in place and active, I tackled the compilation of dosemu again. Reading through the QuickStart file knotted my stomach but I followed instructions. The command:

```
make most
```

started the dirty work of compilation. I do not have **TeX** (an advanced typesetting system) installed, so this was the correct command instead of **make doeverything**. After some time of grinding, the compilation finished. No crashing errors! The instructions then said to copy hdimage.dist to /var/lib/dosemu/hdimage and also to copy and edit /examples/config.dist to /etc/dosemu.conf.

Editing /etc/dosemu.conf was straight-forward. The only lines I really had to change were for my mouse and modem; I also had to uncomment the reference to:

```
disk { image "/var/lib/dosemu/hdimage" }
```

The sample dosemu.conf file that was provided has extensive comments explaining what to do.

With that done, I then prepared the recommended boot disk by returning to DOS and formatting a system disk. Then back to Linux.

Finally, I typed in dos, and sure enough, the boot disk booted and I was staring at the familiar A: prompt. From there if I typed C: I was brought to an image of a C: drive.

Nice, but not what I was expecting. The goal was to be able to access my current DOS system through Linux, not have a faux DOS image.

Going back into the /etc/dosemu.conf file, I switched the hard drive reference to:

```
disk { partition "/dev/hda1" 1 readonly }
```

This would allow me to access the hard drive but not change anything because of the readonly mode. Yet when I started dosemu nothing happened. What the heck, I changed the disk reference to:

```
disk { wholedisk "/dev/hda" }
```

which would allow full and potentially dangerous access. Typed dos and sure enough dosemu started to load. First thing noticeable was a message saying something about QEMM (Quarterdeck's memory manager program) not being

loaded because the processor was already in Virtual 86 mode. Without QEMM, many of the other memory residents didn't get loaded. A check using DOS mem showed that the mouse and CD drivers were missing (not surprisingly, since dosemu provides its own mouse and CD drivers). Yet in all, I had 599K free RAM. Not too bad. However, the real test would be to run a program. [dosemu provides the basic functionality of QEMM—it provides as much EMS and XMS memory as you want it to—ED]

Unfortunately, while the documentation says that wholedisk access is dangerous, it is easy to miss why it is dangerous. It turns out that a DOS file system can be destroyed by using wholedisk access to that partition while that partition is mounted by the kernel, either as a Ms-dos or UMSDOS file system.

After experiencing some file system damage, I found out that there is a way to make my DOS C: drive also be my C: drive under the DOS emulator without doing damage and without booting from a floppy. Taking a hint from a comment in dosemu.conf, and with a little help from a friend, I first edited config.sys on my boot floppy, adding the line:

```
device=a:\emufs.sys /dos/
```

which makes the /dos directory under Linux show up as C: under dosemu. I also added the line:

```
a:\lredir d: linux\fs\
```

to autoexec.bat to make the Linux root directory show up as D: under dosemu, to make it easier to access some Linux files from within dosemu.

I then enter

```
dd if=/dev/fd0 of=/var/lib/dosemu/floppy
```

with my boot floppy in drive A. Then, in the floppy configuration section of dosemu.conf, I changed

```
floppy { device /dev/fd0 threeinch }:
```

to:

```
floppy { heads 2 sectors 18 tracks 80 threeinch
file /var/lib/dosemu/floppy }
floppy { device /dev/fd0 threeinch }
```

by modifying and uncommenting a few lines. I changed the **bootC** line to **bootA** and removed the **wholedisk** line that damaged my files. Now I could boot

without a floppy in the drive, but C: was still C:. [There are other ways to do the same thing, but this works fine—ED]

Sadly, it still seemed that nothing would run. Quattro Pro 5 for DOS crashed mightily. Warlords II did the same. The crashes were bad enough to not only lock up the computer but when I did a Ctrl-Alt-Delete to reboot, the screen filled with colorful garbage. This rather pretty effect also happened if I switched virtual consoles.

Although I hated to ask for any help, I sent a message to James MacLean, who is the current maintainer of dosemu, asking if anyone else had success using dosemu under UMSDOS. Three replies came quickly back from Mr. MacLean, Marty Leisner, and Lars Marowsky-Bree. All offered help, which I politely declined because their very replies gave me the incentive I needed—the knowledge that, indeed, someone had succeeded!

Back in Linux, I glared mightily at dosemu.conf—and bless it if the answer wasn't there all the time. A section of the configuration file deals with the video setup. Originally I had chosen:

```
video { vga console }
```

but I should have chosen:

```
video { vga console graphics }
```

to enable graphics. This worked! When I ran dos and picked the bare boot setup, sure enough, I could now run programs like Warlords II or Quattro Pro 5.

But the mouse wouldn't work. AAARRGHH!! OK, **exitemu** to return to Linux, edit /etc/dosemu.conf, and change the line from:

```
serial { com 1 mouse }
```

to

```
serial {com 1 device /dev/mouse }
```

and try once again. No good. DOS just couldn't find the mouse as if...as if the mouse were already in use. GPM! (General Purpose Mouse driver) My Linux setup automatically loads gpm upon booting. Using top, I located the process number for gpm and **kill**-ed it. [**killall gpm** would also work—ED] Back to DOS, load the mouse driver. YES! DOS recognizes the mouse. YES! Warlords II runs with mouse support! YES! Quattro Pro v5 runs with mouse support! NO! FoxPro 2.6 crashes because of memory problems (I expected that but the lucky streak

was there and I had to press it). [FoxPro 2.6 can also be run with the correct configuration—ED]

Whew! Despite all my problems with compiling, swaps, and pretty color crashes, I consider dosemu an incredible achievement. That DOS can run under Linux is mind-boggling. Sure, dosemu can be rough going for us newbies but the potential reward makes up for it. There are a few things being worked on now that will make it easier. One is the FreeDOS project which is writing a free version of DOS. It will run under dosemu just like MS-DOS, except that dosemu can be distributed with it pre-installed, saving the user (that's you!) the pain of installing a legally-owned copy of MS-DOS, PC-DOS, or DRDOS to get started. Also, the public release of dosemu should be very stable and easier to distribute.

Next month, the lighter side of Linux as Novice-to-Novice descends into the depths of DOOM and tackles blasting sound using a SoundBlaster. Until then...

Dean Oisboid, owner of Garlic Software, is a database consultant, Unix beginner, and avowed Warlords II addict. He can be reached at 73717.2343@compuserve.com.

Archive Index Issue Table of Contents

Advanced search

New Products

LJ Staff

Issue #13, May 1995

Tower Releases Eiffel for Linux, Acucobol Announces Linux Port of ACUCOBOL-85 and more.

Tower Releases Eiffel for Linux

Tower Technology Corporation announces a promotional offer of its object oriented development system for Linux. TowerEiffel for Linux includes a high performance Eiffel 3 compiler, open development environment, programming tools and a base set of reusable software components. Key features include fast executable code, global system optimization, user controllable garbage collection, clear and precise error messages, exception handling, genericity, automatic system builds, automatic documentation generation and built in test support. A unique capability of TowerEiffel is Eiffel, C, and C++ interoperability.

Eiffel is a non-proprietary object-oriented programming language.

For more information, including details of the special promotional offers, visit the Tower World Wide Web Page at http://www.cm.cf.ac.uk/Tower/ Access the Tower Automated Reply System by sending an e-mail message to info@twr.com. Call (800) 285-5124 or (512) 452-9455, or fax (512) 452-1721.

Acucobol Announces Linux Port of ACUCOBOL-85.

COBOL developers can now combine the benefits of the Linux environment with the application development features of ACUCOBOL-85 and take advantage of its many features, including: programmable hot keys, advanced window capabilities, a user replaceable file system and a built-in-source code debugger. Once an application is compiled in ACUCOBOL-85, developers can use Acucobol's full range of COBOL development tools and enjoy its portability to over 600 different platforms and operating environments without recompiling.

For more information, e-mail info@acucobol.com, send mail to Acucobol, Inc., 7950 Silverton Avenue, Suite 201, San Diego, CA 92126, phone (619) 689-7220, or fax (619) 566-3071.

Scientific Instrumentation Libraries for Linux

Renegade Solutions has announced the availability of scientific instrumentation libraries for Linux. Currently offered are a basic driver for the Epix Silicon Video MUX series of video frame grabbing boards, a driver for a low budget serial port D/A and A/D device and a driver for a custom DSP board based on the TI C25 and C50 series DSPs.

The Silicon Video MUX family of framegrabbers provide up to 752x480 8 bit grayscale images making it suitable for demanding computer vision and research applications. The basic driver package includes a simple GUI program to demonstrate the boards capabilities as well as source code for the library, and is priced at $500. Motif based image editing, processing and analysis software is also available. The driver for the serial port module also includes a GUI demonstration program and sells for $200. A driver for a low cost DSP board built by FroZen Technologies is also priced at $200.

Renegade Solutions will develop custom drivers for the above hardware, as well as for other instruments. For more information, call Renegade Solutions at (208) 524-6440 or e-mail renegade@srv.net.

OPEN LOOK, XView and NeWS Archive CD-ROM

March 20, 1995—Darwin Open Systems today unveiled the OPEN LOOK, XView and NeWS Archive. This CD-ROM includes the full text of two books from the O'Reilly X series: the previously-unpublished Volume 3, User's Guide, OPEN LOOK edition, and Volume 7, XView Programming Manual. Both are on the CD in PostScript and in Adobe's Portable Document Format (PDF). There are also many other documents such as FAQs, and man pages for everything (included in troff, cat-able, PostScript and PDF, and usable from the CD-ROM by setting MANPATH). There is source code for the latest XView Toolkit, full source for many other OPEN LOOK programs and games including sample programs from several textbooks, pre-built search indexes for tools such as Glimpse, LQ-Text, viewers for PostScript and PDF, and other goodies. Also included is the full source code for two complete window systems:

- The X Window System, Version 11, Release 6, (latest patch Level 11), including XFree86 (3.1.1.), and
- The MGR Window System by Steve Uhler of Bellcore.

In addition to the textbooks, tool, and source code, there is an online “tour” of OPEN LOOK, XView, and other parts of the CD-ROM. The Tour is written in validated HTML, usable with Mosaic, Netscape, TkWWW, or your favorite Web viewer. If you are on the Internet, you can follow links to the outside. HTML hyperlinks are provided to the original FTP sources for much of the free software so you will be able to obtain updates, later versions, binaries for other platforms, etc.

Contact: Darwin Open Systems, P.O. Box 278, Palgrave, Ontario Canada L0N 1P0, by phone at +1 800-463-2108, or by e-mail to info@darwinsys.com.

New Book Release

*Running Linux*By Matt Welsh & Lar Kaufman600 pages, ISBN: 1-56592-100-3, $24.95 (US)

O'Reilly & Associates, Inc. has just released Running Linux, which covers installation and use of the Linux operating system, including:

- The background and concepts of the Linux system.
- A comprehensive installation tutorial that will lead you through the steps of configuring Linux on your machine from any distribution.
- A chapter on UNIX basics, designed especially for Linux users.
- Information on Linux system administration and maintenance, from managing user accounts, to repairing filesystems, to upgrading software, to building a new kernel.
- Full installation and configuration information for XFree86 3.1 (Version 11, release 6 of the X Window System).
- Power tools available for the system, including the X Window System, the emacs and vi editors, text formatting systems such as TeX, and tools for interfacing with MS-DOS.
- The programming languages and tools available for Linux, giving you a complete software development environment: the gcc C and C++ compiler, the gdb debugger, and other languages such as Perl and Tcl/Tk.
- Network configuration and administration under Linux, including TCP/IP, SLIP, UUCP, electronic mail, and serial telecommunications.
- Information on providing network services from your Linux machine, including configuration of a World Wide Web server.

Contact: O'Reilly & Associates, Inc., 103A Morris Street, Sebastopol, CA 95472. For inquiries: (800) 998-9938. For online orders: order@ora.com.

ParaSoft to Release Linux Insure++ for Linux

ParaSoft has announced that the Insure++ (formerly Insight++) runtime debugging system is available for Beta testing under Linux, and is expected to be publicly released soon. The product includes memory access testing including corruption and leaks, coverage analysis, data use visualization, pointer abuse, and more in C and C++ code. Programmers interested in Beta-testing Insure++ are encouraged to ask.

ParaSoft may be reached at info@parasoft.com, at URL www.parasoft.com, by phone at (818) 305-0041, by fax at (818) 305-9048, or by mail at 2031 S. Myrtle Ave., Monrovia, CA 91016-4836.

Linux International

Archive Index Issue Table of Contents

Advanced search

Anonymous FTP

Mark Komarinski

Issue #13, May 1995

This month Mark shows us the safe and painless way to set up anonymous ftp on your system.

The standard **ftp** daemon that comes with recent Slackware distributions, and which is used by many—perhaps most—anonymous ftp sites, is the wu.ftpd which comes from Washington University at Saint Louis. It offers many useful features, including:

- Extensible support for get filename.gz when only filename exists, and for get filename when only filename.gz exists (auto-gzip and auto-gunzip), and similar functionality for .Z (compress).
- Extensible support for get dir.tar (auto-tar)
- Extensive logging
- Ability to limit the number of connected users based on location
- Ability to easily change configuration

Getting and Compiling

You can get the most recent copy of wu.ftpd at sunsite.unc.edu as /pub/Linux/system/Network file-transfer/wu-ftpd-2.4.linux.tar.gz. Be sure not to get the wu-ftpd-2.4.patch.gz file that exists there—the file wu-ftpd-2.4.linux.tar.gz is already patched, and applying wu-ftpd-2.4.patch.gz removes the patches.

Once you uncompress and untar the file, you'll have a wu-ftpd-2.4.linux directory containing all the source files. The configuration for wu.ftpd is slightly different than most packages you may install for Linux. For example, you don't use make. You execute the build script with the lnx option:

```
trippy:~/wu-ftpd-2.4.linux> build lnx
```

Before you do that, you may want to change the configuration files. The config.h configuration file usually does not need to be changed. The more important configuration src/pathnames.h file tells the daemon where various files, including the bianary, are located. The INSTALL file in the source root directory explains each of the entries in the configuration files.

During the build, you may notice a number of warnings from GCC. Many of them relate to SIGBUS being redefined. The second warning will tell you where it was previously declared. This is okay. It just means GCC noticed that two header files defined the same thing, in this case SIGBUS.

After the build is complete, you may wish to back up and remove the old ftpd program. Using /sbin/pkgtool, which comes with most Slackware releases, is an easy way to do this. You will want to view tcpip. Write down the locations of files that have ftp or in.ftp in them. After this, you can back up and remove the existing files. This way, you can be sure you have a clean installation.

When you are ready to install, use **su** to assume root status, then type **build install**, which will copy the binaries to the correct locations. By default, the daemon itself goes in /usr/local/etc/, support files go in /usr/local/bin/, and man pages go in /usr/local/man/.

Last, create the ftp user, by either manually editing /etc/passwd as root (be careful!) or by using the useradd command. Optimally, ftp should have "*" as an encrypted password, be in its own group (you probably want to create an ftp group for this, if there isn't one already); and have /bin/true as a login shell. Having /bin/true as a shell means that even if someone is able to break into the account, they will be automatically logged out, as /bin/true immediately exits (you can also use /bin/false). When you're done editing the ftp user in /etc/passwd, it should look something like this:

```
ftp:*:401:50::/home/ftp:/bin/true
```

The two numbers may be different, and the /home/ftp may be different if your users have another path to access their home directories (like /user/ftp), but you get the general idea.

Configuring

Now on to the fun part—setting up the configuration files. This may appear at first to be a large task, but once a configuration is set up, it requires very little maintenance—if it is set up correctly the first time. The obvious advantage to having many configuration files is that you don't have to re-compile wu.ftpd to add or modify features.

Before we go into the configuration files, I have to explain how anonymous ftp works from an administration point of view. In order to preserve the security of legitimate users of the system, ftpd changes its root directory so that it can only see a small part of the filesystem. That is, when you type cd / during an anonymous ftp connection, you don't really go to the real root directory, the one that normal users see. A chroot(2) function call changes the root of ftpd to the home directory for user ftp in /etc/passwd. In my case, the root for an anonymous user becomes /home/ftp relative to me, a legitimate user of the system. From here on, I'll call the root for an anonymous user “ftp” root or “~ftp/” and the `/' or `root directory' will be the root directory you know and love.

First, we have to set up files in our root directory. The ~/wu-ftpd-2.4.linux/doc/examples directory contains a number of sample files that will work for most situations. These files include ftpusers, ftpaccess, and ftpconversions. They should be installed in the /etc directory after modifying it for your local setup.

The ftpusers file contains a list of users who are not allowed to connect via ftp. This increases the security by not allowing users such as “root” to connect and possibly overwrite a file like /etc/passwd. Users are listed one per line.

The ftpaccess file is the primary ftpd configuration file. It sets up certain definitions of users, configures the major sections of security, and a previous number of other functions. The ftpaccess file in the examples directory should be fine for most people, but if you expect large numbers of people connecting for anonymous ftp use, you can use the ftpaccess.heavy file. Here are a few things you will want to look at in whichever ftpaccess files you use:

- **loginfails int**—This sets the number of unsuccessful log in attempts allowed before ftpd disconnects.

  ```
  class class typelist addrglob
  ```

  [addrglob...]—This sets up a *class* of users. *typelist* can be **real** for real users of the system, **anonymous** for anonymous users of the system, and **guest** for guest accounts (which aren't quite like anonymous). *addrglob* lists where someone is coming from, either in IP (or dot—198.30.149.*) notation, or name notation (*.satelnet.org). This allows you to set up a group as **local** users (anonymous users who are on your same subnet for example) and a group for **remote** users who might be outside the local subnet. A setup could look like this:

  ```
  class local real,guest,anonymous 198.30.149.*
  class remote real,guest,anonymous *
  ```

  limit class n time msg file—This sets up limits on class of users, limiting access to n users during time. If they can't get in, display the text in the file

~ftp/msg file. Time is defined as in UUCP's L.sys file, but for those of you who can't find that file, the times are rather easy. Sunday is represented as `Su', Monday as `Mo' and so on. These can be followed by times in 24-hour format. So, to limit the number of remote users to 5 except for weekends and at night (when the limit goes to 10):

```
limit remote 10 SaSu|Any1800-0600 /etc/msgs/msg.toomany
limit remote 5 Any /etc/msgs/msg.toomany
```

The default ftpaccess.heavy file seems to have this backwards. Also, if a limit is not valid (i.e. you put fubar as a limit) or it is -1, this is taken to be unlimited users.

- **passwd-check none|trivial|rfc822 warn|enforce**—This sets up how the ftpd should respond when given a password for an anonymous login. There are three methods that you can have for allowing passwords:
- none—We don't care what the password is (you can log in as guest)
- trivial—Has a @ in the name (you can log in as komarimf@)
- rfc822—Must be an RFC822 complaint address (full username and host)

If you have set trivial or rfc822 and the user does not comply with the requested password format, you can do two things:

- warn—tells the user that an invalid password was entered, gives a suggested password, but lets the user log in anyway
- enforce—tells the user the password is invalid, gives a suggested password, then logs the user out.

- **readme file when**—tells the user to be sure to read the ~ftp/<file> during *when*. *when* can be on login or on certain directories. For example, the entries:

```
readme README* login
readme README* cwd=*
```

will notify you of all the README files (README*) upon login, and any time you change directory. Note that you may want to change this to just README if you have a large number of README files in a directory.

- **message file when**—This is almost the same as the readme entries above, only the ~ftp/file is output to the user's screen during when.
- **shutdown file**—If file exists, ftpd will not allow any logins. This gives you a good way of shutting down ftp without having to edit /etc/inetd.conf and restarting inetd, then doing the reverse to get it all working again. In fact, all you need do is type (in this case):

```
touch /etc/shutdownftp
```

and no more logins will be allowed. Notice that connected users are not disconnected automatically via this method. You bump connected users off the system, and I'll get to that in a bit.

Finally, there is the ftpconversions file. One of the features of wuftpd is that it allows you to download files in a format different from the on the remote side. That is, you can automatically compress a text file as it gets sent to you, making the transfer go quicker. Or you can get an entire directory bundled up in one .tar file, ready to put on a floppy disk. There are few modifications you should make to this, as the .tar and .gz (for gzip) already exist to allow you to tar, compress, untar, and uncompress a file on the fly. This should go in /etc/ftpconversions. Other files are placed under ~ftp/. For example, you have to install the gzip and tar binaries, plus the shared libraries necessary to run the commands. Change directory to ~ftp/bin (use mkdir to create it first, if it does not exist) and execute the following:

```
cp /bin/ls .
cp /bin/gzip .
chmod 0111 *
```

This will copy the binaries, then make them both execute only.

Next you'll need the libraries. Go into the ~ftp/lib directory (or make it and do the following:)

```
cp /lib/libc.so.4 .
```

Yes, you can make a hard link, and you can also use statically linked executables. If you feel confident in your abilities to do either, you can do it.

You could also have a copy of /etc/passwd. Without it, when regular users of the system make files available to anonymous ftp users, user and group names (only the literal UID and GID as they are stored in the filesystem) will show up. So to resolve numbers to names, you'll need copies of both /etc/passwd and /etc/group in ~ftp/etc/. However, you don't need encrypted passwords or other information in the passwd file, only the UID, GID, and username. So you can replace all the passwords in the copied passwd file (be sure you're editing the correct file!!) with stars (*), then erase all the fields after the GID, leaving you with entries like:

```
mark:*:401:100:::::
```

The downside of this (for the extremely security conscious) is that a remote user could now know login names for legitimate users. Then again, you can change the names of the valid users of the system to something else. What could be better than letting a cracker spend all day (without success and

increasing his chances of getting caught) trying to crack an account named “bob” on your machine who he saw as the owner of a file on anonymous ftp?

To be sure that the files are set up correctly, run the ~\wu-ftpd-2.4.linux/bin/ckconfig program. It will verify the locations of various files needed to run.

Security and Final Setup

Speaking of security—now is the time to make sure that files are as they should be. You don't want anyone getting free access to your system, do you? Here's how the directories in ~ftp/ should be laid out:

- Execute only (mode 0111 in chmod): ~ftp/bin and ~ftp/etc
- Read and execute (mode 0555 in chmod): ~ftp/pub, ~ftp/usr, and ~ftp/var (if you want it—or it can be a link to ~ftp/usr)
- Write and execute (optionally read too—0333 or 0777 in chmod): /incoming

Incoming is a special directory where users are allowed to place incoming files. From here, the files are then moved by the ftp administrator to another location. Many sites have their incoming directories set up so that any file that is uploaded can immediately be downloaded by another anonymous user. But, an unscrupulous user can upload an illegal program for immediate download by anyone else. In the time it takes the ftp administrator to discover the file and erase it, hundreds of people could have downloaded it. By denying read access to the directory, users can put files there, but not see them. Instead you get a message about “permission denied”.

The execute-only directory is also special in that while anonymous users can run the programs in that directory (tar and ls usually), they can't get a directory of what is there, nor can they put anything in the directory. This is good because it prevents an anonymous user from finding out what kinds of programs are available, and also allows extra security on the ~ftp/etc/passwd and ~ftp/etc/group files.

Now your ftp site should be ready for allowing incoming users. If you had to install wu.ftpd, make your changes to /etc/inetd.conf by adding the following line:

```
ftp stream tcp nowait root /usr/sbin/tcpd /usr/local/bin/ftpd
```

This assumes that you want the tcpd(8) wrapper and that the ftpd binary is located in /usr/local/bin. An ftp line may already be in the file, so you may have to only edit it. Restart inetd with a **killall -HUP inetd**, and sit back and relax.

If you are interested in watching how popular your ftp site is, a few programs can help. The ftpcount tells you how many users of each class are connected to your machine, and what the maximum is. ftpwho gives you slightly better information about who is logged in, and ftpshut is used to bump connected users off at a specific time. The xferstats program in the util directory can give you detailed reports of when most people connect, what gets downloaded, and other information that may help you fine-tune your configuration.

wu.ftpd has many more abilities than I have described here, such as the ability to create “private” directories and groups to allow only certain anonymous ftp users access to directories. There are also provisions for logging access information via syslogd(8) and file transfer information to /var/adm/ftpd/xferlog. There's a man page for xferlog(5) too. Both of these methods of logging are set up by default.

If you have any questions about setting up ftp on your linux machine, or any questions, comments, or even complaints about this article, please e-mail me at komarimf@craft.camp.clarkson.edu.

Mark Komarinski graduated from Clarkson University (in very cold Potsdam, NY) with a degree in Computer Science and Technical Communication. He now lives in Troy, NY, spending much of his free time working for the Department of Veterans Affairs where he is a programmer.

Archive Index Issue Table of Contents

Advanced search

The ELF Object File Format by Dissection

Eric Youngdale

Issue #13, May 1995

The Executable and Linking Format has been a popular topic lately, as people ask why the kernel configuration script asks whether or not to configure loading ELF executables. Since ELF will eventually be the common object file format for Linux binaries, it is appropriate to document it a bit. Last month, Eric introduced us to ELF, and this month, he gives us a guided tour of real ELF files.

Last month, we reached a point where were beginning to dissect some real ELF files. For this, I will use the **readelf** utility which I wrote when I was first trying to understand the ELF format itself. Later on, it became a valuable tool for debugging the linker as I added support for ELF. The sources to readelf should be on tsx-11.mit.edu in pub/linux/packages/GCC/src or in pub/linux/BETA/ibcs2.

Let us start with a very simple program—the hello world program we used last month.

```
largo% cat hello.c
main()
{
        printf("Hello World\n");
}
largo% gcc-elf -c hello.c
```

On my laptop, the gcc-elf command invokes the ELF version of gcc—once ELF becomes the default format, you will be able to use the regular gcc command which produces the ELF file hello.o. Each ELF file starts with a header (**struct elfhdr** in /usr/include/linux/elf.h), and the readelf utility can display the contents of all of the fields:

```
largo% readelf -h hello.o
ELF magic:7f 45 4c 46 01 01 01 00 00 00 00 00
        00 00 00 00
Type, machine, version = 1 3 1
Entry, phoff, shoff, flags = 0 0 440 0
ehsize, phentsize, phnum = 52 0 0
shentsize, shnum, shstrndx = 40 11 8
```

The ELF **magic** field is just a way of unambiguously identifying this as an ELF file. If a file does not contain those 16 bytes in the **magic** field, it is not an ELF file. The type, machine, and version fields identify this as an ET_REL file (i.e., an object file) for the i386. The **ehsize** field is just the **sizeof(struct elfhdr)**.

Each ELF file contains a table that describes the sections within the file. The **shnum** field indicates that there are 11 sections; the **shoff** field indicates that the section header table starts at byte offset 440 within the file. The **shentsize** field indicates that the entry for each section is 40 bytes long. All throughout ELF, the sizes of various structures are always explicitly stated. This allows for flexibility; the structures can be expanded as required for some hardware platforms and the standard ELF tools do not have to know about this to be able to make sense of the binary. Also, it allows room for future expansion of the structures by newer versions of the standard.

```
largo% readelf -S hello.o
There are 11 section headers, starting at offset 1b8:
[0]              NULL            00000000 00000 00000 00 / 0 0 0 0
[1] .text        PROGBITS        00000000 00040 00014 00 / 6 0 0 10
[2] .rel.text    REL             00000000 00370 00010 08 / 0 9 1 4
[3] .data        PROGBITS        00000000 00054 00000 00 / 3 0 0 4
[4] .bss         NOBITS          00000000 00054 00000 00 / 3 0 0 4
[5] .note        NOTE            00000000 00054 00014 00 / 0 0 0 1
[6] .rodata      PROGBITS        00000000 00068 0000d 00 / 2 0 0 1
[7] .comment     PROGBITS        00000000 00075 00012 00 / 0 0 0 1
[8] .shstrtab    STRTAB          00000000 00087 0004d 00 / 0 0 0 1
[9] .symtab      SYMTAB          00000000 000d4 000c0 10 / 0 a a 4
[a] .strtab      STRTAB          00000000 00194 00024 00 / 0 0 0 1
```

Listing 1. Section Table for hello.o

Each section header is just a **struct ELF32_Shdr**. You may notice that the name field is just a number—this is not a pointer, but an offset into the .shstrtab section (we can find the index of the .shstrtab section from the file header in the **shstrndx** field). Thus we find the name of each section at the specified offset within the .shstrtab section. Let us dump the section table for this file; see figure 1. You will notice sections for nearly everything which we have already discussed. Each section has an identifier which specifies what the section contains (in general, you should never have to actually know the name of a section or compare it to anything).

After the type, there is a series of numbers. The first of these is the address in virtual memory where this section should be loaded. Since this is a .o file, it is not intended to be loaded into virtual memory, and this field is not filled in. Next is the offset within the file of the section, and then is the size of the section. After this come a series of numbers—I won't parse these in detail for you, but they contain things like the required alignment of the section, a set of flags which indicate whether the section is read-only, writable, and/or executable.

The readelf program is capable of performing disassembly:

```
largo% readelf -i 1 hello.o
0x00000000  pushl       %ebp
0x00000001  movl        %esp,%ebp
0x00000003  pushl       $0x0
0x00000008  call        0x08007559
0x0000000d  addl        $4,%esp
0x00000010  movl        %ebp,%esp
0x00000012  popl        %ebp
0x00000013  ret
```

The .rel.text section contains the relocations for the .text section of the file, and we can display them as follows:

```
largo% readelf -r hello.o
Relocation section data:.rel.text (0x2 entries)
Tag: 00004 Value 00301 R_386_32    (0 )
Tag: 00009 Value 00b02 R_386_PC32  (0 printf)
```

This indicates that the .text section has two relocations. As expected, there is a relocation for printf indicating that we must patch the address of printf into offset 9 from the beginning of the .text section, which happens to be the operand of the call instruction. There is also a relocation so that we pass the correct address to printf.

Now let us see what happens when this file is linked into an executable. The section table now looks something like Listing 2.

The first thing you will notice is a lot more sections than were in the simple .o file. Much of this because this file requires the ELF shared library libc.so.1.

At this point I should mention the mechanics of what happens when you run an ELF program. The kernel looks through the binary and loads it into the user's virtual memory. If the application is linked to a shared library, the application will also contain the name of the dynamic linker that should be used. The kernel then transfers control to the dynamic linker, not to the application. The dynamic loader is responsibile for first initializing itself, loading the shared libraries into memory, resolving all remaining relocations, and then transferring control to the application.

Going back to our executable, the .interp section simply contains an ASCII string that is the name of the dynamic loader. Currently this will always be /lib/elf/ld-linux.so.1 (the dynamic loader itself is also an ELF shared library).

Next you will notice 3 sections, called .hash, .dynsym, and .dynstr. This is a minimal symbol table used by the dynamic linker when performing relocations. You will notice that these sections are mapped into virtual memory (the virtual address field is non-zero). At the very end of the image are the regular symbol

and string tables, and these are not mapped into virtual memory by the loader. The .hash section is just a hash table that is used so that we can quickly locate a given symbol in the .dynsym section, thereby avoiding a linear search of the symbol table. A given symbol can typically be located in one or two tries through the use of the hash table.

The next section I want to mention is the .plt section. This contains the jump table that is used when we call functions in the shared library. By default the .plt entries are all initialized by the linker not to point to the correct target functions, but instead to point to the dynamic loader itself. Thus, the first time you call any given function, the dynamic loader looks up the function and fixes the target of the .plt so that the next time this .plt slot is used we call the correct function. After making this change, the dynamic loader calls the function itself.

This feature is known as lazy symbol binding. The idea is that if you have lots of shared libraries, it could take the dynamic loader lots of time to look up all of the functions to initialize all of the .plt slots, so it would be preferable to defer binding addresses to the functions until we actually need them. This turns out to be a big win if you only end up using a small fraction of the functions in a shared library. It is possible to instruct the dynamic loader to bind addresses to all of the .plt slots before transferring control to the application—this is done by setting the environment variable **LD_BIND_NOW=1** before running the program. This turns out to be useful in some cases when you are debugging a program, for example. Also, I should point out that the .plt is in read-only memory. Thus the addresses used for the target of the jump are actually stored in the .got section. The .got also contains a set of pointers for all of the global variables that are used within a program that come from a shared library.

The .dynamic section contains some shorthand notes used by the dynamic loader. You will notice that the section table is not itself loaded into virtual memory, and in fact it would not be good for performance for the dynamic loader to have to try to parse it to figure out what needs to be done. The .dynamic section is essentially just a distilled version of the section header table that contains just what is needed for the dynamic loader to do its job.

You will notice that since the section header table is not loaded into memory, neither the kernel nor the dynamic loader will be able to use that table when loading files into memory. A shorthand table of program headers is added to provide a distilled version of the section table containing just the information required to load a file into memory. For the above file it looks something like:

```
largo% readelf -l hello
Elf file is Executable
Entry point 0x8000400
```

```
There are 5 program headers, starting at offset 34:
PHDR       0x00034 0x08000034 0x000a0 0x000a0 R E
Interp     0x000d4 0x080000d4 0x00017 0x00017 R
Requesting program interpreter \
[/lib/elf/ld-linux.so.1]
Load       0x00000 0x08000000 0x00515 0x00515 R E
Load       0x00518 0x08001518 0x000cc 0x000d4 RW
Dynamic    0x0054c 0x0800154c 0x00098 0x00098 RW
Shared library: [libc.so.4] 1
```

As you can see, the program header contains a pointer to the name of the dynamic loader, instructions on what portions of the file are to be loaded into virtual memory (and the virtual addresses they should be loaded to), the permissions of the segments of memory, and finally a pointer to the .dynamic section that the dynamic loader will need. Note that the list of required shared libraries is stored in the .dynamic section.

I will not pick apart an ELF shared library for you here—libraries look quite similar to ELF executables. If you are interested, you can get the readelf utility and pick apart your own libraries.

At the start of this article, I said one reason we were switching to ELF was that it was easier to build shared libraries with ELF. I will now demonstrate how. Consider two files:

```
largo% cat hello1.c
main()
{
        greet();
}
largo% cat english.c
greet()
{
        printf("Hello World\n");
}
```

The idea is that we want to build a shared library from english.c, and link hello1 against it. The commands to generate the shared library are:

```
largo% gcc-elf -fPIC -c english.c
largo% gcc-elf -shared -o libenglish.so english.o
```

That's all there is to it. Now we compile and link the hello1 program:

```
largo% gcc-elf -c hello1.c
largo% gcc-elf -o hello1 hello1.o -L. -lenglish
```

And finally we can run the program. Normally the dynamic loader only looks in certain locations for shared libraries, and the current directory is not one of the places it normally looks. Thus to run the program, you can use a command like:

```
largo% LD_LIBRARY_PATH=. ./hello1
Hello World
```

The environment variable **LD_LIBRARY_PATH** tells the dynamic loader to look in additional places for the shared libraries (this feature is disabled for setuid programs for security reasons).

To avoid having to specify **LD_LIBRARY_PATH**, you have several options. You could copy your shared library to /lib/elf, but you can also link your program in the following way:

```
largo% gcc-elf -o hello1 hello1.o\ /home/joe/libenglish.so
largo% ./hello1
Hello World
```

To build more complicated shared libraries, the procedure is not really that much different. Everything that you want to put into the shared library should be compiled with **-fPIC**; when you have compiled everything, you just link it all together with the **gcc -shared** command.

The procedure is so much simpler mainly because we bind addresses to functions at runtime. With a.out libraries, the addresses are bound at link time. This means that lots of special care must be taken to ensure that the .plt and .got have sufficient room for future expansion and that we keep the variables at the same addresses from one version of the library to the next. The tools for building a.out libraries help ensure all of this, but it makes the build procedure much more complicated.

ELF offers one further feature that is not easily available with a.out. The **dlopen()** function can be used to dynamically load a shared library into the user's memory, and you are then able to call the dynamic loader to find symbols within this shared library—in other words, you can call functions that are defined in these modules. In addition, the dynamic loader is used to resolve any undefined symbols within the module itself.

This may be easiest to explain with an example. Given the following source file:

```
#include <dlfcn.h>
main(int argc, char * argv[])
{
   void (*greeting)();
   void * module;
   if( argc > 2 ) exit(0);
   module = dlopen(argv[1], RTLD_LAZY);
  if(!module) exit(0);
   greeting = dlsym(module, "greet");
   if(greeting) {
     (*greeting)();
   }
  dlclose(module);
 }
```

you can compile, link, and run it (using the shared library english.so which was built earlier):

```
largo% gcc-elf -c hello2.c
largo% gcc-elf -o hello2 hello2.o -ldl
largo% ./hello2 ./libenglish.so
Hello World
```

To expand this example a little bit, you could generate other modules with greetings in other languages. Thus in theory, one could add multi-lingual support for some application merely by supplying a set of shared libraries that contain the language-specific portions of the application. In the above example, I showed how you can locate the address of a function within a shared library. But the **dlsym()** function will also return the address of data variables, so you could just as easily retrieve the address of a text string from the shared library.

As I prepare to close, I should mention some options to readelf which I have not demonstrated. **readelf -s** dumps the symbol tables and **readelf -f** dumps the .dynamic section.

Finally, I should mention something about the timetable. When we first got ELF to a point where it was usable (last September), we decided to spend a relatively long period of time testing it and shaking out all of the problems. Back then I felt that roughly 4-to-6 months would allow people to test it thoroughly, plus we wanted to give an opportunity for certain applications to be adapted for ELF (the most recent versions of insmod and Wine now support ELF, for example). As I write this, no firm date has been set for a public release, but it is possible that ELF will be public by the time you read this.

In these articles I have attempted to give you a guided introduction to the ELF file format. A lot of the material I have covered is not of much practical value to most users (unless you want to hack the linker), but my experience is that there are a lot of people who are curious about how it all works, and I hope that I have provided enough information to satisfy most people.

For more information about the ELF file format, you can obtain the ELF specifications from a number of sources—you can try ftp.intel.com in pub/tis/elf11g.zip. The specifications are also available in a printed format. See *SYSTEM V Application Binary Interface* (ISBN 0-13-100439-5) and *SYSTEM V Application Binary Interface*, Intel386 Architecture Processor Supplement (ISBN 0-13-104670-5).

Eric Youngdale has worked with Linux for over three years, and has been active in kernel development. He developed the current a.out Linux shared libraries before developing much of the new ELF support. He can be reached as eric@aib.com.

Archive Index Issue Table of Contents

Advanced search