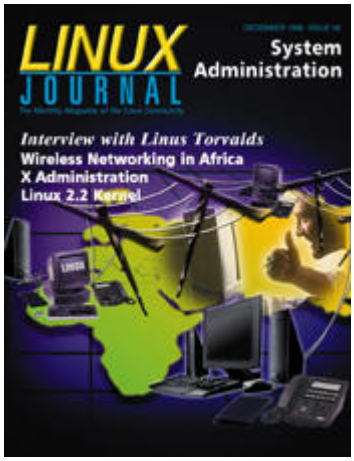
Linux Journal Issue #56/December 1998



Features

News & Articles

Reviews

Columns

Strictly On-line

Departments

New Products

Archive Index

Advanced search

Performance Monitoring Tools for Linux

David Gavin

Issue #56, December 1998

Mr. Gavin provides tools for systems data collection and display and discusses what information is needed and why.

For the last few years, I have been supporting users on various flavors of UNIX systems and have found the System Accounting Reports data invaluable for performance analysis. When I began using Linux for my personal workstation, the lack of a similar performance data collection and reporting tool set was a real problem. It's hard to get management to upgrade your system when you have no data to back up your claims of “I need more POWER!”. Thus, I started looking for a package to get the information I needed, and found out there wasn't any. I fell back on the last resort—I wrote my own, using as many existing tools as possible. I came up with scripts that collect data and display it graphically in an X11 window or hard copy.

What Do We Want to Know?

To get a good idea of how a system is performing, watch key system resources over a period of time to see how their usage and availability changes depending upon what's running on the system. The following categories of system resources are ones I wished to track.

CPU Utilization: The central processing unit, as viewed from Linux, is always in one of the following states:

- *idle*: available for work, waiting
- *user*: high-level functions, data movement, math, etc.
- *system*: performing kernel functions, I/O and other hardware interaction
- *nice*: like user, a job with low priority will yield the CPU to another task with a higher priority

By noting the percentage of time spent in each state, we can discover overloading of one state or another. Too much idle means nothing is being done; too much system time indicates a need for faster I/O or additional devices to spread the load. Each system will have its own profile when running its workload, and by watching these numbers over time, we can determine what's normal for that system. Once a baseline is established, we can easily detect changes in the profile.

Interrupts: Most I/O devices use interrupts to signal the CPU when there is work for it to do. For example, SCSI controllers will raise an interrupt to signal that a requested disk block has been read and is available in memory. A serial port with a mouse on it will generate an interrupt each time a button is pressed/ released or when the mouse is moved. Watching the count of each interrupt can give you a rough idea of how much load the associated device is handling.

Context Switching: Time slicing is the term often used to describe how computers can appear to be doing multiple jobs at once. Each task is given control of the system for a certain “slice” of time, and when that time is up, the system saves the state of the running process and gives control of the system to another process, making sure that the necessary resources are available. This administrative process is called context switching. In some operating systems, the cost of this switching can be fairly expensive, sometimes using more resources than the processes it is switching. Linux is very good in this respect, but by watching the amount of this activity, you will learn to recognize when a system has a lot of tasks actively consuming resources.

Memory: When many processes are running and using up available memory, the system will slow down as processes get paged or swapped out to make room for other processes to run. When the time slice is exhausted, that task may have to be written out to the paging device to make way for the next process. Memory-utilization graphs help point out memory problems.

Paging: As mentioned above, when available memory begins to get scarce, the virtual memory system will start writing pages of real memory out to the swap device, freeing up space for active processes. Disk drives are fast, but when paging gets beyond a certain point, the system can spend all of its time shuttling pages in and out. Paging on a Linux system can also be increased by the loading of programs, as Linux “demand pages” each portion of an executable as needed.

Swapping: Swapping is much like paging. However, it migrates entire process images, consisting of many pages of memory, from real memory to the swap devices rather than the usual page-by-page mechanism normally used for paging.

Disk I/O: Linux keeps statistics on the first four disks; total I/O, reads, writes, block reads and block writes. These numbers can show uneven loading of multiple disks and show the balance of reads versus writes.

Network I/O: Network I/O can be used to diagnose problems and examine loading of the network interface(s). The statistics show traffic in and out, collisions, and errors encountered in both directions.

These charts can also help in the following instances:

- The system is running jobs you aren't aware of during hours when you are not present.
- Someone is logging on or remotely running commands on the system without your knowledge.

This sort of information will often show up as a spike in the charts at times when the system should have been idle. Sudden increases in activity can also be due to jobs run by **crontab**.

Collecting the Data

The file /proc/stat contains current counters for most of the data I wanted, and it is in a readable format. In order to keep the collector script as quick and simple as possible, I saved the data in a readable format rather than as binary data.

Breaking down and reorganizing the data for storage was a good job for **awk**, writing the data out to different files depending on the type of data. The /proc files are formatted nicely for this; each record has an identifying name in the first field. Here's a sample of /proc/stat from my 486 system:

```
cpu 1228835 394 629667 23922418
disk 43056 111530 0 0
disk_rio 18701 20505 0 0
disk_wio 24355 91025 0 0
disk_rblk 37408 40690 0 0
disk_wblk 48710 182050 0 0
page 94533 204827
swap 1 0
intr 27433973 25781314 58961 0 1059544 368102 1 2\
0 0 0 11133 154916 0 0 0 0
ctxt 18176677
btime 863065361
processes 18180
```

I dug into the kernel source for the /proc file system to figure out what the various fields were, as the man pages seem to date back to 1.x.

- **cpu**: contains the following information: jiffies (1/100 of a second) spent in user/nice/system/idle states. I wasn't too concerned about the actual

measurement, as I was just planning on looking at each state as a percentage of the total.

- **disk**: summarizes all I/O to each of the four disks, while **disk_rio**, **disk_wio**, **disk_rblk** and **disk_wblk** break down the total into read, write, blocks read and blocks written.
- **page**: page in and out counters
- **swap**: counts of pages swapped in and out. The swap data in /proc/meminfo is expressed as total pages, used and free. Combine both sets of data to get a clear picture of swap activity.
- **intr**: total interrupts since boot time, followed by counts for each interrupt.
- **ctxt**: the number of context switches since boot time. This counts the number of times one process was “put to sleep” and another was “awakened”.
- **btime**: I haven't found much use for this—it is the number of seconds after January 1, 1970 that the system was booted.
- **processes**: the most recent process identification number. This is a good way to see how many processes have been spawned since the last check, so by subtracting the old value from the current one and dividing by the time difference (in seconds) between the two observations, the number of new processes per second is known and can be used to measure how busy the system is.

Network activity counters are found in the /proc/net/dev file; an example of this file is shown in Table 1.

The lines we want here are the **eth*x*** and **ppp*x*** records. In the collector script, the data is written out to a file using the full interface name. This way, the script is generalized for most any configuration.

Memory utilization can be tracked in the /proc/meminfo file as shown in Table 2.

The memory counters are expressed twice in this file, so we need to save only the **Mem:** and **Swap:** records to get the whole picture. The script matches the keywords at the start of the line and writes the data out to individual files rather than to one large database to allow more flexibility as new fields or data types are added. This makes for a cluttered directory but simpler script writing.

The script that collects the data is shown in Listing 1. Here are some things that are going on in a few key parts, plus comments:

- Line 13: move to the directory where the data is to be stored using **cd**.

- Line 14: get the timestamp for the data records in format HHMM.
- Line 15: get the date for the output data file names in format MonDD.YY
- Lines 19 - 25: select the memory and swap counter lines from /proc/meminfo and write the timestamp and data portion of the record to Mem.MonDD.YY and Swap.MonDD.YY.
- Lines 29 - 36: extract the counters for any network interfaces from /proc/net/dev and write them out to files including the interface numbers, i.e., eth0 data is written out to eth0.MonDD.YY.
- Lines 39 - 79: clip counters for cpu, disk, paging, swap page usage, interrupts, context switching and process numbers from /proc/stat and write them out to appropriate files.

The following line in my crontab file runs the collection script every five minutes every hour of every day:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *\
/var/log/sar/sa 0 0 * * * exec /usr/bin/find\
/var/log/sar/data/ -mtime +14
-exec /bin/rm -f {} \;
```

The data accumulates over the course of the day to provide the data points for analysis. A cleanup script invoked by the second line removes each file after two weeks to keep the disk space requirements down. A possible enhancement might be to compress each file after it is complete, but space hasn't been much of an issue yet.

What Do We Do with the Data?

I now had the data, but since columns of figures are boring, I needed a way to look at the data and make sense of it. I had used **gnuplot** for similar tools on other systems, so it seemed to be a good choice. I started with a script to display CPU utilization, charting the percentages of time spent in idle, user, system and nice states.

The cpu data file has five columns that look like this:

```
0000 4690259 69915 661038 7937582
0005 4690408 69964 661286 7966975
```

Column 1: seconds in idle state since last bootedColumn 2: seconds in system state since last bootedColumn 3: seconds in nice state since last bootedColumn 4: seconds in user state since last bootedColumn 5: time-stamp of observation (HHMM)

My reporting scheme was to get the amount of seconds spent in each state since the last observation, add up the different states and express each one as a percentage of the total. I ran into an interesting issue right away—what about

a reboot? Booting the system zeroes out the counters and subtracting the old from the new generates negative values, so I had to handle it properly to provide useful information. I decided to watch for a counter value that was lower than the last observation's value and, if found, reset the prior values to zero. To make the chart more informative, a data point was set to 100 for a reboot and -1 for a normal record. The -1 value causes the data point to be outside the chart and thus not displayed.

Sometimes a hard copy is preferred when presentations or reports are needed. The gnuplot authors provide for a variety of output formats, and the script will switch between X11 display and PostScript output depending upon which option switches are set.

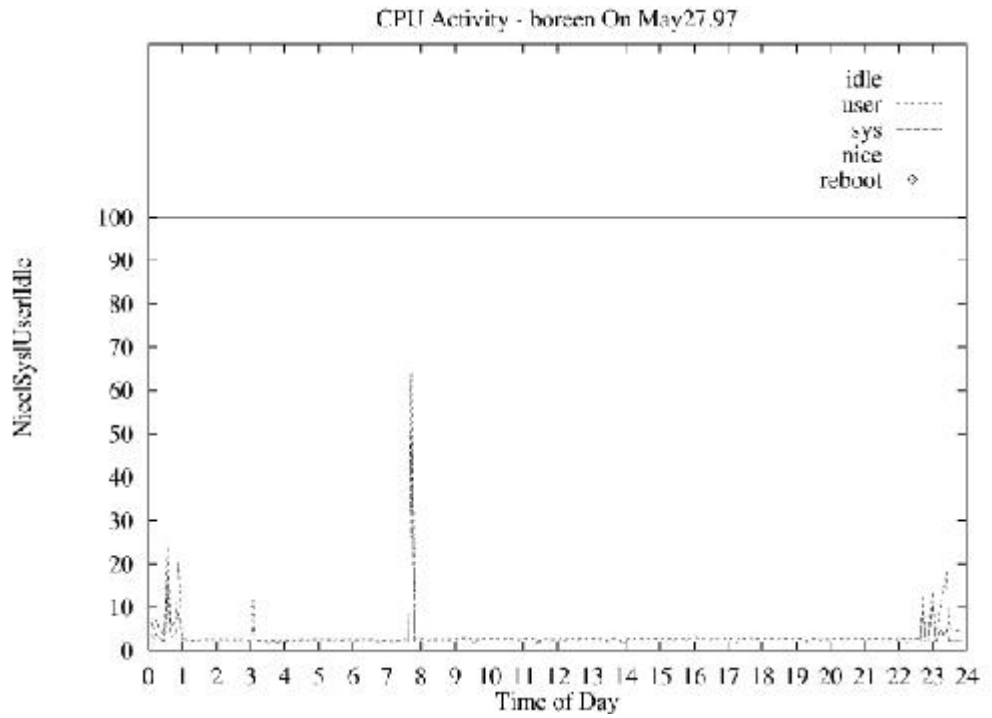


Figure 1. Sample Chart

Figure 1 is a sample chart produced by the graphing script shown in Listing 2. A breakdown of the major parts of this script is included in the archive file on SSC's FTP site, ftp.linuxjournal.com/pub/lj/listings/issue56/2396.tgz. Also included are the collection script, graphing scripts, a sample crontab entry for running the collector script and the following charting scripts:

- **cpu**: charting cpu information as described above
- **ctxt**: charting context switching per second
- **disk**: disk utilization: total I/O, read/writes and block read/writes per second
- **eth**: Ethernet packets sent and received per second and both incoming and outgoing errors

- **intr**: interrupts by interrupt number and charted per second
- **mem**: memory utilization and buffer/cache/shared memory allocations
- **page**: page in and out activity
- **ppp**: Point-to-Point Protocol packets sent/received per second and errors
- **proc**: new process creation per second
- **swap**: swap activity and swap space availability

I'm currently converting this toolkit to Perl and building a web interface to allow these charts to be viewed as HTML pages with the charts as GIF files.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue56/2396.tgz.

David Gavin (dgavin@unifi.com) has worked in various support environments since 1977, when after COBOL training, he had the good fortune to be assigned to the TSO (Time Sharing Option) support group. From there he moved to MVS technical support, to VM and to UNIX. He has worked with UNIX from mainframes to desktops, baby-sitting Microsoft systems only when he couldn't avoid it. He started using Linux back when it meant downloading twenty-five disks over a 2400 BAUD dial-up line.

Archive Index Issue Table of Contents

Advanced search

CIDR: A Prescription for Shortness of Address Space

David A. Bandel

Issue #56, December 1998

This article explains the concept of CIDR and shows you how you can implement it on your network.

CIDR, Classless Inter-Domain Routing, allows you to maximize use of the limited address space under the current implementation of the Internet Protocol version 4 (IPv4). After reading this article, even if you have never configured a computer for network communications before, you should have a good understanding of these references to networking.

Background

CIDR is the current trend in routing and has been for over three years. This concept was introduced in 1993 to alleviate the shortage of Internet Protocol (IP) addresses until the next generation (IP version 6—IPv6, aka IPng for IP next generation) arrives.

Currently in testing, IPng will significantly expand the IP address space by several orders of magnitude. IPng will also come with its own security enhancements. Those desiring to participate in the future today may have the opportunity to do so, since Linux has kernel-level support for IPng. Until IPng is deployed on a wide scale, making the best use of what we have is what CIDR is all about.

To help you understand why we need CIDR at all, let's journey back in time to the beginning of this decade. IPv4, the protocol used by computers to find each other on a network, was in use then, but there really weren't many connections to the Internet or machines needing Internet connections. In fact, a good number of systems still relied on uucp, the UNIX to UNIX copy protocol, where machines “called” each other at predetermined times and exchanged e-mail traffic. At that time, the IP-address pool seemed unlimited. That was also about the time Mosaic, the first web browser, appeared.

IP Basics

Those who consider themselves well-versed in “classful” routing may wish to skip ahead to the next section. Computers understand base 2 numbers (ones and zeroes), and humans understand base 10 (0-9), so engineers worked out a compromise to give computers numbers while keeping it simple for use by humans. All computers on the Internet have a unique IP address which can be represented by a string of ones and zeroes. If that string is divided up into four sets of eight (octets), you get four numbers with a range from 0 (eight zeroes) to 255 (eight ones), which are arranged in the form XXX.XXX.XXX.XXX. This arrangement is called “dotted decimal notation” and makes understanding the significance of each unique IP address a little easier for us humans. These addresses were then further broken down into arbitrary “classes” A-D. Looking at the first half of the first octet:

```
Class A = 0-127 (0000)
Class B = 128-191 (1000)
Class C = 192-223 (1100)
Class D = the rest (1110)
```

The positions beginning from the left represent 128, 64, 32 and 16—see Table 1. Furthermore, Class A uses only the first number as the network number, e.g., 10.XXX.XXX.XXX; Class B uses the first two numbers as the network number, e.g., 172.32.XXX.XXX; Class C uses three numbers as the network number, e.g., 192.168.1.XXX; Class D is reserved for testing purposes. A network address can be thought of as having a network and host portions represented by numbers and XXXs respectively. For a Class C address, the network portion consists of the first three octets with the host portion as the final octet.

The following concepts with respect to networking computers must be understood. Note that the “definitions” I provide here are given to aid in understanding basic concepts for use in this article, and are not the actual definitions of the terms.

- **host address**: A unique address assigned to a communications device in a computer. If a computer has multiple communications devices (e.g., Ethernet cards or modems), each of these devices will have its own unique address. This means that a host (computer or router) can be multi-homed, i.e., have multiple IP addresses. This can also be artificially created by assigning different IP addresses to the same device (called IP aliasing).
- **network address**: The base (lower) address assigned to a network segment, depending on its netmask. This is the first host IP number on a subnet. For example, on the Class C network that extends from 192.168.1.0 to 192.168.1.255, the network address would be 192.168.1.0.
- **broadcast address**: The upper address assigned to a network segment. In the example above, this address would be 192.168.1.255.

- **netmask**: A mask consisting of that portion of the IP address where all greater bits consist of ones (in base 2) and all lower bits consist of zeroes —in other words, ones represent the network portion of the address, and zeroes represent the host portion. For the example above, this mask would be 255.255.255.0.

With this introduction to IP addressing, and remembering that a decade ago almost no PCs participated in networking, it is easy to see why during the 1980s IPv4 seemed to have an endless supply of addresses, even though not all addresses could be assigned. Theoretically, if you could make use of all the usable IP addresses available, you'd have a maximum of approximately 500 million addresses, but even 100 million is extremely optimistic and insufficient for today.

Before leaving this section, I'd like to describe an experiment. This experiment will not work properly if performed in an environment with machines using only the Microsoft Windows IP stack, since its implementation is broken, or at least doesn't follow the rules everyone else plays by. Therefore, you will need to be on a UNIX or Linux machine with other UNIX or Linux boxes on your network. Type the following command:

```
ping -c 1
```

What you will see in response is every UNIX box answering back with its IP address, and each reply following the first one will have (DUP!) next to it, indicating it is a duplicate reply. The **-c 1** argument tells **ping** to send only one ping packet. The number of replies received will depend on how many (non-MS) machines you have on the network. If this is performed from an MS Windows machine (95 or NT), you will receive a reply from the local machine only.

What is the point of this little demonstration? If you change the netmask on a machine, say from 255.255.255.0 to 255.255.0.0 thereby changing its network and broadcast addresses, even though nothing else changed (i.e., it still has the same IP address and is still connected to the network the same way) it will cease talking to its neighbors. In other words, this machine is now on another network and will require a gateway to talk to the other machines on the local net (all bets are off for the Microsoft machines).

CIDR

While IP classifications A-D are still in use in the networking world, those terms are obsolete. For the sake of clarity, I will continue to use them to explain how CIDR works and how you can implement it. Along with CIDR comes the concept of variable length subnet masking (VLSM).

Basically, with a “Class” address, you have a default subnet mask. For a Class C address, this default subnet is 24 bytes long, so putting all ones in the first 24 bytes and zeroes in the rest, we have 255.255.255.0. For class A and B, this would be 255.0.0.0 and 255.255.0.0, respectively. This basically gives anyone assigned a full Class C address 256 unique addresses, of which two are reserved, one each for network and broadcast addresses. Under “classful” addressing, we are limited to providing full Class A, B or C addresses to those requiring IP addresses. With “classless” addressing, we can subnet these addresses quite simply. As stated above, the network portion of the address is equivalent to that portion of the IP address corresponding in base 2 to all ones, and the host address to all zeroes. This means that a Class C address looks like:

11111111.11111111.11111111.00000000 = 255.255.255.0
(**128+64+32+16+8+4+2+1** in the first three positions and 0 in the last). Again, note that this is 24 ones and 8 zeroes, for a total of 32 positions.

Let's say we have one Class C address (192.168.1.0) available for use, but we have two offices with approximately 75 hosts at each location, one in New York and one in New Jersey. While we could simply use the Class A address at each site with each office using unique numbers, we can't connect them together because machines in New Jersey can't find those in New York and vice versa. The reason these two portions of the network can't find each other is because in order for a computer to find another on a network, it assumes an address on its local network (the host portion where all the numbers are zeroes) is directly connected to it, and one on another network is reachable only by going through a gateway.

A gateway is a machine (computer or router) that has two or more network addresses, at least one on the local network and one or more on other networks. A gateway sends any communications not on the local network via one of its other communications devices, depending on the information stored in its routing table. Under classful routing, we would need two half-used Class C addresses for each office, which would be very wasteful of scarce IP addresses.

With CIDR, we can cut the Class C address into two different networks. To do this, we will extend our netmask by one more bit, giving us two separate networks, where before we just had one. This will change our netmask from 255.255.255.0 or 24 ones (hereinafter referred to as /24) to a /25 network, or 255.255.255.128. Both of our new networks will have this same netmask; all other rules remain the same. We now have one network with a network address of 192.168.1.0 and a broadcast address of 192.168.1.127. The other network will use a network address of 192.168.1.128 and a broadcast address of 192.168.1.255.

In the same manner, we can continue slicing up our network into four, eight, sixteen, thirty-two, ... networks. In fact, starting at /8, we can slice and dice until we reach /30. Since we have 32 numbers to work with, a /32 represents just one address, and in this special case, there's no need for network or broadcast addresses. That also means a /31 would represent two addresses, but since one would be the network address and the other the broadcast address, this would leave us with no host addresses—almost certainly undesirable.

Under this scheme, the first octet of the netmask would remain 255, but after that we could change any of the other numbers. Instead of being restricted to 255 and 0, we may find ourselves replacing the first zero in our netmask with any of 128, 192, 224, 240, 248, 252 or 254, except in the last octet as noted above. The network and broadcast addresses would bind each subnet (see Table 2 for details). Now, any network can be referred to by its variable length subnet mask, or the number of ones in the host portion of the address from /8 to /32 (excepting /31). By extrapolation, each host can be referred to directly by its IP address and the VLSM notation, so that it is readily apparent what the network and broadcast addresses and netmask are.

For example, if someone told me to assign my machine 192.168.0.50/27, I would know that the network address was 192.168.0.32, the broadcast address was 192.168.0.63, and the netmask was 255.255.255.224. For those of you who still have problems visualizing how this all translates, I've provided a chart to assist you (Table 3).

You will find more uses for classless addressing than this. CIDR can also give you a way to isolate departments in large organizations to provide better security (by implementing internal firewalls) and decrease traffic on any given network segment, reducing collisions and increasing response times.

Private Address Groups

Another way many companies can expand their pool of usable IP addresses is to take advantage of the private IP addresses set aside for companies and individuals not requiring direct Internet access on all their machines. These numbers can be used as seen fit.

By using a firewall or proxy server that performs network address translation (NAT), called “masquerading” in the Linux community, these machines can still connect to the Internet. The bright side is you won't be routing internal company addresses to the Internet, since most routers are set up to not route these private addresses. Conversely, no one can directly access your systems, so rogue web sites springing up in your company will not come back to haunt you. In order for anyone to access an internal computer, they would have to

either log in to the proxy server first, then continue in, or be redirected by the proxy through the server to the designated machine.

The reference for those addresses we can make use of with no prior coordination is RFC 1918, “Address Allocation for Private Internets”, February 1996. These private addresses are as follows (excerpt from RFC):

```
10.0.0.0 - 10.255.255.255 (10/8 prefix)
172.16.0.0   - 172.31.255.255 (172.16/12 prefix)
192.168.0.0  - 192.168.255.255 (192.168/16 prefix)
```

Note that under the old classful addressing, while the first address segment is one Class A network, the second would actually be 16 Class B networks, and the final segment 256 Class C networks. By implementing a Linux gateway box and setting up some simple rules in **ipfwadm** (generally available with all Linux distributions), we can perform masquerading or Network Address Translation, giving all computers on the private network full Internet access. However, those on the Internet cannot get to any of the computers with private addresses unless one of two things happens. One, the administrator sets up the gateway to act as a proxy server; proxying requests on a particular port to a particular computer, or two, by the Internet user using TELNET to access the gateway box first, then on to the internal computers. Thus, private addresses stay private.

These address groups can also be put to use in private networks that piggy back on the Internet. By using two “live” (non-private) IP addresses, one on each network's “gateway” machine, we can tie two private networks together using Linux's IPIP, IP tunneled inside IP. While this won't provide privacy unless the two gateways are running an encryption program such as **ssh** (secure shell), it can provide a virtual network.

Conclusion

While live Internet addresses are becoming scarce, companies and individuals can maximize use of their current address space and even expand their address space through the use of private addresses. CIDR can also be used to improve security and increase network response time through subnetting.

By staying current with trends in such things as CIDR and Linux's networking software, most obstacles to Internet and Intranet connectivity can be easily circumvented. As CIDR provides everyone with a way to maximize the little we have, private addresses afford us the flexibility to expand beyond those addresses provided by our Internet Service Provider.

David Bandel is a Computer Network Consultant specializing in Linux, but he begrudgingly works with Windows and those “real” UNIX boxes like DEC 5000s and Suns. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from 2,500 feet up in an airplane. He welcomes your comments, criticisms, witticisms and will be happy to further obfuscate the issue. He can be reached via e-mail at dbandel@ix.netcom.com.

Archive Index Issue Table of Contents

Advanced search

User Manager Software

Branden Williams

Issue #56, December 1998

Mr. Williams presents a tool to handle all of your user-administration tasks.

One of the most time-consuming tasks of every system administrator (Linux or not) is user account maintenance. Whether adding or removing accounts, or even occasional suspending or enabling, it is definitely something that can be done by a user other than root. Why spend time going through a stack of papers on your desk with one thousand user accounts to modify when the person who wrote the work orders in the first place could just as easily do it?

Now you may ask, “How are these people supposed to do this job? I am not giving out the root password!” Well, to get around the password problem, I have a quite simple response: use **sudo** (pronounced soo-doo). You can get this neat little gem from http://www.courtesan.com/sudo/. **sudo** allows a permitted user to execute a command as superuser (real and effective UID is set to 0, and GID is set to root's group ID as set in the password file). Using a utility like this, you can permit certain users to run certain programs, such as an **adduser** script or a **chfn** command. Although certain sanity checks must be in place, I have found sudo to be a viable solution to keeping the root password secure.

To get around the problem of letting others mess with user accounts, I created User Manager. Now I rarely spend time dealing with user accounts. User management is done by the technicians, the billing department and the salespeople. User Manager, primarily a Korn shell script, does it all. (The Korn shell can be obtained from ftp://ftp.cs.mun.ca/pub/pdksh/.)

The User Manager script is a framework for your system that can be customized to add in RADIUS support, multi-homing support and domain management. For example, one system I set up builds all DNS, web and stats package configurations and sets up the user account for a multi-home web customer. It was fairly simple to add this support to the script, and it provides a great learning base in system automation.

The Preparation

The User Manager tar archive file can be found at ftp://ftp.inetinc.net/pub/usrmgr/usrmgr.tgz. It includes the **usrmgr** Korn shell script, a C program and a simple Perl script, a README file, an INSTALL file and an ASCII welcome screen. The C program, newpass.c, helps with encrypting new user passwords and the Perl script, loginterp.pl, generates reports from information logged when User Manager is run.

In order to configure the User Manager software, you should know the locations of several common utilities on your system. Some of these include **finger**, **sed**, **edquota**, **sort** and **mail**. You can see the complete list in the sidebar "Programs and their Locations". Make sure all of these are set correctly in usrmgr before running the script. If they are not, the script will not execute properly—steps may be left out.

Once you have the locations of the programs set up, you have some choices to make. Where do you want your log file to go? Where do you want the scripts to reside? Which administrator (or administrators) will be receiving e-mail messages noting the user's actions? Here are the answers I gave when I set up my system:

- Log File **->** /usr/local/adm/usrmgr.logfile
- Scripts Reside in **->** /usr/local/usrmgr
- Administrators **->** brw,matt,billing@inetinc.net

Note that the administrators can be local user names and/or full Internet e-mail addresses. For multiple entries, simply separate each address with a comma and no spaces.

The Report Generator

The reports generated by User Manager's Perl script can be very helpful tools, not only for your system administration team but also for the billing and administrative personnel at your company. With sudo installed, the reports list the user name of each person who ran User Manager, instead of just logging everything as root. Listings 1 and 2 show the two different report formats available.

The simple log in Listing 1 is summary information aimed at system administrators. It lists the number of **adds**, **suspends**, **enables** and **deletes** performed by each user and can be used to track any unwanted or unauthorized users who might be abusing User Manager. If you set up your system to allow only administrators to access User Manager through sudo, you can easily track malicious activity by checking what root is doing. If the machine

where I obtained the Listing 1 data was a production machine, I would be very wary of the one **add** done by root, and would check the detailed logs for more information.

Listing 2 shows a more detailed report that can be turned on or off by setting verbose to 0 or 1 in the loginterp.pl script. My personal recommendation is to leave it on so that you can send these reports to your billing and account managing crew. It is also helpful in investigating any malicious activities which may have shown up in the summary reports.

For example, one entry in the summary report detected root doing an add to a user's record:

```
Function Performed:  User Added
Done by:    root
Login:     jhanish
Password:    ilovesouthpark
UID:      1003
GID:      1003
Real Name:   Joe Hanish
Home:      /home/jhanish
Shell:     /bin/tcsh
Date:      07.29.1998
```

One might deduce that a possible security hole was exploited and now a new user, **jhanish**, has been added to the system. So, we take a look at the /etc/passwd entry to see what else may have happened.

```
jhanish:x:0:1003:Joe Hanish:/home/jhanish:/bin/tcsh
```

In this case, after adding himself to the system, he then created a back door to access the system as root if he wished. Of course, a skilled hacker would not leave traces like this, but someone just playing around can easily be caught.

You may want to set a **cron** job to run loginterp.pl on a weekly or monthly basis to generate report files and send them automatically to administrators through e-mail. For example:

```
6 0 1 * * root /usr/local/bin/loginterp.pl |
mail -s UserMGRLogs root,billing
```

The Scalability

User Manager was built as a basic shell for all your user managing functions. As a system administrator, I realize every system has a unique function, operating system and system administrator's style of managing. User Manager gives you a platform on which to create a customized software package to handle everything you do when managing users. This will give you time to do more interesting tasks without worrying about whether you missed a step in the process.

One other application for this script could be to add in web-hosting support. An ISP that hosts web sites could automate all the steps required to add customers to its systems. To do this, start with the User Manager framework and add in the other steps.

For example, one system I am familiar with is a small web-hosting company that has two main servers. The second server is really only a backup mail spooling system and a secondary DNS. Even though this is a simple example of added functionality in User Manager, the concept can be applied across an infinite number of servers and/or locations.

The User Manager software is on the main system. Once a web customer is added, the script goes out and builds the DNS record, rebuilds the /etc/named.boot file on the fly, passes the configuration to the secondary name server and rebuilds its /etc/named.boot file. After all the configurations are built, it reloads each name server's database.

Once all the DNS is complete, it then takes care of the /etc/sendmail.cw file (this step always caused me problems) and sends a HUP signal to **sendmail** to get it to recognize the changes. When that is done, it actually adds the user account. It then builds the **httpd** configuration on the fly as well as the **stats** package configuration.

The Solution

User Manager is the solution to all your user-administration problems. With the added help of sudo and the report generation program, user management is no longer a worry. Due to the script's scalability and robustness, it can be ported to any system with ease. Even a BSD password database system can have User Manager running on it. Because it is written as a Korn script, it is not limited by any flavor of UNIX. It can be every system administrator's friend and might even cut your work week down dramatically, giving you some time for the things that truly matter in life.

Branden R. Williams is Vice President of I-Net Solutions, Inc. (http://www.inetinc.net/), where he consults with several other companies doing UNIX system and network administration, security management, and system performance tuning. When he is not in the office, he enjoys sailing, playing his acoustic guitar, and astronomy. He can be reached via email at brw@inetinc.net.

X Window System Administration

Jay Ts

Issue #56, December 1998

An introduction to X structure, configuration and customization.

When the X Window System was first released, many complained that the system was big, slow and complicated. My first experience with X was installing one of the earliest releases available for Intel hardware on my 386 running UNIX with 4MB of RAM and an 80MB hard drive. The installation took up most of the drive, and X ran so slowly (with much thrashing of virtual memory) that it was simply unusable. I quickly decided to remove it from my system and went on to “real work”.

However, I got a taste of what X was like and appreciated how the developers took the “high road” in their design. They combined a high degree of versatility and a client-server architecture, at the noticeable expense of performance on what is now generally considered to be archaic hardware.

Today, most computers running Linux have more than sufficient hardware resources to run X with good performance, so running X on an inexpensive desktop system is commonplace. Now that we've all got X running on our desktops, the next hurdle is configuring X and customizing it to meet our needs.

X's Client-Server Architecture

I will start out by presenting a brief description of how X is structured and how the parts interoperate. Once you know this, it will be much easier to make sense of the implementation details.

One of the most basic facts to be aware of is that X is not part of the kernel. Unlike other operating systems, where applications make requests to put up windows, menus, et al., to the operating system API, X is entirely contained in

user space, running as ordinary processes. These processes are classed into two groups: client processes and server processes.

The job of the X server is to handle the interface to the hardware (graphics adapter, keyboard and mouse) and a few additional low-level services such as drawing, color allocation, event handling, inter-client communication and managing a resource database of user preferences.

Clients communicate with the X server through the X protocol, which can be run over interprocess communication (IPC) on one system, or between systems using TCP/IP. This allows an X client to run on one system and use the display, keyboard and mouse on another. Yes, it is even possible to run an X protocol over the Internet.

For example, if I use TELNET to access my ISP and run the command **xeyes**, the program will start up and display a window on my Linux system. The program seems to be running locally, even though it is actually running on the remote system and just using my system for the display. The way this works is that the remote system is running xeyes, while the client (requesting services) and my Linux system are running the X server (which fulfills the client's requests). This, of course, requires that my system be set up to allow it; I use the **xhost** command to allow my ISP's system to use the X server on my system. Also, the **$DISPLAY** environment variable in my TELNET session must be set correctly (see below). I am using this example only as a demonstration; if you want your system to be secure, you will probably not want to allow people to access your X server from an outside network. One trick used by crackers is to put up an invisible window covering your display that catches all keyboard input, including passwords.

Each display is handled by one server process, but many clients can use a display at the same time. One of the most fundamental client types is the window manager, which allows the user to manipulate windows. A window manager performs actions such as drawing “decorations” around windows (borders, title bars and buttons), and provides functionality such as pop-up menus and the iconization of running clients. Desktop environments such as KDE and Gnome are implemented as user-space X clients, as are all other applications that run under X. These applications can be of any level of complexity, from **xlogo** to Netscape Navigator.

X Administration

In the remainder of this article, I will cover some basic X Window System administration. A full treatment would take a whole book, so I'm going to discuss only what I consider the most important points for the novice X administrator. For the most part, I will assume you have X already configured

and running. I will skip over advanced topics such as security and running X over a network (on X terminals or remote systems) as much as possible. This article covers XFree86, which comes with most Linux distributions. If you are running a commercial X server, you may want to skip the section below on configuring the X server, but the rest of the material covered is independent of the server you are using.

X Configuration Files

Most files for the X Window System including executables, libraries, manual pages, include files and miscellaneous other files are kept in the /usr/X11R6 directory tree. There is usually a symbolic link called /usr/X11 to that directory.

The systemwide configuration files for X are in the directory /etc/X11. If you get a listing of that directory, the output will look something like this:

```
X fs mwm xdm
XF86Config fvwm twm xinit
XF86Config.0 fvwm2 wmconfig xsm
```

X Server Configuration

In the above listing, X is a symbolic link to the X server executable, which is in the directory /usr/X11R6/bin. Note that there is also a symbolic link called X in /usr/X11R6/bin, which points to /etc/X11/X, rather than the X server in the same directory. (Yes, there is a reason for doing it that way. Can you figure out why?) XF86Config is the file read by the XFree86 X server while starting up, and contains information about the mouse, graphics card, monitor and a few other things the server needs in order to run. Most of the other objects in /etc/X11 are directories containing sample startup files for programs that have the same name as the directory. The /xdm directory contains startup files for **xdm**, the X display manager. I will discuss the files in that directory later.

The XF86Config file is usually created during the installation of Linux. If you do not yet have X installed (or working properly), you can use the **xf86config** command to create it. Make sure you have collected detailed information about your mouse, graphics card and monitor first. Once you have a working XF86Config, you can modify it to change X's behavior. There are two warnings I want to give before saying anything else.

First, make a backup of the file before modifying it in any way. In fact, this is a good idea even if you do not plan to make any changes. Notice on the previous page that I have the file named XF86Config.0 in /etc/X11. That is simply a copy of the original XF86Config made immediately after Linux was installed. If anything untoward ever happens to my XF86Config file, I can quickly restore it from that backup file. Before making any changes to the “real” file, I make a

backup using a numbered extension, such as XF86Config.1, XF86Config.2, XF86Config.3 and so on. That way, I create a history of my modifications, and can restore the configuration to its original state or to any previous state. This is a fairly common practice among system administrators that I suggest you adopt immediately, if you haven't already. It is also a good idea to make a backup of this type for the version of the file you are currently using, in case it is unintentionally overwritten by XF86Config or a reinstallation of the X software.

The second warning about XF86Config is that the part beginning with **Section "Monitor"** and ending with **"EndSection"** contains very sensitive information. If you mess it up, you could cause physical damage to your monitor! So don't change any of it, at least not until after you have read the man page for XF86Config and the XFree86-Video-Timings-HOWTO.

Now let's have some fun. A couple of simple (and safe) things can be modified in your XF86Config file relating to how your X session appears and functions. Both of these appear near the bottom of the file, in the parts beginning with **Section "Screen"** and ending with **"EndSection"**. Take a look at your XF86Config file. Note that there are a few Screen sections. Each corresponds to a specific X server, labelled by the “Driver” tag. You will need to identify the one that goes with the server you are running. If your graphics card is reasonably new, you are probably running one of the accelerated servers, which correspond to the section that looks like the one shown in Listing 1. Most newer systems use the accelerated server, but if yours does not, don't worry—the sections for the other servers (svga, vga16 and vga2) are similar, just much simpler.

The strings that go with the **Device** and **Monitor** tags are descriptive in nature and not critical. Notice the **DefaultColorDepth** tag, which did not appear in my XF86Config immediately after installation. I added it to set X's default to something more interesting than the usual of 8-bits/pixel, which may run the fastest, but allows for only 256 colors. 16-bits/pixel allows many more (65536) colors, and 24-bits/pixel allows for “true color” of 8 (or more) bits for each of the red, green and blue color components.

Next come the **Display** subsections. Each has a different **Depth** tag, one for each mode (bits/pixel) the server can handle.

The **Modes** tag defines the physical screen resolutions the monitor supports. These must correspond to the **Modeline** entries in the **Monitor** section of XF86Config or they will be ignored. The first valid entry is the default resolution for X server initialization. I am often working with development projects where I target the least common denominator of 640x480, so I use that as my default; however, you might want to start out at 800x600 or higher. If you want to

switch resolutions after the server starts up, hold down both **alt** and **ctrl** on the keyboard and press either the **+** or **-** key on the numeric keypad to cycle forward or backward through the list.

The **ViewPort** tag sets the upper left corner of the virtual display and controls which part of it will appear centered on the screen when X first starts up. This works only if your virtual area is larger than your physical resolution. I prefer to leave this at **0 0**, but try using a few different non-negative values and see what happens. You'll notice the screen is shifted by that many pixels (horizontal and vertical) from normal.

Last, there is the **Virtual** tag, which is used to set the size of the virtual screen area. In 16-bit mode, I may use a physical resolution of 640x480, but I still have the ability to pan around (using the mouse) within a larger virtual area of 1024x768. The virtual screen area is limited by the amount of video memory on the video card. For example, at 1024x768 pixels and 16 bits/pixel (or 2 bytes per pixel), a total of 1.5MB of video RAM is being used. Some cards have only one megabyte of video memory and would not be able to support that configuration. Also, note that the X server may use a small amount of video memory for other purposes, so you may not be able to use all of it for your bit-planes.

Many more things can be done with the XF86Config file than I can describe here. For additional details, check out the XF86Config manual page.

Starting the X Server

There are a few different ways to get the X server started. One of the first methods found by many new users is the **startx** command, which is actually a shell script wrapper for the **xinit** program. Using startx with no arguments should be enough to start up an X session; however, unless you have set the DefaultColorDepth as described above, you will probably get only 8 bit-planes. If you want better colors, you will need to add some arguments, like this:

```
startx -- -bpp 16
```

This starts X with 16 bit-planes. The first two dashes cause the **-bpp 16** to be passed as arguments to the X server, rather than the xinit program. If you get tired of entering the whole command each time, you can create a file named .xserverrc in your home directory and put in it the command to start the X server, like this:

```
exec X -bpp 16 :0
```

See the xinit(1) manual page for details.

When X starts, it will switch to the first available virtual screen and use that for the display. Most Linux systems come with the first six virtual screens assigned as virtual consoles, so the user can switch among them by pressing the **alt** key and a function key (**f1**-**f6**) at the same time. To get to one of those from the X display, it is necessary to add the **ctrl** key, i.e., use **ctrl-alt-f1** to get from X to the first virtual console. Then, to return to the X session, press **ctrl-alt-f7**. If your system doesn't have six virtual consoles enabled, you will have to use a different function key.

Now, if we can have six (or more) virtual consoles, why not have more than one X session? This is done by providing **startx** with more information. The X server needs to know which virtual screen to use and what to name the display.

For example, to start a 24-bit display on virtual screen 8, type:

```
startx -- :1 -bpp 24 vt8
```

and to start an 8-bit display on virtual screen 9, type:

```
startx -- :2 -bpp 8 vt9
```

The **:1** and **:2** are the names that X uses to refer to the displays. The full format for the name is **host:N.M**, where **host** is the host name of the system, **N** is the number of the display on that system, and **M** is the number of the screen (in multi-headed displays using more than one monitor).

The designation **:0** is simply shorthand for the first display on the local system, **localhost:0.0**. The names **:1** and **:2** refer to the second and third displays. To switch among them, simply use the **ctrl-alt-fn** combination.

To see how these work, start up the **:1** and **:2** displays as shown above, and switch to your first display (**:0**) using **ctrl-alt-f7**. Then from a virtual terminal (e.g., xterm, rxvt), run the command

```
xeyes -display :1
```

The xeyes program will run (you won't get another shell prompt), but it is not visible on the screen. Now switch to the second display (**ctrl-alt-f8**) and you will see it. When xeyes exits, you will get another prompt in your shell session on **:0**.

Many X programs support the **-display** option to specify the display to use. Note that the environment variable DISPLAY is set to the default display. If you run the command **echo $DISPLAY** from a virtual terminal in each display, you can see how it is set differently on each one.

When xinit starts up, it starts the X server and then looks for a file called .xinitrc in the user's home directory, which is a shell script that xinit runs. That file usually contains, as a minimum, lines like:

```
xterm &
exec fvwm
```

which start an xterm terminal emulator, then it replaces the xinit process with the FVWM window manager. In turn, **fvwm** looks for its startup file called ~/.fvwmrc. A default for this file can be found in the /etc/X11/fvwm directory. Notice that the xterm process starts running without a window manager. A window manager is not a required part of an X session, but you will probably want to have one.

XDM

Using startx is easy, but if you use X a lot you will probably want to log into it directly without the complication of having to log into a text console first. Direct logins to X are handled using XDM. The files in /etc/X11/xdm are used to define a configuration, and then the simple command **xdm** starts X with an xlogin screen to allow someone to enter their user name and password. Like startx, the xdm command can be entered from a command prompt (as superuser). This is good for testing, but xdm is actually meant to be run automatically during the boot sequence—more on that later. Typing **ls** to get a listing of my /etc/X11/xdm directory outputs:

```
GiveConsole Xresources.0 Xsession.0 Xsetup_2
xdm-config.0
TakeConsole Xservers Xsetup_0 authdir
Xaccess Xservers.0 Xsetup_0.0 chooser
Xresources Xsession Xsetup_1 xdm-config
```

The key file among these is xdm-config, which is the default configuration file for xdm. The xdm-config file defines the basic configuration, including which files to look in for further setup information. The contents of xdm-config look like Listing 2. Note that the names of other files used by XDM are defined here, so it is possible to use different file names or put the files in other directories. The defaults work fine, but be aware that on other UNIX systems, or even different Linux distributions, files may be in a different location. In any case, you can familiarize yourself with the system's configuration by looking at the xdm-config file.

The information in xdm-config is specified using X resources, which is a bit like setting values of data structure fields in a programming language.

The first line of the file sets **DisplayManager.errorLogFile**, which is where xdm writes its error messages. If xdm is not starting properly, take a look at the error file. You will probably find some useful messages there. On older Red Hat

systems and other UNIX systems, the file was placed in /etc/X11/xdm, but in more recent versions (e.g., Red Hat 5.1), it is in /var/log/xdm-error.log. This is in accordance with the Linux File System Hierarchy Standard (FSSTND).

DisplayManager.pidFile (/var/run/xdm.pid) is a file to which xdm writes its process ID. This can be handy if you are adding customizations and you want to restart X to check if they work. Type the command:

```
kill -TERM `cat /var/run/xdm.pid`
```

to kill the xdm process before restarting it. Actually, I prefer the command:

```
killall -TERM xdm
```

which does the same thing. A variation is to replace the TERM (terminate) signal with HUP (hangup); this does not shut down any running X sessions, but does restart xdm with the new configuration (used for any new sessions that are started). If you are doing your X administration from within an X session, you may want to use that method to avoid discontinuities in your GUI services.

The file pointed to by **DisplayManager.servers** (Xservers) is used by xdm to start the X server processes. It contains information that tells xdm how to start each X server process. For example, the line

```
:0 local /usr/X11R6/bin/X -bpp 16 vt7 :0
```

in my Xservers file will start display **:0** on the local system using the command and arguments as provided. To start more than one display, simply add lines to the Xservers file in this same format. If the Xservers file contains these lines:

```
:0 local /usr/X11R6/bin/X -bpp 16 vt7 :0
:1 local /usr/X11R6/bin/X -bpp 24 vt8 :1
:2 local /usr/X11R6/bin/X -bpp 8 vt9 :2
```

three displays will start up when xdm is run—a 16-bit display on virtual terminal 7, a 24-bit display on vt8 and 8-bit on vt9. The **-bpp 16** option is redundant, since I've defined DefaultColorDepth to be 16 in my XF86Config file.

Notice the asterisk in the last few lines of xdm-config. This mechanism is called “loose binding” and is a wild card character used to match all possible field names. The field names in this case are the names of the displays. Display **:0** is referred to as **DisplayManager._0**. (It is _0 for display **:0**, _1 for display **:1** and so on.) The underscore is used instead of the colon because in a resource, the colon is a separator between the resource name and its setting. An asterisk means the same file is used for all of the displays, but when the display is specified explicitly (called “tight binding”), the file is used just for that display. Of

course, it would be possible to use only tight bindings and specify the same file each time, but the loose binding method is easier.

After the X server starts and before the xlogin program is run, xdm looks in the file defined by the **DisplayManager._0.setup** resource (Xsetup_0). This is a shell script containing arbitrary commands, so it has a great deal of versatility. I like to put a more pleasing background behind the xlogin window than the default black and white pattern, so I might use a command like this:

```
/usr/X11/bin/xsetroot -solid darkcyan
```

to make the background (root window) a solid color. To make things more interesting, the lines:

```
/usr/X11/bin/xloadimage -onroot \
  /usr/local/images/tiles/purpleblue2.gif
```

tile the background with an image of my own design. Be sure the xloadimage program is on your system before doing this.

Again, a warning about security. The program(s) run out of Xsetup._* files may have their keyboard and mouse inputs disabled, but if they do not exit before the user successfully logs in, they will continue to run with superuser (root) permissions. For example, if the line

```
/usr/X11/bin/rxvt &
```

were in the Xsetup_0 file, the user who logs in on display :0 is granted a superuser shell, which is not a desired condition. This is an obvious example, but others may not be as obvious, so be careful.

Around Christmas, it might seem cute (and harmless) to put

```
/usr/X11/bin/xsnow &
```

into the Xsetup file to make snow appear to fall while the computer is waiting for a login; however, since it will be running with the user ID of root, the user will not have permission to kill the process after login. Also, many people will get tired of seeing snow falling in the background of their X sessions after awhile. Fortunately, there is a way to make the **xsnow** process exit before the user's login session begins. First, add a line that saves the process ID of xsnow immediately after the one that starts it, like this:

```
/usr/X11/bin/xsnow &
echo $! >/var/run/xlogin_xsnow.pid
```

After the user is authenticated by xlogin and before his X session starts, xdm runs the shell script named in the **DisplayManager._0.startup** resource (GiveConsole). This is normally used to change the ownership of /dev/console to the user, so that error messages directed to the console can be displayed in the X session, using xterm or rxvt with the **-C** option, or with xconsole. However, you can add whatever you want to the script. For example, the following lines force xsnow to exit:

```
kill -9 `cat /var/run/xlogin_xsnow.pid`
rm -f /var/run/xlogin_xsnow.pid
```

Now, here's an exercise for you. Many sites want the ability to shut down the system directly from the xlogin screen without requiring the user to log in or **su** with the root password. With your favorite text editor, create a Tcl/Tk script named **xlogin_buttons** that contains the lines shown in Listing 3, and make it executable with the command:

```
chmod +x xlogin_buttons
```

Now follow the above xsnow example to modify your Xsetup_0 and GiveButtons scripts to use the Tcl/Tk script instead of xsnow. I put the script in /etc/X11/xdm and put these lines in my Xsetup_0 script:

```
/etc/X11/xdm/xlogin_buttons &
echo $! >/var/run/xlogin_buttons_0.pid
```

and these two lines in my GiveConsole script:

```
kill -9 `cat /var/run/xlogin_buttons_0.pid`
rm -f /var/run/xlogin_buttons_0.pid
```

Be sure to check that the Xsetup_0 and GiveConsole files are defined in the xdm-config file.

Along with the **DisplayManager._0.startup** resource is **DisplayManager._0.reset** (TakeConsole), invoked after the X session ends and before xdm resets the X server prior to the next login. Normally, this simply changes the ownership of /dev/console back to root, but you can add customizations there too.

Configuring xlogin

The **xlogin** program identifies and authorizes the user by accepting the user name and password. These are entered at the prompts in the xlogin window. If you want to change the xlogin display, take a look at the file pointed to by **DisplayManager*resources**, which on my system is called /etc/X11/xdm/Xresources. That file contains resource definitions for the xlogin and other programs started by xdm before the user's X session begins. Rather than having xlogin display the host name of my system, I prefer the message

“Welcome to Linux” colored blue. To do this, I define the **xlogin*greeting** and **xlogin*greetColor** resources as shown in Listing 4.

A security consultant might wince at seeing a system that is configured to say “Welcome” to any user who happens to pass by and “Try Again” if they don't guess the right user name/password combination. I do this only on my home system. If you're working in an academic or corporate environment, you might want to use something like:

```
xlogin*greeting: CLIENTHOST
xlogin*fail: Authorized Users ONLY!
```

Color Specification

In the above example, most of the colors are specified as RGB triplets in the form of **#rrggbb**. You can use 1 to 4 hexadecimal digits for each primary color, so to specify a not-totally-bright red, **#c00**, **#c00000** and **#c00000000000** are all equivalent. You can use color names like **white** (equivalent to **#ffffff**) or **black** (equivalent to **#000**), as in the above example. To get a list of color names X knows about, use the **showrgb** command. These two methods have been available in the X system since its first public release and are somewhat limited.

In release 5 of X11, new methods were added. One of the main problems with the older method is that a color specified in the 3-digit format, which provides only 4 bits for each primary color, may work fine for a display with an 8-bit color depth, but on a 16-bit or 24-bit display it will not look right. For example, **#fff** will display as bright white on an 8-bit display, but will be an off-white (**#f0f0f0**) on displays with 16- or 24-bits/pixel. You can get around this as I did above by always using at least 6-digit color specifications or using the new Xcms **RGB** method,

```
RGB:f/f/f
```

which automatically expands to **#ffffff** for displays of more than 8-bits/pixel. As with the original method, 1 to 4 hexadecimal digits are specified for each color. But with the new method, you can use a different number of digits for each. Instead of being taken as absolute numbers, the digits are used as scaling factors. For example, a single digit **9** represents 9/15, and **09** represents 9/255.

If you don't like using hexadecimal digits, you can use the RGBi (RGB intensity) format, like this:

```
RGBi:1.0/1.0/1.0
```

That will also produce a bright white. The values for red, green and blue are specified as floating point numbers between 0.0 and 1.0, inclusive. There are

also other, much more complex color spaces such as TekHVC (hue, value, chroma) and several CIE formats.

The User's X Session

By now, you should have a very good idea of how to configure xdm, so I want to tie up a few loose ends before covering how to start xdm automatically.

One file you might want to take a look at is one named by the **DisplayManager*session** resource in xdm-config. This file (Xsession) is yet another script. It is run by xdm to create the user's X session. Typically, it defines the file .xsession-errors in the user's home directory to be the error log file for X programs (the “clients” of the client-server architecture). The .xsession-errors file is truncated to avoid confusion with errors that happened in the previous session, then both standard output and standard error output is redirected to it. In addition to your xdm error file, the .xsession-errors file is a good place to check for clues if your X session is not starting properly.

Next, the file .xsession in the user's home directory is executed. From the user's perspective, xdm uses the .xsession file in the same way startx uses .xinitrc. However, there are a couple of differences. First, .xinitrc must be a shell script, but .xsession can be any executable program (and must have its execute bit set). This allows for additional flexibility, although .xsession will usually be a shell script that is very similar (and possibly identical) to .xinitrc. It is possible to make one a symbolic link to the other to simplify management and to ensure that startx and xdm both create the same working environment.

Second, when the X session is started by xdm, the user has not yet started a login shell, and the shell's startup scripts (e.g., .bash_profile and .bashrc) have not been run. Because of this, it is necessary to set (in .xsession) those environment variables, such as PATH, that must be available for any programs run from .xsession or any window manager or other program started from that script.

I've just briefly described the default behavior of the /etc/xdm/Xsession script. Usually it is left alone, and customization on a per-user basis is done with the .xsession program in the user's home directory. However, it is also possible to create system-wide customizations by modifying Xsession.

Running xdm Automatically

After you have used xdm from root's command line to successfully start an X session, the next step is to run xdm automatically during system initialization. This can be done in several different ways. I will describe three—the normal

way, an odd way and a weird way. Take a look at your /etc/inittab file. You should find these two lines:

```
id:3:initdefault:
x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

The first line sets the default runlevel to 3 (full multi-user mode, with networking) when the system is booted, and the second tells the init process to run xdm when the system's runlevel is 5. On some Linux systems, such as Slackware, this may be 4.

The normal way to have the system run xdm automatically is by changing the first line to:

```
id:5:initdefault:
```

This will cause the system to boot to runlevel 5 instead of runlevel 3. In the second line, “respawn” tells init that if xdm exits, to immediately restart it. Startup scripts will be run from /etc/rc.d/rc5.d rather than /etc/rc.d/rc3.d. This means if you have configured your runlevel 3 daemons just the way you want them, you will have to do it again for runlevel 5.

If that seems like too much bother, use the odd method and change the second line instead of the first one, like this:

```
x:3:respawn:/usr/bin/X11/xdm -nodaemon
```

This will start up the xdm process in runlevel 3 instead of runlevel 5, preserving your runlevel setup.

Finally, the weird way is to start xdm like any other daemon process and ignore the /etc/inittab file entirely. Add a script to the directory /etc/rc.d/init.d that looks like this:

```
#!/bin/sh
# /etc/rc.d/init.d/X.init - Start X Window System
echo "Starting X Window Services: xdm"
/usr/X11/bin/xdm
```

Then, put a symbolic link to the script in the directory /etc/rc.d/rc3.d. When the system is booted, init runs these scripts in the same alphanumerically sorted order that the **ls** command would display them. On my system, I put in a link called S97X that causes X to be started after almost everything else. Take a look at the other files in the rc3.d directory (using **ls -l**) and follow their examples. This method can be handy, because it doesn't restart xdm each time xdm exits, and sometimes that might be desired. A simpler way to do the same thing using inittab is by typing the line:

```
x:3:once:/usr/bin/X11/xdm -nodaemon
```

One note of caution is needed here. The /etc/inittab file is one of the most critical files on your system. If you mess up your inittab file, your system may not be able to boot, so maybe that weird method isn't so bad after all.

Conclusion

Well, there you have it. I did my best to crunch a book on X Window System administration into one magazine article. I've covered most of the basics of managing X, but also left out quite a bit. If you want more information, check out the sources of definitive documentation listed in the “Resources” sidebar.

Jay Ts has been using UNIX since the year 6 B.X. (before X), and now provides consulting services for Linux. He can be reached at jayts@bigfoot.com; his web page is at http://www.kachina.net/~jay/.

Archive Index Issue Table of Contents

Advanced search

LJ Interviews Linus Torvalds

Marjorie Richardson

Issue #56, December 1998

With 2.2 on the horizon, *LJ* once again talks to the man who started it all—Linus Torvalds.

Marjorie: Everyone wants to know about the new kernel in 2.2. What new features have been added? Anything going away?

Linus: Most of the new 2.2 features are about performance, especially on high-end hardware. The SMP support is much better, and the kernel is more aggressive about caching file names, etc. Also, it works on more hardware.

There have been some nervous people wondering whether it would still work on the small machines, and I essentially spent the last week making sure it still does. We had some tweaks that needed to be made simply because there had

been bit-rot with respect to small machine behaviour—none of the main developers use small machines much during the development phase.

Features going away? I think we're phasing out some of the support for things that people no longer seem to be using very actively, so yes, some of the esoteric code may be gone, partly because nobody cared to maintain it. On the whole, it's all there, larger and better than ever.

Marjorie: You seem to be quite excited about the addition of SMP—tell us about it. Will having this feature open new doors for Linux?

Linus: We had SMP support in 2.0 too, thanks mainly to Alan Cox. The new thing about 2.2 is that the SMP support is no longer just a tacked-on feature no one actually trusts, but is much better integrated. I've done all my development on SMP machines for the last year and a half.

With 2.2, the Linux SMP code comes of age—it's still not a stately old statesman, but more like a boisterous young teenager—it's there and it's reliable. It's going to be polished up in the future, because we're still cutting some corners to get it up easily and reliably, but now it's really more a matter of polishing than rewriting completely.

Marjorie: I've been talking to one of the local ISPs, and they have been having problems with both NFS and NIS. The main problem with NIS is that the newest version hasn't been ported to the SPARC. Has any work been done in these areas? Who is working on it?

Linus: I have to admit I haven't followed the SPARC port very closely at all. Mostly I've been involved with the Alpha, and even that has been surpassed by my main interest in SMP. You'd better ask others about the SPARC side.

Marjorie: Is there anything different in the installation procedures that users should be watching out for?

Linus: When it comes to the kernel, not really. The module loading is different, and some of the system utilities need to be reasonably current with the latest kernels, but on the whole if you have a reasonably recent distribution, you can just plop in a new kernel and it will work (apart from maybe upgrading your **pppd**, etc., details).

Marjorie: What do you think is missing from Linux? What new features will we be seeing in the future?

Linus: There are many “big system” features we haven't quite gotten to yet, but little that impacts most users. Clustering, high-end SMP scalability (expecting

scaling from 8 to 32 CPUs is fairly unrealistic right now), journaling file systems, etc. Few people need them; the ones who do will get them done eventually.

I think most of the missing functionality a lot of people care about tends to be in “user land” rather than the kernel. That was true some time ago; it's become even more true these days. Happily, it's also being addressed more, both on a high-end server scale (Oracle, etc.) and on a lower-end desktop scale (Corel, KDE, Gnome, etc.).

Marjorie: What is the status of Wabi and Wine? Will the time come when MS Windows 95 and 98 programs will run on Linux?

Linus: I don't think you'll see all Windows programs running, but yes, I still believe that Wine can do it. (Wabi seems to be a dead product, and fairly uninteresting these days since it does only 16-bit programs.) Wine is still improving, and some people are actually using it for what they need (mainly Quicken and a few games, it seems).

Marjorie: Do you think a standard GUI is a necessity for Linux? Will any of the currently available desktops (KDE, GNOME, etc.) fill the bill?

Linus: I don't think it's a necessity to have a *standard* GUI. We need a better interface than plain X and TWM, but FVWM took us a long way, and KDE and Gnome are tackling the integration and complete desktop issues rather than just the window manager. I think we'll end up with both KDE and Gnome for awhile, and they'll eventually do the same things and work pretty much the same. Then you'll choose whichever you like more, because all the programs you want to run will run on both.

Marjorie: People always ask you, “What direction do you want to see Linux take?” Let's reverse that: “What directions do you want to make sure Linux does not take?”

Linus: I want to make sure Linux doesn't stagnate. I like to see new areas open up—the people who did the port of Linux to the PalmPilot must be crazy, but I enjoyed seeing that kind of thing happen.

Marjorie: When did you realize that the “chaos Linux development” model was really working?

Linus: I had never realized it wouldn't work. It wasn't planned chaos, and I never thought there would be problems. And there never really were. It worked out of the box—it's just scaling up to a larger scale.

Marjorie: You seem to be at every show that involves Linux. How do you balance traveling, work and family and stay sane?

Linus: Lots of medication.

Heh. Actually, I haven't been traveling too much lately. I did a reasonable amount of flying earlier this summer because there were a few conferences close to each other, but on the whole I try to avoid going to too many conferences. They're fun, but only when done in moderation.

Not traveling still doesn't mean I have tons of free time, obviously, and there have certainly been some weeks when I considered myself too busy. Things tend to calm down every once in a while, and I can take a breather.

Marjorie: What can you tell us about working at Transmeta? (Or is it still a secret?)

Linus: I still can't tell you anything but that it's a ton of fun.

Marjorie: On a personal note, tell us about your family and living in the U.S.

Linus: We like it here. My younger daughter was born here, and as such is a dual citizen of both the U.S. and Finland. We haven't really had too many problems getting used to it being sunny all the time, and going around in T-shirts and shorts for 70% of the year.

We've had a few silly paperwork issues (can you believe getting a driver's license took nine months because it had to go through the INS?), but all in all, it's been very enjoyable.

Marjorie: What's your favorite breakfast?

Linus: Breakfast? You have time to eat breakfast? I drink a cup of cappuccino every morning to wake up.

Marjorie: Any words of wisdom you'd like to leave us with?

Linus: Nope. “Be good to each other” and “don't you eat that yellow snow” have both already been taken.

Marjorie: Thanks for your time.

Archive Index Issue Table of Contents

Advanced search

Building a Web Weather Station

Chris Howard

Issue #56, December 1998

Mr. Howard tells us how he gathers and outputs weather information to the Web using Linux, Perl and automated FTP.

Last fall, my family and I moved from central Iowa to the little mountain resort town of Estes Park, Colorado. Estes Park is a beautiful town at the east entrance of Rocky Mountain National Park. More than three million tourists visit the park each summer.

When we moved here, I brought along my fledgling consulting business, Daylight Software, and set up web pages to drum up a little work. In a flash of inspiration, I decided I would either buy or build weather station equipment and offer weather data on the Web. Visitors from around the nation—and the globe—would see my web pages, and Daylight Software would be established as a Linux consulting powerhouse. Well, maybe it wouldn't lead to global domination, but it would surely be a good thing. I saw it as a community service, since no public weather reporting service was available, other than the time and temperature sign on one of the local banks.

After some investigation, I decided that my hardware development skills were not sufficient to design and build weather sensors. I shopped around and eventually purchased a Texas Weather Instruments “Weather Report” WRL-25 system from American Weather Enterprises of Media, PA (http://www.americanweather.com/).

The WRL-25 is like many weather stations in that it includes an RS-232 connection and comes with DOS/Windows software for downloading and viewing the gathered data. It has sensors for wind speed, wind direction, temperature, humidity, atmospheric pressure and rainfall. Using a regular television antenna mast and fittings, I mounted the weather station sensors on the roof of my house, snaking the sensor cables through the attic space and

down to my office in the basement. I mounted the handsome display unit on a shelf in the office.

Across the room from the weather display is my array of computers, including my Linux workstation. I am running the Red Hat 4.0 Colgate release, 2.0.18 kernel, on my AMD 486DX-4 120MHz ISA server. I built a cable and attached the weather display to the Boca AT-66 serial card in the server. I then wrote some Perl scripts to build HTML and GIF files and upload them to my ISP, and to manage the data readings put out by the weather station.

Equipment Installation

The sensors are mounted on two ten-foot sections of steel TV antenna mast, available in the U.S. from stores such as Radio Shack. Approximately four feet are buried in the ground, and the rest of the mast is vertical at the gable end of our single-story house. The mast is attached with a TV mast bracket at the roof line of the house. About six feet of mast projects above the roof line.

Figure 1. Weather Sensors on TV Antenna Mast

The sensors came supplied with clamps and hardware to attach them to the mast. (See Figure 1.) I followed the installation instructions and mounted the wind direction/speed sensor module pointed north at the top of the mast. The temperature and humidity sensors are in a “pagoda” enclosure to protect them from direct sunlight, and the pagoda is mounted about three feet above the roof line. The rain collector is mounted at the roof line; I used a carpenter's level to mount it properly.

Figure 2. Wires from Junction Box into House

Multi-wire cables from the sensor modules go into the bottom of the junction box, where they are plugged into matching connectors from 100-foot cables which run to the weather display unit. (See Figure 2.) All excess cable is coiled up and attached securely under the eave. Cables going into the junction box were left drooping slightly, to encourage rain to drip off instead of flowing into the junction box.

The 100-foot cables run through the attic and out through a hole at the peak of the eave. The hole was later filled with caulking to discourage squirrels and other pests from getting in. From the attic, I drilled a hole in the top plate of an interior wall and snaked the cables down to the basement. A fancy wooden switchplate made for cable TV installations serves to mask the hole where the cables come through the wall into my office/computer area. The cables connect to the back of the WRL-25 display unit.

I ended up crawling through the blown-in fiberglass insulation in the attic more times than I care to remember. If you do this, be careful—wear a dust mask or respirator and long sleeves and trousers. The attic was hot. I considered this to be the most difficult part of the installation.

RS-232 Connection and WRL-25 Configuration

I had to set the current time on the display unit and change some of the option settings. Other than that, the sensors and other features of the WRL-25 were all calibrated and ready to run.

On the back of the display unit is a 9-pin serial pigtail. The WRL-25 came with a mix-and-match 4-ended 25 and 9 pin serial cable for adapting to whatever connectors are available. I did not use that particular cable. My Boca Io-AT-66 six-port serial card uses RJ-45 sockets for its connections. So, I made a cable from a ten-foot scrap of eight-wire twisted pair that already had an RJ-45 connector crimped on one end. Pin assignments and wiring information were included in the Io-AT-66 documentation.

The WRL-25 can be programmed to periodically send a status report over the serial line. This can be used to print directly on a serial printer. I programmed my unit to send a reading every 5 minutes. I also changed an optional setting to allow the rainfall rate report and to print a daily Max/Min report at the end of each day. I set the serial line data rate to 9600bps.

The unit can also be directed using single-character commands through the serial connection. I rarely use this feature. The PC software that came with the unit can be used to set various options and settings, as can the buttons on the display panel. So, more sophisticated programming could be used to query the unit on demand or change various features. I elected to use the simple logging feature to gather my data.

I was able to run a quick test on my setup by using the following command:

```
cat < /dev/ttyS19
```

This command takes any input appearing on the /dev/ttyS19 serial line and echoes it to the screen. Pushing the manual report button on the WRL-25 produced a one-line report on the screen. I was a happy camper!

WRL-25 Data Report Format

Reports that come down the serial line look like this:

```
17:05 08/09/97 WSW 00MPH 460F 069F 057F 085% 23.42F 00.00"D
01.39"M 00.00"R
```

The first two fields are time and date. The current time and date are maintained on the WRL-25 display unit. I have not noticed any great drift in the time setting.

The second two fields are wind direction (WSW—West by South West) and speed (00 MPH—miles per hour).

Fields 5 through 7 are temperature readings in degrees Fahrenheit. Field 5 is not connected to any sensor, so it should be ignored. Field 6 is indoor temperature and field 7 is outdoor temperature.

Field 8 is relative humidity, which in this example is 85%. Field 9 is atmospheric pressure in inches of mercury accompanied by a single character for falling (F), rising (R) or steady (S).

The last three fields are daily rainfall and monthly rainfall, both in inches, and rainfall rate in inches per hour.

At the end of each day, two lines of daily minimum and maximum readings are reported:

```
Max 08/08/97 WSW 24MPH 460F 074F 081F 100% 23.42" 00.00"D
01.39"M 00.00"R
Min 08/08/97 SW 00MPH 460F 068F 044F 021% 23.28" 00.00"D
01.39"M 00.00"R
```

These lines are in the same format as the other reports, except for the first field which marks these records as Max/Min reports. The readings for daily min/max are independent. In the above example, the high temperature for the day was 81 degrees F and the highest humidity reading was 100%. These readings did not occur at the same time and are unrelated, except that both are the maximum for that particular statistic.

Programming for Data Collection

My first pass at a data collection script was a simple cat command:

```
cat /dev/ttyS19 >> data1
```

It worked—mainly by accident. The next time my machine was rebooted, it didn't work at all. When I fired it up, the weather station console started spitting out all sorts of long reports. After a little head scratching, it became obvious that regular character echoing was feeding back command characters—not at all what I had in mind.

My current script is called weatherd.pl and is shown in Listing 1 with blank lines removed. Line 8 sets the variable **$TTY** to be the first command-line argument.

Line 10 resets the terminal to the appropriate speed, parity and non-echoing. I used the **stty** command to get the terminal settings the way I wanted them, then saved the setup to a file. This stty command reads from that file and sets the terminal to the saved configuration.

Data is kept in a file, with the name of the file being the current date. Each day's data goes into one file (lines 14-16). The print statement on line 16 helps me feel confident things are working right. Beginning with line 27, for each data line that comes in from the tty, we check to see if it is a minimum or maximum and that it is still today's data. Minimum and maximum data go into separate files.

To start weatherd.pl, I added the following single line to my /etc/inittab file:

```
ws:2345:respawn:/home/weather/bin/weatherd.pl /dev/ttyS19
```

This line starts up weatherd.pl and respawns it if it should die for any reason.

Programming for Data Display on WWW Page

Now I had the data coming in from the weather station, getting picked up by weatherd.pl and thrown into a file using the date as its name.

The next step was to format the data into an HTML file for display over the Web. I wrote a Perl script (Listing 2) that takes the last line from the current data file, in combination with a template HTML file, and fills in the weather information. It also calculates the corrected atmospheric pressure and the dew point. (The dew point calculation was given to me by John Kleist, Colorado Climate Center, johnk@loki.atmos.colostate.edu.) Last of all, it looks in yesterday's Max/Min files and puts those values in the output HTML.

I also wrote a script, plotdays.pl (Listing 3), that plots some of the interesting statistics for the last few days.

Programming for Reliable Periodic FTP Upload

Finally, I have a master program called loop.pl, which I wrote to reliably connect to my ISP, set up my PPP connection, transfer e-mail, set the system time and upload the weather data.

First, we have to look at how to automate an FTP connection. FTP is usually used for interactive network file transfer. To use regular FTP, you need to have an account on the remote machine. In this case, I use my ISP shell account with Front Range Internet (frii.net). A user can set up an automated FTP session by using a file called .netrc in their $HOME directory. I added to my system a user

called “weather” dedicated to owning the weather data files and scripts. In the /home/weather/.netrc file, I have the following lines:

```
machine ftp.frii.net
login:
password:
macdef init
cd public_html
put /home/weather/WWW/wscurrent.html wscurrent.html
put /home/weather/WWW/wsplot.html wsplot.html
put /home/weather/WWW/temp.gif temp.gif
put /home/weather/WWW/pressure.gif pressure.gif
put /home/weather/WWW/raind.gif raind.gif
put /home/weather/WWW/winds.gif winds.gif
quit
```

The statements in this file define and execute a macro called **init**. All I have to do is start FTP and this script attempts to run the macro, uploading my HTML and GIF files to the appropriate place on my ISP account.

Both Listing 1 and Listing 3 are called from a short script called doup, Listing 4, which also does the actual FTP call.

Last but not least, loop.pl (Listing 5), is executed continuously by the root user. The only tricky part of loop.pl is the necessity to recover if a command gets stuck due to a loss of connection with the ISP. I wrote a replacement for the regular Perl “system” function which allows me to specify a timeout for each command. If that command exceeds the time alloted, it and all of its children are killed. Loop.pl also has other jobs to do, like downloading and uploading my e-mail and setting the system time using **ntpdate**. While those jobs are happening, I have a child process simultaneously running the doup script. If the ancillary jobs finish first, they give doup a little more time to complete. With a good connection, there is no problem getting everything done in less than one minute.

I have two different dial-up scripts, because there are two possible telephone numbers for the computer to try when dialing my ISP. If the primary number fails, it tries the second number. If both fail, it sleeps for 60 seconds and tries again. The goal of the sleep statements in loop.pl is to have the connection occur approximately every 15 minutes. Otherwise, the comments in the code pretty much cover all of the details.

Unresolved Issues

One of my main concerns is lightning. Fortunately, our site is in a wooded area that is somewhat protected from direct lightning strikes. Still, I would feel even better if the remote weather sensors were electrically isolated from the display unit and my Linux machine. Some sort of optical isolation would probably work.

Also, I may have to develop an automated way to set the clock on the weather display unit.

Otherwise, the system seems to be working pretty well. With weatherd.pl running from the inittab, and with loop.pl in my startup rc files, the weather station monitoring continues after a reboot without manual intervention. Power loss or server failure will interrupt the service, but neither of those are common occurrences.

Figure 3. Weather Web Page

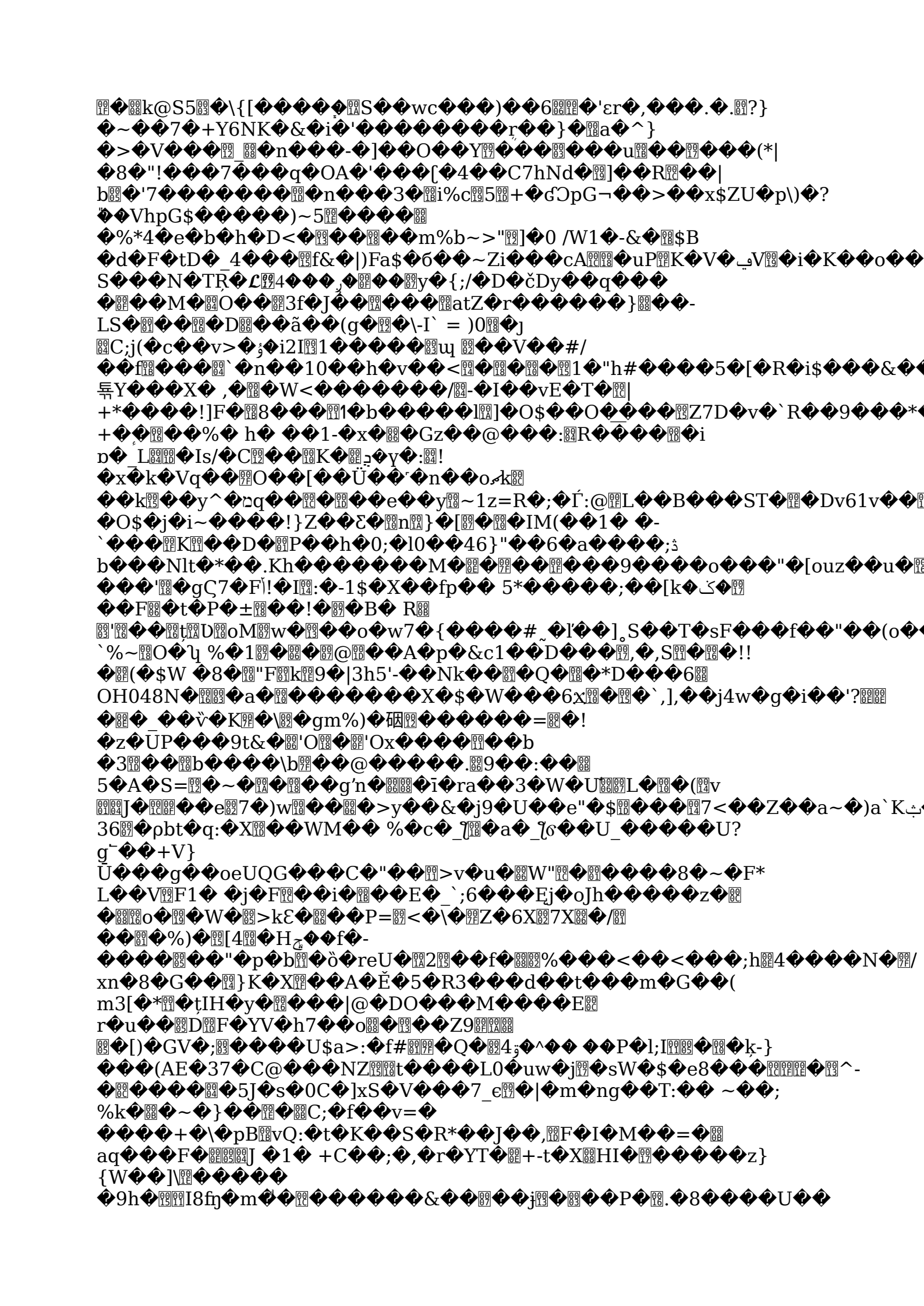
My weather station web page (see Figure 3) can be found at http://www.frii.net/~daylight/wscurrent.html. Please stop by and check it out. If you have the opportunity, come to Estes Park and see what a beautiful place it is.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue56/2538.tgz.

Chris Howard (daylight@frii.net) noodles around with Linux stuff in Estes Park, Colorado, where the winters aren't too cold and the summers aren't too hot, and the view out the window is always beautiful.

Archive Index Issue Table of Contents

Advanced search



Samba's Encrypted Password Support

John Blair

Issue #56, December 1998

How SMB-encrypted passwords actually work, and a walk-through of the steps required to enable encrypted passwords in Samba.

By default, Samba uses plaintext passwords to authenticate clients who access network resources. Samba also supports the use of LanManager- and NT-encrypted password authentication. Using encrypted passwords with Samba has its advantages and disadvantages. On the positive side, encrypted passwords mean that plaintext passwords cannot be “sniffed” off the network when users log in to a Samba share. This is particularly important when users connect to a Samba server across a public wide-area network, like the Internet. Furthermore, the latest service packs for Windows 95 and Windows NT do not allow plaintext authentication to be used when connecting to an SMB server. When using the latest version of Windows, either Samba must be configured to use encrypted passwords, or the registry must be edited to enable plaintext passwords.

On the negative side, using encrypted passwords requires some extra administrative work. The SMB-encrypted-password algorithm is incompatible with the standard UNIX encryption method. As a result, a second password file containing the LanManager- and NT-password hashes for each user, must be created. If someone makes use of other services on the server, a technique to keep both password files synchronized will have to be used.

As of Samba version 1.9.18, the best reason to not use encrypted-password authentication has been eliminated. Previous versions of Samba made use of a Data Encryption Standard (DES) library to compute LanManager password hashes. Because it was linked against a DES library, a compiled version of Samba would be illegal to export from the United States. Strong encryption, like DES, is still considered a munition by U.S. law. To make it easier for stateside mirrors of the Samba FTP archive to distribute Samba, precompiled Samba binaries usually did not contain support for encrypted passwords. Beginning

with version 1.9.18, Samba uses a crippled version of DES that is still suitable for calculating LanManager hashes, but is legal to export from the United States.

Since the latest service packs to Windows NT and Windows 95 disable the ability to connect to shares, setting Samba to process encrypted passwords has become even more desirable. While it is possible to edit the registries on all of your machines to re-enable the use of plaintext passwords, it is probably easier to configure Samba to use encrypted passwords.

SMB Password Hashes

There are two styles of SMB-encrypted-password authentication: LanManager and Windows NT. Both techniques use a file which contains hashed values of a user's password, not plaintext passwords, just as the standard UNIX authentication method does. However, each uses its own technique to generate this hash.

LanManager-style hashes are generated using this algorithm:

1. Convert the password entered by a user to all capitals.
2. Either truncate the resulting password to 14 characters if it is longer, or pad the password with null bytes if it is shorter than 14 characters.
3. Use this 14-byte value as two 56-bit DES keys to encrypt a secret 8-byte value twice, creating a 16-byte value. This value is the hashed password which is stored in the password file. This secret value is a string consisting of the characters **KGS!@#$%**.

Unfortunately, this algorithm has a serious weakness. First, the password is converted to all uppercase before it is hashed. This reduces the number of possible characters in the password from 95 to 69. However, since most punctuation characters are also denied, the number of possible characters is closer to 40. This reduces the actual size of the keyspace from 9514 to about 4014. Further, each half of the password is encrypted independently. This means that either half of the password can be recovered without recovering the other half. A better approach would have been to “chain” the two encryptions together by feeding the output of the first encryption into the second encryption. This technique is called cipher block chaining. The entire 16-byte hashed password has a possible keyspace of 2128, or 3.4 x 1038. Not using cipher block chaining reduces the number of possible hashed passwords from this value to 2(407) or 3.2 x 1011.

As a result, it is possible to use brute force to crack the LanManager passwords in a reasonably short period of time. L0phtcrack, from L0pht Heavy Industries (http://www.l0pht.com/), has been demonstrated to exhaust the keyspace in 62

hours on a quad Pentium Pro 200 SMP box. Since even paranoid users rarely change their passwords more frequently than every few weeks, systems are vulnerable to system crackers with more conventional hardware at their disposal.

In contrast, the Windows NT hashing algorithm is much stronger. The NT hashing algorithm consists of computing a 128-bit MD4 hash of a Unicode version of the user's password. Since the password is not truncated and makes the entire Unicode character set available for use, this technique makes use of the entire 128-bit keyspace. Unfortunately, for the sake of backwards compatibility, nearly all SMB servers allow access using either hashing technique. This means that an SMB server, like Samba, must store both hashes in its password database. As a result, at least for the immediate future, it doesn't matter that the NT algorithm is stronger.

The Process of SMB-Encrypted Authentication

The process of SMB-encrypted authentication is the same whether LanManager or NT encryption is being used. When a client indicates that it can support encrypted-password authentication during the protocol negotiation stage, the server will respond with a random 8-byte value known as the challenge. The challenge is different for each client request. The server stores the challenge until the client is authenticated or denied access.

After the client obtains the password from the user, it computes the hash value using one of the previously defined algorithms. The resulting 16-byte value is appended with 5 null bytes. This 21-byte value is used as three 56-bit DES keys to encrypt the 8-byte challenge value three times. The resulting 24-byte value is known as the response.

The server also executes the same algorithm, using the stored hashed password. If the value the server computes matches the value returned by the client, the client had to have known the password or at least the 16-byte hash value generated from the password. As a result, access will be granted as an authenticated user. Otherwise, access is denied. In either case, a plaintext password was not passed over the network, where it could be sniffed by an eavesdropper.

However, there is a snag with using this technique. Unlike the UNIX password hash, the SMB password hash is a password equivalent. This means that even though it isn't plaintext, it might as well be. It is the responsibility of the authentication client to accept a plaintext password and generate a hash before using it to encrypt the challenge from the server. Unfortunately, a custom client can be written that, rather than generating the password hash from a plaintext password, simply accepts a password hash and uses it to

generate the appropriate response to the server. **smbclient**, a component of the Samba suite, can be modified to accomplish this task. To sum up, even though it is possible to crack the LanManager password in a reasonably short period of time, it isn't actually necessary to gain access to a share if you already know the password hash. The bottom line is that the Samba-encrypted-password file and the NT Security Accounts Manager (SAM) both contain sensitive information. Don't let the fact that it is “encrypted” lead you to believe that you don't have to protect it from snoopers.

Using Encrypted Passwords in Samba

Configuring Samba to use encrypted passwords is easy—just include this setting in the global section of your configuration file:

```
encrypt passwords = yes
```

Encrypted passwords work with all three security levels: share, user and server. Setting the security option to **user** or **share** requires that the Samba-encrypted-password file exist. If security is set to **server**, no further configuration is necessary, since all authentication requests will be passed off to a different SMB server. The server security option provides an easy way to integrate a Samba server into an existing NT domain. However, most installations of Samba will use user- or share-level security. The most common configuration is this:

```
security = user
encrypt passwords = yes
```

Both the share and the user modes require the smbpasswd file, which contains the LanManager and NT password hashes for each user who will be accessing the Samba server.

The Samba-encrypted-password file, smbpasswd, is stored by default in /usr/local/samba/private. This directory is normally owned by root, with its permissions set to 500, so that only root can look at its contents. However, this configuration isn't strictly required—your smbpasswd file can be stored any place you wish. The Samba Red Hat package stores the smbpasswd file in the sensible location of /etc/smbpasswd. Wherever the smbpasswd file is stored, its permissions should be set to 600 (only user read and write) and it must be owned by root. It must not be possible for any user other than root to read this file.

Users can be added to the smbpasswd file in several ways. The best way is to use the **smbpasswd -add** command. For example,

```
smbpasswd -add jdblair foobar
```

will add an entry for **jdblair** with a password of **foobar**. When adding a user while the Samba server is running, this command must be used to ensure that the smbpasswd file is properly locked before it is modified.

Another way to create a new smbpasswd file is to use the mksmbpasswd.sh script that comes with Samba. This script is, oddly enough, stored in the /source subdirectory of the Samba distribution. For example:

```
cat /etc/passwd | mksmbpasswd.sh > \
        /usr/local/samba/private/smbpasswd
```

If the system uses NIS, you should use this command:

```
ypcat passwd | mksmbpasswd.sh > \
        /usr/local/samba/private/smbpasswd
```

After using the mksmbpasswd.sh script, edit the file by hand to remove root, bin and daemon just to be on the safe side.

Finally, to allow users to update their encrypted password, set the permissions on smbpasswd to be setuid root as follows:

```
chmod u+s /usr/local/samba/bin/smbpasswd
```

The last problem to note is keeping the smbpasswd file synchronized with the default UNIX authentication method. If users access the UNIX machine only through the Samba server, this won't be a problem. However, most systems also allow users to access shell accounts, pop servers or other services that will authenticate using the default UNIX password file. Many techniques can help keep these two files in sync. A hacked version of the **passwd** command is available that will update both files at the same time. Many people use **expect** scripts to update a user's password, entering both the passwd and the smbpasswd command after prompting the user for a new password. Reportedly, a PAM module to handle updating almost transparently is in the works and may be available by the time this article is printed. Asking on the Samba mailing list (samba@samba.anu.edu.au) for solutions other people have cooked up to alleviate this problem can be a big timesaver.

Closing Thoughts

In spite of the lengthy explanation of problems associated with SMB-encrypted passwords, it is still a good idea to make use of them. Even a slightly half-baked encrypted-password algorithm is superior to transmitting plaintext passwords across a network. Keeping the smbpasswd file secure and making sure users don't choose easily-guessed passwords will help minimize the risks.

John Blair currently works as a software engineer at Cobalt Microserver. When he's not hacking Cobalt's cute blue Qube, he's hanging out with his wife Rachel and newborn son Ethan. John is also the author of Samba: Integrating UNIX and Windows, published by SSC. Feel free to contact him at jdblair@cobaltmicro.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

X-ISP and Maintaining Multiple Account Records

Chris LeDantec

Issue #56, December 1998

Even for the experienced administrator, X-ISP provides an easy way to manage multiple accounts, keep track of usage expense and time on-line.

Connecting your Linux box to a PPP server can be a frustrating task, especially if you are not familiar with the scripting requirements, authentication modes or network settings your ISP expects you to use. X-ISP, a program written by Dimitrios P. Bouras (dbouras@hol.gr), takes what could be a complex task and simplifies it with a clean, effective GUI front end.

What makes X-ISP such a great piece of software? First, it is a user-friendly X-based front end to PPP/CHAT. Second, X-ISP provides a dialer, **xispdial**, and a stripped-down terminal, **xispterm**, which help work out some of the difficulty in establishing a connection. By using PPP/CHAT, X-ISP leaves most of the work to the operating system, while also giving experienced users the freedom to hack familiar connection scripts for any special needs.

X-ISP can be found at http://users.hol.gr/~dbouras/. To run X-ISP, you need to have X11R6, Xforms-0.88 and ppp-2.2.0f, and your modem must be set to verbose mode so that X-ISP can pick up connection status correctly. The home page provides comprehensive information and screen shots of the configuration and main screens. Once downloaded, the installation README gives clear instructions on how to compile X-ISP on a variety of systems.

How Does it Work?

X-ISP adds two helper modules called by **pppd** (see Figure 1). The first, xispdial, takes care of the dialing. **xispdial** sits between pppd and **chat** and is used by X-ISP to start the connection. **xispdial** reads an environment file residing in the user's directory. After getting the information for the ISP selected, it calls chat to make the connection. Once connected, control of the modem is turned over to the second module, xispterm. This terminal program allows the user to log in

to the system manually, or in the case that ip-up/ip-down scripts exist, xispterm runs those scripts. X-ISP also runs user-specific scripts, .xisp-up/.xisp-down, which reside in the user's directory and provide increased versatility.

Figure 1. X-ISP Functional Layout

A named pipe sends information from pppd, chat, xispdial and xispterm to the terminal window in X-ISP. The pipe gives immediate feedback as to system and connection status. The terminal window provides a clean view into the connection sequence and invaluable information for tracking down any connection errors.

Setup and Connecting

The user interface provides all actions through five buttons, three pull-down menus and a drop list for selecting which account to activate. The first step is setting up an account through the **Options** menu.

First, open the “Account Information” item. Enter the name of the account to add, then enter the phone number, user name and password. You will also need to know which authentication protocol your ISP uses. (See Figure 2.)

Figure 2. Account Information Setup

The next item, “Dialing and Login”, has you set the login and connect environment. Set the number of retries, connect notification, ISP callback and login setting. For manual login, X-ISP will start xispterm, or if you know the sequence, you can put together an automated login script and send user name, password and any initialization commands to the remote host. (See Figure 3.) X-ISP will use the connection script for automatic login or ISP call-back. For PAP or PAP/CHAP-Secrets authentication, the script will be ignored.

Figure 3. Dialing and Login Preferences

“Communication Options” allows you to control settings for the modem, baud rate, flow control, initialization and reset strings.

Figure 4. TCP/IP Options

The final item, “TCP/IP Options”, sets up Network Addressing. The most notable point here is the “Support for ip-up/ip-down” scripts. If set to “yes”, the DNS addressing can be set for each account, providing great flexibility when maintaining multiple accounts by dynamically updating the /etc/resolv.conf file. (See Figure 4.) When using ip-up/ip-down, X-ISP sends the **ipparam** option to pppd, which passes a string to ip-up/ip-down containing the pipe name for X-

ISP, the description set in Account Information, and if set, the DNS addresses specified. X-ISP will also display any information from ip-up/ip-down on the terminal window via the named pipe.

The next menu choice is **Logging**. If you are located in Europe, you will get the most advantage of this feature as the telephone companies (TelCo) listed are applicable only there. By selecting the appropriate TelCo and zone, X-ISP will keep track of on-line cost.

The **Statistics** item displays usage data in a window with a text summary and a bar graph.

The last item is the **Help** menu. Any issues you come across will probably be addressed in this very thorough help file. The "about" screen displays the version number and how to get in touch with the author.

Once your account is set up, you are ready to connect. First, check the following to be sure the rest of the system is in order:

- Set pppd for all users (i.e., other than just root):
  ```
  chmod u+s /usr/sbin/pppd
  ```
- Check the permissions on all X-ISP modules (xispdial, xispterm ...) so that group root can execute them (this should be done by the install script):
  ```
  chmod g+x
  ```
- Add any users who will use X-ISP to the dialout or root group for Debian or Red Hat releases, respectively.

Now, to connect, click on the "Connect" button in the main window of X-ISP. You should see the connection feedback in the small terminal window to the right. (See Figure 5.) Once the connection is established, the status windows will report the assigned IP address, modem connect status, connection speed and time since connected. The status is updated every 15 seconds, so if your connection is dropped, a maximum latency of 15 seconds will pass before X-ISP recognizes the dead connection and allows you to reconnect.

Figure 5. Main Window, Establishing a Connection

Hopefully, everything now works correctly. If there is a problem, the terminal window in X-ISP should give you an indication of where to look.

Security Issues

As with any program that allows users to connect or disconnect the system to or from a network, security concerns must be addressed. The areas of most

concern to me are the entering of account passwords at setup time (not dial-up time) and the transmission of authentication data.

X-ISP saves all account information in the .xisprc file in the user's home directory, including user name and passwords for the accounts. The rc file is readable only by the owner, so as long as there has not been a breach of the user's security, there should be no problem. As a secondary line of defense, X-ISP encrypts the password in the rc file using encrypt(3). The key used to encrypt the rc file is scrambled to remove any visibility in the binary. Since the encryption key resides in the source code, the possibility exists that someone could come up with the key and decode a user's rc files. Therefore, it would be best to change the encryption key in the source code before compiling X-ISP. The documentation outlines the procedure for changing the encryption key.

For PAP authentication, X-ISP calls pppd with the **+ua** option. PPPD version 2.3 no longer supports the **+ua** option, so if you are using that version of pppd, the PAP authentication mode will not be available. X-ISP creates a temporary file with login details in the user's home directory before calling pppd during a connect request, then removes the file as soon as the connection is established. This prevents any plaintext files with login details from sticking around. For PAP/CHAP-Secrets login, the appropriate files must be edited aside from X-ISP.

A potential liability may occur since X-ISP requires the user to be a member of the root group. Two remedies exist: either create a new group for X-ISP and add appropriate permissions to the program and data files, or use **sudo**. Creating a new group and adding users and files to it is probably the most straightforward way to tighten security on X-ISP. However, by allowing users access through sudo, the system administrator can allow the use of X-ISP without creating a new group or adding users to the root group and still maintain security integrity.

What's to come?

This should give you a good start with X-ISP, as well as a little insight into how it works. In the next release of X-ISP, a PTT editor will enable users to add entries to the TelCo database. The PTT information editor envisioned for X-ISP version 2.4 enables editing of all tariff rules for PTTs known to X-ISP, and also adding your own PTT information through a versatile GUI interface. The fields of the editor pop-up window shown in Figure 6 are the result of analyzing the PTTs currently known to X-ISP (version 2.3p7) plus a handful more which haven't yet made it into the distribution.

Figure 6. Information Form (under development, courtesy of Dimitrios P. Bouras).

Chris LeDantec is a Computer Engineering undergraduate at the University of Arizona. Aside from Linux, his passions include skiing, the Grateful Dead, philosophy of mind and National Parks. He may be reached at ledantec@engr.arizona.edu.

Archive Index Issue Table of Contents

Advanced search

Linux in Banking

Idan Shoham

Issue #56, December 1998

Mr. Shoham tells us how his company set up an Internet banking system using Linux for a bank in Western Canada.

M-Tech is a computer security products and services company based in Calgary, Alberta, Canada. This article describes our experience developing an Internet banking system for a major financial institution in Western Canada. Since it discusses the computer security infrastructure of this organization, they have asked that we not name them explicitly here. The system is now in production, and allows thousands of users to make financial transactions on-line.

This article will describe:

- What components are needed to build an Internet banking system
- How Linux made developing the Internet banking system easier
- The deployment of Linux servers as key components of the final system

What is Internet Banking?

Internet banking may be defined as any system that gives customers of a financial institution the ability to execute financial transactions across the Internet. Since the connection between the customer and the financial institution is electronic, we are limited to transactions that do not require the exchange of money or documents. What remains are the following:

- Funds transfers
- Account balance and history inquiries
- Bill payments
- Loan applications
- Retrieving information about services, branch locations, etc.

- Sending feedback to the financial institution

While a given Internet banking system might not support every one of these features, it is possible to implement any of them. Internet banking is basically a user-friendly, secure and distributed user interface to existing banking systems. With this in mind, we were hired to do the following:

- Design and deploy a network and application infrastructure to support the new application.
- Help implement a WWW-based user interface in Java.
- Help integrate the new system with existing business logic in our customer's mainframe.

Functionality

As outlined earlier, the Internet banking system must be able to support any transaction type where an exchange of physical items (such as cash) is not required. Since the set of possible transactions will evolve, it should be easy to extend the system to support new transaction types.

User Interface

The application is intended for use by thousands of users, many of whom have limited experience with computers. Accordingly, the user interface should be suitable for people whose computer skills consist only of having access to a computer, knowing how to start a WWW browser and how to type in a URL.

Additional requirements for the Internet banking system we were hired to construct are:

- Different groups in our customer's organization must be able to easily implement different user interfaces to the application with different graphics, advertising and menus.
- Our customer must be able to customize and extend the system's functionality in the future.

Security

By far, the most important feature of an Internet banking system is that it should do no harm. In particular, the system must ensure that:

- The existing “backend” (a mainframe used to process transactions) should be invulnerable to attack. Most importantly, it must not be vulnerable to denial-of-service attacks.

- A third party, connected to the network somewhere between the user on the Internet and the Internet banking system on our client's network, should find it impossible to decipher or alter the communication between those points.
- Users should be authenticated using as reliable a mechanism as economically feasible.

As a general rule, the system should be as safe, both for the user and the financial institution, as transactions made by the user in a bank branch.

Hardware

When deployed, the system consists of four physical components:

1. Client workstations, which include a WWW browser with Java and SSL capabilities
2. One or more firewall systems to protect the Internet banking servers against external assault
3. One or more application gateways: the Java user interface applet is downloaded from these and must communicate with the backend through them.
4. A backend transaction processing system: for most financial institutions, including our customer, an IBM mainframe is used.

Networks

There are three conceptual network segments between these hardware components:

1. The Internet between the client and an external firewall
2. Network-1 between the external firewall and the application WWW server
3. Network-2 between the application WWW server and the mainframe

This arrangement is illustrated in Figure 1.

Figure 1. Linux in the Loop

A Robust Development Environment

In developing our customer's Internet banking system, we first deployed a Linux-based development environment consisting of a Compaq Prosignia server, 64MB of RAM and a pair of 4.3GB Ultra Wide SCSI disks. We installed a Debian Linux distribution on this system. On the platform we installed a wide range of tools, shown in Table 1.

We used this setup as our primary development environment. Using Apache-SSL, we were able to test various browsers to check the application's behaviour and to adjust its appearance.

Using some customized Makefiles and libraries, we were able to write Java client code, C-language CGI and daemon programs, C-language mainframe programs and COBOL program stubs all on this environment.

By implementing an RPC system, where a code generator translates a master transaction-description file into source code for each target platform, we were able to add new transaction types to the Internet banking application with just a few short commands.

Network Security

To secure the final system against denial-of-service, eavesdropping and impersonation attacks, we used Linux to implement an external firewall. Since our access control rules are simple and static, we were able to use the Linux kernel's built-in packet filtering features to limit remote access to just the minimum required services—DNS and HTTPS.

As a further precaution against unauthorized access, no system on the Internet can connect to our customer's mainframe without first passing through our application code. There is simply no path for network packets from the Internet to the mainframe. Our application system cannot connect to any computer other than the mainframe; this minimizes the exposure of other systems on our customer's network.

To ensure private communication, we use the SSL protocol, embedded in HTTPS, to protect the communication between the application WWW server and the client machine. We also do not entirely trust the physical connection between the application WWW server and the mainframe, so all communication between our application on the WWW server and our transaction management software on the mainframe is encrypted.

Finally, since users are likely to log in and walk away from their terminals, we implemented a token management system between the client Java applets and the mainframe, where tokens have an implicit timeout of a few minutes. Unattended sessions look exactly like attended ones on the client's machine, but are blocked from making new transactions.

Redundancy and Availability

An important concern in a live system of this magnitude is the possibility of down time, caused by power outages, hardware failures or high load. Our

system is protected against power outages by uninterruptible power supplies (UPSs). In addition, we implemented two identical application servers, each of which serves as both a DNS and an application server. One system is configured as the primary DNS, and the other is the secondary DNS. If the primary should fail, DNS queries will cause clients to connect to the secondary system instead.

Finally, all connection-state information is stored in a database on the mainframe. Since the application servers are stateless, consecutive transactions can be routed through an arbitrarily large number of application servers, all operating in parallel. And since the application servers are PCs, we can scale the system up to handle an arbitrarily large amount of traffic simply by buying more PC servers, along with an intelligent router.

An Open-minded Customer

To take full benefit of Linux's technical advantages, however, our customer had to be open-minded. We have been fortunate to work with this customer, since they judged the merits of this architecture on the grounds of its reliability, features, extensibility, maintainability and cost. We are convinced that the implementation team's open-minded approach to technological alternatives allowed them to implement the best possible solution, rather than just the same technology as their competitors.

A Mature Development Environment

We have used Linux as a development platform before and chose it because of the rich set of tools included with most distributions. This project held no surprises—Linux was a convenient, productive and reliable development platform. We never had problems with any of the development tools, and never experienced system downtime.

In the past, we have developed software for Windows NT as well as for other UNIX platforms. Linux compares very favorably with these as a development and deployment platform—it is simply more full-featured and better supported.

A Robust Production Platform

Linux is proving its worth as a production environment as well. It runs on inexpensive hardware and, along with Apache SSL, offers excellent WWW server performance. We would be hard-pressed to find comparable systems on which to run our WWW servers, development environment and firewall without spending much more money and settling for a less comprehensive tool set.

In this project, the rich set of network features found in Linux proved especially useful. In particular, the following:

- Setting up the firewall was simple, and the resulting system is quite effective.
- Our development server uses **ssh** for secure remote connections, the X Window System for convenient access to source files and tools, and Samba to allow developers to access files directly from their PCs.
- BIND made it easy to implement a failover from the primary to a backup server.
- Various shell tools make it easy to keep the software on the backup server current.

Linux is not only feature rich, but also well-supported. We have found that whenever new security exposures are discovered, Linux is invariably the first system for which patches or workarounds are available. For instance, the ping-of-death vulnerability was reportedly fixed in three hours, and a Linux patch for a common buffer-overrun vulnerability was released alongside the discovery of the bug itself. We doubt that any vendor could match the response time of the worldwide community of Linux programmers.

An Open Future

In this project, we built just one application—a user interface with which our customer's clients can make financial transactions. However, there is nothing about our technology that is specific to the Internet or even to banking.

Within our customer's organization, the same technology could also be used to enable thin clients to function as teller workstations, process loan applications and support communication with automated teller machines and telephone voice response systems. Beyond the financial sector, this technology could be used for any transaction processing system with a broad or geographically distributed user community. Examples include travel booking systems, libraries, government registries and more.

References

Postscript

Idan Shoham is one of the principals of M-Tech. To learn more about M-Tech, please visit the company's site at http://www.m-tech.ab.ca. Mr. Shoham can be reached via e-mail at idan@m-tech.ab.ca.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Preventing Spams and Relays

John Wong

Issue #56, December 1998

The smtpd package is a useful mail daemon for stopping spam, thereby saving money and resources.

The Internet has been around since the 1970s with people using it mainly for electronic mail (e-mail). This is still true even today as we see increasing numbers of multimedia-based applications on the Internet. People still use e-mail more than web browsers and file transfer programs. The reason is simple—it is a low-cost yet effective method for communicating with others all over the world. More and more people are getting onto the Internet, and the first thing they do is get an e-mail address.

Realizing this, certain individuals and even organizations have taken the opportunity to send unsolicited e-mail to Internet users. Most of this mail is commercial advertising. They send thousands of copies of the same message to a list of e-mail addresses obtained from either Usenet postings or companies' home pages. Such action is called spamming.

The consequence of spam is that the receiver ends up paying for the cost of the e-mail, since the ISP (Internet Service Provider) usually charges on the basis of connect time and downloading mail takes time. At companies where each employee has an e-mail address and the company has a fixed line for e-mail, that line can become quite expensive if it is targeted by spammers.

To hide their identity, spammers usually send from a forged e-mail address and use mail servers that have not been configured to prevent such activities. Relaying, as it is usually called, will cause the targeted e-mail server to send mail on behalf of the spammer to hundreds or even thousands of users. Spamming can seriously affect the performance of the mail server and cause massive bandwidth loss for the company affected. Not only will the mail server's performance be affected, but recipients of the spam will be given the impression that the company is in business with the spammer.

For more information on this, visit the number one anti-spam site on the Internet at http://spam.abuse.net/.

The smtpd Package

My company was recently the target of a spammer who used our mail server as a relay. The spammer was using an account from a well-known service provider, and my guess is that it was probably a trial account. I had not installed any preventions against such attacks and was thus targeted. When I discovered the attack, I dropped all my current work and began searching for a solution. I started by looking at the Sendmail home page (http://www.sendmail.org/). It had links that led to more tools and tips for spam prevention. I chose to implement the solution based on the **smtpd** package from Obtuse Systems Corporation for the following reasons:

1. **smptd** is a small package with only two executable files to manage and one configuration file to tweak.
2. It works nicely with my mail server (Sendmail v8.8.8).
3. The configuration file is very flexible and easy to configure.

As of this writing, the smtpd package is at version 2.0 and can be obtained from ftp://ftp.obtuse.com/pub/smtpd/smtpd-2.0.tar.gz.

smtpd works with other MTAs (Mail Transport Agent), but I have been using only Sendmail. The latest Sendmail can be retrieved from ftp://ftp.sendmail.org/ucb/src/sendmail/sendmail.8.9.1.tar.gz.

Note that you must have a working MTA before installing smtpd. It acts only as a mail proxy, storing and forwarding mail to other MTAs for the actual delivery.

Briefly, here's how smtpd works: the smtpd daemon runs and accepts mail instead of your regular mail server. It accepts mail from the Internet as well as your own domain. **smtpd** can be configured to reject mail based on several criteria:

1. IP address of the sender
2. Host name or domain of sender
3. E-mail address of sender
4. E-mail address of receiver

Based on the configuration file, the mail is either rejected or accepted and spooled. The second program, **smtpfwdd**, will do the actual forwarding of the spooled mail to the MTA (Sendmail in this case).

Compiling and Installing the Package

Once you've obtained the package, unarchive the files to a directory. Assuming the files are to be put in the directory /usr/src/smtpd-2.0, do the following:

```
cd /usr/src
tar xvzf ~/smtpd-2.0.tar.gz
cd smtpd-2.0
```

Now, by typing **ls**, you'll see many files and subdirectories. Be sure to read README and INSTALL as these files contain valuable information on the installation of the mail proxy.

To compile the package, do the following:

1. Choose a user and group for running the smtpd daemon. Your choice must be defined as a trusted user in the sendmail.cf file. I chose to use the user **daemon**. If you are not sure what to use as trusted user, check the /etc/sendmail.cf file for lines like this:

   ```
   # Trusted users #
   Troot
   Tdaemon
   Tuucp
   ```

   In this example, the trusted users are **root**, **daemon** and **uucp**. Do *not* use **root**. **smtpd** works without any root privileges; thus, it is more secure to run it as some other user.
2. Create a directory in which smtpd can store spooled mail before smtpfwdd processes it. Change permissions and also the user/group of that directory so only that user has full rights to it. If you put it in the /home/smtpd/spool directory, execute these commands:

   ```
   mkdir /home/smtpd
   mkdir /home/smtpd/spool
   chown -R daemon.daemon /home/smtpd
   chmod 700 /home/smtpd
   ls -lad /home/smtpd
   ```

   The output from **ls** will look like this:

   ```
   drwx------ 3 daemon daemon 1024 Mar 26 01:34
    /home/smtpd/
   ```
3. Edit the Makefile in the source directory to reflect your choice. The changes we need to make for our example are as follows:

   ```
   SMTP_USER = daemon
   SMTP_GROUP = daemon
   SPOOLDIR = /home/smtpd
   SPOOLSUBDIR = /spool
   EHLO_KLUDGE=1
   JUNIPER_SUPPORT=0
   #LD_LIBS=-lresolv
   CHECK_IDENT = 0
   ```

EHLO_KLUDGE is needed to fix a bug in Netscape Communicator. **JUNIPER_SUPPORT** is set to 0 unless you're using Obtuse's Firewall Kernel. **LD_LIBS** is commented out, as my Linux distribution does not require the external library libresolv. **CHECK_IDENT** has been set to 0 to disable IDENT checking. I personally do not believe in IDENT checks—they take time and do not return any useful information.

By default, the Makefile has been configured to be compiled on Linux, so no other changes need to be made.

4. In the source directory, type **make** to compile smtpd and smtpfwdd.
5. Once the compilation is finished, you will find two executable files in the directory. Copy them to another location in your PATH. To copy them to the /usr/local/sbin directory, type:

```
cp smtpd /usr/local/sbin
cp smtpfwdd /usr/local/sbin
```

6. Create a few subdirectories under the /home/smtpd/ directory by typing:

```
cd /home/smtpd
mkdir etc usr
mkdir usr/lib
mkdir usr/lib/zoneinfo
```

Because smtpd does a **chroot** to the directory /home/smtpd, we need to copy (or make symbolic links) into this directory some files that are required for the proper execution of smtpd/smtpfwdd. The files and the directory in which each should be located are:

- /etc/resolv.conf **->** /home/smtpd/etc/resolv.conf
- /usr/lib/zoneinfo/localtime **->** /home/smtpd/usr/lib/zoneinfo/localtime

The file resolv.conf is needed so that smtpd can do DNS queries (look up IP addresses of hosts). The file localtime has your time zone setting and is required to put the proper time stamp on e-mail. The location of localtime may be different on your system, so you'll have to find the exact path and create a duplicate under the /home/smtpd directory.

7. Configure smtpd and smtpfwdd to replace the running mail server.

Configuring smtpd

The mail proxy reads its configuration from a file (smtpd_check_rules) in the /etc directory, in our example, /home/smtpd/etc/smtpd_check_rules. Each line in the file beginning with a # is a comment. Blank lines are allowed. Rules have the following format (one line):

```
[allow|deny|noto]:SourceList:FromList:ToList[:XXX message]
```

where XXX is the error message number. The first rule that matches will be taken and the check ended, so placement of rules should be done carefully.

The first field states the action to either allow an SMTP connection, deny the SMTP connection and close the session or **noto** which will return an error for the matching rule but will still continue for the session.

The second field is a list of IP addresses and/or host names to match the source SMTP connection. IP addresses may be specified with a netmask to include a whole network. The format of this is *XX.XX.XX.XX/bits* where bits is the netmask bits for the network. For instance, a network 192.168.0.0 with netmask 255.255.255.0 would be written as 192.168.0.0/24. A few special reserved identifiers that can be used are:

- **ALL**: any IP address and host name
- **KNOWN**: only IP and host names which are DNS resolvable
- **UNKNOWN**: IP and host names which are not DNS resolvable
- **EXCEPT**: exceptions
- *****: wild-card character

The third and fourth fields are used to match e-mail addresses and have the format user@host. The special word **ALL** can also be used in these fields.

The fifth field is optional and is used to return error messages from **deny** and **noto** to the SMTP client. The following special variables can be used to return information in the error messages:

- **%F**: mail from address
- **%T**: recipient address
- **%H**: connecting host name
- **%I**: connecting IP address
- **%U**: user from the host

All three fields (**SourceList**, **FromList** and **ToList**) must be matched in order for action to be taken.

Listing 1 is an example of a set of rules that assumes the internal network is 10.0.0.0 and a mail hub is at 10.0.0.9. Note that **noto_delay** will pause for a certain amount of seconds before action is taken. This option was introduced to delay relayers and spammers and the parameters that control this timeout are set in the Makefile:

```
NOTO_DELAY = 60
DENY_DELAY = 60
```

A few other configurations can be done that I have not shown here, namely the **NS=** *pattern-check* and the use of the IDENT protocol for identifying users.

Users who need a more detailed setup of this file should read the file smtpd_address_check.txt in the source directory. Examples for filtering spams and relays can be downloaded from Obtuse's FTP site.

Running smtpd

After creating the configuration file, the running mail daemon must be stopped and replaced with smtpd/smtpfwdd. For Sendmail, this can be done by typing:

```
> ps ax | grep sendmail
24569 ? S 0:00 sendmail: accepting connections on port 25
> kill 24569
```

This will effectively shutdown the mail daemon. Now, check for queued mail that the daemon has not yet sent out by issuing the command:

```
/usr/lib/sendmail -bp
```

If the mail queue is not empty, flush the queue by typing:

```
/usr/lib/sendmail -q
```

If mail is still in the queue after awhile, this command can be resent at a later time so the installation of smtpd/smtpfwdd can continue. No new mail will be accepted while the mail daemon is down.

Start the smtpd daemon by issuing the command:

```
/usr/local/sbin/smtpd -c /home/smtpd -d /spool\
-u daemon -g daemon -D
-L
```

The smtpd daemon will start accepting mail and spool it to the /home/smtpd/spool directory. The parameters on the command line are defined as follows:

- **-c /home/smtpd**: the smtpd home directory
- **-d /spool**: the directory where spooled mail should be stored
- **-u daemon -g daemon**: user/group smtpd
- **-D**: instruction to run as daemon and listen on the SMTP port
- **-L**: instruction to suppress children in daemon mode from making an **openlog** call

Once smtpd is running, check the directory—it will be full of files with the prefix **smtp**. These files are the spooled mail messages and need to be processed by the MTA. This is the job of smtpfwdd. We run smtpfwdd by typing:

```
/usr/local/sbin/smtpfwdd -d /home/smtpd/spool -u\
daemon -g daemon
```

Once it begins running, smtpfwdd will check the spool directory /home/smtpd/spool and starts processing the spooled mail by running the MTA, in this case Sendmail.

A good idea is to run the MTA in such a way that it periodically processes its mail queue and sends out any mail present. Note that we actually have two spool directories here: one used by smtpd and the other by sendmail (usually in /var/spool/mqueue). To run sendmail in non-daemon mode in order to process the queue every 15 minutes, type:

```
/usr/lib/sendmail -q15m
```

Once everything is running fine, edit your startup files to run smtpd/smtpfwdd by default instead of sendmail.

Summary

The Internet is no longer the “friendly global village” we once thought it was. Living among the netizens are unscrupulous individuals and even companies that take advantage of the openness of the Internet for their own benefit while making others bear the cost. We must take preventive measures against these attacks if we want to avoid becoming victims. Proper policy on the e-mail server will help to ensure this. With smtpd in place, you will have more control of your e-mail server.

John Wong can be reached at johnw@extol.com.my.

Archive Index Issue Table of Contents

Advanced search

Mathematica v3.0

Patrick Galbraith

Issue #56, December 1998

Instead of having various packages or tools for a variety of mathematical functions, Wolfram has integrated them all into one package.

- Manufacturer: Wolfram Research
- E-mail: info@wolfram.com
- URL: http://www.wolfram.com/
- Price: $1,495.00 US
- Platform: Caldera OpenLinux 1.2, i486/100, kernel 2.0.33, ATI Mach32 video
- Reviewer: Patrick Galbraith

About three years ago, I was taking various math courses—different levels of calculus and differential equations. I actually enjoyed both, finding them a constant challenge to master. At that time, I was also working on mastering another exciting puzzle—Linux. Back then, there weren't a lot of commercial applications or even books on Linux. I had just spent a whole night downloading and copying to floppies, and migrated from Sunsite to ordering the InfoMagic set and printing out all the FAQs and HOWTOs I possibly could. I also had to maintain a MS Windows/DOS partition to run my math software, which was then Maple V. I wasn't happy to have to reboot to use the software, but when projects requiring Maple were due, I had to do it.

Today, many commercial products are available for Linux. One of them is Mathematica, a full-featured, powerful, math lover's paradise. I would have loved to have had it three years ago.

Recently, I had the privilege of reviewing this product and will look at as many features as possible in such a brief forum.

What is Mathematica?

Wolfram states, “Mathematica is the world's only fully integrated environment for technical computing.” Instead of having various packages or tools for a variety of mathematical functions, Wolfram has integrated them all into one package. For this review, I tested the mathematical capabilities and ease of use of this integrated package, using the many math examples given in the huge book by Stephen Wolfram that is included with the package.

There is no limit to the things one can do with Mathematica; each would take an entire book in itself to review. I will stick to simple uses here, and keep the beginner in mind.

Package Contents

Mathematica comes in a big box that includes the following:

- *The Mathematica Book* by Stephen Wolfram, which is 1403 pages of excellent examples, applications and colorful illustrations of Mathematica in action.
- Getting Started Guide
- Standard Add-On Packages Book
- Mathematica System Administrators Guide
- License Certificate
- Installation media on CD

Installation

The installation of Mathematica was quite smooth. The installer is a simple shell script that asks various questions, such as directory and program password. The password can be obtained at Wolfram's web site upon product registration and is required in order to have a fully functioning program. Other options for registering are via fax or mail. The registration links the copy of Mathematica to the host name on which you are installing. Once the installation is performed and the password obtained and included in the installation step that asks for it, you can begin using Mathematica. It is possible to install the password at a later time, but you will be able to run in MathReader Mode only.

Usage

There are two ways to run Mathematica: via X, using the notebook and palette window (see Figure 1) or via the command line. The benefits of using the

notebook and palette are the point-and-click interaction and the ability to save each session as a worksheet.

Figure 1. Palette

The benefit of the command line is quick computations. Note that graphics are a separate process. If you are running from the command line in an xterm, any graphics you generate will go to Mathematica's graphics output window. If you are running from a virtual terminal, the graphics will be plain ASCII.

The language to interact with Mathematica is quite simple and intuitive and also well-documented. If you have any rudimentary programming experience, it will be even easier to use. When using the worksheet window, it checks syntax and gives a system bell if you use the wrong type of bracket.

The first two examples of this are plotting (sin 1/x2)(e-x) (Figure 2) and power series (Figure 3):

Figure 2. (sin 1/x2)(e-x)

Figure 3. Power Series

I was running the equations in Figures 2 and 3 on a 486/100, and all of the computations ran quite fast.

More Advanced Computations

One of the major parts of one math course I took was power series. One such equation done with Mathematica is shown in Figures 4 through 6.

Figure 4. Differential Equations

Figure 5. Integration

Figure 6. Graphical

This is just the tip of the iceberg in regard to the many mathematical equations you can perform with Mathematica. You can also program Button Boxes to perform specific actions when you click on them. This option enables you to create interactive worksheets (see Figure 7). Other capabilities include sound, animation and transforming input files (such as an image file) or external sounds. Mathematica's language can be used to read in files, output to files, read directory contents (explicitly and type-globbing), change to a different working directory and delete files, all of which I found to be quite useful. Any programmer will appreciate the Mathematica feature that converts a

Mathematica expression to a C or FORTRAN expression—an extremely useful feature (Figure 8).

Figure 7. Worksheet

Figure 8. Conversion to C and FORTRAN

External Functions

Mathematica has an external function which converts Mathematica worksheets to HTML or TeX.

I tested the HTML conversion, and it basically produced an HTML document with links to images containing all of the worksheet contents, both text and graphics (see Listing 1). Output from Mathematica can be sent to an external file, or one file can be combined (spliced) with another. External commands can be launched from Mathematica, e.g., to start Netscape or any one of the various word processing packages.

Conclusion

Mathematica is an excellent tool with limitless features. I was impressed at how smoothly it ran with both Netscape and StarOffice running at the same time. I truly wish this had been available when I was in school, not only because it is available for Linux, but because of all the useful features and the fun I have using it. I like the flexible language Mathematica uses, and the documentation is excellent. I found *The Mathematica Book* to be full of great examples and explanations on usage. Best of all, it runs on Linux and takes full advantage of Linux's speed, both in computations and graphics rendering.

Users of this product could be engineers, scientists, chemists, teachers, students and general math enthusiasts. I highly recommend Mathematica to anyone who is involved in the sciences. This is one tool you will always find invaluable, and it is well worth the price.

Patrick Galbraith currently works as a senior software developer for the Cobalt Group in Seattle, WA (http://www.cobaltgroup.com/), developing automotive web sites using Linux/Perl/Oracle. He also consults with Horvitz Newspapers, his previous employer, publisher of local area newspapers including Eastside Journal, *South County Journal* and Tennessee-based *The Daily Times* (http://

www.eastsidejournal.com/, http://www.southcountyjournal.com/, http://www.thedailytimes.com/). The rest of his time is spent doing yard work, cooking, hiking, tweaking his Linux system, and reading slashdot.org.

Archive Index Issue Table of Contents

Advanced search

Happy Hacking Keyboard

Jeremy Dinsel

Issue #56, December 1998

According to PFU America, the keyboard's design makes it easier for programmers to reach the keys they want quickly and efficiently.

- Manufacturer: PFU America Inc.
- E-mail: hhkb-support@pfuca.com
- URL: http://www.pfuca.com/
- Price: $229 US for keyboard including 3 cables, $189 with one cable option
- Reviewer: Jeremy Dinsel

The Happy Hacking Keyboard is a cute and fuzzy streamlined keyboard designed specifically with programmers in mind. While not a single bit of fuzz is actually on the keyboard, its size makes it cute, if not disorienting, to people used to the standard IBM PC keyboard.

According to PFU America, the keyboard's design makes it easier for programmers to reach the keys they want quickly and efficiently. They claim having fewer keys on the keyboard increases efficiency by preventing users from overextending their fingers on certain keystrokes.

Installation

The Happy Hacking Keyboard arrived in a tiny box shortly after I agreed to do a review of the product. Inside were the keyboard and three cables (for a PS/2, Macintosh and Sun computer) along with the usual manual and warranty information.

PFU America recently changed the package, and lowered the price. The Happy Hacking Keyboard now comes with only one cable (of the customer's choice), but additional cables are available for $35.00 each. The cables are expensive because they are handmade by the people at PFU America.

The manual was fairly straightforward—after all, almost everyone knows how to hook up a keyboard. However, with the many cables that accompanied the keyboard, it was comforting to know that documentation was available should it be needed.

After the computer was powered down, I said goodbye to my 101 Enhanced keyboard and hello to blissful days of Happy Hacking. Or so I thought—I had to grab a PS/2 to AT keyboard adapter first.

Life is a Series of Adjustments

The keyboard is streamlined, containing only 60 keys. A function key is included that can be used in combination with other keys; as a result, awkward finger positioning is sometimes required. My first days using the keyboard reminded me of playing Twister and trying to reach the red dot by squeezing my arm past two opponents while keeping my feet on the orange and blue dots on opposite sides of the mat. In fact, two weeks later, I was still finding myself reverting to my old PC keyboarding habits. Some complex key sequences were hard to complete correctly, as old habits die hard.

Also, in the beginning, the backspace key didn't work; however, this turned out to be primarily my fault. Being lazy and excited to test out the new keyboard, I refrained from reading all the way through the manual to the final (third) page where a table and accompanying figure would have taught me how to program the keyboard using a slider switch. Eventually, I toggled the switch and had the backspace key working to my satisfaction.

Since I started using Linux before Windows 95 was introduced (I stopped using MS products long before that), I did not miss the extra “Windows” keys found on most PC keyboards. I did, however, have to get used to console cruising with the new keyboard. Switching from X to the console requires a four finger/key combination (**ctrl-alt-fn-f***, where fn is the function key), while cruising through consoles requires a three finger/key combination (**alt-fn-*arrow-key***).

Even in a non-**vi**-type editor without command mode movement keys, the Happy Hacking Keyboard makes the user adjust to finding the location of the arrow pad and remembering to hit the function key. In all fairness, it took me less than a week to become oriented with the key locations. (It does remain comical to watch others try to wander through the key selections for the first time.)

Unlike a laptop, the size and shape of the keys are the same as on a PC keyboard, making it easier to adjust. I never overreach the true location of the keys and don't have a difficult time typing something on other people's computers (who don't have a Happy Hacking Keyboard). However, I am now known to complain about how “weird” other keyboards are.

Happy Hacking

While the keyboard did not cure me of my sarcastic nature, I did find the escape key much easier to reach since it's located to the immediate left of the “1” key. In vi, I can quickly switch out of insert mode since I never have to look down to relocate the escape key or reposition my fingers afterwards; thus, cruising through vi has become even easier.

For XEmacs programming, the control key is located in the “right” place, directly left of the “A” key. This makes it easy to use without any odd movements or taking your fingers away from the home row. (Yes, I learned to type before I learned to program.)

Both of these key locations, escape and control, have allowed me to quickly negotiate commands without having to reposition my fingers. This has the benefit of reducing the frustration of trying to return to the home keys after each command—my fingers never wind up in odd locations as they did on a typical PC keyboard.

Disgruntled Gamer

As a part-time game player (Linux Quake), I'm accustomed to using the keyboard for all player movements, such as turns and running. With this keyboard, I'd have to hold the function key down constantly (to select the arrow keys) or figure out how to use the mouse. Otherwise, keeping the function key depressed (two keys away from the arrow keys) and trying to fumble around with the arrows might increase the probability of developing carpal tunnel syndrome.

After a few games of Quake, I think I'll be comfortable with the bizarre fingering required. Also, using the keyboard to program in XEmacs helped in the adjustment needed to get into the gaming world.

Technical Support and On-line Documentation

Documentation is also available on-line. While I haven't had to use their tech support e-mail, it is readily available—my contact at PFU America was quick to reply to any e-mail I sent. Furthermore, all of the information needed to install and hook up the keyboard can be found on-line. All of the information in the manual is included in their on-line documentation.

In Closing

Overall, I would be hard-pressed to sum up this review with anything but a positive remark. With the price tag recently dropping by $40, the keyboard is more affordable. I'm sure other hackers will be quite happy to own it.

For someone who hasn't experienced the keyboard, it's hard to believe everything reported about the Happy Hacking Keyboard by PFU America. In fact, I was skeptical about the remarks I had heard before I became a Happy Hacking Keyboard user. Now, one month after laying my fingers on it, I can't imagine using any other keyboard. I wonder if PFU America makes a Happy Hacking tote bag.

Jeremy Dinsel is an almost-graduate of California University of Pennsylvania, where he studies computer science and operates the Math and Computer Science Linux server. He welcomes questions and comments and encourages western Pennsylvanians to join WPLUG—a Linux organization (http://sighsy.cup.edu/~dinselj/wplug/). He is also the webmaster for SSC and can be reached at info@linuxjournal.com.

Archive Index Issue Table of Contents

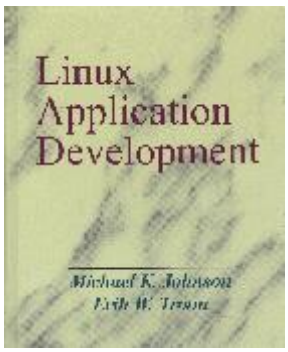
Advanced search

Linux Application Development

Andrew Johnson

Issue #56, December 1998

This book does not attempt to teach programming or C, but serves as a topical reference for experienced C programmers to become familiar with the Linux programming model.



- Author: Michael K. Johnson & Erik W. Troan
- Publisher: Addison Wesley
- Price: $45.95 US
- ISBN: 0-201-30821-5
- Reviewer: Andrew Johnson

Linux Application Development is a solid introduction to Linux programming. It does not attempt to teach programming or C, but serves as a topical reference for experienced C programmers to become familiar with the Linux programming model.

The book is divided into four major parts. Part One, “Getting Started”, contains three short chapters covering the history of Linux, licenses and copyright issues, and the availability and locations of Linux documentation, mailing lists and other books and sources of information.

Part Two provides an introduction to the Linux development environment and tools. Some of the coverage is minimal; for example, the section on the GNU

debugger, **gdb**, contains only a short list of the essential debugger commands and references to two other books which offer tutorials on the debugger. More extended coverage is given to memory debugging tools. This includes source examples and information about creating and using libraries. There is also brief but important coverage of **make**, the **gcc** compiler and its options, system calls and common error codes.

The twelve chapters of Part Three, "System Programming", comprise the bulk of the book. These chapters, as elsewhere, are heavily subdivided into subsections, which I found a little distracting on first reading, but quite convenient for relocating information later.

The authors give an excellent balance of breadth and depth of coverage, with chapters focusing on processes, simple and advanced file handling, directory operations, signals, job control, terminal handling, socket programming, dates and timing, random numbers and console programming. Virtually all of these topics are augmented with small source code examples.

A larger example program, **ladsh**, is a simplified UNIX command shell which is developed over the course of several chapters and eventually supports simple built-in commands, command execution, I/O redirection and job control. The final version of this program is 710 lines of code, and working through its development provides a good exercise in tying together some of the basic elements of Linux system programming.

Part Four describes a few important development libraries such as the S-Lang terminal library, the Berkeley database library and the **popt** option parsing library. This section also provides brief discussions of regular expressions, dynamic loading with **dl**, and the names and user databases.

Finally, three appendices cover direct I/O port access, the final source version of the ladsh program and the GNU licenses.

Overall, the book is well-organized and the writing and explanations are clear and concise. Although designed explicitly as a reference for experienced C programmers making the switch to Linux, I would recommend it as a good additional resource for anyone just beginning to learn C in a Linux environment.

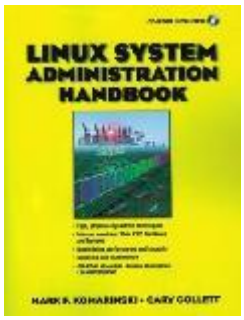
Andrew Johnson is currently a full-time student working on a Ph.D. in physical anthropology and a part-time programmer and technical writer. He resides in Winnipeg, Manitoba with his wife and two sons and enjoys a good, dark ale whenever he can. He can be reached by e-mail at ajohnson@gpu.srv.ualberta.ca.

Linux System Administration Handbook

David A. Bandel

Issue #56, December 1998

Managers who don't actually administer systems, but make decisions regarding whether Linux is “appropriate”, or intermediate users who can use the better-fleshed-out chapters to complement other manuals they have, could find the book a good value.



- Authors: Mark F. Komarinski and Cary Collett
- Publisher: Prentice Hall Computer Books
- E-mail: info@prenhall.com
- URL: http://www.prenhall.com/
- Price: $39.95 US
- ISBN: 0136805965
- Reviewer: David Bandel

The Linux System Administration Handbook was written by Mark F. Komarinski and Cary Collett. The authors claim to have nearly two decades of experience between them in system administration, most in Linux, though some in other UNIX systems. Reading the cover, I attempted to gain some insight into what lay in the pages ahead. The cover did promise much in the line of system administration tasks, security, hardware configuration, and much more. The book also included the gratuitous CD-ROM. (What Linux book doesn't these days?) The cover, however, gave no indication of the kind of audience it was aimed at, so I began reading, not exactly sure what to expect. The primary task I

took on myself, then, for this review was to determine what audience the authors were aiming for.

The first few chapters cover boot-up and shut-down and a review of System V start-up, then dive into user administration and user shells. These chapters do not discuss Linux installation per se, but I am very familiar with Caldera's OpenLinux Lite, the distribution included with the book, and it really doesn't require much in the way of installation instructions, at least not for anyone vaguely familiar with Linux installation. Besides, all of Caldera's pertinent documentation is included in the Appendices.

As I continued reading I was impressed with some of the jewels of wisdom and little-known (or not well-documented) facts about Linux that only seasoned administrators would know. Unfortunately, this was often offset by short, terse explanations, short chapters and one- or two-sentence summaries for those basic chapters toward the front of the book.

Going a little further and getting into the coverage of nitty-gritty network administration, the chapters grow longer. Like an administrator who's finally been given his favorite project to work on, the chapters suddenly take on more detail and more life. The authors go into great detail explaining some of the lesser-understood and lesser-used of the well-known services, the kinds of problems you can expect, and how to configure, troubleshoot and maintain them. In fact, most of the chapters where they went into this kind of detail are quite well-done. Much of the information presented showed the authors do indeed know a good deal about those programs and services to which they have had personal exposure.

Continuing on, they discuss some of the important issues for decision makers wanting to know about applications for Linux. Here, they took on the daunting task of trying to do justice to all the applications beginning to show up for Linux, from open-source software to commercial native Linux applications to those which can be adapted to Linux. They did a creditable job and warned the reader they would discuss only those programs with which they had some familiarity. However, I was still surprised they didn't do a bit more homework for the reader, including a few more applications which they don't use, but are available.

For example, in the section on databases, they mention that Oracle, while not supported on Linux, can use the SCO binaries with iBCS. While they don't mention it, the same is also true of Informix. Native Linux applications such as Adabas and YARD (both from Germany) were not mentioned. I find this even more curious since Adabas is sold by Caldera, and YARD rivals Informix in its ability to do nested outer joins and other complex SQL queries. YARD is also

ANSI SQL 92 and SQL3 compliant, something most open-source Linux databases can't begin to boast about. The authors talk about distributions later on and mention WGS, which positions the Flagship database as its premier Linux product, but this isn't mentioned in the database section.

While I was hoping that this would be a good book to help beginners discover Linux, mostly due to the exceptionally easy-to-install OpenLinux distribution, it is hardly that. Some omissions and skimpy coverage of basics would lead me to conclude that this is not a good book for a novice administrator. For example, in the rather short chapter on “Common Features”, where the authors discuss setting environment variables under the bash shell, they don't mention the **export** command, its usage or implications. This kind of oversight could have novices wondering why subshells or programs invoked by the shell haven't inherited a particular environment variable.

On the other hand, I also can't recommend this book to experienced system administrators on the strengths of its detail in the lesser of the well-known services, even though these sections are well done. While the book would make an excellent addition to a library lacking the details of network news (NNTP) or other services, it isn't justified because of the light or non-existent treatment given to other areas.

I can, however, recommend it to managers who may have noticed their system administrators have begun to use Linux on their network or are contemplating allowing this to occur. The book does a good job of introducing Linux in a way that would give managers a good feeling about this oft-called “Renegade OS” being put to work in their companies.

On a scale of one to five, I would have to rate this book a solid three. Its apparent lack of focus, its terse coverage of some important areas, and redundant coverage of some network issues among the Networking and the Internet Connectivity chapters make me conclude this first edition doesn't deserve a higher rating.

For those who think I may be a little harsh, my initial impression was much lower. While I may be tolerant of a few clichés or awkwardly worded phrases, the authors trounced solidly on a pet peeve of mine throughout the first few chapters by misusing the phrase “try and” when they really wanted you to “try to” do something.

However, I hope the authors will soon begin work on the second edition, because I can see the framework for a good Linux system administration handbook. Some other things I hope the authors will consider is either changing the distribution to Red Hat, which is what they talk about in many

examples, or changing their examples and discussions in the book to reflect the distribution. For those who don't know, Caldera OpenLinux, while using the RPM system, is not a Red Hat distribution, but is based on the German LST distribution. I would also like to see more discussion of the File Hierarchy System (FHS), partitioning schemes and disk recovery with emphasis on **fsck**, and other pertinent commands. Hopefully, they'll also close some of the really gaping holes, like forgetting to even mention one of the major Linux distributions, Debian, not to mention all the foreign distributions, such as S.u.S.E.

Overall recommendation for this book: look twice before you buy. Managers who don't actually administer systems, but make decisions regarding whether Linux is “appropriate”, or intermediate users who can use the better-fleshed-out chapters to complement other manuals they have, could find the book a good value. Otherwise, I'd hold out to see if the next edition is a more solid investment.

David Bandel is a Computer Network Consultant specializing in Linux, but he begrudgingly works with Windows and those “real” UNIX boxes like DEC 5000s and Suns. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from 2,500 feet up in an airplane. He welcomes your comments, criticisms, witticisms and will be happy to further obfuscate the issue. He can be reached via e-mail at dbandel@ix.netcom.com.

Archive Index Issue Table of Contents

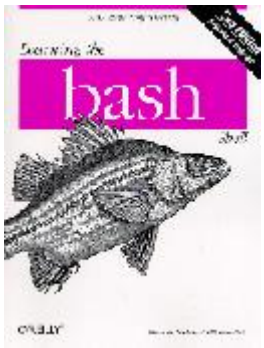
Advanced search

Learning the Bash Shell, 2nd Edition

Bob van Poel

Issue #56, December 1998

bash, an acronym for “Bourne again shell”, is a large, complicated and powerful program.



- Authors: Cameron Newham and Bill Rosenblatt
- Publisher: O'Reilly & Associates, Inc.
- E-mail: info@oreilly.com
- URL: http://www.oreilly.com/
- Price: $29.95 US
- ISBN: 1-56592-347-2
- Reviewer: Bob van der Poel

The user shell is the most-seen and used program on any UNIX system. Through the shell, a user can type commands for file maintenance, launch various application programs, and automate tedious day-to-day system administration. Some computer systems (such as Windows 95) don't come with a shell, but Linux includes a multitude. Most Linux systems have csh, sh, tcsh, ksh and bash pre-installed. **bash** seems to have become the standard shell for Linux.

bash, an acronym for “Bourne again shell”, is a large, complicated and powerful program. It has been developed over many years and is intended as the

standard shell for GNU systems. The GNU distributions of bash come with a manual page and info-style documentation. If you are familiar with UNIX shell programming, then those will probably be all you need to get bash to do what you want. On the other hand, if you've read the man pages a few times and are still scratching your head, you may need more help.

Learning the Bash Shell, as the authors summarize in their preface, “is designed to address casual UNIX and Linux users who are just above the raw beginner level. You should be familiar with the process of logging in, entering commands, and doing some simple things with files.” After reading the book a few times, I agree; however, the usefulness of this book goes beyond simply being a beginner's tutorial. If you haven't earned all the stars on your UNIX-wizard cap, you will find it an often-used reference.

Learning the Bash Shell is a 320-page book divided into eleven chapters, five appendices and an index. It is written in an easy-to-follow style which avoids, as much as possible, the terse style of man pages and jargon. The layout and typesetting make it easy to navigate though the various explanations and examples.

The first three chapters give an overview of the functions of a shell, the extensive command-line editing capabilities of bash and the setting up of a customized environment. The section on command-line editing is one I will read a few more times—the more I understand and begin to use bash's power, the less typing I do. Considering the state of my typing ability, this is a Good Thing.

The next five chapters are certainly the most valuable—they deal with actually programming bash. All the command words, variables and built-in functions are covered. Much of this expands on material in the man/info documentation, with additional comments on when you might want to use a particular command, its history, and in some cases the suggestion that you may never need to use it. Furthermore, most of the explanations are accompanied by examples showing exactly how and when to use the command and its arguments. In good pedagogical style, the authors present a simple example; then, as the reader learns more features, the examples are revisited and expanded until one has a useful and solid shell program. Suggested exercises are included for the reader to work on in order to improve programming skills.

If you've done any programming, you know about the tedious chore of tracking down bugs in your programs. Writing scripts in bash is no different—you will have bugs, and sometimes they will be hard to find. Actually, with the arcane syntax of shell scripts, they may be very hard to find. Chapter nine will help. The standard debugging methods (lots and lots of print statements) are covered,

and an extensive debugger is presented by the author. The most interesting part of the debugger is that it is actually a bash script—neat.

The final two chapters deal with the rather mundane topics of installing bash as your user shell, some security issues, and obtaining and installing bash on your own system. Fortunately, since bash comes pre-installed as the standard shell on most Linux systems, this section can be skimmed through.

The five appendices contain the expected reference lists, BNF (Barkus-Naur Form) syntax, etc.—good, useful information. Finally, the 14-page index makes it easy to use the book as a standard desktop reference.

Learning the Bash Shell was written for the 2.x version of bash; however, any features which are not supported by earlier versions of bash are noted.

Programming in a shell language like bash is quite different from using a high-level language like C. To me, it seems that bash has been kludged together by a large number of individuals who, when adding needed features, merged their preferred syntax into the shell language. The result is that the bash language can be rather convoluted, and it is easy for a beginner to get bogged down in details. For this reason alone, *Learning the Bash Shell* is an important tool. Be aware that writing shell scripts, especially if you have root permissions, can be dangerous to the health of your system. Fortunately, the authors warn you when their examples can compromise system integrity or security.

The examples are short enough for the user to type in, and also available from O'Reilly's FTP site—well, the authors state they are. I followed the instructions in the book and downloaded the source file. Unfortunately, the file is for the first edition of the book and misses the entire debug script. I contacted O'Reilly by e-mail and received a courteous and timely reply advising me that they would contact the book editor to track down the source. Perhaps by the time this review is in print, the correct source will be available.

We have learned to expect professional, well-written, technically correct books from O'Reilly, and *Learning The Bash Shell* is no exception. Writing technical reference books is always a difficult task. It is even more trying when the knowledge base of the intended audience is as varied as the book's potential readers. I think the authors have succeeded in making both an introductory primer for new users and a valuable reference for the more experienced. I highly recommend this book to anyone who wants to become more productive using bash, as well as those who wish to learn to write moderately complex shell scripts.

Bob van der Poel started using computers in 1982 when he purchased a Radio Shack Color Computer complete with 32KB of memory and a cassette tape recorder for storing programs and data. He has written and marketed many programs for the OS9 operating system. He lives with his wife, two cats and Tora the wonder dog on a small acreage in British Columbia, Canada. You can reach him via e-mail at bvdpoel@kootenay.com. If he's not too busy gardening, practicing sax or just having fun, he'll probably send a prompt reply.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Wireless Networking in Africa

PhD. Enrique Canessa

Fulvio Postagna

Carlo Fonda

Gabriel O. Ajayi

Sandro Radicella

Issue #56, December 1998

The experiences of the members of an Italian project in establishing wireless networking with Linux in Africa.

Global connectivity in Africa is in an early stage due to installation costs, insufficient basic infrastructures, low quality of available telecommunication services and limited financial support. The application of wireless technology is an effective choice to overcome some of these problems, at least within smaller areas. This is true even if transmission speeds are lower than the ones achieved by wired networks.

Within a university campus, it is easier to install a radio link system than to place cables or expensive optical fibers in the ground. Furthermore, radio installations are easier to protect from external natural phenomena such as flood, landslide, etc. At first glance, wireless LANs look more expensive than wired LANs, but in the long term they have lower maintenance costs and are relatively easy to configure. The use of Linux and standard radio-communication technologies, in conjunction with the many Linux software applications, makes this task even easier.

With this scenario in mind, the “Programme of Training and System Development on Networking and Radio Communications” was initiated in 1995 at the Abdus Salam ICTP, Trieste, Italy. The objective of this programme is to provide technical assistance and training to academic and scientific institutions in developing countries—institutions with a need for small area computer

networks and a connection to the Internet, either directly or through national networks.

The Abdus Salam ICTP in Italy and the Obafemi Awolowo University (OAU) of Ile-Ife in Nigeria agreed to collaborate in the establishment and future evolution of a Pilot Educational and Research Computer Network at OAU. Such a network, based principally on personal computers running Linux, today provides connectivity between several faculties and departments on the campus.

Training technical staff on the hardware (PCs, cabling, radio techniques) and software (network and system administration) took place initially in Trieste. The developmental and simulation work was completed in four months, ending in January 1996, when all the necessary equipment was sent to OAU. The system was installed in April 1996. At that time, staff members of other Nigerian universities came to Ile-Ife in order to benefit from this exercise and be introduced to Linux for the first time. Besides getting acquainted with the new technology, this experience led to further connections to the OAUNET. The campus network has been in operation since June 1996 without any major problems and has proven to be highly beneficial for academic life at the University.

Figure 1. Diagram of Wireless Network at OAU

The First Campus Network

As shown in Figure 1, the wireless campus network (OAUNET) is based on a radio system in the UHF band; it initially involved three separate buildings and had the capacity to be rapidly extended to other university structures. The wireless link uses a spread-spectrum, direct-sequence technique providing data transmission at 2Mbps. The so-called “spread-spectrum” is a digital coding method in which the signal is transformed or spread so that it cannot be received by any receiver except the designated one that understands the transmitted signal code. It minimizes interference to other users and normally does not require an operation license in the ISM (International Scientific and Medical Band), depending on the regulation adopted by the country.

Inside each building, an Ethernet 10-BASE 2 cabling structure is installed in order to keep the initial costs as low as possible (i.e., no hubs, less cable) and to ensure the local availability of spares (BNC), etc. In each of these buildings, a Linux PC acts as “faculty server” and provides mail services for the local users and does routing to the backbone. This strategy has been selected to keep the user-generated traffic local and reduce the access to the main backbone. All services are TCP/IP-based to keep the system as standard as possible with

Internet protocols, avoiding future modifications when full connectivity might be provided to the university.

The academic network gateway and the main mail host are located the Department of Computer Science at the university. Due to national regulations and the lack of a permanent connection to the Internet, the gateway is linked on a dial-up base (**uucp**) using an international direct-dialing line to the ICTP computer network in Trieste, Italy. Software was developed by OAU staff with some assistance from ICTP to refine the basic uucp mail transfer: a custom **sendmail** delivery program batches mail in intermediate-sized, BSMTP (batch simple mail transfer protocol) formatted files; these files are compressed as much as possible before being transferred over uucp. To cope with telephone line instabilities, a uucp relay was placed in Lagos; the uucp configuration takes care of selecting the path either directly to Trieste or through the Lagos relay, automatically choosing the one that works.

Sticking to Linux

Previous in-house experiences with the UNIX system (SunOS and Sun Solaris) led us to test, within the project, the commercial Solaris 2.4 (x86 version) and Linux. While this version of Solaris required specific hardware components to function on the available (486) PCs, Linux was found to be more flexible than Solaris in terms of hardware compatibility and low memory requirements. On top of that, the possibility of having high-quality free compilers and software applications motivated us to continue using Linux. In 1995, Windows NT was just starting to become popular, so this possibility was not considered. At that time, Linux was also unknown in the Nigerian academic world. The main operating systems available there were MS-DOS and Windows 3.1.

The Linux distribution chosen was Slackware. Although more difficult to install than other distributions, right from the first trials the system configuration (start-up scripts, etc.) was easier to locate, understand, manage and, most importantly, to teach. After a whole cycle of Linux setup experiences and training, autonomous management of all aspects is today a reality. Part of the system installed at the OAU is shown in Figure 2.

Figure 2. OAU Computer Room

Campus Wireless Connectivity

In order to achieve campus wireless connectivity, the requirements needed from the system administrator's point of view were to have a network working all day and night, the least amount of human intervention and reasonable throughput (bandwidth). The technology adopted to do this job was, as mentioned above, spread spectrum.

The first attempts were implemented using a direct-sequence, spread-spectrum card supported by the Linux kernel. Most recently, alternative spread-spectrum equipment that does not require Linux kernel support is being tested, not because of Linux unreliability, but because this new equipment has an Ethernet interface instead of an ISA (industry standard architecture) card that plugs into a PC. This configuration makes the installation of a wireless link much easier and more flexible, because you can place the radio unit very close to the antenna (keeping attenuation very low) and connect it to the PC using a UTP (unshielded twisted pair) cable. In the case of an ISA card, the RF output is at the rear of the PC, and it is there that the antenna is connected. Due to the small output power and the attenuation of the cable, it is not advisable to use more than 10m of coax. This new equipment uses a different spreading method called frequency hopping and provides a higher bandwidth.

The possibility of using packet radio technologies, which are well supported and documented under Linux (AX-25 HOW-TO), was analyzed at the beginning of the project. However, due to the specific training required, implementation of this technology was postponed. An advantage of the present spread-spectrum installation (running two years without major interventions) is that it is almost plug, play and go.

The local network is stable and does not suffer if there is a failure of the main power supply, because a standby generator and UPS facilities are available. The number of registered users of the network increased from 150 in August 1996 to about 290 in September 1996 to more than 600 at present.

On-Line Services

Campus-wide services such as e-mail, FTP, WWW and NFS are available today within the OAUNET. As connection to the rest of the world is done on a dial-up line, only e-mail exchange is provided freely to local users. There is no limitation on the amount and size of the information being transferred on campus.

Some of the communication applications like TALK or WRITE became important due to poor performance of the local PABX (private automatic branch exchange). For example, after the launch of the network, people who used to walk to another location to speak with an administrative officer or colleague now enjoy a TALK between two buildings located a few kilometers apart. To reduce bandwidth usage, only text-based conference tools are implemented. We will also experiment with voice-over input in the future.

All of these new communication tools are certainly providing a revolutionary change in the local academic life.

Next Steps

The next phase of the wireless network requires the installation of other Internet services on the OAUNET. For example, arrangements are being made to provide connectivity to the library and other faculties such as Agriculture.

The success of the Ile-Ife experience stimulated cooperation with other Nigerian universities. A large program of cooperation with the National Universities Commission for the establishment of a national academic network started in 1996 and is still progressing. Linux is part of the technical basis of this activity. The most interesting applications of radio have been in individual universities. Among these is the Bayero University (Kano) which decided to build a link to connect the new campus with the old campus of the university (about 9 kilometers). This connection was implemented using commercial wireless equipment and two Linux machines as routers, with one of them as the whole university mail server and uucp gateway.

Following these results, a series of additional proposals have been received at our headquarters in Trieste. The first of these new projects, being carried out in Ghana in collaboration with countries such as the Democratic Republic of Congo, Ivory Coast and Morocco, will begin soon.

These initial experiences with Linux in Africa are proof of the success and reliability of Linux.

Acknowledgements

Any of the authors can be reached at radionet@ictp.trieste.it.

Dr. Enrique Canessa is a theoretical physicist currently working as a scientific consultant at the ICTP. His main areas of research and interest are in the field of Condensed Matter and scientific software applications. He has been lost in the Internet since 1987.

Mr. Fulvio Postagna is an Electrical Engineering student at the University of Trieste and a Scientific Consultant at the ICTP. He has been a radio amateur since high school and has carried out radio and telecommunications studies. He enjoys upgrading the Linux kernel for our wireless communications.

Mr. Carlo Fonda is a Physics student at the University of Trieste, a Scientific Consultant at the ICTP and radio amateur. From time to time he visits our laboratory after long walks in the marvelous Trieste countryside. He also sends us greetings via radio and enjoys drawing.

Professor Gabriel O. Ajayi works in the Department of Electronic & Electrical Engineering of the Obafemi Awolowo University. Besides computer networking using wireless radio techniques, his main research topics include rain attenuation on earth-satellite paths in the tropical regions.

Professor Sandro Radicella is head of the Aeronomy and Radiopropagation Laboratory and coordinator of the Programme of Training and System Development on Networking and Radiocommunications at the ICTP. His interests include, among others, radiopropagation issues and the relationship between social and economic development and telecommunications.

Archive Index Issue Table of Contents

Advanced search

Sharing Pedagogy with Java

Robert A. Dalrymple

Issue #56, December 1998

An educator uses Linux and Java to share teaching tools with others.

Let's begin by looking at a particular teaching problem I have. Wave motion of all types, including those on the surface of a body of water, can be decomposed into trains of sinusoidal waves which propagate at different speeds and amplitudes. If the frequencies of these waves are similar, the wave motion is characterized by wave groups. In the simplest case of a wave group, take two waves that propagate at nearly the same speed, while the group they create by superposition travels at a different speed (depending on water depth for water waves). Now, how do you clearly present this concept to a class?

Or, on the other hand, what happens when the two wave trains are traveling in opposite directions? Is there a wave group? If the two oppositely directed wave trains have the same frequency, a standing wave system results with no propagation of energy. How do you conveniently introduce these latter variations on the same theme?

Finally, what happens when more wave trains are added or when a wave group reflects back onto itself?

The traditional method of explaining these ideas to a class is to take chalk in hand and draw pictures illustrating the concepts. These pictures, however carefully drawn, are static and fail to transmit the idea that the waves and wave groups are dynamic entities, moving at different speeds and superimposing to create the resulting wave forms.

An alternate method is to use a graphical computer program, allowing students to see the result of superimposing wave trains with any frequencies, amplitudes and directions they wish. By trying different combinations of wave amplitudes, directions and wave periods, students can empirically discover the behavior of the water surface. Also, by allowing them to see the separate wave

component waves, it is clear, for example, that a standing wave is comprised of two oppositely directed progressive wave trains.

The graphical program I use to demonstrate this concept and more than a dozen others was developed with Java on my Linux computer. By writing these programs as applets, they are available via the Internet to students with a Java-enabled browser (preferably Netscape Navigator), to use in classroom instruction, or for anyone who is curious about waves and coastal processes.

Figure 1. The Wave Superposition Applet, Showing a Wave Group

Sharing the Pedagogy

In the traditional model of university instruction, a professor develops the notes for a course. A textbook is often used to provide either a structure to the course material or to augment the professor's lectures. These textbooks thus provide a passive means of utilizing the expertise of another individual to augment the professor's own knowledge, or they contain a desirable format or outline for presenting the material. However, the textbook forces the instructor to adopt most, if not all, of the author's style or approach to a subject. A major financial commitment is made by the students, who must invest in one or more texts.

Java and the Internet provide another avenue of utilizing other pedagogies without making a large investment of time or money in someone else's approach. In addition, the applets can provide useful graphical and computational tools.

The applets discussed here are provided on the Internet (address given in the following section) and are used by a number of other professors around the world to augment classroom instruction as homework, and as laboratory experiments for courses in water wave theory and coastal processes. They were written to support two graduate courses taught at the University of Delaware. Each applet illustrates a single concept, such as the particle motions and velocities under a wave of the user's choice. By typing in new data, the user can explore different scenarios, such as the nonlinear influence of wave height on water wave speed (Stream Function Wave Theory applet).

Java and the Applets

Using Java-enabled web browsers, these programs can be run locally by accessing my web site that hosts the applet code. As an alternative, these programs can also be provided as *applications*, which run locally on the user's machine equipped with Java—especially easy for Linux machines that recognize Java byte code.

My contribution is the Java Applets for Coastal Engineering site that I have developed. This web site provides Java applets which can assist in the teaching of water waves and coastal processes and provide useful tools for others. The applets are hosted at http://www.coastal.udel.edu/faculty/rad/ on my Coastal Engineering Java Page. The index.html page at this URL lists the titles of the applets, a terse description of their purposes, and dates that indicate when the Java code for the applet and the HTML page was last modified. Clicking on an applet title on the list results in the applet running on the user's computer. The applet generally consists of several graphical windows: the first allows the user to input data to the applet and the second shows the results of the computation. In addition, HTML text associated with the applet provides a description of the use of the applet, the concepts it illustrates, and some idea of this theory in the Java program.

Once compiled, all the program elements become separate code segments, referred to as classes. These elements have been converted by the compiler into machine-independent byte code, which is downloaded by clicking on the applet's name on the Coastal Engineering Java Page. The user's browser converts the byte code into a local version of the applet, which then runs on his/her machine. The load on my machine is the downloading of the byte code.

While the web site is hosted by a Sun workstation, the programming and debugging was done with a PC running Linux and the Linux port of Java. The site, http://www.blackdown.org/, has the latest information on the Linux port of Java, an HTML version of the Java-Linux HOW-TO and a list of mirror sites for downloading the Java Development Kit (now JDK 1.1.5). Another reference is the October 1996 issue of *Linux Journal* (http://www.linuxjournal.com/issue30/), which had a number of articles on Java.

Implementation

As this project has developed, the applets have evolved, in part due to user comments and suggestions. The HTML page for each applet has a **mailto:** tag to allow the user to send e-mail directly to me. Also, I have learned new Java tricks and thought of improvements. So, the code-change dates are provided on the index page. Future improvements will include homework-type problems with each applet.

After several months of hosting the applets, it became clear that some academic programs were being hampered by long download times in using the applets over the Internet. I now provide an archive file created with **tar** containing all the source code and HTML pages, so that the Java programs can be run locally at other universities. This tar file contains a Makefile, which provides a convenient way for the system administrator to compile all the

source code at once. After installation, the local user points a web browser at the directory containing the applets and then proceeds in the same way as the user on the Internet, but faster. If you want to try this or see the source code, the anonymous FTP site is www.coastal.udel.edu and the file javapp.tar.Z is in the /pub/programs directory.

I could improve access speed for the Internet user in a couple of ways. For each applet, I could compress all the class files for a particular applet into a single zip file, reducing the number of times the downloading browser has to connect with my machine. (For Internet Explorer users, the files would need to be in CAB format.) Another option would be to move to the Java bean model. The problem with this solution is that the language is evolving rapidly, so the applets would also have to evolve.

Implications

I view the pedagogical advantages of applets such as these to be paramount and potentially leading to large changes in the way education material is delivered (until the next more convenient tool comes along). Another major implication deals with free delivery of course content. What is the motivation for people to provide these programs and what is the benefit to their institutions, particularly if there is a reduced need for textbooks?

Several scenarios may play out in the future. Free course content may soon dominate the Internet. This would follow the model of the Linux operating system and other free software packages. What do the developers get? Recognition. What do the institutions get? The same as they now get with textbooks—recognition. Alternatively, the advent of methods to bill small amounts of money safely over the Internet might permit such sites to charge for each use. This will most likely happen for sites that deliver an entire course on-line.

Conclusion

Java offers a new and flexible way to provide active educational content to augment classroom instruction, both for the local institution and institutions everywhere. Through the Internet or local downloads of applets or applications, the examples developed by one instructor can be shared by all.

Acknowledgements

Robert A. Dalrymple (rad@udel.edu) directs the Center for Applied Coastal Research (http://www.coastal.udel.edu/) at the University of Delaware. He has written *LJ* articles on Scilab, Xfig, Xfm and EXT2tools. He has been using Linux at home and work since 1.0 came out. This spring he built ORCA, a parallel

computer consisting of 8 Pentium II machines linked with a high-speed Ethernet switched network.

Archive Index Issue Table of Contents

Advanced search

Embperl and Databases

Reuven M. Lerner

Issue #56, December 1998

This month, Mr. Lerner returns to the subject of Embperl, showing us how it can be used to edit database records.

Those who have read more than a few of my “At the Forge” columns know that I am a great fan of HTML/Perl templates, which allow us to mix the two in a single document. In October, I introduced Embperl, a templating system that can function as a stand-alone CGI program, but can also be integrated into the mod_perl module for Apache. This month we will take a closer look at Embperl, exploring ways in which it can allow us to edit records in a database.

There are a number of good reasons to use templates. First of all, by putting code and design in the same document, designers and programmers can each modify the elements for which they are responsible. No longer is the programmer the bottleneck when a site decides to change its design, as is the case when dynamic output is produced by CGI programs.

Even when you are unlikely to change the look of a dynamically generated HTML page, Embperl (and similar in-line templating mechanisms that allow you to mix code and HTML) enables you to stick it all together, making the logic easier to follow. I have written many CGI programs in which the dynamic output was dwarfed by the static output—but because even one portion of the resulting HTML page had to change over time, the entire thing had to be within the province of the program.

Since the time I wrote October's introduction to Embperl, the package has been improved significantly. Perhaps the most significant change is that recent releases of Apache 1.3.1 and mod_perl 1.15 free you from having to recompile everything when installing a new version of Embperl. Now, Embperl can be installed and upgraded separately from Apache and mod_perl, just as you install and upgrade other Perl packages from CPAN. Please see the “Resources”

sidebar to learn where to obtain the latest information, including installation instructions, on Apache, mod_perl and Embperl.

Why Databases?

Databases are an increasingly important part of the Web. Using them, we can create customized and personalized sites, bringing people the specific information they want, rather than simply handing them all the information we have.

In addition, databases are designed to store and retrieve information easily. If text files and DBM files are too insecure or unstructured for your needs, consider using a relational database. Relational databases store their information in tables, where each table has columns (describing the various fields) and rows (with one record stored per row). Using multiple tables is where the “relational” part comes in, and it can be an extremely powerful tool. You could probably program this functionality on your own, but doing so would be quite complicated—and besides, someone has already done the work for you.

Relational databases are manipulated using SQL, the Structured Query Language developed by IBM in the 1970s. You don't write programs in SQL; instead, you write “queries” that manipulate one or more tables. Using SQL, you can create tables, modify their contents and request combinations of columns and rows containing particular types and pieces of data.

SQL is not a programming language, so it must be created and submitted to a database server by a programming language. In the past, each database product required its own version of Perl in order to allow access; this led to versions known as Oraperl, Sybperl, et al. Recently, the generic DBI (database interface) has produced a stable and portable database engine that allows access to any relational database with the same interface. The database-specific parts are kept in DBDs (database drivers) loaded dynamically by DBI. Assuming you stick to standard SQL rather than database vendors' proprietary extensions, you should be able to switch database brands by modifying a single Perl statement.

The relational database I use in these examples is MySQL, described by its author as a “mostly free” database. I have been using MySQL for quite some time now, and while it does not have all the optimization and locking features of its larger competitors, it performs admirably—and more features are on the way. For more information on MySQL, see the “Resources” sidebar.

Once you have installed Embperl, you need to tell Apache which documents should be interpreted with Embperl rather than as a straight HTML document.

On my computer (running a modified version of Red Hat Linux 5.1), I put the following in the srm.conf configuration file:

```
Alias /embperl/ /usr/local/apache/share/embperl/
```

In addition, I put the following in the access.conf configuration file:

```
<Location /embperl>
SetHandler perl-script
PerlHandler HTML::Embperl
Options ExecCGI
</Location>
```

In other words, I told Apache that any URL beginning with /embperl refers to files actually in /usr/local/apache/share/embperl, and that any files in /embperl should be interpreted by the HTML::Embperl content handler. After restarting Apache, Embperl was up and ready to run.

Creating our Table

This month, we will create a database consisting of a single table, a list of clients for a consulting practice. One of the central tables in this system is the Clients table, which contains basic information about each client.

Here is the SQL necessary to create this table:

```
CREATE TABLE Clients (
        id MEDIUMINT UNSIGNED NOT NULL
           AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(40) NOT NULL,
        address1 VARCHAR(40) NOT NULL,
        address2 VARCHAR(40) NULL,
        city VARCHAR(40) NOT NULL,
        state VARCHAR(40) NULL,
        country VARCHAR(40) NOT NULL,
        zip VARCHAR(40) NULL,
        contact_name VARCHAR(40) NOT NULL,
        contact_phone1 VARCHAR(40) NOT NULL,
        contact_phone2 VARCHAR(40) NULL,
        contact_fax VARCHAR(40) NULL,
        initial_contact_date DATE NULL,
        dollars_per_hour TINYINT NOT NULL,
        UNIQUE (name)
 );
```

Again, we cannot enter this SQL directly into a relational database server; we must use a program that has been compiled with the correct client libraries. MySQL comes with a program (**mysql**) that allows interactive communication with the database; alternatively, we can use DBI to send the above SQL.

Each column in Clients is defined as a VARCHAR, that is, a variable-length text field. The length of the field is determined by the number in parentheses, which I set to 40 mostly to make other elements of the programming easier. (Over time, I expect to make most of these fields quite a bit shorter.)

The **id** field is special, not only because we define it as an unsigned integer (giving us the option of including up to 16 million different clients), but because it is set to be the "primary key". As far as the database is concerned, each row can be identified uniquely with the primary key alone. We set **id** to **AUTO_INCREMENT**, meaning that MySQL will give the first client an ID of 1 in the archive file, the second an ID of 2 and so forth. Each client will receive an automatically generated, unique ID number.

We also declare the **name** column to be unique, since having more than one client with a given name could be confusing for the people involved. The database would accept several identically named columns, as long as the ID numbers were different. However, we will avoid the possibility of having two clients named "IBM" by checking for this in the database.

You may wonder why we didn't use **name** as the primary key, since it is guaranteed to be unique. We could have done so, and everything would work fine (perhaps a bit slower, since text strings are larger than integers). But consider what will happen if a client changes its name—we would have to update all of the references to that client, since old ones will no longer point to the right place. By making our primary key independent of any information the client changes, we can continue to keep track of the client regardless of what information changes.

Inserting Records into the Table

Now that we have defined our table, we will create an Embperl document that will let us insert new records. (Right now, our table is empty.) An Embperl document is largely the same as an HTML document, so you can use the **<H1>**, **<P>** and **<Blink>** tags as well as regular text, and it will work just fine.

However, you can insert Perl code within the Embperl document by putting it within special square brackets. Here are the four types of square brackets that Embperl understands:

- **[- CODE -]**: Evaluate CODE.
- **[+ CODE +]**: Evaluate CODE, inserting the final value into the HTML document.
- **[! CODE !]**: Evaluate CODE as **[- CODE -]**, but only once.
- **[$ Meta-code $]**: Evaluate Embperl meta-commands.

Thus, we can include this statement:

```
[- $foo = 5; -]
```

and **$foo** will be set to 5—a value that persists over multiple invocations, since mod_perl and Embperl cache such values. If instead we include:

```
[+ $foo = 5; +]
```

then a "5" will appear in the document where the brackets were. If you are unfamiliar with the idea of a "final value from an expression", you might want to end every Embperl block with the name of a variable. Variables return their values, so if you type:

```
[+ @reverse_list = reverse @list; $foo +]
```

then a "5" will be inserted into the HTML document at that point.

Listing 1, add-client.html, is a simple Embperl document that adds a client to the database. It does not check the data we hand it—since MySQL will do much of that for us—although it will show the user any database errors that might occur.

Creating the Form

If you are new to templates, it might take a while to understand the idea of a single file containing both an HTML form and the program necessary to process it. Consider that this is no different from a CGI program producing the form from which it can get input.

Listing 1 contains two parts: form processing and form creation. While Embperl looks at the former before the latter, we will look at creation first, since it is generally easier to handle, especially when working with templates for the first time.

We will have one HTML form element for every column in our table except for **id**, since MySQL generates the ID for us automatically. Later, we'll expand this program to handle editing and deleting of rows in our table, which means we will need to handle one form element for each column and row in our database, in addition to one for the "new" record we will be submitting.

My solution is to give each form element the name of the column to which it is attached, followed by a hyphen and the ID number. The "city" column for the row with id = 5 will be an element named "city-5", and the name of the client with id = 30 will be an element named "name-30". Since MySQL starts auto-incrementing ID with 1, we can use "name-0", "address-0" and so forth for our new entry.

Early on in our program, we will define the **@colnames** array, which will contain the names of the columns in our database:

```
@colnames = (id name address1 address2 city
        state country zip
         contact_name contact_phone1 contact_phone2
        contact_fax
         initial_contact_date dollars_per_hour);
```

Now that we have defined **@colnames**, we can create the HTML form with Embperl's meta-commands. We want to create an entry for each element (except for **id**, since modifying that would create serious problems), so we will iterate through each element of **@colnames**, adding the necessary HTML and remembering to skip **id**. This part of my implementation looks like this:

```
[$ foreach $column @colnames $]
[$ if $column ne "id" $]
<TR> <TD>
        [+ $column +]
 </TD> <TD>
        <input type="text" name="[+ $column +]-0"
         size="40" maxlength="40" >
 </TD> </TR>
 [$ endif $]
 [$ endforeach $]
```

The above code looks a lot like Perl, with good reason. It uses a **foreach** loop, which iterates over the elements of an array (**@colnames**), putting each successive element of the array in a scalar (**$column**). We can then use that scalar value by putting it in square-plus brackets at the appropriate points in our HTML.

You are probably not used to seeing the **endif** and **endforeach** meta-commands in the square-dollar brackets. These tell Embperl where the **if** and **foreach** meta-commands end their scope, just as closing curly braces would do in a standard Perl program.

We set the maximum length of each field to “40”, just as the fields in our table are all defined to be **VARCHAR(40)**. If we were to modify the table definition such that each column were set to a more reasonable size (e.g., **name** should probably be closer to 60, and **contact_phone** closer to 15), we would also want to modify the size of each field in the HTML form. Otherwise, users will blindly enter too many characters, and their input will be silently truncated by the database server. The MySQL DBD (**DBD::mysql**) has a **length** attribute that can be used for such purposes, if you wish.

Processing the Form

Now that we have created the form, let's think about how we can process it once we receive it. The Embperl document will receive the form's name-value pairs exactly as if they were being submitted to a CGI program, although we will have to extract them somewhat differently. The pairs are sent in the **%fdat** hash, in which the hash's keys are names of the submitted HTML form elements, and the hash's values are those values. We can grab the name of the

new client with **$fdat{"name-0"}**, the main telephone number with **$fdat{"contact_phone1-0"}** and so forth.

Inserting a record into a table follows the pattern:

```
INSERT (column1, column2, column3) "
 VALUES ("value1", "value2", "value3")
```

We will want to do something like this:

```
INSERT (@columns)
 VALUES (%fdat)
```

Of course, life isn't quite that easy; we must first create a new array, **@insert_colnames**, with the names of the columns we wish to insert—in other words, everything except **id**:

```
[- @insert_colnames = grep !/^id$/, @colnames; -]
```

Then we turn that into a comma-separated list, which is what we will need for the first part of the INSERT:

```
[- $insert_colnames = join ', ', @insert_colnames; -]
```

With that accomplished, we will use Perl's built-in **map** function to turn **@insert_colnames** from an array of column names into an array of column values. We then convert the resulting array into a scalar, in which each value is separated by a comma and surrounded by double quotation marks:

```
[- $values = join '", "', map {$fdat{$_ .
"-0"}}
        @insert_colnames -]
```

If **@insert_colnames** were to consist of

```
(column1, column2, column3)
```

the above use of map would turn it into:

```
($fdat{"column1-0"}, $fdat{"column2-0"},
$fdat{"column3-0"})
```

which **join** would then turn into:

```
$fdat{"column1-0"}",
"$fdat{"column2-0"}",
"$fdat{"column3-0"})
```

There aren't any quotes at the beginning or the end, but we can add them when we finally construct the query:

```
[+ $sql = "INSERT INTO Clients ($insert_colnames)
        VALUES (\"$values\")"; +]
```

We use square-plus brackets here in order to see (and debug, if necessary) the query we send to the database. Don't forget that if we are using double quotes

to take advantage of variable interpolation, we must escape the double quotes we wish to send in our query with backslashes.

We finally send that query with the statements:

```
[- $sth = $dbh->prepare($sql); -]
[- $sth->execute; -]
```

If there are any errors, print them for the user:

```
<P><B>[+ $sth->errstr +]</B></P>
```

Our new record is now inserted in the database.

This entire form-processing section is unnecessary if the user has not submitted any form elements. In Listing 1, you can see how we used the Embperl **if** meta-command to exclude evaluation of this entire block of code if the user has already done something.

The first time you run this, don't be surprised if everything seems to work *and* you get your original form back. As they say, that's not a bug—it's a feature! If Embperl finds fields in an HTML form that match the name-value pairs in **%fdat**, it fills them in automatically. You can turn this option off by modifying the **EMBPERL_OPTIONS** bitmask field, described in the Embperl documentation.

Creating an All-Purpose Editor

Now that we have seen how to enter new records using Embperl, let's expand the template such that it will allow us to modify and delete existing records, as well as add new ones. You can see the complete listing for such a template in Listing 2 in the archive file, client-editor.html.

The first task is to retrieve existing elements from the database and turn them into a list of form elements the user can grab. As we saw earlier, it will be easiest if we give each form element the name of the column with which it is associated, along with a number indicating its record ID number.

The first order of business is to retrieve rows from the current database. We do that with a SELECT statement, whose syntax looks like this:

```
SELECT column1, column2, column3 FROM Tablename;
```

We set up our query as follows:

```
[- $sql = "SELECT $colnames FROM Clients"; -]
```

Now we prepare and execute the query using the standard DBI syntax:

```
[- $sth = $dbh->prepare ($sql) -]
[- $sth->execute -]
```

The result from a SELECT is a table, which we can retrieve in a number of different ways. Perhaps the easiest method is to grab it as an array reference, then turn that array reference into an array containing the name-value pairs, continuing to fetch array references until we run out. If we use Embperl's **while** meta-command, we can do that fairly easily:

```
[$ while ($record = $sth->fetchrow_arrayref) $]
```

We then grab the **id** column:

```
[- $recordid = $record->[0]; -]
```

We can turn that array reference into an array, using Embperl's **foreach** meta-command to iterate over each element, printing each one except **id** in a table row. If we store the current record (row) number in **$recordid** and the current field number in **$fieldcounter**, we can create this by iterating over the following code:

```
<TR>
<TD>[+ $colnames[$fieldcounter] +]</TD>
<TD>
 <input type="text"
 name="[+ $colnames[$fieldcounter] .
        '-' . $recordid +]" size="50"
maxlength="100"
        value="[+ $field +]" >
</TD>
</TR>
```

We will also add a set of three radio buttons to indicate whether the user wishes to delete this record, modify it or do nothing. We will set “nothing” as the default, since we don't want users to inadvertently delete any elements. We create the radio buttons, using the **modify-** stem just as we would in normal HTML. However, we will add the current ID number to that stem:

```
<P><input type="radio" value="nothing"
 name="modify-[+$recordid +]" checked> Do nothing
 <input type="radio" value="modify"
 name="modify-[+$recordid +]"> Modify this client
 <input type="radio" value="delete"
 name="modify-[+$recordid +]"> Delete this client </P>
```

As you can see in Listing 2, we also added a check box to the initial “new client” form to indicate whether a user is interested in adding a new client. This check box can be hardcoded in HTML, since we are allowing users to add new elements from only that one form, with the pseudo-ID of 0:

```
<P><input type="checkbox"
name="modify-0">
 Add this new client <P>
```

Inserting, Updating and Deleting

Just as add-client.html (Listing 1) was divided into a processing section (the first part) and the form-generation section (the second part), so too is our full client-

editor.html (Listing 2). The above section describes how we will use SELECT to create the HTML form, so all that remains is describing the processing section, which comes at the top of the template.

With add-client.html, we could assume that the user wanted to add a new client. There are now four possibilities: adding a new client, updating an existing client, deleting an existing client and doing nothing at all. While **add** can be true only for **modify-0** (the new record), we have to check every set of HTML form elements that comes to us. The simplest case, of course, is when the **modify-** radio button is set to “nothing”.

If the user wants to add a new record, the element **modify-0** will be checked. We can use an Embperl **if** meta-command to check for its existence:

```
[$ if $fdat{"modify-0"} ne "" $]
```

In other words, if the user checked **modify-0**, we will add a new record, just as we did in add-client.html.

Finding out if the user checked **modify** for one of the records is a bit trickier. We take the names of all submitted form elements **(sort keys %fdat)**, and use **grep** to grab all of those with the **modify-** stem:

```
[$ foreach $clientid
 (grep {($_ =~ /^modify-\d+$/) && ($fdat{$_} eq
        "modify@bb:1.      )}
 (sort keys %fdat)) $]
```

If the above looks a bit intimidating, remember that **$_** contains the value of the scalar currently being handled by grep. We tell grep to return only those array elements that match **modify-\d+** (that is, **modify-** followed by one or more digits), and whose value is **modify**. We then take the array returned by grep and iterate over it using Embperl's **foreach** meta-command.

Once inside the foreach loop, how do we create the SQL query? We first have to grab the ID of the element in question, so that we will update only the appropriate record. We do that by giving:

```
$clientid =~ m/(\d+)$/;
```

This puts the ID value in the temporary variable $1. We then use a combination of grep, map and join to create the list of name-value pairs necessary to complete an UPDATE statement syntax with:

```
UPDATE Clients SET
name1="value1",name2="value2"
 WHERE id = $1
```

We use grep to grab all column names except for **id** (once again, we don't want to change that value). We then filter that result through map, turning the list of column names into a list of **name="value"** pairs. Finally, we join that list together with commas, resulting in the scalar **$pairs**:

```
$pairs = join ', ',
 map {"$_ = '" . $fdat{$_ . "-$1"} . "'"}
 grep (!/^id$/, @colnames);
```

We can then set up the SQL query as follows:

```
$sql = "UPDATE Clients SET $pairs WHERE id = $1";
```

Deleting elements is easier than updating, since we don't need the name-value pairs. We can use the statement:

```
$sql = "DELETE FROM Clients WHERE id = $1";
```

where **$1** matched the number of the current element.

Conclusion

Believe it or not, we are done. This client editor obviously needs some help with its user interface, since it is still possible for someone to enter an illegal value (e.g., a bad **DATE** element for **initial_contact_date**, or a fraction for the **TINYINT** column **dollars_per_hour**). If you have more than three or four clients, this interface quickly becomes tedious. The lack of truly descriptive names for each column gives a hard-to-use look to a program that is far easier and less error-prone than entering straight SQL would be.

However, improving the interface is fairly straightforward once you understand how to perform the four basic database operations: **INSERT**, **SELECT**, **UPDATE** and **DELETE**. Indeed, we have seen that doing all of these in Embperl can be quite simple. Creating alternative interfaces should not be hard to do, given the examples we have already seen.

More importantly, this Embperl template is useful for much more than just the Clients table. By modifying the value of @columns and the name of the table, you could use this same template to modify nearly any record in any table.

I hope you have enjoyed this romp through the world of Embperl and templates. A number of templating systems are now available for doing similar things; even if you are unaccustomed to using such templates to communicate with databases, you should consider getting one of the available packages and trying it. Their power may convince you of their utility, too.

Resources

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. You can reach him at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Linux Security for Beginners

Alex Withers

Issue #56, December 1998

Mr. Withers takes a look at basic security issues and how to solve them using available tools

Security is one of the biggest issues on the Internet today. It affects everyone in one way or another. If you use Linux, it should be a big concern to you. You may think security is for system administrators managing 20 or more machines and not for the average user with a simple PPP link to the Internet. This may in fact be true, for the chances of anything happening are rare. Are you willing to take a chance and trust the security of your system right out of the box?

Ignorance on your part may turn into a powerful tool in the hands of a cracker willing to compromise your system. Is knowing every in and out truly necessary to keep your system secure enough for safe usage? Not really, but one of the best things you can do is become aware of what is available. Many people are intimidated by the subject, since it covers a wide area, but you don't have to be a security guru to be safe. On the other hand, you do need to be willing to get your hands a little dirty.

TCP/IP Basics

Before talking about security, the basic underlying principles of the TCP/IP protocol suite must be understood. There are two parts to TCP/IP: **tcp** and **udp**. I won't go into great detail about the difference between them—mainly, tcp is connection-oriented and udp is connectionless. Both have their advantages and disadvantages, and both are used differently.

These two protocols are the underlying base for applications run over TCP/IP networks. Each machine connected to a TCP/IP network has its own IP address to uniquely identify it. Each application has its own port number on that IP address. A normal connection to the Internet is no different, since it could be considered a giant TCP/IP network. The two files which govern an application's

port and protocol are /etc/services and /etc/protocol. The first, /etc/services, identifies the machine's services and the port number and protocol for each particular service. The second file, /etc/protocol, simply identifies the protocols used in /etc/services.

These two files identify only each service, its port number and its protocol. Where is the application? Instead of having an application running in the background listening for its respective port and protocol and perhaps generating hundreds of daemons, we have only one: **inetd**. **inetd** listens for each service, and when it notices a remote host making a call, it spawns the application bound to that port number. How does inetd know which application goes with which service? It uses its configuration file, /etc/inetd.conf. This file matches the service found in /etc/services with an application found on the system.

For example, let's take a look at a small chunk of that file:

```
ftp stream tcp nowait root /usr/sbin/ftpd in.ftpd\
        -l
telnet stream tcp nowait root /usr/sbin/telnetd\
        in.telnetd
finger stream tcp nowait bin /usr/sbin/fingerd\
        in.fingerd
```

You may be familiar with some of these common Internet applications. But what does all this mean? Beginning with the first column, the variables correspond to the service, the socket type (depends on tcp or udp), the protocol to be used, wait or nowait (depends on tcp or udp), user field, the application or server to be called, and the arguments passed to the application.

TCP/IP Security

Where is the security problem in all this? All of these services offer some kind of access to your system and are the principal means by which a cracker can compromise your system. How do we police it? Let us look again at the fields of the code chunk shown above. The first field, representing the service, can be understood with common sense. If you won't be using that service, there is no reason to offer it. If no one besides yourself will be using your box, comment the line out of your /etc/inetd.conf file. The same thing applies to those services that run independently of inetd, such as web servers (**httpd**) and mail servers (**sendmail**). Each has its own daemon running in the background which must be killed to eliminate them as potential security risks.

The next field of concern is the user field. Run applications under the least privileged user possible. If an application doesn't require root to run properly, don't run it with root privileges.

The last field of concern is the most important for those services you do require to be available. My example above works fine when offering those services, but inetd doesn't give you much control. A far better alternative comes with most Linux distributions: **tcpd**. This daemon wrapper is executed instead of your usual server application, and offers far more protection. It will log requests for services to syslog, and it can allow and deny hosts based on rules specified in the /etc/hosts.allow and /etc/hosts.deny files. The rules can do very complex things you wouldn't normally be able to do, such as allowing or denying certain services for certain hosts. It can also trigger applications based on access of services or requests by remote hosts. The list of possibilities is endless. Details on this subject can be found in the August 1997 *Linux Journal* (issue 40) in the excellent article entitled "Wrap a Security Blanket Around Your Computer" by Lee Brotzman. Many security and administration books covering this subject are also available.

Focusing on your System

Now that you have commented out those services that aren't needed, what do you do about those that are? As we discussed above, you could use tcp wrappers, but that only cuts it for services offered by inetd, and tcpd doesn't necessarily mean your system is secure—those applications can still be exploited. Also, those services independent of inetd and those people who *do* have access to your system must be considered.

Being Aware

The best thing to do is be aware. If you run a news server right out of the box, you could be taking a severe security risk. On the other hand, if you learn everything including known security holes, then you have the opportunity to search for a patch or solution. There are also alternatives such as using different programs. Instead of using an insecure application like TELNET, use one that is more secure and designed with security flaws in mind. A secure replacement for TELNET would be **ssh**; for sendmail, which is notorious for its security flaws, a secure alternative is **qmail**.

What about users who have authorized access, or those who don't but manage to gain access? There are all kinds of known security holes, back doors and other nasty things which can be used for no good. Since you can't beat them, join them. By this, I mean learn all about those exploits; new ones are discovered every day and patches are made to remedy the situation. Several web sites thoroughly document these problems and solutions (see Resources).

The system can also be exploited through **setuid** programs. These are programs which run with the privilege of the program's owner when executed. These programs could even be setuid root and, as a result, when executed they

have the permissions of root. Crackers can use this to gain root privileges. The best way to deal with this situation is to learn about possible problems with programs that run with the root setuid bit set on and disable those programs which are not needed.

Access Restriction

With all of the above in mind, let's look at some nifty tools and methods for internal security. Obviously, someone can compromise your system if they have access. To limit user access on a machine, you use two files: /etc/securetty and /etc/login.access. The first file defines which ttys terminals can be logged into by root. The second limits user access, but is far more flexible. Lines in this file follow the format:

```
permission : users : origins
```

where ***permission*** is either access granted (+) or access denied (-), ***users*** is a list of login names, group names or **ALL**, and ***origins*** specifies "where" a user can log in. An example would be the following line:

```
- : ALL EXCEPT bob : ALL
```

This instruction means **bob** is the only one allowed to log in from anywhere—everyone else is denied access to log in from all ttys, hosts, domains, etc.

```
- : ALL : .anytown.state.us console
```

This statement denies access to everyone except those in the domain **.anytown.st.us** and those from the console. With a bit of imagination, one could come up with some pretty complex rules for logins.

setuid

As I mentioned above, setuid programs can be hazardous. One way to deal with these programs is to find them first. This can be done with a simple script, using the **find** command as shown in Listing 1.

Be aware that this script will generate a file containing sensitive information. After viewing it, you should delete it. Once you've looked at the list and found any scripts or programs that aren't necessary, you could disable them as root using **chmod** like this:

```
chmod 644 filename
```

Once setuid is disabled, the script or program is no longer a security risk.

Tools

So far, we have discussed a couple of techniques to tighten system security. What about testing the security on your system? Is it vulnerable to attack? Are there back doors? Several tools are available to answer these questions. Satan can scan a system for any back doors or holes that might become potential security risks. Other programs like **netwatch** and **tcpdump** can monitor network traffic on your system. A packet sniffer program, SniffIt, can also help you in many ways. Packet sniffers have a bad reputation, because they can be a security risk to your system, but they can also help you find problems. A lot of network clients/hosts send information using plaintext, which presents a severe security risk.

Using **sniffit** you can test various combinations to see if there is any potential risk. The program can be downloaded from the URL shown in Resources. I won't discuss compiling and installing sniffit, for that's another topic. Once you have the program up and running, you can give it a test drive. To use the interactive mode, which has a nice curses-based interface, type the following command:

```
sniffit -i
```

In Figure 1, you can see two IP addresses: a destination and a source. The source IP is sending packets from port 19 to the destination IP, 192.168.1.2. Notice that port 19 is “chargen” and does nothing but send characters. (Packet sniffing works only in situations with high bandwidth.) If the source and destination port are changed to 21, any TELNET sessions from 192.168.1.1 to 192.168.1.2 can be picked up, thus allowing the viewer to see what the TELNET user is typing in his session. If the user is using **ssh** instead of TELNET, the viewer would see only useless garbage.

Figure 1. SniffIt Screenshot

Conclusion

I have presented only some of the basics of security; however, there is far more to it than this. The best way to make your system more secure is to learn more about Linux security and to grab some of the tools I have mentioned (see Resources). Security is like philosophy—there is no definitive answer, just a lot of questions and books.

Resources

Alex Withers lives in Anchorage, Alaska during the summer where he tries to convince the hordes of tourists that they need a Linux box at home. The rest of

the year, you'll find him studying computer science at Gonzaga University. Alex can be reached at awithers@gonzaga.edu.

Archive Index Issue Table of Contents

Advanced search

bc: A Handy Utility

Alasdair McAndrew

Issue #56, December 1998

Mr. McAndrew shows us how the bc command can be used for prototyping numerical algorithms.

Linux, as with almost all UNIX systems, contains a vast number of little utilities tucked away in such places as /usr/bin and /usr/local/bin. One of these is the GNU utility **bc**.

bc is an arbitrary precision calculator language. It can perform arithmetic (both integer and real) to arbitrary precision, and it supports simple programming. It is started by the command:

```
bc -l files
```

The optional **-l** flag loads a mathematics library, and ***files*** (also optional) is a list of files containing bc commands. There are some other flags, but they do not greatly change the functionality. The mathematics library makes the following functions available to bc:

- **s(x)**: the sine of x in radians
- **c(x)**: the cosine of x in radians
- **a(x)**: the inverse tangent of x (The result is returned in radians.)
- **l(x)**: the natural logarithm of x
- **e(x)**: the exponential function ex
- **j(n,x)**: the Bessel function of order n of x

I used version 1.04 of GNU bc to generate all the examples below. Other versions of bc may be restricted in their capabilities.

Basic Usage

Let's look at a few examples of bc in action, assuming it has been started with the **-l** flag:

```
2^400
258224987808690858965591917200301187432970579282 9\
223512830659356540647622016841194629645353280137 8\
3143590317197274749337 6
scale=50
pi=4*a(1)
e(pi*sqrt(163))
262537412640768743.99999999999925007259719818568 8\
783937098755173667 78
scale=100
l(2)
.693147180559945309417232121458176568075500134360\
255254120680009493393621969694715605863326996418 6\
875
```

The value **scale** is one of bc's internal variables: it gives the number of figures to the right of the decimal point. Other versions of bc do not allow arbitrary values for **scale**. We could easily use 1000 instead of 10 in the following example, if we wanted more decimal places.

```
scale=10
4*a(1)
3.1415926532
```

On my computer, a Pentium 133, calculating pi to 1000 places takes about one and a half minutes to complete.

bc provides most of the standard arithmetic operations:

```
scale=0
920^17%2773
948
.^157%2773
920
```

The period (.) is shorthand for the last result. The percentage sign **%** is the remainder function; it produces the standard integer remainder if scale is set to zero. When bc is invoked with the **-l** flag, the value of **scale** is set to 20.

Programming

Statements in bc are computed as quickly as possible. Thus, when using bc interactively, as shown above, statements are evaluated as soon as they are typed. A program in bc is simply a list of statements to be evaluated. The programming language provides loops, branches and recursion, and its syntax is similar to that of C. A simple example (from the man page) is the factorial function:

```
define f(x) {
if (x <= 1) return (1);
```

```
return (x*f(x-1));
}
```

It is convenient to place such definitions in a file (called, say things.b), and read them into bc with the command:

```
bc -l things.b
```

Then, the output from bc is:

```
f(150)

57133839564458545904789328652610540031895535786 01\
12641825483758331798291248453983931265744886753 11\
14537710787874685420416266625019868450446635594 91\
95922066574942592095735778929325357290444962472 40\
54167907221184454371222696755200000000000000000 00\
000000000000000
```

We can easily write little programs to calculate binomial coefficients:

```
define b1(n,k) {
if (k==0 || k==n) return (1);
return (b1(n-1,k)+b1(n-1,k-1));
}
```

This is a rather inefficient program. The solution:

```
b1(20,10)
184756
```

takes some time to compute. We can, of course, write a much faster program:

```
define b2(n,k) {
auto temp
temp=1;
if (k==0) return (1);
for(i=1; i<=k; i++) temp=temp*(n+1-i)/i;
return (temp);
}
```

Here **auto** is a list of variables which are local to the current program. It is instructive to play with these two implementations of computing binomial coefficients: **b2** gives the result almost immediately, whereas **b1** is very slow for all but very small values of n and k. **bc** also supports arrays; here we use arrays to compute the first 100 values of Hofstadter's chaotic function:

```
h[1]=1
h[2]=1
for (i=3;i<=100;i++)
h[i]=h[i-h[i-1]]+h[i-h[i-2]]
h[10]
6
h[50]
25
```

We can then print out all these values:

```
for (i=1; i<=100; i++) {
print h[i],"       ";
if (i%10==0) print "\n;"
}
1    1    2    3    3    4    5    5    6    6
6    8    8    8    10   9    10   11   11   12
12   12   12   16   14   14   16   16   16   16
```

```
20   17   17   20   21   19   20   22   21   22
23   23   24   24   24   24   24   32   24   25
30   28   26   30   30   28   32   30   32   32
32   32   40   33   31   38   35   33   39   40
37   38   40   39   40   39   42   40   41   43
44   43   43   46   44   45   47   47   46   48
48   48   48   48   48   64   41   52   54   56
```

We see that bc is particularly well suited to prototyping simple numerical algorithms. To give two final examples: computing amicable numbers, and Simpson's rule for numerical integration. First, two integers are *amicable* if each is equal to the sum of the divisors of the other:

```
scale=0
define sf(n) {
auto sum,s;
sum=1;
s=sqrt(n);
for (i=2;i<=s;i++)
  if (n%i==0) sum=sum+i+n/i;
if (s*s==n) sum=sum-s;
return (sum);
}
define amicable(m) {
for (j=1;j<=m;j++)
  if (sf(sf(j))==j && sf(j)!=j && j<sf(j)) print
        j,"        ",sf(j),"\n";
print "Done.\n";
}
```

Then, the command **amicable(2000)** will list all pairs of amicable numbers, at least one of which is below 2000.

Second, Simpson's rule for numerical integration:

```
define simpson(a,b,n) {
auto h,sum_even,sum_odd;
h=(b-a)/(2*n);
sum_even=0;
sum_odd=0;
for (i=1;i<=n;i++) sum_odd=sum_odd+f(a+(2*i-1)*h);
for(i=1;i<n;i++) sum_even=sum_even+f(a+2*i*h);
return ((f(a)+f(b)+4*sum_odd+2*sum_even)*h/3);
}
```

Defining a function *f(x)* by, say:

```
define f(x) {
return (e(-(x^2)));
}
```

and then the command:

```
simpson(0,1,10)
```

returns the result of Simpson's rule for the integral of *f(x)* between 0 and 1, using 20=2*10 subintervals. (The result is **.74682418387591474980**, which is correct to six decimal places.)

Conclusion

In my opinion, bc is a real find: it is small, efficient, self-contained and an extremely useful utility. It is not to be considered a replacement for a good fast programming language such as C, C++ or FORTRAN. But as a means for quickly prototyping numerical algorithms before coding them in a high-level language, it is excellent.

Resources

Alasdair McAndrew lives in Melbourne, Australia, with his wife, three young children and a grumpy cat. He is a Senior Lecturer at Victoria University of Technology, where he teaches mathematics and computing. He is an enthusiastic and satisfied user of Linux, and has been since kernel 0.99; currently he is running Linux on both a desktop and a laptop. He enjoys trawling the Internet for Linux software suitable for children, and when he has time, playing the viola da gamba.

Archive Index Issue Table of Contents

Advanced search

The Wonderful World of Linux 2.2

Joseph Pranevich

Issue #56, December 1998

Mr. Pranevich gives us a look at the changes and improvements coming out in the new kernel.

As any kernel developer can surely tell you, the advent of Linux 2.2 is imminent. Linux 2.1 is approaching near astronomical version numbers in its slow march to completeness, 2.1.108 as of this writing, and all eyes are looking toward the day when 2.2.0 will ship standard in the various distributions. Even if you don't actually follow the Linux kernel version by version, 2.2 is an important milestone to understand. This article is my take on the Linux kernel developments of late, with a significant bias towards x86, the Linux I use most often at home.

Chips Galore

Development in the world of Intel chips is fast and interesting to follow if you have nothing better to do. Merced, Celeron, MMX—the names of Intel technologies float past to be replaced by new cutting-edge technology. (Whether or not these technologies are worthwhile is debatable.) In addition, AMD, Cyrix and other companies have become solid competitors in the market, and each has its own little optimizations, quirks and bugs. It is a mess to keep up with.

Linux 2.2 will be the first stable Linux version to support optimizations for each of these chips and a selection for the processor vendor in the kernel configuration tool for even better fine-tuning. Perhaps even more importantly, Linux 2.2 (and later revisions of 2.0 for obvious reasons) supports bug fixes and workarounds for widespread processor bugs including the infamous F00F Pentium bug. Other bugs that can't be worked around, such as several AMD K6 bugs, are reported during startup.

Merced hasn't arrived yet and probably isn't immediately forthcoming; however, Linux 2.2 has already been ported to SPARC64, Alpha and other 64-bit platforms, so the infrastructure for a 64-bit native kernel is (happily) already in place. There are, of course, other obstacles that would have to be overcome before Linux/Merced could be released, but having a 64-bit ready kernel is an important step.

Multiple-processor machines will now operate much more efficiently than they did in Linux 2.0, with problems such as the global spinlock removed. Up to 16 processors are supported (the same as with 2.0), but the performance difference should be amazing. Also, there is now greater support for the IO-APIC on Intel boards that will make SMP generally better supported.

In terms of other ports, Linux 2.2 will feature improved support for a large number of “mainframe” machines such as SPARC, SPARC64, ARM and Alpha machines. As for “desktop” machines, Linux 2.2 has been ported to both m68k and PPC flavors of the Macintosh with varying degrees of hardware support. (You can expect that support will only get better as we approach 2.4 or whatever comes next.)

On somewhat of a tangent, there is continuing work to support a subset of the Linux kernel on 8086, 8088, 80186 and 80286 machines. This will not likely be usable in time for 2.2, but is something to look for in the future.

System Busses and Assorted Ilk

Although somewhat less crucial, Linux 2.2 will support a much larger percentage of the existing x86 computers with the addition of complete support for the Microchannel bus found on some PS/2s and older machines.

In addition to hundreds of minor patches to the bus system, including many new PCI (protocol control information) device names, larger improvements have taken place. PCI, in particular, has undergone several major changes. First, the PCI device reporting interface has been changed and moved to allow for easier addition of new information fields. This particular change doesn't spell much of a difference for an end user, but it makes the lives of developers much easier. Additionally, it is now possible to choose whether you wish to scan your PCI bus using your compatible PCI BIOS or through direct access. This feature allows Linux 2.2 to work on a larger set of machines: several PCI BIOSes were incompatible with the standards and caused booting problems.

Sadly, little kernel support is available for Plug-and-Play ISA devices. While that would be a great addition, a few problems with the currently proposed system will need to be resolved at some time in 2.3. Fortunately, a great user-level

utility, **isapnp**, is available for setting up PnP devices; it requires a tad more work than I'd like, but gets the job done in true Linux fashion.

IDE, SCSI and USB—Oh My!

As far as Linux IDE is concerned, very few obvious changes have been made. The most obvious one is that it is now possible to load and unload the IDE subsystem as a module, just like SCSI. This has the added bonus of allowing use of a PnP-based IDE controller. For less bleeding-edge machines, the updated IDE driver now supports older MFM and RLL disks and controllers without having to load an older version of the driver. Linux 2.2 also has the ability to detect and configure all PCI-based IDE cards automatically, including the activation of DMA bus mastering to reduce CPU overhead and improve performance. Finally, more drivers have been developed for controllers that are buggy or simply different. It is amazing how even excellent things can continue to become better.

Elsewhere in the IDE world, parallel-port IDE devices have become more common, and are for the most part now supported by Linux 2.2. It is a good assumption that many devices currently not supported will be added as 2.2 progresses.

Unfortunately for devices such as rewritable CD-ROMs, there are still instances where you need to use the newly-added SCSI-emulation driver as a kludge for support. I don't like it, but that's the way it is. This limitation may be removed in future versions of the CD-ROM driver, but will likely still be present when 2.2.0 ships.

The SCSI subsystem's main improvement has been the addition of many new drivers for many new cards and chip sets—too many to even begin to name.

The bad news concerns an ongoing effort to support USB (universal serial bus) and USB devices; so far, any progress made in this area has not been included in a Linux 2.1 release. While this could change before the official 2.2 release, it is unlikely that such a large feature would be included this close to release.

Ports: Parallel and Serial

Nothing much is new on this front; Linux has always had incredible support for these basic building blocks. The parallel-port driver has been rewritten with cross-platform issues in mind, and thus what was once just a “Parallel Port” is now a “PC-Style Parallel Port”, functionality-wise. Note that the naming convention used to label parallel ports has changed, so you may find your lp1 has become your lp0. Distributions should allow for this change automatically.

Serial support is chugging along as well as it always has, with one notable difference. Previously, a serial device such as a modem involved two devices, one for call-in and one for call-out (ttyS and cua, respectively). As of Linux 2.2, the two are combined in one device (ttyS), and accessing the cua devices now prints a warning message to the kernel log. On the bright side, Linux 2.2 includes support for having more than four serial ports, it allows serial devices to share interrupts, and it includes a number of drivers for non-standard ports and multi-port cards. My only complaint about serial support is its lack of support for the standard methods used by modules to pass device parameters at module-load time via the modules.conf file and kmod. Instead, these parameters are set using the **setserial** command.

CD-ROMs, Floppies and Removable Media

Thankfully, the hodgepodge of hundreds of CD-ROM standards has solidified behind the “standard” of ATAPI CD-ROMs. This reprieve has given developers time to completely rewrite the CD-ROM driver system to be more standardized in terms of support. Small, quirky differences between the individual drivers have now all been fixed for better support.

Rewritable CD-ROMs aren't supported as well as I would like. SCSI CD-ROMs are well done, but IDE drives may require the SCSI-emulation kludge driver. This limitation may be removed in a future version of the CD-ROM subsystem, but is something that must be coped with for now.

Floppies are working as great as ever. New developments have been made in terms of large volume floppies, and it remains to be seen whether or not all of these will be supported. The so-called ATAPI floppies already have a driver.

IOMEGA's Zip drive, an increasingly popular storage solution, is fairly well-supported under Linux 2.2. These beasts come in two versions: SCSI and parallel. Under SCSI, the Zip drives are supported just as any other disk would be. The parallel version of these drives actually uses a kind of SCSI-over-parallel protocol, also supported in Linux 2.2. Other IOMEGA solutions such as DITTO drives may also be supported using the **ftape** drivers.

The issue of DVD is something for which no one seems to know the answer. It is highly probable that DVD is already supported in some way, most likely through the IDE ATAPI driver interface. DVDs are much like CD-ROMs. If a standard emerges that Linux 2.2 does not support, it is fairly certain that it will be added sometime during the 2.2.x stabilization cycle following the initial release.

Other removable media may or may not be supported under Linux 2.2. If the device in question connects through the parallel port, it is possible that it is

supported using one of the parallel-port IDE device protocol modules included in the kernel.

Glorious Sounds

At long last, the sound code has been partially rewritten to be completely modular from start to finish. Distributions will be able to more easily include generic sound support out-of-the-box for their users as well as making it easier for the rest of us to load and configure sound devices (in particular, pesky Plug-and-Play ones). Many new sound devices are supported as well, and it looks as if this is one area where Linux will truly improve in the next year.

One notable defect is the lack of support for the PC internal speaker, if only for completeness. Then again, Windows doesn't do it either, so who am I to judge?

Video4Linux

Linux 2.2 now has amazing support for a growing number of TV and radio-tuner cards and digital cameras. This is truly a bleeding-edge addition to 2.1's roster, so some uncorrected problems may remain, but it is reasonable to assume they will be fixed in time. In my opinion, this is just an amazing area for Linux to be in at all.

Back Me Up, Scotty!

Linux 2.2's backup- and tape-device subsystem has not changed much since the 2.0 release. More drivers for devices have been written, of course, and substantial improvements have been made for backup devices that work off of the floppy-disk controller (including the IOMEGA DITTO).

Rewritable CD-ROMs have become a popular solution for backing up data, and they are supported under Linux 2.2 either natively or by using the SCSI Emulation driver. There are still remaining problems in this regard—see my note above on CD-ROMs.

Joysticks, Mouse and Input Devices

Joysticks will be better supported in 2.2, including a large number of new joysticks and ones with an inordinate number of buttons.

Mice in 2.2 aren't very different from those in 2.0. As in 2.0, some inconsistencies regarding mouse support will be addressed in the future. For the most part, mouse control is provided through a daemon external to the kernel. Some mouse drivers deliberately emulate a Microsoft-standard mouse. The reasoning behind this is obvious, but it would be nice if it was decided on in one way or the other. My only other complaint is that Microsoft mice with the

little spinning wheel have no real support, not even using the wheel as a third button. Again, that really isn't a kernel issue. No big problems are present, though.

Additionally, several other input devices are now supported under Linux 2.2, including some digitizer pads. If your devices emulate a mouse (as many do), it is already supported by Linux 2.2 (and, in fact, by Linux 2.0.)

Bits and Pieces

Many smaller additions have been made to the Linux 2.2 kernel to make it more robust, and many of these honestly don't fit in any other category. The loopback driver, which allows you to mount disk images as if they were real drives, has been improved to support better encryption, although there may be issues here with U.S. laws. Also, support is now provided for “initial RAM disks” to allow a Linux user or distribution to boot a kernel with *no* hardware support compiled in, and to load the required device drivers from a small RAM disk. This is useful for systems with Plug-and-Play devices that can't be accessed until after a user-mode configuration program is run. A driver has also been provided in Linux 2.2 to access CMOS (complementary metal oxide semiconductor) RAM directly for whatever reason. A similar driver to access the flash memory of many BIOS was not put into 2.2, but may be included in Linux 2.4. It may still be necessary to boot DOS from a floppy to update your computer's flashable BIOS. Finally, Linux 2.2 allows you to share raw disk images over a network.

File Systems for the World

Linux 2.2 has a wide array of new file systems and partition types to provide interconnectivity. For the Microsoft nut, Linux will now read (and maybe write) NTFS (Windows NT) partitions and Windows 98 (and Windows 95 OSR2) FAT32 partitions. Linux 2.2 also understands Microsoft's Joliet system for long file names on CD-ROMs, and a new type of extended partition invented by Microsoft.

Drivers to read and write Microsoft and Stacker compressed drives are being developed but are not yet included in the kernel.

For Macintosh connectivity, an HFS driver for reading and writing Macintosh disks has been included. HFS+ and older Macintosh file systems are not yet supported. Macintosh partition tables can also be read by the kernel; this allows Macintosh SCSI disks to be mounted natively.

Sadly, OS/2 users will still not be able to write to their HPFS drives. Some updates have been made to the HPFS driver to support the new “dcache” system, but not the hoped-for overhaul.

If there are any Amiga users left, they will be pleased to know that the FFS driver has undergone some minor updates since 2.0. This may be especially useful if the new generation of PPC Amigas uses the same disk format.

For connectivity to other UNIX systems, Linux 2.2 has come forward in leaps and bounds. Linux 2.2 still includes the UFS file system which is used on BSD-derived systems, such as Solaris and the free versions of BSD. Linux 2.2 can also read the partition-table formats used by FreeBSD, SunOS and Solaris. For SysV-style UNIX systems, Linux 2.2 features an updated version of SysVFS. It can also read Acorn's RiscOS disks. Finally, Linux 2.2 features an updated version of the ever-popular Minix file system that can be used for small drives and floppies on most UNIX systems. With so many incompatible formats and Linux 2.2 reading so many of them, it is amazing anyone ever got any work done.

In other news, support for “extended” drives (the format used by much older versions of Linux) has been removed in favor of the “second extended” file system. (This shouldn't matter to many people; Ext2 is far superior to its predecessor.) With the increased support of initial RAM disks, a “romfs” has been created which requires a very minimal amount of overhead.

While not quite a file system, Linux 2.2 includes enhanced support for stretching a file system across several disks transparently. At present, this support can be used in RAID 0, 1, 4 and 5 modes as well as in a simple linear mode.

Video

Perhaps the most surprising and cutting-edge addition to the Linux kernel for inclusion in version 2.2 is what is called the “frame-buffer console” driver (or **fbcon**, for short).

Previously, the Linux kernel (for Intel-based machines) understood and manipulated the video devices only in text mode. Graphical support was to be provided by two other systems: svgalib for console-based graphics and a specialized X server for window-based graphics. This kludgey system often required configuration information to be repeated, and each system supported only a limited slice of the myriad of video devices in common use.

Since this addition is rather new, it remains to be seen whether it will truly replace the previous long-standing duality. Unfortunately, it will be nearly a year after Linux 2.2 ships before this new system is robust enough to support

the cards and technologies we already take for granted as working. My personal opinion is that this is the right idea, but I will hold judgment until I see exactly how far Linus and the developers decide to take this feature.

It is also possible to remove support for “virtual” terminals as provided by the kernel. This allows very memory-conscious people to save just a tad more.

Although unimaginable to the desktop user, Linux can now work even better on systems that do not actually include any sort of video device. In addition to being able to log in over serial or networked lines, as Linux 2.0 and previous Linux versions allowed, it is now possible to redirect all the kernel messages (usually sent to the console directly before any hardware was initialized) to a serial device.

Amateur Radio

Linux 2.2 supports a large array of solutions for amateur radio operators, including a large number of enhancements from Linux 2.0. Unfortunately, this is not my forte—I've never even seen a Linux-based amateur radio station.

Networking: Ethernet, ISDN and the Lowly Modem

I don't have much experience here; I've been using the same network cards in all my machines for several years. However, it is not hard to see that the number of Ethernet and ISDN devices supported in Linux 2.2 has risen sharply. I have been told that newer solutions such as cable modems are also supported.

On the low end, not much has changed. PPP, SLIP, CSLIP and PLIP are all still available for use. I guess some things don't need much improvement. Each of those drivers has been updated in one way or another.

My only gripe in this regard is the continued non-support of so-called Winmodems. Not that I blame Linux for their absence (making modems that are 80% software is a dumb idea anyway), but the idealist in me hopes that one day these pesky devils will be supported like their more usable cousins.

Networking II: Under the Hood

On the protocol front, a lot has happened that I simply don't understand completely. The next-generation Internet protocol, IPv6, has made an appearance. SPX, an alternate version of IPX, is new as well. DDP, the protocol of choice for AppleTalk networking has also been added. Just as you would come to expect by now, the existing protocols have been improved. I only wish I had the need to use some of this stuff.

The list keeps going, however. Linux 2.2 will have an excellent new networking core, new tunneling code, a new firewalling and routing system called “ipchains”, support for limiting bandwidth consumption and a ton more.

File- and printer-sharing protocols have also been markedly improved and enhanced. SMB, the protocol for accessing MS Windows-based shared file systems, has been improved with bug fixes and the like. If you are a fan of NetWare, you'll be happy to know that Linux 2.2 supports a large number of improvements in this area, including access to two different kinds of NCP long file names. Trusty NFS has also been improved, both at the server level and the client level. Finally, those guys over at Carnegie Mellon University have been hard at work developing the new distributed network file system, Coda. This file system supports a large number of highly requested features, including disconnected operations for laptops, an advanced cache system and security improvements.

And, Finally

There's quite a lot that honestly doesn't fit into any of the categories above.

For one, the old system of loading “in and out” drivers (called modules) has been replaced with a system that doesn't require a separate daemon and allows for a smaller memory footprint. This is the **kmod** system which replaces the **kerneld** system. I have to say I think this is a good thing.

Also, the old method of access to file systems has been replaced by the “dcache” system, which may be the fastest virtual file system for any OS currently on the market. It makes you proud to support Linux.

Joseph Pranevich can be reached via e-mail at knight@baltimore.wwaves.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Linux System Initialization

David A. Bandel

Issue #56, December 1998

Mr. Bandel takes a look at system initialization for various distributions.

As the title indicates, I will discuss, in one form or another, how Linux system initialization works. System initialization starts where the kernel bootup ends. Among the topics I intend to explain include system initialization à la Slackware—a BSD (Berkeley Software Distribution) knock-off—as well as System V (five) initialization à la Red Hat, Caldera, Debian, et al., and also point out the differences between them. You'll soon see that the systems are truly more similar than they are different, despite appearances to the contrary. I will also cover passing switches through LILO to init during the boot process—this is used mostly for emergencies.

What I will not discuss (for brevity's sake) are the details in some of Red Hat's, Caldera's or other initialization scripts, specifically configuration information found in the /etc/sysconfig or /etc/modules directories. For those details, you're on your own. Besides, those details are more subject to change from one release to the next.

BSD vs. System V

Back in the days when UNIX was young, many universities obtained “free” copies of the operating system (OS) and made improvements and enhancements. One was the University of California at Berkeley. This school made significant contributions to the OS, which were later adopted by other universities. A parallel development began in a more commercial environment and eventually evolved into what is now System V. While these two parallel systems shared a common kernel and heritage, they evolved into competing systems. Differences can be found in initialization, switches used by a number of common commands (such as **ps**: under BSD, **ps aux** is equivalent to System V's **ps -ef**), inter-process communications (IPC), printing and streams. While Linux has adopted System V inits for most distributions, the BSD command

syntax is still predominant. As for IPC, both are available and in general use in Linux distributions. Linux also uses BSD-style printcaps and lacks support for streams.

Under initialization, the biggest difference between the two (BSD and System V) is in the use of **init** scripts. System V makes use of run levels and independent stand-alone initialization scripts. Scripts are run to start and stop daemons depending on the runlevel (also referred to as the system state), one script per daemon or process subsystem. System V states run from 0 to 6 by default, each runlevel corresponding to a different mode of operation; often, even these few states are not all used. BSD has only two modes (equivalent to System V's runlevels), single-user mode (sometimes referred to as maintenance mode) and multi-user mode. All daemons are started essentially by two (actually more like four to six) scripts—a general systems script, either rc.K or rc.M for single- or multi-user mode, respectively, a local script and a couple of special scripts, rc.inet and rc.inet2. The systems script is usually provided by the distribution creator; the local script is edited by the system administrator and tailored to that particular system. The BSD-style scripts are not independent, but are called sequentially. (The BSD initialization will be most familiar to those coming from the DOS world.) The two main scripts can be compared to config.sys and autoexec.bat, which, by the way, call one or two other scripts. However, the likeness ends there. Having only these few scripts to start everything does not allow for the kind of flexibility System V brings (or so say some). It does, however, make things easier to find. In System V circles (but only in System V circles), BSD initialization is considered obsolete—but what do they know? Like a comfortable pair of shoes, it won't be discarded for a very long time, if ever.

Recall that earlier I said Slackware did a BSD knock-off, and yet it still uses the rc.S/rc.M, et al., scripts. This is because **inittab**, (which we'll look at later) uses the same references to runlevels, and uses those (very much System V) runlevels to decide which scripts to run. In fact, the same **init** binary is used by all the distributions I have looked at, so there is really less difference between Slackware and Red Hat or Debian than appears on the surface, not at all like older BSD systems that reference only modes “S” or “M”.

init: Where It All Begins

Once the kernel boots, we have a running Linux system. It isn't very usable, since the kernel doesn't allow direct interactions with “user space”. So, the system runs one program: **init**. This program is responsible for everything else and is regarded as the father of all processes. The kernel then retires to its rightful position as system manager handling “kernel space”. First, init reads any parameters passed to it from the command line. This command line was the LILO prompt you saw before the system began to boot the kernel. If you had more than one kernel to choose from, you chose it by name and perhaps

put some other boot parameters on the line with it. Any parameters the kernel didn't need, were passed to init. These command-line options override any options contained in init's configuration file. As a good inspection of what's really going on will tell you, runlevels are just a convenient way to group together “process packages” via software. They hold no special significance to the kernel.

When init starts, it reads its configuration from a file called **inittab** which stands for initialization table. Any defaults in inittab are discarded if they've been overridden on the command line. The inittab file tells init how to set up the system. Sample Slackware, Red Hat and Debian inittabs are included later in this article.

inittab Specifics

Reading inittab, we'll be skipping any lines that begin with a “#”, since these are comments and ignored by init. The rest of the lines can be easily read as many other typical UNIX-like configuration tables, i.e., each column is separated by a “:” (***id*:*runlevel*:*action*:*process***) and can be read as follows:

- *id*: This first column is a unique identifier for the rest of the line. On newer Linux systems, it may be up to four alphanumeric characters long, but is typically limited to two. Older systems had a two-character limitation, and most distributions have not changed that custom.
- *runlevel*: The second column indicates what runlevel(s) this row is valid for. This column may be null or contain any number of valid runlevels.
- *action*: This can be several different things, the most common being **respawn**, but can also be any one of the following: **once**, **sysinit**, **boot**, **bootwait**, **wait**, **off**, **ondemand**, **initdefault**, **powerwait**, **powerfail**, **powerokwait**, **ctrlaltdel** or **kbrequest**.
- *process*: This is the specific process or program to be run.

Each row in inittab has a specific, unique identifier. Normally, you will want this to be something easily associated with the specific action performed. For example, if you want to put a **getty** on the first serial port, you might use the identifier **s1**. When I execute **w** to see what processes are running, I can more easily identify who is logged in via the modem on com1 when that user is identified as being on s1.

The runlevels are identified as 0 to 6 and A to C by default. Runlevels 0, 1 and 6 are special and should not be changed casually. These correspond to system halt, maintenance mode and system reboot, respectively. Changing runlevel 1, for example, can have far-reaching consequences. Note that to enter maintenance mode (state 1), you can pass init (via *telinit2*) the argument **1**.

Alternately, you can use **S** or **s** for maintenance mode. If you change what transpires for state 1, the same changes will apply when **S** or **s** is passed. However, runlevels 2 through 5 can be customized as desired.

Many systems have the command **runlevel** (usually found in /sbin). Executing this command will output the previous runlevel and the present runlevel as follows: **N 2**. The **N** indicates no previous runlevel. If you make a change, say, to state 3 and then reissue the runlevel command, you'll see **2 3**.

Since a good demonstration will illustrate better than just telling you about it, try this on your system. (Note that I have done this successfully on Debian 1.3 and a few others, such as an older Red Hat [perhaps 3.0], but not many others, so your mileage may vary.) As root (only root can tell init to change states), issue the init command. You should see a usage message telling you to pass init an argument consisting of a number from 0 to 6, the letters A to C or S or Q. Lowercase letters are syntactically the same as their uppercase counterparts. If you pass init anything other than legal values, you should receive this same usage message. Now pass init the argument **8**, as in **init 8** (or **telinit 8**, if you wish). If nothing appears to happen, don't worry. Now type **runlevel** again, and you should see **2 8**. If you don't have runlevel on your system, try **ps ax | grep init** and you may see **init [8]**. You may or may not see the runlevel listed in square brackets. Once you have confirmed that you actually did change to runlevel 8, change back to your previous runlevel. Note that, should your gettys die, they won't respawn at this runlevel, so you could have a problem logging in again after you log out. If you are unsure what your default runlevel is, look in inittab near the top for a line where the first column is **id** and the third is **initdefault**. The second column in this line is the default runlevel. An example line looks like this:

```
id:3:initdefault
```

This demonstration was designed to show you that while runlevels 7 to 9 are undocumented, they actually are available for use should you need them. (I'll explain later why nothing happened when you changed states). They aren't used only because it's not customary. The customizable states for Linux (2 through 5) are usually more than sufficient for anyone.

The letters A to C are used when you want to spawn a daemon listed in inittab and have this “runlevel” designation on a one-time basis (on demand). Therefore, telling init to change to state C doesn't change the runlevel, it just performs the action listed on the line where the runlevel is listed as C. Perhaps you want to put a getty on a port to receive a call, but only after receiving a voice call first (not every time). Let's further suppose you want to be ready to receive either a data call or a fax call, and when you get the voice message, you'll know which you want. You can put two lines in inittab, each with its own

ID, and each with a runlevel such as A for data and B for fax. When you know which you need, you simply spawn the appropriate one from a command line: **telinit A** or **telinit B**. The appropriate getty will be put on the line until the first call is received. Once the caller terminates the connection, the getty will drop, because by definition, an on-demand process will not respawn.

The other two letters, S and Q, are special. As I noted earlier, S will bring your system to maintenance mode which is the same as changing state to runlevel 1. The Q is necessary to tell init to reread inittab. **inittab** may be changed as often as required, but will be read only under certain circumstances: one of its processes dies (do we need to respawn another?), on a powerfail signal from a power daemon (or the command line), or when told to change state by telinit. So the Q argument will tell init, “I've changed something, please reread the inittab.”

Before I delve into sections grouped by distribution, I'd like to emphasize that they don't stand alone. Each of the following sections will complement the others.

Slackware (BSD) inittab

Let's take a look at the sample Slackware inittab in Listing 1. I've numbered the lines for easy reference. The numbers don't appear in your inittab—your inittab will begin two spaces to the right of the line numbers. Within the inittab file, lines beginning with a “#” sign are disabled and left as explanatory remarks or examples for possible future use. Be sure to read all the comments throughout; they were inserted to help you and may give you a hint on how to better customize your own inittab. Most programs, such as **mgetty** or **efax**, that were meant to run from inittab come with examples of how to implement them.

Since you already know how to read a line (**id:runlevel(s):action:process**), I'm going to cover only those few lines of special interest.

As I've already mentioned, Slackware isn't a true BSD system in the old style. Rather than having just a single-user mode and multi-user mode, it actually uses runlevel 3 as its default runlevel. It runs a system initialization script first, rc.S. This script is designed to be run only once at bootup. Then it runs rc.M. It skips the line with rc.K unless a system operator intervenes and deliberately changes to that state. When changing states between single-user and multi-user modes, the appropriate script is called. (See Listing 1, lines 15, 18 and 21.)

rc.0 and rc.6 are each files that are also run when the system is brought down. (See Listing 1, lines 27 and 30.)

You will see power management (UPS power management) handled in the script as well as the **ctrl-alt-del** key sequence. (See Listing 1, lines 24, 33, 36 and 39.)

Something odd you should notice about this inittab (which was lifted straight from a distribution CD): while the default init runlevel is 3, if a power daemon signals the system to shut down, then power is restored, the shutdown is canceled, and the system is brought back up at runlevel 5. However, since runlevels 3 and 5 are essentially identical (they run the same rc scripts), there is no difference in this case.

Now we come to the standard part which all inittabs were specifically designed to handle: initializing and respawning gettys. When UNIX was young, dumb terminals hung off serial ports. These dumb terminals were called teletype terminals or simply TTYs. So, the program that sent a login screen to the tty was called **getty** for “get TTY”. Today's getty performs the same basic function, although the TTY today is not likely to be quite so dumb. Adding and subtracting virtual terminals is as easy as adding or subtracting lines in the inittab; you can have up to 255.

Next, you'll see a line that allows the X Display Manager (XDM) to be respawned in runlevel 4.

About the only thing I haven't mentioned is that the scripts which do all the work on the Slackware system are all located in /etc/rc.d. Look them over. Slackware uses a minimal number of scripts to start background processes. Specifically referenced by inittab are rc.S, rc.K, rc.M, rc.0 and rc.6. Called by scripts (such as rc.M), but not by init, are rc.inet, rc.inet2, rc.local, rc.serial and others.

Sys V inittab (à la Red Hat)

Take a look at the Red Hat inittab (Listing 2). In this file are some good explanations of what Red Hat does with runlevels. I won't belabor it further here. Note that the runlevels chosen for use by Red Hat are just one convention and not indicative of all System V UNIX systems, not even other Linux System V initializations.

As you can see, Red Hat defaults to runlevel 3, but you can change this to 5 once you have the X server properly configured. (See Listing 2, lines 18 and 56.) Given the number of graphical tools Red Hat has put together, you'd think they'd encourage the use of runlevel 5, but using that as the out-of-the-box default would cause trouble if X was not properly configured first.

Just below the default runlevel, you'll see the system initialization script (Listing 2, line 21). This is run once when the system boots. Then init jumps down to (in this case) line 13 (Listing 2, line 26). The lines for 10 through 12 and 14 through 16 are skipped because our default runlevel is 3.

Notice that **ud**, **ca**, **pf** and **pr** run regardless of the runlevel. When the runlevel column is null, the process is run in every runlevel.

The getty lines should look familiar to you. Don't be bothered by the fact that Red Hat chose **mingetty** over getty. They both do the same thing: send a login banner to the tty.

Finally, runlevel 5 spawns XDM (X Display Manager).

Under Red Hat, you'll find all the system initialization scripts in /etc/rc.d. This subdirectory has even more subdirectories—one for each runlevel: rc0.d to rc6.d and init.d. Within the /etc/rc.d/rc#.d subdirectories (where the # is replaced by a single digit number) are links to the master scripts stored in /etc/rc.d/init.d. The scripts in init.d take an argument of **start** or **stop**, and occasionally **reload** or **restart**.

The links in the /etc/rc.d/rc#.d directories all begin with either an **S** or a **K** for start or kill respectively, a number which indicates a relative order for the scripts and the script name—commonly the same name as the master script found in init.d to which it is linked. For example, S20lpd will run the script **lpd** in init.d with the argument **start** which starts up the line-printer daemon. The scripts can also be called from the command line:

```
/etc/rc.d/init.d/lpd start
```

The nice part about System V initialization is that it is easy for root to start, stop, restart or reload a daemon or process subsystem from the command line simply by calling the appropriate script in init.d with the argument **start**, **stop**, **reload** or **restart**.

When not called from a command line with an argument, the **rc** script parses the command line. If it is running K20lpd, it runs the lpd init script with a **stop** argument. When init has followed the link in inittab to rc.d/rc3.d, it begins by running all scripts that start with a K in numerical order from lowest to highest, then likewise for the S scripts. This ensures that the correct daemons are running in each runlevel, and are stopped and started in the correct order. For example, you can't start **sendmail** or **bind/named** (Berkeley DNS or Domain Name Service daemon) before you start networking. The BSD-style script Slackware uses will start networking early in the rc.M script, but you must always be cognizant of order whenever you modify Slackware startup scripts.

Remember when we changed to runlevel 8 above and nothing happened? Since no subdirectory rc8.d exists and consequently no kill or start scripts, no scripts were run when we changed states. Had we come from boot directly to runlevel 8, we would have had a problem. Only the kernel, init and those daemons started via the **sysinit**, **boot** or **bootwait** commands in the inittab would have been running. I'll let you look at the scripts in the ../init.d/ directory for yourself, but an example for those with Slackware systems is shown in Listing 3.

For those who find editing links to add or delete scripts in any particular runlevel a tedious task or who are just not comfortable doing this, Red Hat distributes a program called **tksysv**. This program uses a graphical interface (using Tcl/Tk) to read the script names in /etc/rc.d/init.d and displays them on the far left side of the application box. If you have a system with init.d in a different location, you can install symbolic links (for each of the rc#.d directories) and it will function just fine, or hack the script and customize it to your system. The system also reads the links in each of the rc#.d subdirectories and displays them for each runlevel from left to right with start scripts above and kill scripts below. (See Figure 1.) You can add, delete and even change the order of execution as you see fit.

Figure 1. System V Runlevel Manager

SysV inittab (à la Debian)

Now take a look at the sample Debian inittab. While similar to Red Hat's inittab, it also has some differences. First, you'll notice that while Red Hat used runlevel 3 for non-graphical mode and runlevel 5 for graphical mode, Debian uses runlevel 2 for both (see Listing 4, line 5). The difference is in Debian's use of a start/kill script for XDM.

I'd also like to draw your attention to a very special line, line 12. The line begins with "~~" (two tildes). Note that in single-user mode (state 1 or S), **sulogin** is called. This prevents someone from just booting the system and becoming root. While it doesn't prevent other tricks from being used to "back door" the system and isn't a substitute for physical security of the system, it does prevent the casual user from obtaining root access simply by rebooting. The use of the command:

```
boot from c: only, vice boot a: then c:
```

combined with password protection of the BIOS setup screens, and a lock on the case to prevent someone from resetting the BIOS on the motherboard, and finally setting LILO to 0 seconds, the computer is almost 50% of the way to being secured from unauthorized tampering. (You can get almost another 45% from the system itself, but note that the last 5% is effectively out of reach.)

Just below the script calls for each runlevel is another line to put a login screen up for root in runlevel 6. This is only for emergencies, should something go wrong with the kill scripts in runlevel 6 and the system does not halt properly. It should never run. (See Listing 3, lines 22 to 30).

The Debian inittab also includes some examples to enable gettys on modem and serial lines, should you find a use for them. The line that invokes mgetty, however, will obviously not work unless you've installed the mgetty package.

Following the logic through a boot-up, during a normal boot **init** knows it will run in state 2. Armed with this information and not overridden during boot-up, init first runs the /etc/init.d/boot script. Once this script has run, init then executes /etc/init.d/rc with an argument of 2. **init** also runs the commands associated with **ca**, **kb**, **pf**, **pn** and **po**. If you read up on **powerfail**, you'll see that nothing will happen until a change occurs with the power. Next, we see that init spawns gettys on the virtual terminals. In this case (runlevel 2), it will spawn six (see Listing 4, lines 50-55). The rest of the lines are commented out, and not used.

Looking at the /etc/init.d/rc script, you can see how it determines what to run to achieve a state change or to bring the system to the initial state.

Emergencies

Editing inittab or any of the rc scripts requires some degree of caution. Even the best tests cannot simulate a complete system reboot, and a script may appear to function properly after a system has initialized but fail during system initialization. The reasons are diverse, but usually involve getting things out of order.

In Caldera's Network Desktop, which ran on a 1.2.13 kernel and used modules, I had modified a script to start the **kerneld** process early in the boot sequence. When I upgraded the system to Caldera's OpenLinux v1.0 which ran a 2.0.25 kernel, I made the exact same changes to the same script, tested it and when I was satisfied all was well, I rebooted. Much to my dismay, the boot process hung, and guess where—yes, loading kerneld. I found that in the newer kernels, kerneld needed to know the host name of the computer, which was not yet available. Things like this can happen to anyone. Something as simple as typing the wrong key or forgetting to give the full path name of a file can leave you in the lurch.

Fortunately, you can pass boot-time parameters to init. When the system boots and you see: **LILO:**, you can press the **shift** key, then the **tab** key to see the kernel labels available for booting. You can then add a kernel label and follow it by any required parameters to boot the system. Any parameters the kernel

needs are used and discarded. For example, if you have more than 64MB of RAM, you need to pass that information to the kernel in the form **mem=96MB**. If you pass the **-b** switch, the kernel won't use it, but will pass it on to init. The same goes for any single-digit number or the letters S or Q in either upper or lower case.

By passing any of the numbers or letters to init, we are overriding the defaults in inittab, as I stated earlier. Most of these numbers or letters do exactly what they would do if passed from a command line on a running system. However, the **-b** is special: it is the emergency boot parameter. This parameter tells init to read the inittab, but for some special exceptions not to execute any of the commands, just drop into maintenance mode. Thus, no rc scripts will be executed. You may mount the system read-write and fix it. One exception to not executing any inittab commands is the process id ~~ that should have as its process **sulogin**. This will give you a prompt for root's password so no unauthorized person can alter system files such as /etc/passwd or /etc/shadow.

What if you've made a mistake in the inittab file? Can the system be saved? Yes, but I must warn you not to do this unless absolutely necessary. Coded into the kernel is the instruction to start init once it is completely loaded and in memory. If the /etc/inittab is corrupted to the point that init can't run, not even with the **-b** switch (I've personally never seen this), it is possible to tell the Linux kernel to run a different program at bootup instead of init. Instead of issuing the **-b** switch, substitute **init=/bin/sh** after the kernel name. This will cause the kernel to run the bash shell, and you will be logged in as root. Be careful here, as nothing else is running, e.g., system logging or the update daemon. This is not a normal mode of operation for the system. Fix whatever is necessary and reboot.

Standards

Now that I've explained a significant part of how Linux system initialization works, I'll tell you how Linux compares to some of the systems I've worked with.

For BSD-style systems, the first time I saw Slackware, I was amazed at its similarity in boot-up to Ultrix which I was using on some DEC-5000s—it has the same structure with the rc scripts in /etc/rc.d and the same names. If Slackware used any system as a pattern, Ultrix could have been one of them. I haven't used any newer BSD-style systems, so I cannot comment further.

For System V, I can compare the various Linux distributions to several others. The one with the most resemblance seems to be Sun Solaris, which uses the same structure as Debian, but uses runlevel 3 as its default and implements XDM startup as Debian. Also, runlevel 5 is used for system shutdown, and the rc scripts are moved to /sbin. HP-UX 10.20 is also similar, but HP puts the init.d,

rc.d and other runlevel directories under /sbin. IBM's AIX uses System V style initialization, but with most of the individual scripts for subprocesses called directly from its inittab. Finally, SCO OpenServer uses a system similar to Debian for its boot-time initialization, but does not use symbolic links to init.d. Instead, all start-kill scripts are located in rc2.d.

The latest Filesystem Hierarchy Standard (FHS) v2.0 for Linux dated 26 October 1997 states either BSD or System V style initialization is acceptable. It stopped short, however, of outlining exactly where the rc scripts would go, except to say they would be below /etc, and future revisions to the standard may provide further guidance. I find that unlikely, since Red Hat and Debian, both very popular distributions, do it a little differently. I have no particular preference, and in fact my system has symbolic links which make each look like the other in case an install process makes an invalid assumption about how my systems are configured. I will tell you that as lazy as I am, less typing to start and stop daemons is more to my liking, so /etc/init.d/ gets my vote.

Summary

While this article hasn't been all-encompassing by any means, hopefully you've gained some knowledge of how your Linux system initializes during boot-up. All these tables and scripts are simple ASCII text files easily modified with **vi** or any text editor of your choice. Just read them and follow their logic. I've shown you how to read and interpret /etc/inittab and provided you with basic information regarding how init works.

I've also shown you how to recover in case you've managed to create a script that hangs the boot process or prevents init from starting. Take a look at your inittab and the scripts it runs to better understand your system and optimize it for your own use.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue56/3016.tgz.

David Bandel (dbandel@ix.netcom.com) is a Computer Network Consultant specializing in Linux, but he begrudgingly works with Windows and those “real” UNIX boxes like DEC 5000s and Suns. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from 2,500 feet up in an airplane. He welcomes your comments, criticisms, witticisms and will be happy to further obfuscate the issue.

!���d����j�� Y��0x'�9��W Md"")ىU1ﻫ_ ...
...
...

Letters to the Editor

Various

Issue #56, December 1998

Readers sound off.

Correction

In my article “Little Devil Called tr” in issue 53, a mistake was made by the editors. In my submission I wrote:

> Many UNIX editors allow some text to be processed by the shell. Take for instance vi with:

```
!}tr A-Z a-z
```

> It replaces all uppercase characters of the next paragraph to lowercase. Another example:

```
!jtr a-z A-Z
```

> This one capitalizes the current and next line (the character after the “!” is a movement character).

The editor changed this by prepending a “:” to the commands. That is definitely wrong; in that case, you would start a subshell and it would try to run **}tr A-Z a-z** and **jtr a-z A-Z**. Both of which would most likely fail. Without the “:” prepended, some lines of text (to be determined by the movement character) are piped to tr and the output is inserted back.

—Hans de Vreught hdev@kbs.twi.tudelft.nl

Author's Correction

In “Training on a Token Ring Network” (September 1998), I referred to the wrong IBM token ring card. The article should have stated the IBM token ring ISA card. I apologize for this bug in my article.

—Charles Kitsuki kitsukic@computer.org

PPP User Interface

There has been a discussion going on about PPPui (a GUI for **pppd**) and the ways to check a PPP connection. It is not difficult to do. There is no need for special programs or to direct syslog to Console 9 as one reader suggested. I think the easiest way is to run **pppd** with the **-d** (debug) option and **chat** with the **-v** (verbose) option and then, during the process of establishing the PPP connection, just run **tail -f /var/log/messages** and all the details will be output (including the assigned IP address and so on) to this file.

—Mihai Bisca mbasca@univermed-cdgm.ro

How Do You Spell Stability?

I know it may sound a bit childish, but I wanted to show this to the community:

```
bernward:~$ uname -a
Linux bernward 1.2.13 #2 Mon Dec 9 10:33:11 MET 1996 i486
bernward:~$ uptime
  5:32pm up 430 days, 1:55, 3 users, load average: 0.00, 0.02, 0.00
```

This Linux box has performed admirably since I installed it back in 1995. It is an aging 486/33 with 32MB RAM and some 2GB of SCSI disks. It serves as a primary DNS and mail relay for our whole European WAN (a few dozen sites) and handles an average of 200+ MB of e-mail a week. It also carries out other menial tasks, such as network monitoring and some form of gateway between UNIX and Netware. It has never crashed once.

That said, I know I'm not the only one. A recent poll organized on http://slashdot.org/ showed that around 10% of the participants had a Linux box with an uptime above the one year mark.

—Philippe Andersson philippe_andersson@ste.scitex.com

Article Listings

Some of the articles in your excellent magazine include program listings. I know it is possible to get these listings if I know the exact location/file name; however, if I don't remember this, I am lost. I can browse the “Table of Contents” on the web site and find the article in question, but there is no link or clue as to where the program listings might be found. How about including a link to each article's program listing from the web site TOC?

—Jan Thomas Moldung janth@x2.hd.uib.no

Good suggestion-we've put links into the TOC. All listings are located at ftp://ftp.linuxjournal.com/pub/lj/listings/issue##/, where ## is the issue in question. Inside each issue directory is a README file identifying the article to which each archive file corresponds —Editor

Congratulations

I have been using Linux for some time now and like to show it off to my friends. One embarrassing problem for me lately has been the need to boot from a floppy, because a hard drive was too large for my BIOS and I couldn't configure LILO properly. A recent response in the “Best of Tech Support” column supplied my answer, and now it boots fine.

The technician who installed the drive under Windows 95 had to shorten it to accommodate the BIOS and Windows 95. Linux has allowed me to reclaim the rest of the drive and ditch Windows 95.

Your magazine is my favourite Linux resource. Congratulations.

—Stephen Roach stroach@altnews.com.au

Minor Correction—Issue 53

I just finished another fine read—nice issue as always. One minor error I noticed though is on page 45, “Technical Considerations” by Richard Kent.

He says, “Within these toolkits are functions which have a variable number of arguments, much like the standard **printf** system call.”

Ah, and there's the rub—printf isn't a system call. It is a function from the standard library (section 3). System calls are, as you know, described in section 2.

I know this is minor, but students new to the C programming language have enough trouble with the documentation and how to read it without reference problems.

—Wayne Bjorken wab@courier.cb.lucent.com

How Many Distributions?

I have done years of FORTRAN, C and assembly programming on CPM, NS32000 and DOS systems, developing a system for operating laboratory equipment, and processing and displaying experimental data. After having seen so many platforms disappear, each time forcing a painful migration, I have just

begun moving to Linux. The first issue I confronted, and resolved by making a semi-random choice, was selecting the distribution. There seems to be no guidance for newbies in this matter. After I got it installed, I started trying to learn how to program it. Here I ran into another obstacle. I have been spoiled by the packaged and documented software from Borland. Now I have to find tools for Linux and instructions for using them by looking in a huge collection of books and Internet sites. I find there is a horrendously steep and bewildering initial portion of the Linux learning curve, which could easily be a barrier to many people. Distributions need to address this if Linux is going to compete with MS Windows.

—Bill McConnaughey mcconnau@biochem.wustl.edu

Interview with Charles Andres

I read your interview with Charles Andres in the August, 1998 *LJ* with great interest.

I just want to add one point on your question about “How does Sun feel about the Open Source movement?” If Sun feels that it might be advantageous for their business to give the source code to everyone, they will do so. Proof of this statement: When Sun tried to start a “Motif vs. OpenLook” war, they freely gave away the source code of XView (which was one of the best X toolkits around). It didn't help them win that war, but in the Linux community XView can still be used (guess what I am looking at ...) for free. All this happened long before the issue hit the newsstands.

Same story with Netscape: if a company feels it can win something, it will open the source. Sun will do it again if they think it would be good for them, but they won't do it for political or philosophical reasons.

—Erwin Dieterich e.dieterich@ndh.net

I'm paying for the content, not advertisements

I have been reading *Linux Journal* for quite some time. It is a very informative magazine and I like it a lot.

However, I cannot help noticing one very bad thing: the portion of *LJ* occupied by advertisements has reached approximately one third of the entire magazine. It seems to be growing even further at the cost of actual content.

While it is clear to me that you earn good money from the advertisements, in the end you still need your subscribers. I am afraid you might lose at least one,

if you continue your transformation from a journal into an advertising bulletin. I hope you will realize this before it is too late.

—Denis Havlik havlik@lisa.exp.univie.ac.at

Actually, compared to other magazines, 30% advertising is low. We need at least this much to stay in business without raising subscription rates. Our November issue was at 35%. If advertising either stabilizes or increases, we will most likely expand the magazine by another 16 pages. It is also true that many readers find value in the ads—I even had one who said we should increase the number of ads —Editor

About the Inventor Article

I am Guy Barrand from Linear Accelerator Laboratory (LAL) at Orsay (France).

In the article “Open Inventor” by Robert Hartley (September 1998), Mr. Hartley mentions the Apprentice project. I have looked at the Apprentice code, and a question has occurred to me.

How far can one go in re-implementing commercial software? In Apprentice, the API differs from that of Inventor only by the prefix “Ap” that replaces the Inventor “So”. A good use of the **tr** command (also documented in the same issue of *LJ*) could easily transform an Apprentice distribution to an Inventor one. Does the Apprentice developer have the right to use the “So” prefix?

In general, is it legal to reuse a commercial product API and to provide a free implementation of this product? I assume that the Linux community has looked at these problems for a long time. Can you enlighten me on these points?

—Guy Barrand barrand@lal.in2p3.fr

Good question—I don't have the answer. Perhaps, one of our readers will know the legalities and let us know —Editor

OSDD Article

I track operating environments for IDC and just read Phil Hughes' article on the Open Source Developers Day (November 1998). Just wanted to take a moment to echo your thoughts at the end of the article regarding applications and Linux.

I often speak to my clients and vendors about Linux (or the “Linux Experience” as I've taken to calling it; one has to have some fun). I have been telling these clients and vendors that applications drive OS sales. It really is that simple.

Although there are many barriers to Linux in enterprise, the recent application development advancements/announcements are a positive step for Linux. However, now is not the time to rest on these victories. When I think about Linux “standards” and applications, I believe that involvement from the beginning by the application vendors is crucial.

—Bill Peterson wpeterson@idc.com

Archive Index Issue Table of Contents

Advanced search

Venture Capital Invested in Red Hat

Dwight Johnson

Issue #56, December 1998

The good news brought by the announcement is that a lot of money is likely to be made using Linux as it penetrates the enterprise market and makes further gains as an Internet OS.

On September 29, 1998 at ISPCON Fall '98 in San Jose, the computer industry was galvanized when Red Hat Software announced that Intel, Netscape Communications and venture capital firms Greylock and Benchmark Partners had taken minority equity positions in the company. Red Hat further announced that with the investment they would establish an Enterprise Computing Division to offer enterprise-grade products and services for support of global and mission-critical applications.

The good news brought by the announcement is that a lot of money is likely to be made using Linux as it penetrates the enterprise market and makes further gains as an Internet OS.

The undisclosed size of the investments was not what caused the excitement -- it was the “who”. Intel was taking a direct, financial interest in the fortunes of this upstart OS called Linux. Everyone recognized this to be a major turning point. There could be only one reason for Intel's interest in Linux—opportunity for profit.

Only the week before, Intel's CEO Craig Barrett, speaking to the British House of Commons, said that estimates of Internet world trade by the year 2002 currently stood at between $450 billion and $500 billion.

Linux has established itself as the leading Internet server OS, capturing 27% of the market, while Windows NT has established a reputation as notoriously unscalable and unstable. Because of the unique leverage of its Open Source development model, Linux is also rapidly becoming the dominant UNIX platform.

When Oracle, Informix, Computer Associates, Sybase and IBM all invest in porting their products to Linux and Netscape Communications invests both money and products in Linux, we can be sure they see a major opportunity.

Any Bad News?

Free software advocates have concerns about whether the entry of big business may actually harm the movement. In particular, buying a stake in only one Linux distribution may deepen already existing divisions. All distributions use essentially the same software; however, Intel's decision to buy into Red Hat amounts to an endorsement that will give Red Hat a decided sales advantage.

Another concern is that Intel may have a corrupting influence. Considerable resentment has already been aroused with the proprietary Intel-endorsed I2O specification. Will comparable adulterations be introduced into the products Intel helps Red Hat produce?

To weigh these concerns, we must consider that advocates have been trying for years to get free software accepted. Enterprise is now asking for Linux, but Linux is not ready. Many free software projects are struggling to produce a product with volunteer help. When we wanted free software, it seemed only logical that we invest our time and energy to create it. Now that enterprise wants free software, it is just as logical for enterprise to make whatever investment is required to create the products it wants.

This is a challenging and pivotal time for free software. Publicly held corporations do not have a moral imperative to protect the spirit of free software but only to bring the greatest return to their shareholders. Many of the free software licenses offer no protection against turning free into proprietary software. Of the Open Source licenses most often used, the GPL offers the most protection that the source code, its distribution and the products derived from it will stay free. Linux and the most essential software development tools are protected by the GPL. Therefore, we need not fear the entry of big business into free software.

Red Hat is deeply committed to the free software movement. There is little possibility that Red Hat will begin producing proprietary products with Intel's money.

This investment should result in more widely deployed free software. The concern that Red Hat is gaining an unfair advantage over other distributions will prove unfounded when it gives the software it develops back to the free software community. This good will most certainly offset current concerns.

Red Hat has shown itself well deserving of the opportunity this investment gives them to lead Linux into the enterprise.

Quotes

Dwight Johnson spends most of the time he isn't sleeping working on http://linuxtoday.com/.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #56, December 1998

Our experts answer your technical questions.

PLIP or SLIP?

Is there an easy way to connect my laptop running MS Windows for WorkGroups to my PC running Red Hat 5.0? Should I use PLIP, SLIP or something else? —Thomas Svanelind

PLIP is probably the cheapest and simplest networking setup around (although it has limitations), and it is supported by both platforms. —Scott Maxwell, maxwell@pacbell.net

Time Synchronization

How does someone configure Time Synchronization with Debian 1.3.1? Specifically, I want my Linux computer to get the current time from another computer. —Michael Breton, michaelb@geiger.com

A pair of time protocols is available for use over the network. The easiest choice is **rdate**, which gives accuracy to within one second. For example, by having this line in my root crontab file, my host “morgana” resyncs with “hyppy” at 10 AM every day:

```
0 10 * * * rdate -s hyppy
```

--Alessandro Rubini, rubini@prosa.it

Netscape Mail

I am using Red Hat 5.0. I want to use Netscape for mail; however, the SMTP mailer (Sendmail) and the POP3 mailer clash to the extent that some mail is coming into Netscape and some is going to Sendmail, accessed via **elm**.

I would like all mail to go through Netscape (POP3) via the ISP. How do I stop Sendmail or halt it? Thanks in advance for any help. —Bill Lunnon, bill@mirrim.demon.co.uk

Sendmail is most likely sending out mail using your full host name. When a recipient replies to a message you sent from elm, the reply is sent directly to your computer and not to your mailbox at your ISP. The easiest workaround is to set the **Reply-to:** field in Netscape's mail preferences to your correct e-mail address. A better solution is to edit the /etc/sendmail.cf file to set the domain Sendmail masquerades as. This will exclude your host name from mail sent out using Sendmail. —Peter Struijk, info@linuxjournal.com

Changes After Restoring /home

Somewhere along the line I thought I messed up the file systems under the /home directory (I have a number of user accounts here), so I restored /home using a **tar** backup file. Ever since then, I have not been able to get information when running various things like **finger** or **whoami**.

When I do whoami, I get the following message instead of the alias name: "**whoami: cannot find user name for UID 500**" where I used to see **grb**. When I do a finger while logged on as myself or another user, I get this message: "**No one logged on**" instead of a list of users logged onto the system.

When I type the command **ls -al**, I get the user ID number and group ID number instead of the login name: e.g., instead of user and group ID both being **grb**, they have the value **500**, which is correct according to what I assigned when originally setting up the accounts. However, I used to see the alias names.

I cannot get the Caldera Desktop to run while logged in as any user, and before, everything ran just fine. When running the **startx** program—it tries to initialize and I even see a color background—the server bombs and goes back to the /dev/tty screen. Looking at the Xerror logs doesn't give me any information as to why it is now failing.

I can still start a customized X session with a TWM-type desktop while running the XwinPro PC X package on my PC. I can also run programs on the client X server with no problem. I am at a loss as to what to do to get the system restored or corrected at this point. Can you offer an explanation and a possible fix to this problem? Thank you in advance for any help you may offer. —George R. Boyko, grb99@nni.com

These symptoms sound a lot like problems with permissions. This crops up sometimes when you restore with **tar**. If you do not set your umask to 0 before doing the restore, then tar will obey the current umask when restoring files. In

addition, tar finalizes permissions at the end of the restore process. So, it is possible to end up with incorrect ownership or permissions if the restore is interrupted before completion. If you restored /home by doing a full backup and then stopping it once /home had been restored, you could end up with home directories or other files with wrong ownership or permission bits.

To fix your problem, you will have to check that all of the directories in /home are owned by their proper owners and that the permissions give at least user read and write. Use **chmod -R** and **chown -R** to recursively change permissions or ownership of directories.

You should also check the /etc/passwd and the /etc/group files to be sure they are world-readable (**chmod a+r**). If they are not, this would explain the problem with ls reporting UID instead of name. Also, verify that the files /var/log/wtmp and /var/run/utmp are world-readable as well. If they are not, this would explain the problem with finger.

As for the Caldera Desktop and X, that could be a problem with the permissions of /tmp or /var/tmp. Type:

```
chmod a+rwxt /tmp /var/tmp
```

to set proper permissions (world read-write, sticky) on those directories. —Bob Huack, bobh@wasatch.com

LILO Question

I use Slackware 3.5. A long time ago, I recall being able to type the word “single” at the LILO prompt. This would immediately drop me into the system as root. It was extremely useful when something wasn't working. Recently I had a need for it, and it didn't work.

Did it truly go away? If so, when and why? How can I put it back? Is there some way of specifying the runlevel from LILO? —Walt Stoneburner, wls@wls.wwco.com

Try using **linux single**. If your default boot configuration in /etc/lilo.conf is not named **linux**, substitute the proper label.

Similarly, you can specify the runlevel from the boot prompt with **linux N**, where **N** is the runlevel number. You should be able to use an **append=** statement in lilo.conf to do this as well, although it is probably simpler just to edit the /etc/inittab file. —Bob Huack, bobh@wasatch.com

VSF Error Messages

I am using Red Hat 5.1. How do I correct a VSF error message? I recently compiled a new kernel from the newly released Red Hat 2.0.35 source tree.

I did a **make dep; make clean; make zImage**. Everything went fine during the compile phase. I ran LILO and set up my lilo.conf file to test my new kernel. Now, every time I boot, my system stops and displays:

```
VFS: Cannot open root device 16:01 Kernel panic:
VFS: Unable to mount root fs on 16:01
```

How do I resolve this problem? —Marlon, yu133048@yorku.ca

The kernel tries to mount your root directory from /dev/hdc1 (16:01 is a hex number representing the /dev/hdc1 as major:minor number).

It looks as if you don't have a valid Linux partition on /dev/hdc1. You should add **root=/dev/hda1** (or whatever your root partition is) to the LILO prompt or to the **append=** line in /etc/lilo.conf. The LILO-mini-HOWTO describes this in detail. —Alessandro Rubini, rubini@prosa.it

Networking

I need a little help with hardware. Here is the scenario: I am a networking student at a local college. Besides learning about networking with NT, Windows 95/98 and Novell, I am also learning to speak Linux. I have Red Hat 5.0 and can install it without any problems. My problem is not with installing or running the system (so far), but rather it stems from hardware compatibility, as I would like to run (in different partitions, of course) NT, Windows 95 and Linux (and possibly Novell) on the same machine.

I have gone through the compatibility lists for both NT and Linux and found (from the hardware vendors I've contacted) that most, if not all, of the hardware listed in both the compatibility lists is outdated and can barely be obtained. I need NICs and video cards that are compatible with NT, Windows 95, Linux and Novell for the same machine.

Granted, I could not and would not venture to try to learn all the operating systems at the same time while hoping to maintain sanity, but I will be running a Windows NT server and Linux on one machine and Windows 95 and an NT Workstation on another as I venture to learn one system at a time and put some time into learning Linux in whatever spare time I have.

A contact for a Linux hardware vendor in my area (Miami, Florida) would be awesome. Any help with this will be greatly appreciated. —Tim Rodriguez, twr@bellsouth.net

Lots of NICs and video cards are compatible with both Linux and Windows. I do admit most recent hardware such as video cards may not be immediately supported by Linux (though they come with a Windows 95 driver), but I don't think you'll have to wait too long to see them working on Linux.

High-performance graphics cards such as ATI AGP are supported by XFree86, and a version of XSuSE is available for the latest G200 Matrox card. If you want to use up-to-date graphic adapters, you can consider Linux commercial servers such as XiGraphics (http://www.xigraphics.com/) which supports most high-performance adapters.

For Ethernet cards, you should check out http://cesdis1.gsfc.nasa.gov/linux/drivers/. —Pierre Ficheux, pierre@lectra.com

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #56, December 1998

Raritan MasterConsole MX4, VRtuoso, Debian GNU/Linux ARM and more.

Raritan MasterConsole MX4

Raritan Computer, Inc. has announced its MasterConsole MX4 switch, an extension of its MasterConsole product line. The MX4 is a matrix KVM switch that enables up to four users simultaneously to control up to 256 computers (PC, Macintosh, Sun, DEC Alpha, RS/6000, HP9000 and Silicon Graphics) each from a single keyboard, monitor and mouse. It eliminates the cost and clutter of unnecessary peripherals, reduces equipment space and improves operational productivity for applications such as multi-user management of multiple servers. MX4 models are available for 8 or 16 computers, with suggested retail prices of $2,695 US and $5,195 US, respectively. Both models support high-resolution video up to 1600x1280.

Contact: Raritan Computer, Inc., 400 Cottontail Lane, Somerset, NJ 08873, Phone: 732-764-8886, Fax: 732-764-8887, E-mail: sales@raritan.com, URL: http://www.raritan.com/.

VRtuoso

Bittco Solutions has announced the release of VRtuoso. VRtuoso is for anyone, particularly the business professional, who needs to harness information available on the World Wide Web. VRtuoso reads and organizes information, presenting web sites, searches and bookmarks in a searchable and meaningful VR landscape. VRtuoso clusters web documents in the VR landscape according to content and meaning. This form of presentation enables users to quickly

locate, correlate and apply information on the Web to tasks at hand. Pre-release versions of VRtuoso are available for download from Bittco's web site.

Contact: Bittco Solutions, 220, 80 Chippewa Road, Sherwood Park, Alberta, Canada T8A 4W6, Phone: 403-417-8850, Fax: 403-417-5199, E-mail: dean@bittco.com, URL: http://www.bittco.com/.

Debian GNU/Linux ARM

The Debian GNU/Linux ARM distribution is a port of Debian to the ARM architecture and will run on Corel's NetWinder. Development will eventually target other ARM processors as well. The ARM port will most likely be included in the next distribution of Debian. Information on the current status of the distribution can be found at http://www.debian.org/ports/arm/.

Contact: Debian GNU/Linux, E-mail: press@debian.org, URL: http://www.debian.org/.

LinuxCAD

Software Forge, Inc. has announced the release of the computer-aided drafting program LinuxCAD. LinuxCAD supports 2-D drafting, 3-D modeling, print and plot, customization and CAD application development. User interface for the program is very similar to the interface for AutoCAD, so former AutoCAD users will have no problem migrating to LinuxCAD. The price for LinuxCAD starts at $99 US.

Contact: Software Forge, Inc., Phone: 847-891-5971, E-mail: sales@softwareforge.com, URL: http://www.linuxcad.com/.

Perspective for Java

Three D Graphics has announced Perspective for Java, a 100% pure Java Class Library, JavaBean and a fully pre-configured Java 1.1 applet that permits real-time creation and manipulation of professional charts live on the Web. The program has a full set of properties, methods and user interface tools (widgets) for Java developers who wish to create data-driven graphics. Perspective for Java works in any Java-compatible development environment, browser or operating system. This product is also designed to support the latest in browser technology, including Netscape 4.0. Site license fees start at $4900 US with a minimum of 100 seats. It is available for developer evaluation through a download on the company's web site.

Contact: Three D Graphics, 1801 Avenue of the Stars, Suite 600, Los Angeles, CA 90067-5908, Phone: 310-553-3313, Fax: 310-788-8975, URL: http://www.threedgraphics.com/.

MetaCard 2.2

MetaCard Corporation has announced the release of MetaCard 2.2. MetaCard has a fully functional development environment on all platforms and doesn't rely on limited-function “players”. The product includes built-in support for popular audio, video and image formats; no shared libraries, virtual machines, browsers, installers or other add-ons are required. This results in no DLL or JVM version conflicts, and high reliability and lower support costs for distributed applications. MetaCard's entire development environment including graphical editor, script editor and debugger was built in MetaCard. The free MetaCard Starter Kit is available now from the MetaCard web site and the FTP site at ftp://ftp.metacard.com/MetaCard/. Pricing is $995 US for a single user.

Contact: MetaCard Corporation, 4710 Shoup Pl., Boulder, CO 80303, Phone: 303-447-3936, Fax: 303-499-9855, E-mail: info@metacard.com, URL: http://www.metacard.com/.

EtherPage Version 3.0

Personal Productivity Tools, Inc. has announced the release of EtherPage Version 3.0 of its EtherPage client/server-to-pager messaging system for Linux. EtherPage delivers messages from computer networks to wireless devices, including alphanumeric and two-way pagers and digital cellular phones. One feature of the product is web-based administration, which allows users to configure their own system through a web browser. It also includes user-configurable HTML templates that allow customized interfaces, an enhanced web interface including user list searching, message status display and support for thousands of users. Pricing starts at $595.

Contact: Personal Productivity Tools, Inc., 14141 Miranda Road, Los Altos Hills, CA 94022, Phone: 650-917-7000, Fax: 650-917-7010, E-mail: sales@ppt.com, URL: http://www.ppt.com/.

D3 Linux v.7.1

Pick Systems, Inc. announces the release of D3 Linux v.7.1 with Red Hat Linux 5.1. Pick's D3 DBMS offers rapid applications design, development and deployment with scalability from a single client to thousands. D3 also provides seamless integration with Windows environments. Additional features include built-in B-trees, FlashBASIC language, Access Query Language with unlimited views of information, business rule support, trigger, ODBC and SQL

connectivity. The product offers direct APIs to Visual Basic and an HTML language tool called FlashCONNECT. D3 is available for $300 US per user.

Contact: Pick Systems, 1691 Browning, Irvine, CA 92606, Phone: 714-261-7425, Fax: 714-250-8187, E-mail: sales@picksys.com, URL: http://www.picksys.com/.

Archive Index Issue Table of Contents

Advanced search