

Linux Journal Issue #80/December 2000



Features

Focus: System Administration by *Don Marti*

Port Scans and Ping Sweeps Explained by *Lawrence Teo*

Lawrence Teo explains two common network probes and what can be done to detect them.

High Availability Cluster Checklist by *Tim Burke*

With a variety of clustering services on the market, the ability to determine how well options meet your specific business needs is necessary.

Monitoring Your UPS with apcupsd by *Riccardo Facchetti*

We delve into the details of apcupsd, a program for monitoring and controlling APC UPSes.

PVFS: A Parallel Virtual File System for Linux Clusters by *Ibrahim F. Haddad*

An introduction to the Parallel Virtual File System and a look at how one company installed and tested it.

A Linux-Based Automatic Backup System by *Michael O'Brien*

A step-by-step procedure for establishing a backup system that will save time and money.

Linux System Administration A User's Guide by *Marcel Gagné*

An excerpt from our French chef's upcoming book.

Indepth

Jigsaw: A Revolutionary Web Server for Linux by *Ibrahim F. Haddad*

The design philosophy and essential features of the Jigsaw Web Server exposed.

Elegance of Java and the Efficiency of C++---It's Ada by Frode Tennebø

Tennebø recommends taking a look at Ada.

PHP4 and PostgreSQL: Building Serious Web Applications with Open-Source Software by Tim Perdue

A walk-through of a simple web application to demonstrate the features of PHP and Postgres.

About the Mod: Part 1 by Dave Phillips

An expansion and revision of material found in Linux Music & Sound

Debian Package Management, Part 1: A User's Guide by David Blackman

A how-to for Debian package management.

Typesetting with groff Macros by Wayne Marshall

Reports of troff's death are greatly exaggerated.

SISAL: A Safe and Efficient Language for Numerical Calculations by D. J. Raymond

The benefits of SISAL and a call for action.

Toolbox

At the Forge ATF Jubilee Edition by Reuven M. Lerner

Cooking with Linux Saucy Administration Tools by Marcel Gagné

Columns

Linux for Suits The End of the Tube by Doc Searls

Linley on Linux One World, One Processor? by Linley Gwennap

Focus on Software by David A. Bandel

Focus on Embedded Systems by Rick Lehrbaum

The Last Word by Stan Kelly-Bootle

Reviews

PowerPlant by Jim Gilbert

Debian 2.2 Potato: Memorial to a Hacker by Stephanie Black

Linux in a Box for Dummies by Ralph Krause

Two Books on PHP by Phil Hughes

Programming Perl 3rd Edition by Paul Barry

Linux Music & Sound by Deric Mendes

Building Linux Clusters by Glen Otero

Departments

Letters

upFRONT

From the Editor The Trouble with the Bastard Operator from Hell
by Don Marti

Best of Technical Support

New Products

Archive Index

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Focus: System Administration

Don Marti

Issue #80, December 2000

In this issue we have a bunch of articles on system administration.

In this issue we have a bunch of articles on system administration, so if you're thinking of breaking into this exciting career, where the stress is high but the hours are long, now is a perfect time to pull up a spare Linux box and get started.

Replacing one big UNIX server with a rack full of inexpensive Linux machines sounds like a great way to save money and increase your server application's reliability and performance. But if you're interested in setting up a high-availability Linux cluster, have a look at how well your clustering technology of choice handles the four failure scenarios in Tim Burke's "High Availability Cluster Checklist".

While we're on the subject of clustering, those of you interested in pushing the envelope on Linux cluster performance will want to read Ibrahim Haddad's article on the Parallel Virtual File System. Don't try it on the accounting department's server, though—PVFS is about speed, speed, speed and doesn't offer the level of redundancy they're probably expecting.

One of the first concerns for anyone starting out in system administration should be making backups. In "A Linux-Based Automatic Backup System" Michael O'Brien explains not just how to back up your Linux systems but also how to run a script on your Linux system to back up files on your legacy Microsoft Windows machines.

J. R. "Bob" Dobbs tells us that "Too much is always better than not enough." And that certainly goes for scripting. Marcel Gagné takes scripting to the extremes with an introduction to Expect, the tool whose motto is "Curing Those Uncontrollable Fits of Interactivity". When you want to write scripts to do everything, you might run into a brick wall—a program that tells you to navigate

a menu or enter a command. Instead of giving up and doing it manually, write an Expect script to bend it to your will. You are the sys admin, you are in control, not the machine, not the software, you.

Last month, we complained that APC has not yet published the protocol for communicating with their UPSes. Fortunately, Riccardo Facchetti has figured out how to talk APC-speak—whether APC likes it or not—and so those of you with APC UPSes can safely shut down when the UPS is drained. It's long, but worthwhile reading if you want to protect your whole network from power outages. I'm running apcupsd at home now, and it works great. One important tip: don't forget to check what kind of serial cable you're using and put it in the config file.

You might tend to think of port scans and such as tools for breaking into a network, not for administering it. But read Lawrence Teo's article and try running one on your own network some time. You might find compromised hosts participating in a distributed denial of service attack, improperly configured systems, or just some weenie who decided to put a “personal web server” to serve out stuff that you don't want public.

If you have a system administration idea that works for you, please let us know. We might just want you to write an article about it, and when you send it in, you can hear somebody say, “thank you.”

[System Administration Feature Articles](#)

—Don Marti, Technical Editor

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Network Probes Explained: Understanding Port Scans and Ping Sweeps

Lawrence Teo

Issue #80, December 2000

Network probes are important clues in detecting intrusions. In this article, Lawrence Teo explains two common network probes and what can be done to detect them.

Almost any system administrator of a large network will tell you that their network has been probed before. As cracking tools become more popular and increase in number, this trend is likely to continue. Although network probes are technically not intrusions themselves, they should not be taken lightly—they may lead to actual intrusions in the future. As the saying goes, better be safe than sorry. In this article I'll explain the concepts behind two common network probes, as well as how they're performed and what can be done to detect them.

Port Scans

The most common type of network probe is probably the port scan. A port scan is a method used by intruders to discover the services running on a target machine. The intruder can then plan an attack on any vulnerable service that she finds. For example, if the intruder finds that port 143 (the IMAP port) is open, she may proceed to find out what version of IMAP is running on the target machine. If the version is vulnerable, she may be able to gain superuser access to the machine using an "exploit" (a program that exploits a security hole).

A port scan is actually very simple to perform. All we have to do is to connect to a series of ports on the machine and find out which ports respond and which don't. A simple port scanner can be written in under 15 minutes by a good programmer in a language such as Java or Perl. However, this kind of port scan is easily detectable by the operating system of the target machine. Listing 1

shows the traces produced by such a port scan in a log file (usually `/var/log/messages`) on a Linux box. Notice that a series of connections to various services occurred in a short span of three seconds. Since it's so easily detectable, most intruders will not run this kind of port scan against a machine these days.

Listing 1

Another sneakier, “stealthier” kind of port scan is called the “half-open” SYN scan. In this scan, the port scanner connects to the port but shuts down the connection right before a full connection occurs (hence the name “half-open”). Since a full connection never happened, the operating system of the target machine usually does not log the scan. This concept will be clearer if you're familiar with the inner workings of TCP/IP. In a normal TCP/IP connection, two devices need to complete a three-way handshake before initiating transmission. In a “half-open” SYN scan, the three-way handshake is never completed—the port scanner judges whether the port is open by the response given by the target machine.

Now that we've covered the basic concepts of port scanning, let's talk about the most popular and powerful network probing tool available today—Nmap (Network Mapper). Nmap is capable of conducting both types of port scans that I've discussed so far. It's also capable of performing other kinds of probes—some of which I'll cover later. Listing 2 shows a typical Nmap scan against a machine.

Listing 2

Now (you might be thinking) if “stealth” port scans are possible, how are we supposed to detect them? The good news is that such port scans are detectable using special tools. Solar Designer has developed such a tool called scanlogd, which is a daemon that runs in a background and listens on the network interface for port scans. If a port scan is detected, scanlogd writes one line describing the scan using the syslog mechanism. Listing 3 shows a scan detected by scanlogd.

Listing 3

There are other tools that can detect port scans as well. I'll not cover them in this article, however. If you're interested, you can check out the Resources section at the end of this article. You might in particular want to check out tcplogd, a configurable TCP port scan detector (you can specify what kind of packets to log, avoid flooding and specify trusted hosts with it).

Ping Sweeps

A ping sweep is another kind of network probe. In a ping sweep, the intruder sends a set of **ICMP ECHO** packets to a network of machines (usually specified as a range of IP addresses) and sees which ones respond. The whole point of this is to determine which machines are alive and which aren't. It's a bit like knocking on your neighbors' doors at 3 a.m. to see who's asleep and who's not (okay, don't try that). Once the intruder knows which machines are alive, he can focus on which machines to attack and work from there. Note that there are legitimate reasons for performing ping sweeps on a network—a network administrator may be trying to find out which machines are alive on a network for diagnostic reasons.

fping is a tool that can be used for conducting ping sweeps. **fping** takes a list of IP addresses and sends ping packets to them. Unlike normal ping, **fping** sends one ping packet to one IP address, and then proceeds immediately to the next IP address in a round-robin fashion. Listing 4 shows a simple Perl script that is used to generate a list of Class C IP addresses (**192.168.0.1** to **192.168.0.20**, in our example). Listing 5 shows how that Perl script can be used in conjunction with **fping** to discover which machines in that IP address range are alive on the network. The **-a** switch is used to show only machines that are alive without it (**fping** will show machines that are unreachable as well).

Listing 4

Listing 5

Like port scans, ping sweeps are detectable using special tools as well. **ippl** is an IP protocol logger that can log TCP, UDP and ICMP packets. It is similar to **scanlogd**, where it sits in the background and listens for packets. Listing 6 shows a line in the log file that demonstrates a ping packet being intercepted by **ippl**. Be careful when using **ippl** though—if you're on a busy Ethernet network, you might find that your **ippl** log files (usually at **/var/log/ippl/***) may fill up rather quickly.

Listing 6

There are other ping detection tools available besides **ippl**—unfortunately, I haven't been able to look at them in detail. One that caught my interest was **pingd**, which is a userland daemon that handles ICMP traffic at the host level. One neat feature of **pingd** is that it integrates with TCP Wrappers to allow you to specify who can ping you and who can't in TCP Wrappers' access control files (**/etc/hosts.allow** and **/etc/hosts.deny**).

Other Network Probe Features

Port scans and ping sweeps are just two of many types of network probes. Current network-probing tools have matured significantly. Their continued development and availability means that system administrators will see more interesting probe patterns in the future.

To examine some of these other network probes, let's go back to Nmap. Nmap is able to perform decoy scans. When such a scan occurs, you'll see scans from unique IP addresses on your system, but you won't be able to pick out which one is the real IP address that scanned you. The point of this is to confuse the system administrator, of course.

Besides decoy scans, Nmap also has the ability to remotely identify the operating system running on the target machines. This is done using a technique called TCP/IP stack fingerprinting. We have already seen this in Listing 2, where Nmap correctly identified my target machine as running Linux 2.1.122 - 2.2.14 (my machine was actually running 2.2.12). At the time of writing, the current version of Nmap (2.53) is capable of identifying 465 different versions of operating systems, routers and other devices. Such ability is useful for the intruder because it enables the intruder to identify the weaknesses on a machine since security holes are usually operating system-specific.

If you're interested in other kinds of probe patterns, I highly recommend that you read Stephen Northcutt's book (see Resources). Fyodor's articles on port scanning and TCP/IP stack fingerprinting in *Phrack* magazine are also interesting.

An Ongoing Process

I hope that this article has been useful to you in understanding two common network probes and how they can be detected. However, security, as always, is an ongoing process. Network probes are going to increase, new security holes are going to be discovered and you'll definitely read about these things in the news almost every day. It pays to be up-to-date. For that reason, I recommend that you subscribe to a security mailing list (BUGTRAQ is highly recommended!) or visit newsgroups and security-related web sites frequently.

Resources



Lawrence Teo (lawrenceteo@usa.net) is currently researching distributed intrusion detection techniques for his Honors degree at Monash University in Melbourne, Australia. When he's not working on his thesis or tinkering with UNIX boxes, he can be found wandering around Melbourne in search of good Japanese restaurants.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

High Availability Cluster Checklist

Tim Burke

Issue #80, December 2000

With a variety of clustering services on the market, the ability to determine how well options meet your specific business needs is necessary.

In today's competitive environment, the adage "time is money" takes on literal meaning. Keeping your business' data on-line and accessible is the foundation of overall system uptime. Whether it be database back ends, web servers or network file systems (NFS) used as e-mail and user directories, outages in your data storage tier can be catastrophic.

The most cost-effective approach to increasing your site's overall reliability is to implement a fail-over cluster. Fail-over clusters involve pooling together multiple computers, each of which is a candidate server for your file systems, databases or applications. Each of these systems monitors the health of other systems in the cluster. In the event of failure in one of the cluster members, the others take over the services of the failed node. The takeover is typically performed in such a way as to make it transparent to the client systems that are accessing the data.

A typical fail-over cluster implementation consists of multiple systems attached to a set of shared storage units, such as disks, connected to a shared SCSI or FibreChannel bus. Each of the cluster members usually monitors the health of others via network (e.g., Ethernet) and/or point-to-point serial connections. Historically, enterprise-quality cluster offerings were the domain of proprietary vendors such as Digital, HP or IBM. Recently, viable Linux-based cluster offerings that run on commodity hardware have become available.

A quick perusal on the Web will uncover a range of Linux-based clustering alternatives. The majority of them look great on paper. They will tout amazingly quick fail-over times for large number of services on clusters consisting of any number of nodes. It is easy to fall into the trap of purchasing the wrong cluster product. The truth is that not all high-availability clustering alternatives safely

increase the reliability and availability of your data. Rather, choosing the wrong type of product can leave your valuable file systems and databases vulnerable to corruption. Some products neglect to mention this fact; others only will state this fact if you dig deep under the hood in related white papers.

Being in the UNIX/Linux high-availability business for more than seven years, I have seen cluster products come and go. It's unnerving to see cluster products promoted for jobs they are ill-equipped to perform. Risking end-user data to corruption gives the whole cluster scene a bad name. I have culled through years of investigation to create a simple four-point checklist that serves as a guide for evaluating whether a high-availability cluster product matches your needs. In fact, these points are not particular to UNIX or Linux; they apply across any hardware and operating system implementation. So before dedicating any money (and your company's data) to a high-availability cluster solution, be sure you know how the solution protects you from the following four failure scenarios:

1. Planned maintenance and shutdown
2. System crash
3. Communication failure
4. System hang

We will be discussing each of these points in detail and pointing out typical pitfalls. But before getting into the analysis of these four points, it is crucial to have an understanding of what data integrity is all about. The fundamental point of data integrity is knowing that your data is accurate and up-to-date. Sounds simple enough. In a cluster environment, preserving the integrity of the data is of paramount importance and supersedes even data availability.

Using examples helps to illustrate the point. The diagram in Figure 1 depicts a two-node cluster (I am using a two-node cluster for simplicity, the concepts apply to clusters composed of more than two nodes as well) with cluster members A and B connected to a shared SCSI bus with Disk 1.

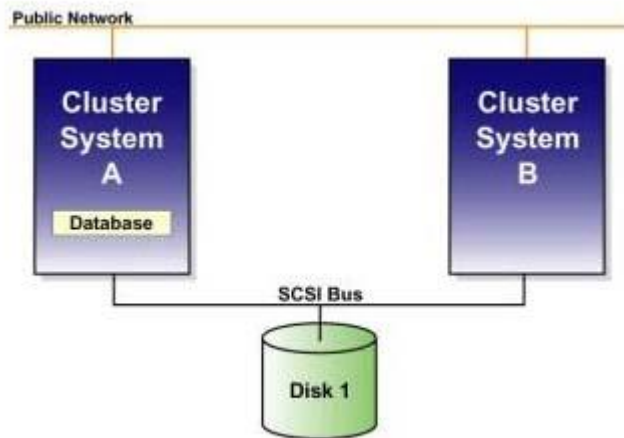


Figure 1. Two Node Cluster with a Shared SCSI Bus

Typical operating systems provide access to disk-based storage via file systems that, in turn, access disk storage. Commonly, the file system mounts the disk volume and then accommodates user access. In the interests of performance, file system implementations typically cache recent copies of file system data in memory. Consequently, the most up-to-date version of your data (being served by node A) is actually the combination of what is cached in system A's memory plus the on-disk data.

Now extend this example to the other cluster member (node B). If node B were to mount the same file system and access it, the true contents of your file system would now consist of the data being cached on node A's memory, plus the data being cached in node B's memory, plus the on-disk data. Making this work correctly requires implementing a file system that coordinates the in-memory cached data of multiple systems in addition to the on-disk data. Such a model, where all cluster members can concurrently mount the same file system, is referred to as a cluster file system. Few UNIX offerings implement a cluster file system and no Linux variants implement a production-ready cluster file system today (although efforts are underway, see the GFS project <http://www.gfs.lcse.umn.edu/>).

In the absence of a cluster file system, what happens if multiple cluster members concurrently access the same file system? Possible outcomes include:

- Inaccurate data—suppose your trip to Las Vegas went particularly well, and you have \$100 to deposit into your bank account. Consider that the deposit transaction was handled by node A, and it added the \$100 to your prior balance of \$25 resulting in a grand total of \$125; node A then keeps your most recent balance in its memory resident cache. You then take a flight home and realize you need to withdraw \$50 to get your car out of the parking garage. This transaction is now being handled by node B,

which goes to the disk and retrieves your balance of \$25 and bounces you out for insufficient funds! All this transpired because the true balance of \$125 is cached in node A's memory. When it comes to a cluster implementation you need to answer this question: How damaging would it be to your company if the wrong data were supplied?

- System crash—in addition to storing user data, such as an account balance, file systems also store their own metadata on disk that describes how user data is organized (consider it an index or table of contents). For performance reasons, metadata is also cached in memory. File systems get particularly confused and upset if *their* metadata becomes scrambled and often resort to temper tantrums (better known as system crashes or panics). In the absence of a true cluster file system, if you ever have more than one cluster member concurrently mounting the same file system, it will result in each node having a differing idea of what the metadata represents, usually resulting in a system crash.

When a file system's data or metadata becomes scrambled, data corruption ensues. To correct a data corruption problem typically means restoring from a tape backup (you do this regularly, right?). The problem here is that since the backup frequency is low in relation to transaction rate, the time it takes to recover from data corruption is often measured in days rather than the small number of minutes or seconds you expected from deploying a high-availability cluster.

The above concepts about requiring cluster members to synchronize their access to file system data to protect against data corruption also apply to databases. Most database implementations do not allow multiple cluster members to concurrently serve the same underlying disk data. Notable exceptions to this include Oracle Parallel Server (currently being ported to Linux) and Informix Extended Parallel Server.

The upshot of all this is that the cluster implementation you choose must ensure that an individual file system or database can only be served by a single cluster member at any point in time—pretty simple, if you can find a cluster product that does this in all cases. Now, let us proceed to how this holds up under the four scenarios mentioned earlier.

Planned Maintenance

One of the greatest benefits of a high-availability cluster, which is ironically overlooked, is the ability to cleanly migrate services off a cluster member so you can perform routine maintenance without disrupting service to client systems. For example, this allows you to upgrade your software to the latest release or add memory to your system while keeping your site operational.

Virtually all high-availability cluster offerings accommodate planned maintenance.

System Crash

If you believe that a particular operating system is crash proof, give me a call and I'll sell you the Brooklyn Bridge to go along with that OS. Let's face it, system crashes are facts of life; it is merely a matter of minimizing their frequency. In response to a system crash, the other cluster members will conclude that a server has become nonresponsive and commence a take over of the services formerly provided by the failed node.

In the event of a system crash, virtually all fail-over cluster implementations will correctly takeover the services of a failed node. So far so good—it looks like just about any fail-over cluster product will suit you. Not so fast; the following points separate the credible offerings from the not so credible.

Communication Failure

Typical high-availability cluster implementations consist of a set of cluster members, each monitoring the other's health over a variety of “cluster interconnects”. Historically, many proprietary cluster vendors have depended on custom hardware for their cluster interconnects. While this provides a solid cluster implementation, by nature it tends to be very expensive and locks you into a single vendor. To provide a cost-effective alternative, other cluster implementations monitor system health over commonly available network interconnects (commonly Ethernet) and serial port connections. In these configurations, the cluster members periodically exchange messages, and based on the response (or lack thereof) conclude whether the other members are up or down. This exchange of system health-monitoring messages is commonly referred to as a “heartbeat”.

A common problem with “heartbeat” based clusters is communication partitions. This is when cluster members (or a set of members) are up but are unable to communicate with one another. Take, for example, the diagram in Figure 2 depicting a two-node cluster with an Ethernet and Serial connection between the nodes over which heartbeat messages are exchanged.

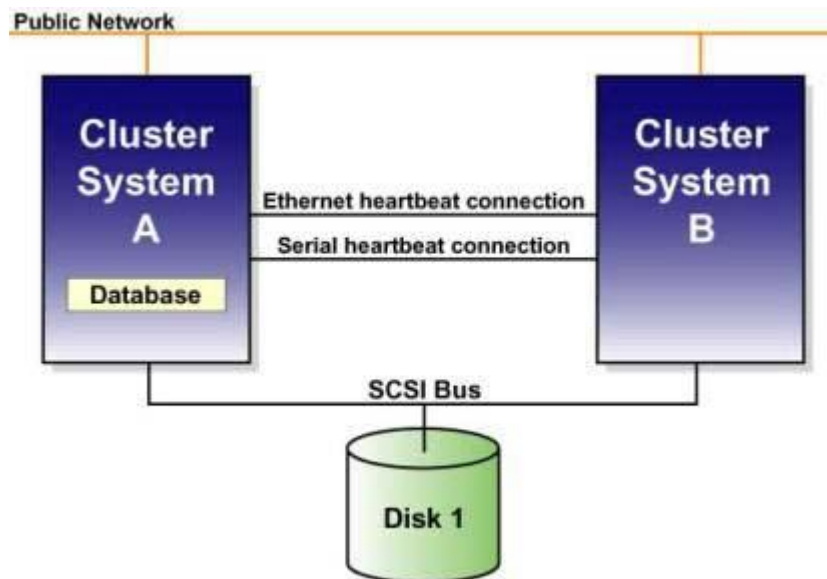


Figure 2. Two-Node Clustet with Ethernet and Serial Connections

Let us suppose you had set up your high-availability cluster and gone off to Las Vegas for the weekend, lulled into complacency with your company's new on-line ordering system deployed in this configuration. Further imagine the cleaning person accidentally knocking out the Ethernet connection with a broom. Now your two cluster members' cluster software running on each node must decide how to respond to this scenario in the interest of preserving high availability. Since the members can't communicate, they have to make the call in isolation. Here's some policy options commonly used by some cluster products:

- Pessimistic assumption—Node A knows that it's serving the database but is unaware of node B's state, so node A continues to serve the database. Node B can't communicate with node A and assumes that node A is down. Node B then commences serving the database resulting in two cluster members serving the same database further resulting in database corruption and possibly a system crash. (As weak as this sounds, this policy is employed in some offerings!)
- Optimistic assumption—After a site wide power outage, node A and node B both boot up at the same time. Neither node can ascertain the state of the other node and, just to be safe, they each assume that the other node is up so they do not start serving the database (to avoid data corruption). This results in a scenario where neither cluster member is serving the database. So much for spending money for a redundant cluster server! Actually, you're better off having your database unavailable than to have it corrupted. There are other failure scenarios that manifest themselves as a communication failure. For example:
 - An Ethernet adapter fails
 - The systems are connected to a common hub or switch that fails

- The Ethernet cable fails

To avoid these forms of communication partition, a common clustering practice is to employ multiple communication interconnects. For example, you may have the systems monitor each other's health by heartbeating over multiple Ethernets or a combination of both Ethernet and serial connections. Similarly, you may have each of the network connections go through separate hubs/switches or be point-to-point links.

Most cluster implementations allow you to configure multiple communication interconnects to eliminate the possibility of a communication partition. (If they do not, you should probably quickly move on to another vendor.)

System Hang

This is the most serious failure scenario that can confront any cluster implementation. If you didn't buy the bridge from me earlier, then perhaps I could interest you in one if you believe that systems never hang. This is another unfortunate fact of life in the computer biz. We've all seen systems mysteriously "lock up" where your only recourse is to reset or power cycle the system to get it responsive again. Fortunately, this is a relatively rare occurrence.

Just as mysteriously as computers can hang, they can also unhang. Surely you've been in scenarios where a system will "lock up" and then after a period of time become responsive again. This can happen on any operating system.

The pivotal question in evaluating cluster products is to understand how a cluster would respond in a hang/unhang scenario. Here's why the question is so important: in a hang scenario, node A becomes completely unresponsive. Suppose you learned your lesson in the prior section describing communication failures, and constructed a cluster with two Ethernet connections and a serial connection so that if any one of them failed, your cluster would still be operational. Well, in response to a system hang it wouldn't matter if you had 50 redundant connections—they all would fail to receive any response to system monitoring heartbeat requests. In response to this, node B would notice that node A has failed to respond to heartbeats over all three channels and conclude that node A has gone down. Following this, node B would mount the file systems or start up the databases formerly served by node A.

At this point, node A could become unhung and commence to update the file system. This results in a situation where two nodes are concurrently mounting and modifying the same file system, creating a data integrity violation.

This is the true litmus test of any cluster implementation. In order to protect against data integrity compromises (i.e., system crashes or invalid data) a cluster member must, before taking over services of a failed node, ensure that the failed node cannot modify the file system or database. This is commonly referred to as I/O Fencing or I/O Barrier.

In order to dodge this scrutiny, some proprietors of cluster products will dismiss the node hang/unhang scenario as an unlikely occurrence. Thankfully, in practice, system hang/unhang scenarios are infrequent. But, before dismissing this criteria entirely, remember it is your data and all the implications of having it corrupted are on the line.

Summary

If you care enough about your system's availability to warrant a cluster deployment, then it is crucial that you select a fail-over cluster implementation that ensures data integrity under each of the four failure scenarios. Keep in mind that the most valuable asset of your IT infrastructure is to have valid, accurate data. The cost of failing to ensure that data integrity is maintained is prolonged system downtime or loss of transactions, each of which can be catastrophic.

Tim Burke is Cluster Engineer at Mission Critical Linux, Inc. He can be reached at [http://www.burke@missioncriticallinux.com/](mailto:www.burke@missioncriticallinux.com/).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Monitoring Your UPS With `apcupsd`

Riccardo Facchetti

Issue #80, December 2000

Riccardo delves into the details of `apcupsd`, a program for monitoring and controlling APC UPSes.

Computers, like any other kind of electronic device, are very sensitive to utility power quality. According to American Power Conversion definitions (see Table 1) power anomalies can be classified as sags (better known as brownouts), blackouts, spikes, surges and noise. These power events may damage your electronic equipment, or cause loss of data if allowed to pass through the power supply to sensitive devices.

Table 1. Power Anomalies

The solution for all or part of these power anomalies is to connect an Uninterruptible Power Supply (UPS) to sensitive devices. Key to the UPS' function is a battery, used like a buffer, where power is accumulated when the utility power is normal, and released during low-voltage or blackout events. For high-voltage events, the UPS is usually equipped with filtering electronics that are capable of reducing the voltage. UPS output power is guaranteed to be sinusoidal alternating current kept within utility power specifications.

While a UPS alone can solve the immediate problem of utility power anomalies, when a blackout occurs the UPS' battery is used continuously, and its discharge time is faster in proportion to the load applied on the UPS output. Obviously, if a blackout is long enough, the battery charge will eventually be completely exhausted and cease to deliver power to the connected devices.

If a computer is connected to an exhausted UPS, it will suffer a system crash due to lack of power, as if the utility power suffered a blackout event. To overcome this problem, most UPSes on the market today have a built-in interface to communicate their status and to receive commands from a

computer. This interface has traditionally been a serial port, but UPSes are now being shipped with USB interfaces.

UPS Alerts

Because this article is about `apcupsd`, we will focus on American Power Conversion (APC) brand UPSes. UPSes from other manufacturers share the same general behavior but, to date, `apcupsd` does not communicate with non-APC UPSes.

A UPS protocol must accomplish two main tasks. The first and most important task is to asynchronously notify when power events happen. Second, and perhaps less important, is to allow the computer to query the UPS for status information.

Table 2 lists the UPS events that are sent by APC UPSes to the computer through the serial line. Looking at this table, we find that some of these events are paramount for computer integrity during utility power anomalies.

Table 2. UPS Events

Now let's see the most important events at work. In Figure 1, the state machine of a UPS is described in a simplified form. When the utility power is present, the UPS monitors it for failures. If a failure is detected, the UPS sends a Line Fail alert to the computer and switches itself to battery power. At this time, the computer is powered by the UPS' battery. The UPS starts monitoring for battery power failures. If the utility power returns, the UPS sends a Return from Line Fail alert and switches back to utility power. If the battery power goes down, below a security level, the UPS sends an About to Shutdown alert and, after a defined amount of time, switches itself off.

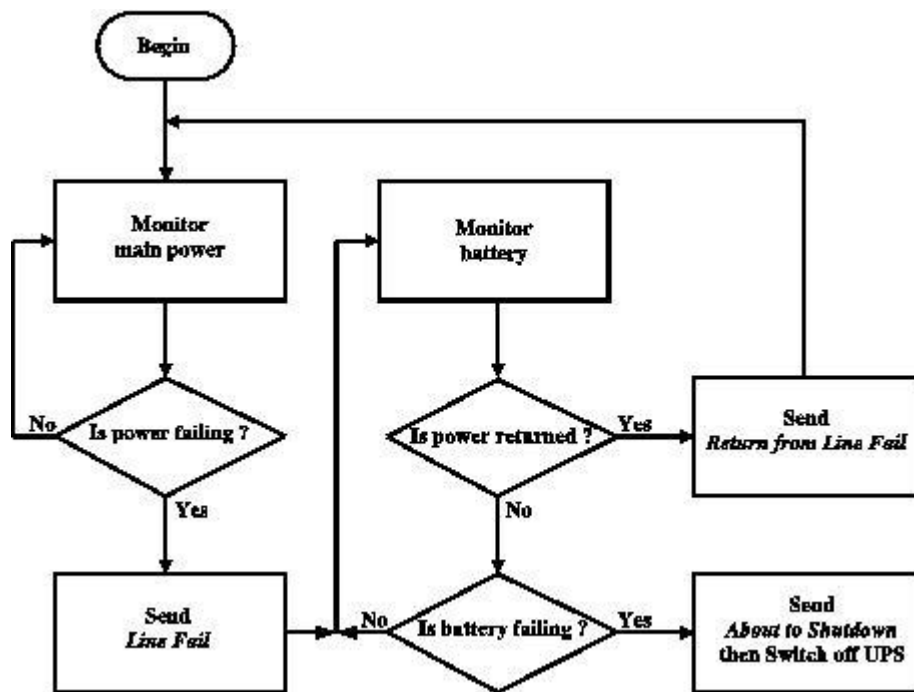


Figure 1. UPS State Machine

On the computer side, the alerts sent by the UPS are very important because they can be used to decide the actions to be executed during utility power anomalies. When the UPS switches to batteries, it could be wise to deny user logins. When the UPS is about to switch off, it is very important to shut down the computer operating system. The UPS will usually switch off after a defined delay that in some cases can even be configured, and this delay is usually long enough to allow the computer to shut down itself.

It is clear now why having an UPS does not mean, by itself, that our computer is safe. To make it really safe, the computer must constantly monitor the UPS and must be able to take actions in response to UPS alerts.

Monitoring Software: *apcupsd*

APC, in its catalog, offers a number of different monitoring software packages for its UPSes. Unfortunately, at the time we needed it in 1996 and 1997, no APC software for Linux was available. Nowadays there are at least four Linux products that can be downloaded from the APC ftp server including the one described here, *apcupsd*.

Brief History

In late 1996, Andre M. Hedrick started a Linux project called *apcupsd*, a *dæmon* whose purpose was to monitor APC UPSes for power alerts and to shut down the computer when needed. Development of this software was possible because of information gathered by the Internet, and by directly analyzing APC's protocols.

I joined the project in October 1997 after having bought a Smart UPS v/s 650 that I still have. However, due to old age, its battery is now dead. (Many thanks to APC who gave me a new Smart UPS 1400INET, allowing me to continue development of apcupsd.)

From the beginning, apcupsd had been licensed under the GNU Public License (GPL). The GPL license was chosen because apcupsd was intended to be software for anyone, with full sources distributed without charge, and with the best support its developers were able to give.

In mid 1998, legal issues raised by APC forced Andre to remove apcupsd from its public place and distribute it as a binary-only package, removing the GPL license from its source code. This caused a lot of discussion in the Internet community during the following year.

On April 7, 1999, APC withdrew its legal objections and started actively helping our team. This allowed apcupsd to return to its original license. Nowadays, APC monitors our project on our development mailing-list and gives help on technical issues.

In September 1999, Kern Sibbald joined the project. Being an experienced software developer, he quickly became one of the main apcupsd developers.

Theory of Operation

apcupsd's main task is to monitor the UPS status continuously, and take action based on information received from it. Of course in real life it is not so simple.

apcupsd must be run at startup time, when the operating system services are loaded: in fact, apcupsd is just another OS service. Typically, apcupsd is run as a dæmon (i.e. in the background) with root privileges, in order to be able to take the actions needed to keep the computer healthy. Usually, it is run by the system startup scripts when the system goes multiuser.

Because of its tight relationship with the OS, the source tree contains automated installation of startup and shutdown scripts. During the compilation stage the apcupsd, initialization and control scripts are customized for the local OS. Let's see which system files are modified, and which new files are installed by apcupsd's installation process in a SuSE Linux system.

apcupsd's main init script file is installed in:

```
/sbin/init.d/apcupsd
```

This script is responsible for starting up `apcupsd` during system startup and shutting down `apcupsd` during system shutdown. It is also symbolically linked to these paths:

```
/sbin/init.d/rc2.d/K20apcupsd
/sbin/init.d/rc2.d/S20apcupsd
/sbin/init.d/rc3.d/K20apcupsd
/sbin/init.d/rc3.d/S20apcupsd
```

They are present only on runlevel 2 and 3 because `apcupsd` is run only in multiuser runlevels; that means runlevel 2 or 3 on SuSE Linux OS.

To be able to shutdown the computer properly on power failures, `apcupsd` relies on its own service script located in:

```
/etc/apcupsd/apcontrol
```

and on a patched halt script. When `apcupsd` detects a situation that needs an emergency shutdown, it first creates two files called **`/etc/nologin`** and **`/etc/apcupsd/powerfail`**.

Then it initiates the system shutdown. `apcupsd` will be killed by the system during this phase. The no login file is needed to inhibit user logins during emergency, while the power-fail file is needed during system shutdown. It is used by **`/sbin/init.d/halt.local`** as shown in Listing 1. This script will be run during system shutdown after the main halt script is run. When shutting down a computer connected to a failing UPS, we must make sure that UPS power will be removed well after all processes have been killed, and the system disks have been unmounted. In order to do this, `apcupsd` installation modifies the `halt.local` file to execute the needed actions. If the power-fail file is present during a system shutdown, all processes are killed, then all the local disks are remounted read-only, and finally we send a power shutdown command to the UPS. The final effect is that, first of all, the system is put in a safe state, and only then is the UPS power is switched off.

The computer is switched off as well by lack of power, but its main switch button is left in the ON position. When utility power returns, the UPS will, after a delay to make sure the power is stable, switch on the output power, and the computer will be restarted automatically. No user intervention is needed during these operations.

Listing 1

Running `apcupsd`

When `apcupsd` is run, it first initializes the serial port and tries to determine if there is a UPS connected. Once it finds a UPS, it spawns a number of child

processes, each doing a specific task. This is a design decision, due to the number of things that apcupsd must do simultaneously.

The main apcupsd process creates an IPC-shared memory pool and an IPC semaphore, then forks the apcmain task, becoming a dæmon, before exiting. The job of apcmain is to fork all the other asynchronous tasks, wait for child termination, and listen for signals. It acts as a watchdog to avoid zombies or lost signals, and it does so simply by waiting with the wait() system call.

After becoming a dæmon, apcmain forks more processes depending on which configuration has been selected in the configuration file. Figure 2 shows the operations of apcupsd once started. Let's see them in detail.

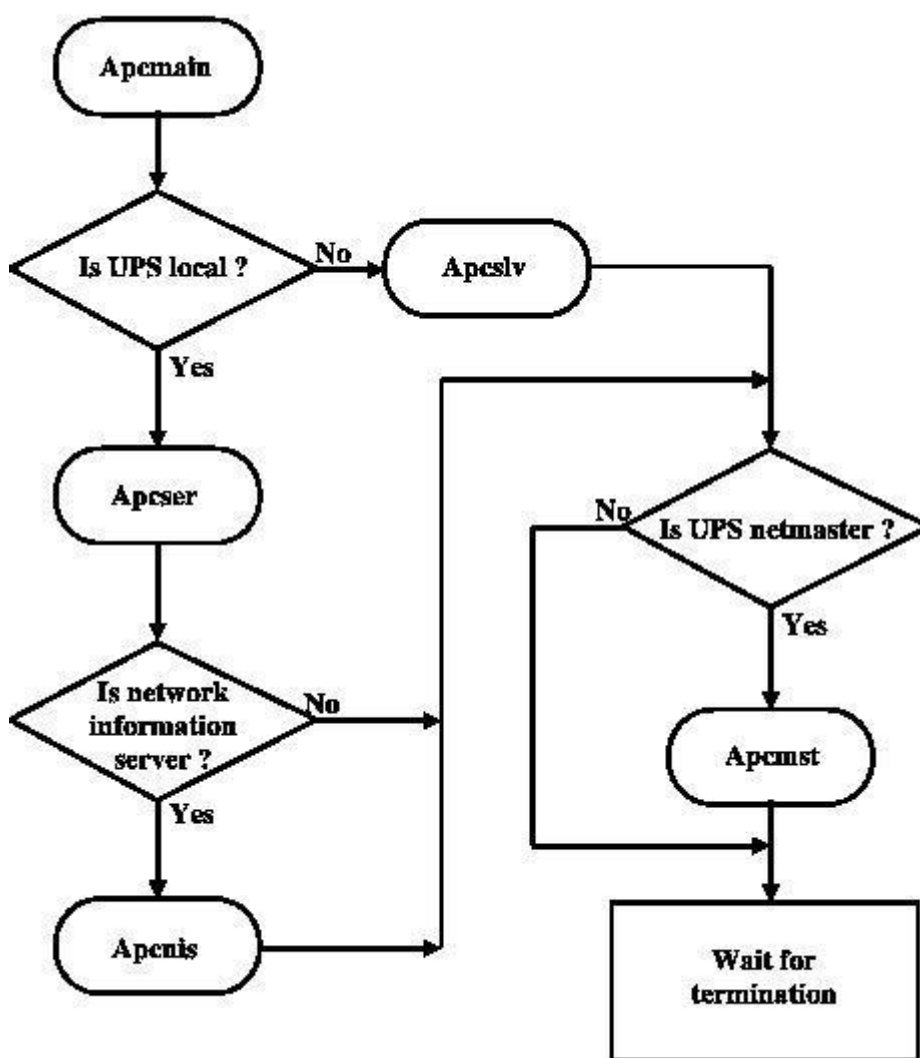


Figure 2. Startup Actions

If the UPS is local to the computer apcupsd is running on, the first task that is started is apcser. This task listens to the serial port for UPS alerts, and chats with the UPS to gather all the information needed to know the status of the UPS. In this configuration, apcser is the most important process running on the system, because, besides chatting with UPS, it executes all the actions needed

to run and shut down, if necessary, the computer safely during power failures. In addition, this task generates the reports that are written to system and private log files for history purposes.

Also, if `apcupsd` is asked to act as a Network Information Server, it will start **apcnis**. `apcnis` is a task that accepts connections from clients on the network and delivers information gathered from the UPS to them.

If the UPS protecting local computer's power utility is managed from a remote computer on the network, instead of starting `apcser`, the `apcmain` task will start the **apcsv** task. `apcsv` is a slave task that sits on a socket waiting for a connection from an `apcupsd` remote master. This process is the most important task of a networked `apcupsd` slave. When the network master sends the UPS' alerts to the slave, `apcsv` executes the same action routine executed by `apcserial`. This means that a slave that receives a battery-exhausted alert from its master will shut down the computer to ensure data and hardware safety.

Finally, if the UPS is local and it powers a utility line with other slave computers connected, `apcupsd` will start `apcmst`, the master process for network alert. `apcmst`'s task is to send, at defined intervals, the status of the UPS to every slave configured. Also it is very important for the slaves because if the UPS sends alerts to the computer, `apcmst` bypasses the time-outs and reports them immediately to the slaves. In this way we can make sure that the slaves receive power alerts almost immediately.

It is interesting to note that `apcupsd` processes communicate with each other through IPC shared memory areas and semaphores. This means that `apcupsd` will not run on a computer where System V IPC or some equivalent (such as memory mapped files) is not present.

Apupsd Configuration

Before running `apcupsd` it is necessary to configure the daemon to work as needed with the local hardware configuration and the desired behavior. In a standard installation, the configuration file is in

```
/etc/apcupsd/apcupsd.conf
```

This file is a plain ASCII file that contains all the configuration directives needed by `apcupsd`.

The directives must be properly configured before `apcupsd` can operate correctly. With these directives you specify the kind of cable, the model of UPS

connected to the computer, the device where the UPS is connected, how to operate on power failures, logging options and much more.

Stand-Alone Configuration

A typical stand-alone configuration includes a computer and a UPS connected to its serial port.

Listing 2

Listing 2 shows a configuration file for a stand-alone SmartUPS connected to the first serial port of the computer. On power failure, apcupsd will shut down the computer when the battery level falls below 5% of full charge or the UPS remaining run-time falls below three minutes, whichever happens first. apcupsd will send messages to user consoles every five minutes sending the first message one minute after the power failure occurs. apcupsd will not disallow user logins during a power failure. UPS events are logged in **/etc/apcupsd/apcupsd.events**. The UPS status can be read from our CGI interface or from **/etc/apcupsd/apcupsd.status**, which is updated every minute.

Networked Configuration

When you have a UPS that is powering more than one computer, and you want to protect every computer from power failures, you can configure apcupsd to work as network master/slave service. Listing 3 and 4 describe how to configure apcupsd as master and a slave in order to shut down both in case of a power failure.

Listing 3

Listing 4

The master apcupsd is configured as described above in stand-alone configuration, with the exception that the configuration directive NETACCESS is now true and the master will use port 6666 to communicate with slaves. The apcupsd master will connect to the slave at my.network.slave.com in order to communicate alerts and power events. The slave apcupsd shares the same configuration except that instead of using the serial port, it will listen to my.network.master.com to get all the information needed to operate correctly.

Advanced Configuration

apcupsd configuration is very flexible. But adding to this flexibility, you can also configure your control scripts to execute customized actions, and you can even configure your UPS eeprom to suit your needs.

Customizing apcupsd Actions

When apcupsd detects anomalies from your UPS device, it will make some decisions that usually result in one or more calls to the script located in `/etc/apcupsd/apccontrol`. The `apccontrol` file is a shell script that takes the first argument that apcupsd passes to it to do actions. These actions are set up by default to sane behavior for all possible situations apcupsd is likely to detect from the UPS. Nevertheless, you can change the `apccontrol` behavior for every action. To do so, simply create a file with the same name as the action you want to customize, which is passed as the first argument (`argv[1]`, or `$1` for shell scripts). Place your script in the `/etc/apcupsd/` directory. The arguments that `apccontrol` can recognize are shown in Table 3.

Table 3. Arguments apccontrol Recognizes

Listing 5

If, for example, you want to write your own routine for the on-battery action, you can write your own shell script, as shown in Listing 5, called `onbattery` and put it in the `/etc/apcupsd/` directory. Doing so will run the customized script before the default action. In case you don't want the default action to be taken, terminate your customized script with an exit code of 99. If you want to write customized scripts to replace the default behavior, you are encouraged to edit the `apccontrol` script and at least mimic its behavior in your own script. Please be aware that writing faulty scripts may cause your system to crash during power failures.

Configuring the UPS eeprom

In SmartUPSes there are at least 12 different values stored in the eeprom that determine how the UPS reacts to various conditions such as high line voltage, low line voltage, power down grace periods, etc.

In general, for the moment, it is not recommended to change eeprom values unless absolutely necessary. There have been several reported cases of problems setting the Low Transfer Voltage. Consequently, if at all possible, do not attempt to change this value.

If, despite these warnings, eeprom values must be changed, we recommend to connect your UPS to a Windows machine running PowerChute and making the changes.

In the absence of alternatives, eeprom values can be changed also with apcupsd. Before doing so, it is recommended that you make a printed copy of UPS parameters as they are before any eeprom changes, so that they can be

checked against the changes made. This can be done by printing a copy of the output from `apcaccess status` and also of the output from `apcaccess eprom`.

Once this is done, choose which eeprom values you want to change. Choose the new values from the list provided by `apcaccess eprom`. For the battery date, and the UPS name, you can use any eight characters.

To make the eeprom changes with `apcupsd` you must first stop the `apcupsd` daemon. See "Stopping Apcupsd" in the appropriate section of `apcupsd` manual. Then edit the appropriate configuration directive in `/etc/apcupsd/apcupsd.conf`.

It is recommended to change one eeprom value at a time, by defining only one configuration directive at a time. After each change, check that everything is okay before proceeding to the next value you wish to change.

To actually change the eeprom, as root with the `apcupsd` daemon stopped, enter:

```
apcupsd -c
```

When it has completed the reprogramming of the eeprom, it will print the new STATUS report. Check that you got the expected results before continuing.

Apcupsd Security Issues

apcupsd runs as root. This needs to be considered when installing the software. It needs access to the serial port, System V IPC services and the shutdown program.

If you have switched on the `NETSERVER` directive in your `apcupsd.conf` file, be aware that anyone on the network can read the status of your UPS. This may or may not be a problem. If you don't consider this information privileged, there is little risk. In addition, if you have a firewall between your servers and the Internet, intruders will not have access to your UPS information. Additionally, you can restrict access to your `apcupsd` server by using the `INETD` services and using access control lists with a TCP wrapper.

If you are running master/slave networking, with a single UPS powering multiple machines, be aware that it is possible for someone to simulate the master and send a shutdown request to all of your slaves. The slaves do check that the network address of the machine claiming to be the master is the same as the address returned by DNS corresponding to the name of the master as specified in your configuration file.

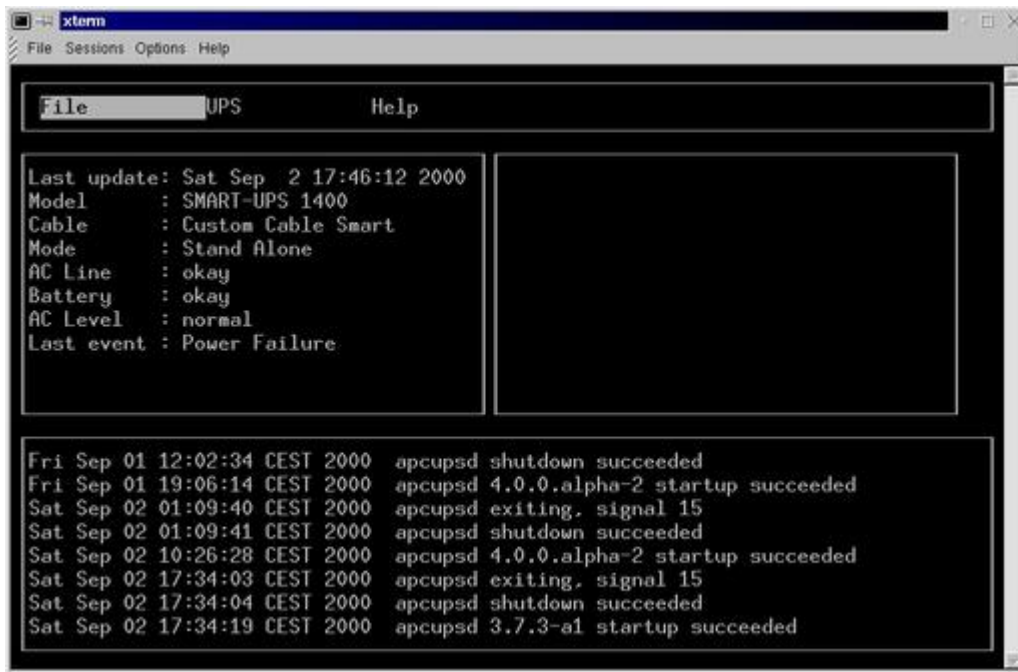
apcupsd's Clients

To ease UPS monitoring, apcupsd offers quite a lot of client facilities. We have already seen apcaccess, the output of which can be seen in Listing 6. apcaccess is the main client tool for monitoring UPS status.

Listing 6

Another console-based client is powerflute that can be used to monitor the UPS status continuously as seen in Figure 3.

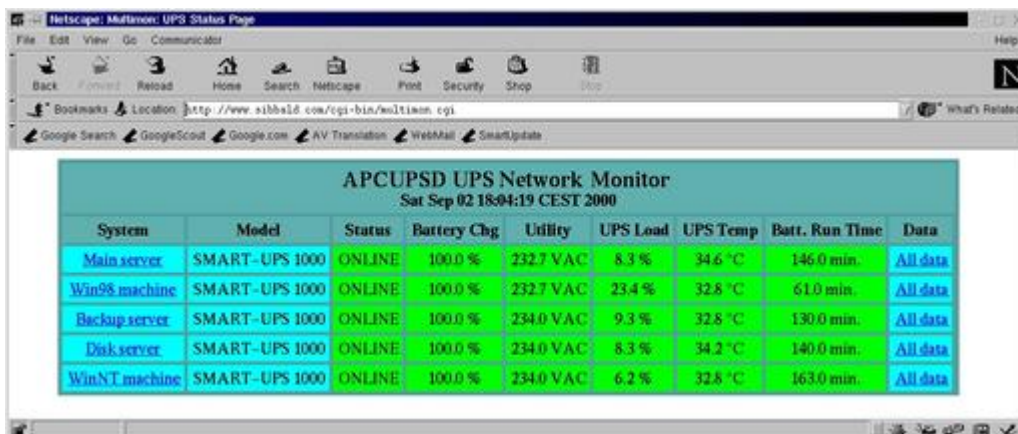
In Figure 4 another useful client for networked apcupsd is shown. It is the CGI interface that was integrated into apcupse and enhanced by Kern. If you want to be able to view UPS status from the Web, this is the best choice. Figure 5 shows the status of the UPSes connected to Kern's main server (www.sibbald.com/cgi-bin/multimon.cgi).



```
File Sessions Options Help
File UPS Help
Last update: Sat Sep 2 17:46:12 2000
Model      : SMART-UPS 1400
Cable      : Custom Cable Smart
Mode       : Stand Alone
AC Line    : okay
Battery    : okay
AC Level   : normal
Last event : Power Failure

Fri Sep 01 12:02:34 CEST 2000  apcupsd shutdown succeeded
Fri Sep 01 19:06:14 CEST 2000  apcupsd 4.0.0.alpha-2 startup succeeded
Sat Sep 02 01:09:40 CEST 2000  apcupsd exiting, signal 15
Sat Sep 02 01:09:41 CEST 2000  apcupsd shutdown succeeded
Sat Sep 02 10:26:28 CEST 2000  apcupsd 4.0.0.alpha-2 startup succeeded
Sat Sep 02 17:34:03 CEST 2000  apcupsd exiting, signal 15
Sat Sep 02 17:34:04 CEST 2000  apcupsd shutdown succeeded
Sat Sep 02 17:34:19 CEST 2000  apcupsd 3.7.3-a1 startup succeeded
```

Figure 3. Powerflute



System	Model	Status	Battery Chg	Utility	UPS Load	UPS Temp	Batt. Run Time	Data
Main server	SMART-UPS 1000	ONLINE	100.0 %	232.7 VAC	8.3 %	34.6 °C	146.0 min.	All data
Win98 machine	SMART-UPS 1000	ONLINE	100.0 %	232.7 VAC	23.4 %	32.8 °C	61.0 min.	All data
Backup server	SMART-UPS 1000	ONLINE	100.0 %	234.0 VAC	9.3 %	32.8 °C	130.0 min.	All data
Disk server	SMART-UPS 1000	ONLINE	100.0 %	234.0 VAC	8.3 %	34.2 °C	140.0 min.	All data
WinNT machine	SMART-UPS 1000	ONLINE	100.0 %	234.0 VAC	6.2 %	32.8 °C	163.0 min.	All data

Figure 4. CGI Interface: Overview

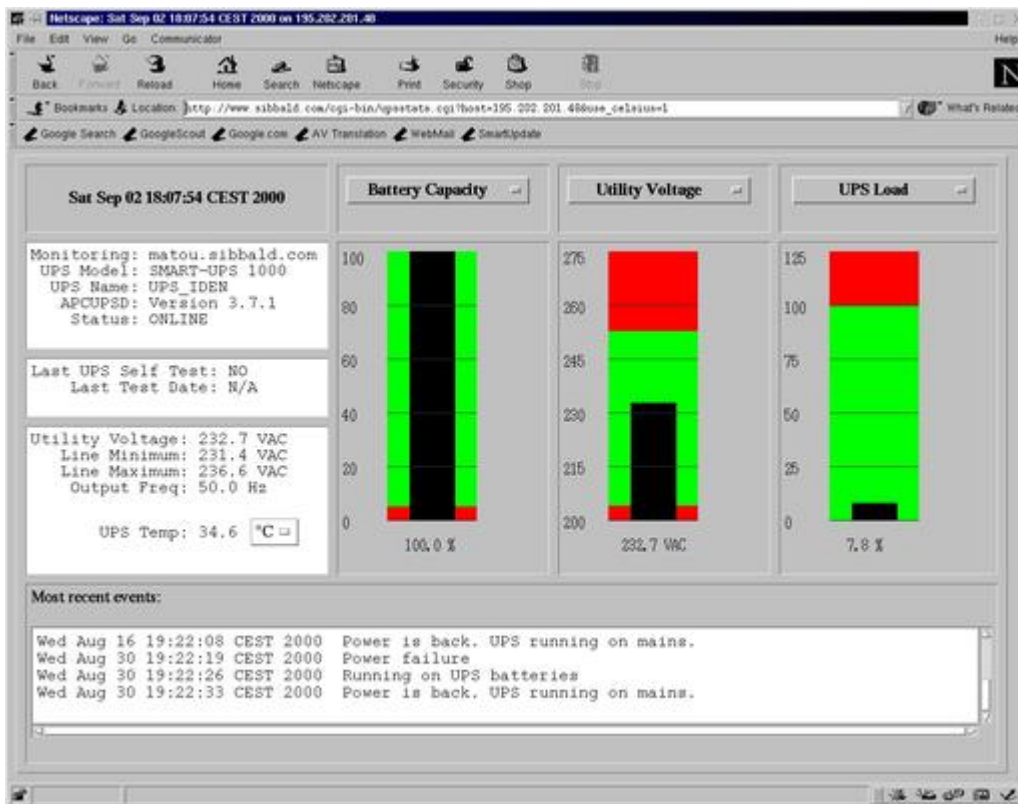


Figure 5. CGI Interface: Details on One Host

apcupsd is being ported to Win32 (9x and NT) by Kern. At the time of writing, Kern had produced a beta version of apcupsd for Windows. In Figures 6 and 7 apcupsd for Win32 status client is shown.

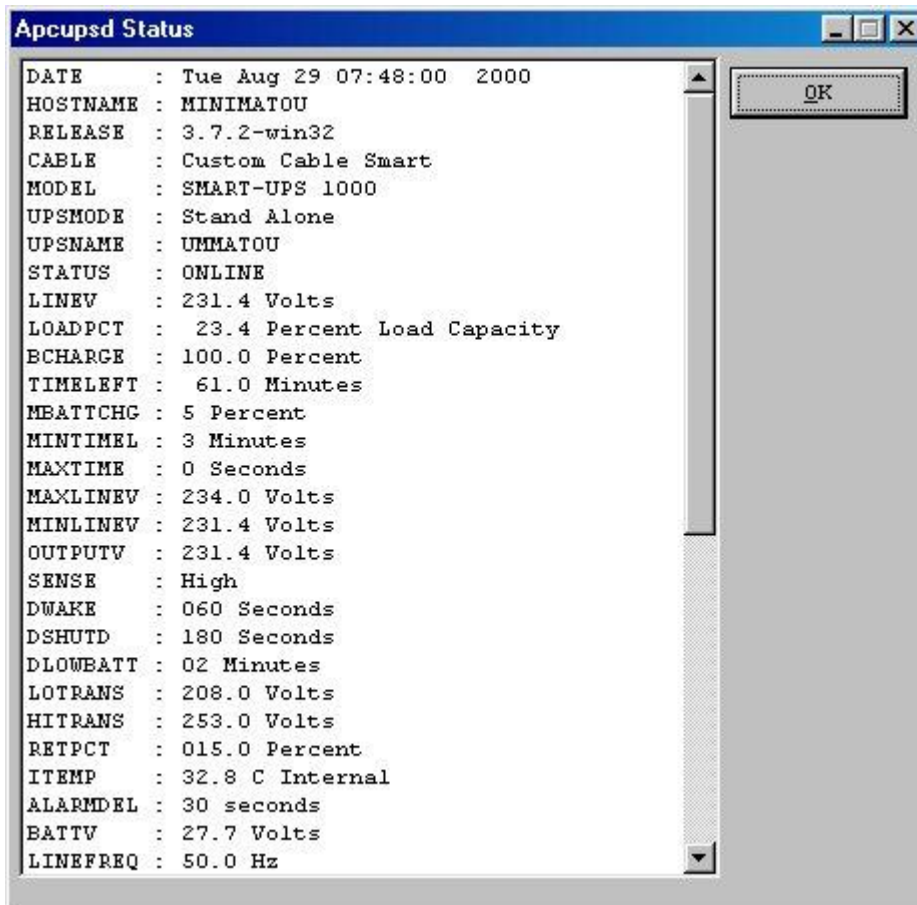


Figure 6. apcupsd for Win32

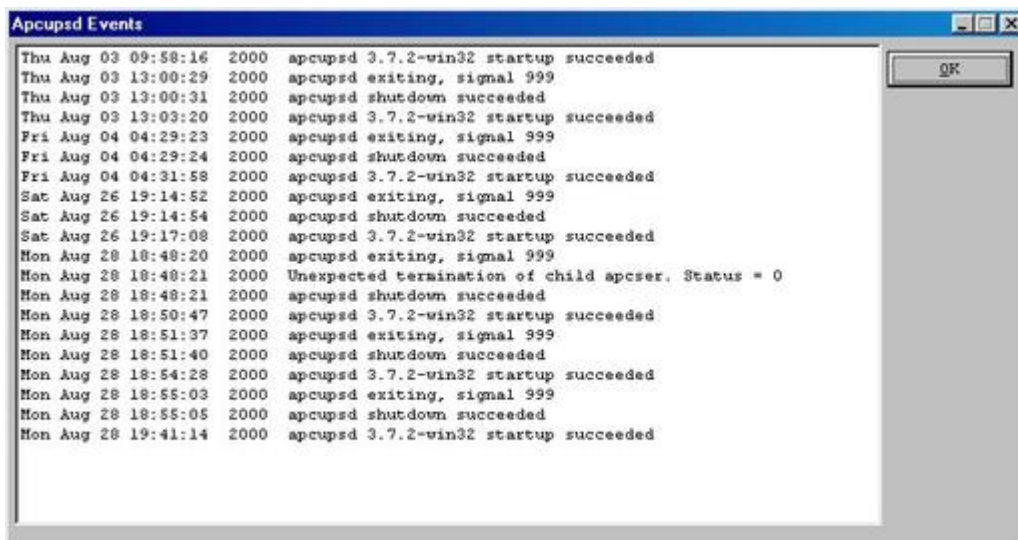


Figure 7. apcupsd for Win32

What's Next

apcupsd is still growing. The next version of apcupsd will be a major version change and it will contain the following new main features:

- Multiple UPS control apcupsd will be able to control more than one UPS connected to the same computer.

- Network code rewrite `apcupsd` will be a true network `dæmon` with strong security features integrated into the code.
- Complete Win32 support.
- Ability to perform UPS tests from within client programs.
- If time permits, another feature, that now is more a dream rather than a development target, is to support other manufacturer's UPSes.



Riccardo Facchetti (riccardo@master.oasi.gpa.it) is a research engineer for a scientific instrumentation firm in Milano, Italy. When not sitting in front of his computer, he travels worldwide.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

PVFS: A Parallel Virtual File System for Linux Clusters

Ibrahim F. Haddad

Issue #80, December 2000

An introduction to the Parallel Virtual File System and a look at how one company installed and tested it.

Using networked file systems is a common method for sharing disk space on UNIX-like systems, including Linux. Sun was the first to embrace this technology by introducing the Network File System (NFS), which provides file sharing via the network. NFS is a client/server system that allows users to view, store and update files on remote computers as though they were on the user's own computer. NFS has since become the standard for file sharing in the UNIX community. Its protocol uses the Remote Procedure Call method of communication between computers.

Using NFS, the user or a system administrator can mount all or a portion of a file system. The portion of your file system that is mounted can be accessed with whatever privileges accompany your access to each file (read-only or read-write).

As the popularity and utility of this type of system have grown, more networked file systems have appeared. These new systems include advances in reliability, security, scalability and speed.

As part of my responsibilities in the Systems Research Department at Ericsson Research Canada, I evaluated Linux-networked file systems to decide what networked file system(s) to adopt for our Linux Clusters. At this stage, we are experimenting with Linux and clustering technologies and trying to build a Linux cluster that provides extremely high scalability and high availability.

An important factor in building such a system is the choice of the networked file system(s) with which it will be used. Among the tested file systems were Coda, Intermezzo, Global File System (GFS), MOSIX File System (MFS) and the Parallel Virtual File System (PVFS). After considering these and other options, the

decision was made to adopt PVFS as the networked file system for our test Linux cluster. We are also using the MOSIX file system as part of the MOSIX package (see Resources) that enhances the Linux kernel with cluster-computing capabilities.

In this article, we cover our initial experiences with the PVFS system. We first discuss the design of the PVFS system in order to help familiarize readers with the terminology and components of PVFS. Next, we cover installation and configuration on the 7 CPU Linux Cluster at the Ericsson Systems Research Lab in Montréal. Finally, we discuss the strengths and weaknesses of the PVFS system in order to help others decide if PVFS is right for them.

PVFS Overview and Goals

Linux cluster technology has matured and undergone many improvements in the last few years. Commodity hardware speed has increased, and parallel software has become more advanced. Input/Output (I/O) support has traditionally lagged behind computational advances, however. This limits the performance of applications that process large amounts of data or rely on out-of-core computation.

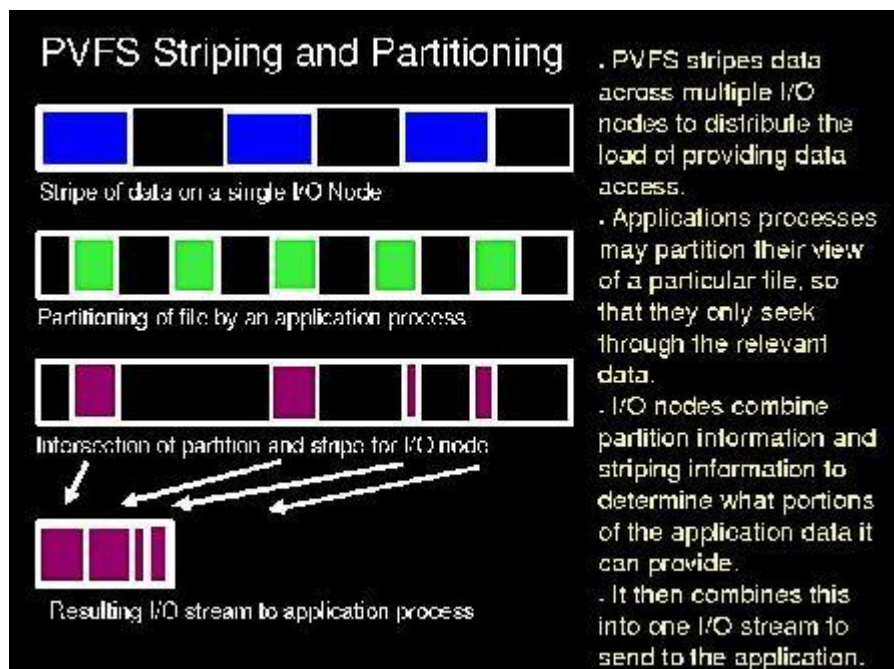


Figure 1. PVFS System Architecture

PVFS was constructed with two main objectives. The foremost is to provide a platform for further research into parallel file systems on Linux clusters. The

second objective is to meet the growing need for a high-performance parallel file system for such clusters. PVFS goals are to:

- Provide high bandwidth for concurrent read/write operations from multiple processes to a common file
- Support multiple APIs, including a native PVFS API, the UNIX/POSIX I/O API, as well as MPI-IO (through ROMIO)
- Support Common Unix utilities such as **ls**, **cp** and **rm** for PVFS files
- Provide a mechanism for applications developed for the UNIX I/O API to work with PVFS without recompiling
- Offer robustness and scalability
- Be easy to install and use

PVFS Node Types

One machine, or node, in a cluster, can play a number of roles in the PVFS system. A node can be thought of as being one or more of three different types: compute, I/O or management. Typically, a single node will serve as a management node, while a subset of the nodes will be compute nodes and another subset will serve as I/O nodes. It is also possible to use all nodes as both I/O and compute nodes.

PVFS exists as a set of dæmons and a library of calls to access the file system. There are two types of dæmons, management and I/O. Typically, a single-management dæmon runs on the management node and a number of I/O dæmons run on the I/O nodes. The library of calls is used by applications running on compute nodes, or client nodes, in order to communicate with both the management dæmon and the I/O dæmons.

Management and I/O Dæmons

Management dæmons, or managers, have two responsibilities: validating permission to access files and maintaining metadata on PVFS files. All of these tasks revolve around the access of metadata files. Only one management dæmon is needed to perform these operations for a file system and a single-management dæmon can manage multiple file systems. The manager is also responsible for maintaining the file system directory hierarchy. Applications running on compute nodes communicate with the manager when performing activities such as listing directory contents, opening files and removing files.

On the other hand, I/O dæmons serve the single purpose of accessing PVFS file data and correlating data transfer between themselves and applications. Direct connections are established between applications and I/O servers in order to directly exchange data during read and write operations.

Accesses to Client Nodes

There are several options for providing PVFS access to the client nodes. First, there is a shared, or static, library that can be used to interact with the file system using its native interface. This requires writing applications specifically to use functions such as **pvfs_open**, however. As an alternative, there are two access methods that provide transparent access. The preferred method is to use the PVFS kernel module, which allows full access through the Linux VFS mechanism. This loadable module allows the user to mount PVFS just like any other traditional file system. Another option is to use a set of C library wrappers that are provided with PVFS. These wrappers directly trap calls to functions such as `open` and `close` before they reach the kernel level. This provides higher performance but with disadvantages in that the compatibility is incomplete, and the wrappers work only with certain supported versions of **glibc**.

A final option is to use the MPI-IO interface, which is part of the MPI-2 standard for message passing in parallel applications. The MPI-IO interface for PVFS is provided through the ROMIO MPI-IO implementation (see Resources) and allows MPI applications to take advantage of the features of MPI-IO when accessing PVFS. It also ensures that the MPI code will be compatible with other ROMIO-supported parallel file systems.

Installation Environment

The test system at Ericsson Montréal started as a cluster of seven diskless Pentium grade CPUs with 256MB of RAM each. These CPUs first boot using a minimal kernel written on flash using a tool provided by the manufacturer. They then they get their IP address and download a RAM disk from a Linux box acting as both a DHCP and a TFTP server. This same machine also acts as an NFS server for the CPUs, providing a shared disk space.

When we decided to experiment with PVFS, we needed some PCs with disks to act as I/O nodes and one PC to be the management node. We added one machine, PC1, to be the management node and three machines, PC2, PC3 and PC4, with a total disk space of 35GB, to be the I/O nodes. The new map of the cluster became:

- Seven Diskless Client CPUs
- One Management Node
- Three I/O Nodes

Installation Steps

While PVFS developers provide RPMs for all types of nodes, we chose to recompile the source in order to optimize installation on the diskless clients.

This went over without a hitch using the PVFS tarball package. For the manager and I/O nodes, we used the relevant RPM packages. The manager and I/O nodes are using the Red Hat 6.2 distribution and the 2.2.14-5.0 kernel. The diskless CPUs run a customized minimal version of the 2.2.14-5.0 kernel.

Setting up the Manager

The first step towards setting up the PVFS manager is to download the PVFS manager RPM package and install it. PVFS will be installed by default under /usr/pvfs. Once the automatic installation is done, it is necessary to create the configuration files. PVFS requires two configuration files in order to operate: "pvfsdir", which describes the directory to PVFS and "iodtab", which describes the location of I/O daemons. These files are created by running the **mkiodtab** script (as root):

```
[root@pc1 /root]# /usr/pvfs/bin/mkiodtab
```

See Listing 1 for the **iodtab** setup for the Parallel Virtual File System. It will also make the .pvfsdir file in the root directory.

Listing 1

When we ran mkiodtab on the manager, PC1, it complained that it did not find the I/O nodes. It turned out to be that we had forgotten to include entries of my I/O nodes in /etc/hosts. We updated the /etc/hosts file and reran mkiodtab; everything went okay. mkiodtab created a file called "iodtab" under /pvfs. This file contained the list of my I/O nodes. It looked like the following:

```
-----/pvfs/.iodtab-----  
pc2: 7000  
pc3: 7000  
pc4: 7000  
-----
```

The default port number used by I/O daemon software to allow clients to connect to it over the network is 7,000.

After running mkiodtab, we did the following to start PC1 as the PVFS manager:

```
Start the manager d&aelig:  
% /usr/pvfs/bin/mgr  
% /usr/pvfs/bin/enblemgr
```

Running **enblemgr** on the management node ensures that the next time the machine is booted the daemons will be automatically started, so that it doesn't need to be started manually after rebooting. The enblemgr command only needs to be run once to set up the appropriate links.

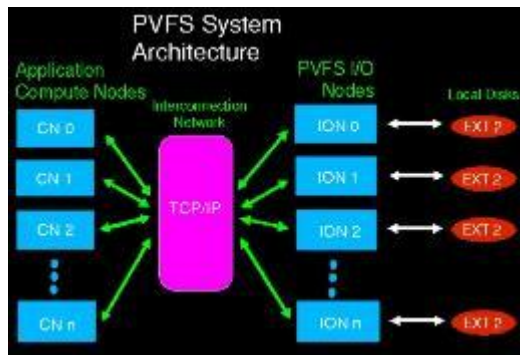


Figure 2. PVFS Striping and Partitioning

Setting up the I/O Nodes

Installation of I/O nodes is equally simple. First, we installed the RPM, then started each I/O daemon as follows:

```
% /usr/pvfs/bin/iod
% /usr/pvfs/bin/enableiod
```

Running **enableiod** on the I/O nodes ensures that the next time the machines are booted, the daemons will be started automatically. The **enableiod** command only needs to be run once to set up the appropriate links.

The I/O daemons rely on a configuration file, `/etc/iod.conf`, to tell them where to store data. This file is automatically created by the RPM and directs the I/O daemons to store data in a directory called `/pvfs_data`. We created this directory on each of the I/O nodes with:

```
% mkdir /pvfs_data
```

Setting up the Diskless CPUs as Compute Nodes

The installation of the client CPUs was more delicate since, as mentioned above, we needed to minimize the installation to use less space on the RAM disk. The minimal set of installation files that we used for the client nodes were:

```
----- List of files installed on the Compute Nodes -----
/etc/pvfstab
/usr/local/pvfs/pvfsd
/usr/local/pvfs/pvfs.o
/usr/local/pvfs/mount.pvfs
/usr/local/pvfs/libpvfs.so.1.4
-----
```

The `/etc/pvfstab` is used by the compute nodes to determine the locations of the manager and the PVFS files. Its format is very similar to the `/etc/fstab` file. For our setup, the `/etc/pvfstab` file looked like the following:

```
-----/etc/pvfstab-----
pc1:/pvfs      /pvfs pvfs port=3000 0 0
-----
```

This configuration file specified that:

- The management node is PC1
- The directory where the manager is storing metadata is /pvfs
- The PVFS file system is mounted on /pvfs on the client
- The port on which the manager is listening is 3000

The PVFS daemon is /usr/pvfs/bin/pvfsd. It works in conjunction with the kernel module to provide communication with the file system through the kernel. The daemon uses the same PVFS library calls that a custom user application would, but it translates them into a form recognized by the kernel module so that it is hidden from applications not specifically compiled for PVFS. This is similar to the approach used by the Coda file system in which a user-level daemon cooperates with the Coda kernel code to access the file system (see Resources).

/usr/pvfs/bin/mount.pvfs is the special mount command supplied with PVFS. The client CPUs use it to mount the PVFS file system on a local directory. For these CPUs, we have created a small shell script, /etc/rc.d/rc.pvfs, that is executed when the CPUs are started to ensure that they start up automatically as PVFS compute nodes without any manual intervention. The content of rc.pvfs is the following:

```
-----/etc/rc.d/rc.pvfs-----  
#!/bin/sh  
/bin/mknod /dev/pvfsd c 60 0  
/sbin/insmod /usr/pvfs/bin/pvfs.o  
/usr/pvfs/bin/pvfsd  
/usr/pvfs/bin/mount.pvfs pc1:/pvfs /mnt/pvfs  
-----
```

The script creates a node in /dev that will be used by **pvfsd**. It loads the PVFS module, starts the PVFS daemon and mounts the PVFS file system locally under /mnt/pvfs.

As noted earlier, any I/O node or management node can also serve as a compute node. To enable this, we simply installed the PVFS client RPM on each I/O node, as we are not worried about conserving disk space on the I/O nodes. The /etc/pvfstab and /etc/rc.d/rc.pvfs were then set up to be identical to those used on the diskless clients. Now, both the diskless clients and the I/O nodes can access the file system in the same manner.

Testing the Installation

After completing these installation steps we were able to copy and access files within the PVFS file system from all of the machines. The RAM disk that was installed on the CPUs included as part of the setup the Apache Web Server and Real Server, a video streaming server from Real Networks. We used WebBench (from ZDNet.com) to generate web traffic to the CPUs and changed the

configurations for both Apache and Real Server to place the default root document inside the PVFS file system. This scenario allowed every CPU to run as a stand-alone web server with its own IP address and serve multimedia requests using Real Server. This allowed hosting web files, including big files such as mp3 and rm files, from within the PVFS file system.

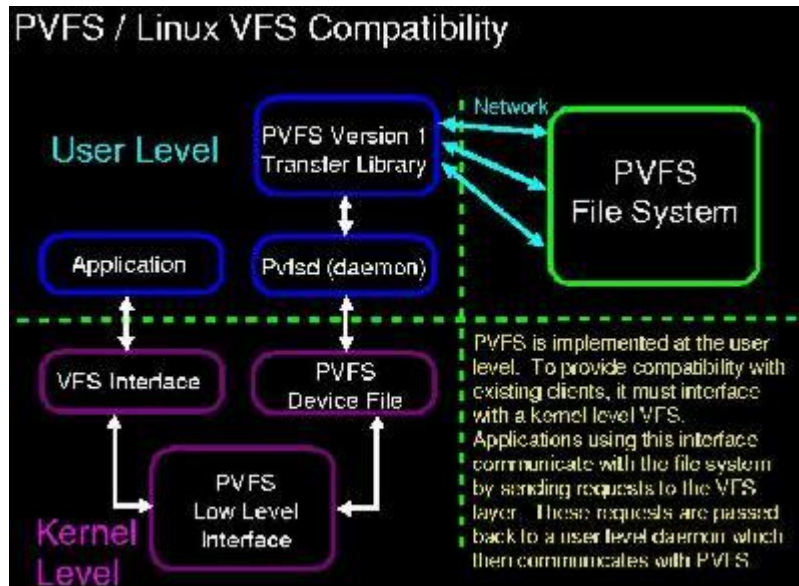


Figure 3. PVFS/Linux Compatibility

Coexistence with Other File Systems

As certain types of applications perform better on certain file systems due to their access patterns, it is important to us that PVFS be able to coexist with other file systems. The PVFS system had no problem operating in the same environment with JFS, NFS, SFS and even the MOSIX file system. This neat setup served large I/O requests such as mp3 files on the Web. The MOSIX file system was used by MOSIX to migrate processes on the cluster to the most appropriate CPU at the time.

Typically, PVFS sits on top of the ext2 file system. However, the next generation of Linux file systems will be journaling file systems. This protects against hardware or software failures by producing a log of changes-in-progress that records changes to files and directories in a standard way to a separate location, possibly a separate disk. If the primary file system crashes, the journal can be used to undo any partially completed tasks that would leave the file system in an inconsistent state.

The next step in this perspective is to see how well PVFS performs on top of ext3 and GFS as native file systems. This is left for experimentation on the new cluster (see below).

Scaling up the Installation

Another important factor in choosing a file system such as PVFS is to check how well it can scale up with more client and I/O nodes. Having one nonredundant management node might seem like an inherent bottleneck. However, the manager is not involved in any read or write operations, as these are handled directly between clients and I/O daemons. It is only when many files are created, removed, opened or closed in rapid succession that the manager is heavily loaded.

We wish to test the scalability of this configuration, however so the upcoming PVFS installation will be on a cluster consisting of 16 PIII 500 MHz CPUs with 512MB RAM each. Eight of the CPUs have 18GB SCSI disks each with a mix of RAID 1 and RAID 5 setup. The projected installation will have one Manager, seven I/O nodes and 14 clients (I/O nodes are also clients). This cluster will allow us to better understand how PVFS will scale for our applications and will additionally allow us to compare PVFS performance with the performance of alternative file systems, such as NFS, for systems of this size. Tests of PVFS on other clusters have shown it to be scalable to systems of more than 64 nodes for other workloads. (See "PVFS: A Parallel File System for Linux Clusters" at PVFS's web site in Resources.)

PVFS Advantages

PVFS is easy to install and configure. It comes with an installation guide that walks administrators through the installation procedure. It provides high performance for I/O intensive parallel or distributed applications. It is also compatible with existing applications so that you don't have to modify them to run with PVFS. PVFS is well supported by the developers through mailing lists.

PVFS Vulnerabilities

PVFS currently contains neither data redundancy nor recovery from node failure. There may also be potential bottlenecks at the manager level as the number of client nodes increases. PVFS endures restrictions introduced by TCP/IP dependence, such as limits on the number of simultaneous open system sockets and network traffic overhead inherent in the TCP/IP protocol. As for security, PVFS provides a rather unsophisticated security model, which is intended for protected cluster networks. Also, for the time being, PVFS is limited to the traditional Linux two gigabyte file size.

Types of applications that benefit the most from PVFS are:

- Applications requiring large I/O, such as scientific computation or streaming media processing

- Parallel applications because the bandwidth increases as multiple clients access data simultaneously

Types of applications for which PVFS is poorly suited:

- Applications requiring many small, low-latency requests, such as static html pages (there is quite a bit of overhead in network traffic for multiple small file requests)
- Applications requiring long-term storage or failover ability—PVFS does not provide redundancy on its own

Writing PVFS Programs

As noted earlier, existing applications can access PVFS through either the kernel module or the library wrapper interface. This does not require any modification from the user's point of view. However, to obtain the best performance for parallel applications, developers must modify their applications to use a more sophisticated interface. There are two options for this approach as well. The first is to use the native PVFS library calls. This interface allows advanced options, such as specifying file striping and number of I/O nodes. It also lets each process define a “partition” or particular view of the file so that contiguous read operations access only specific offsets within the file (see Figures 2 and 3). Documentation for this is available in the PVFS user's guide.

MPI-IO is the preferred option for writing PVFS programs. It layers additional functionality on top of PVFS, including collective file operations and two phase I/O. This interface is documented as part of the MPI-2 standard.

Security Issues

As mentioned before, PVFS does not implement many security features at this time. It is primarily intended for use on private cluster networks that can insure trusted clients. There is no restriction on client connections, nor is there any encryption or keys used to verify user authenticity. Client nodes are generally trusted to provide accurate UID information for use in file permissions and ownership checks, just as in NFS.

The Future of PVFS

There are many changes and advances in store for PVFS. The existing generation of PVFS is undergoing modifications and testing to support a higher degree of scalability. These mostly address issues with supporting large numbers of TCP/IP sockets. They will also resolve the issues inherent in supporting 64-bit (greater than two-gigabyte) file sizes and offsets. This will

allow PVFS to scale to the current size of large-scale clusters that utilize hundreds or thousands of nodes.

PVFS as a whole is undergoing a full redesign at the same time. This will result in a complete rewrite of PVFS that incorporates new technology and lessons learned from the previous implementation. This version of PVFS will not be available for quite some time but is already in active development.

Some of the features that will be supported in the next generation are:

- Reactive scheduling that allows PVFS to adapt policies based on system state and application load
- Modular support for a variety of networking systems, so that the file system is no longer bound to TCP/IP but can take advantage of more advanced messaging protocols as they become available
- Modular support for a variety of storage methods to allow I/O dæmons to access local data through various methods, such as raw I/O or asynchronous I/O
- Multiple manager support
- Redundancy of both I/O data and metadata in case of system failure
- Improvements in the UNIX compatibility layer
- More advanced options for data distribution as well as data representation

After evaluating several distributed file systems, we chose to use PVFS for applications that require intensive I/O. PVFS, in its current state, does not provide any redundancy or high security features. However, the research is still ongoing, and we have high hopes in this regard. We believe that if PVFS were to provide access security, data redundancy and management node redundancy, then it would be more suitable for adoption as part of a highly scalable, reliable and fault-tolerant Linux cluster. As it stands now, however, it is more suited for application domains (such as scientific computing) in which optimal performance is paramount rather than high availability.

I had a pleasant experience with PVFS and with the developers who provided a lot of help as we achieved the above setup and contributed much to the writing of this article.

If you are interested in distributed file systems and you need support for high-performance I/O, I highly recommend that you try out PVFS. PVFS is freely available under the GPL and can be downloaded from Clemson University's web site (see Resources).

Acknowledgments

Primary PVFS developers: Robert Ross, Mathematics and Computer Science Division, Argonne National Lab. Philip Carns and Walt Ligon, Parallel Architecture Research Lab, Clemson University, with support from NASA's Goddard Space Flight Center as well as Argonne National Lab. **Ericsson Research Canada:** The Systems Research Department at Ericsson Research Canada for providing the facilities and equipment as well as approving the publication of this article.

Contributors

Philip Carns (pcarns@hubcap.clemson.edu) is a graduate student at the Parallel Architecture Research Lab at Clemson University. **Robert Ross** (ross@mcs.anl.gov) is employed at the Argonne National Laboratory by the Mathematics and Computer Science Division. He will receive his Doctoral degree in Computer Engineering from Clemson University in December 2000.

Ibrahim F. Haddad (ibrahim.haddad@lmc.ericsson.se) works for Ericsson Research Canada in the Systems Research Division. He is currently a DrSc candidate in Computer Science at Concordia University in Montréal.

Resources

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

A Linux-Based Automatic Backup System

Michael O'Brien

Issue #80, December 2000

A step-by-step procedure for establishing a backup system that will save time and money.

Frequently people take computers for granted. This behavior becomes very dangerous when people rely on a computer to store and manipulate important data but fail to back up those data. If you are reading this, then you are probably aware of the need for reliable backups. However, you may work with people who are not, and your job may be seriously affected by a loss of their data.

I work in a scientific research group. Our laboratories are modern, and almost all of our data acquisition is performed by computers running Windows 95. In essence, our whole business is to acquire information that is stored on computers. Data loss can end up costing thousands of dollars, especially when one considers the salaries of all the people who helped produce that data.

To protect our group from data loss, I proposed an automatic, network-based backup system for our irreplaceable data. The costs were negligible (we had a 486/66 computer that was not in use and a 3GB hard disk that cost us little more than one hundred dollars). I went through several versions of this system over the past two years, starting with a Windows 95-based system and ending up with a fast, powerful Linux-based system. The current version is easy to implement, inexpensive, powerful and reliable. Assuming you have a networked Linux machine ready, you should be able to use this article to set up your own automatic backup system in a short time.

Necessary Tools

All the tools that are needed for the automatic backup system are included with most Linux distributions. The first is Samba, an excellent open-source package that allows UNIX-type systems to communicate with Windows-based systems

over a TCP/IP network. The Linux version includes a utility called **smbmount**. It uses the **smb** file system kernel support unique to Linux, allowing any directories on Windows computers to be mounted to the Linux file system and manipulated as if they were on the Linux machine's hard disk. This will allow the archiving programs (in their update mode) to check to see if a file on the Windows machine needs to be backed up *before* it is transferred through the network, thereby reducing the network bandwidth requirements, CPU load and hard disk wear *dramatically*.

There are numerous archiving programs available for Linux, including **tar**, **bzip2**, and even the simple **cp** command. However, I chose to use tools from the open-source Info-ZIP project. These tools are included with most Linux distributions are available for various other platforms, are fast and small, and use an established file standard for Windows systems. Furthermore, the compression abilities of the Info-ZIP tools allow one to significantly reduce the size of the file archives on the Linux backup system.

Preliminary Steps

Network shares (a hard drive or any directory with all its subdirectories) must be set up on the Windows computers to be backed up. If file sharing is not already enabled, you can set it up from the Windows network control panel. Then, in the Windows Explorer, right click on the drive or folder you want to access from the network and choose the Sharing option from the pop-up menu. I recommend allowing read-only access so that crackers cannot alter or destroy your data if they somehow obtain your passwords. Make sure to record the names of these shares. It is a good idea to place the netbios names, DNS names and IP numbers of the Windows computers in your `/etc/hosts` file of the Linux machine (as directed by the comments in `/etc/hosts`), especially if your computers lie across different subnets.

Once this is done, you must prepare your Linux system to access and store the data. First create a mount point for the Windows shares by typing **mkdir /mnt/smb**. After that, you must decide where you will put the archived backups.

I put the backup files on a separate 1GB vfat (Windows) partition that remains unmounted at all times except when the actual backup processes are running. This way, the files are protected as much as possible from file system damage due to power outages, and the hard drive can be temporarily removed from the Linux computer and put into a Windows computer to facilitate recovery. In order to accommodate this, I created a mount point called **/mnt/backups**.

Scripts

A *script* is a text file containing commands that one would normally type at the Linux command prompt. You can use them to easily accomplish very complex tasks repeatedly. Making a script is as simple as typing the text into your favorite editor, saving it and then using the **chmod u+x** command on the file.

Listing 1 shows the script that backs up the DATA directory from the d_drive share on the computer named "higgins". This script runs on my Linux computer, "magnum", and is stored as the file root/backup/higgins.

Listing 1. DATA Directory Backup

The first line, while looking like a comment, actually instructs the computer to use bash to execute the script. Next comes all the shell variables that the main part of the script will use to back up the data on higgins. This practice of putting the case-specific values in variables at the beginning of the script allows the user to make new versions for new computers very quickly by copying the basic script and changing a few easily seen values. Listing 2 shows a different set of variables for a Windows 98 machine ("rick" with a shared C: drive) and a Windows NT machine ("tc" with a shared folder named "data"). Note how the Windows NT variables need to specify a user name and the password associated with that username.

Listing 2. Variables for the Windows Machines

The remaining lines actually do the work. The command **export PASSWD** puts the password in an environment variable that the smbmount program reads automatically. The smbmount command is executed next in case someone forgot to unmount an SMB share from the mount point. (If there is nothing there, smbmount returns a harmless error message and the script continues.) The smbmount program then attempts to mount the remote share. **-N** switch instructs it not to ask for a password to replace the value of the PASSWD environment variable. The **-n** switch communicates the username to smbmount.

An if statement checks to see if the specified backup files actually exist before doing any backup work in case the network may be down or the remote computer is switched off. In this case the script will terminate after making the mount point available again.

If the Linux machine can access the remote files, all archiving is done with the **zip** command. The **-r** switch is the standard recursion option, which makes zip go through every subfolder of the data directory. The **-u** puts zip in update mode, where it will only add or change files that are not already archived or

those that have changed. The **-v** parameter instructs zip to verbosely show the names of every file it checks on the display—a useful option for troubleshooting.

After a backup script has been set up for each computer, you can make a simple script named **master** to call each of the backup scripts sequentially. An example of my master script is shown in Listing 3.

Listing 3. Master Script

Activating the System

After all the scripts have been written, you can put a symbolic link to the master script in one of the `/etc/cron.d` subdirectories so that the computer will take care of the backups automatically. For my setup, I typed **ln -s /root/backup/master /etc/cron.d/weekly/master** to set automatic weekly backups. You can back up on a daily basis if you need to since the update option of archiving utilities minimizes resource requirements.

The first usage of a backup script, however, will require a lot of network bandwidth and CPU time. Hence, you may want to consider running it for the first time by hand or with the **at** command at night.

Caveats

Five important points should be noted:

1. Any shell script with passwords should be made unreadable by anyone but the owner by using the **chmod go-r** command.
2. If your data is very sensitive, you need to set up adequate security measures to keep industrial spies from hacking into your Linux machine and stealing your centralized data. See the Linux security HOWTO for more information.
3. The `smbmount` program tends to vary slightly across different distributions of Linux. Hence, if the scripts in this article don't work quite right for you, check out the man pages to see how your version of `smbmount` handles its command-line options.
4. Users of the Windows computers must be taught to keep their data under a central directory, such as "users" or "data", instead of several random directories spread across the hard drives. Some people are too lazy to move their files into a central directory, despite the fact that it takes only five seconds. You may have to actually move their files yourself before they will even start using the centralized directory. Remember, though, that these users may be the *greatest* threat to your organization in terms

of data loss since they never bother to make backup copies of their own data.

5. Finally, a hard drive is a very practical place to put the backups of irreplaceable data. My archive files use less than 400MB of hard disk and contain more than a 1.5GB worth of data. However, you may want to consider obtaining a large-capacity, removable drive for your Linux machine. With this, you can occasionally copy the archive files from your hard disk to a removable disk and take them home in case of physical destruction or theft of the machine.

Conclusion

A Linux-based network backup system for irreplaceable data files on many networked computers is inexpensive, reliable, easy to set up, trivial to expand and extremely practical. With just an hour of time you can potentially save your group or company many thousands of dollars in the case of a hard drive crash. Currently, my Pentium 150 workstation keeps archives of years of mission-critical data from eight computers spread across three buildings and two subnets. It takes me less than two minutes to add a new computer to the system due to the use of shell variables in the scripts.

This is the kind of task Linux was *born* to do. You can take an old surplus computer, make it “headless” with no keyboard or monitor and stick it somewhere in a closet where it will humbly do its work unseen. You can also run it on your personal workstation since the Linux tools can run in the background. You can set up an FTP server on the Linux machine on the fly if you need to restore files to a crashed computer or simply take the hard drive out and stick it inside a Windows machine. Since Linux has been designed to coexist with many different computers and operating systems, one can adapt the scripts to back up many different kinds of computers, including other Linux machines via NFS and even Macintosh computers with the netatalk and hfs packages.

Resources



Michael O'Brien is a graduate student at the University of New Mexico where he studies optics. Computers are both a hobby and tool that end up helping him get his work done. He manages a small computer room in his spare time and

likes to help Linux users on the Usenet newsgroups. He may be reached at mobrien@unm.edu.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux System Administration: A User's Guide

Marcel Gagné

Issue #80, December 2000

An excerpt from our French chef's upcoming book.

Next year, Addison Wesley Longman will be publishing my new book, *Linux System Administration: A User's Guide* (Copyright 2001, ISBN 0-201-71934-7). Since the focus of this issue is system administration, the kind folks at *Linux Journal* have provided me with an opportunity to give everyone a sneak peek at what is to come.

So, let me set the scene. It is a dark and stormy night (I've always wanted to write that), and a lone sysadmin is working late looking for ways to get home and still get his work done. This is a snippet from Chapter 15, a chapter I call "Creative Laziness". Along with a sizable crowd of others over the years, I have been preaching, I mean *speculating*, that creative laziness can be a wonderful tool. The kind of lazy person I admire works hard to find easier ways to do things and is always looking for the simpler, more elegant solution to any problem. If I may quote one of the greats of science fiction, Robert A. Heinlein, "Progress is made by lazy men looking for easier ways to do things."

The introduction to this chapter covers a variety of automation tools before arriving at this point. The hour has just struck 23:00. The pizza is long gone, and the cafeteria is out of coffee. *Lights. Camera. Action....*

Scripting for Interactive Sessions: **expect**

At first glance, it would seem that you are out of luck if what you want to automate requires human intervention. There are things that need a human: picking from a menu option, entering passwords or making decisions based on the information presented. Interactive applications require a user's reaction, don't they? The answer, for the cleverly lazy system administrator, is not always—thanks to a little program called **expect**.

While I had heard of expect sometime before, I discovered how useful this language was only a few years ago. My partner and I were developing a web-based system that required regular updates from the main computer's database, a database that would not allow command line scripting. The data we needed required the execution of an SQL statement that could only be entered via the vendor's menu interface. That SQL statement would then generate the data file we needed for the web interface. The whole process hinged on writing something that mimicked a user sitting at a terminal and entering information as the various prompts were presented to him or her. expect, a software suite/language based on Tcl, was the answer to this dilemma. Later, expect would make it possible to stretch this web tool well beyond what we, *ahem*, expected at the time.

Still wondering whether *you* need it? Remember that laziness discussion at the beginning of the chapter? Well, pretend that you are working late and the last thing you need to do before leaving is to log on to your remote site, make sure that a specific application has been completed (it is always done by 3:00 a.m.), and then download the file that application generates back to your local site. It is now 10:00 p.m. and you would much rather go home than wait there for the magic moment when the file is ready. You could just launch an **at** job that starts **ncftp** for the download, but you don't know the file name since the output name changes at each run. You find the name by logging into the menu system and checking the completion log. I am purposely making this complicated to demonstrate that there are instances that are hard to automate with a simple shell script.

The basic format of an expect script is this:

```
#!/usr/local/bin/expect
# Comments on this script (name, what it does,
# optional)
spawn some_command
set response myanswer
expect "Some prompt . . . ."
send $response\r
close
```

Here's what happens. The "spawn" keyword tells expect to begin some program. This could be a shell (**spawn /bin/bash**) or any kind of command through which the session will take place. With the "set" keyword, I am setting the variable **response** to some predetermined response. The language's namesake, the "expect" keyword does exactly what it sounds like it does. It scans the output of whatever command we invoked with spawn, searching for matching text. Then, "send" responds to the expected text with the first variable, **\$response**. Let's do something real now.

I run an Apache web server on my system with OpenSSL extensions for secure transactions. Starting Apache with the OpenSSL extensions running requires

me to enter a security passphrase in order for it to start up, because the private key files on the server are encrypted (see Chapter 26, “Building a Secure Web Server”). This is all fine if I am here to enter the passphrase, but what happens if the server goes down when I am not there? It hasn't happened for months, but these things happen and we do go on holidays sometimes. It could be something as crazy as me adding an SCSI card for my new tape drive. I might have forgotten (it has happened) to restart the web server with OpenSSL running. What then? To get around this problem, I wrote the simple expect script you see below:

```
#!/usr/bin/expect
# Routine: startapachessl
# Purpose: Start web server with OpenSSL active
#
log_file -a /tmp/expectlog
#log_user 0
spawn /bin/bash
sleep .2
send "usr/local/apache/bin/apachectl startssl\r"
expect "Enter pass phrase*"
sleep .2
send "mysecretphrasegoeshere\r"
sleep .2
close
```

When the system restarts, whether I am there or not, this script will restart my Apache web server with OpenSSL running. Looking at the script, you'll notice a couple of interesting things. For instance, the **log_file** parameter is new. What this does is define a log file for the execution of this script. Whether the file is written to or not is defined by the **log_user** parameter. If set to “1”, then logging will take place. I tend to use **log_user** when I am still testing the script, but you may decide you want to capture the output at all times. Notice as well that I am spawning a bash shell to execute the script that starts my server. Then, there are the **sleep** statements. In all cases, I have the shell wait one-fifth of a second before continuing. Finally, the **close** statement tells the spawned process that there is nothing more to come. At this time, expect terminates and returns to the process that spawned it.

There is no doubt that you could program these functions with other languages, but expect makes it easy. What you will find as you go along is that not every tool is perfect for every job. For quick and dirty automation of interactive applications, nothing beats expect. Fully exploring expect would require a book of its own (in fact, there is one). What I am trying to do is give you a taste of what you can do with it rather than explain every aspect of the language. Before I let you run off to do your own exploring, let me take these examples one step further.

We all know that changing passwords on a regular basis is as good a thing as choosing good passwords (see Chapter 6), and it is a fairly easy thing to have users do when they log in, but it is somewhat more difficult if they do not have

a login account. I'm talking about e-mail—only users, the ones whom you allow POP3 mail pickup (or web-based e-mail) but no actual command prompt access. A number of offices have precisely this kind of setup for their Linux system—it serves as an e-mail or Internet gateway and allows no logins. So, how do you allow users to change their passwords when they aren't allowed to log in? After all, changing passwords is an interactive activity as the following dialog will attest:

```
[root@myhost] # passwd
New UNIX password:
```

Even more complicated is that in order for non-root users to change their passwords, they must first enter their old password, so the dialog is even more complex. What now?

You could create a web-based form whereby a user could enter all that information up front (see Figure 1).




Figure 1. A Password Change Webform

A Perl script behind the form would extract the variables and pass them to an expect script that does the rest. If you are curious about this little web application or would like to use it, feel free to download it from my web site. In the meantime, have a look at this segment from the application:

```
#!/usr/bin/expect
# Routine: psdcmd
# Purpose: to change a user's password with expect
log_user 1
set uservar [lindex $argv 0]
set currpassword [lindex $argv 1]
set newpassword [lindex $argv 2]
set renewpassword [lindex $argv 3]
#
# log_file -a /tmp/expectlog
# send_user "Spawning passwd command with uservar.\n"
spawn su -l -c "passwd" $uservar
expect "Password:"
sleep .1
send "$currpassword\r"
sleep .1
```

```

#
expect {
    "(current) UNIX password:" {send
"$currpassword\r"}
    "su: incorrect password" {exit 0}
}
sleep .1
expect {
    "su: incorrect password" {exit 0}
    "New UNIX password:" {send
"$newpassword\r"}
}
sleep .1
expect {
    "BAD PASSWORD:" {exit 0}
    "Retype new UNIX password:" {send
"$renewpassword\r"}
}
sleep .1
expect {
    "su: incorrect password" {exit 0}
    "New UNIX password:" {exit 0}
}
#End of password change routine

```

The **set varname [index \$argv num]** construct represents arguments passed to the expect routine. Notice that our "spawn" parameter calls does an **su** to the username in order to change the password. By default, CGI scripts on your web server execute as some unprivileged user like "nobody" or "www", so we need to change our effective user in order to change the password. Incidentally, you want to keep the default user for your Apache server as non-root. The alternative constitutes a potentially horrific security weakness.

There is one other new item in the script. Look at the **send_user** parameter. This is essentially a print statement. I left it in the sample script because I wanted to show you a clever way of debugging your expect scripts. Every programmer has inserted debug statements into his or her code to monitor how things were going. This is the same idea in this case. You can use **send_user** as a means of communicating with the outside world in the course of the script's execution. Since I capture the output of the expect script via my Perl script, I will see these messages as well. By the way, the Perl script calls the routine in this way (the entire command is on one line):

```
$return_code =3D `./psdcmd "$username" "$currpassword" "$newpassword" "$renewpassword" `;
```

As you can see, the expect script is called with the username, current password, the new password and the new password repeated. We could simply have passed the new password twice, but we wanted to keep the verification aspect of the password change routine to be as close to what the user would experience at the command line. More to the point, it also is a good idea to force the user to confirm the password before changing it.

Automating Interactive Automation

Now that you have had your introduction to scripting with expect, I am going to make the process almost *impossibly* easy. Rather than manually creating an expect script, how about letting a program do that for you, too? When you install expect, you will also install a cool little program called **autoexpect**. Simply put, autoexpect will watch whatever you are doing in an interactive session and create the expect script for you. Here is the format of the command:

```
autoexpect -f script_outputfile command_string
```

For instance, let's imagine that we wanted to log in to a remote system that is behind a firewall, essentially a two-step login process. After we log in to the firewall, we then execute a login (telnet, ssh, etc.) to yet another system on the internal network, then execute a standard menu program. We would like to have this whole process of logging in twice and starting this menu automated for us. From the command prompt, we would then type this command:

```
autoexpect -f superlogin.script telnet firewall.mycompany.com
```

When you have finished your login, you can exit the menu and log out. autoexpect will have captured the entire session for you. Before running your new script, you will probably want to do some editing to clean things up a bit. autoexpect's output is probably a little wordier than you want. Furthermore, you will want to remove the lines that exit from your menu and log out, but the basics of the script and all the prompts are captured there for you. Make the script executable and you are almost done.

There is still one other thing you will want to add. At the end of your new expect script, add this command: **interact**

This tells expect to return control to you after it has done its work. Without it, expect closes the spawned process, and all you've managed to do is log in and log out very quickly.

In no way do I intend this to be the definitive reference on expect. I do, however, hope that this little introduction (indeed, this whole chapter) will serve to whet your appetite and inspire your imagination to explore other ways of developing constructive laziness. After all, we all have other work to do.

What's all this on your screen about a magic cloak?



Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. In real life, he is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy and edits *TransVersions*, a science fiction, fantasy and horror magazine (now an anthology). He loves Linux and all flavors of UNIX and will even admit it in public. You can discover lots of other things from his web site, <http://www.salmar.com/marcel/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Jigsaw: A Revolutionary Web Server for Linux

Ibrahim F. Haddad

Issue #80, December 2000

Mr. Haddad exposes the design philosophy and essential features of the Jigsaw web Server, an open-source project from W3C, and provides a brief guide for installing it on Linux.



Jigsaw is the World Wide Web Consortium's (W3C) leading-edge web server platform. It is a full functioning, object-oriented web server written in Java that provides a sample HTTP 1.1 implementation, and offers a variety of features on top of an advanced architecture.

The Jigsaw Web Server is designed to be a demonstration in technology rather than a full-fledged release. Initially, it was intended as a project to experiment new technologies. However, as of Jigsaw 2.0, the server broke the rules of test platforms to be more robust than the average web server, making it worthwhile to take a serious look at its features, potentials and possible future deployment.

Design Philosophy

The design philosophy of Jigsaw is to make it as portable, flexible and extensible as possible, while still providing a functional and robust web server. The design goals are met by having the Jigsaw server run within any Java-supported environment.

At its core, having an object-oriented design and implementation, Jigsaw is nothing more than a set of Java classes and extension modules. Therefore, adding capabilities to the server is not complicated. We can dynamically add

our own modules where every resource available to the server is an object, as opposed to a CGI script, and any object is available to end users via HTTP. The server can thus be extended by writing new resource objects. This is the replacement for CGI, where server extensions have to be written as processes. Jigsaw also supports CGI for use with existing CGI scripts.

Jigsaw's developers emphasize providing a well-structured source code, a full set of core Application Program Interfaces (APIs) and a high-quality set of documentation.

These factors offer a complete experimental platform that can be used by as many researchers as possible. This contributes to the success of Jigsaw as an open-source project providing a valuable draft to the future of the HTTP protocol and object-oriented web servers.

Supported Platforms

The Jigsaw server runs on any platform supporting Java. It has been tested on Windows 95/NT and Solaris 2.x. Many people have also reported successful installation and use on other platforms such as OS/2, MacOS, BeOS, Linux, AS-400 and AIX. I installed the Jigsaw server on two workstations powered by Red Hat 6.1 and 6.2, with JDK 1.1.8 and JDK 1.2.2 respectively, and in both cases it worked fine.

Getting Started

To install the Jigsaw server, you need to have JDK installed on your system. Downloading the latest version from <http://java.sun.com/> is recommended.

After installing JDK, you need to set up the PATH permanently in the startup file to have access to the JDK bin directory. If you are using the C shell, edit the `~/.cshrc` file in your home directory and add the following line:

```
set path=(/usr/local/jdk1.2.2/bin $path)
```

Please note that you need to change the path according to your own installation path. Then load the startup file `~/.cshrc` to activate the changes just applied.

```
% source ~/.cshrc
```

Now you will be able to access the Java binary directory without typing the full path.

Downloading Jigsaw

The latest (non-stable) distribution, Jigsaw 2.1.1, can be downloaded from the W3C home page. It contains the Java source code, the documentation and the pre-compiled classes. The 2.1.1 version, released in March 2000, includes new features such as XML-based serialization, Servlet 2.2 implementation, a new RFC2616 compliant cache, image metadata extraction using content negotiation, as well as digest authentication and ACL-based authentication.

Installation Procedure

Having installed JDK and set your path, you can proceed to install Jigsaw on your Linux system by following three steps:

1. Unpack the distribution file
2. Set up the environment
3. Build the property files

In the following subsections, we will examine each of these steps.

1. Unpack the Distribution File

The distribution comes in the form of `jigsaw-x.x.x.tar.gz`. You need to choose a place to unpack it. I installed it under `/usr/local/jigsaw`. However, you can install it in a directory of your choice. Since the selection of the installation directory will not be the same for all of us, we will call this directory `INSTDIR`. You have to change `INSTDIR` with the absolute path where you have unpacked the distribution.

To unpack the distribution, at the shell prompt you apply:

```
% tar -xzvf jigsaw-x.x.x.tar.gz
```

This will create a number of directories under the Jigsaw main directory:

Jigsaw/src jigsaw sources
Jigsaw/scripts sample scripts to start the Jigsaw server
and the JigAdmin server
Jigsaw/classes pre-compiled classes
Jigsaw/lib native code support for Solaris
Jigsaw/Jigsaw the root directory to run the server in

The `Jigsaw/Jigsaw` directory contains:

- The configuration directory for the server
- The configuration directory for the administration server
- The directory for log files
- The directory for caching when using Jigsaw as a caching proxy

- The exported file space

2. Set up the Environment

Next, we specify to the Java interpreter the place where Jigsaw classes are stored. We do this by setting the CLASSPATH environment variable. For Jigsaw 2.1.0 and up, we set it as follows:

```
% CLASSPATH=INSTDIR/Jigsaw/classes/jigsaw.jar:INSTDIR/Jigsaw/classes/sax.jar:INSTDIR/Jigsaw/classes/xp
```

3. Build the Property Files

The last step is to build the property files. We do that by switching into the Jigsaw subdirectory:

```
% cd INSTDIR/Jigsaw/Jigsaw
```

and execute:

```
% java Install
```

Running Jigsaw

After completing the installation procedure, we are set to run Jigsaw. We switch into the installation directory and type in the following command:

```
% java org.w3c.jigsaw.Main -host host -root INSTDIR/Jigsaw/Jigsaw
```

where host is the full IP hostname of the machine, and **INSTDIR** is the absolute path of the location where we have unpacked the distribution file.

Alternatively, we could run the provided script that would start the Jigsaw sever for us:

```
% ./script/jigsaw.sh &
```

Jigsaw will run and produce a debug message such as the following:

```
[root@byblos ]# ./scripts/jigsaw.shloading properties from: /usr/local/Jigsaw/Jigsaw/config/server.pro
*** salvaging resource manager state...
*** resource store state salvaged, using: 1
*** Warning : JigAdmin[2.1.1]: no logger specified, not logging.
JigAdmin[2.1.1]: serving at http://byblos.lmc.ericsson.se:8009/*** salvaging resource manager state...
Jigsaw[2.1.1]: serving at http://byblos.lmc.ericsson.se:8001/
```

Testing the Installation

To verify that the Jigsaw server is up and running, start your favorite web browser and point it to your workstation address at port 8001, which is the

default port for the Jigsaw server. *Et voilà!* We get the Jigsaw server home page (see Figure 1), similar to the Apache installation home page.

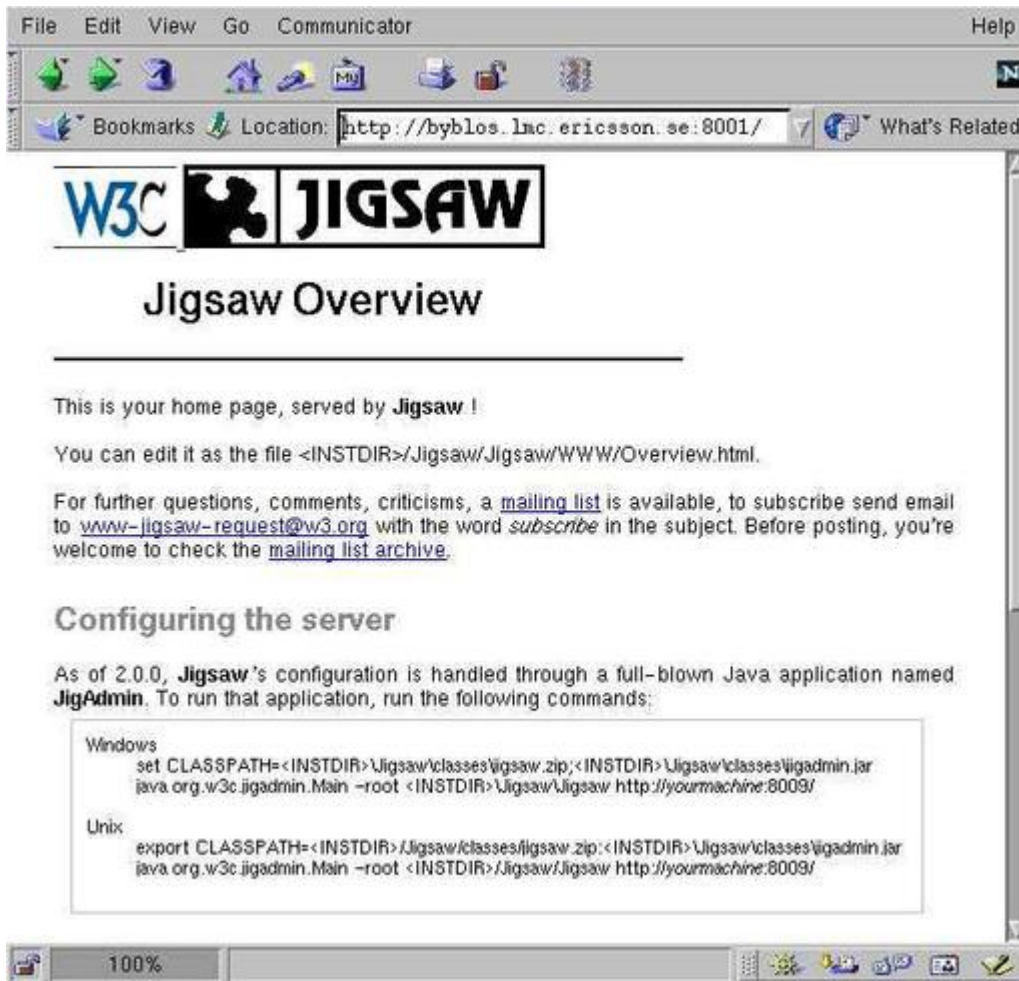


Figure 1. Jigsaw Server Home Page

Similarly, to check if the JigAdmin server started properly, we follow the same procedure except that we specify port 8009, which is the default port for the JigAdmin Server.

Setting Jigsaw to Run at System Startup

Now the Jigsaw server is up and running. However, if you reboot the machine, Jigsaw will not automatically start. To set up Jigsaw to start at boot time, you need to follow three steps:

1. Create a file called, for instance, jigsawstart with the following contents:

```
#!/bin/sh# Jigsaw Launcher Script
#
# Note:
# The paths JIGSAW_HOME and LD_JIGSAW_LIBRARY_PATH need
# to be adjusted to reflect your own installation
# Define Jigsaw Path
JIGSAW_HOME=/usr/local/Jigsaw/
export JIGSAW_HOME
```

```
# Define Jigsaw libraries Path
LD_JIGSAW_LIBRARY_PATH=${JIGSAW_HOME}/lib
export LD_JIGSAW_LIBRARY_PATH

CLASSPATH=${JIGSAW_HOME}/classes/jigsaw.jar:${JIGSAW_HOME}/classes/sax.jar:
ar:${JIGSAW_HOME}/classes/xp.jar:${JIGSAW_HOME}/classes/servlet.jar
export CLASSPATH
/usr/local/jdk1.2.2/bin/java -Xms16m -Xmx128m org.w3c.jigsaw.Main
-root
${JIGSAW_HOME}/Jigsaw $*
```

2. Save the file in /bin or /usr/bin and allow execution permission on it by typing
% chmod +x jigsawstart

3. Edit /etc/rc.d/rc.local and add an entry to start up the jigsawstart script. All we need to do is add the following lines at the end of rc.local:

```
echo "Running Jigsaw ..." /usr/bin/jigsawstart &
```

This procedure guarantees Jigsaw start up at boot time.

Administration Tool

Jigsaw comes with an administration tool called **JigAdmin**. JigAdmin is a graphical interface that communicates with the JigAdmin Administration Server. This server can administer multiple Jigsaw servers running on the same machine, provided that those servers have been launched by the same Java Virtual Machine.

The version of JigAdmin that comes with Jigsaw 2.1.1 is built with Swing components; it is easy to use with drag-and-drop features. Additionally, it comes with an extensive documentation explaining how to run JigAdmin, the command-line options, and a complete explanation for its menus and their functionality.

Starting the JigAdmin Server

The default configuration files provided by the default installation are designed to start two servers, an instance of the Jigsaw server and one JigAdmin Server. However, to start the JigAdmin, we can also use the provided sample script:

```
% ./scripts/jigadmin.sh &
```

We get the authentication window (see Figure 2). The realm used to access the server is admin, the default user is admin and the default password is also admin. It is highly recommended to modify the username and password after you log in the first time.



Figure 2. JigAdmin Authentication Window

After the authentication phase, we receive the JigAdmin main window (Figure 3), from which we control the server configuration (Figure 4).

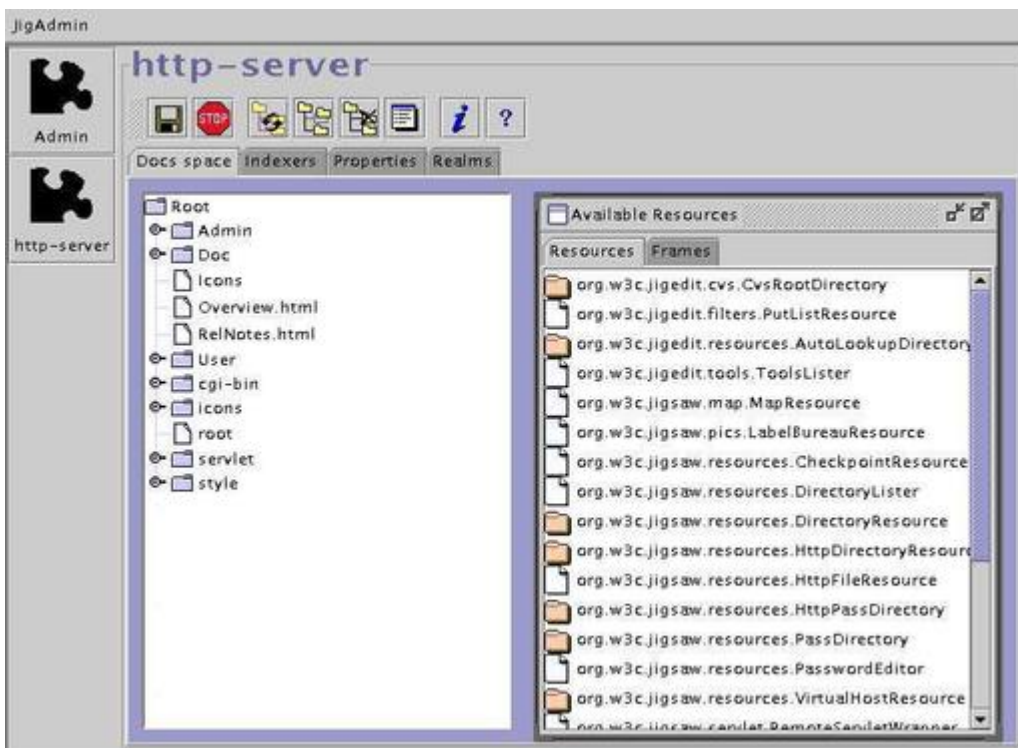


Figure 3. Main Screen for JigAdmin httpd-server



Figure 4. Server Properties Screen Shot

JigAdmin Client

Now that the JigAdmin server is running, we can access it with the following command:

```
% java org.w3c.jigadmin.Main [-root root] [url]
```

The default root is your current directory. So if you are in the same directory where you started Jigsaw, you do not need the **-root** option. If you are running the administration server on the same machine, using the default port 8009, you do not need to provide a URL; it is the administration server's. However, if you are not in the root directory, you can access the administration server with:

```
% java org.w3c.jigadmin.Main -root INSTDIR/Jigsaw/Jigsaw/
```

Jigsaw Documentation

Jigsaw comes with a very organized and rich set of documentation that is divided into six sections:

- A Quick Start document covering the basics on how to install and run the server.
- A Basic Concepts document discussing Jigsaw's design, authentication and access control.

- The Jigsaw Configuration Manual covers the basics of installing and running the server to descriptions of the most complex configurations you can do with Jigsaw.
- A Frequently Asked Questions document that answers all the FAQs about Jigsaw and its use.
- A Programmer's Documentation guide describing Jigsaw from a programmer's point of view and indicating how to extend it to fulfill your own needs.
- A User's Guide.

Other Jigsaw Packages

W3C released a Jigsaw Proxy Package that is a ready-to-run Jigsaw server configured as a proxy server. It is configured with a pre-installed caching proxy module and comes with an HTTP/1.1 server and client. Jigsaw also supports SSL with Jigsaw-SSL 2.01 beta which is public domain software that allows Jigsaw to use iSaSiLk as an SSL-provider. IAIK Jigsaw-SSL provides a SSLv3 supporting extension to the W3C HTTP Jigsaw server architecture for dealing securely with any incoming client request. IAIK JigsawSSL has been updated to operate on W3C's Jigsaw version Jigsaw 2.0.1.

Conclusion

Jigsaw is not the sort of web server around which you would build an enterprise-level Internet presence. Nonetheless, if you are serious about staying ahead of the curve on web protocols and infrastructures, you will want to have a test-bed machine running Jigsaw.

W3C is responsible for overseeing web standards, and anyone wishing conformity with HTTP/ 1.1 and the upcoming HTTP-NG (Next Generation) will want to do some testing with Jigsaw, as the latest version is totally HTTP 1.1-compliant. I highly recommend trying it out.

Resources



Ibrahim F. Haddad (ibrahim.haddad@lmc.ericsson.se) works for Ericsson Research Canada in the Systems Research Division. He is currently a DrSc Candidate in Computer Science at Concordia University in Montréal.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Elegance of Java and the Efficiency of C—It's Ada!

Frode Tennebo

Issue #80, December 2000

Tennebo recommends taking a look at Ada.

In his article "Comparing Java Implementations for Linux", Michael Hirsch (*LJ* August 2000) searches for "a programming language with the elegance of Java and the efficiency of C". This language already exists and it's called Ada.

The current version of Ada, Ada 95, is a revision of the Ada standard (Ada 83 was the original ANSI/ISO standard). It was standardized while C++ and Java were still in early childhood. Ada is often considered a bulky, antiquated language without the coolness of Java or the seriousness of C++. However, Ada is a powerful and modern language.

Ada has all the features of Java in addition to some of its own (such as generics and operator overloading). Actually, Ada provides all the features of Java *and* C++ (except for direct support for multiple inheritance) in one language, their web site (see Resources) provides a good comparison of the languages. You can even build j-code applets from Ada source. Thus, you have all the advantages of Java and more.

Ada is a strongly typed language which, if used correctly, can lead to less ambiguous code and more maintainable programs. The result is that errors can be detected easily at compile time instead of at debug time. More information on the Ada language can be found at their web site.

There are a number of Ada compiler vendors around, and you can get one for virtually any platform which, again, can cross-compile to (almost) any other platform. The most well known vendors are Rational, Aonix, R. R. Software and Green Hill Software (see Resources).

However, Michael Hirsch also wanted an open-source project. In reply, GNAT from Ada Core Technologies is to Ada as gcc is to C/C++, almost. GNAT is

written almost entirely in Ada and uses the gcc code generator. The public version of GNAT (see Resources for site where you can download it) is distributed under the FSF copyleft license. ACT also provides a nonpublic version of their compiler under a support contract.

I have written the equivalent Ada program to Michael Hirsch's Java and C++ programs as shown in Listing 1. As you can see, the listing is highly readable and looks very much like both the Java and C++ program. It is divided into three units, a package specification (like the C header files), a package body and the main program (a procedure). Ada has specific rules concerning what goes in which unit, and GNAT enforces this by requiring each unit to be in a separate file, where the file name must be the package/procedure name postfixed with .ads for specification and .adb for package/procedure.

Listing 1. Ada Equivalent of Java and C++ Programs

You can type in the example (or download it) into one file and run the utility **gnatchop**, which is included in the GNAT distribution. This will split the file into the three required files with the respective filenames. To compile and link, you can use the specially modified GNAT version of gcc, but it's more practical to use the **gnatmake** utility. Just type: **gnatmake perftest** and gnat will compile the required units and then bind and link the main procedure.

Since I do not have the same hardware as the author, I have performed the same Java and C++ benchmarks, in addition to the Ada benchmark, on two Linux machines and a ten-inch Sun Ultra. The results are shown in Table 1; the elapsed time is objects per millisecond.

The Ada compilers used were Gnat 3.12p, Gnat 3.13p and JGNAT 1.0p (p = public). The latter two were just released at the time of this writing . Please note that there are other Ada compilers that produce executables that may run faster or slower than the GNAT compiler.

Table 1. Results of Ada, Java and C++ Benchmarks

JGNAT is a peculiar beast, as it compiles Ada code into Java byte-code which can, in turn, run on any JVM. As you can see, it is only about 15% slower than the SUN JDK and JRE 1.2.01 is actually faster! And that is for precisely the same code compiled with the GNAT compiler, using the very first version of the JGNAT compiler! This can only get better.

I tried running the JGNAT-compiled programs using **kaffe** on Linux, but got:

```
[ft@modesty jtest]$ ~/kaffe/bin/java perftest@bbKaffe: mem/gc-incremental.c:823: gcMalloc: Assertion <
&&
size != 0' failed.
```

This suggests a bug in kaffe.

A small note on Ada in general and GNAT in particular: By default, the compiler compiles in many of the runtime tests to certify that everything goes smoothly. Therefore, if there's a problem, it can make a controlled action to avoid premature program termination. Usually, the action is to raise an exception. The **-gnatp** disables many of these tests and, hence, makes the program run faster. I do not recommend doing this, but it does provide for a better comparison with the C++ speeds.

For this particular test, Ada is about 19% slower on average than C++. On the Sun platform, this is down to about 8% using a comparable code generator (GNAT uses gcc 2.8.1 as its base). Note that this is a *very* limited test, and it only goes to show that Ada can produce running speeds comparable to C++. In some cases, Ada will actually run faster than the C++ counterpart. My guess is that the difference will level out, perhaps slightly in C++'s favour. The usual disclaimer applies.

However, it is in development and, later, in maintenance that the real savings can be harvested with Ada. Refer to www.rational.com/products/whitepapers/337.jsp for a thorough comparison between Ada 83 and C.

Conclusion

If you want a programming language with the elegance of Java (and more) and the speed of C/C++--take a look at Ada. It will even produce Java byte-code.

Resources



Frode Tennebo (frodet@nvg.org) lives in Halden, Norway. He first encountered Linux in 1994 and has been a Linux enthusiast ever since. He holds a seat on a local LUG board. Currently he spends most of his spare time with his five-month-old baby girl, leaving less time for computers.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

PHP4 and PostgreSQL: Building Serious Web Applications with Open-Source Software

Tim Perdue

Issue #80, December 2000

Tim walks us through a simple web application to demonstrate the features of PHP and PostgreSQL.

It wasn't long ago that to build a serious web application with ease meant laying out serious cash for a Cold Fusion license, a proprietary database like Sybase, and a Sun server. Fortunately, those days are long over. With the vast reach of Apache and recent maturation of free databases, there are realistic, perhaps superior, alternatives to proprietary software.

The best of the open-source breed is PHP, a scripting language that could be likened to Perl; and PostgreSQL, a powerful object-relational database. When you put PHP and PostgreSQL together, you can build anything from a simple guest book to a vast web-based accounting application. PHP provides the brains while Postgres provides the brawn.

I'm going to introduce a very basic PHP shopping cart and inventory application that takes advantage of Postgres' transaction ability. You can download the source code and read other tutorials on my web site, PHPBuilder.com.

The first thing I want to cover is the structure of the application. In every web application I build with PHP, I start by setting up a comprehensive library, which gets included on every page throughout the site. This library will be called `common.php` and stored in a directory called `include`.

Our library will handle routine tasks for us, such as setting up a connection to the database, figuring out who the user is, setting up the header/footer of the site, etc. By keeping these functions in one place, our application will be much cleaner and easier to maintain. For example, rather than hardcoding a database connection on every page, we code it once in the library.

Listing 1. Sample Library Coding

There, the first rev of our library is already useful. It connects us to the database and provides a simple HTML abstraction for us. As our site HTML grows more complex, we can include most of it in **site_header/site_footer** and have the changes apply throughout our entire application.

The library also includes a simple abstraction layer for issuing Postgres queries. I do this simply to reduce the amount of code I have to write.

And finally, we make a call to PHP4's built-in session management code. Calling **session_start** will cause PHP4 to reload any variables that you registered so they are available to the rest of your application.

Each page on our site will look fundamentally like this:

```
<?php<\n>
//include the common library
require ($DOCUMENT_ROOT.'/include/common.php');
echo site_header('Example Page');
/*
    page logic
*/
echo site_footer();
?>
```

Generally, when building an application, it's wise to separate the heavy-lifting logic from the actual presentation (in this case, HTML) as much as possible. I generally do this by wrapping logic inside of function calls, which are then included and called from the pages throughout your site. The reason is that you may eventually want to build different interfaces for your applications—perhaps an extremely lightweight interface for wireless applications. If the logic were interwoven with the HTML presentation, you would have to duplicate the logic to create another interface. If it's separated into function libraries, both interfaces can reuse the same logic.

The problem with function calls in PHP is that there is no standard exception-handling process. If there is an internal error in a function, how does the calling code know and pass that information to a user? In other languages, like Java for example, you would throw an exception inside of your functions (methods in Java). When you call the method, you would wrap it in a try/catch statement and handle the problem accordingly.

I generally solve this problem in PHP by always returning either true or false from a function call, and setting a global variable called **\$feedback**. The result is then testable and **\$feedback** can be printed to the screen if needed. There is an effort underway called PEAR (<http://pear.php.net/>), which is making an attempt

at standardizing error handling and database access, among other things, but nothing is concrete yet.

Here is an example of how you can call functions using my true/false method:

```
<?php<\n>
$result=function_call_name();
if (!$result) {
    //there was an error-display it
    echo $feedback;
} else {
    //continue on with success
}
?>
```

Now let's start thinking about our shopping cart. We need a few fundamental data structures to let us store our shopping cart data. For starters, we need an inventory database that lists item name, part number, price, and quantity on hand. We'll also need to keep track of our customers and the items the customer is purchasing. That's as complex as we'll get here.

Listing 2. Shopping Cart Data Structures

That should be enough to give us a rudimentary shopping cart. To normalize our database schema, I created a separate table that lists the contents of a customer's cart. This lets a customer have multiple items in their cart while allowing us to easily join the cart contents with the inventory database.

Now we need to think about the actions the various functions of an on-line store. The most basic functions include getting a new cart, adding items to that cart, then checking out. A real on-line store will need a lot more, such as the ability to browse the items, adjust quantities, etc., but I'll leave those features up to you.

I'll start with a simple function call to create a new customer. All that really entails is getting the next value out of the customer sequence we created, inserting that into a new record in the customer table, then registering that number in PHP4's built-in session management code.

Listing 3. Creating a New Customer

That took a little more code than I would have liked, but it shows you how to start and terminate a transaction in Postgres properly and how to check each query for errors. I will use the same routine error checks throughout all my code, and you should as well.

Always plan out how you will handle the situation when a query fails. Will you terminate your script outright, try the query again or just go on as if nothing

happened? Carefully consider the consequences of each option. For example, if you fail to get the next **customer_id**, you can't really go on to create a new customer record. If creating the customer record fails, you can't later update her address information or add items to her cart. Logical, right?

Now let's look at the process for adding items to a cart. This too is relatively easy. Before inserting an item into the cart, we should first check that the item exists in the database. This is considered best practice, since the item number will be coming in from a web browser, and could have been tampered with.

Once we know that the item exists, we can then test to see if it's already in the cart. If it is in the cart, increment the quantity, rather than insert another row. If it is not in the cart already, insert it into the shopping cart with a default quantity of one.

Listing 4. Adding Items to a Cart

Now we can create new customers and they can add items to their carts. Now we just need to be able to check out from the store and reduce the store inventory at the same time. This is the most complex part of the entire store and makes good use of Postgres' transaction and advanced-locking scheme.

To start, we will make use of Postgres' SELECT...FOR UPDATE syntax, which effectively locks selected rows so you can update them and commit your changes within a transaction.

By using this syntax within a transaction, you can guarantee that your data stays consistent. With some databases, like MySQL, you can't easily lock specific rows of data to prevent other processes from decrementing the inventory while you are also trying to do the same. What you wind up with is inaccurate numbers and a useless inventory count.

This query will also make use of subselects, another standard feature on the more powerful databases available. Subselects allow you to basically tie two queries together into one to make your life easier.

After we lock the rows, we need to issue a query to decrement the inventory count for each item in our cart. For simplicity's sake, we won't warn of lack of inventory, but we will successfully set the **item_count** negative if we fall below a 0 inventory level. You could then set up an admin page to view items with a negative inventory balance and order fresh inventory.

And finally, we will want to update the customer's table with the visitor's credit card, shipping info, and total sale amount, and then destroy the customer's session from PHP4.

Listing 5. Checking Out and Reducing Inventory

Now that's a reasonably complex transaction, each step of which must execute properly. If it doesn't, the entire transaction must be rolled back to put everything back in the proper order.

If you didn't have the luxury of transactions in this example, and a query failed in the middle of updating one of the inventory items, you would be in a lot of trouble. You'd have part of your inventory updated and part not updated. If the visitor retried the page, how would you know which inventory items to decrement again? The answer is, you wouldn't, and you would wind up with an inaccurate inventory.

This article by no means provides a comprehensive shopping cart solution—I could probably write an entire book on that, if I had the time. It does, however, lay the groundwork, and demonstrate the design and execution methodology that I recommend for every web developer. For further information and discussion, visit PHPBuilder.com.

Some of the pieces that I'm leaving out include pages to view the cart contents and the ability to view items in the store. Those pieces, along with all the code here, are available in a .zip file, along with a discussion/comment board; www.phpbuilder.com/columns/linuxjournal200009.php3.



Tim Perdue (tim@perdue.net) lives in Sunnyvale, California and is lead architect of SourceForge.net and founder of PHPBuilder.com and Geocrawler.com. He also enjoys sailing in the San Francisco Bay and studying the stars. Tim and his wife, Lisa, are expecting their first child in November.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

About the Mod: Part One

Dave Phillips

Issue #80, December 2000

The following article is an expansion and revision of material found in my book, *Linux Music and Sound*, published by No Starch Press.

There are many module file types, including the MOD, IT, S3M/STM, XM, MED and 669 formats. The original MOD format was used by ProTracker, one of the first trackers (mod composition software) for the Amiga. Many of the mod filename extensions indicate their origins on a particular tracker: the IT format comes from the Impulse Tracker, S3M/STM is from the ScreamTracker, MED is from the OctaMED tracker and so forth. The various formats differ in the number of tracks allowed, the number of samples supported and the permissible bit resolution of the samples. Fortunately for Linux users, the most popular formats (MOD, XM, and IT) are supported by the available trackers and players.

It should be noted that although trackers load and save modules in only one or two formats, mod players typically support a wide variety of file types. For example, the popular MikMod player, included in almost every mainstream Linux distribution, handles at least fifteen module formats, and the MODPlug plug-in for the excellent XMMS player supports more than twenty mod file types.

What Is a Mod Tracker?

A mod tracker is an application for composing music with only your computer and some sampled sounds. The basic design of a tracker is similar to a MIDI pattern sequencer. A pattern is defined by a number of beats that act as slots in which you place (track) your samples. Each beat includes information about the musical pitch for your sample, its instrument number and volume setting, and optional effects such as vibrato, filters and panning. Patterns are strung together in arbitrary sequences to create a song. The song is then saved in one or more (depending on the tracker) of the various mod formats.

Mod trackers first appeared on Amiga computers. Those machines enjoyed on-board sound support capable of handling up to four channels of 8-bit monaural sampled sound. With the advent of decent affordable PC soundcards, MS-DOS became the next platform of choice for module composers. Today excellent trackers are available for Windows, the Mac and, of course, Linux.

Trackers are especially well suited for making beat-oriented music such as pop/rock, techno and other dance styles, but because any kind of sample can be used the software is certainly not limited to any particular musical style. Check the MOD Archive, MODPlug Central and the United Trackers web sites listed at the end of this article for links to mod collections showing off the wide range of music made with trackers.

Of MIDIs and Mods

A tracker resembles a looping pattern MIDI sequencer. A series of beats or measures defines the loop period, events are placed on beats within the looping pattern, and there is some degree of fine control over the individual event. An event here means any sound file: events can be as simple as a single beat of a kick drum or as complex as an entire drum pattern or violin solo.

A MIDI file is very small compared to a mod, but it contains no sample data and must rely on a soundcard or external synthesizer to process its sounds and effects. A mod file includes sound sample data along with the sequence timing information and is accordingly much larger than a MIDI file.

The General MIDI (GM) patch map provides a specification for a common layout of sounds for all soundcards. However, cards from different manufacturers may fill their GM patch maps with samples of dramatically differing quality. Because a module contains sample data, a mod can be played on any computer with any soundcard, and listeners will hear your music played with exactly the same sounds that you used to compose it.

By now you might be thinking "So why use MIDI at all?" There are some very good reasons: MIDI sequencers are more highly evolved composition tools, with more possible connections to external hardware and auxiliary software utilities; file sizes may be a consideration, particularly if transmitted across low-bandwidth network lines; and the MIDI Manufacturers Association provides an industry-standard specification with a focused set of definitions of MIDI's capabilities.

By contrast the mod scene seems more chaotic. Many trackers have devised their own file types, which has led to a rather bewildering profusion of formats, and there is no governing body to help determine the organized definition and expansion of module capabilities. However, if you want to compose using

sampled sounds, if you want listeners to hear your music with exactly the same sounds as you composed it, and if you can live with a rather “middleweight” file format, then module tracking may be just what you're looking for.

Of WAVs and MP3s

Fortunately the situation is not an either/or scenario. Programmer Guy Thornley has written a useful little program called GMid2Mod that converts a standard MIDI file (preferably with a General MIDI patch map) to an XM format module, employing the default Gravis Ultrasound samples used by the TiMidity MIDI player. Using GMid2Mod I converted a four-channel MIDI file (with four GM instruments) to an XM-format mod. I used the MikMod player to convert the module to a CD-quality (44.1KHz, 16-bit) stereo WAV file, and I then used BladeEnc MP3 encoder to convert the WAV to an MP3 file with a bitrate of 128KBps.

As the following comparison of the file sizes indicates, it makes better sense for the composer of mods to distribute works in the original module format:

```
Interlude.mid 34KB
Interlude.xm 737KB
Interlude.wav 28MB
Interlude.mp3 2.5MB
```

Another advantage shared by MIDI and mod files (over the WAV and MP3 formats) is the ease with which they can be studied, viewed and/or rearranged in compatible composition software. For example an XM module can be loaded into any tracker with XM support, just as a standard MIDI file can be loaded into any MIDI sequencer that supports the Standard MIDI File format. Compositions and performances in the WAV and MP3 formats are not easily rearranged or dismantled into their constituent instruments.

Incidentally, if you want to head in the other direction, Kokai Istvan has written Xm2Mid, a utility for converting XM-format mods to standard MIDI files with a GM patch layout. It works best if your module is arranged using an instrument set identical to the GM patch map.

Linux Mod Trackers

As of August 2000 I counted 13 trackers listed on the Linux Sound & Music Applications site. Which one(s) you prefer to try will depend on your available resources, particularly your graphics capabilities, as well as your interest in developing a tracker. For X users, Michael Krause's SoundTracker (see Figure 1) is designed with an excellent GTK interface graphics, while Cedric Roux's powerful Xsoundtrack (see Figure 2) uses common Xlib graphics. Jason Nunn's FunktrackerGOLD (see Figure 3) is an excellent console-based tracker requiring only the ncurses library for its graphics.

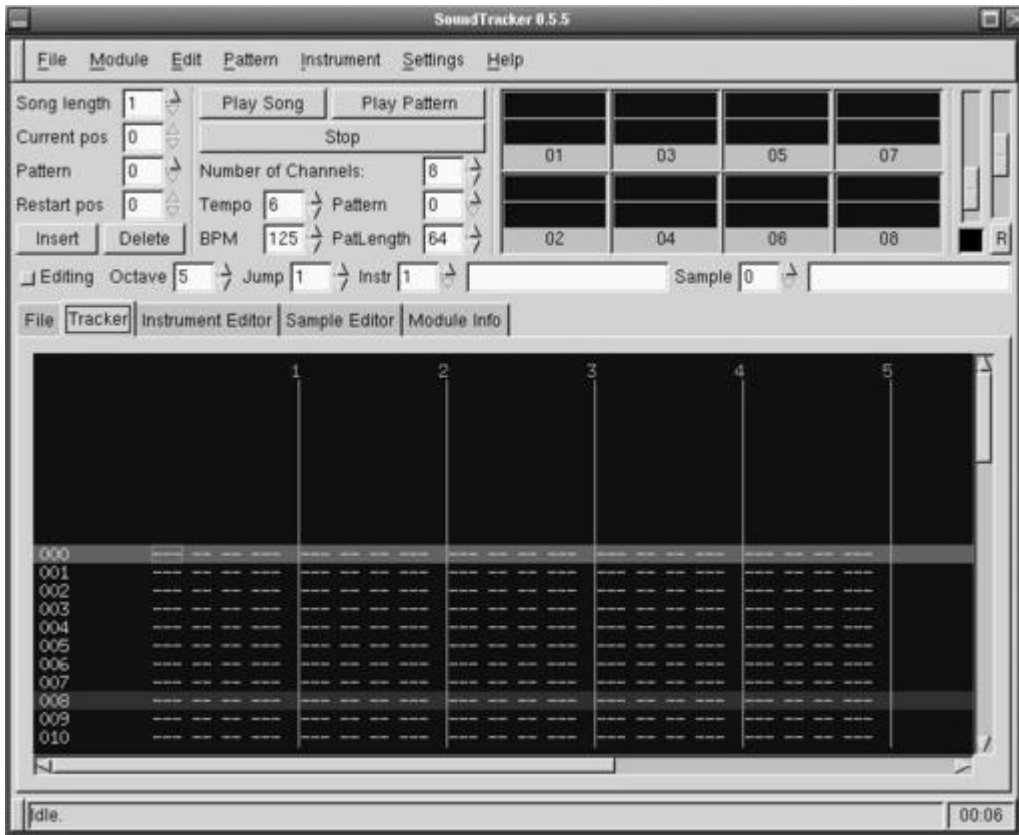


Figure 1. SoundTracker

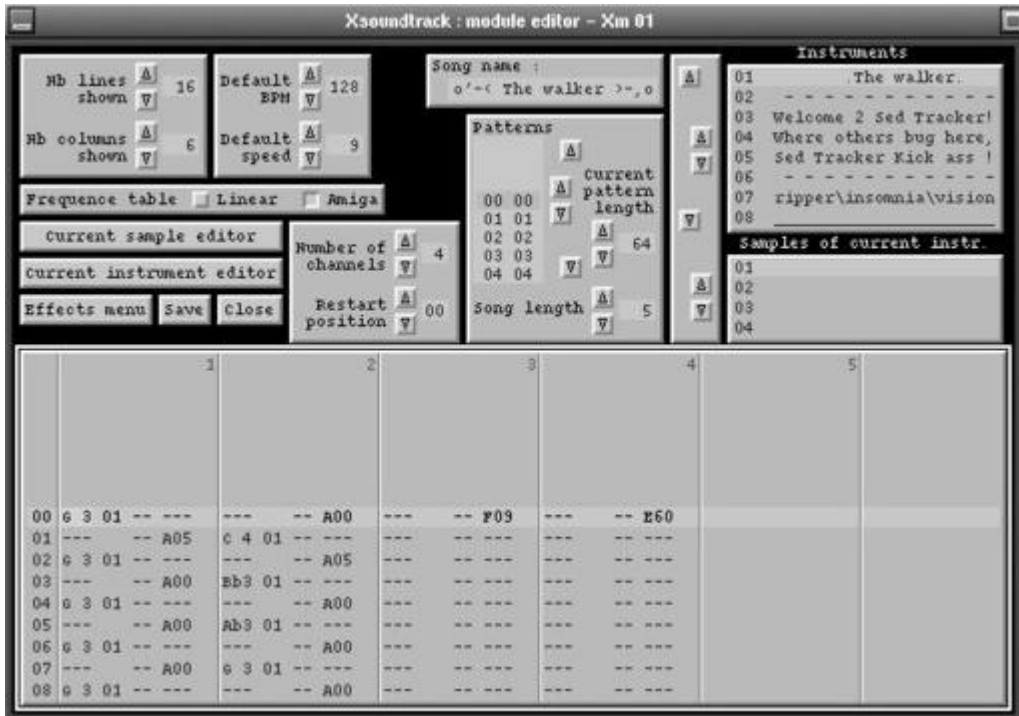


Figure 2. Xsoundtrack

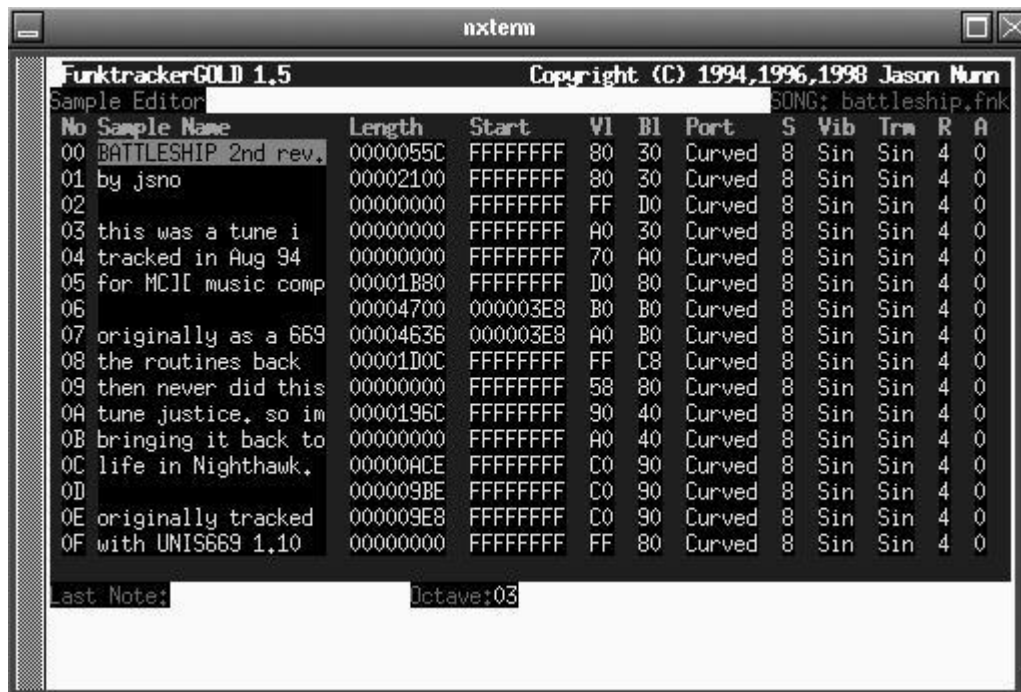


Figure 3. FunktrackerGOLD

Those three trackers have been developed to a stable and usable status. Other Linux trackers include the Sarah Tracker, Stupid Tracker [sic], and ocsatracker for the Linux console and the Industrial Tracker, the Rapid Audio Tracker, and Insotracker for X displays. All of this software is in various stages of development.

Non-Tracker Trackers

Tracker-style interfaces also appear in music software that does not create modules. Juan Linietsky's unique Shake Tracker (see Figure 4) combines the module tracking interface with MIDI output. If your soundcard includes a hardware synthesizer with SoundFont (sf2) support you can use its sound banks directly. Shake Tracker has just begun its development course, but it is already usable, and the author welcomes feedback and suggestions from users.

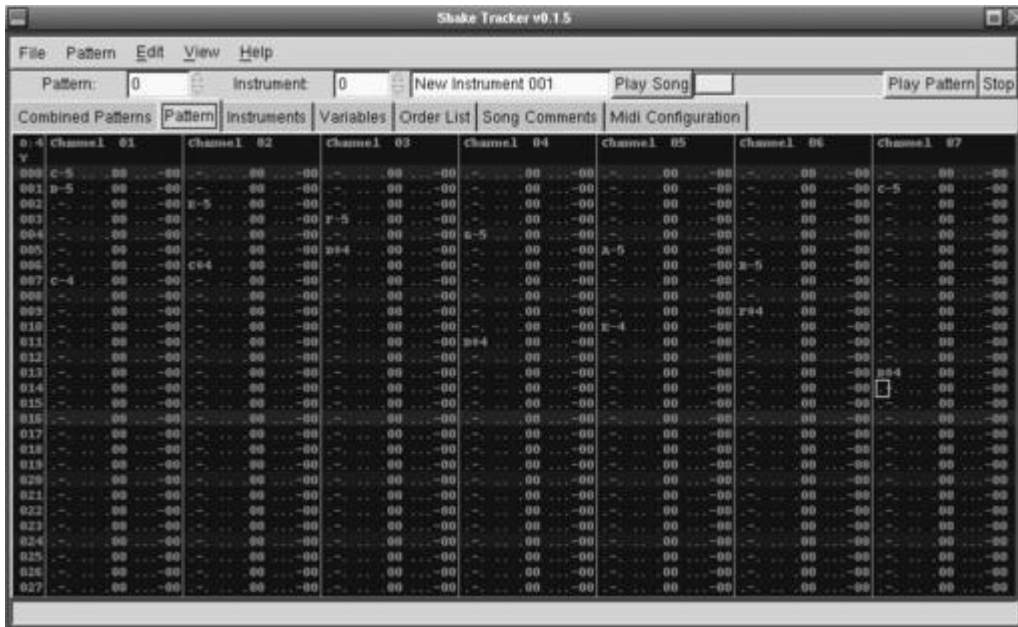


Figure 4. Shake Tracker

Tim Janik and Olaf Hoehmann have created the BEAST/BSE system which is an ambitious project that combines an audio synthesis network with a tracker's composition interface (see Figure 5). Currently, files are saved in the BSE format and are not compatible with mod trackers and players. Like Shake Tracker, BEAST/BSE is in early development, but it works and is already quite impressive.

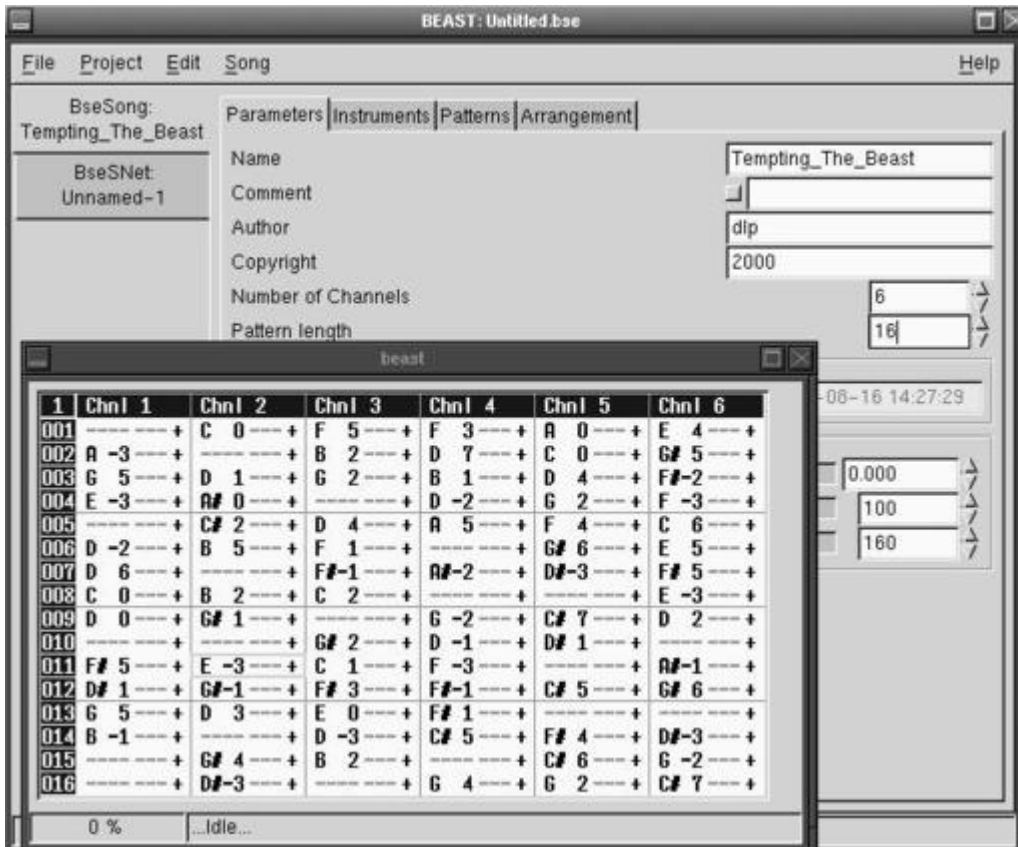


Figure 5. BEAST/BSE Pattern Editor

David O'Toole's GNU-OCTAL project plans to be the Linux equivalent of the Buzz tracker for Windows. Buzz differs from other trackers because it includes generators for sound synthesis, thus eliminating the need for a separate sample library. GNU-OCTAL is similarly designed, and although still in early development, the project is definitely worth watching (or joining: remember, this is Linux, where you too can get involved in the exciting world of audio software development!).

Linux Mod Players

Mod players for Linux are also available in console and X interfaces. As noted earlier, almost every Linux distribution includes MikMod (see Figure 7), which is available in various incarnations (e.g., console, GTK, Qt, Xforms, and Java). The popular MODPlug player for Windows has been ported as a plug-in for the excellent XMMS player (see Figure 6) and is also the playback engine for Gmodplay.

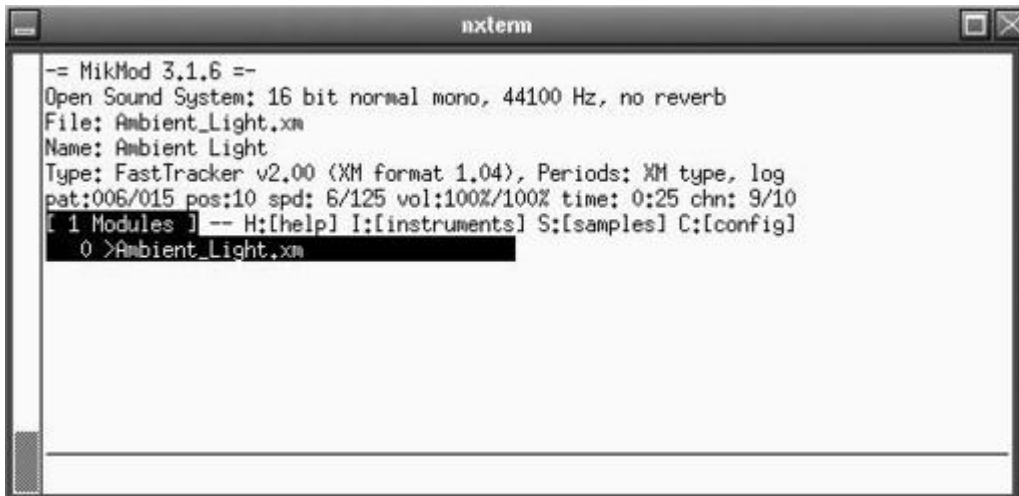


Figure 6. MikMod in Console Mode

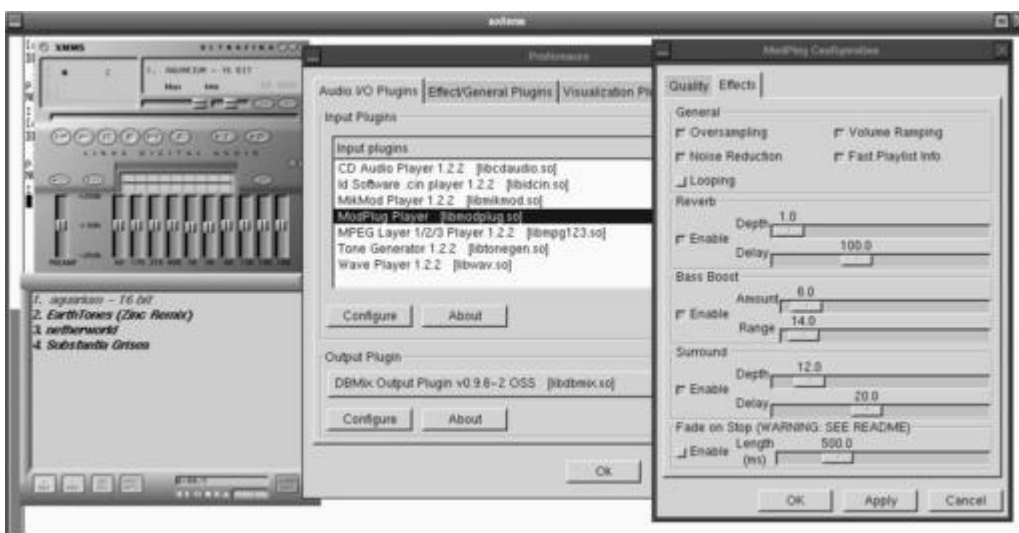


Figure 7. XMMS With MODPlug Configuration Options

Table 1 lists the available Linux trackers, Table 2 lists the players. The status ratings indicate the program's level of usability. The higher the number of stars, the more complete the application. Devel indicates that the application is at the alpha or early beta development stage. Entries marked with ? indicate that I was unable to build and/or run the application.

See the Linux Sound & Music Applications site at sound.condorow.net/mod.html for a current list of these and other Linux mod trackers and players.

Some of the popular mod trackers and players for Windows and MS-DOS may also run under WINE, VMWare, or DOSemu, but I haven't yet experimented with them. If you do try some mod software in those emulation environments please let me know how well they perform.

Demos and Mods

A demonstration program shows off neat graphics and animation hacks which are often accompanied by a musical score. The music is usually in a mod (or MIDI) format played by an embedded player (MikMod's libmikmod is very handy for playing mods in Linux demos). I highly recommend seeing the outstanding Loop, at mustec.bgsu.edu/pub/linux (binaries and sources are also available there), and State Of Mind, at skal.planet-d.net/mind/index.html. Both of those demos use an embedded libmikmod to play their scores.

Learning More about the Mod

The United Trackers and MODPlug Central sites are excellent guides to finding trackers and players, sample archives, links to mod collections, and tracking tutorials. The MOD Archive lists an enormous number of mods for your enjoyment and study. You might also want to keep up with the alt.binaries.sounds.mods and alt.music.mods newsgroups, where many independent musicians post their creations, questions, and information regarding the very active mod tracking scene. (See Resources)

In the next issue we'll take a look at SoundTracker.

Resources



Dave Phillips maintains the Linux Music & Sound Applications Web site and has been a performing musician for more than 30 years. His work with music software dates back to 1985, and he has been a Linux user since 1995. He is a

founding member of the Linux Audio Development group and has been active in the Linux audio software development community since he began using Linux. His publications include contributions to *The Csound Book* (MIT Press, 2000) and several articles in the *Linux Journal*. *Linux Music & Sound* (No Starch Press, 2000) is his latest publication.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Debian Package Management, Part 1: A User's Guide

David Blackman

Issue #80, December 2000

A HOWTO for Debian package management.

Debian has one of the most powerful and versatile package management systems of any Linux distribution. It is also incredibly cryptic. However, once you start using it, I promise it does get easier.

Debian's basic package management tool is **dpkg**. **dpkg** is actually built on top of **dpkg-deb**, but I'll get into that later. There are a suite of tools built on top of **dpkg**, including **dselect**, **apt-get**, **console-apt** and others. This article functions as a HOWTO, providing commands and instructions on ways to manipulate **dpkg** and receive pertinent information about your Debian, or Debian-based system. I'm not going to cover everything, but enough so that, you'll be able to use the package management capabilities of your Debian-based system with proficiency.

Every Debian package is an archive ending with the extension ".deb". For this article, I'll refer to a package as a "deb". debs are usually named in the following manner:

```
zsh_3.1.6.pws21-1.deb
```

For the examples in this article, I use "foobar.deb" when a deb needs to be substituted, and "foobar" when a package name is to be substituted.

For those familiar with the Red Hat Package Management (RPM) system, **dpkg**'s approach to package management is much different in its approaches. RPM is a file-based package manager, meaning it checks for specific files required by a package like **libgtk-1.2.so.0**. In contrast, **dpkg** is package-based, so it checks to see if you have a specific package, such as **libgtk1.2**.

Dpkg: The Root of All Debian

Most people will not use dpkg for day-to-day package management, but it is an incredibly powerful and useful tool. Dpkg's most basic action is to install a package, which is done with the command **dpkg -i foobar.deb**. This will install the package, while backing up any existing versions of the package. The command **dpkg -i -R /foo/bar** will install all of the debs in a directory.

We all know from experience that sometimes a package won't install properly or refuses to configure. Or maybe the user aborted the configure process. dpkg has some helpful tools for making the configuration process easier. In these instances, **dpkg -configure <package>** will finish the configuration of the specific package, and **dpkg --configure --pending** will configure all packages with configuration pending.

It is just as easy to remove packages with a few simple commands. Using either **dpkg -r <package>** or **dpkg -remove <package>** will remove a package and leave its configuration files. If you want to remove all files related to the package, including its configuration files, use the command **dpkg --purge <package>**.

Getting Information

Debian package management provides several ways to find information on what packages are currently installed and what files each package provides. One way to sort packages is by using a pattern, facilitated with the command **dpkg -l <pattern>**. If necessary, a wildcard can be used as the <pattern>. Using dpkg -l alone will provide a list of all the packages currently installed on your system.

Using Debian package management also allows you to see what files were installed by a specific package with the command **dpkg -L <package>**. Alternatively, to find out which package owns a file use **dpkg -S <file>**. These searches can also be done with a pattern, including wildcards.

When you need to get information about a particular deb, this list of commands is helpful:

dpkg -l foobar.deb lists detailed information about a deb
dpkg -c foobar.deb lists the contents of a deb, similar to tar's -c option
dpkg -x foobar.deb <dir> extracts a deb into the specified directory
dpkg -X foobar.deb <dir> lists the files as it extracts them, like tar's -v option

Force It!

In addition to all of the useful information that dpkg accesses for productive package management, it also has a rich set of options for screwing up your package management system royally. To quote dpkg's manpage, "Warning: These options are mostly intended to be used by experts only. Using them without fully understanding their effects may break your whole system." I won't cover this set of options here, as you should never really need to use them. However, if you ever need to force dpkg to ignore dependencies, overwrite files installed by other packages, ignore conflicts or anything else the Debian package management system is designed to *prevent*, look at `man dpkg`.

APT

APT stands for "Advanced Package Tool". It does a wonderful job of simplifying the life of a Debian user. If the last session discouraged you, fear not! APT is what I use for most of my package management. Why then, you ask, did I force dpkg on you? Because it's important to know what is underneath APT.

The APT system consists of three major parts: the configuration file, `sources.list`, and two programs called `apt-get` and `apt-cache`. There also are some minor parts, like `apt-cdrom`.

`sources.list`

Once you learn how to use it, `/etc/apt/sources.list` will become one of the most important files in your system. For the most part, **`sources.list`** consists of ftp and http addresses where APT can go to pull information, and is a great way to organize your resources in one location. A typical entry looks something like:

```
deb http://http.us.debian.org/debian unstablemain contrib non-free
```

The first part of the entry is either "deb", for a line that specifies where to acquire binaries from, or "deb-src" which specifies where to acquire debian source packages. The next part is called a URI, which is similar to a URL; this is the root of the debian directory. After the URI comes the distribution, which is generally listed as "stable", "frozen" or "unstable", although a specific distribution, like Hamm, Slink or Potato, can be used. Distribution can also be given as an exact path, in which case it needs to end with a "/", and no components can be specified. Components are usually "main", "contrib", "non-free", "non-us/main", "non-us/contrib" or "non-us/non-Free". That's about all you need to know to take advantage of `sources.list`. CD-ROM entries are generated by `apt-cdrom` (more on this later), so you don't need to write those by hand.

Editing the `source.list` file is done to change the sites APT uses (changing mirrors), to change which distribution APT will be getting packages from, or simply to add new URIs for “unofficial” debs, or sites which maintain newer debs, like KDE and HelixCode, respectively. After editing this file, you should always run **apt-get update**. For a listing of Debian mirrors to use, check <http://www.debian.org/>.

apt-cdrom

One way to add your Debian CD-ROMs to APT's database is with the remarkably easy-to-use **apt-cdrom**. The command **apt-cdrom add** should automatically mount your CD-ROM (provided `/dev/cdrom` is correctly linked), scan it and create the correct entry in `/etc/apt/sources.list`.

As with anything in Linux, that's not the only way to do it! If your Debian CD-ROM is mounted (the mount point must be listed in `/etc/fstab`), you can use a few flags to accomplish the task. Use **-d <mount point>** to specify the mount point, **-f** to make **apt-cdrom** **not** check the individual packages, and **-a** for a thorough package scan that will look for package files everywhere on the CD-ROM.

apt-get

Debian's greatest claim to fame is **apt-get**, an incredibly smart (okay, smart *most* of the time), easy-to-use package tool that automatically deals with package dependencies and conflicts. The first thing you should do (and do it every time you edit `/etc/apt/sources.list`) is run **apt-get update**, which pulls the package information used by **apt-cache** and **apt-get** from these locations.

Now, you can go about installing packages. APT's most used action is probably `install`. To install a new package (and automatically meet its dependencies), use **apt-get install <package>**. If there are other packages that must be installed, APT will let you know before downloading them.

If you install a deb without meeting the requirements, or a problem is encountered during `install` or `remove`, you will often be unable to work with `dpkg` until the problem is resolved. One method to use for solving these problems is the command **apt-get -f install**, which will try to work everything out and ask questions as it goes along.

APT can also download sources with **apt-get source <package>** if you have any `deb-src` lines in your `/etc/apt/sources.list`. To remove packages, use **apt-get remove <package>**. If you want to clean out the archives, there are two options. To remove all archives, the command is **apt-get clean**; if you only want to remove old archives, use **apt-get autoclean** instead. When the time comes to

upgrade all packages to their latest version, one easy command, **apt-get upgrade**, will take care of it. Meanwhile, **apt-get dist-upgrade** is suited for upgrading to a new version of Debian, automatically reconfiguring dependencies for packages whose names have changed.

A few other command-line options are helpful when using apt-get: “-d” only downloads archives but doesn't try to install or unpack them; “-s” for simulation, it won't actually do anything; “y” answers yes to all questions; and “-b” tries to build a source package after downloading it.

Additionally, apt-get can use regular expression patterns such as **apt-get install '*.mame.*'** or **apt-get remove 'mozill.'**. Apt-get will also try to match an element appearing at a specific point in the string. For example, to match “pilot” only in the beginning position, like “pilot-manager” and not “gnome-pilot”, use “^” (which is the “beginning of line” character) in an expression like **apt-get install '^pilot.*'**.

Upgrading

I have not yet found another distribution that makes upgrading to a new version as easy as Debian does. As mentioned previously, when you want to upgrade your version, simply change the distribution part of the URI in `/etc/apt/sources.list`, then do an **apt-get dist-upgrade**. Often, it helps to run **apt-get dist-upgrade** a few times to get everything happily installed and configured. You can only use this to go to a newer version, from stable to frozen, stable to unstable or frozen to unstable. You **cannot** downgrade.

apt-cache

apt-cache is great for finding information about packages, even packages you don't have installed! The command **apt-cache show <package>** will print various information about the package, including dependencies, full name, what it provides, the short and long descriptions, and, most importantly, the size when unpacked. **apt-cache depends <package>** provides a list of what other packages the selected <package> needs installed to work properly. To print a complete list of every package available, use **apt-cache pkgnames**.

The most useful feature of apt-cache that I've found has to be **apt-cache search <string>**. This option will search through all package names and descriptions looking for occurrences of your <string> selection. Obviously, this can be a great time-saving device.

To further assist your managing capabilities, there are some options that apt-cache can utilize. For example, **-i** lists only the important dependencies, **-f** prints

full records (just like "show") after a search and **-names-only** limits searches to package names.

Frontends!

While I can usually get by with dpkg for package management, apt-cache for searching and apt-get for installation, sometimes I want to get a better idea of what's available to me. This is where dselect, console-apt and gnome-apt come in.

dselect

dselect is the granddaddy of the Debian frontends. As the first part of the installation process, it is the first thing new Debian users are greeted with, and it scares them. And it's true that dselect can be incredibly scary as well as incredibly dense. This section will be the *short and sweet* introduction to dselect. First, my best advice when using dselect is to read all of the on-screen help. Although not very user-friendly, everything you need to know is contained there.



```
dselect - inspection of package states (avail., priority) verbose:v help:?
EIDM Pri Section Package Inst.ver Avail.ver Description
nn Std net ipmasqadm <none> 0.4.2-1 Utility for configuring e
nn Std net netkit-inetd <none> 0.10-3 The Internet Superserver
nn Std net netkit-ping <none> 0.10-3 The ping utility from net
nn Std net netkit-rpc <none> 0.10-3 The rpcgen and rpcinfo ut
nn Std net portmap <none> 5-1 The RPC portmapper
-----
New Standard packages in section otherosfs
-----
nn Std otherosf floppyd <none> 3.9.7-1 Daemon for remote access
-----
New Optional packages
-----
New Optional packages in section admin
-----
nn Opt admin aide <none> 0.7-6 Advanced Intrusion Detect
ipmasqadm not installed ; new package (was: new package). Standard
ipmasqadm - Utility for configuring extra masquerading functionality

Ipmasqadm is used to configure extra masquerading functionality, usually
provided by additional kernel modules.
```

Figure 1. Dselect-Inspection of Package States

When you first start dselect, you'll want to go to "[A]ccess" to select the correct access method for installation. Meaning, select "nfs" if you have a local nfs mirror of Debian, and "apt" if you want to install over the Internet (http/ftp) or if you've configured your sources.list for local addresses. Other methods available are CD, Multi-CD (if your distribution has multiple CD-ROMs as opposed to one), Floppy and Mounted.

Once you've selected your access method, dselect needs to find out what packages it has access to; to accomplish this, select "[U]pdate" and wait a minute.

"[S]elect" is where the actual package management happens. First, you'll be presented with a help screen. **READ IT!** When you finish **READING IT**, press space to get out of help or "." to read the keybindings. You can move around the package listings by searching or by using up/down, page-up/page-down, home/end and left/right. The following are some basic, useful keystrokes in dselect:

/ search\ repeats the last search? brings up helpd scrolls down the package informationu scrolls up it

When you've highlighted a package you want to do something with, use:

+ install or upgrade- remove= leave in present state

When you're ready to leave the selection screen, select:

<enter> to confirm, quit and check dependenciesQ to quit, confirm and override dependenciesX or Esc to abandon all changes

After you press return, you may be presented with a list of packages; these may be either dependencies of or conflicts with the packages you selected. Look at the packages; the bottom half of the screen will tell you what the problem is. Once you've resolved these problems, press enter.

After you've made your changes in "[S]elect", either "[I]nstall" or "[R]emove" which will bring your system up to date with the changes. "[C]onfig" is only necessary if any package configuration failed.

Gnome-apt

Gnome-apt is a GUI package manager for Debian built around Gnome. It is incredibly intuitive and can be quite useful. Gnome-apt displays package sizes, dependencies and just about any other relevant information in a well laid-out interface. The packages are presented in a sorted tree (see Figure 2), and gnome-apt can display packages broken down alphabetically or by section, status or priority. It also has easy and powerful search functionality. Gnome-apt is included in Debian 2.2 (Potato) and the current unstable distribution (Woody).

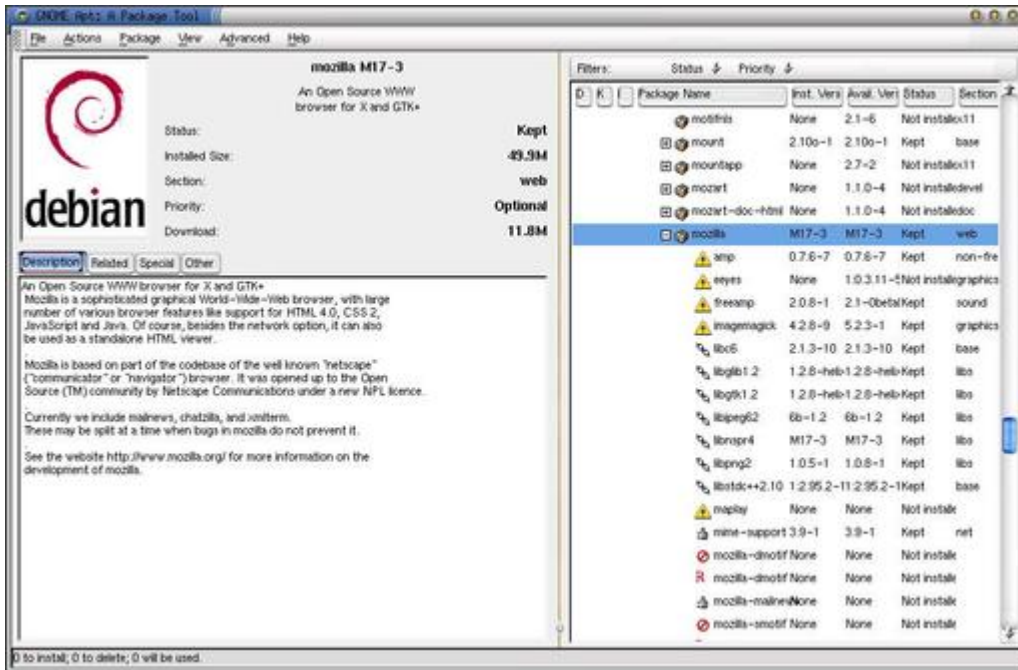


Figure 2. Gnome-apt: A Package Tool

Kpackage

KPackage is the KDE package front end for both RPM and Deb. It uses a combination of tabbed and tree interfaces (see Figure 3), and the functionality is similar to gnome-apt. One of the cool features of Kpackage is that the dependencies are hyperlinks to the packages they reference. Kpackage also lists the files included in each installed archive and checks to make sure they all exist. It can be found at <http://www.general.uwa.edu.au/u/toivo/kpackage/>.

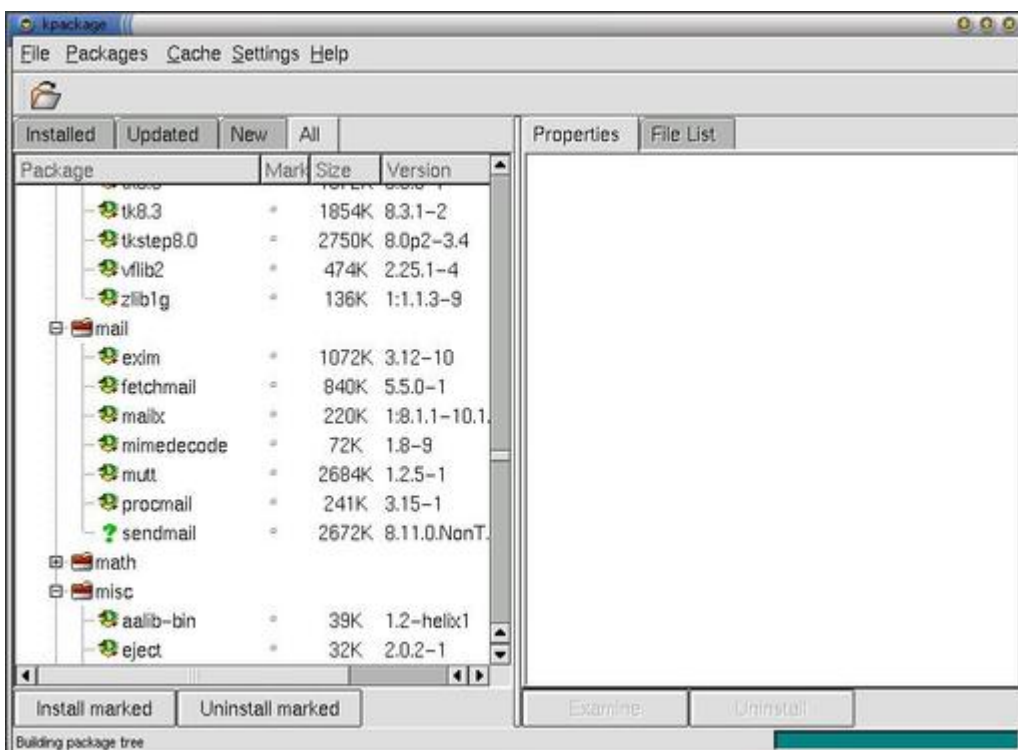


Figure 3: Kpackage: Building Package Tree

console-apt

console-apt is a new front end just for APT. Currently, it is only available in the unstable distribution of Debian; as it is still in development, I won't say too much about it. Console-apt does have some nice features, though, including a much more intuitive and usable interface, progress indicators for downloads, selective upgrades and the ability to search, sort and filter the package listings.

The End

Using these options and features, you should now be able to maintain and manage your Debian package system with ease. Trust me, it sounds more complicated than it actually is. I've tried to provide as many options as Debian offers but had to leave some out; I have not yet been able to pry myself away from APT. In conclusion, I haven't seen any RPM front end that I feel is on the same level as APT.



David Blackman is a student system administrator at Stuyvesant High School in New York City. He has recently fallen in love with Helix Code Gnome. He enjoys pointer arithmetic in C and the evils of Perl.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Typesetting with groff Macros

Wayne Marshall

Issue #80, December 2000

“Reports of troff’s death are greatly exaggerated.” —W. Richard Stevens, 1998

“In the beginning was the word.” And from the wordy primordial void there soon arose the blank page, the toner cartridge and the now ceaseless human craving for print. If you have a desire to look good in print, or just need to knock out a memo, term paper or letter to mom, you should know about **groff**. groff is a rich yet accessible set of document formatting tools and is available as standard equipment on every Linux system. groff can help take your words and typeset them beautifully on the printed page.

groff refers specifically to the GNU and updated version of **troff** (that venerable document formatting system developed for UNIX in the prehistoric era, before the Internet, compact disc and microwave popcorn). Traditional troff was first written in the early 1970s by Joseph Ossana at Bell Labs, rewritten a few years later by Brian Kernighan and designed for the computers and typesetting equipment available at the time. The GNU version of troff—first called **gtroff**, now simply groff—was written in the early 1990s by James Clark. While remaining compatible with traditional troff, groff offers several key enhancements making it easier to use, more powerful and containing fewer limitations than the program it supersedes. GNU groff is actively maintained and continues to evolve. In addition to Linux and other UNIX/UNIX-like systems, ports of groff are available for most of the other platforms out there. This ubiquity and open-source freedom lets you publish and share your documents portably and freely among platforms.

Using groff’s macro capabilities for generating printed output is the focus of this article. It should also be mentioned that groff serves as the formatting engine for the on-line manual pages produced by the **man** command. If you need a sample of the typesetting prowess of groff, simply generate a printed manual page with the **-t** option to man:

```
man -t troff >troff.man.ps
```

This will produce a PostScript version of the manual page for groff, which you can then view on-screen with one of the PostScript previewers (**gv**, **mgv**), print directly with a PostScript printer or print to a non-PostScript printer using a PostScript interpreter such as GhostScript. (You should really take a look at this man page, by the way. It provides a thorough summary of all the additional features available in GNU groff, with more detail than presented here.)

groff offers all the niceties of computerized typesetting, including automatic ligatures, kerning, hyphenation and end-of-sentence spacing. groff also provides low-level control over all aspects of page layout by means of typesetting commands embedded into an otherwise plain text file. Most often these commands—or, in groff parlance, *requests*—are specified with a period in the first column of the line containing the command. For example, the following snippet of document has embedded commands for increasing the left indent and decreasing the current line length:

```
This is an example of a groff document..in +0.5i
.ll -0.5i
When formatted by groff, the text continuing
here will appear indented by one-half inch
from both of the previous margins.
```

Although it is possible to format a document completely using such “raw” groff requests, it is more typical for endusers to work with a collection of predefined *macros* that encapsulate sequences of raw requests into single commands. For example, if we wanted to create a macro for the block indent commands in the previous snippet, it might look like this:

```
.de Bi.in +0.5i
.ll -0.5i
..
```

The **.de** request begins the definition of our macro named **Bi**, and the double period on the last line marks the end. Invoking a macro within a document follows the same syntax as using a raw request (the name of the macro follows on a line with a period in the first column). Our new macro used in a document would look like:

```
This is another section of my groff document..Bi
Oh boy, now the text continuing here is indented
from both margins!
```

If at some later time we want to increase the block indent to three-quarters of an inch, we need only change the macro definition. All instances of **Bi** throughout the document will then format with the new dimensions.

So far, we haven't seen a whole lot here to get excited about. One of the limitations of traditional troff is that the names of all commands, macros and

other variables are limited to two characters. Two measly characters? As mentioned earlier, troff was developed in the veritable stone age of computing, when every bit mattered, and succinctness was sublime. While the developers of troff and the standard macro packages have done their best to devise naming schemes that are as mnemonic as possible within this two-character constraint, the resulting interface is about as user-friendly as 80x86 assembly language (which at least uses three characters for most of its instruction set!).

Fortunately, GNU groff eliminates this two-character naming limitation. For both the macro developer and the eduser, the

most significant enhancement of groff is that all names, including macros, numbers, strings, fonts and environments, can be of arbitrary length. Groff also allows for the *aliasing* of troff commands, macros and variables to provide alternative names for existing ones. We will exploit this feature heavily through the rest of the article. In fact, let's begin right now by aliasing the groff alias command itself:

```
.als ALIAS als
```

We can now use this command to provide a set of longer names for other key groff commands:

```
.ALIAS MACRO de.ALIAS NUMBER nr  
.ALIAS STRING ds
```

Sure, your old-time, hard-core troff jocks will gnash their teeth at the syntactic sugar. But the rest of us will have an easier time figuring out what in Sam Hill some macro is doing when we get back to work on it after a long and pleasurable weekend—or some other lapse into real life.

Macro Markup

Listing 1 demonstrates an obligatory “Hello, world!” program implemented with groff macros. In it we can see that groff offers commands for creating user-defined variables of type number and string, and that these can be used within the macros we develop. Note the addition of more command aliases and number registers in this more practical example:

Listing 1. “Hello World Program”

```
.\"(this is a traditional troff comment).\"  
\# (this is a gnu and improved comment!)  
\#  
\# define additional aliases:  
.ALIAS BRKFILL br  
.ALIAS SKIP sp  
.ALIAS NEED ne  
.ALIAS TINDENT ti  
\#
```



```

\# define number registers:
.NUMBER #PARINDENT 0.5i
.NUMBER #PARSKIP 0.8v
.NUMBER #ORPHANS 2
\#
\# define user interface
\# tag for a new paragraph:
.MACRO <p> __END__
. BRKFILL \" break and spread pending output
. SKIP \\n[#PARSKIP]u \" paragraph prespace
. NEED \\n[#ORPHANS]v-1v+1u \" orphan control
. TINDENT \\n[#PARINDENT]u \" indent 1st line
.___END__

```

As may be clear from our selection of alias names and in-line comments, the macro definition of `<p>` provides a markup tag for a new paragraph with the following features when formatted by groff:

- completes formatting and forces output of any pending line currently in process
- creates vertical prespace for the paragraph to follow by the value in the **#PARSKIP** variable
- controls orphans by keeping a minimum of **#ORPHANS** lines together
- temporarily indents the first line of the paragraph by the value in the **#PARINDENT** variable

Usage of the new paragraph macro in a document file would look like:

```

.<p>This is my new paragraph. Notice how groff
lets me create HTML-like tags.
.<p>
Here is my next paragraph...

```

Although this example is simplified from a final implementation, it demonstrates how we can export a user interface built up from basic groff macros and create structured markup tags for our documents. Notice also that another macro file could alternatively define the `<p>` macro when publishing the same document to the Web:

```

.MACRO <p> __END__<p>
.___END__

```

A macro name can be any string of any characters, and groff is case sensitive. In our example named `<p>`, the angle brackets have no special meaning; they are just part of the macro name we have devised to simulate an HTML-like tag.

We should, however, expand the definition of the macro given above. Recall that the **.MACRO** command itself is an alias we have given to the raw groff request **.de**. This command accepts two arguments: the first is the name of the macro (here `<p>`); the second is an optional termination label (here **END**). Any arbitrary string may be used to mark the end of a macro definition. We use **END** in these examples, but one could also use `<<<` or `*****`, or any other convention that helps to improve the readability of a collection of macro commands in a file.

The **macro** also demonstrates different forms of comments. The first form (`.\"`) with a period in the first column actually functions as an *undefined request*, with the effect that the entire line is silently ignored. The second form (`\#`) is a GNU groff extension and ignores everything on the line beyond the comment *including the new_line*. The third form (`\"`) can be used on the same line as groff commands and ignores everything on the line beyond the comment, *not including the new_line*. If one were to use this last form of comment (`\"`) on a line by itself, and without a period in the first column, groff would interpret the `new_line` and generally convert it into a space or new line (depending on fill mode) in the output. Unintended spaces and blank lines can be a source of misery and anguish, especially to the novice macro developer trying to figure out why extra space is creeping into the document. Generally, the GNU form of comment is preferable for single-line comments, while the traditional form is required for comments following on the same line as groff commands.

Finally, you probably noticed that while groff command syntax requires a period in the first column, the name of the command itself may be indented to any level on the same line. By using logically indented source code, together with comments, you will greatly improve the readability of the code for yourself and others in future generations of groff users to come. (The preceding comment is a public service announcement as required by the Surgeon General of Computer Science and is based on extensive scientific evidence that such conventions will prolong the life expectancy of your source code.)

Variables and Name Spaces

Groff has a set of about 50 predefined variables called number registers. These are the internal gauges of groff's typesetting machinery. While processing an input file, groff maintains these registers with the current value of such variables as page number, position on page and point size. Number registers are in a separate namespace from strings and macros, and are aliased with their own alias command, as in the following:

```
.ALIAS ALIASNR aIn.ALIASNR      _PTSIZE .s
.ALIASNR      _LEADING          .v
```

In this example, we first alias the command for aliasing numbers, adapting the methodology we used earlier. Then, we alias the read-only registers for the current point size and vertical line spacing, choosing to use the traditional typesetting terminology—"leading"—for the latter. Although not required, the above example also demonstrates the use of a specific convention we follow, to prefix aliases for system variables with an `"_"` (underscore character).

You can, of course, follow your own heart in these matters. But the use of a naming convention may help to distinguish the variables themselves from the names of the commands that set the variables, such as:

```
.ALIAS   PTSIZE   ps.ALIAS   LEADING   vs
```

These might be used in a macro as follows:

```
.MACRO   <fontsize:>   __END__.   PTSIZE   \\$1
.   IFELSE   "\\$2"   \\{
.   LEADING ( \\n[_PTSIZE]u * 120/100 )
.   \\}
.   ELSE   \\{
.   LEADING   \\$2
.   \\}
.   __END__
```

With usage in a document:

```
.<p>A message to the world:
.<p>
.<fontsize:> 18p
Is groff great or what?
```

The first line of the macro sets the current point size to the value of the first argument to the macro. The second line introduces a compound if/else statement, using groff's string comparison syntax for the logical test. If the second argument is empty, the leading is set by taking the value of the point size now in the numeric register **_PTSIZE**, and increasing it by 20%. Otherwise, the leading is set to the value provided by the second argument.

Parentheses in a numeric expression permit the use of spaces within the expression. Otherwise, in the example above, we would need to use the less legible form without any spaces:

```
.LEADING \\n[_PTSIZE]u*120/100
```

Numeric expressions are evaluated simply left to right, there are no operator precedence rules, and parentheses are required to explicitly change the order of evaluation.

All arithmetic operations and number registers are ultimately integer based. Groff internally translates all dimensional measurements into machine units (based on 72,000 units per inch for PostScript devices), providing a functional "illusion" of fractional dimensions and point sizes. This allows us to specify decimal terms such as **8.5i** and **11.5p**, which, in fact, evaluate to 612,000 and 11,500 machine units respectively. Numeric values can be specified in any of the units shown in Table 1.

Table 1. Groff Units

In practice, groff's internal use of integral math can have significant consequences for the macro developer. Consider what would happen if the expression above were instead stated:

```
.LEADING (\\n[_PTSIZE]u * (120/100))
```

Using integer division, the parenthetical term of 120/100 would evaluate to one and the entire expression would then evaluate to the current point size, and not 20% larger as intended.

As it turns out, not all predefined number registers are, in fact, numeric. For example, the name of the current file being processed is in the read-only register **.F**:

```
.ALIAS      MESSAGE tm.ALIASNR  _LINE      .c
.ALIASNR   _FILE    .F
.MESSAGE   Currently processing file \n[_FILE], line \n[_LINE].
```

Although both variables are evaluated using the syntax for number registers, `_FILE` returns the name of the current file as a string. Despite this anomaly, groff permits only numeric expressions in user-defined number registers. The example here, by the way, is one means of inserting debugging messages in your macro file during development. The **.tm** request—aliased above to **.MESSAGE**—sends any text that follows to the standard error stream.

Macros and “Copy” Mode

Observant readers may be wondering why the syntax for evaluating the number registers inside the `<p>` macro have two backslashes (e.g., `\\n[#PARSKIP]u`), rather than one (e.g., `\\n[_LINE]`) as are shown above. The difference is subtle but important.

The reason for using two backslashes inside macro definitions is that we usually don't want the expression inside the macro to be evaluated at the time the macro is first read. Rather, we would like the expression to be evaluated every time the macro is played back. A double backslash is groff's escape sequence for the backslash character itself, providing the means of getting a single backslash to print in your output. When groff is reading in a macro for the first time—in what is called “copy mode”—it interprets everything as it usually does, including escape sequences. So when a double backslash is encountered in a macro definition, groff converts it to the single backslash the sequence represents. Then, whenever the macro is played back, the single backslash remaining is interpreted in the usual manner.

Although we could define macro variables with a single backslash, such as:

```
.MACRO <p>.SKIP \n[#PARSKIP]u
\# etcetera
```

This macro would always execute with the amount of paragraph pre-space specified in the variable **#PARSKIP** at the time the macro was first read. You would be stuck with the same **#PARSKIP** for your whole document. By using two backslashes, as in our original definition of **<p>**, we can dynamically change the **#PARSKIP** variable anywhere in the document and as often as we like, for example:

```
\# user interface for setting parskip:.MACRO <parskip:> __END__
. NUMBER #PARSKIP \\$1
.__END__
\#
\# tighten spacing between paragraphs:
.<parskip:> 0.4v
```

The new setting will now affect the format of all instances of the **<p>** macro that follow.

As we could expect, **groff** offers a useful extension in this area as well. The **"\E"** sequence represents an escape character that will *not* be interpreted in copy mode. So, our **<p>** macro could just as easily be written:

```
.MACRO <p>.SKIP \En[#PARSKIP]u
\# etcetera
```

The **"\E"** sequence will provide the same result as the **"\V"** double backslash sequence.

Pseudo-Arrays and Real Loops

We have shown above that the **groff** syntax for evaluating a number register with names of arbitrary length is

```
\n[anyname]
```

Similarly, the syntax for evaluating other registers is

```
\*[anyname] string\f[anyname] font
\[anyname] special character
```

groff only has scalar variables, lacking compound structures or subscripted arrays. But it is possible to combine definitions and numeric variables in such a way as to fake the effects of compound data types. Here, we will demonstrate a "pseudo-array" that may come in handy for your bag of macro tricks.

Consider the following string definitions for days of the week:

```
.STRING $DAYNAME1 Sunday.STRING $DAYNAME2 Monday
\# etcetera
.STRING $DAYNAME7 Saturday
```

groff provides a number register representing the current day of the week as a numeric value **1** through **7**, and, of course, we alias it again to fit in with our scheme:

```
.ALIASNR _DOW dw
```

Now, we can initialize a variable with the current dayname using the pseudo-array of strings we defined above as follows:

```
.STRING $TODAY \[$DAYNAME\n[_DOW]]
```

Anytime we need the name of the current day in a macro or document, we need only use the string variable **\$TODAY**:

```
.<p>Thank goodness it's \[$TODAY].
```

Human reaction to this message will likely be most favorable when the **_DOW** variable evaluates to **6**.

groff also has an extension that enables the use of looping constructs within macros. Together with pseudo-arrays, this feature gives you considerably more power and flexibility over traditional troff, which only has if/else branching logic. Following the example above, if you needed a macro to create a header for a list of tab-separated columns for each weekday, (Monday through Friday) you might cobble up something like the following:

```
.ALIAS TABSET ta.ALIAS WHILE while
.MACRO <weekdays> __END__
. NUMBER IX 1 1
. NOFILL
. TABSET T .75iC
. WHILE \n[IX]<6 \{\
    \[$DAYNAME\n+[IX]]\c
. \}
. FILL
. __END__
```

In the example above, the **TABSET** command makes use of groff's **"T"** extension for repeating tabs, set here every 3/4 of an inch. The loop test variable, "IX", demonstrates the auto-increment syntax with a number register (the **"+"** sign in the **\n+[IX]** expression). This has the effect of preincrementing the variable, so the first time through the loop **IX** will evaluate as 2, printing "Monday" from the pseudo-array **\$DAYNAME**. Finally, the printed line is terminated with the **\c** escape sequence, to continue output on the current line without inserting the new line that would otherwise be inserted in nofill mode.

groff's implementation of "while loops" includes **.break** and **.continue** statements. These give groff more of the flow-of-control facilities of a complete programming language. Although you probably won't be using groff for solving

multiple regression problems, groff's while loops do make it easier to write macros for, say, printing columns of address labels on precut forms, without using an external processor.

Page Layout and Traps

The basic page layout model for groff is in keeping with groff's minimalism. The only dimensions groff requires are the vertical length of the page, the hard left margin and the horizontal length available for printable lines. Each of these dimensions is set with its own request that we alias below:

```
.ALIAS PLENGTH p\l.ALIAS PGOFFSET po
.ALIAS LNLENGTH ll
```

Usually, though, we need a document to have other layout parameters, such as a top and bottom margin, possibly with running headers and/or footers. All these may be configured using groff's **trap** mechanism in combination with additional parameters and macros for the page transition that we devise.

Let's imagine that we would like a top and bottom margin of one inch for our main body text and a centered page number one-half inch from the bottom of every page. In addition, we want to define these parameters so they will work whether we are using letter, legal or A4 sized paper. The first step is to define our own set of number registers to hold all of our layout parameters:

```
\# parameters with default settings:.NUMBER #PAGELENGTH 11.0i
.NUMBER #PAGEWIDTH 8.5i
.NUMBER #LEFTMARGIN 1.0i
.NUMBER #RIGHTMARGIN 1.0i
.NUMBER #TOPMARGIN 1.0i
.NUMBER #BOTMARGIN 1.0i
.NUMBER #FOOTMARGIN 0.5i
\#
\# layout initialization macro:
.MACRO SET_LAYOUT __END__
. PLENGTH \n[#PAGELENGTH]u
. PGOFFSET \n[#LEFTMARGIN]u
. LNLENGTH \
\n[#PAGEWIDTH]u-\n[#LEFTMARGIN]u-\n[#RIGHTMARGIN]u
.__END__
\#
\# initialize layout with defaults:
.SET_LAYOUT
```

The next step is to write our page transition macros and put them into position with the trap mechanism. The following snippet demonstrates:

```
\# some more aliases:.ALIAS CENTER ce
.ALIAS RIGHT rj
.ALIAS NEWPAGE bp
.ALIAS SETTRAP wh
\#
\# macro for header:
.MACRO MYHEADER __END__
' SKIP |\n[#TOPMARGIN]u
.__END__
\#
\# macro for footer:
```

```

.MACRO MYFOOTER      __END__
'   SKIP |(\n[#PAGELENGTH]u -
\n[#FOOTMARGIN]u)
.   CENTER
\n[_PAGE]
'   NEWPAGE
.__END__
\#
\# position to invoke header/footer:
.SETTRAP 0          MYHEADER
.SETTRAP 0-\n[#BOTMARGIN]u MYFOOTER

```

The example shows two macros, **MYHEADER** and **MYFOOTER**, which define the actions taken at the top of the page (position 0) and at the bottom margin (-1.0i). The syntax in these macros shows the deferred break control character, the apostrophe, ', used with groff commands that would otherwise cause the output to be immediately forced out.

The page layout parameters are defined with default values, but here we will create a user-interface for changing the papersize:

```

.MACRO <papersize:> __END__.  IFELSE "\\$1"letter" \{\
.   NUMBER #PAGELENGTH 792p
.   NUMBER #PAGEWIDTH  612p
.   \}
.   ELSE .IFELSE "\\$1"a4" \{\
.   NUMBER #PAGELENGTH 842p
.   NUMBER #PAGEWIDTH  595p
.   \}
.   ELSE .IFELSE "\\$1"legal" \{\
.   NUMBER #PAGELENGTH 1008p
.   NUMBER #PAGEWIDTH  612p
.   \}
.   ELSE \{\
.   MESSAGE \
Missing or unrecognized papersize,\
file \n[_FILE], line \n[_LINE]
.   \}
.   \" re-initialize layout:
.   SET_LAYOUT
.__END__

```

The enduser can now set the paper size of the document, which initializes the margins accordingly. Putting together the elements we have looked at so far, we have defined an interface to groff that allows the enduser to create a document that looks like this:

```

.<papersize:> a4.<fontsize:> 11.5p
.<parskip:> 0.8v
.<p>
This document is typeset by groff!

```

Conclusions

Just like the **vi editor** and **gcc compiler**, groff is one of the mainstay classics in the standard UNIX/Linux environment. We have seen just a few ways of using groff's extensive macro capabilities to define markup and page layout interfaces that readily turn plain text files into typeset-quality print.

The features covered here are by no means the whole story. For example, groff also includes native facilities for drawing lines, curves, circles, ellipses and polygons with shaded filling. And, this does not even begin to cover groff's suite of preprocessors for graphs (**grap**), pictures (**pic**), equations (**eqn**), tables (**tbl**) and bibliographic references (**refer**). As is customary with GNU and Linux software, groff comes with thorough and high-quality documentation. (See Resources for more information.) And there are, of course, active mailing lists for staying current with groff and interacting with its user community.

This article has been aimed at the creation of short documents, but groff is capable of printing works of any length. In fact, groff is likely the typesetter used in the publication of your favorite O'Reilly title. For tour-de-force examples of groff in action, not to mention some of the best books on UNIX programming ever published, see any of the series by W. Richard Stevens. (The late Dr. Stevens is quoted at the beginning of this article from his colophon to *UNIX Network Programming, Volume 2*, Prentice-Hall PTR, 1999.) Much like the C programming language born of the same era, groff has an enduring and powerful minimalism that continues to lend itself well to typesetting tasks of all sizes. And if you should hear of reports suggesting groff's demise, just remember, some folks used to make similar claims about UNIX as well!

Wayne Marshall (guinix@yahoo.com) is a UNIX programmer and technical consultant currently living in Guinea, West Africa. He enjoys traveling, hiking, photography, Africa, strong black tea, popcorn and baking cookies.

[Blank Line Macro](#)

[Installing PostScript Fonts](#)

[Why Groff?](#)

[Resources](#)

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

SISAL: A Safe and Efficient Language for Numerical Calculations

D. J. Raymond

Issue #80, December 2000

The benefits of SISAL and a call for action.

Back in the misty, early days of computing, famed computer scientist John Backus invented a programming language called Fortran. Given his other accomplishments, most computer scientists have probably forgiven him for this. After all, how was he to know that his invention would grow into a Frankenstein, sweeping away all attempts to replace it with more pleasant and useful tools?

Functional Languages

Later, Backus became interested in *functional languages*. The best way to explain what a functional language does is to show you one that isn't. Let's look at the following C code:

```
int increment_me(int deposit){
    static int balance = 0;
    balance = balance + deposit;
    if (balance < 0) {
        fprintf(stderr, "Balance negative!\n");
    }
    return(balance);
}
```

This C function keeps track of your bank balance by adding deposits to your account and subtracting withdrawals (negative deposits). It also gives a warning if you overdraw your account. Your balance is initially assumed to be zero. Subsequent calls add to or subtract from this balance. In addition, a warning is given if you reach a negative balance.

Increment_me is a function in the C language sense, but it is not a function in the mathematical sense. A mathematical function, like sine or logarithm, gives

the same answer to a particular question each time you ask it. For instance, the sine of 90 degrees is always one. In contrast, **increment_me** stores your current balance internally, which means that the result returned from calling it depends on the results of previous calls.

It is easy to turn **increment_me** into a true mathematical function by rewriting it as follows:

```
int increment_me(int deposit, int balance){
    balance = balance + deposit;
    if (balance < 0) {
        fprintf(stderr, "Balance negative!\n");
    }
    return(balance);
}
```

You, rather than **increment_me**, have to keep track of your bank balance in this version, but at least this C function is now also a mathematical function—it always gives the same answer to a particular question!

What are the advantages of the functional structure of this routine? Suppose **increment_me** is part of a complex accounting system, keeping track of many different balances. In this case, storing a single balance inside the C function would not get the job done since the function would only work for that particular account. Furthermore, if some other aspect of the accounting system needed to access your balance, it could not do it in the original version of **increment_me**, since the balance is buried in the function.

What is the downside of the new version of **increment_me**? Suppose **increment_me** were ten levels down in a stack of C function calls. Then, it would be a considerable nuisance to transfer the balance in which you are interested downward through this stack. One might be tempted to use the C language global-variable capability instead:

```
global int balance;int increment_me(int deposit)
{
    balance = balance + deposit;
    if (balance < 0) {
        fprintf(stderr, "Balance negative!\n");
    }
    return(balance);
}
```

This way, anyone anywhere in the system could insert the balance for a particular account in the global variable **balance** and initiate the cascade of calls that eventually results in **increment_me** being called and the balance being incremented.

What is wrong with this way of doing things? Non-locality. If something goes wrong, do you blame it on a bug in our humble function, or is the error

somewhere else in the system? If you have ever dealt with this kind of a program, you know that it can be very difficult to debug.

Functional programs avoid these problems by following three rules:

1. Functions do not remember anything about previous invocations.
2. Functions do not change their calling arguments.
3. Functions do not access global variables.

Such functions are said to be free of *side effects*. Functional languages are languages that enforce these restrictions. Examples of partially functional languages are the venerable Lisp language and its derivatives such as Scheme, and the language ML. Haskell and the original version of Lisp are purely functional.

SISAL and Parallel Computing at LLNL

The advent of parallel computers introduced a whole new set of problems to the world of computing, the main one being how to divide up a task so that multiple processors can make progress on it simultaneously. Attempts to devise automatic parallelizing compilers for ordinary languages have met with mixed success. The objective is to define sub-tasks so that the flow of data between them is minimized since typically the communication bandwidth between individual processors in a parallel computer is limited. This is difficult in ordinary languages since the flow of data is immensely complicated by the existence of global variables and side effects—witness our third example of **increment_me**.

Enter a team of computer scientists led by James McGraw of Lawrence Livermore National Laboratory. In 1985 they invented a functional language called SISAL for the specific purpose of parallelizing scientific numerical calculations. The functional nature of the language allowed the compiler to trace the flow of data through the program, and therefore, to make intelligent decisions as to how to split up the work between processors in parallel computers.

Functional behavior on the procedure level, as in our procedure for incrementing or decrementing one's bank balance, was still insufficient for their purposes. In order to obtain sufficiently fine-grained information about data flow, they needed to make *every program statement* functional in a mathematical sense. To understand the implications of this requirement, notice that commonly used programming statements of the form:

```
a = a + 1
```

would be illegal in such a language since in mathematics a variable such as a cannot be both the input and output in an explicit function definition. One would have to write instead:

```
b = a + 1.
```

The implication is that variables can have a value assigned to them once and only once. That's why the language is called SISAL, the Streams and Iteration in a Single Assignment Language. The bit about "single assignment" refers to the above characteristic of the language. The obvious challenge is how to do iteration in such a language. "Streams" refers to an abstraction that was meant to be used in the language for input-output processes, but this was never developed.

Everything Is a Function Definition

The single assignment nature of SISAL makes its use very different from commonly used languages such as C and Fortran. It also gives SISAL a steep, though short, learning curve, especially for those used to conventional programming techniques.

The key is to stop thinking in terms of a computer as a set of memory cells alterable at will by a sequence of instructions. Instead, think of SISAL as a nested sequence of mathematical definitions. One example is familiar to C programmers. A normal if statement in C might run something like this:

```
if (i > 0) { a = i;
}
else {
  a = -i;
}
```

We recognize this as *assigning* the absolute value of i to a . However, an alternate form of the C language if statement is the assigned if, which accomplishes the same thing:

```
a = if i > 0 ? i : -i;
```

This is much closer to the spirit of the corresponding SISAL statement:

```
a := if (i > 0) then i else -i end if;
```

Returning to the two types of C language ifs, notice that the first form is subject to a possible error which, though it is syntactically correct, is probably not what you want to do; just drop the **else {a = -i;}** clause! This is still legal C, but it leaves the variable a undefined when $i \leq 0$. Such an error is simply not possible in SISAL, or in the second type of C language if, because all possibilities must be considered for the statement to be syntactically correct.

The assigned if in C is quite limited in power. For instance, if several assignments need to be done, then several assigned ifs need to be created, and the logical test is repeated for each assignment. In SISAL one simply does the following:

```
a, b, c := if (i > 0) then          expression for a,
          expression for b,
          expression for c
        else
          alternate expression for a,
          alternate expression for b,
          alternate expression for c
        end if;
```

Alternatively, if it takes more than a simple expression to compute a result, then something like the following construct may be used:

```
a := if (i > 0) then          let
      x = ...;
      y = ...;
    in
      if (x > y) then x else y end if
    end let
  else
    ...
  end if;
```

The **let...in...end let** construct allows us to replace a simple expression with an arbitrarily complex calculation.

How do we get around the single assignment provision in iteration? Let's first see what the problem is. A loop in C that computes the cumulative sum of the array **a** might take the form:

```
b[0] = a[0]; i = 1;
while (i < n) {
  b[i] = b[i - 1] + a[i - 1];
  i = i + 1;
}
```

Note the presence of our now-banished friend **i = i + 1**. All loops in C and similar languages have some construct like this, whether it is implicit or explicit. However, as we have noted, such a construct is illegal in SISAL. In addition, the array **b** is assigned to **n** times, which also violates the single assignment nature of SISAL. In SISAL we do the following:

```
b := for initial      i := 0;
     bvalue := a[0];
     while (i < n - 1) repeat
       i := old i + 1;
       bvalue := old bvalue + a[i];
     returns
     array of bvalue
  end for;
```

The for initial clause is executed once, setting up the initial definitions of **i** and **bvalue**. Then the loop is begun. Each time the loop hits the while statement, the

variable **i** is renamed **old i** and **bvalue** is renamed **old bvalue**, leaving the names **i** and **bvalue** unassigned and ready to be reused. Finally, after the looping is finished, the **returns** clause gathers up the values of **bvalue**, including that from the **for initial** clause, and returns them as an array. While this way of doing loops is really a bit of a slight-of-hand for a single assignment language, it again satisfies the goal of keeping data dependencies explicit. The existence of an old clause clearly indicates that values produced in a loop depend on values generated in the previous iteration of the loop.

An alternate looping construct is available if computations in later iterations do not depend on results from earlier iterations. For instance, to multiply every third element of an array **a** by **-1**, SISAL does the following:

```
newa := for i in 0:n - 1      a1 := if (mod(i,3) = 0) then
    -a[i]
    else
    a[i]
    end if;
returns
  array of a1
end for;
```

If a loop can be cast in this form, it makes parallelization trivial since each branch of the loop can be executed by a different processor with no intercommunication between processors.

A complete SISAL function takes the form:

```
function fun_name(argument_list returns return_list) expression, expression, ...
end function
```

The argument list specifies the names and types of all arguments. For instance, an argument list containing two integers named **i** and **j** followed by a real array named **a** would be specified **i, j: integer; a: array[real]**; A return list consisting of two real arrays would look like **array[real], array[real]**. The expression can be simple or complex, involving **let**, **for**, **if** or other statements. The number of expressions in the body of the function must match the number of returned values.

In addition to the basic types **boolean**, **character**, **integer**, **real** and **double_real**, there are compound types **array**, **stream** (like an array, but elements accessed sequentially), **record** (like a C structure) and **union**. Each of the compound types can, with minor restriction, consist of either simple or compound types. For instance, a two-dimensional array in SISAL is simply an array of arrays.

All variables are dynamically allocated, and variable types are determined from context. User-defined types are declared with statements like:

```
type oner = array[real];type complex = record[u: real_part; v: imag_part];
type complex_array = array[complex];
```

SISAL Prevents Many Mistakes

SISAL turns out to be a very safe language to use. Compared to C, I invariably find that I am much farther along the road to a working program with SISAL when the code first compiles successfully. SISAL has many conventional safety features, such as strict type checking, prototyping of external functions and array bounds checking. Arrays carry information with them on array length and starting value of the array index. Multidimensional arrays are actually arrays of arrays, so bounds checking works individually for each index as well as for the array as a whole. Since bounds checking slows program execution, it can be turned off when debugging is finished.

SISAL derives much additional safety from its functional and single assignment natures. As I showed above, undefined alternatives in if statements are impossible in SISAL. I find that this has the effect of forcing one to think through an algorithm very thoroughly as one is coding the program. Thus, coding a SISAL program often takes more time than coding the first round of the equivalent C program, but the extra effort pays off a hundred-fold in the debugging stage and in a better understanding of the calculation.

Speaking of debugging, conventional debuggers such as **gdb** don't work well with SISAL, so the developers of the language provided its own debugger, **sdbx**. In addition, the "peek" function allows one to examine the value of variables during program execution.

In many ways SISAL is easier to debug than conventional languages, due to its functional and single assignment natures. However, certain behaviors take a bit of getting used to. For instance, in the function:

```
function polar(x, y: real; returns real) let
  r := sqrt(x*x + y*y);
  theta := atan(y/x);
in
  r
end let
end function
```

all traces of the variable **theta** disappear. Since the computation of this variable contributes nothing to the final answer, **r**, the statement generating **theta** is dead code and is removed by the SISAL optimizer. This gives rise to an iron-clad rule in SISAL: if a calculation doesn't contribute to the final result, it is ruthlessly eliminated. If the value of **theta** is really needed in the above function, then it should be included in the output by changing the function definition to **...returns real, real)** and the **r** in the in clause to **r, theta**. Otherwise the computation of **theta** should be deleted from the code.

Execution Speed and the OSC Compiler

If this were all there were to SISAL, it would be an elegant, but useless language. Since new variables are created for every assignment, any significant SISAL program would be one giant memory leak. This is particularly significant when it comes to arrays. In SISAL, a completely new array is apparently created each time an element of an array is changed!

I say apparently, because the back end of LLNL's SISAL compiler, **osc**, is quite good at optimizing away the horrid inefficiencies of single assignment semantics. This, in fact, is the major contribution of the SISAL development team. **osc** first converts the SISAL code into an intermediate form called "IF1". This intermediate code is then extensively massaged before being converted into either C or Fortran code. Thus, SISAL is really a fancy C (or Fortran) preprocessor, which means that it is quite portable, in nonparallelizing form to architectures with a good C compiler, such as **gcc**. Virtually any UNIX or Linux system will compile SISAL code in single processor mode. The developers of **osc** claim that optimized SISAL code typically executes within 20% of the time required by the same algorithm coded directly in C or Fortran, and my experience with the language supports this claim. Interestingly, a fast Fourier transform routine written in SISAL by John Feo of LLNL actually ran *faster* in parallel mode on a Cray computer than Cray's own parallel fast Fourier transform routine, even though it was slightly slower in single processor mode.

Creating parallel SISAL code is somewhat more difficult. SISAL needs a C or Fortran compiler and an underlying operating system that implements parallelism in order to produce parallel code. Given the wide variety of parallel system types, and the variety of custom interfaces to these systems, porting SISAL to a new parallel system is not trivial. Thus, even though Linux now has symmetric multiprocessing available, I have not attempted to make SISAL use it.

Scientists and engineers often write computer programs that take advantage of libraries of useful code. These libraries are almost always written in Fortran. The importance of such libraries is often cited as a reason for not migrating away from Fortran.

SISAL bows to this reality by providing interfaces to both Fortran and C. The connection goes both ways; SISAL can call C and Fortran routines, and C and Fortran can call SISAL routines. The Fortran interface is more developed than the C interface, which is somewhat dismaying to Fortran-averse folk like me, but the good news is that C can easily be used to emulate Fortran code, thus allowing C programs to take advantage of the Fortran interface.

SISAL Input-Output

The natural way to do input and output in a functional language is to funnel input through the argument list of the top-level function and funnel output through function return values. The SISAL developers took this conservative approach, designing a special input-output system called FIBRE. FIBRE represents primitive types, arrays, records, etc., in a special ASCII form. This form also contains information about the data, such as array size and record contents, in addition to the data itself.

Though having a certain elegance, this format is severely lacking if, for instance, you are trying to decipher the output of a global weather prediction model.

It is here that the SISAL interface to C and Fortran comes to the rescue. This interface can be used to perform any type of input and output desired by the user. However, this solution is somewhat inelegant, as it requires the user to delve into the grungy details of the inter-language interface. This can be a daunting experience to the average scientist or engineer, and it defeats the purpose of providing safe, efficient and easy-to-use computational tools.

This problem can be solved if flexible interfaces can be written between SISAL and widely used standard data formats. I have written just such an interface to the NetCDF library. NetCDF is a format and an application programming interface for storing and retrieving gridded numerical data. Developed by the UNIDATA program of the University Corporation for Atmospheric Research, it is rapidly increasing in popularity in the fields of meteorology and oceanography and is starting to spill over into other fields as well. Many tools currently exist to manipulate and display data in this format, with more appearing all the time. The NetCDF library is open-source software and is available from UNIDATA's web site. However, if you are lucky, your Linux distribution already has a prepackaged version of NetCDF available for installation—for instance, it is available in the Debian GNU/Linux distribution that I use.

Listing 1 shows a complete SISAL application (solution of the heat transport equation in a long rod). NetCDF is used to write out the results. The time evolution of the temperature pattern along the rod is shown in Figure 1 using the GRI graphics package of Dan Kelley. (See the July 2000 issue of *Linux Journal* for an article on GRI.)

[Listing 1. Sample of a Complete SISAL Application](#)

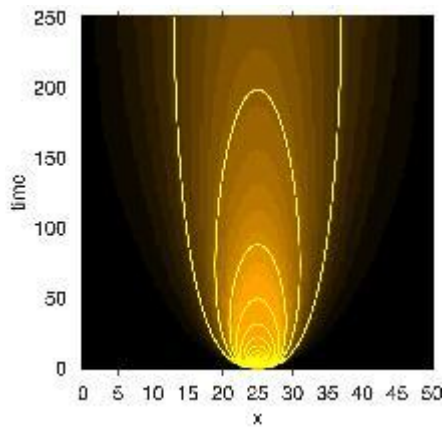


Figure 1. Output of the Sample SISAL Program

The Fate of SISAL at LLNL

Around 1990 the SISAL folks at LLNL decided that SISAL was ready to be smoke-tested outside of the confines of Livermore and offered people time on a Livermore Cray in exchange for test-driving SISAL. I took them up on their offer and quickly became enthralled with the language. At the time I was working on Sun workstations, but eventually began a conversion to Linux on PCs. SISAL wouldn't compile on Linux at the time but the problems turned out to be relatively trivial, so with the help of Pat Miller at LLNL, we ported SISAL to Linux.

As a real smoke test of SISAL, I wrote a tropical weather research model that I still use, and that currently consists of 4,600 lines of SISAL code. I consider the osc compiler to be relatively robust, though undoubtedly there are still bugs that I haven't exercised.

The current version of my model talks to the outside world using a C interface to my Candis (C language analysis and display) package. Developing this interface contributed to the ease with which I was able to implement the NetCDF interface to SISAL.

Unfortunately, the SISAL project was canceled in 1997. However, with James McGraw's help, an open-source copyright was placed on the osc compiler source code. As one of the few current SISAL enthusiasts, I now maintain the web site for SISAL.

What Needs to Be Done?

The SISAL project represents a relatively rare event in recent times, namely a significant effort to focus the talents of computer scientists on an issue of real importance to physical scientists and engineers. The result is an exceptionally interesting computer language for high-performance numerical computing, along with the hooks needed to make it truly useful to the targeted clientele.

Though the SISAL project ultimately failed to make a significant impact at LLNL, its transfer to the world of open source gives it another chance. What is needed are people willing to try it out and ascertain whether it will help them get their work done more effectively than the current alternatives. Also needed are knowledgeable people who are willing to fix problems and build on what has already been accomplished. An obvious extension would be to make SISAL use Linux SMP for parallel computation. A more ambitious objective would be to extend it to the distributed memory model of parallel computing, perhaps enabling it to make use of the Message Passing Interface (MPI) commonly used on massively parallel computers such as Linux Beowulf systems.

Resources



Dave Raymond (raymond@kestrel.nmt.edu) is a professor of physics at New Mexico Tech in Socorro, New Mexico. Dave does research in atmospheric physics and teaches physics. He has been involved with computing since 1966 and with Linux since the original SLS distribution came out. His computers at work and at home all run Debian Linux, though his younger daughter is reputed to have an iMac hidden in her bedroom.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

ATF Jubilee Edition

Reuven M. Lerner

Issue #80, December 2000

To celebrate his 50th column, Reuven casts his vision to the future of web development and suggests some current trends that will affect how the job gets done.

Welcome to the jubilee installment of At the Forge. This is the 50th column that I have written for *Linux Journal* (or for SSC's short-lived *Websmith* magazine) since early 1996. Over the last few years, we have explored a large number of Web-related technologies, techniques and applications, ranging from simple CGI programs to sophisticated database-backed applications written using `mod_perl`.

This month, I want to spend a bit of time prognosticating, looking into the future of web application development. On the one hand, things have never been more exciting for web application developers; the technology continues to advance at a remarkable rate, making it easier and easier to create sophisticated applications. At the same time, the increasingly crowded field of embedded programming languages, application servers and database adaptors makes it harder to decide which technology is most appropriate.

Because this column describes where I believe web technologies and application development are headed in the coming years, it should also serve as a sort of guideline for what future issues of ATF will contain. You can think of this month's installment as an indication of where my consulting firm is headed professionally, and thus what you can expect me to suggest and describe in the year (or more!) ahead. Since this is *Linux Journal* and Linux is my company's primary server platform, I will focus here on items that run with Linux and, preferably, those that are free software.

Where Have We Been?

Web application development began soon after the Web itself was formed. Ever since the first dynamically generated content was sent to the first browser—an act which predates the CGI standard, to say nothing of Netscape, Internet Explorer and Apache—programmers have been designing increasingly sophisticated applications for use on the Web.

CGI, or the “common gateway interface”, soon arrived on the scene. CGI got its name because dynamically generated content was originally a means to give a web interface to non-web applications. With the advent of CGI, it was suddenly possible to create portable server-side programs. Most web applications continue to be written using CGI, because of its simplicity and its extreme platform-independent nature, as well as the fact that web space providers can give their clients CGI access without endangering the server's stability.

You can write a CGI program for any web server, in any language, on any operating system, and be virtually guaranteed that it will work. However, CGI has a number of drawbacks. In particular, it requires that the web server spawn a new process for each HTTP request aimed at a CGI program. In other words, a web site that receives 100 hits/minute is spawning more than one new process every second.

By itself, this should not scare you. After all, a basic Linux box should be able to handle the creation of one new process each second, right? However, the size of the new process, as well as the speed with which it starts up, are both important factors.

Perl, my programming language of choice for the last few years, has proven itself as a powerful means for creating CGI programs. The CGI.pm module provides an amazing array of functions that do nearly everything you would ever want from a CGI program (as well as a number of things that I would never consider doing). Moreover, Perl includes a powerful pattern-matching engine, along with modules that handle most popular Internet standards and protocols. The DBI (database interface) module has proven to be an additional boon, making it easy to include the output from an SQL query in a dynamically generated page.

However robust, flexible and secure Perl might be, the CGI standard was never designed for producing a large volume of dynamically generated pages on the fly. Each invocation of a CGI program written in Perl forces the computer to create a new process, load Perl into memory, load your program into memory, compile your program into Perl's internal opcodes and then, finally, interpret it using the Perl run time mechanism. This all takes time and means that CGI

programs will not scale well over the long term. Indeed, it does not take a lot of concurrently running CGI programs to bring a typical server to its knees.

At the same time, CGI has been successful because it's so easy to use. With no other API can you write a “hello, world” program as simple as the following:

```
#!/usr/bin/perl -wT    use strict;
use CGI;
my $query = new CGI;
print $query->header("text/html");
print $query->start_html;
print "P>Hello, world!</P>\n";
print $query->end_html;
```

Embedded Languages

In the last few years, advanced web development has become a specialty of its own, requiring that programmers learn something about administering systems, networks and databases, while keeping in mind good programming and security practices.

There are three current trends in the world of web development which are beginning to improve things dramatically for users as well as for developers. When the three are used together, they are often called an “application server”.

The first trend is architectural, moving away from single-shot CGI programs and toward programs that are cached within the web server or another environment. The only reason we use CGI programs for creating dynamic content is that the web server itself cannot create our custom HTML files on the fly. We could theoretically write a new module for Apache in C, and compile that into our configuration—but that is far too much work in most cases, and the time savings is not worthwhile.

However, there is a middle ground between putting the custom code inside of the server and leaving it completely outside. What if we put an entire programming language inside of the server, making it possible for us to add new functionality in that language? If the language is interpreted, then we can modify and debug our new functionality without having to recompile or restart the server.

This is the idea behind **mod_perl**, which embeds a copy of Perl inside of Apache. It gives us a Perl-language interface to Apache's internals, making it possible to access and modify anything having to do with the request object. Everything that a C-language module can do for Apache can also be done inside of mod_perl, from creating custom response handlers to changing the way in which authentication is performed.

In stark contrast with CGI programs, where Perl compiles the program once, executes it once and exits, `mod_perl` caches a compiled version of the program and then executes that repeatedly. (This can sometimes cause extreme memory growth and requires that programmers be especially careful.)

While `mod_perl` was once the only embedded language module for Apache, others have come along recently. **`mod_snake`** does for Python what `mod_perl` does for Perl, making it possible to write custom Apache handlers in Python. There is even a **`mod_tcl`**, which provides embedded Tcl inside of Apache, although I am not aware of any sites that are using its capabilities.

Another open-source web server, AOLServer, has long contained an embedded Tcl interpreter. Tcl procedures can thus be used to create dynamic output, connect to a relational database and make code conditional—all within the server itself, without having to go to an external CGI program.

If you would prefer to use Python over Tcl, a beta version of PyWX (Python Web Extensions) recently became available. PyWX provides a Python API to all of the Tcl functions that AOLserver normally provides. While this makes PyWX incompatible with most of the Tcl code available for AOLserver, it does make it easier to perform certain functions, given the wealth of Python modules available on the Web.

Mixing Code and HTML

The second trend involves embedding code inside of HTML. Microsoft's Active Server Pages are perhaps the best example of such a practice, but there are plenty of other ones as well. On Linux, we can choose from a variety of different systems, ranging from Java Server Pages (JSPs), HTML::Mason (which works with `mod_perl`), PHP and ADP.

I have worked with Java off and on since it was first introduced, and was long ago convinced that it would be nice to spend some time working with the language. Like many other people, I was turned off by the idea of applets, which were slow, insecure and buggy. However, in recent years, server-side Java has become increasingly prevalent. Each Java “servlet” is a class that runs inside of a Java Virtual Machine (JVM). Servlets can accomplish all of the things that we might want to do when generating dynamic content—they can talk to databases with JDBC, they can retrieve and modify HTTP headers and they can produce responses whose content depends on the user's preferences.

JSPs make it easier to work with servlets by assuming that everything is literal HTML except for what is contained within `<%` and `%>`. When a JSP is invoked from a web browser, the JSP is compiled on the fly into a Java servlet, which is in turn compiled into a Java `.class` file. This `.class` file is loaded into the servlet

engine, executed and kept around for future invocations. JSPs and servlets can use Java “beans”, objects that can be used to model persistent behavior and to implement the “business logic” that sits in the middle of most modern three-tiered web applications.

mod_perl is a very powerful tool for creating Apache handlers, but it can sometimes force you to work at too low a level. For this reason, a fair number of Perl modules exist that allow you to mix Perl code and HTML in some way or another. HTML::Mason, which I profiled in a series of articles earlier this year, is the system that I prefer because of its simple syntax and the way it allows templates to incorporate one another. While at YAPC::Europe in London this fall, I saw a demonstration of the Template Toolkit, which seems to be similar to HTML::Mason in its philosophy, except that it adds the notion of “plug-ins”.

While Java and Perl are general-purpose programming languages that are well-equipped for server-side web programming, PHP is a language designed expressly for creating dynamic web pages. PHP includes a large number of functions for working with a variety of different kinds of files, databases and Internet standards. Recent versions of PHP even allow you to work with Java objects, and a CORBA adaptor is expected to be released in the near future. At the same time, PHP requires recompilation every time you change the included feature set; there is no notion of dynamically adding or deleting modules from the system. If you install PHP before deciding that you want to work with PDF files, you may find yourself recompiling it simply to add such features.

Users of AOLServer have a similar system at their disposal, known as ADP (“AOLServer Dynamic Pages”). An ADP page allows you to mix Tcl with HTML, where the Tcl can use any of a number of special procedures that are defined within AOLServer. You can thus create an ADP page that retrieves information from a database, interprets the contents of an HTML page returned from another server or simply performs calculations based on a user's HTML form inputs.

Pooled Database Connections

The third trend in the world of server-side programming is the issue of persistent database connections. Database servers were originally designed to handle one connection from each user every day, rather than once per minute or once per second. Consider this: if a CGI program connects to a relational database server once per second, you are exercising the connection mechanism more than 86,000 times what it was originally intended. In some cases, this does not mean very much, but there are many databases for which each connection is an expensive operation.

One solution is thus to open a database connection when the server first starts up and reuse that connection each time a program needs to contact the database. This is roughly what the Apache::DBI module does when working with Perl, Apache and mod_perl. Each time you disconnect from a database with `$dbh->disconnect`, Apache::DBI silently ignores your request and keeps the connection around for the future. When you call `DBI->connect`, Apache::DBI looks at the connection string and tries to reuse an existing connection before starting a new one. Since each Apache process services only one HTTP request at a time, each process thus needs only one database connection. The savings from this connect/disconnect sequence can be substantial. At the same time, it means that every child Apache process needs its own database connection, which can lead to dozens or hundreds of simultaneous connections on a heavily loaded server.

AOLServer cuts down on the number of database connections by using multiple threads rather than multiple processes. Because threads exist within the same process, they can share data. AOLServer takes advantage of this to create a small pool of database connections, choosing a connection at random and handing it to the thread handling an HTTP request as necessary. Database connections are not dedicated to a particular thread and can be shared as necessary, reducing the number of connections that the server must open with a database.

Working with Java servlets and JSPs requires a different model altogether. The Jakarta-Tomcat servlet/JSP implementation normally exists outside of a web server, meaning that they're always on Tomcat process, regardless of how many Apache child processes are on the system. Within that Tomcat process, there may be any number of servlet threads executing concurrently. Normally, servlets and JSPs (and Java beans, which JSPs and servlets can use to provide persistence and/or a high level of abstraction) connect to a database using JDBC. But JDBC does not automatically provide connection pooling; while JDBC 2.0 does provide this capability, it is not completely automatic, and not many JDBC 2.0 drivers exist as of this writing.

Other languages take a different tack. For example, database drivers for PHP allow persistent database connections but require that the programmer ask for them. That is, you can connect to a PostgreSQL database with `pg_connect`, or you can create a persistent connection to PostgreSQL with `pg_pconnect`. The onus is placed on the author of a database driver to provide two different access functions, and on the PHP programmer to use the appropriate function for his or her needs.

Of these, I find AOLServer's technique of persistent, pooled connections to be the most elegant, since it works for all languages --although that is almost

always going to be Tcl—and scales extremely well. mod_perl's Apache::DBI is a great solution for Perl programs, especially since it means that individual Perl programs and modules do not need to be changed in order to take advantage of the persistent connections. The fact that Apache::DBI only provides persistence, and not pooling, is a direct result of Apache's multiple processes; it is probably safe to assume that Apache 2.0, which will support threads as well as processes, will come closer to AOLServer's model when it is released.

JDBC's pooling is good, particularly after it seemed that everyone was writing their own class for connection pooling. However, it will only work for Java servlets and will not help on a server that requires a pool for multiple services, such as mod_perl and JSP. PHP's system is perhaps the crudest because it provides neither a standard database API, nor a means for database drivers to pool connections automatically, nor a way for programs to take advantage of those connections. However, the persistence does work and can certainly result in a significant speedup.

Where Are We Going?

While I generally dislike the term “application server” for its ambiguity, it is clear that this is the direction in which the Web is moving. No longer will you design applications by writing one or more programs that exist on their own; rather, you will write a program using a set of objects and modules provided by the application server and into which your application fits naturally. In many cases, you can create relatively sophisticated applications with a minimum of work, simply because someone else has done the majority of the work for you.

Of course, this means that we're increasingly seeing operating systems as the underlying layer for an application server, where the latter is the truly important element. Just as a client-side application author must decide whether to write for Windows, UNIX or Macintosh, web application developers must increasingly decide which application server they prefer to use. As with operating systems, it is very difficult to move from one application server to another. This means, unfortunately, that choosing an immature, slow or difficult-to-modify server may be painful in the future. Even application servers that conform to the same standards and use the same language, such as Enhydra and ATG Dynamo, provide different objects and functionality and make it difficult to move from one to the other.

To a free software devotee like myself, this means that open-source application servers are at least as important as open-source operating systems. Luckily, there are a number of open-source application servers available for download from the Internet. They differ radically in their operation and functionality, but I must admit that I have had only a little exposure to each of the following technologies. While I hope to learn more about them in the coming months, I

mention them because it is clear that web developers need to learn more about all of them.

Perhaps the best-known application server platform is Zope, which comes with a number of parts and isn't well understood. Zope is an object database, a templating system and even a basic content management system. I have not yet had a chance to play with Zope in a serious way, but the little that I have read and heard about it seems very impressive, particularly if a module is already available for the particular functionality you need.

Another application server that has been getting a lot of publicity is the ArsDigita Content System, written and maintained largely by the ArsDigita consulting company and released under the GNU Public License. One main problem with ACS has been its dependence on Oracle as a database; while Oracle is an excellent database product, it is both expensive and its source is quite closed. A volunteer effort known as OpenACS has been working to solve this problem by porting the ACS software to use PostgreSQL as a database. The software is not quite complete but does include a great deal of functionality and will undoubtedly improve over time.

XML has been a hot topic in the Web community for several years now, but only in the last six to nine months have we begun to see its widespread adoption. XML describes content semantically, completely ignoring the way in which it should be displayed.

Enhydra is a Java-based application server that seems similar to Zope in many ways, except that it works with XML, Java servlets, JSP and Enterprise Java Beans. Enhydra appears to be quite complex, but also provides a large framework on which to create applications.

If you want to work with XML, then you might also want to look at the Cocoon and AxKit projects. Cocoon, which is sponsored by the Apache Software Foundation, is working on a Java-based server for XML data. AxKit provides XML-based content generation using Perl, making it possible to separate programs from content, and content from graphic design, using XML, XSL and XSLT along with Perl.

Finally, I should mention Oracle's latest entry into the world of application servers, its Internet Application Server (IAS). IAS is a module within Apache that works with a Java run time system, Enterprise Java Beans, JSP and JDBC, along with Oracle. As of this writing, the system is largely new and untested. Of course, Oracle does not provide access to its source code. At the same time, IAS runs under Linux and may well be a popular choice among Oracle users and administrators.

Where Am I Going?

Until now, the majority of my consulting work has been with Perl, which I still find to be a powerful language for working with the Web. Indeed, I used to tell people that about 80% of my work was with Perl, with the other 20% a mixture of Java, Python, Tcl and C.

But with the recent explosion in web programming environments, and with the shift to application servers, I (and my staff) have had to change direction somewhat. In many cases, we will prefer to use Perl, especially when coupled with `mod_perl` and `HTML::Mason`. However, we are increasingly using Java servlets and JSPs for projects, particularly with the Tomcat servlet/JSP engine and with the PostgreSQL database. Our familiarity with `mod_perl` is naturally leading us to look at `AxKit`, while servlets are forcing me to take a serious look at `Enhydra`.

We have already begun to use ACS for some large jobs, in no small part because of the very large number of working applications—not just underlying tools—that come with it. Moreover, the fact that ACS is free software and works with Linux makes it easy to work with since we can rely on the community to provide functionality, documentation, testing and bug fixes.

In other words, there are lots of technologies out there, many of which have sprung up only within the last year or so. As I complete this 50th ATF column and look toward the future, I see a world of possibilities and opportunities for web developers, particularly those who believe in free software and use Linux. The coming years promise to be exciting and interesting for web developers—and over the coming months and years, I hope to share with you my experiments and experiences in working with such tools, as well as sample pieces of software that can be used with them.

Resources



Reuven M. Lerner owns and manages a small consulting firm specializing in Web and Internet technologies. As you read this, he should be (finally!) finishing *Core Perl*, to be published by Prentice-Hall later this year. You can reach him at reuven@lerner.co.il, or at the ATF home page, <http://www.lerner.co.il/atf/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Saucy Administration Tools

Marcel Gagné

Issue #80, December 2000

This month, our chef presents several ways for sysadmins to stay on top of their many, many tasks—including the world's smallest corkboard.

*Mais oui, François, a system administrator is very much like a restaurant owner and chef. There is far more to successful administration than simply making sure people can log on, just as there is more to the restaurant business than bringing plates of food to tables. We must constantly be on the lookout for exciting new recipes to tantalize our guests' palates. Our wine list must remain the finest in the land. And we must continue to provide impeccable service. This does not even take into consideration the dreadful dealings with *la banque, non?**

When we cook with Linux, we draw on as varied a skill set as when we prepare a new and exciting crêpe. The customer, *mon ami*, drives both the restaurateur and the system administrator.

Speaking of customers, our guests are here, François. *Bonjour, mes amis*. Come in. Sit down. François! *Du vin!* I think the 1982 Pauillac would be an excellent choice, *n'est-ce pas?* Today, *mes amis*, François and I were chatting about system administration in honour of our current issue. As you are well aware, part of the system administrator's job is keeping several different things on the go at all times. This includes system issues, software, hardware and people as well. Ah, people...the calls, the hustle, the bustle, your *raison-d'être*.

How can we make sense of all this? How can even the most highly trained system administrator keep up with the furious pace of calls that never stop coming? These are wonderful questions for which I hope to provide some answers.

For those of you using the K Desktop Environment, you may have already noticed a nice little tool called the Personal Time Tracker. You'll find it under the Utilities menu but can also execute it from the command line by simply typing **karm**. What the program allows you to do is create tasks that you can start, pause and restart at will. For system administrators who are working on multiple things at once (that would be all of us, *non?*) this is a way of keeping the clock ticking on various projects. Some of you will need to bill for services while others may just need to know where the time is going. Have a look at Figure 1 for a snapshot of KArm in action.

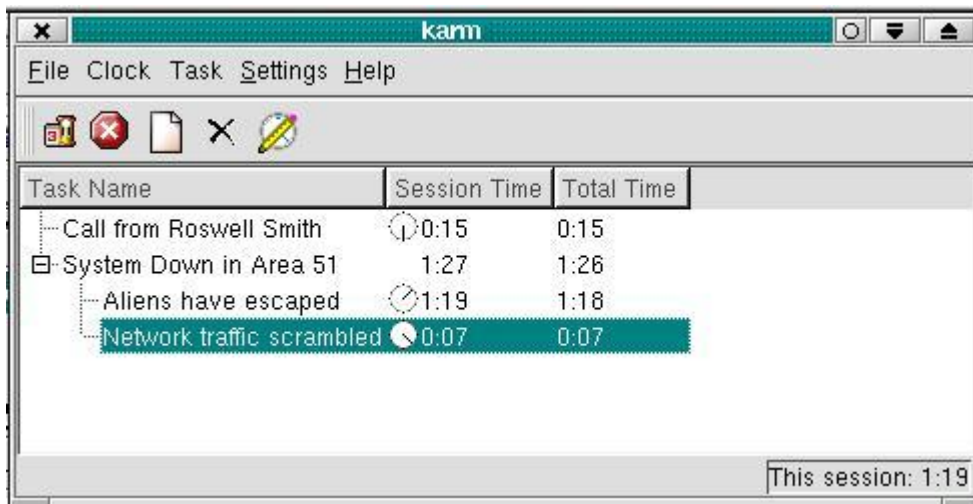


Figure 1. KArm Tracks the Seconds

Incidentally, the KDE time tracker is available in both the 1.X release and 2.X. However, for those of you using the new KDE 2.X release, the program does add some refinements like sub-tasks that track individually and cumulatively based on the master task. It is simple and easy to use.

Many times, I find that my thoughts seem altogether too random. There are a million things to do and finding some way to organize this mental traffic jam sometimes calls for a *different* approach.

ThoughtTracker is an interesting little program in that it lets you create seemingly random lists and cross-link them to other ideas so that you start to create some kind of organization out of those ideas. In some ways, it's a *strange* little program, but it may be worth a look. Think of it as a kind of brainstorming tool. The author is Marco Götze and, while I am at it, I would like to point out one other little program I found on his web site. Now, I did talk about little yellow stickies as a means of organization in the past, but Marco's take on it is interesting and maybe just a bit more fun.

Marco has also created a cool little WindowMaker applet (which I've run from both KDE and WindowMaker) called **wmpinboard** (see Figure 2). This is essentially a very small dock applet that looks like a corkboard. When you click

on the ToDo label on the applet, you get colorful little notes that you can fill out and pin to the board. It's cute; it works. It takes up next to nothing in terms of space on your desktop. I must warn you, though, that your notes and reminders must be small as well.



Figure 2. The World's Smallest Corkboard?

The real heart of any system administration organization is the support desk, where young IT hopefuls sharpen their skills alongside the older, tougher, I-have-seen-it-all techies. Not only do the system administrators invariably and irrevocably find themselves involved in support, but in smaller organizations, they often *are* the support desk. Handling problems and dealing with a nonstop barrage of calls is probably more than can be efficiently handled by a few yellow stickies. The ideal is more along the lines of a system where calls are recorded through a user interface of some sort (a browser perhaps), problems are described, priorities are assigned, and the appropriate person is either dispatched or takes the call. *Mes amis*, your calls have been queued and answered. Please, sit, be comfortable. While François pours you another glass of wine, allow yourselves to savour these selections from tonight's menu.

The first recipe is something called **IRM**, which stands for Information Resource Manager. Written by Yann Ramin and Keith Schoenefeld, this package uses MySQL as its database. (You will need to be running an Apache web server with PHP4 extensions compiled in.) IRM is many things. First, it allows you to build a database of all the computers in your company or organization, organized in any way you see fit. This can include OS type, memory configuration, network card, IP addresses and more. There is also a separate list for network devices, hubs, cards and switches. *Mais oui*, you can add others as well. A similar function exists to record the software in your inventory (serial numbers, versions, etc.). You can then update the computer's profile by "adding" software so that when you click on that computer, it displays installed programs as well (useful when tracking licenses for that other OS).

What makes this listing of devices and software even more useful is that you can then enter trouble tickets (by clicking on Request Help) based on the devices or computers you have entered in your database. Then, using the Tracking function, you can assign the call to specific support representatives, and add follow-up information until such a time as the problem is resolved (marked "old").

The reporting is fairly light as distributed, but IRM comes with information on creating your own custom reports. Keep in mind that since the package uses MySQL, you can also write your own queries ad hoc even without building it into the interface. For instance, you may want to have **cron** run a report of open calls each morning at 8:30.

Installing IRM is not difficult. You will, of course, need an Apache web server with the appropriate modules (PHP, MySQL). Then, you will need the IRM source which you can get from the web site (all addresses are in the Resources section at the end of this article). Here are the basic steps for installing IRM. Start by changing to your web server's document root:

```
cd /your_webserver_document_directory
```

(The above might be **/usr/local/apache/htdocs** or **/home/httpd/html** depending on your install)

```
tar -xzvf irm-1.0.2.tar.gzcd irm-1.0.2
```

Notice that the application doesn't install itself into any other directory, so if you want to change the name (for your web access) then this is the time to do it. For example, rather than having users type the version number in their browser, I changed the name of the directory from **irm-1.0.2** to **irm** with the **mv** command. Of course, if you are offering the URL from an intranet menu, it probably doesn't matter.

The next step is to edit the **irm.inc** file in your distribution directory. The only thing to change here is one line near the top of the file. In this example, I have already changed it using the path to my web server's document directory:

```
$root_path="/usr/local/apache/htdocs/irm";
```

Now, we need to create the database using the template provided with the IRM distribution. This is done by feeding the **database.txt** (in the docs directory) into the **mysql** command. Before you do this, use your favorite editor and add this single line to the top of the **database.txt** file:

```
use irm;
```

We are almost finished. One last thing to do:

```
cd docsmysql -u root -p < database.txt
```

You will be asked for a password after which IRM's data structure will be created for you. You can now access the IRM system by using your web browser, **http://yoursystem/irm/**. At the login screen, enter your username and password. Figure 3 shows a screenshot of IRM's tracking system.

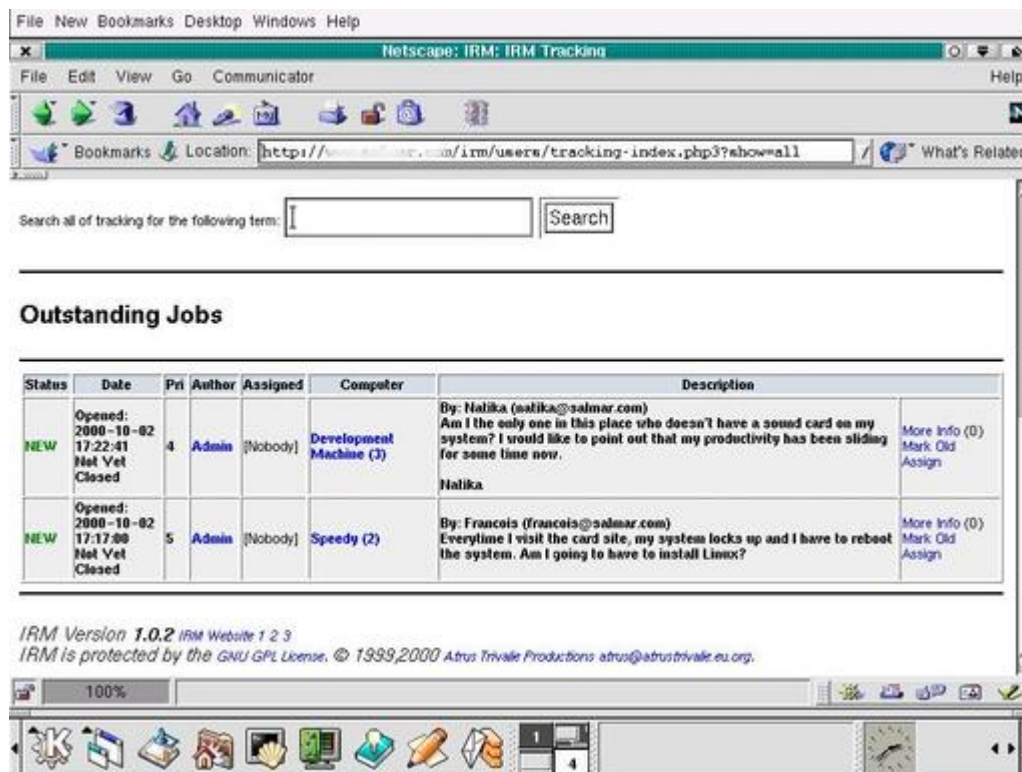


Figure 3. Trouble Tracking with IRM

The next package I would like you to sample is called **Request Tracker**. Request Tracker is the brainchild of Jesse Vincent, who, along with his faithful helpers, Deborah Kaplan and Mary Alderdice, has created a great support desk system. Unlike IRM, Request Tracker steers away from the complete inventory management model and concentrates on the support queue. Essentially, it focuses on the life cycle of a problem as submitted to any support desk. Calls come in, get assigned, worked on, then resolved and closed.

There are two browser-based interfaces to Request Tracker; an administrative screen and, the one in which you will be spending most of your time, the support queue. Using the administrative access, you can add new message queues. For instance, I created a separate queue for generic requests, one for hardware problems and one for application specific requests. You can also assign additional users and define their roles (administrative, post-only, etc.).

The main interface is the support queue itself. From there, it is possible to view requests, sort them according to personal or job preference, update, follow-up or close them. You can even set the queue to refresh on a regular basis so that you will always be aware of new calls coming in. Figure 4 provides a sample of the queue display's presentation.

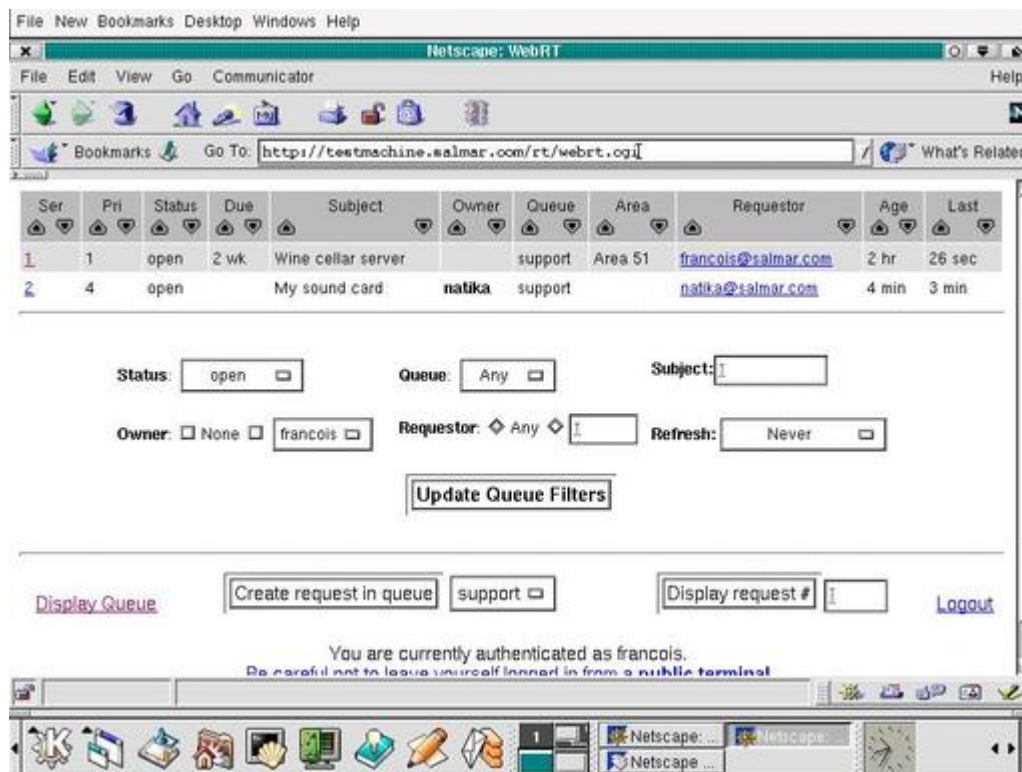


Figure 4. Request Tracker's Support Queue View

Request Tracker also uses MySQL to store its information, so that is a prerequisite. The program's access to the database is all Perl CGI rather than PHP, so you will need to get and install a couple of Perl modules. These are the **MySQL/mSQL** DBD module (available from www.mysql.org/downloads/api-dbi.html), the **Digest::MD5** module and the **CGI.pm** module (both available from cpan.perl.org). It is best to get the latest version of these modules even if you have already installed them with an earlier release. I had some trouble due to the fact that my CGI.pm module was a bit, shall we say, aged. Good for wine, but not for software, *non?*

The format for installing any Perl modules is simple. The following dialog is pretty much the same no matter what package you are installing or what version you are using. From your downloaded source, extract and build your modules like this:

```
cd /usr/local/temp_dirtar -xzf Perl-module-X.XX.tar.gz
cd Perl-module
perl Makefile.PL
make
make test
make install
```

C'est simple. Now, you will need to install Request Tracker. First, you must download and extract the latest distribution from their web site. The current version was 1.0.4 when I did my install:

```
tar -xzf rt.tar.gzcd rt-1.0.4
```

You will need to create a user with id "rt" and group "rt":

```
adduser rt
```

Using your favorite editor, open the Makefile and modify your local parameters. The Makefile is well documented, so read through it and make changes where necessary. In particular, you need to put in the host name of your system, what e-mail address support updates will be mailed to, and the MySQL admin login and passwords so that the install can create the Request Tracker database for you:

```
make install
```

Before you can use the package through the web interface, you will need to add a couple of lines to your httpd.conf file and restart Apache. Those lines are as follows:

```
Alias /webrt/ "/path/to/rt/lib/images/"  
ScriptAlias /rt/"path/to/rt/bin/cgi/"
```

To access Request Tracker's admin web interface, use this URL: <http://rthost.yoursystem.com/rt/admin-webrt.cgi/>.

The one user you have access to at this point is "root" and the password is "webpass". You should change that immediately before creating additional users. For the support queue screen, simply change admin-webrt.cgi to webrt.cgi.

As I mentioned, both these help desk packages currently use MySQL as their database. For your humble chef, this meant spending a little time installing and configuring MySQL since PostgreSQL is the restaurant's database of choice. Databases often tend to be a matter of choice, a choice that may be based on familiarity, perceived ease of use, support, licensing or many other issues (such as inclusion on your Linux distribution disks). Recently, MySQL became fully GPLed so the licensing issues are no longer a factor, and Red Hat will be distributing it with release 7.0. Still, I am pleased that the authors of both these packages are working on developing their products so that future releases will support other databases, including my old friend, PostgreSQL.

The clock is saying that it is closing time yet again. I will have François refill your wine glasses a final time before you leave us. There will be taxis waiting in the back for the sysadmins. Seriously though, the support help desk can be a complicated, stressful and challenging environment. In some ways, I think it is an ideal place for future system administrators to hone their skills. The opportunity for learning is as enormous as the pace is dizzying. Today's features may be just what you need to slow that pace and allow you to organize your thoughts and tasks.

Until next time, your table will be waiting here at *Chez Marcel*.

A votre santé! Bon appétit!

Resources



Marcel Gagné lives in Mississauga, Ontario. In real life, he is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy, and edits *TransVersions*, a science fiction, fantasy and horror magazine (soon to be an anthology). He loves Linux and all flavors of UNIX and will even admit it in public. In fact, he is currently working on *Linux System Administration: A User's Guide*, coming soon from Addison Wesley Longman. He can be reached via e-mail at mggagne@salmar.com. You can discover lots of other things from his web site at <http://www.salmar.com/marcel/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

The End of the Tube

Doc Searls

Issue #80, December 2000

I remember the day I learned that television was a narcotic.

To put the matter abruptly, the advertising industry is a crude attempt to extend the principle of automation to every aspect of society.

—Marshall McLuhan

We are, as an energy source, easily renewable and completely recyclable. All they needed to control this new battery was something to occupy our mind.

—Morpheus, in The Matrix

In the garage of life there are mechanics and there are drivers. Mechanics wanted.

—Sign in a auto repair shop

I remember the day I learned that television was a narcotic. It was in the spring of 1985. I arrived home from work one evening, and in the living room my teenage daughter and son were staring, green-skinned and slack-jawed, at our crummy 14-inch Toshiba TV. I stepped in front of it, reached behind me and turned the sound down.

“Just curious”, I said. “What are you watching?”

They both wore the facial expressions of fish in a tank. I could see the blank thought balloons over their heads, popping like bubbles in the air.

“What?”

"Do you know what you're watching?"

"Uhh..."

They didn't know. Really. They had no idea.

"Ummm..."

"Man...can you, like, move?"

In fact, we did move—across the country to Palo Alto, which mercifully lacked a cable system. Suddenly, we were back to watching what came in through the roof antenna, which was the video equivalent of outdoor plumbing. No more MTV. No more HBO. No more TNT, A&E or CNN.

Then the Toshiba TV died, and I replaced it with an old 21-inch RCA that I bought for \$10 at a garage sale. The thing came with a perfect snooze alarm in the form of an unintentional screen-saving feature: it progressively darkened until you slapped it hard. Since it took about five minutes to go from bright to black, there was a built-in limit to its sedating powers.

So we went cold turkey. Sure, we'd watch a movie on the VCR every once in a while, or the occasional PBS special. But sedation wasn't part of the package any-more. No more staring at advertising bait for hours on end.

Soon enough, both kids started doing better in school. Between her junior and senior year, my daughter cranked her SATs up by hundreds of points. She eventually graduated from Berkeley with a 3.9-something. My son followed a similar path.

Since then, I have waited for snooze-averse boxes to deliver the same salvation to the rest of Consumer culture.

And now it's here. The bait-and-hook business we call commercial TV is ready for the hospice, thanks to a modern—and far more intentional—version of our \$10 de-snoozing television.

It's called TiVo. Or RePlayTV. Both are boxes that constantly spool everything you watch, and then give you a choice about how and when you want to watch it. At a basic level, they improve on the VCR by recording to disk instead of tape (and far more simply). They also give you a way to pause a program while you answer the phone or go to the bathroom. But the real killer is that these products let you skip over ads in an instant. That's the one that blows away commercial TV.

In an excellent article for *The New York Times Magazine*, that can be viewed at www.nytimes.com/library/magazine/home/20000813mag--boombox.html, Michael Lewis reports that TiVo and RePlayTV customers don't watch 88% of the advertising they spool on their boxes. They probably edit out a lot of other crap, too, but that doesn't matter. What matters is that, in Lewis' words, "If no one watches commercials, then there is no commercial television."

Say ding-dong for that witch.

And when you run the credits, notice the operating system in the TiVo box. It's Linux, the same OS that hosts the smarts in set-top boxes from Intel and various OEM customers of the embedded Linux suppliers, all of whom see a huge business in what amounts to device drivers for television—and for whatever succeeds the television industry, once its passive consumers morph into active customers.

The irony here is that commercial television has always been conceived as a device driver for consumers. What we call "consumerism" is actually a kind of producerism. It's defined entirely on the production side of the shipping system that terminates at the wide end of the tube in our living rooms.

To some degree, all of us are still part of that system. We still think and talk about business in shipping terms. Our goods are "content" that we "address", "package" and "deliver" to a "consumer" or an "end user". We even speak of "delivering" services. We think inside the tube. Business isn't a handshake or a relationship, it's a conduit. We do business to people, not with them.

For proof, look at the shipping assumptions in the preposition abbreviated by the number 2 in acronyms like B2B and B2C. There's a huge difference between doing business to people and doing business with them. Talking in 2s puts us as deep in the tube as any record company or TV network executive.

And don't exclude the relatively enlightened creators of TiVo and RePlayTV. Their ultimate business model involves shipping extremely personal advertising straight to your choosy brain. That's why they have a user-spying system that makes DoubleClick look like the FSF. These boxes watch every choice you make and record it in what Lewis calls "atomic" detail. The ostensible reason is to make the box a knowing servant that can make educated guesses about other shows you might like. But the obvious economic reason is to draw a smaller and smaller bull's-eye on the back of your head.

When you read about these new boxes and their expected effects on society, watch the point of view. Notice that it almost always locates itself at the producer's end of the tube. It still abstracts "the market" as something remote:

a force, a demographic, a category or a synonym for demand. Never a real place where communities meet to do business and make culture, which is what it was until industry won the Industrial Revolution. We find a good example of this unconscious point of view in this passage from Lewis' piece:

Many things will change when television is able to whisper finely tuned messages to like-minded consumers rather than hollering crude messages through a bullhorn at millions. One thing that will change is the price of the messages. If they are to become more valuable, the targets must shrink, and as the targets shrink, the tools used to hit them must shrink as well.

We are “them”, the third person plural.

Well, let's look at them. Are they just staring back at production from the consumption end of the business tube, from the wide end of the bullhorn? Oh, no. Out here it's a real market. This is the bazaar. The bullhorn people can't understand it, because they never had a financial relationship with it. For the supply side of the commercial television business, viewers were never customers. They were consumers. Or, in the perfect words of Jerry Michalski, “gullets who live only to gulp products and crap cash”. The real customers were advertisers.

Out here in the bazaar, commercial TV is as lost as the man who walks down the street in Paul Simon's “You Can Call Me Al” It's a street in a strange world:

Maybe it's the Third World. Maybe it's his first time around. He doesn't speak the language. He holds no currency. He is a foreign man. He is surrounded by the sound, sound.... Cattle in the marketplace. Scatterlings and orphanages....

Markets have always been about interactions between customers and craftspeople. The craftspeople who matter here are the same ones who like to take commodity PCs and make them into everything else. These guys are in a movement that isn't organized by large producers, but by countless conversations about inventing products, solving problems, and enabling both by building new infrastructure that works for everybody because it's owned by nobody—just like they did with the Net and with Linux.

Both the Net and Linux are cheaply and easily put to use, all over the place. The threshold of invention and connection are both extremely low, and not just for converting PCs into Linux boxes. Now, it's getting easier and easier for small teams of designers and programmers to invent and embed intelligent control into pretty much anything. The range and scope of stuff for which we need Big

Manufacturing is getting smaller and smaller. The big manufacturers that matter are the silicon fabricators—the Intels, Motorolas, Samsungs and Hitachis of the world. Embedded Linux can only improve the cost, efficiency and time-to-market for small, custom manufacturing that puts commodity silicon (and other discrete parts) to work.

A few weeks ago, Don Marti and I were talking with a programmer who had recently left one of the Linux boxhardware companies and went to work for Kerbango, which makes a Linux-based radio for tuning audio streams on the Web. He had grown bored with working on server clusters and the rest of “the usual”. “That stuff was nice”, he said, “but this is really cool sh--.”

That guy was just one of the first. There are thousands, maybe millions, more of us out here in the real world, itching to invent all kinds of cool sh--. Thanks to embedded Linux, that's only going to get easier and easier.

As a cure for consumption, it's pretty hard to beat.



Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal* and coauthor of *The Cluetrain Manifesto*. His opinions are his alone and do not express those of *Linux Journal* or SSC.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

One World, One Processor?

Linley Gwennap

Issue #80, December 2000

The continuing dominance of Intel processors is pushing RISC system out of the server market.

One of the great advantages of Linux is that it runs on practically any processor you can find. Most competing flavors of UNIX are limited to a single CPU architecture. But in the server world, processor choices are diminishing. While this may not hurt Linux, it certainly doesn't help. These changes have been driven by Intel, which would love to turn the server market into a clone of the PC market, with hundreds of vendors selling similar systems based on Intel silicon.

Although far more PCs than servers are sold each year, the revenue from these two markets is actually similar, since servers are much more expensive, and have higher profit margins. So if Intel can grab its typical share of those profits (that is, most of them), it would be a great coup for the company.

So far, this strategy is off to a great start. In 1998, Intel launched a one-two punch, introducing its first Xeon processors designed exclusively for servers and disclosing the design of its Itanium processor for very large servers. With the exception of Sun, all major server vendors quickly adopted a two-track strategy, offering customers the choice of either Intel-based systems or systems using in-house RISC processors. The problem is that customers, for the most part, have been choosing the Intel-based systems, which use standard operating systems (Windows 2000 or Linux) and offer compatibility with systems from a variety of vendors. The RISC systems tend to be more expensive and rely on proprietary UNIX operating systems. Intel now holds nearly all of the market for low-cost servers (under \$10,000) and more than half of the market for more expensive systems.

This market shift has left the RISC vendors with less revenue to support their processor lines. During the same period, high-end microprocessor designs

have become more complex, requiring a greater investment to develop new products. With these two trends moving in opposite directions, Compaq, HP and SGI have slowed the pace of their RISC architectures (Alpha, PA-RISC and MIPS, respectively) to the point that they have fallen behind Xeon in performance on many key applications. This decline has caused a downward spiral in sale for the RISC vendors.

As a result, HP and SGI have already announced they will eventually discontinue their RISC lines, and Compaq is likely to follow suit. IBM continues to invest in its PowerPC line, but the bulk of the company's servers rely on Intel processors. At the recent Microprocessor Forum, the CPU industry's premiere event, IBM was the only one of these four companies to present a paper on a future RISC processor.

Sun has avoided this slippery slope by focusing exclusively on its SPARC/Solaris platform, and the company has never been healthier. But even with about 20% market share for servers above \$10,000, the company is having problems developing new processors. It recently rolled out its UltraSparc-3 processor nearly two years behind schedule and announced an 18- to 24-month delay in its plans for UltraSparc-4 and UltraSparc-5. Intel solves this problem by leveraging the same processor designs between its PC line and its server line. Thus, it can invest far more than any other company in the design of its processors. As CPU designs become more complex, this level of investment is needed to maintain competitiveness.

Even as Intel dominates the processor market, it splits the market between two different architectures: Xeon and Itanium. But so far, Itanium has been a dud. After many delays, the new processor should finally appear in systems over the next few months. But sources indicate its performance is disappointing, and most vendors are sticking with Xeon. Intel's new Pentium IV technology will appear in the Xeon line in early 2001, giving that line a further boost. Meanwhile, Itanium's hopes are pinned on a next-generation version that isn't likely to appear until 2002. Until then, Itanium will be a niche player.

AMD is gearing up to push its Athlon processor into the server market. If AMD succeeds, this move will put pressure on Intel to cut Xeon's price and boost its performance, much as the company was forced to react to Athlon in the PC market this year. Since Athlon and Xeon both use the same x86 instruction set, AMD's entry will strengthen that platform and make it even more successful than it is today.

Thus, for the next few years, Xeon and compatible processors will power the vast majority of servers. This convergence benefits Microsoft, which is focused exclusively on that platform, and takes away one of Linux's selling points. Linux

must go head-to-head with Microsoft in features and performance to continue its gains in the server market.



Linley Gwennap (linleyg@linleygroup.com) is the founder and principal analyst of The Linley Group (www.linleygroup.com), a technology analysis firm in Mt. View, California.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Focus on Software

David A. Bandel

Issue #80, December 2000

pemail, phpweather, scanSSH and more.

Mailing lists are something that strike fear into the hearts of mail administrators everywhere. But mailing lists are essential. They are also fun, funny, frustrating, time-consuming and a whole list of other adjectives (particularly when no one on the mailing list can help you with your particular question). But the best thing that I found from some of the lists that I have subscribed to is that they are educational. I don't have much time for lists, and I lament sometimes when I have to /dev/null several days, worth of mail without so much as reading the titles (thank goodness for filters, hate to miss mail from my readers). As long as the list you are on isn't drowning in trolls or hate mongers (or those who flame newbies mercilessly) you are probably picking up tips all the time. Myself, I'm often reminded just how confusing this stuff called Linux is to novice administrators. So don't sell mailing lists short. Several years ago, my participation on a mailing list landed me an offer to write a book. They have also landed me several contracts. So pick your mailing list carefully, write well and have fun. Just remember that the other members of the list only know you by your posts.

pemail: <http://www.the-den.org/pemail/>

I wonder how many people have ever used the UNIX mail command. Personally, I love the command. But it has some drawbacks, not the least of which is that it only looks at the mailbox on the local host. That is okay if you use NFS to mount your mail server, but if you POP your incoming mail off a server on the Internet, the UNIX mail command is not so hot. Perl Mail (pemail) will read your mail off a remote server. Perl Mail also shows how many messages you have, and a list returns the message number and the size of the message (which is nice if you don't want to download a monster message). The list does not return a line. Once the developer adds pemail, a subject line and maybe even sender line, this will be a killer app (at least for command-line

mavens). Requires: Perl and the Perl modules Mail::POP3Client, MIME::Lite, Term::ReadKey.

phpweather: <http://www.gimpster.com/php/phpweather/index.php/>

Want to keep an eye on the weather? phpweather reads the METAR data from NOAA and displays it in your browser. You can choose from a short display (the raw METAR itself) or a generated human-readable page (rather than meteorological gobbledegook). The only problem this suffers is with non-standard METARs. If you access US or most European station data, this should not pose a problem, but some countries do not return properly formatted data. I guess the National Oceanographic and Atmospheric Administration (NOAA) figures badly configured data is better than no data, but it sure makes parsing the stuff painful. Requires: Web browser with PHP and MySQL support.

scanSSH: <http://www.monkey.org/~provos/scanssh/>

I am a firm believer in encryption. I encrypt most Internet-bound traffic and I even encrypt stuff on my system. (Whoever said, "Running Linux means never having to delete your love letters" knew what they were talking about). Anyway, I find it easy for little things (like the large number of servers I maintain) to get their respective OpenSSH versions out of sync. Now with scanSSH I can run a scan and find out quickly if my servers are up-to-date or I've missed something on one of them. ScanSSH helps me with all the services I run, not just SSH. Requires: glibc.

HTAdmin: www.bilcag.net/hdogan/php/htadmin.php3

If you have experience, administering .htpasswd files is no big deal. Just use htpasswd. And if you can't remember where you put the file, the locate command will find it. If anyone other than you needs to administer a large list, it might not be so easy for them. Do you get tired of going over with people (for the hundred and twenty-first time) the incantation for htpasswd? Then stick this little utility on your web server, give them access (perhaps via a different .htpasswd file) and let them handle it. Provide a link on their desktop and (just perhaps) you won't hear from them every five minutes—at least not about the .htpasswd file. Requires: Web server w/PHP.

phpPhotoAlbum: <http://www.phpphotoalbum.com/home/>

If you have a bunch of photos (jpeg, gif, png, psp and other formats) and you want to publish them on the Web, phpPhotoAlbum will help you. The only thing lacking is a way to see a thumbnail of a picture before you look at the whole enchilada. Probably not a big deal on those less than 100K pix, but for the really

big ones, a thumbnail might work better. If you have ever set up a PHP app before, phpPhotoAlbum is easy to set up. Requires: Web server w/PHP compiled in web browser and pictures.

SING: <http://www.sourceforge.net/projects/sing/>

SING stands for Send ICMP Nasty Garbage. With SING you can do much more than ping. You can ping (send an ICMP echo request) if you want, but you can do a lot more. I started using it to test some netfilter settings, and the accompanying man page is extremely complete and explains SING very well. SING also supports creating big ping packets (note that this is not the recommended way to find your vulnerable systems) as well as performing source routing and source rewriting. In fact, there is not much that you can't rewrite in the ICMP header with this tool. Requires: libnsl, libresolv, glibc and af_packet.

nettop: <http://srp.portico.org/scripts/>

The nettop is one that goes directly in to the must-have category for all network administrators. You can run it on nearly any system, even a RAM or CPU challenged system and get a good idea of network traffic. Columns show percent of packets, total number of packets, percent of throughput, total throughput, avg packet size and packet type. Rows are color-coded by traffic type. For example, purple is ipv4 and arp; green is ipv4 broken down into udp, tcp, and icmp; and blue shows protocols like http, ssh, POP3, etc. At the top of the screen you can see the current system time and date, and how long nettop has been running. In my opinion this program only lacks one thing—a way to dump information to a file at regular intervals. Requires: libslang, libpthread, libdl, libm, glibc.

PHP DB Form Creator: <http://sourceforge.net/projects/phpdbform/>

This utility looks like a good start on a simple, good-looking forms interface to a MySQL database. The basic concept appears sound and setting up and creating forms is fairly simple. While it is not intuitively obvious (or even explained), it is possible to link tables. However, it does not yet appear possible to handle one too many situations, and reports are still listed as a TODO item. This utility is still very Alpha, so these more advanced features may come with development. Requires: Web server w/ PHP and MySQL support, MySQL, and a web browser.

Until next month.



David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Embedded Systems Conference 2000

Rick Lehrbaum

Issue #80, December 2000

At this year's Embedded Systems Conference, Linux loomed large in almost every booth. Talk about a year changing everything.

If you could travel back in time to the Embedded Systems Conference of September 1999, you would find that the “Embedded Linux Market” simply did not exist one short year ago. Sure, a growing number of developers and a handful of companies were starting to embed Linux, but as a market that anyone tracked, or paid attention to, embedded Linux simply hadn't made it onto the radar screens.

How many (and which) companies counted themselves as “Embedded Linux Companies” one year ago? How many embedded Linux news releases and new product announcements were issued at ESC in September 1999? Not many. One year ago, embedding Linux was a relatively rare phenomenon and was mostly the result of developer innovation—not the fruits of marketing plans and promotional strategies.

There were some exceptions, some pioneers. To find out what was going on in the “world of embedded Linux” one year ago, I turned to my favorite research tool (can you guess?) and scanned headlines around the time of ESC, September 1999. That search turned up many of the seeds of today's embedded Linux market. Companies making headlines about embedded Linux support included (alphabetically): Caldera, Cygnus, EMJ, FSM Labs, Lineo, MontaVista, PROSA and Zentropix. Early hardware partners for these embedded Linux pioneers included: Force, JUMPtec, Megatel, Motorola Computer Group, Synergy Microsystems and Ziatech. In particular, three

embedded Linux announcements around the time of 1999's ESC foreshadowed the later formation of the Embedded Linux Consortium:

- Zentropix announced RealTimeLinux.org, an effort to create momentum and consensus in real-time Linux solutions. Zentropix was later acquired by Lineo.
- Cygnus announced the EL/IX API “in an effort to pre-empt the fragmentation of embedded Linux in the embedded computing segment.” Cygnus was later acquired by Red Hat.
- Lineo announced an Embedded Linux Advisory Board (EMLAB), “an independent and vendor-neutral organization to promote and advocate the use of Linux in the embedded systems arena”.

Transporting back to the present—ESC, September 2000—where does embedded Linux stand today?

“Embedded Linux” Has Become a Disruptive Force in the Market

“Embedding Linux”, primarily an activity of innovative software developers one short year ago, has become the central focus of a rapidly growing number of commercial endeavors. The Embedded Linux Consortium, which didn't even exist seven months ago, already boasts over 75 corporate members. Major investments in embedded Linux, totaling hundreds of millions of dollars, have been made by industry powerhouses like Motorola, IBM and Intel.

It's important to understand that the embedded market would be going through a major transition, right now, with or without Linux. Independent of Linux, developers would be scrambling to satisfy the growing demand for both intranet and Internet connectivity. They would be hastening to take advantage of the opportunities presented by new low-cost 32-bit RISC processors coupled with abundant program and storage (Flash) memory. High integration system-on-chip processors based on MIPS, PowerPC and ARM cores make it both easy and inexpensive to embed full-system features in even the simplest and most cost-constrained systems. These emerging technologies have dramatically boosted the capabilities of embedded devices, and have also greatly elevated expectation levels.

In short, embedded Linux arrived on a scene that was already in the midst of great upheaval, with developers working hard and fast to apply newly available technologies to newly defined challenges. Because it provided low-cost, open-source, yet state-of-the-art functionality, Linux was well positioned to be swept along by the tide of change in the embedded sea. The open availability of source, coupled with today's unheralded ease and speed of collaboration and communication, turned out to be compelling factors that enabled developers to

quickly and efficiently adapt to the challenges of a rapidly changing landscape. So Linux began to spread like wildfire in the embedded market.

Everyone Now Has a Linux Strategy

As compared with last September's ESC, Linux support was found practically everywhere this year. Today, nearly every company has a Linux strategy—whether how to take advantage of Linux or how to defend against it. Non-Linux stalwarts like Wind River, Microsoft and QNX all exhibit symptoms to varying degrees of the necessity of coming to grips with life in a world where embedded Linux is an increasingly major factor. For example, both Wind River and QNX joined the Embedded Linux Consortium last spring as founding members.

In addition Jerry Fiddler, Wind River founder and chairman, now devotes several slides during his talks to defining his company's position relative to embedded Linux. In the open-source debate last ESC, John Fogelin, Wind River's VP of Technology, said, "We see point-of-sale, ATMs, Industrial PC and Internet Appliance applications as an opportunity where Linux can replace DOS and Windows NT. We embrace Open Source and are evaluating Linux as an OS option for Wind River customers. We are prototyping solutions based on Linux, now." In the latest ESC open-source debate, Fogelin reiterated Wind River's support for Open Source—while missing few opportunities to cast a FUD net over embedded Linux.

These days, even Microsoft's Embedded & Appliance Platforms Group endeavors to promote a new image of openness and flexibility. Although no explicit mention is made of Linux, expressions like "source access", "simplified licensing", "flexible business model" and "Windows Embedded Developer Community" have crept into the new Microsoft lexicon.

While not directly supporting Linux, QNX Software Systems announced a strategic initiative last June to reposition QNX, a POSIX-compliant RTOS, as "Linux-like". Among the changes were the opening up of source code for many QNX modules other than the still proprietary and royalty-based QNX Neutrino kernel, plus no-cost availability of the QNX development toolkit to individuals and developers. The get.qnx.com web site launched around the start of this year's ESC, and, by the end of the show, QNX reported that well over 100,000 copies of the free QNX toolkit had been downloaded by developers.

The most radical adaptation strategy of all came from Lynx Real-Time Systems, which last fall embarked on the path of adding Linux to its product line, along side the company's POSIX-compliant RTOS, LynxOS. Half a year later, the company took the further step of changing its name in order to more directly reflect its dual-OS (LynuxOS + Linux) strategy, becoming "LynuxWorks".

One Measure of Success

Not surprisingly, this year's *EE Times* embedded market survey reports a huge jump in the use of Linux as a tool platform among developers, a 1,400% increase over the last 12 months.

Unfortunately, however, useful data was not gathered regarding the use and planned use of Linux—or even Windows—as a target platform in embedded systems, due to what appears to be a catastrophic flaw in the only relevant survey question. The single question on target OSes was worded in a manner that restricted its applicability to only real-time operating systems. The question asked in the survey was: Which of the following commercial real-time operating systems have you (A) used previously, (B) are you currently using or (C) plan to use within the next year? Only RTOSes, such as VxWorks, QNX, LynxOS, OS9, etc., were included in the list of OS choices. Not surprisingly, neither Linux nor Windows showed up in the results.

I strongly recommend that the question be revised so that it encompasses all categories of target system OSes, not specifically real-time ones. Whether the target system requires hard, soft or non-real-time performance should be the subject of a secondary question. Other questions of interest might include connectivity options, standards compliance (like POSIX, for example), etc.

Accelerating from 0 to 100+ in Six Short Months

Another telling measure of the expanding embrace of Linux by the embedded market is the meteoric rise of the Embedded Linux Consortium, which now claims over 114 members. Founded in March 2000 by 22 companies, the ELC held its first general membership meeting at this year's ESC in order to provide an opportunity for members to meet the newly elected board of directors and one another. Attendance at the meeting was estimated to be over 75 members and guests.

After brief talks by ELC chairman Inder Singh and “yours truly” on the state of the embedded Linux market and the mission and objectives of the ELC, and by John Cheuck, vice chairman of EMBLIX (the Japan Embedded Linux Consortium), the ELC began the process of selecting specific projects and activities. To assist that process, a questionnaire will shortly be distributed to ELC members asking for input on what the ELC should do, as well as what each member feels the ELC should not do.

Currently, the consensus appears to be strong for the ELC to stay away from being a standards organization, but, rather, to focus its efforts on building the sense of “Embedded Linux” as a brand. Speaking of which, the ELC has commissioned a permanent logo—one with the obligatory penguin—which the

group wants to see strongly promoted by all of its members on their web sites and in their collateral materials.

Other likely projects for the ELC include a much larger booth for trade shows, including the possibility of hosting exhibits from individual members. The ELC may also form working groups to develop guidelines and recommendations that might be submitted to various standards bodies. Other ideas are being solicited.

Growing Architectural Diversity

There certainly seems to be a trend away from embedded PC architecture. That shouldn't be too surprising, given that the embedded PC is, after all, over 15 years old.

Intel's product announcements were roughly equally weighted between the latest embedded Pentium processors and the new StrongARM-derived XScale Microarchitecture. Other X86 processor vendors exhibiting at ESC included STMicroelectronics and ZF Linux Devices, with X86-based system-on-chip products, and AMD with its latest embedded K-series CPUs. Surprisingly, National Semiconductor, maker of the highly popular Geode X86 architecture system-on-chip processors, wasn't an exhibitor at the conference.

ARM, StrongARM, MIPS, PowerPC and other non-X86 architectures were well represented both at the booths of their manufacturers, and also at the booths of tool, OS and board vendors. Zilog, a corporate executive founding member of the ELC, seems to be bouncing back to life, and has revived use of the revered Z80 brand in more modern, connection-oriented processors. Who knows...maybe by next ESC, Zilog will have ported embedded Linux to the Z80!

On the board side, the traditional buses are still around. It's too soon to predict with certainty, but there's a hint of an emerging trend towards busless SBCs based on PowerPC, ARM and MIPS processors. Examples are the ADS Bitsy (StrongARM), Embedded Planet RPX (PowerPC), Intrinsic CerfBoard (StrongARM) and others. Even Ampro, one of the first and most prominent makers of embedded PC SBCs, has announced its intention to roll out MIPS versions of its new EnCore PCI-based platform.

What's Cool?

In addition to the above trends and observations, here are two Linux-related items that especially caught my fancy.

Lineo's real-time clock demo, you've heard of real-time clocks, but this one was different! It's a bit hard to describe, so please bear with me. Fantazein makes a

clock that “projects” the time via a set of LEDs on a wand that oscillates back and forth (see Figure 1 and 2). The LEDs are modulated at precisely the right rate to show the time, as if projected in space, as the wand sweeps back and forth. Okay, read the explanation on their web site (www.fantazein.com/how.html)—but hurry back!



Figure 1. Fantazein Clock

Now that you've got the picture, I'll explain what Lineo did. They rewired the inputs to the LEDs so that they are driven by a PC parallel printer port. With an otherwise idle Linux system, words projected via the “clock” appear stable and readable. Then, the system starts doing some file transfers, and the display becomes completely unreadable. Now, control of the LEDs is switched to RTAI, a hard real-time Linux system. You guessed it; even with file transfers taking place, the display is rock solid. It turns out that even a Linux kernel with low-latency patches would not be able to make the display remain stable in the presence of high loading on the system. RTAI (or RTLinux), on the other hand, can do it easily. Bravo, on a great demo of hard vs. soft (or non-) real-time performance!

With all the talk about open-source operating systems and related software, how often do we hear about open-source BIOS? Sure enough, there are a couple of projects going on out there. But to date, there has been no truly well-supported effort to provide comprehensive technology for system initialization and startup. That situation appears to be changing, as a result of some new software being developed by Red Hat called RedBoot. The project is new, and there is much yet to be done, but RedBoot may soon be “booting” that proprietary BIOS out of many embedded Linux systems. If you've ever tried to use an embedded PC in a non-PC application, you probably had to struggle with issues like eliminating a long list of copyright messages, creating a custom splash screen, speeding the boot process, supporting custom hardware initialization or adding robust system diagnostics. RedBoot to the rescue!

On, to the Future!

Well, ESC is a really large show and there were, no doubt, many other important trends and interesting demos and products that I didn't see and haven't included. I guess you really do need to be there, to take it all in. See you next ESC? Perhaps at the rate Linux is spreading, they'll need to change the name to ELSC—the Embedded Linux Systems Conference!



Rick Lehrbaum (rick@linuxdevices.com) created the <http://www.LinuxDevices.com/>, “the embedded Linux portal”, which recently became part of the ZDNet Linux Resource Center. Rick has worked in the field of embedded systems since 1979. He cofounded Ampro Computers, founded the PC/104 Consortium, and was instrumental in launching the Embedded Linux Consortium.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Deadlines

Stan Kelly-Bootle

Issue #80, December 2000

Deadlines are significant when debating the pros and cons of proprietary and open-software development.

In former pure-market-driven days, circa 1800 C.E., a product specification and honest delivery date was announced. I know, I was there. "Stan," said George Stephenson, "you don't have to be a Rocket scientist to appreciate my latest steam engine." Rivals would rush to improve the spec and/or beat the "to-market" deadline. Time passes, things change. You know, you've been there. In these PC (pronounced Pursey by Shakespeare) days, software announcements are now announced as soon as someone devises a catchy acronym. The current ploy, successfully borrowed by Microsoft from IBM, tries to discourage the competition.

Part of the problem is that software predictions can be woven into a bland, grandiose, uncontroversial jargon. Who could dispute the value of a globally web-aware, multi-paradigmatic, pattern-sensitive, cross-development architectural environment? (Have I missed your favorite desideratum?)

Nevertheless, deadlines are significant when debating the pros and cons of proprietary and open-software development. The former is often driven and accelerated by corporate job pressures. The latter, alas, relying on "casual" evolution, lacks the "next-week-or-else" imperatives. Replacing the trad Vaporware, we now meet the corporate term Vampireware (related to the Death March syndrome). For once, we can pin down the originator, Trygve Lode, CEO, Lode Data Corporation. Vampireware: A project capable of sucking the lifeblood out of anyone unfortunate enough to be assigned to it. The project never actually sees the light of day, but nonetheless refuses to die.

Yet many open and closed projects eventually emerge from the agonizing shadows to our clear screens. Dum codo spero?

I Have Mail

Dan Jurca writes from California State University, Hayward:

I just read your article in the October 2000 issue of *Linux Journal*. Because of your interest in $(2^{6,972,593}-1)$ I thought you might enjoy visiting: reality.sgi.com/chongo/tech/math/prime/merdigit/m6972593/prime-d.html

By the way, this Landon (Kurt) Noll is the person who, with one Laura Nickel, discovered in October 1978 that $(2^{21,701}-1)$ is prime; and soon after that, and after Miss Nickel lost interest, he discovered that $(2^{23,209}-1)$ is also prime. They used the library and computer resources at California State University, Hayward to do their research and perform the long boring computations.

Also, because you appear to be interested in "long numbers" you might enjoy the following tidbit. Consider the equations: 1. $A^{(A^A)}=10^{(10^{(10^{10})})}$ and 2. $B^{(B^{(B^B)})}=10^{(10^{(10^{(10^{10})})})}$

Reader challenge (huge prizes): Determine

a. Which of A and B is the greater, and b. the numerical difference between A and B.

I'll report Dan's solution anon. Meanwhile, I see you rushing to the sublime Mathematica. We older farts still swear by Napier's Bones.

Stan Kelly-Bootle (ska@atdial.net) has been computing on and off since his EDSAC I (Cambridge University, UK) days in the 1950s. He has commented on the unchanging DP scene in many columns ("More than the effin' Parthenon" Meilir Page-Jones) and books, including The Computer Contradictionary (MIT Press) and UNIX Complete (Sybex). Stan writes monthly at <http://www.sarcheck.com/> and <http://www.unixreview.com/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

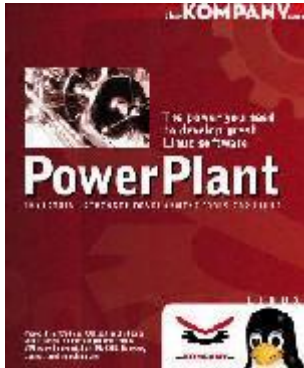
[Advanced search](#)

PowerPlant Review

Jim Gilbert

Issue #80, December 2000

PowerPlant Linux is a collection of various libraries and developer tools that aren't generally on a distribution CD or, if they are, are older than the latest, greatest and full-featured version you get here.



- Manufacturer: TheKompany.com
- E-Mail: info@thekompany.com
- URL: <http://www.thekompany.com/>
- Price: \$49.95 US
- Reviewer: Jim Gilbert

Do you consider yourself a Linux programmer? Have you ever found everything you need on a single Linux distribution? If so, do you never download newer versions of software off the Net?

Few people can answer yes to all three questions above, and this is where PowerPlant Linux comes in handy. It is a collection of various libraries and developer tools that aren't generally on a distribution CD or, if they are, are older than the latest, greatest and full-featured version you get here.

Software

Basically, if you're a developer and don't have the time, or don't want to write code for a specific need, there is a good chance you will find something helpful on CD. I am not going to list every piece of software that is included since you can find a complete listing on their web site.

First, you can find under-development versions of KDE and GNOME that are a must-have for anyone who is doing programming work. These versions tend to get rather old with time, but that's what the automatic update is for (more on it later).

If you are still searching for the perfect IDE, you have over ten to choose from, including one sponsored by the product makers, KDE Studio. You also have KDevelop, the official KDE IDE and, of course, its Gnome counterpart, gIDE, among others. For those in search of the perfect programming language, you can find implementations and tools for C/C++, Pascal, BASIC, Java, Perl, Python and many others that, I must admit, I had never heard of before.

The keyword in PowerPlant is latest. PowerPlant gives you the latest DOSEMU, XFree86 4.0 (which only now is beginning to be included in distributions), PHP 4, Perl 5.6, the latest 2.2 and 2.3 kernels, and a load of other applications and libraries. Every one of the applications and libraries are in their most recent version, including, but not limited to SQL servers, debuggers, game libraries and implementations of the ICQ and IRC protocols.

Installation

The mostly red PowerPlant box contains four CDs and a manual. The first CD has RPM versions of the included software, the second CD has Debian packages and the third CD has .tar.gz sources. The manual details their installation program, Magnum. Since I use Mandrake, I will include a short description of the RPM installer here.

The installer—which must be run as root—starts with the registration screen. Registration is not compulsory unless you also subscribe to their update feature, in which case it just makes sense to register. There is a Never Register button that you can use and live happily ever after (see Figure 1); I have registered and haven't received any spam from them yet.



Figure 1. PowerPlant Registration Screen

The installer next presents you with a small window (I chose the graphical installation, there is also a text one) with an Install and Uninstall button among others. Choosing Install brings you to the list of provided applications, unless the RPM database says they're already there (see Figure 2). When you choose a package, the installer presents you with the package description and the choice between a standard install and an install using advanced options (see Figure 3). You will get to the advanced screen if anything bad happens during normal installation (see Figure 4). The RPM version of the installer allowed me to upgrade a package if it was already there, force install it and/or skip running the package scripts. If you don't know the RPM command line by heart, you will find all those options described in the on-line help.

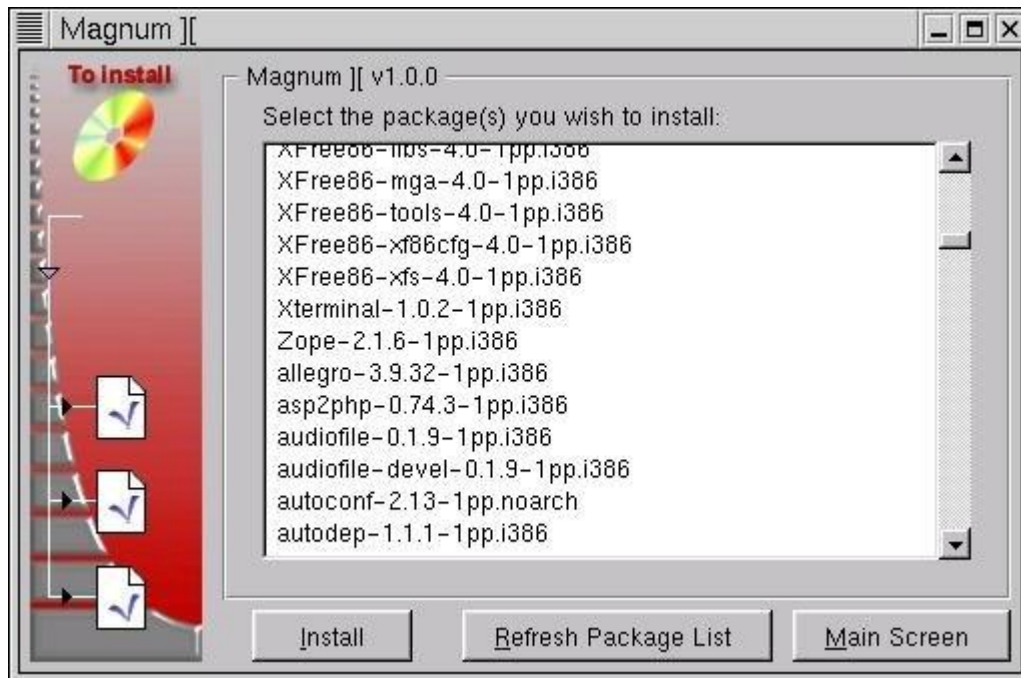


Figure 2. Install Screen

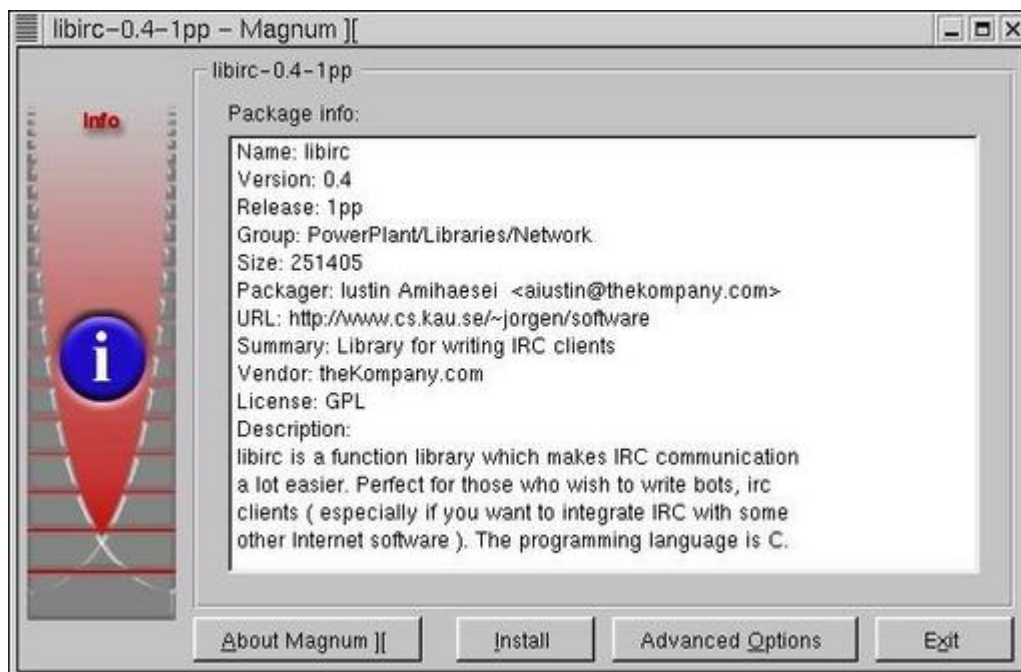


Figure 3. Package Description and Install Options

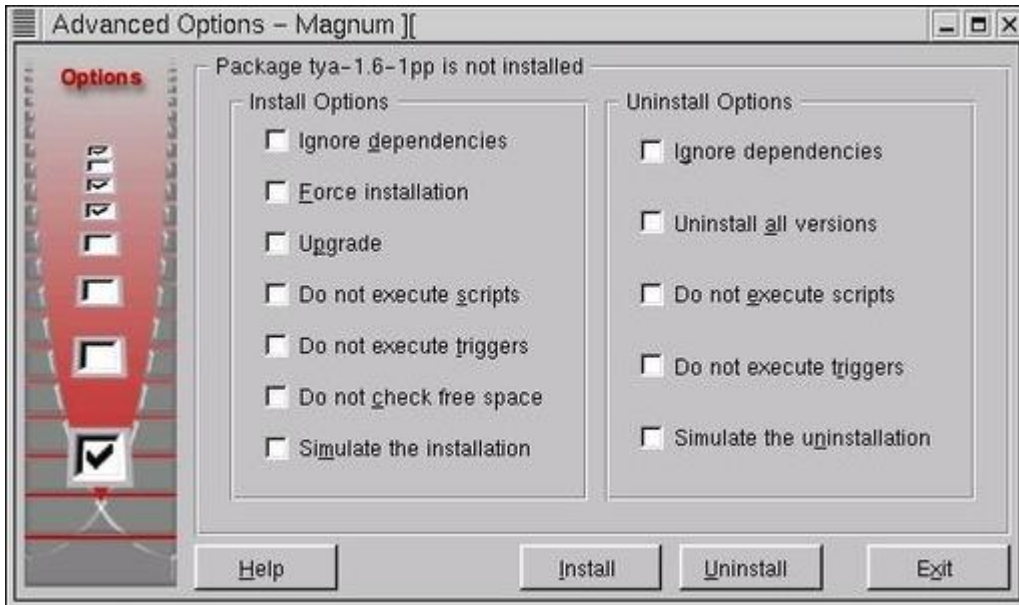


Figure 4. Advanced Options

The Uninstall button brings a list of PowerPlant packages that have been previously installed on your system (see Figure 5). The uninstallation process is similar to the installation process.

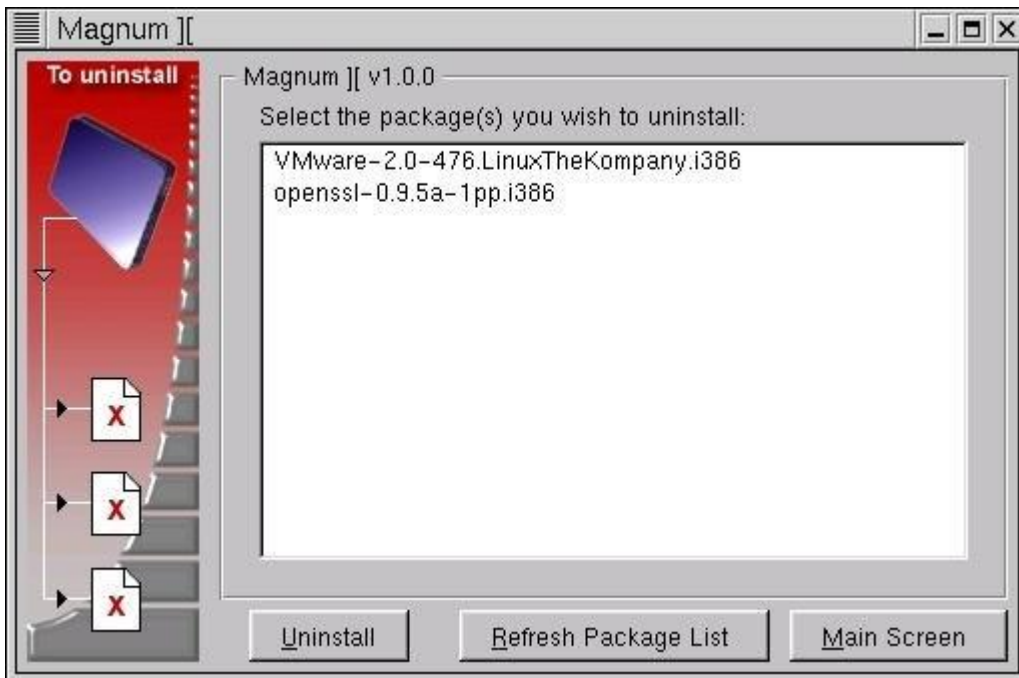


Figure 5. Uninstall Screen

The Icing on the Cake

You may have noticed that I counted four CDs in the previous section, but I described only three of them. That is because the fourth CD, although targeted at Linux developers also, is a bit different; it is targeted at your spare time.

The fourth CD contains demos of the (in)famous Loki ports of some of the best games out there: Civilization—Call to Power, Myth II, Railroad Tycoon II, Quake III, Heroes III and Heretic II. You can evaluate them for hours and hours before you decide if you want to buy anything from Loki. Of course, all their released open-source software can be found on the other PowerPlant CDs, such as SDL and OpenAL, to name two.

To make the Loki CD more than a bunch of demos that you can pick up from any game magazine, they added the full version of one of their games, Eric's Ultimate Solitaire. It is a great tool when waiting for long compilations to finish; it doesn't take many CPU cycles, and game times tend to be short.

PowerPlant Developer Network: Auto-Update Feature

Of course, the KDE version included with PowerPlant Linux was already outdated when I bought the CD, and I was an early adopter. It is a sad fact, but you cannot avoid outdated versions no matter whose CD you buy. This is where PowerPlant Developer Network comes into the game with their auto-update feature. At the time I wrote this, Red Hat had announced their subscription service; The Kompany.com had one in place four months ago when their product launched.

The PowerPlant installation program, Magnum, is both an RPM front end and an auto-update tool; it automatically scans the company web site for updates and offers to download them to your computer automatically. This way, my KDE 1.9x, and everything else in PowerPlant, is always up to date—while I sleep.

This subscription service is called PowerPlant Developer Network and includes a yearly CD update besides the automatic download. It's been less than a year since The Kompany.com launched PowerPlant, but I promise to keep you updated on the next CD.

Conclusions

Note that PowerPlant is not a Linux distribution; you need to be already using Linux. Thanks to its developer-oriented content, this is not a problem. PowerPlant may also seem small due to its focused nature, but it has about 150 different applications and will continue to grow with time.

Questions to their support e-mail address are answered quickly: they were very responsive when I had a little problem with KDE: they provided me with a patch on their ftp site.

If you're doing even hobbyist development on Linux, this product is nice to have. Of course, if you have too much time on your hands you can always

download all the software yourself, but assuming you have work to do, PowerPlant is a great help.

Pros and Cons

Jim Gilbert (jgilbert@hpjobs.com) has been in the IT industry for about 15 years. The core of his experience is with the HPe3000 platform but also includes Windows and web development. About a year ago, he decided to take the plunge and install Linux and has been a happy hacker since. Jim works as an independent IT consultant.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Debian 2.2 Potato: Memorial to a Hacker

Stephanie Black

Issue #80, December 2000

From installation to usage to tweaking, Debian 2.2 is a release that is fun for the techies and, well, much less of a Waterloo for novice Debian users.

- Manufacturer: Debian/GNU Linux
- E-Mail: leader@debian.org
- URL: www.debian.org
- Price: Available from various CD-ROM vendors or free via download from the Debian web site
- Reviewer: Stephanie Black

There's a lot to be said for an organization that not only depends on volunteers to develop its distribution and continually work to improve it, but acknowledges the contributions of those who participate in its growth.

In July of this year, Debian lost one of its more celebrated developers, Joel "Espy" Klecker, to Duchenne's Muscular Dystrophy. Debian, true to its philosophy, has recognized Joel's vast contributions by dedicating the Debian 2.2 release to his memory. (The written dedication can be viewed at ftp.debian.org/debian/doc/dedication-2.2.txt.)

The distribution itself is beginning to display the same kind of generosity. Debian's reputation has been further from the "novice-user" category than many distributions; this is slowly beginning to change. From installation to usage to tweaking, Debian 2.2 is a release that is fun for the techies and, well, much less of a Waterloo for novice Debian users.

Setup and Installation

Hardware Profile:

- 500MHz K6/2
- 64MB RAM
- 9.2G Quantum Fireball
- 3-Com ADSL modem
- SoundBlaster Live!
- Diamond Viper 770 Video
- DLink 530-TX NIC
- kernel 2.4.0test8

There is all manner of documentation and clear instruction for obtaining the required boot, root and driver disks with which to install Debian (available at www.debian.org/releases/2.2/i386/install). There is also a caveat about expecting perfect boot disks in one go. Good floppy disks make things easier and faster.

While not all users have DSL, the network installation of Debian proves itself to be sturdy and straightforward, even when using a command-line interface. Within a half hour, the installation was complete. This is probably the most significant change in Debian: easy download and installation via FTP.

The user is offered a choice of simple or advanced installation: the former provides a quick installation of some of the more commonly used packages through a selection of application types (e.g., C++ Development, GNOME games, etc.); the latter allows the more seasoned Debian user to select individual packages. **dselect** (which, as of this writing, is to be usurped by a new installer in future releases) provides dependency checking and simplifies additional package retrieval and configuration.

Configuration

Generally, dselect takes care of configuring most packages, allowing the user both the option of keeping the .deb packages and of setting up a running system. In an FTP download, the pertinent network configuration takes place prior to installation for obvious reasons. For some "non-free" packages, the user will be required to download additional software from sites hosting the original software first, such as the case of Real Player or IBM's JDK, and then obtain the remainder of the package via apt-get before configuration can take place. It's a bit detailed but worth it for the software. In addition, there are some packages that aren't permitted to be distributed in anything other than source packages. These require building; the process was, in the case of Pine,

clearly laid out in the README. The resulting packages are then built into .deb files installable by dpkg.

I'm a bit on the lazy side and don't like wasting good prewritten code. There are some scripts included with Fetchmail that preclude much of the headaches that some associate with setting up a mail client. These scripts are found in `</usr/doc/fetchmail/contrib>` and prove extremely useful to those new to, or uncomfortable with, MTA's.

Pros and Cons

Debian has a long-lived reputation for stability that has made it attractive to companies like Storm, Libranet and Corel, all three of which have capitalized on Debian's lack of easy installation. It's apparent that those who are installing Potato from a CD are likely to run into problems with, among other things, disks that have errors on them. I haven't heard if this is strictly the case with official CDs; if not, Debian may want to look at putting QA restrictions on those producing unofficial versions of the installation media. The network installation has improved tremendously, however, both in speed and security of the download.

The 4.x series of XFree86 is not included in this release, but the developers are in the process of adding it to the next release, or so I've been told. This would be a marked improvement; the current version of 3.3.6x is, at best, erratic. Xservers don't work (and don't not-work) with any kind of consistency. There are still issues with GLX (and the drivers thereof) for certain cards. RIVA-based cards (G-Matrox, among others) are rumored to be wanting more of the support provided in the XFree86 4.x series. To be fair, this problem is not specific to Debian.

The selection of packages in 2.2 is rivaled only by that of SuSE Linux and predictably runs the same risks—so many choices, so many dependencies and so many package conflicts. The choice is good, but several of the packages are in need of updating. This is especially obvious in the vast numbers of libraries included, many of which are present for nothing more than backward compatibility. (Anyone willing to volunteer to fix this?) The variety in the kinds of applications included is stellar, from math and science applications to games, editors and GUIs. The “corporates” don't have much to rejoice about in Debian, but there are certainly lots of tools and toys for developers, graphic artists, academics and hobbyists.

Debian isn't intended for the absolute Linux newbie. Help is available, however, from the users, list `<debian-user@lists.debian.org>`, which is quite active; most list members are willing to show new “Debs” the ropes. Be warned: traffic on this list is extensive (upwards of 500 messages daily).

How Well Does It Run? Can I Update?

Debian's only real failure in the past has been its lack of easy installation, and while the developers haven't succumbed to the trend of default GUI installations, they have simplified, clarified and tightened up the process considerably. In short, it's easy for seasoned Linux users to get, maintain and update.

Future Directions

A point 1 release of Potato was, at the time this was written, planned to present a number of security and bug fixes that cropped up in 2.2.0, including one that appeared in one version of the boot floppy. (There is a plan to remove the boot floppy altogether from future distributions.)

There's already an unstable or developer's version of Debian available for those who'd like to get in on Debian development. This will likely become Debian 2.3 or 3.0 and is available at ftp.debian.org/debian/dists/woody.

As the new IPv6 standard becomes more of a reality, Debian developers are making more of a concerted effort to make their code compliant with it.

Conclusion

An incredible amount of work on the Debian distribution has taken place since the 2.1 release, and it's been a treat to run it for the past ten days. It's on its way to becoming a distribution of choice for many who are not inherently technical but who are willing to learn a little to get a lot from their computers. For the technically apt, it's already arrived.

Joel Klecker would be proud.

The Good/The Bad



Stephanie Black is a recent migrant to IT and owns and runs Coastal Den Computing, a Linux consultancy. She has spent 80% of her coding life working with Linux can be reached at alphafemale@radiant.net.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

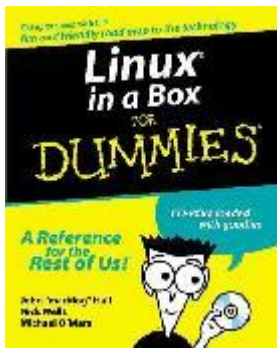
Advanced search

Linux in a Box for Dummies

Ralph Krause

Issue #80, December 2000

This yellow box contains three CDs and a 128-page installation booklet.



- Authors: John "maddog" Hall, Nick Wells, Michael O'Mara
- E-Mail: siteemail@idgbooks.com
- URL: www.idgbooks.com
- Price: \$24.99 US
- ISBN: 0-764-50714-1
- Reviewer: Ralph Krause

Linux in a Box for Dummies is one of the latest offerings for new Linux users from IDG, the Dummies company. This yellow box contains three CDs and a 128-page installation booklet. The CDs contain Caldera OpenLinux 2.3, StarOffice 5.1a and two e-books in PDF format: Caldera OpenLinux for Dummies and StarOffice for Linux for Dummies. Adobe Acrobat Reader 4.0 for Linux is also provided. For this review, I'll focus on installing the software and using the booklet as Caldera OpenLinux and StarOffice have been reviewed in more detail in other articles.

The installation booklet is written by Jon "maddog" Hall and Nick Wells and leads the reader through the installation of OpenLinux and StarOffice. The

booklet provides clear instructions accompanied by numerous screen shots, which guide the reader through the installation process.

The booklet's first chapters show the reader how to determine what hardware their machine contains and how to make room for Linux if they want to share the system with Windows. Instructions for using Partition Magic CE and FIPS to repartition a Windows hard drive to make space for Linux are provided, but Partition Magic CE is not included with this package.

Once the hardware has been determined and the hard drive configured, the booklet leads the reader through the installation of OpenLinux. The installation chapter contains screen shots of Caldera's Lizard installation program in action along with step-by-step instructions guiding the reader through the install process.

Following the Linux installation instructions is a chapter detailing the programs that can be used to configure X if it wasn't configured properly during installation. The Caldera lizardx program is covered along with the XF86Setup and xf86config programs.

After the installation and X troubleshooting chapters is a chapter on using Caldera OpenLinux for the first time. It briefly covers such things as booting the computer, using the KDE desktop, adding users, changing passwords and shutting the system down.

The final chapter in the booklet covers the installation and starting of StarOffice. The chapter provides step-by-step instructions for installing and starting StarOffice but not for using it. While a brief introduction to the StarOffice applications is provided here, using StarOffice is covered in the StarOffice for Dummies e-book.

The two e-books on CD 3 are meant to provide more information for new users. In addition to more installation information, the Caldera OpenLinux for Dummies e-book contains all the chapters that could be expected in a beginner's Linux book. There are chapters on the Linux file system, text editing, the BASH shell, connecting to the Internet, getting more information about Linux and troubleshooting.

The StarOffice for Linux for Dummies e-book covers the features of the StarOffice suite. The first section of the book introduces the reader to the StarOffice desktop and applications, while the following sections cover the office applications in greater detail. StarOffice is a fully featured office suite containing the following applications: StarWriter for word processing, StarCalc for spreadsheets, StarDraw for drawing, StarImpress for creating slide shows,

StarSchedule for keeping track of events, StarBase for creating databases, an address book, and web, e-mail and newsgroup capabilities. The e-book contains plenty of screen shots and examples of the applications in action.

While the e-books contain plenty of information at a minimal cost with low shelf-space requirements, they are a little clumsy to use. It is difficult to read a book page-by-page on a computer screen, and if you need to use the OpenLinux e-book to help with installation, you will either need a second computer or have to keep rebooting into Windows.

Linux in a Box doesn't include the Windows version of the reader, so Windows users might have to download it before they can view the manuals before they install OpenLinux.

Linux in a Box for Dummies provides a very capable system with OpenLinux and StarOffice along with two complete e-books. The installation booklet, while short, provides step-by-step instructions and screen shots, and the choice of Caldera's OpenLinux with its Lizard installation program makes installing OpenLinux quite easy. The e-books help new users of both Linux and StarOffice. New Linux users, who have access to a knowledgeable Linux friend, or experienced Linux users looking to get OpenLinux and StarOffice at a good price, should find this package a very attractive offering.



Ralph Krause (rkrause@netperson.net) lives in Michigan where he runs a small consulting company and writes. He has been working with Linux for over two years and recently started investigating the various BSD Unicies.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Two Books on PHP

Phil Hughes

Issue #80, December 2000

While there has been a lot of web development in Perl, and Python is growing in this area, PHP is a tool designed specifically for dynamic web development.



- Author: Julie C. Meloni
- Publisher: Prima Tech
- URL: www.prima-tech.com
- Price: \$39.99 US
- ISBN: 0-7615-2729-X



- Author: Tobias Ratschiller and Till Gerken
- Publisher: New Riders
- Price: \$39.99 US
- ISBN: 0-7357-0997-1
- Reviewer: Phil Hughes

I chose to review these two books together because they complement each other. That doesn't mean they are the same quality—just that they cover different things in the same area.

Before you ask, “Why PHP?”, let me answer. While there has been a lot of web development in Perl, and Python is growing in this area, PHP is a tool designed specifically for dynamic web development. That, along with its C-like syntax, makes it an easy choice for a programmer wanting to add dynamic web content generation to their tool kit.

As a general comment about both books, they claim to cover PHP4. This is pretty much true because PHP4 is not very different from PHP3. This is actually good news because most existing PHP applications are still in PHP3, and a book that didn't cover PHP3 would be relatively useless.

PHP Essentials

This is the introductory book of the pair. It introduces PHP and works through many examples all the way to setting up an e-commerce site. Three of the eight chapters deal with databases in general and MySQL specifically. As the power of PHP comes from how it interacts with a database, this makes complete sense.

Once you get comfortable with what PHP is, the most important part of the book is the Essential PHP Language Reference in Appendix A. This appendix starts off with an introduction PHP syntax, and then covers all the built-in functions available to the PHP programmer. This section is well written and can act as a decent reference even for a newcomer to PHP.

When I finished reading this book, I felt I had the knowledge to go off and write some serious PHP code. Fortunately, instead of writing code, I read the other book.

Web Application Development with PHP 4.0

This book is the real winner of the pair. By that, I mean it goes well beyond my expectations for any book on any programming language. New Riders seems to be on a roll with their recent language books, as the last one I reviewed, *Python Essential Reference*, was also a winner.

This is not a beginning PHP book and certainly not a beginning programming book. In the introduction, they state that the target audience is PHP programmers who want to take their skills to the next level, know other programming languages or want to extend PHP's feature set. I think they are correct with this statement.

This book is organized into nine chapters, each designed to extend your knowledge in a particular direction. The first chapter deals with development concepts. With over 30 years of programming experience in a dozen or more languages under my belt, I immediately wanted to skip this chapter. Fortunately, they anticipated this and included a sidebar titled “Why You Should Read This Section”. Okay, they were right. Too often, we want to get to the fun part—writing code. This first chapter sets the whole tone for the book, showing us the significant benefits of doing the design work first.

The next chapter, titled “Advanced Syntax”, introduces concepts such as classes, which can be ignored in basic programming but make your life a lot easier when you get into serious development. Chapters three through five deal with application design presenting concepts you need to think about, with code to fill in the blanks. Not quick reading but clearly a good investment.

Chapter six gets into database access. Like *PHP Essentials*, this book also concentrates on MySQL, but the presentation is very different. PHPLib is introduced, and readers are shown how to develop an application that is database-independent. PHPLib adds a level of abstraction for other interfaces as well as the database, and this chapter convinced me that using this library can amount to huge savings in development time.

Chapter seven goes on to use PHPLib to develop a knowledge repository. Note that all this code is on the included CD along with PHP, MySQL, Apache, TurboLinux, PostgreSQL and even the entire book as a PDF document. This chapter also includes an overview of XML and how it can be used to benefit the developer.

Chapter eight presents some PHP success stories, useful information that could also be used to convince your boss that PHP is real and does solve real problems. The final chapter shows how PHP can be extended.

Conclusions

If you are doing serious PHP development, *Web Application Development with PHP 4.0* is essential. It will make the difference between “my program finally works” and “I feel good about my program”. *PHP Essentials*, while not a great book, offers needed information to get you up to speed. What's missing in sophisticated programming is made up for in terms of the ease at which it can be read and its value as a language reference.



Phil Hughes is the publisher of *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

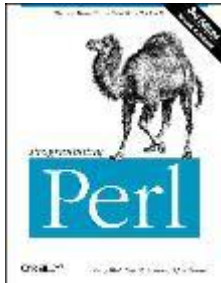
Advanced search

Programming Perl, Third Edition

Paul Barry

Issue #80, December 2000

The third edition of Programming Perl starts being different from the second edition as early as the cover.



- Authors: Larry Wall, Tom Christiansen & Jon Orwant
- Publisher: O'Reilly & Associates, Inc.
- E-mail: nuts@oreilly.com
- URL: www.ora.com
- Price: \$49.95 US
- ISBN: 0-596-00027-8
- Reviewer: Paul Barry

The Perl programming community had much to celebrate this year. Shortly after the millennium, along came the first major release of Perl since the middle of the last decade, when Perl 5 made its debut. Perl 5.6 was a long time cooking, but it is here now. And, of course, we can't have a new release of Perl without an update to "The Camel", the Perl bible: Larry Wall's *Programming Perl*.

The third edition of *Programming Perl* starts being different from the second edition as early as the cover. It's right there, at the top, the Perl mantra: "There's More than One Way to Do It." This phrase, more than anything else, sums up why Perl differs so dramatically from most other programming

technologies. Perl isn't just a programming language, it's a culture, and this book—more than any other—is your guide, introduction and reference to the culture that is Perl.

It's not just the cover that's different. Practically all of the content in this edition has been rewritten, revised, reorganized and expanded upon. There's now well over 1,000 pages, up from the 600 or so pages in the previous edition. What used to take nine chapters, now takes five parts and a whopping 33 chapters! This reflects the extent of the rewrite that this third edition represents.

Part I, “Overview”, is a high-level introduction to the Perl language and its culture. The goal is to give the reader an idea of the flavor of Perl, and the authors succeed at doing just that. But, be warned, this is not a tutorial for those of you wishing to learn to program Perl. In fact, this is the *wrong* book if you are new to Perl. This book is a reference. I like to tell people that “The Camel” is the *second* Perl book they should buy.

Part II, “The Gory Details”, is just that—gory. In 13 chapters, the reader is bombarded with a very detailed description of Perl as a programming technology. This can be very heavy going if you read the book straight through (as I did for this review), but, when used as a “dip in and sample” reference text, the material is easier to digest. Again, this book isn't a tutorial—it is the definitive description of the language. There's an awful lot of good material here. I was pleased to see regular expressions (Perl's programming language within a programming language) get its own chapter, as well as a greatly expanded treatment. There's over 60 pages of patterns and regular expressions in Chapter 5, “Pattern Matching” and, although it's all great stuff, my head was spinning by the end of it!

Part III, “Perl as Technology”, has chapters covering Perl's support for Unicode, IPC, Threads and Compilation. When it comes to Unicode and Threads, Perl is playing catch-up with other programming technologies (most notably Java). But, hey, the thought of using Java fills most Perl-folk with dread, so having to wait for these features to appear in Perl has always been tolerable, even though their implementation may still be classed as “experimental” in Perl 5.6 (as is the case with Threads). Additional chapters cover using Perl from the commandline as well as using the debugger. An extra (and very welcome) chapter on extending Perl with, and embedding Perl in C, rounds off this part of the book.

Part IV, “Perl as Culture”, is all about community. The reader learns the importance of sharing their code with others (to help make the world a better place), as well as how to protect their code both from themselves and from an increasingly unfriendly Internet. The chapter “Common Practices” is full of useful advice both for the novice and the expert Perl programmer. Another

welcome addition is a chapter on the issues surrounding code portability, and things to look out for when writing Perl code that will be deployed on disparate systems. A good programmer is a *lazy* programmer in the Perl world, and rather than have everyone ask you the same questions over and over again about your code, you produce documentation to go along with (and append to) any Perl modules you write. To this end, Perl provides POD (plain old documentation) and Chapter 26 tells you all you would ever want to know about this simple markup technology. The final chapter in Part IV is called “Perl Culture”. As everyone likes a good story, we are presented with a brief history of how Perl came to be. And what would a culture be without poetry? Also included are samples of the Perl poetry phenomenon. Competitions are held on a regular basis and poetry contributions are welcome.

Part V, “Reference Material”, provides details on Perl's built-in variables, functions, standard library and Perl's diagnostic messages. A nice touch in the “Functions” chapter is the use of descriptive icons to indicate the side-effects each of the built-in functions have on the Perl run-time environment. I was disappointed to see all of the standard modules listed but only a handful of them discussed in detail. The authors cite space constraints as the reason for this, but I will miss this material. Granted, all of the standard module documentation is available on-line (and it comes with the Perl distribution), but there is something to be said for having it in printed form. Perhaps the time has come for a separate volume on the Perl standard library?

To round things off, there's 29 pages of “Glossary”. You will not find a more entertaining glossary in any other computing text, anywhere. Imagine that, the word “entertaining” and the word “glossary” in the same sentence! Check out the entries for: algorithm, bucket, C, conditional, dweomer, eclectic, freely redistributable, funny character, hubris, impatience, invocation, laziness, Pern, portable, RTFM, script kiddie, shebang, string and (my personal favorite) UNIX.

My copy of *Programming Perl, 3rd Edition* was a very early printing and suffers a bit as a result. The authors fell over themselves trying to finish the book in time for the Summer 2000 Perl Conference, and a few things made it through the quality control checks that shouldn't have. For instance, in my copy, the black chapter tabs for Chapter 31 and 32 are reversed—what reads “Modules” should read “Pragmata” and vice versa. Another minor annoyance surfaced while I worked my way through Chapter 15 on Unicode. The author's discussed the **utf8** pragma, so I looked up **use utf8** in Chapter 31, “Pragmatic Modules”, in Part V, only to find that it wasn't there! The material's there, in Chapter 15, but it should *also* be in the Reference Part of the book. The index needs a little bit of tidying up, too. The on-line errata maintained at perl.oreilly.com indicates that the authors are aware of these problems and that some have already been fixed in more recent printings.

All in all, this is a great book, and the third Camel is the best Camel yet. It is written in a style and humor that is all its own (and which I suspect comes mostly from Larry Wall). The authors are all respected members of the Perl community—there's the Inventor (Wall), the Caretaker (Christiansen) and the Chronicler (Orwant). Between them, they know more about Perl than any mere mortal could possibly be expected to remember, which helps explain why this book is the *essential* Perl reference. You will not find a more authoritative treatment of the language in any other book. Just remember: when buying your copy, make sure it's the most recent printing.



Paul Barry (paul.barry@itcarlow.ie) lectures in Computer Networking at The Institute of Technology, Carlow, in Ireland (www.itcarlow.ie). Much to the dismay of his academic colleagues, he frequently and publicly admits that his favorite programming language is Perl.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux Music and Sound

Deric Mendes

Issue #80, December 2000

The courageous Dave Phillips attempts to lead us on a safari of sound brought to us through different audio software and the jungle of Linux.



- Author: Dave Phillips
- Publisher: No Starch Press
- E-Mail: info@nostarch.com
- URL: www.nostarch.com
- Price: \$39.95 US
- ISBN: 1-886411-34-4
- Reviewer: Deric Mendes

In most studies of yoga, the term mantra is used frequently as a defining factor of life. To the yoga guru, mantra is sound, sound is forever echoing through the chasms of the universe, and the universe is god. And, to most of the Linux gurus I've conversed with, this is the way they feel about Linux in the world of operating systems. So the collision of these two worlds, to some, could be as tragic as spilling that morning coffee into your hard drive just before you finish that project that was due an hour ago. Or it could be so exciting that you can't help but be nice to that man who cut you off on your way to work, and sent you swerving into the other lane. After reading *Linux Music & Sound* I hope that, with a little mentorship, mutating into a hybrid of musician and Linux geek will be a safe adventure for the fearful reader.

As to what that adventure truly is, the courageous Dave Phillips attempts to lead us on a safari of sound brought to us through different audio software and the jungle of Linux. He starts our journey by giving a very brief history of Linux. He then describes what the main applications of audio editing and creating sound software are and how to maximize your system to benefit from this digital repository of tools. After explaining most of the audio programs that should have been included with the current version of X, the author leads us further into the wild electric savannah to several different soundfile editors, such as MIXViews, Snd, Kwave, Broadcast 2000 and Ceres. The following chapters lead us through different types of programs such as Mod applications, MIDI, MP3, hard-disk recording, mixing, sound synthesis, music notation, networking audio, digital DJ, drum machines, operating emulators and video games.

The charging rhino of this book is the ninth chapter on sound synthesis. If you have very little or no experience with Linux and digital audio, this chapter will be as incomprehensible as why people listen to the Backstreet Boys and Britney Spears. If you understand the tribal tongue of Linux, you will learn how Csound and all of its helpers can be very powerful tools to compose and edit soundfiles and MIDI. This chapter gives some help as to configuring and understanding the code; however, it's more of a guide to performing tasks and loading Csound helpers. It also provides some understanding of what they do.

Tutorials for digital studio recording are uncharted territory in this book. There is no guidance to teach the common musician how to use Linux with his digital recording experience. Phillips does address every type of audience—the Linux musician, the Linux programmer and the non-Linux digital musician—but not every software evaluation.

In some chapters, the author's explanations are so elementary that he mentions what kind of cord to use and how to plug it into the jack: "Make sure the plugs are firmly in the jacks: Loose or shaky connections can result in dropouts, static, and other noise conditions." In others, he assumes that one paragraph on sound synthesis and one on MIDI is enough explanation to convey a complete understanding of how to apply them to a digital studio. I hope that anyone who can use a computer or plug something in knows that a loose jack is not a good thing. However, I know a few people who have recorded full digital CDs and don't understand what synthesis and MIDI are and how to use them. This book should not be your first choice if you want to learn about digital recording or Linux applications and their difference from non-open-source software. The book, along with some of the included software, cannot be well understood unless you are already a Linux programmer and a knowledgeable digital producer. If you're not a lion tamer avoid the lions.

The enlightening part of *Linux Music & Sound* is that it briefly introduces many programs, and, since they are open-source, if they're not on the CD provided with the book, they can be downloaded for free. URLs are conveniently provided throughout the text. If you understand the capabilities of the programs, you will appreciate that the author points out all of the audio beasts and lets you choose which ones are best suited for your needs, rather than pushing his own opinions.

Being more of a digital musician than a Linux user, I can see why it is not as widely used as other software. Other software, such as Pro Tools, Logic, and Sonic Foundry's Acid and Sound Forge, do not require many technical skills and have everything built into one program, so there is no need to switch files from one program to another. These programs range in price from \$30 to \$400 (I spent \$60 on my studio), which is not as good as free, and certainly not open source. This is something that can be frustrating to a programmer. I have never felt that these programs have hindered my recording process and, to a musician without a knowledge of Linux, spending money for software is easier than learning a new operating system.

However, most of the software released for Linux has not even reached version 1.0, yet can already handle professional procedures and produce as high-quality sound as its costly opponents. If program writers continue progress on the software, and the music world starts to take notice of Linux, I believe it will cause many studios to switch from the products they currently use to Linux, as is happening in other arenas.

If you are looking for a how-to book for digital recording on Linux, look elsewhere. The best way to think of *Linux Music & Sound* is as a buffet of audio software, which I found to be quite tasty. If you can't presently figure out what to use to record, or where to find software, this book should be your first desire. If nothing else, it will lead you deep into the jungle of new toys for your Linux machine.



Deric Mendes has studied guitar and music theory for several years and has also

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

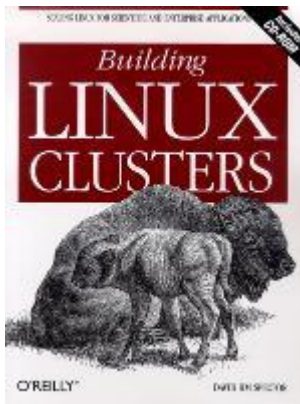
Advanced search

Building Linux Clusters

Glen Otero

Issue #80, December 2000

Ever since I started using Linux two years ago, I've wanted to build a cluster.



- Author: David H. M. Spector
- Publisher: O'Reilly & Associates
- URL: <http://www.ora.com/>
- Price: \$44.95 US
- ISBN: 1-56592-625-0
- Reviewer: Glen Otero

Ever since I started using Linux two years ago, I've wanted to build a cluster. During the early days of Linux self-instruction, I scoured the Internet for clustering information, looking forward to the day that I would be able to run a parallel computer in my own home. Everything I read on the subject made it sound like clustering with Linux was straightforward, almost simple. But it was slow going for a newbie, and I didn't get very far. Clustering information was dispersed all over the Internet, and I couldn't make a cohesive whole out of it. I was elated when Sterling, Salmon, Becker and Savarese published *How to Build a Beowulf*. I figured I was set. I bought five computers with AMD 400MHz K6s in anticipation of my homemade supercomputer. Alas, the book was very

informative but lacked the one thing I had trouble finding everywhere else: the software for the cluster installation. I was stuck.

The moment I discovered that O'Reilly was going to publish *Building Linux Clusters* I ordered the book. The advertisement claimed that the book would contain the software to build a cluster. I was saved. But I had months to wait for its arrival, so I planned and schemed about the really cool things I would do with my cluster, and I don't mean just crunching DES keys.

I was at the O'Reilly Open Source Software conference in Monterey, California, July 2000, when I discovered that O'Reilly was going to ship the title to the meeting. Two days later, I had what I hoped was the Beowulf grail in my greedy little hands, with a 20% off promotional discount to boot. In Monterey, separated from the boxes I wanted to turn into a cluster, I jammed through the 280 odd pages of *Building Linux Clusters* in under 24 hours. It had everything I, or anyone else, would ever need. I was ecstatic.

The material covered in the first four chapters is well written and designed to ease newbies into the waters. Topics covered include: the history of parallel computing, concepts of networks and parallelism, parallel programming systems and libraries, cluster types, cluster design, construction and assembly considerations, and hardware. Expectant cluster builders, like myself, will be familiar with most of the introductory material but may glean some useful bits on advanced topics such as meshes, hypercubes and symmetric multiprocessing (SMP). The author's recommendations for entry-level, mid-level and advanced-level cluster configurations will seem unrealistic to most Linux users. I felt that the hardware I had assembled for my cluster was pretty extravagant for a hobbyist to be experimenting with. However, my hardware didn't measure up to the seven 450MHz Pentium processors suggested for the entry-level cluster. At least I had that configuration beat on Ethernet bandwidth and memory. The author is aware of the low to no-budget, roll-your-own approach taken with most Beowulf clusters in the past, but he clearly stated his intention was to cover the construction of "serious" systems. Of course, the "seriousness" of a cluster is clearly relative, and the author's recommendations should be regarded as guidelines, not rules. However, if you do install the clustering software in this book, you will need about 1G of hard drive space per node, as the software requires about 946M of disk space.

The last three chapters contain excellent information and resources on parallel programming development environments, tools, libraries and applications. Common tools like GNU Emacs, GCC, G77, the Fortran compiler and GDB (the GNU debugger) are included on the CD. Two sets of parallel programming tools, PADE (Parallel Applications Development Environment) and XPVM are also found there. These packages are incredibly helpful GUI tools that allow

users to create and modify parallel virtual machines (PVMs) on their cluster. The Local Area Multicomputer (LAM) package is also included. LAM helps users execute programs that utilize the message passing interface (MPI) libraries (also included) by lending a hand in creating a processing environment for MPI-based programs. All of these packages come with on-line documentation as well. Several parallel programming libraries like SMARTS (Shared Memory Asynchronous Run time System), SILOON (Scripting Interface Languages for Object-Oriented Numerics), PAWS (Parallel Application WorkSpace), POOMA (Parallel Object-Oriented Methods and Applications), as well as the math libraries PETSc, PLAPACK and ScaLAPACK, are included on the CD. Several debugging and performance tools such as TAU (Tuning and Analysis Utilities), PCL (Performance Counter Library) and PDT (Program Database Toolkit) are also included. For those hard-chargers that want to move beyond the typical Beowulf and build more tightly integrated systems, two sets of kernel patches, BPROC (Beowulf-distributed process space) and MOSIX, are included on the CD.

As if that weren't enough, the CD includes two applications, mp3pvm and PVMPOV, to test your cluster with.

Clearly, the CD is loaded with copious and formidable tools to build and run parallel programs. The coverage of the topics found in the seven chapters I've summarized above is extensive. I consider this book serious reference material for the beginner as well as the advanced clustering enthusiast. That said, let's talk about why I can never recommend this book to anyone.

The editing and proofreading done on this book was nearly nonexistent. Eight of the first eleven figures in the book were either missing or out of order. The number of spelling and grammatical mistakes is unacceptable. We've all come to expect more from O'Reilly, and this sloppy offering came as a shock. Through lack of editing, enough stumbling blocks are encountered to quickly befuddle the Linux beginner. Those familiar with Linux and the introductory material can probably get by with a few assumptions and guesses as to what the author meant. I did just that and moved to the installation phase.

In a nutshell, the author's methodology for building a Linux cluster involves the Kickstart installation method developed by Red Hat, DHCP and installation of Linux over a network. The software is provided to create two types of boot floppies. One floppy is created for a master node. The master node boot floppy is used to perform a CD installation of Red Hat Linux 6.2 on the master node. Once the master node is up and running, it is configured to act as a DHCP server for the rest of the nodes in the cluster. The commands to enter the information the master node requires to serve DHCP requests for IP addresses is clearly noted. Once the master node is ready, the second type of boot floppy created is used to perform automated Kickstart installations of Linux onto the

remaining nodes from the master node CD over the network. In theory, it's simple and straightforward. So much so that the author mentions he was able to complete software installations on several clusters in about 20 minutes. Just the kind of software solution I needed.

There are so many mistakes, including errors in the commands to create the boot floppies, configure the DHCP server and perform the installation, that my software installation took eight hours. The very first command line I needed to enter in order to create the first installation boot disk was incorrect. It was to be one of many such obstacles. Scripts were also frequently mislabeled and often were not where they were supposed to be. The disparity between what is written and what is on the CD is appalling.

After hours and hours of making assumptions, cruising the file system, editing the boot disk software and a little black magic, I completed the installation. Now, I just had the simple task of initializing the cluster database using the cluster management software provided, and I would be able to run the parallel applications provided. You would think that I had endured the worst of it. But it's always darkest before dawn.

The cluster management software (another feature I was thrilled to see) included on the CD simply doesn't work. Essentially a bunch of Perl scripts, it was designed to work through both a web browser interface and at the command line. However, several links are missing on the cluster management package home page, making the browser interface moot. I possess a decent amount of Apache knowledge, but I couldn't make things work. Maybe I don't know enough Perl. In any event, when I tried to skirt the browser interface and populate the cluster database manually with the provided script, I was greeted with what I consider the death knell of this project. Running the script returned the message: This command will be provided in an updated Cluster Management Package. What? Not only was the software not working, it was incomplete. Things accelerated steeply downhill from there.

Refusing to give up, I tried adding nodes to my "cluster" (the single master node) by setting up a PVM using XPVM, no luck. I tried adding nodes manually at the `pvm>` command line prompt, no luck. My experience up to that point suggested that nothing was going to work without a struggle. So I reluctantly quit before giving the LAM package a try.

Apparently, I'm not alone in my suffering and disappointment. While writing this article, eight of the nine reader reviews posted at oreilly.com conveyed sentiments similar to my own. Other readers have submitted error reports that cover in greater detail what I've mentioned here. To O'Reilly's credit, they are

posting confirmed errata on their website. It's quite the uphill battle for them in this case.

I'm dumbfounded at how this work made it out of any publishing house, let alone O'Reilly's. To avoid this kind of debacle in the future, let me go on the record as stating that I will personally review any upcoming revisions/editions of *Building Linux Clusters* and test the software within, for free.

Building Linux Clusters should be considered a beta release. Much work needs to be done on the next version before I will be able to recommend it to anyone. Technically, I am ahead of where I was with my cluster before I got the book. But it sure doesn't feel like it.



Glen Otero has a PhD in Immunology and Microbiology and runs a consulting company called Linux Prophet in San Diego, California. He can be reached at gotero@linuxprophet.com. Surfing, in the ocean that is, is his favorite pastime.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Letters

Various

Issue #80, December 2000

Readers sound off.

Setting Permissions on Open Linux

I've had the same problem with not being able to write to the Windows directory when running a dualboot of Windows 98 and Open Linux 2.4 (Best of Technical Support, September 2000). What Mr. Rubini does not mention is that Caldera sets permissions in the /etc directory as -rwxr-xr-x. This most likely was done to prevent the user from inadvertently deleting Windows files.

Permissions to the fstab file need to be changed to "rwx" for the group and/or others. The fstab file can then be changed from "ro" to "rw". If permissions are to be changed for non-root users, they should be changed to "rw,umask=000".

—James Schoch jschoch@siscom.net

Graphic Card Follow-up

In the October 2000 issue, Matt Matthews' article, "Graphics: Pick a Card...Any Card", states "...the 3500TV has a TV tuner that is supported under Linux". I have been searching and can find no support...yet. Where is it?

—Al al_h@technologist.com

Matt Matthews responds: There are two projects working on the TV tuner that I am aware of, <http://sourceforge.net/projects/v3tv/> and <http://sourceforge.net/projects/tdfxTV/>. The former looks like it has a workable solution with some bugs. It also looks like development has slowed. The latter seems to be in its infancy, having only peer review code available.

—Editor

We Love Hackers

The cover of the October issue of *Linux Journal* states: Make Life Difficult for Spammers & Hackers. This calls “crackers” “hackers”, which they are not and, thus, throws mud on real “hackers” who are the very ones who give so much of themselves to make Linux and all open-source software possible in the first place.

Real hackers have been struggling against the media for years to recover the name “hackers”. And while it is normal for more ignorant segments of the media to use the term incorrectly, it is very annoying to see the term so misused and on the very cover of *Linux Journal*!

Do you really feel that Linus is someone that you need to “make life difficult for”? You really think he is going to break into your systems? As your magazine is directly and/or indirectly about hackers and their accomplishments, it seems to me like shooting yourselves in the foot to basically insult them by giving their group name a bad meaning.

—Terry Mackintosh—a hacker terr@terrym.com

As a longtime reader (I have Issue 1, Volume 1) who also sold *Linux Journal* at hamfests back in the early days, I say SHAME, SHAME, SHAME! I am talking about the cover headline that says “Make Life Difficult for Spammers & Hackers” on the cover of the October 2000 issue. You people should know better. A hacker is someone like Linus or RMS. Someone who breaks into computers is a CRACKER. Are you getting too preoccupied with the suits? This is the kind of headline I would expect from *Time Magazine* or *The Washington Post*, not from *Linux Journal*.

—Ken Firestone ken@firestone.net

We made a mistake. Everyone who worked on the cover will have to sing “Join Us Now and Share the Software” at the next Linux conference. We are committed to running articles by hackers about hacking.

—Editor

Not a Book at All

I have just read Daniel Lazenby's review of *The XML Handbook* by Goldfarb and Prescod in the October issue of *Linux Journal*.

I must say I cannot agree with your positive appraisal of this book. First, this can hardly be called a “book”—it is essentially a collection of corporate white papers

bound up with a CD-ROM of product demos. Prentice-Hall duped consumers by using Goldfarb's reputation to generate sales. Imagine the disappointment when readers discover that Goldfarb and Prescod have in fact authored only a small part of this work—the rest of the chapters were sold to vendors to hawk their wares. Prentice-Hall should steer clear of this tactic in the future, or they will be demoted to the ranks of Que, Sam's and other third-rate technical publishers.

To say this book was panned in the XML/SGML community would be an understatement—for many, this has seriously tarnished Goldfarb's and Prescod's reputations.

—Brad Clawsie bjc@yahoo.com

Assabet's Contribution to iPaq

I read with great interest the articles on embedded Linux and, in particular, the applications for handheld devices in the September 2000 issue of *Linux Journal*. I would like to emphasize though, that while the author of the article (“Compaq's Approach to Linux in Your Hand”) correctly points out the lineage of the iPaq as being the Itsy project at Compaq, one should be aware that Intel's Assabet reference design also contributed to iPaq's development. This is clearly seen when one looks at the system specification of the iPaq; this verifies that the system control registers are similar to the Assabet design. In particular, the unique audio codec solution on the Assabet has been identically implemented on the iPaq design. The Assabet platform is based on Intel's StrongARM* architecture (recently renamed as Xscale*), compatible SA-1110 processor and SA-1111 companion chip. The Assabet design offers significant technical attributes over the Itsy design, specifically the stereo audio, color display, power management, CF support and backlight to name a few.

Key developers of the Assabet platform are at a new company called ideaLogix, which is currently developing wireless video streaming products and services for visual information management.

—Jumbi Edulbehram jumbi@idealogix.com

Following Up with Security

Thank you for the good article “Good-bye Bandits, Hello Security” in October 2000's *From the Editor* column.

I believe it is of vital importance to do a follow-up study of the major Linux distributions such as Red Hat, SuSE, Debian, Slackware and about 20 more, and see to what extent security is implemented in the next versions. Please

examine the security, ease of use and default setup from installation program. With such a study, *LJ* will force the major distributions to set standards for this issue. As you mentioned in your article (aimed at distributors?, good) there are many new Linux users hooked up permanently by DSL or cable, and if security is not set up by default, or is too difficult, they will choose some other OS (I will not spell it out).

—Bertil Palmqvist Bertil.A.Palqvist@telia.se

Clarification for DNS Beginners

I was reading “Securing DNS and Bind” in the October issue and noticed an error that some beginners to DNS may not understand. When talking about the computer querying for an IP address to wiredmonkeys.org (or whatever it was), the author states that it will look to ns.isp.org instead of ns.someisp.org. This can be confusing to people who may wonder why it would leave their domain (someisp.org) and search a name server in isp.org.

—Brandon Shirey brandons@mail.esi.net

Shocked

I'm shocked at your response to Dale Lakes' letter in the September issue (Disgusted, page 6). It is ridiculous for a magazine like yours to hire a company which uses NT and IIS to fulfill your web hosting and e-commerce needs.

Don't you get it?

The Linux OS needs the support of the IT community. We need to be able to answer the objections of corporate managers and home users who can't think outside of the Microsoft Box. We all need to prove to our friends, family, coworkers—everyone!—that we can expect a PC to be reliable, stable and secure. We don't have to be overcharged for less than promised. We can concentrate on using our computers rather than being used.

However, when a Linux magazine has to rely on Microsoft to have a web presence, you make us all look like fools. You'll have no credibility and deserve none until you start practicing what all of us are preaching every day: “Expect More—Use Linux!”

By the way, your response said you're looking for a company that can do web hosting and e-commerce on Linux. My Google search (<http://www.google.com/>) returned 97,000 hits. Three of the companies listed had ads in your September issue.

—Randy Cook randy@rcook.com

LJ Store Web Site Provider Responds

I would like to respond to the reader whose letter was published in the September 2000 issue of *Linux Journal* under the headline "Disgusted". The author complained because *Linux Journal's* e-commerce and fulfillment partner (that's us) runs the *Linux Journal* Store web site (<http://store.linuxjournal.com/>) on Microsoft NT and IIS.

While the complaint is accurate, it doesn't begin to discuss how we use computing resources at WAS Inc. Let me say to this customer, and every other potential customer, that our use of technology is completely subordinate to the task at hand: satisfying the shopper. We don't pick operating systems due to popularity, we pick them for their suitability to our needs. (For various reasons, NT happened to be the best choice for our internally-developed software. We use, in addition to NT, Linux, IBM AIX, OS/2, PC-DOS 7, Mac OS and Windows 98. All singing, all dancing, all SAMBA-ing.)

We are, in fact, currently in the process of moving from NT to AIX as our primary platform, for the same reason: NT did not meet our needs. We discovered some serious scalability problems and needed to think hard about our platform of choice. We write our own e-commerce software (we do not use Microsoft's), so we have more invested in object-oriented technology and our current code structure than we do in any one operating system. Our product runs today on OS/2, Windows NT, IBM AIX, Sun Solaris and HP-UX. We'd be on Linux today if IBM supported our language there. (Please write and ask them to make VisualAge Smalltalk available on Linux.) We will be on Linux in the future even if they don't. But our immediate move will be to AIX on RS/6000 systems.

While I appreciate your correspondent's feelings, I wish he knew us better. We're Linux fans and Linux friends. We've run Linux since our first distribution—that's Softlanding Systems (SLS) 0.99. (I don't even know if they're around any more.) We did not discover Linux when Red Hat came along. I've personally been a *Linux Journal* subscriber in the past, way back in the first year. (I admit that now I read the back issues as they come into the warehouse—don't tell anyone.)

NT made business sense once. Linux and UNIX make more sense to us for the future than Windows 2000. This is where we get off the MS Windows boat. I would think our correspondent would be cheering.

We're not Redmond's mind slaves!

—Les Kooyman Vice President, WAS Inc.

[Errata](#)

[Archive Index Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

UpFRONT

Various

Issue #80, December 2000

Stop the Presses, *LJ* Index and more.

TALE OF TWO PLATFORMS

The press (present company included) loves a fight. Or better yet, a war. That's why we're eager to cast two successful competitors—whether or not they're succeeding at each other's expense—as warriors fighting over market share.

The best copy, of course, are David vs. Goliath stories. IBM was Goliath for decades. Davids came and went. There were the BUNCH (*there's a memory test*) then there was Digital, then Apple. Steve Jobs loved the role of David. After Microsoft took over the Goliath part, the Steve-less Apple made a pathetic David. Marc Andreessen and Netscape put in a much better performance. Now Marc has been replaced by Linus Torvalds.

But Linus isn't following the script. In the server business, Linux is turning into another Goliath, even though Microsoft isn't going away. As the latest IDC numbers show, both Linux *and* Microsoft are winning, big time. The losers are NetWare and UNIX. The sad news is that IDC lumps UNIX—Sun, HP and the rest of them—into one shrinking non-Linux group.

Between 1996 and 2003, IDC expects UNIX to lose half its share. NetWare was already declared dead by the press back when it led the pack, in 1996 (I remember, because I was working for those guys back then).

On the client side, the story isn't quite as interesting because there are no Davids, including Linux. It's almost all Windows.

Not that Linux is chopped liver. IDC shows Linux edging ahead of Macintosh on the desktop by 2003, leading by 5.5 to 5.2%. Amazingly, it also shows Windows gaining with almost a 90% share.

Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal* and coauthor of *The Cluetrain Manifesto*.

home.earthlink.net/~searls/dec00/idc_numbers.html

LJ INDEX—December 2000

1. Percentage of Linux developers who plan to develop applications for internal corporate use: **>50**
2. Percentage of Linux developers who plan to develop e-commerce applications: **40**
3. Percentage of Linux developers writing applications for mobile devices: **20**
4. Percentage of the above over the same number from six months earlier: **50**
5. Billions of dollars (US) that will be lost by record labels and book publishers by 2005 from “increased piracy and as artists and authors break away from publishers to go independent”: **1.5 to 3.1**
6. Billions of dollars (US) that will be gained by musicians in the same shift: **1**
7. Billions of dollars (US) that will be gained by authors in the same shift: **1.3**
8. Billions of dollars (US) that will be gained by third-party service companies: **2.8**
9. Billions of dollars (US) in projected on-line sales for the 2000 holiday season: **19.5**
10. Billions of dollars (US) in sales for the 1999 holiday season: **10.5**
11. Percentage chance that a potential customer searching an e-tailer's site will make a purchase: **2.7**
12. Percentage of commerce-driven on-line searches that “produced results that failed basic tests such as finding all relevant information or ordering procedures”: **92**
13. Percentage of advertising that goes unwatched by TV viewers using TiVo and RePlay boxes, which both allow viewers to skip over commercials: **88**
14. Size in billions of dollars (US) of the venture fund formed by Bertelsmann for investment in “an evolving media marketplace”: **1**
15. Number of new domains registered every second, as of January 2000: **1**

SOURCES

- 1-4: Evans Research (see <http://www.evansdata.com/implement.html/> for the complete data table)
- 5-8: Forrester Research
- 9-12: Gartner Group
- 13: *New York Times*

- 14: TheStreet.com
- 15: Internet Software Consortium (<http://www.isc.org/>) and Doc Searls' page at home.earthlink.net/~searls/dec00/ljindex_dec00.html

THEY SAID IT

"Computers are useless. They can only give you answers."

—Pablo Picasso

"Kaa's Law: In any sufficiently large group of people most are idiots."

—Kaa

"It doesn't matter who you are. Most of the smartest people work for somebody else."

—Bill Joy

"Technology lies on the leading edge of life."

—Rush

"Teach a man to make fire, and he will be warm for a day. Set a man on fire, and he will be warm for the rest of his life."

—John A. Hrastar

"Two rules to success in life: 1. Don't tell people everything you know."

—Sassan Tat

"We are perfecting markets. We are back in the bazaar."

—Kjell Nordström

"Brazil is the country of the future and always will be."

—Carl Steidtmann

"Prediction is very hard...especially when it's about the future."

—Yogi Berra

DOMAIN NAME ORPHANAGE

According to the Internet Software Consortium (www.isc.org), 72.4 million domain names were taken by this past January. That was up 16.2 million over the prior six months. That comes out to about 88 thousand a day, 37 thousand an hour or a little over one every second.

And yet some domain names are still safe from adoption. We prove that every few months by offering another list of domain names that prove untakable. Last time, we suggested "coloncam", "celeprosy" and "butthook", among others, all of which are still yours for the low, low price of 70 bucks or less for two years.

If you're one of those types that like to run with the Joneses, you're probably looking for one of those "nt" names, like Scient, Lucent, Viant, Cerent and Teligent (see <http://www.enormicom.com/> for the full list). But our crack research department (that sits right here in my chair) has uncovered at least ten "nt" variants that are still available. They include: Boviant, Annoyant, Terminant, Reodorant, Cementent and Inexperient. So there you go; register at will.

Now, here's this month's orphanage, filled with children that probably will remain safe from adoption. All are available in .com, .net, .org and every other form.

- stenchofwindows
- missthepoint
- carpolite
- undertux
- thundertux
- teenyweenylinux
- talktodeath
- pantsdot
- overpants
- birdturbine
- bozoretentive
- earfloss
- tonguewax
- foohost
- crapmap
- leisuresuitlinux
- anitadrink

- doubtfarm
- quarterdog
- halfahead
- halfawake

—Doc Searls

The Penguin

by Rob Flynn and Jeramey Crawford

Once upon a term'nal dreary, while I hack'ed, weak and weary,
Over many a quaint and curious volume of forgotten code--
While I nodded, nearly napping, suddenly there came a beeping,
As of someone gently feeping, feeping using damn talk mode.
"Tis some hacker," I muttered, "beeping using damn talk mode--
Only this. I hate talk mode.

"Ah, distinctly I remember it was in the bleak semester,
And college life wrought its terror as the school year became a bore.
Eagerly I wished for privileges--higher access I sought to borrow
For my term'nal, unceasing sorrow--sorrow for a file called core--
For the rare and radiant files of .c the coders call the core--
Access Denied. Chown me more.

"Open Source," did all mutter, when, with very little flirt and flutter,
In there stepped a stately Penguin of the saintly days of yore.
Quite a bit obese was he; having eaten lots of fish had he,
But, by deign of Finnish programmer, he sat in the middle of my floor--
Looking upon my dusty term'nal in the middle of my floor--
Came, and sat, and nothing more.

Then the tubby bird beguiling my sad code into shining,
By the free and open decorum of the message that it bore,
"Though thy term'nal be dusty and slow," he said, "Linux be not craven!"
And thus I installed a new OS far from the proprietary shore--
The kernel code open but documentation lacking on this shore.
Quoth the Penguin, "pipe grep more!"

Much I marveled this rotund fowl to hear discourse so plainly,
Though its answer little meaning--little relevancy bore;
For we cannot help believing that no living human being
Ever yet was blessed with seeing bird in the middle of his floor--

Bird or beast sitting in the middle of his cluttered floor,
With such instructions as "pipe grep more."

But the Penguin, sitting lonely in that cluttered floor, spoke only
Those words, as if its soul in that instruction he did outpour.
Nothing more did he need utter; understood did I among that clutter--
Understood his command as I could scarcely do a few moments before--
I typed as furious as was willed me, understanding just a minute before.
Again the bird said "pipe grep more!"

"Amazing!" said I, "Penguin we will conquer the world if you will!
By the network that interconnects us--by that Finn we both adore--
We'll take this very world by storm!" For now grasped I what he'd meant,
The thing I do while searching /usr/doc/* for that wond'rous lore--
Those compendiums of plaintext documentation and descriptive lore.
Quoth the Penguin, "pipe grep more!"

And the Penguin, never waddling, still is sitting, still is sitting
In the middle of my room and still very cluttered floor;
And his eyes have all the seeming of the free beer I am drinking
And the term'nal-light o'er him glowing throws his shadows on the floor;
And this OS from out the shadows that is pow'ring my term'nal on the
floor
Shall be dominating--"Pipe grep more!"

Linux Bytes Other Markets: City of Garden Grove Adopts Linux

by Drew Robb

Free Linux Download Snowballs into Citywide Government Deployment.

Five years ago, the IT department in the City of Garden Grove, California faced significant budgetary constraints. Rather than continue to pay for proprietary software, Charles Kalil, acting information systems manager for the city, decided to check out Linux. He downloaded it for free, installed it and liked what he saw.

Five years later, Linux is running on six servers and 386 PCs. It serves everyone from the public works department to the fire department, and they couldn't be happier with the results. "Our citywide Linux network has operated continuously for over a year without crashing," said Kalil.

As an experienced NT system manager, he believes that it is possible to achieve similar performance from NT. But that means limiting each NT box to only one

type of service. "Linux can stably handle file/print serving, mail server and more on one machine," said Kalil. "NT can't."

When the city began experimenting with Linux, it was originally looking at buying an NT network. But that meant purchasing multiple servers, licensing agreements and added software costs. "Linux came as a free download that also included a Web server, mail server, Samba file, and print sharing, and Network Files System (NFS) capabilities," said Kalil. "With the alternatives such as SCO and NT, these were either not included or you had to purchase them separately."

It wasn't all clear sailing for Garden Grove, however. Running Linux in 1995 was much more adventurous than today due to lack of support and a shortage of applications. Despite that, the city set up a Linux database that has been running ever since.

From a cost standpoint, the difference was substantial. Garden Grove replaced a \$400,000 Data General minicomputer with two Pentium 90 servers that cost \$5,000 combined.

Kalil also found that certain applications could not be ported to Linux. "We still use NT for imaging software on our optical jukebox," he said. The City of Garden Grove also maintains a GIS server running NT. Once again, the GIS software does not have a Linux port.

But as far as price, reliability and availability are concerned, the city is fully committed to being a Linux-based shop. "We found that we didn't need high-price servers due to the efficiency of the Linux kernel," said Kalil. "Further, we can obtain the same results as NT with about half the memory." [See *LJ* Issue 35 for a previous article on Linux and the city of Garden Grove—Editor]

▮ Drew Robb is a Los Angeles-based freelancer specializing in technology issues.

STOP THE PRESSES: A NEW COLOR FOR SUN—COBALT

There has always been a strange relationship between Cobalt's primary business model and its most obvious product: the self-branding Qube. Every time I spoke with him, Stephen DeWitt made it clear that his company's business was selling rackmounted servers—its RaQ brand, especially—to ISPs who, in turn, would sell box-resident, value-added services to their customers. In fact, Cobalt has developed a large number of third parties whose applications could be packaged with Cobalt RaQs and sold to ISPs.

The Qube was a great little product, easily put to use anywhere one could find a constant Net connection and an available IP address, basically for SOHO settings. It was an easy product to love—a bright blue cube with a wide greenish light in the front. But the server appliance market was yet another one of those zero-billion dollar categories that would get around to delivering their promise when broadband was less the exception than the rule. As Qube observer Luke Tymowski puts it, “There's more money to be made selling RaQs to ISPs than Qubes to you and me.”

But when Cobalt sold itself to Sun Microsystems a few days ago (I write this on September 30, 2000), the Qube and the Appliance Category seemed to be the whole story.

The *San Jose Mercury News* told a typical story. Under the headline “Sun to Buy Cobalt for \$2 Billion” ran the subhead “Deal gives company market for low-cost server appliances”. In the first sentence, Cobalt was identified as “the maker of a compact server-in-a-box”. The obvious manifestation of that label is the Qube, but the practical one is the RaQ. And Cobalt has done a remarkable job of productizing RaQs as appliances—as plug-and-serve devices. Its on-line literature says, “Now in its third generation, the Cobalt RaQ is a mature, proven server appliance already in use in 1/3 of all the Tier 2 and Tier 3 service providers around the world. In fact, Cobalt RaQ is the server of choice globally because the Linux-based system does not require the constant attention of very expensive IT engineers.”

Cobalt has had very good marketing instincts from the beginning, playing the Linux label much the same way as it played the appliance label. The question now is whether Sun will tamper with that success. Sun has always gone out of its way to say it “supports” Linux but remains anything but a “Linux company”. Now with Cobalt it has bought one of the most familiar Linux companies in the world.

However, Cobalt, unlike VA Linux and other Linux hardware companies, has been a Linux company only to the extent that it employed Linux as a small, handy commodity OS. It also used a small, handy commodity microprocessor. Any idea what it is? Hint: it's not Intel or Motorola. There's marketing at work for you.

Among all the literature provided to me by Cobalt during the Spring Linux World Expo in 1999, the only mention of Linux was in 6-point type on the back of the company's data sheets. But, Cobalt quickly welcomed identification as a “Linux Company” and surely benefitted from that association with a big IPO in the fall of 1999. Around that time, “appliance” was becoming a hot term. Cobalt

soon wisely identified nearly all its products as “server appliances”. Looks like it paid off.

Once Cobalt is part of Sun, there won't be much semantic leverage left in the word Linux, simply because of Sun's antipathy to the commodity OS. And sure enough, Sun is already reportedly thinking about dumping Linux from Cobalt servers and replacing it with Sun's own appliance-specific version of its operating system, Solaris. One wonders if they'll also insist on SPARC processors. But Luke Tymnowski says the fact that “they are making noises about moving the RaQs over to Solaris from Linux doesn't mean much. It wasn't the OS that was remarkable, it was the web administration interface.”

Indeed. It's hard to imagine a simpler UI for a server than the one Cobalt designed for its appliances. Let's hope for their sake that they keep it that way.

Tech Tips

Debugging Tip

Handy for debugging and watching what's going on while it's going on. Use tail with the “f” option, which lets you read the end of a growing file. Examples:

```
$ tail -f /var/log/messages$ tail -f /var/log/syslog
$ tail -f /var/log/mail.log
$ tail -f /usr/local/httpd/logs/error_log
```

Typescript

Need to capture some output to a terminal that can't be redirected easily to a file with “>”? Use **script**. At a command prompt type **script**, then do what ever you need to log and exit. The log of ALL the stuff sent to your terminal finds its way into a file called typescript! Example:

```
tux@coollinuxbox:/home/tux$ scriptscript: WARNING: script session is not secure against
eavesdropping/hijacking!
script: read /usr/doc/bsdutils/README.script for details.
Script started, output file is typescript
tux@coollinuxbox:/home/tux$ python
python commands
control-D
tux@coollinuxbox:/home/tux$ exit
Script done, output file is typescript
tux@coollinuxbox:/home/tux$ cat typescript
Script started on Thu Oct 12 12:03:22 2000
tux@coollinuxbox:/home/tux$ python
Python 1.5.2 (#1, Dec 15 1999, 11:15:06) [GCC 2.7.2.3] on linux2
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> 45+89+12.25+63.21
209.46
>>> 70/12
5
>>> 70%12
10
>>>
tux@coollinuxbox:/home/tux$ exit
```

```
Script done on Thu Oct 12 12:04:43 2000  
tux@coollinuxbox:/home/tux$
```

Dolly the Hardisk

Wanna make a clone of one hardisk to another? Use **tar**. Hook up your soon-to-be-cloned hardisk to your system (power off during this operation). Boot your box. As root, cd to /. Mount the new hard drive on /mnt. Then run the following command:

```
$ tar cLf - . | ( umask 0; cd /mnt; tar xvf - )
```

c = createl = stay on local filesystem (don't cross filesystem boundaries)
f = file (the next argument is the name of the tarfile or "-")
- = write to standard out or read from standard in
x = extract
v = verbose

"umask 0" ensures that the new files have the same permissions as the old ones.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

The Trouble with the Bastard Operator from Hell

Don Marti

Issue #80, December 2000

Welcome to our special issue on violence in the workplace, I mean system administration.

Welcome to our special issue on violence in the workplace, I mean system administration. Despite the picture on the cover, *Linux Journal* does not normally advocate violence as a solution to system administration issues.

But why is the system administrator as an individual, and system administration as a profession, always in some kind of crisis? Why is a system administration job a never-ending flow of fires to be fought and ill-planned new information technology purchases to be integrated somehow into the operation? Where is the respect or consideration for the system administrator's expertise in how to make information technology work? Hey, stop laughing.

The answer to that last question is: "Nowhere, of course. Welcome to the race against burnout. Don't try to change the organization; just make your big score and get out." Business experts everywhere advise managers not to keep a dog and try to bark, too. But it's an article of faith that management thinks nothing of inflicting well-marketed but idiotic information technology products on the poor, battle-weary sysadmin. But, it doesn't have to be this way.

In *The Trouble with Dilbert*, Norman Solomon writes, "One of the best ways to teach people not to rebel is to offer plenty of ruts for fake rebellion." And sysadmins have nothing if not ruts for fake rebellion. With an escape hatch as close as the nearest ssh client, sysadmin culture has flowered under management repression into newsgroups, mailing lists and web sites all dedicated to the proposition that all sysadmins are created superior. That's not surprising for a bunch of people with such diverse backgrounds and skills. Show me one sysadmin who majored in Computer Science, and I'll show you three historians and a molecular biologist. Sysadmins are, on the balance, excellent writers and creative people.

So why does the job of being a sysadmin suck so much? Simon Travaglia's *Bastard Operator from Hell* is the folk hero of system administration, and his stories are linked to from everywhere. If you haven't read the BOFH stories, he's a self-described [Don, did you really think we could put that word in *LJ*? — Ed.] who delights in tormenting any user who asks a question (and some who don't, just to be on the safe side). But, just messing with the users isn't making the system any better. The BOFH is entertaining, but the way of the BOFH leads only to more clueless users to deal with, more problems and less help.

What the BOFH has may seem like power, but real power lies in the ability to construct something useful. And you're not going to get that done just working around the demands of idiots and taking revenge where you can. You don't work for the Ministry of Information here. In the real world, a system administrator can walk away from idiots. And given the current job market for people who know Linux, you can walk away from the next idiot, and the next. Imagine a company that works your way because you built a system that works your way, or just works, period. It's possible. The Bastard Operator from Hell can only destroy. By abandoning foolish companies to seek out the good ones, a system administrator can really create something.

Resources



email: info@linuxjournal.com

Don Marti is the technical editor for *Linux Journal*. He can be reached at info@linuxjournal.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Best of Technical Support

Various

Issue #80, December 2000

Our experts answer your technical questions.

Trouble with Netscape

I am having trouble getting Netscape to run on Linux. I have Windows NT in the first 4G and installed Linux in the remaining 4G partition. Linux detects the Ethernet card when first installed, but when I log off to use NT and then go back to Linux, Linux no longer sees my Ethernet card. It gives me a "domain not bound" error. I would like to know what is happening. —Yong, yboone@novell.com

Netscape is known to have lots of bugs and problems and not only with Linux in my experience. I recently reinstalled the 4.75RPMs (from the Red Hat ftp site) and, as of now, there seem to be fewer problems, especially related to Java. It is still quite usual for the first process of Netscape to die, but new instances of the program will run fine. Regarding your Ethernet card, you do not mention the brand/model, but I solved a similar problem by turning off the PnP feature of a 3Com 3c590 Ethernet card and, under Windows, making it work as ISA or EISA. Linux worked fine after that. —Felipe E. Barousse Boué, fbarousse@piensa.com

Adjusting Ethernet Cards

I am setting up a cable modem. I have a 10/100 Ethernet card running at full duplex. How do I lock the card to 10MB and half duplex? Is this in the "hwconf" file? If so what does the syntax look like? Any additional help is appreciated. Also, I am running a RCA Cable Modem. —David A. Bower, davidbower@iwon.com

You do not specify the brand of Ethernet card you have. But, usually there is a diskette that comes with the card (or you can download it from the card manufacturer's site) that allows you to turn off the Plug-and-Play and duplexing features of the card. Bear in mind that for some cards this change doesn't

really do anything to the behavior of the card, in respect to Linux at least. I would suggest looking at the Ethernet-HOWTO at www.linuxdoc.org/HOWTO/Ethernet-HOWTO.html where you can find additional information about your specific card's settings. —Felipe E. Barousse Boué, fbarousse@piensa.com

Cable modems can be frustrating because they often use a form of DHCP designed for Windows systems only. Setting your card to run half duplex may help and can be done by using the driver as a module rather than as a built-in driver. Then use **insmod mydriver.o full_duplex=0**. Note that there are a few cards that do not support this parameter. I have had problems even after doing that. To get my cable modem working under Linux, I had to specify ALL of the following switches on the command line of "dhcpcd": **-r -h myhostname.in.windows -l 1:xx:xx:xx:xx:xx -l 3600 eth0**. This tells the program to use the RFC1541 (obsolete) protocol, to specify the host name (which you can get by running WINIPCFG from your working Windows system), and to specify the Ethernet address and card (from the same machine; may not be required for you). —Chad Robinson, crobinson@rfgonline.com

Num Lock on Dual-Boot

My machine is set up to dual-boot Red Hat Linux 6.2 and Windows 98. When it boots to Windows 98, the Num Lock stays on, but when it boots to Linux, the it goes off. Is there a way to change this behavior so I don't have to press the Num Lock key each time I start Linux? —Michael Kaneshige, kaneshige@uswest.net

If you are working in text mode, look at the man page for the **setleds** command. In one of the initialization scripts, let's say /etc/rc.d/rc.local, add something like:

```
for i in 1 2 3 4 5 6 7 8 do setleds -D +num < tty$i done
```

This will turn on Num Lock by default when booting up your system. If you are using a Graphic User Interface, there is usually an Options setting specifically for the Num Lock status at boot time. —Felipe E. Barousse Boué, fbarousse@piensa.com

You can add these commands to your /etc/rc.d/rc.local boot script:

```
INITTY=/dev/tty[1-8] for tty in $INITTY; do setleds -D +num < $tty done
```

--Pierre Ficheux, pficheux@com1.fr

PCMCIA Card on a Laptop

I'm installing Red Hat 6.1 on a laptop. How do I get it to read/load the pcmcia card and not the eth0? —Anthony G., anthonyivs@relaypoint.net

You should look at the `/etc/rc.d/rc3.d/*` startup scripts to see the order in which the pcmcia and eth0 boards are initialized. Renaming the files with higher or lower numbers (`S10xxxxx`, `S20xxxx`, etc.) will redetermine that order. Many startup services are stored there, so be careful when playing with these files. Also, check the `chkconfig --help` command, so you can turn one service on and the other off. Lastly, to manually initialize or stop the pcmcia or Ethernet boards, use the commands:

```
/etc/rc.d/init.d/pcmcia [stop|start|restart]ifdown eth0
ifup eth0
```

--Felipe E. Barousse Boué, fbarousse@piensa.com

Emulating VT100

When I telnet from a Linux PC to a Linux server and edit a file on the remote PC using vi, I get a screen full of ANSI X3 cursor control characters. My PC does not seem to emulate the VT100. I have set `TERM=vt100` in my `.bash_profile` and used `EXINIT='term=vt100'` and even used command mode in vi (`:term=vt100`), all to no avail. What am I doing wrong? Thanks in advance. —Dominic Wild, wildd@optusnet.com.au

Seems you need to adjust the `TERM` variable to the correct values. You are doing the right thing except you may not be using a vt100 emulation. Try using "ansi". Bear in mind that you may have to set the correct emulation mode on both the machine you are logging into and the machine you are logging from. That is, every emulation on your setup must be the same in order for it to work. To check which emulations are in use on the machine you telnet from, check the environment variables with the "env" command. —Felipe E. Barousse Boué, fbarousse@piensa.com

The terminal settings (`$TERM`, etc.) have no effect on the terminal itself. They only tell applications what the current terminal is, i.e., what escape sequences it generates and accepts. Therefore, if you force `$TERM` to vt100, applications will send vt100 commands to a TTY that is not a vt100. I suspect your terminal name should be "linux", "xterm", "rxvt" or whatever is appropriate for the terminal emulator you are using. Since `$TERM` is propagated across Telnet sessions, you cannot change only its value. —Alessandro Rubini, rubini@linux.it

Cannot Load Tulip Driver

I have a SMC 8432T Ethernet adapter. All the info I've gathered is that it's a DEC 21041 chip, and the driver is tulip.o. Problem is that KDE won't load the tulip driver, only 24x5. But my biggest frustration is that I can't find a tutorial for manual setup anywhere. —Al Smolkin, alik713@home.com

You have to load the tulip driver from the actual Linux boot process, not from KDE or any other GUI. My machine also has a tulip-based card, and I handle it like this in my Red Hat 6.1 system. In file /etc/conf.modules, there is a line:

```
alias eth0 tulip
```

*I compiled the module tulip.c into tulip.o and placed it in **/lib/modules/2.2.16-3/net/tulip.o**, where it gets loaded at boot time. Once loaded, you can run level 5; that is, you may turn on any graphic environment you desire. To manually load the module (after it has been compiled), try the command:*

```
insmod tulip
```

*Then you can **cat** the file /proc/modules to check if the tulip module has been loaded into your system.* —Felipe E. Barousse Boué, fbarousse@piensa.com

Make sure the 21041 is really what you have; it might actually be a 21040, in which case the IRQ and port won't always be found automatically. You can tell this by looking at the large black chip on the card, it will have this number stamped on it. To be quite honest, old SMC Ethernet cards can be a pain; I've used them for years, and I have to do something different for each one. You can get your card to work, but if you're frustrated, why not try the Etherpower II? It will be recognized instantly, and they're pretty cheap these days. —Chad Robinson, crobinson@rfgonline.com

Configuring Sendmail and POP3

Please help me out in setting up sendmail to handle POP mail for my internal network. The network has one Linux server, satish.enet.com, which has sendmail and DNS. DNS is configured and I am able to send and receive mail to users on the Linux machine, but I am not able to send and receive mail through Outlook Express in Windows 9x. —Dasi Satish, sdasi@manraonline.com

To enable POP3, you have to uncomment the pop3 entry on the file /etc/inetd.conf. After that, you have to reinitialize the inetd process using the command:

```
/etc/rc.d/init.d/inet restart
```

Then, configure the Outlook client to use `satish.enet.com` for the POP server (incoming mail server). You need to have usernames and passwords configured on the Linux server for POP to work correctly. Also, sendmail must be running on your Linux server to be able to provide SMTP service (outgoing mail) to your Outlook clients. Therefore, set up the SMTP server to use the same machine. You may need to set `/etc/mail/access` with lines such as:

```
aaa.bbb.ccc.ddd RELAY
```

where `aaa.bbb.ccc.ddd` are the IP addresses of the PCs on your LAN. This allows them to use the mail facilities of the Linux box without getting “relaying denied” messages. Erase the `/etc/mail/access.db` file, restart sendmail with:

```
/etc/rc.d/init.d/sendmail restart
```

and you should be sending and receiving mail through your Linux box to the Outlook clients. —Felipe E. Barousse Boué, fbarousse@piensa.com

Sendmail is not a POP server. POP is a pull technology that clients who are not always connected use to get their mail. SMTP (Sendmail's bailiwick) is a push technology that expects the remote system to be connected at all times. You need a POP daemon. There are several available on the Internet, and most likely, one was installed on your system by default. —Chad Robinson, crobinson@rfgonline.com

Setting Inodes for Small Files

I need to fine-tune the ext2 file system of a new server to handle a large amount of extremely small files (10K or less). How do I format the drive so that it can handle these small files without running out of inodes? —Ether Trogg, ethertrogg@mindspring.com

All you need to do is increase the number of inodes on the system. You can do so using the “-i” parameter of “`mke2fs`”. Most systems use 4096 as the default value, which means you will get one inode for every 4096 bytes on your system. This should be fine for your 10K requirement, since that will average two or three inodes per file. However, if you find yourself with many files <4K in size, try values of 2048 or 1024. Anything lower is probably counterproductive. —Chad Robinson, crobinson@rfgonline.com

Mapping Keys Independently

I need to map keys to mean different things depending on which console I am using. For example, I need to have a menu screen on one session and a POS order taker on the other. Each has to have keys set up to mean different things. So far, it appears that Linux only supports a global change. I need them to act

independently, which worked well under SCO UNIX. We are using Debian Potato. —Dave A., anderson@webace.com.au

You can do this under Linux in much the same way as you do in SCO. The problem is that most default distributions don't expect you to, so you have to do a bit of work yourself. The `/etc/termcap` file determines the keyboard mapping definitions (among other things). What you need to do is define two entries, for example, `linuxtty1` and `linuxtty2`. Once you do this, modify your `/etc/profile`. You may optionally place these changes in the user account's `.profile` or `.cshrc`, depending on the shell used. You must write code to determine the user's login TTY and set the `TERM` environment variable based on that. I can't give you an example without knowing which shell you are using, but a less-than-trivial bash example can be found in `/etc/profile` on most systems. (It sets `TERM` based on local versus Telnet logins.)--Chad Robinson, crobinson@rfgonline.com

Running a Slave Name Server in a Chroot Environment

In the October 2000 *Linux Journal*, Michael D. Bauer's article on Securing DNS and BIND explained how to run your name server in a chroot environment. When my name server also works as a slave for some domains, it needs to be able to run `/usr/sbin/named-xfer`, to transfer zonefiles from the master. I spent quite some time solving why my slave domains didn't work, and I finally got it. But lots of less experienced users might waste too much time on this with a final result of "it doesn't work", and run non-chrooted names again, which decreases their security.

The solution is as follows. `$CHROOT` is the directory in which you are running your chroot BIND.

1. Check which shared libraries `named-xfer` uses, using the command:

```
ldd /usr/sbin/named-xfer
```

1. Create a new directory, `$CHROOT/lib`, and copy the required libraries into the new directory.
2. Make the other required directories inside the chroot environment, with

```
mkdir -p $CHROOT/etc $CHROOT/usr/bin $CHROOT/lib
```

1. Make an empty `ld.so.conf` file and create the necessary symlinks to the libraries:

```
touch $CHROOT/etc/ld.so.confldconfig -v -r $CHROOT
```

1. Copy named-xfer into the new usr/sbin directory under the chroot directory:

```
cp /usr/sbin/named-xfer $CHROOT/usr/sbin
```

And voilà--it works! Of course, your directory for slave-files needs to be writable for users who runs named. —Michal Ludvig, michal@logix.cz

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

New Products

Heather Mead

Issue #80, December 2000

APPX Release 4.1, The Bitsy, The 2.5-Inch ATX25 and more.

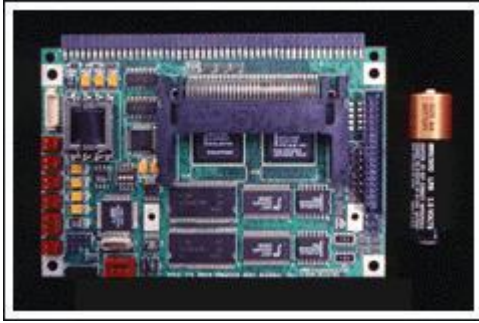
APPX Release 4.1



PPX Software, Inc. recently released APPX version 4.1. It features an enhanced GUI design and runtime capability, and includes the capability to design and run applications with either a character mode interface or a Java-based graphical user interface. Database interfaces have also seen improvement and include new functionality as well as improved performance.

Contact: APPX Software, Inc., 11363 San Jose Boulevard, Suite 301, Jacksonville, FL 32223, 800-879-2779 (toll-free), 904-880-6635 (fax), info@appx.com, <http://www.appx.com/>.

The Bitsy



The Bitsy, by Applied Data Systems, is the first commercially available 3" x 4" embedded StrongARM SA-1110 platform. The company's smallest single-board computer, it is targeted for the PDA/handheld markets. The Bitsy is built around the Intel 32-bit StrongARM™ SA-1110 RISC processor and SA-1111 companion chip, and its processor runs at less than 450MW at 206MHz. The system runs from unregulated 6-12VDC, includes a battery charger and supports a backup battery. The Bitsy supports up to 16MB of SDRAM and 32MB of flash memory, includes a Type II PCMCIA slot and supports an array of operating systems, being one of the first OEM handheld embedded computers to support Linux.

Contact: Applied Data Systems, Inc., 9140 Guilford Road, Columbia, MD 21046, 301-490-4007, <http://www.flatpanels.com/>.

The 2.5-Inch ATX25

BiTMICRO NETWORKS introduced a 2.5-inch Ultra EIDE E-Disk, the ATX25, designed for embedded systems. This flash-based drive features a 4.3GB capacity and greater than 14 MB/sec sustained random read/write rates. The company claims it is the fastest 2.5-inch ATX interface flash drive available.

Contact: BiTMICRO NETWORKS, 45550 Northport Loop East, Fremont, CA 94538-6481, 510-623-2341, 510-623-2342 (fax), info@bitmicro.com, <http://www.bitmicro.com/>.

Abria SQL Standard



AbriaSoft offers a turnkey installation of MySQL 3.22. The bundle includes Webmin, a web-based administration interface for UNIX systems, as well as Apache web server, version 1.3.12 and Perl 5.00503. The package claims an install time of ten minutes.

Contact: AbriaSoft, 39465 Paseo Padre Parkway, #3450, Fremont, CA 94538, 877-922-7429 (toll-free), 510-623-9726, 510-249-9125 (fax), <http://www.abriasoft.com/>.

Tuxtops Easy Install Linux Software



Tuxtops now offers preconfigured versions of the Red Hat 6.2 and Debian Linux distributions. The Tuxtop versions, available on CD-ROM, make for quick and easy installs.

Contact: Tuxtops, Inc., 1253 Lakeside Drive, Suite 300, Sunnyvale, CA 94085, 877-735-0638, 408-585-3429 (fax), <http://www.tuxtops.com/>.

InstallAnywhere 3.5



InstallAnywhere allows software producers to create customizable solutions for the process of installing and configuring software across multiple platforms. The latest version features several enhancements, including Java 1.3 compatibility and support for all Linux platforms.

Contact: Zero G Software, Inc., 514 Bryant Street, San Francisco, CA 94107, 415-512-7771, 415-723-7244 (fax), <http://www.zerog.com/>.

Forte for Java Community Edition Software

This IDE from Sun Microsystems provides a suite of wizards, utilities, and templates, extensive support of open standards and an extensible, object-oriented framework that can be extended with plug-in modules. Download for

free or order the CD-ROM at tm0.com/sbct.cgi?s=70626416&i=248161&d=409605.

Contact: Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, 800-786-404 (toll free), <http://www.sun.com/>.

ViewletBuilder2, Version 2.1

Qarbon.com announces a Linux version with their latest release of this development tool for creating Viewlets. The new version also includes greater stability and better audio quality.

Contact: Qarbon.com, 84 West Santa Clara Street, Suite 790, San Jose, CA 95113, 408-792-3800, 408-792-3808 (fax), sales@qarbon.com, <http://qarbon.com/>.

CodeWizard

CodeWizard is ParaSoft's tool for the enforcement of coding standards for C and C++ programmers. The newest version includes additional standards for embedded developers, new rules for C programmers and additional compiler support.

Contact: ParaSoft, 2131 S. Myrtle Avenue, Monrovia, CA 91016, 888-305-0041 (toll free), info@parasoft.com, <http://www.parasoft.com/>.

Linux Planet 1.2

This product offers a Linux-based development platform utilizing Motorola embedded PowerPC and featuring USB host and slave capability.

Contact: Embedded Planet, 749 Miner Road, Cleveland, OH 44143, 440-646-0077, 440-461-4329 (fax), <http://www.embeddedplanet.com/>.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.