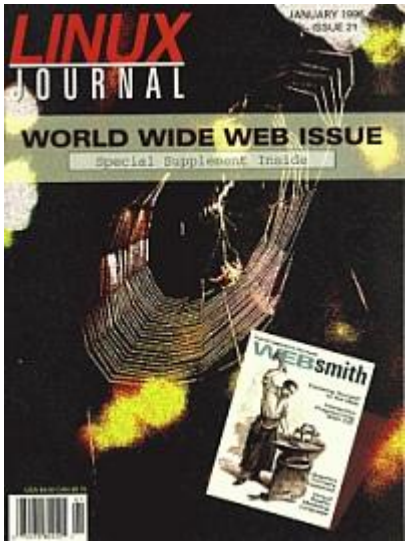
Linux Journal Issue #21/January 1996



Features

An Introduction to Python *by Jeff Bauer*
Do you need help in the rapid development of applications? Jeff explains why Python could be the language for you.
Using Linux and DOS Together *by Marty Leisner*
Taking the pain out of installing Linux on a machine for the first time.

News and Articles

CVS: Version Control Beyond RCS *by Tom Morse*
Ever have conflicts when more than one person works on the same file? CVS offers a solution.
The Quintessential Linux Benchmark *by William van Dorst*
All about the “BogoMips” number displayed when Linux boots.

Columns

From the Publisher WEBsmith
Letters to the Editor
From the Editor
Linux Systems Administration Maximizing Linux Security, Part I
New Products
Take Command The chmod Command
Kernel Korner Linux on Alpha AXP
Book Review Linux Universe

An Introduction to Python

Jeff Bauer

Issue #21, January 1996

Python is an extensible, high-level, interpreted, object-oriented programming language. Ready for use in the real world, it's also free.

If you've been programming on a Linux system, you may be coding in C or C++. If you're a systems administrator, you may be programming in perl, Tcl, awk, or one of the various (sh/csh/tsh/bash) shell scripting languages. Maybe you wrote a script to do a particular job, but now find that it doesn't scale up very well. You might be writing C applications, but now wish you didn't have to be bogged down in the low-level details. Or you may simply be intrigued by the possibility of doing high-level, object-oriented programming in a friendly, interpreted environment.

If any of the above applies to your situation, you may be interested in Python. Python is a powerful language for the rapid development of applications. The interpreter is easily extensible, and you may embed your favorite C code as a compiled extension module.

Python is not one of the research languages which seem to get promoted solely for pedagogical reasons. It is possible to do useful coding almost immediately. Python seems to encourage object-oriented programming by clearing the paths, rather than erecting parapets.

Getting Started

To execute the standard hello program, enter the following at the command line:

```
$ python
Python 1.2 (Jun  3, 1995) [GCC 2.6.3]
Copyright 1991-1995 Sitchting Mathematisch Centrum, Amsterdam
>> print "hello, bruce"
hello, bruce
>> [CONTROL]-D
```

Most Python programs, though developed incrementally, are executed as a normal script. The next program illustrates some extensions to the original. The new version will identify who you are based on your user account in /etc/passwd.

```
 1  #!/usr/local/bin/python
2
3  import posix
4  import string
5
6  uid = `posix.getuid()`
7  passwd = open(`/etc/passwd')
8  for line in passwd.readlines():
9      rec = string.splitfields(line, `:')
10      if rec[2] == uid:
11          print `hello', rec[0],
12          print `mind if we call you bruce?'
13          break
14  else:
15      print "I can't find you in /etc/passwd"
```

A line-by-line explanation of the program is as follows:

- 1 --- Command interpreter to invoke
- 3-4 --- Import two standard Python modules, **posix** and **regsub**.
- 6 --- Get the user id using the **posix** module. The enclosing backticks (**`**) tell Python to assign this value as a string.
- 7 --- Open the /etc/passwd file in read mode.
- 8 --- Start a **for** loop, reading in all the lines of /etc/passwd. Compound statements, such as conditionals, have headers starting with a keyword (**if**, **while**, **for**, **try**) and end with a colon.
- 9 --- Each line in /etc/passwd is read and split into array **rec[]** based on a colon **:** boundary, using **string.splitfields()**
- 10 --- If **rec[2]** from /etc/passwd matches our call to **posix.getuid()**, we have identified the user. The first 3 fields of /etc/passwd are: **rec[0] = name**, **rec[1] = password**, and **rec[2] = uid**.
- 11-12 --- Print the user's account name to stdout. The trailing comma avoids the newline after the output.
- 13 --- Break the **for** loop.
- 14-15 --- Print message if we can't locate the user in /etc/passwd.

The observant reader will note that the control statements lack any form of BEGIN/END keywords or matching braces. This is because the indentation defines the way statements are grouped. Not only does this eliminate the need for braces, but it enforces a readable coding style. No doubt this design feature will turn off a few potential Python hackers, but in practice, it is useful. I can think of numerous times I've spent tracking bugs in C resulting from misinterpreting code that looked like any of these fragments, usually deeply nested:

```
if (n == 0)
    x++;
    y--;
z++;

if (m == n || (n != o && o == q)) { j++; }
    k++;
q = 0;
while (y--)
     *ptr++;
     if (m == n) {
         x++;
     }
```

A coding style enforced in the language definition would have saved me much frustration. Python code written by another programmer is usually very readable.

Libraries

You might object that we did a lot of work in the program above just to demonstrate Python language features. A better method would be to use the pwd module from the standard Python library:

```
print `hello', pwd.getpwuid(posix.getuid())[0]
```

This points out another nicety about Python that is critical for any new language's success: the robustness of its library. As mentioned earlier, you may extend Python by adding a compiled extension module to your personal library, but in most cases *you don't have to*.

Take the **ftplib** module for instance. If you wanted to write a Python script to automatically download the latest FAQ, you can simply use ftplib in the following example:

```
#!/usr/local/bin/python
from ftplib import FTP
ftp = FTP(`ftp.python.org')     # connect to host
ftp.login()                     # login anonymous
ftp.cwd(`pub/python/doc')       # change directory
ftp.retrlines(`LIST')           # list python/doc
F = open(`python.FAQ', `w')     # file: python.FAQ
ftp.retrbinary(`RETR FAQ', F.write, 1024)
ftp.quit()
```

Python has numerous features which make programming fun and restore your perspective of the design objectives. The language encourages you to explore its features by writing experimental functions during program development. Several notable Python features:

- Automatic memory management. No malloc/free or new/delete is necessary—when objects become unreachable they are garbage-collected.
- Support for manipulating lists, tuples, and arrays

- Associative arrays, referred to as “Dictionaries” in Python
- Modules to encourage reusability. Python comes with a large set of standard modules that may be used as the basis for learning to program in Python.
- Exception handling
- Classes

Python Has Real Class

With the next example, I'll try to demonstrate some of these features. The StackingThings class will allow the user to stack items on top of each other until a breaking point is reached.

```
1       #!/usr/local/bin/python
2
3       StackingException = `StackingException'
4
5       class StackingThings:
6       names = (`llama', `spam', `16 ton weight',
7       `dead parrot')
8       weights = {}
9       weights[`llama']         =     300
10      weights[`spam']          =       1
11      weights[`16 ton weight'] =   32000
12      weights[`dead parrot']   =       2
13      breakpt = {}    # breaking points
14      breakpt[`llama']         =     200
15      breakpt[`spam']          =    1000
16      breakpt[`16 ton weight'] = 1000000
17      breakpt[`dead parrot']   =      15
18
19      def _ _init_ _(self):
20      self.items_stacked = []
21      def add(self, item):
22              if item not in self.names:
23                      raise StackingException, \
24                              item+`not a stackable object'
25      self.items_stacked.insert(0, item)
26      try:
27              self.test_strength(item)
28                except StackingException, val:
29                      print item, val
30      def test_strength(self, item):
31          wt = 0
32          bp = 1000000
33          for i in self.items_stacked:
34              wt = wt + self.weights[i]
35              if wt > bp:
36                 self.items_stacked.remove(item)
37                 raise StackingException, \
38                     `exceeds breaking point!'
39              bp = self.breakpt[i]
40
41      # user code to test StackingThings class
42
43  s = StackingThings()
44
45  s.add(`llama')
46  s.add(`spam')
47  s.add(`spam')
48  s.add(`spam')
49  s.add(`dead parrot')
50  s.add(`16 ton weight')
51
52  print `items stacked = ', s.items_stacked
53
54  try:
```

```
55      s.add(`bad object')
56  except StackingException, msg:
57      print `exception:', msg
```

This script produces the following output:

```
16 ton weight exceeds breaking point!
items stacked =  [`dead parrot', `spam', `spam',
     `spam', `llama']
exception: bad object not a stackable object
```

The StackingThings class itself consists of 3 methods: **_ _init_ _()**, **add()**, and **test_strength()**. When initiating StackingThings, we use the special **__init__** method to create its initial state by initializing the list of stacked items: **items_stacked = []**. The **add()** method is essentially the only method that is accessed by the user of StackingThings. And **test_strength()** is called by **add()** to verify that we have not exceeded our breaking point.

The first argument to each method in our example is called **self**. This is just a convention, but it makes our code much more readable. The first argument to a Python method is used in a somewhat similar fashion as **this** keyword in C++.

Python provides for exception handling, both built-in (i.e. **ZeroDivisionError**, **TypeError**, **NameError**, etc.) and user-defined exceptions. The latter is especially useful in developing robust classes. Python uses the **try**/**except** syntax for exception handling:

```
try:
    DenominateZero()
except ZeroDivisionError, val:
    print `Whoops:', val
```

Our **add()** method is used to try an exception in **test_strength()** and raise an exception when we pass it an illegal stacking item.

Two of the built-in methods for Python lists that are demonstrated in the example on lines 25 and 36 are **insert()** and **remove()**. Other supported operations on list objects include **append()**, **count()**, **index()**, **reverse()**, and **sort()**.

The data attributes may be accessed by the methods of the class as well as the user code: *Either* **print self.names** (within a class method) *or* **print s.names** (from the user code) will print the list of legal stacking things:

```
[`llama', `spam', `16 ton weight', `dead parrot']
```

Look It Up!

Dictionaries (associative arrays to all you awk/perl hackers) are one of the most useful Python data types. Unlike a normal array, which is indexed by number,

associative arrays are indexed by strings. The value of this utility is worth describing in some detail.

I frequently deal with *ICD-9-CM* codes in medical applications. These codes are usually numeric, but sometimes alphanumeric. They usually have a decimal point, but sometimes don't. Some of the codes may be further sub-divided into additional ICD-9 codes. Furthermore, codes are added and deleted periodically, but most don't change. Normally, the lookup of ICD-9 codes will be done in a relational database, but it is also convenient to use small data sets within an application. For example, given the dictionaries **icd9** and **subdivide**:

```
 x              subdivide[x]          icd9[x]
---             ------------    ------------------------
`692'                 1         `Contact dermatitis'
`692.0'               0         `Due to detergents'
`692.2'               0         `Due to solvents'
`692.7'               1         `Due to solar radiation'
`692.70'              0         `Unspecified dermatitis'
`692.71'              0         `Sunburn'
`692.72'              0         `Other: Photodermatitis'
```

We can manipulate the ICD-9 codes in the following manner:

```
for code in icd9.keys():
  if subdivide[code]:
      print `ICD-9',code,'may be further subdivided'
        else:
      print `Description for',code,`is:',icd9[code]
```

This would produce the following output:

```
ICD-9 692.7 may be further subdivided
Description for 692.70 is: Unspecified dermatitis
Description for 692.0 is: Due to detergents
ICD-9 692 may be further subdivided
Description for 692.71 is: Sunburn
Description for 692.2 is: Due to solvents
Description for 692.72 is: Other: Photodermatitis
```

Lines 8-17 of our **StackingThings** example use dictionaries, but the initialization was broken into several lines for clarity. This could be reduced to:

```
weights = {`llama':300, `spam':1, \
    `16 ton weight':32000, `dead parrot':2}
breakpt = {`llama':200, `spam':1000, \
    `16 ton weight':1000000, `dead parrot':15}
```

Finally, inheritance is provided in Python, although it is not demonstrated in this example. The derived class may override methods of its base class or classes (yes, multiple inheritance is supported in a limited form). In C++ parlance, all methods in a Python class are “virtual”.

Where Do We Go from Here?

Python is currently available in source or as a Linux binary from ftp.python.org. Various modules have already been developed and become part of the standard Python Library. To mention just a few: support for strings, regular expressions, posix, sockets, threads, multimedia, cryptography, STDWIN, Internet/WWW, Expect, and a large number of other contributions, are submitted periodically.

Python is extensible. If you can program in C, you can add a new low-level module to the interpreter. We are currently doing this at our company for a distributed database system. The Python interpreter will be the high-level command language for many of the applications.

In addition to Linux, Python runs on several other platforms: OS/2, Windows, Macintosh, and many flavors of Unix. And like Linux, all of these versions are freely available and distributable.

The documentation for Python is of a very high quality, written by Guido van Rossum, the creator of Python. Four separate user manuals in postscript format are available at the Python ftp site (see sidebar "Python Information"). These documents have also been converted to HTML and Microsoft help file formats. A Python FAQ, quick reference guide, and testimonials are also available. O'Reilly and Associates also intends to publish *Programming Python* early next year.

Python has its own active newsgroup (comp.lang.python) as well as a mailing list which receives the same messages as the newsgroup. To subscribe to the mailing list, send mail to python-list-request@cwi.nl. Various Python special interest groups have been formed: Matrix-SIG, GUI-SIG, and Locator-SIG.

Finally, The Python Software Activity ("PSA") has been established to foster the common interests of the Python development community. The PSA, unlike the GNU Project, does not do the actual development of software (although many of its members probably do), but rather acts as a clearinghouse for Python software modules developed by others. It also hosts workshops and related activities to help promote the use of the Python language. Additional information about the PSA may be obtained by visiting the Python home page: www.python.org.

Special thanks to Mark Lutz, Aaron Watters, the PSA, and, of course, Guido van Rossum.

Jeff Bauer has spent the past 16 years developing health care software. His current project involves interfacing pen-based computers with Unix systems to track clinical information.

Archive Index Issue Table of Contents

Advanced search

Using Linux and DOS Together

Marty Leisner

Issue #21, January 1996

Installing Linux on a machine for the first time is often a painful experience. There are a number of useful programs and techniques for running Linux on machines which run both DOS and Linux, some of which appeared in DOS 5. Understanding and using these techniques makes it possible to use them under DOSEMU whenever relevant.

When machines come from a factory with DOS pre-installed on them, the hard disks are normally arranged so the whole disk is the C: drive. This is very inflexible, even if you only want to run DOS, and unbearable if you also want to run other operating systems. PC partition tables support the following configurations (with a total of 4 allowed):

- One primary DOS partition
- One extended partition (containing a number of logical partitions)
- One or more non-DOS partitions (good for Linux)

Resizing Your Partitions

In order to run Linux, you typically have to repartition your disk, which is often a good idea whatever operating system you run for the following reasons on (DOS or UNIX):

- Flexibility
- Crash resistance Typically, disk problems are reserved to one logical disk on the media. By having more than one logical disk on a physical device, if anything goes haywire, it is reserved to one partition.
- Control You may not want to allocate your whole hard disk to any application. Giving sets of applications a partition limits the amount of disk space they can use. You may get “out of disk space” on partitions, but not necessarily for the whole disk.

When you want to repartition your disk, standard procedure used to be:

- Back up to floppies
- Erase your hard disk
- Repartition
- Recover your floppies

This may have worked with a few meg to back up, but now PCs normally come with 200 Mb installed in a Microsoft Windows system, with little documentation about what's important and what's not. There is a very clever utility called fips, written by Arno Schaefer, schaefer@rbg.informatik.th-darmstadt.de. It is kept on sunsite.unc.edu in /pub/Linux/system/Install/fips12.zip.

fips non-destructively shrinks your primary partition, leaving all your files in place. You run defrag (an MS-DOS program) to pack all the files into contiguous sectors and then fips to shrink your primary partition.

You can then reboot and create an extended DOS partition (with the DOS fdisk program), and use Linux to create Linux partitions (with the Linux fdisk program). fips is a wonderful tool that solves a very real problem. Please read the instructions carefully before using it; any tool which writes your partition table, like fdisk, should be used with caution.

N.B.: Be very careful using fdisk. Don't do anything destructive to your media (i.e mkfs or format from DOS) until you are sure DOS and Linux agree on where the partitions are. I've noticed very “unnice” features with the DOS fdisk program—it read the free space as 100 Mb (which was correct), but when I allowed it to make a partition with all the free space, it made a 350 Mb partition (very naughty!)

Using Extended Partitions

Extended partitions are a way around the 4 partition limit on a physical disk. An extended partition can serve as a container for more partitions, which can be DOS, Linux (native or swap) or any other type. Remember, non-DOS partitions need to be made in a non-DOS version of fdisk. Extended partitions are handy for generating more than the 4 partitions normally found. I have never seen a good discussion of adding swap space to a system—a good way is using extended partitions.

Extended partitions are a good idea—for running DOS. Each logical partition in the extended partition is given a letter by DOS (i.e. D:, E: , F:). Each of these drives can then be formatted and used under DOS. I use an application called join, which as of DOS 6 is no longer distributed with DOS, but you can still get a

copy via ftp from ftp.microsoft.com in /peropsys/msdos/public/supplmnt/. join essentially allows you to “mount drives” so you have a single hierarchical tree (like Linux).

For instance, I might configure a system with drive D: for personal stuff and E: for djgpp, a port of the GNU C compiler to DOS. Then, in my root directory on my C drive I create directories for \marty, \gnu, and in my autoexec.bat, I have:

```
join d: \marty
join e: \gnu
```

so that I don't have to deal with drive letters. I also join A: to \a, so my floppy disks appears on the tree. But if you do something like this you'll break the DOS format program and almost every DOS install/setup program for commercial software. It seems that they can't deal with anything except A:.

Unfortunately, you can't use join with network drives. More on this later when we talk about DOSEMU.

Using loadlin and config.sys

I found it effective to use loadlin, a DOS-based loader; this has the distinct advantage of always booting a working system before running Linux. My experience with LILO has been if you don't do it right, your system is only useful as a paper weight. In my MS-DOS config.sys, I take advantage of the menus, and the result is shown in Listing 1.

```
menuitem=dos
menuitem=simple.dos
menuitem=linux.1.2.8
menuitem=scandisk
menudefault=linux.1.2.8,5

[linux.1.2.8]
SHELL=C:\loadlin\loadlin.exe \loadlin\zimage.128 root=/dev/hdc2 -v ro

[simple.dos]

[dos]
DEVICE=C:\DOS\HIMEM.SYS
DEVICEHIGH=C:\MAGICS20\CDIFINIT.SYS /T:X
DEVICEHIGH=C:\MTM\MTMCDAI.SYS /D:MTMIDE01
DEVICEHIGH /L:2,12048 =C:\DOS\SETVER.EXE
DOS=HIGH
STACKS=9,256

[scandisk]
SHELL=c:\dos\scandisk.exe /all /checkonly
```

I boot Linux with a delay of 5 seconds, the advantage being that the system can always boot DOS and will work in some capacity. I find this preferable to using LILO and modifying the master boot record on your hard disk (if you do anything wrong, you need to boot from floppy to recover).

One can easily select several kernels and/or configurations from the command line. Using loadlin, you have to make a compressed kernel (make zImage), and then put it on a DOS partition. I find this strategy effective even when installing Linux the first time (instead of dealing with a boot and root floppy, the system can boot the kernel with only a root floppy needed). You can easily add to the menu to have several different kernels to boot from. Remember, you can use the **rdev** utility to build defaults (like the root device) into the kernel.

In your autoexec.bat you can use the strategy:

```
goto %config%

:simple.dos
PATH=C:\marty\bin;C:\gnu\bin;C:\dos;C:\
goto end

:dos
SET SOUND16=C:\MAGICS20
C:\MAGICS20\SNDINIT /b
SET BLASTER=A220 I7 D1 T4
C:\DOS\SMARTDRV.EXE 512 512 /C
C:\DOS\IMOUSE.COM
PROMPT $p$g
SET PATH=c:\gnu\bin;C:\MARTY\BIN;C:\WINDOWS;C:\DOS;

SET TEMP=C:\DOS
JOIN d: \marty
JOIN f: \gnu
goto end

:end
```

The **simple.dos** setting is conceptually the same as booting Linux in single user mode. I find it very useful for debugging a DOS system. If you want, you can add config.sys menu entries to boot different kernels, boot Linux in single user mode, boot Linux from floppies, etc.

The UMSDOS File System

In the standard Linux kernel configuration, the UMSDOS file system isn't enabled. UMSDOS has a number of major advantages if you need file systems to be shared between Linux and DOS. It retains full Unix semantics, so you don't have to always be handicapped by DOS problems such as:

- lack of links
- restrictions of 8+3 file naming conventions
- restrictions of characters in file names
- one date (instead of access/change/modify time)
- lack of owner/groups

Using UMSDOS you can take advantage of a file system shared between DOS and Linux, with the appearance of being a Linux file system when you run Linux. If you want files to be portable between MS-DOS and Linux, restrict

yourself to DOS filenames (8+3 characters). Don't use links if you want the files to appear under DOS. With a Linux file system, it's easy do things like create “dot files”, do **gzip-r** on trees, and create links and backup files. Any file is readable in MS-DOS; however, if you don't conform to the MS-DOS file naming conventions, files are “munged” (that is, their names are squeezed to fit within the 8+3 namespace). This munging is similar to what happens in mfs; those who use PC-NFS are probably familiar with this.

When you start running the UMSDOS file systems, remember to run the application called umssync, which creates consistency between the **--linux-.---** files and the directory contents. You can have problems if you add or delete files under DOS without Linux knowing about it. Call umssync from /etc/rc.d/rc.local or /etc/rc.d/rc.M after the mount takes place, and this shouldn't be a problem.

I've noticed a problem in UMSDOS files systems—the mount points are owned by root, only writable by root, and the date is the beginning of the epoch. A simple workaround is after mounting, do chown/chmod to the mount points as appropriate (in your /etc/rc.d/rc.local file. Also, I find it useful to occasionally run scandisk from DOS (notice the scandisk target in config.sys).

There is a performance penalty for DOS and UMSDOS file systems compared to normal ext2. The penalty becomes severe if you have several hundred files in a single directory (when you do an ls, get a cup of coffee). What I've noticed is sequential I/O (with a tester called Bonnie) is marginally faster on ext2 than UMSDOS.

But UMSDOS is ideal if you're doing work with DOSEMU. You put DOS files on UMSDOS partitions, and you can easily access them from DOS, DOSEMU or Linux. If they keep within the DOS file system bounds of 8+3 characters, they look the same on both DOS and Linux. UMSDOS partitions provide a big advantage when sharing files with DOS (much more so then the MSDOS file system, since it treats Linux files as Linux files), but performance has to be watched.

DOSEMU's View of File Systems

DOSEMU can access files in several different ways, which integrate with DOS and Linux in different ways. The methods are:

image
: A file which is arranged to look like a DOS hard disk. It is a “virtual” hard disk stored in a file.

partition

Direct access to an MS-DOS partition. If the partition is also being used on Linux, it should not be writable. Be aware that you can use mounted partitions as DOSEMU file systems, which can destroy the file system. It is safest if they are both used readonly; if you want to make them writable you should only make one of them writable at a time. In addition, if the DOS partition is writable from DOSEMU, multiple DOSEMU sessions can cause the same kinds of filesystem destruction.

whole disk
: Use the whole disk directly. Be vary careful with this. When used, it is useful to set it [cw]readonly[ecw].

redirected access Access any Linux directory via a redirector. This is extremely interesting—read on to learn more about this.

Typically, DOSEMU boots off a small image file (a specially constructed file which appears to DOSEMU like a hard disk, with its own file system and master boot record). Floppy disks are treated like conventional floppy disks. DOSEMU can read them—and you need a bootable MSDOS floppy to start the process. To start setting up the virtual hard disk as C: drive, you first boot off the bootable MSDOS floppy, and then do:

```
A>fdisk /mbr
A>sys c:
```

Then you can boot off the virtual hard disk C:. This is covered more fully in the DOSEMU documentation.

The image hard disk is often used just to get DOSEMU going. You can treat this image as a large virtual hard disk, but the disadvantage is you can only access this disk from DOSEMU. The other forms, which will be explained, can all be accessed from Linux, and MS-DOS partitions can be accessed from raw MS-DOS.

DOSEMU supports whole disk access (such as /dev/hdc) and partition access. I have never used whole disk access and there doesn't appear to be a good reason to do it. I have, however, used partition access. Those partitions cannot be mounted by Linux at the same time, since DOSEMU manipulates the physical partition, which will confuse the kernel, and potentially destroy the partition. DOSEMU needs to have access to the physical partitions (you have to make sure you have the permission to read and write).

The most interesting method I've found is the redirector. This allows you to treat a Linux file system as a network drive. If you redirect the root of your Linux file system, you can easily access all your linux files in DOSEMU. If you

have NFS mounts or an auto mounter running, you can even traverse to other machines seamlessly. Note that everything it finds it must convert to an 8+3 MS-DOS namespace.

It works well if no munging is necessary. However, you may see this:

```
F:\dir a*

 Volume in drive F is s2/dist/X11
 Directory of F:\

ARCH                        05-26-95   1:01a
ACM-4~YX GZ         971,391 06-02-95  11:02p
ARENA    TAR        604,160 05-19-95   9:43p
ARENA~D0 GZ         530,468 05-22-95   8:35p
```

instead of

```
leisner@compudyne$ ls -d a*
acm-4.7.tar.gz   arch/
arena-96.tar.gz  arena.tar
```

Most of the time you can figure out what is meant. I've noticed some problems identifying files which are spelled the same way except for the case of some characters. On Unix they're distinct, but DOS has no notion of case in file names (you will have a problem with makefile and Makefile, for instance).

Booting DOSEMU on Linux

You shouldn't do much on your virtual hard disk beyond booting. I found it effective to have a directory ~/dos. My config.sys on the virtual hard disk looks like this:

```
# make sure we support ems
devicehigh=c:\ems.sys
# the last drive is m, it can range up to z:
# the default is f:
lastdrive=m
FILES=40
SWITCHES=/f
# make a copy of c: drive on l:
install=c:\subst.exe l: c:\
# this is the fun part
# change the concept of c: drive
install=c:\lredir.exe  c: LINUX\fs>{home}\dos
```

The last few lines are the most interesting. I'm making the virtual hard disk accessible to dosemu through the L: drive. If you want to “lock down” the virtual hard disk, you make the file readonly with the **chmod** command. Then, continue booting from the user's ~/dos directory (where an autoexec.bat is expected). This means that autoexec.bat is just a regular Linux file. You can edit it with any Linux editor, but you have to remember to put **\r** at the end of each line (that's a control-M character; in vi do

control-v-m,

in Emacs do control-q-m). In my autoexec.bat I have:

```
lredir f: linux\fs\${PWD}
lredir e: linux\fs\
set PATH=e:\dos\gnu\bin;e:\dos\c\dos;c:\;c:\bin
f:
```

The syntax **${...}** allows environment variables to be substituted. PWD is the current working directory. Bash doesn't normally export it for you; I explicitly add

```
export PWD
```

to my .bashrc file.

I just map the F: drive to my current working directory. This is very convenient, because when I'm working with DOS files on Linux, I can start up DOSEMU wherever I am at the moment.

I map my entire filesystem to E:. This makes almost any file accessible under Linux also accessible under DOSEMU. This includes NFS files.

Some programs have a problem with a redirector, since it acts as a network drive. For these programs, you need to use either partition access, image access or a ram disk.

Booting from the Installed DOS System and Win95

Extending the above scenario further, we can actually boot from a DOS hard disk using

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

This has a number of advantages—primarily the virtual hard disk does not have to be created and maintained (note the virtual hard disk is only readable within DOSEMU, making maintenance cumbersome). DOSEMU allows you to select the extension for the system files (config.sys and autoexec.bat) either in the configuration file (using EmuSys or EmuBat) or from the environment (using AUTOEXEC and CONFIG). This boot disk isn't writable, so switch to a writable C: drive with lredir.

I typically have a config.sys file for DOSEMU called config.emu. In it I just change the C: drive (from the virtual hard disk) to a ~/dos directory, and have an autoexec.bat file there. I also have links to commonly used DOS programs (i.e. command.com).

Win95 throws some curves into this scheme. I've been using Win95 since the official release and am favorably impressed with it (anything could improve on Windows 3.1 problems). Win95 uses the file MSDOS.SYS to control the boot process as another ASCII configuration file. In order to activate a config.sys menu to either boot DOS or Linux, the following works in MSDOS.SYS:

```
[Options]
Logo=0
BootMulti=1
BootGUI=0
BootDelay=0
```

In this case, after you run Linux, booting DOSEMU will allow you to run DOS Version 7.

You can also run an older DOS (if this was an upgrade) if you press F4 when it starts booting. But in this case, if you boot Linux and then start up DOSEMU off the DOS hard disk, the boot loader gets hopelessly confused, since it shuffles files like msdos.sys, config.sys, and autoexec.bat between Win95 and an older DOS system, putting the appropriate file in the appropriate place for the appropriate DOS (Win95 config files end in .w40, and older DOS files end in .dos). Obviously, you aren't expected to run DOSEMU under Linux!

Conclusions

I use DOS occasionally, but do a lot of work in MS-DOS since I'm working on DOSEMU and an alpha djgpp. I have found that you can do very flexible things with your partitions through extended partitions, and that Linux treats DOS filesystems quite nicely (especially UMSDOS).

I've found cross-development of MS-DOS applications to be ideal for DOS software development, you can write portable software and try it on Linux—then use Linux compilers to generate .EXE djgpp files and run the djgpp binaries in DOSEMU.

Marty Leisner (leisner@sdsp.mc.xerox.com) is a professional programmer for Xerox Corporation who was first exposed to Unix on a PDP 11 running V7.

Archive Index Issue Table of Contents

Advanced search

CVS: Version Control Beyond RCS

Tom Morse

Issue #21, January 1996

If you have tried version control, but are frustrated by the need to set explicit locks every time you want to edit a file, or perhaps dislike being unable to edit a file when other developers have already locked a file, CVS is for you.

CVS (Concurrent Versions System) is a version control system. With CVS, developers are able to review the change history of any source file, retrieve any revision of a particular file, avoid overwriting one another's changes, and keep track of releases and the file revisions that go along with them.

First, a short introduction to some of the concepts of CVS. All the sources for a project are stored in a central location called the repository. The sources in the repository are organized into modules. Typically each module represents a separate project. A module can represent any number of files and directories.

When a developer wishes to work with a particular set of sources he uses CVS to retrieve a local copy of them. This local copy can represent the most recent revision of each file or some past revision.

Typical version control systems require a user to “lock” each file he wishes to edit, preventing other developers from editing these files until the first user has committed his changes. This can be quite time consuming as developers either wait for one another to commit changes, or work simultaneously and manually merge the changes together later on. CVS solves this problem by allowing concurrent editing.

Concurrent editing allows two or more developers to work concurrently on the same files. Ordinarily, concurrent editing leads to one developer overwriting another's previously committed changes.

CVS prevents overwrites by forcing a developer to merge into his local copy any changes that have been committed to a file in the repository since he retrieved

his local copy of that file. Only then can the developer commit his changes. The process is mostly automatic, but if, for instance, changes have been made to the same line of code in both the repository and the local copy, manual resolution of the conflict is necessary.

A day in the Life of…,

Let's look in on our software development team in action. Due to the confidentiality of their project, all the file names have been changed to protect the innocent. Their project is to “build a better mouse trap.”

Fixing Bugs

Fezzik, though primarily a documentation writer, *(This explains why all our team's manuals are in rhyme.)* does some detecting and squashing of software bugs in the course of his work. Let's watch as Fezzik fixes a critical bug in a previous release of the product.

Version 4.0 of the product was released a few months ago. Now a critical bug has been found by a very important customer. The bug must be fixed in the released version of the software, a point release made for the customer, and the fix must be merged into the working version of the software.

In investigating the problem, Fezzik has determined that the bug was introduced between the 3.3 and 4.0 versions of the software. His first step is to get a copy of the 4.0 sources. He does this with the command:

```
cvs checkout -r PROD_REL4-0 mousetrap
```

This will check out all of the directories and source code files for the entire project. The version of each source file will be the version that was included in the release with the name PROD_REL4-0, exclusive of any changes that have been to that source file since the release.

Now Fezzik has a new directory called mousetrap. In this directory are all the project sources as they were when the release 4.0 was made. Because development on the project has continued since release 4.0, Fezzik must start a new branch on the revision tree from this point. He uses the following commands:

```
cvs tag -b POINT_RELS_REL4-0
cvs update -r POINT_RELS_REL4-0
```

The revision tree will look like this:

```
                    |HEAD - dev for next full release
```

```
POINT_RELS_REL4-0\  |
                   \ |
                    \|
                     +PROD_REL4-0
                     |
                     |
                     |
                     +PROD_REL3-3
                     |
                     |
```

Fezzik could have created the branch tag before checking out the project, eliminating one step from the method he used. The following sequence would produce the same effect as above:

```
cvs rtag -b -r PROD_REL4-0 POINT_RELS_REL4-0 mousetrap
cvs checkout -r POINT_RELS_REL4-0
```

CVS has many command and option combinations. Therefore, there is often more than one way to produce the same or similar results. This flexibility allows CVS to be adapted to a wide range of development processes.

Fezzik suspects that the file cheese.c contains the bug, so he decides to view all the changes made to that file between releases 3.3 and 4.0. To do this he uses the diff command:

cvs diff -r PROD_REL3-3 cheese.c

This produces a listing of the differences between the file cheese.c in our current directory and the revision used in PROD_REL3-3. Fezzik could have specified a second revision with a

-r PROD_REL4-0

and had the same effect.

In reviewing the differences, Fezzik finds what he believes to be the bug he is looking for. He would like to clear any modifications he plans with the person who made the change, so he looks at the log file for cheese.c.

cvs log cheese.c

He sees that only Buttercup modified this file between the two releases, so he is able to talk to her about the fix. After making the fix and testing it, Fezzik is ready to commit his changes. He issues the following command from the top directory of the project:

cvs commit -m "Fixed bug #1202"

Fezzik did not specify a specific file to commit, so this command will commit all modified files in this directory and, recursively, in all of its subdirectories. Each file committed will have the message “Fixed bug #1202” added as a log message. Since this fix will be sent to a customer, Fezzik decides to tag the set of files that will be sent:

cvs tag PROD_REL4-0-1

Now Fezzik needs to create the patch file to be sent to the customer:

cvs rdiff -r PROD_REL4-0 -r PROD_REL4-0-1 mousetrap > patch4.0-4.0.1

This creates a Larry Wall format patch file which the customer can feed into the patch program to update his sources. The patches will update the customer's 4.0 sources to the new 4.0.1 sources.

Now Fezzik needs to merge the fix into the current development sources. First he updates his sources to the latest revisions on the main thread by using this command:

cvs update -A

This has the same effect as deleting the local copy of the module, and doing a new checkout of the module, thereby getting the latest revisions.

Then he has CVS automatically merge in the changes from the 4.0.1 point release with:

cvs update -j PROD_REL4-0-1

CVS will automatically merge the changes Fezzik made on the branch into these latest sources. If the merging of the two sets of sources causes conflicts, CVS will announce this. At the conflict points in the file, there will be delimited regions containing the text from both sources. These regions will need to be merged manually.

Once everything is merged, Fezzik can commit all of the changes using another commit command:

cvs commit -m "Merged in fixes in PROD_REL4-0-1"

Contention

Now we turn to Westley, who will demonstrate how CVS is used when working on a multiple-person project.

Westley has been off the project for a month *[not too surprising, what with his having been dead and all]*. The first thing he needs to do is bring his sources up to date.

cvs update

Since he has not modified anything since his last commit everything simply updates; no conflicts could exist. Now Westley would like to see what has happened since his departure on April 10. He uses the following command to view all the log messages for the commits that were made after April 10th:

cvs log -d>4/10 -b

After reviewing the log file he feels caught up, so he jumps right into modifying sources. During the course of his work he needs to create a new file, happiness.c. After he creates the file happiness.c on the disk, he issues the following command:

cvs add happiness.c

Westley also obsoletes a file, and after deleting the file, he issues the following command:

cvs delete agony.c

Both the add and delete commands will not take full effect until the CVS commit is done.

After a day of work, Westley has added a new feature and is ready to commit his modifications. He issues the following command:

cvs commit -m "Made a number of wonderful improvements"

CVS informs him that someone else has already committed changes to some of the files that he has made changes to, so he must update his files with the other developer's changes before committing his own. A simple call to the update command will do this:

cvs update

After manually editing to resolve any conflicts between his changes and the changes already made to the repository, Westley tests his new feature again. Everything looks okay, so he attempts his commit again:

cvs commit -m "Made a number of wonderful improvements"

This time it works. This method of working is quite an improvement over many systems where a person must first lock a file before it can be edited, and anyone else wanting to edit the file must wait until the first person has committed changes and unlocked the file.

Releases

Now Buttercup will demonstrate how to release a new version of the software. She begins by checking out the latest sources.

cvs checkout mousetrap

After verifying that these sources are the exact versions that should be released, she tags the release:

cvs tag PROD_REL4-1

Then she creates a set of directories containing all of the sources to be delivered by using the export command:

cvs export -R PROD_REL4-1 mousetrap

This will create a directory structure filled with only the correct sources and none of the CVS administration directories or files.

Summary

The full power of CVS exceeds the scope of this article, but I hope I have provided enough of a taste to entice you to try CVS. We have been using it for 5 months at Lernout & Hauspie, and are more than pleased with its performance.

Per Cederqvist has written an excellent introduction to CVS called *Version Management with CVS*. It can be found on the WWW by following links on the CVS page at http://www.winternet.com/~zoo/cvs/. This manual will be included with the next version of CVS, version 1.4.

For prior adventures of this software development team, see (or read) William Goldman's *The Princess Bride*.

Tom Morse (tmorse@lhs.com) has been working in Unix for the past 10 years and is currently employed at Lernout & Hauspie Speech Products. When he is not chained to a computer, he spends his time mountain biking, hiking, and attempting to learn Dutch.

Archive Index Issue Table of Contents

Advanced search

The Quintessential Linux Benchmark

Wim van Dorst

Issue #21, January 1996

When Linux boots, it displays a “BogoMips” number. What does that mean? Is the number displayed correct? What use is the infromation? This quintessential part of Linux is demystified in this article.

Some device drivers in the Linux kernel need timing delays. Either they need a very short delay, or the delay must be very accurately determined. A simple non-busy loop cannot do this. Therefore, Linus Torvalds added a calibration in the boot procedure to predetermine how often a specific busy-loop algorithm can be calculated in one second. This predetermined value, called **loops_per_second**, is used in the device drivers to delay for precisely measured times.

For fun, Linus also added a print statement presenting this predetermined value (divided by 500,000) as BogoMips. Linus apparently loves it when millions of Linux users are gazing at their computer, baffled by these bogus MIPS. Note that BogoMips have nothing to do with the million instructions per second that the name suggests; that is why they are bogus.

The only serious reason for paying attention to the BogoMips presented on booting Linux is to see whether it is in the proper range for the particular processor, its clock frequency, and the potentially present cache. 486 systems are especially prone to faulty setups of RAM caching, turbo-buttons, and such things.

Which Value to Expect

People continuously ask on Usenet: “I have an *XYZ* CPU running at *clock* MHz. How many BogoMips should it do?” The answer can be calculated from the following table:

```
Intel/AMD 386SX    clock * 0.14 (± 0.01)
Intel/AMD 386DX    clock * 0.18 (± 0.01)
```

```
Cyrix/IBM 486      clock * 0.33 (± 0.04)
Intel/AMD 486      clock * 0.50 (± 0.01)
Pentium            clock * 0.40 (± 0.01)
680x0              (insufficient data)
PowerPC            clock * 0.77 (± 0.02)
Mips               (insufficient data)
Alpha              clock * 0.99 (± 0.01)
```

From the above calculation we see several important points. First of all, the Intel and AMD 486 CPUs are not showing the same BogoMips as Cyrix and IBM 486 CPUs. This does not mean that they have a different perceived performance; it just means that they process the busy-loop algorithm differently.

The table also shows that the Pentium processor doesn't have the expected extrapolated multiplication factor. This is due to the fact that the specific busy-loop algorithm is not optimized for the parallelism of the Pentium processor.

The BogoMips calculations for the Motorola, PowerPC, Mips, and Alpha processors are similar to the Intel type processors calculations. Because the non-busy loop algorithm is coded in Assembler, however, they cannot be identical. It clearly shows that comparison of BogoMips between CPUs is really bogus, even between two different Intel type CPUs.

Last but not least, you see an allowed variation in the multiplication factor of about 0.01. The BogoMips calculation loop is “quantizised” (Linus's term), so it is likely that you will get exactly the same number all the time. Yet, if the speed is just on the edge, small variations, such as different lengths for interrupts, will cause your machine's BogoMips to vary.

The Most Frequently Asked Question

“When I boot Linux I get the message:

```
Calibrating delay loop.. ok - 23.96 BogoMips
failed
```

Where or why has the calibration delay loop failed?”

The obvious answer is that it *didn't* fail. If it had failed the text would have been:

```
Calibrating delay loop.. failed
```

What likely *did* fail was a driver for some gadget which may not be in the machine. The point is that just after calculating the BogoMips, all device drivers are initialized: first the SCSI devices, then Net devices, etc. Any failure in these initializations is duly reported. The AHA152x drive is noted for such failures. Other effects of failing drivers (and not of failing BogoMips calculations) are systems crashes, long waits, and complete system lock-ups.

Since Linux 1.2, many error messages have improved, so upgrade to at least that version to find out which particular driver is failing.

Standalone BogoMips Program

For people without Linux systems, or for those people who do not want to reboot their system time and again, a stand-alone program for calculating BogoMips is available in the standard archives (e.g., on sunsite.unc.edu in /pub/Linux/system/Status/bogo-1.2.tar.gz). On Linux, by default, it runs the same code that is used in the Linux kernel while booting, but runs as a user program. Note that due to system load, values calculated with the stand-alone program may be lower than expected for the CPU you are running, and lower than reported during boot. For the non-Linux systems, a portable C version is available that may run on any system that supports an ANSI C compiler and library.

Complete Reference Table: BogoMips Mini-HOWTO

The BogoMips mini-HOWTO gives a full table of reported BogoMips for various systems. More than 250 BogoMips references as reported on Usenet, or sent directly by e-mail to the maintainer, are listed with information about CPU type, clock speed, BogoMips, and the name and e-mail address of the reporter. For example, the lowest and highest BogoMips reported in the current version of The BogoMips Mini-HOWTO are:

The Lowest: H. Peter Anwin pa@nwu.edu 386SX/16 387 nocache 0.57 BogoMips
The Highest: David Mosberger-Tang davidm@cs.arizona.edu Alpha 21064A/275 273.37 BogoMips

In the BogoMips mini-HOWTO, values that do and do not comply with the aforementioned BogoMips calculation methods are listed. The non-complying group is named “Oddly or Faultily configured” because non-compliance does not necessarily mean that the system is faultily configured.

Benchmarking

The BogoMips may be used to see whether your system is faster than mine. Of course this is completely wrong, unreliable, ill-founded, and utterly useless, but all benchmarks suffer from this problem, so why not use it? This inherent stupidity has never before stopped people from using benchmarks, has it? [Note for the humor-challenged: no angry letters to the editor will be accepted on this point. —Ed]

Wim Dorst (Dorst) Isolde van Dorst is the beautiful daughter of the author. She is just over one year old, and is now playing around in the garden, walking over

that still unfamiliar, ticklish stuff: grass. She can be reached by e-mail at isolde@clifton.hobby.nl

Archive Index Issue Table of Contents

Advanced search

WEBsmith

Phil Hughes

Issue #21, January 1996

Many people (including myself) have found answers to Web-related questions in those and other issues of *LJ*.

In *Linux Journal*'s 21-issue history we have already had two issues that focused on the World Wide Web. Many people (including myself) have found answers to Web-related questions in those and other issues of *LJ*.

Early in 1995, SSC, the company that publishes *Linux Journal*, decided to establish a Web site. We decided, of course, to base it on a Linux system. We cautiously promoted it while waiting for some problems to appear. We watched page hits grow from less than 10,000 per day to over 35,000 per day. Performance continues to be excellent as we happily provide articles from back issues of *LJ*, links to advertisers, SSC's own product catalog and more. (If you have Web access and haven't taken a look. you should. The URL is www.ssc.com)

Back in June, I was reading one of the Web-related articles in *LJ* and realized that the growing hoards of Web developers needed the same sort of concise technical information that *Linux Journal* is offering the Linux community. After talking to Belinda Frazier, Associate Publisher of *Linux Journal*, about the idea and seeing that we were in agreement, we started doing some research. Our conclusion was that while there were lots of Web and Internet magazines, most primarily addressed the “consumer”, i.e., the Web Surfer.

With virtually exponential growth in the Web community and substantial drops in the cost of providing a Web presence it was clear that the development end needed to be addressed. And the amazing speed at which changes are appearing dictated that a magazine was a better approach than a book or series of books.

At this point, Belinda paid the price of agreeing with me. She got saddled with the job of Publisher of the new magazine. My contribution consisted of sparing her the same mistakes we made starting *Linux Journal*. Happily, I can say that we fewer mistakes have been made in this effort than with *LJ*. After we start 50 or more magazines we may have it all down pat.

One of the best decisions we made was to introduce **WEBsmith** as a supplement to *Linux Journal*. After all, the idea came from articles in *LJ*, much of the Linux community has been involved in the Internet, and the Web and Linux make a first-class Web server.

A recent survey showed that 9% of all Web servers are Linux-based. That makes it th second most popular Unix-like platform (behind Sun). While *WEBsmith* is not intended to be another Linux magazine, my personal hope is that we can show the growing Web community that we have a pretty handy operating system here for doing Web work. If you are interested in the Web—either as a developer or a manager who needs to build a Web presence for your company—I encourage you to subscribe to *WEBsmith*. Expect the same high-quality technical articles you have come to expect in *Linux Journal*, addressing your concerns and answering your questions.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #21, January 1996

Readers sound off.

Did you folks get my e-mail from out Humanitarian Support Operations Conference? The Center for Excellence in Disaster Management and Humanitarian Assistance's Humanitarian Support Operations Conference was held recently at the Ilikai Hotel in Waikiki. The event drew over 170 people from 23 Asian-Pacific countries including Thailand, China, South Korea, New Zealand, Papua New Guinea, Fiji, Mongolia, Australia, the Philippines, Indonesia and many others.

Supporting the conference was a pair of Linux machines providing Internet access (WWW and Mail) running NCSA's Mosaic and the COE's Conference Home page. Attendee photos were digitized and put online using an Apple QuickTake camera. Events included a working session and introduction to Linux, which was a first experience for many of these countries.

Future COE plans include travelling to these countries with Linux laptops, as we've chosen to use the OS as our Internet connectivity platform used with developing nations. In short, choosing Linux gives us a cost-free method of connecting dozens of dissimilar host sites with a similar operating environment. We'll begin building custom database applications, namely a front end for our Oracle SQL server (Sparc 20 based) for Linux by the first week of October.

We're very excited about Linux, and everyone at the conference was equally enthusiastic about seeing the “Linux Work-Servers” and the power they give to otherwise blah-entrenched x86s. One of the delegates from India asked me if his 486-100 would be suited for such a project...I had to laugh when I told him that the machine he was using was a 486-33! (We had 20 inch monitors on the desk and the boxes underneath...so it looked like we had some real powerstations going!)

If you'd like additional information about what our future plans for Linux hold, please don't hesitate to ask.

Rob!

LJ Founds a University (Not!)

In case you hadn't gotten this, I enclose the following reply sent to Keith Briggs. In his letter, Mr. Briggs called to my attention my apparent invention of a university:

Dear Mr Wilder:

I quote from ***Linux Journal*** #17, page 22: "...comes from the Australian Technical University in Melbourne,..." I am sorry to have to inform you that there is no such place in Melbourne. There is not even a place with a similar name! (The Australian National University comes closest, but it is in Canberra).

Keith Briggs

Dear Keith Briggs,

I don't know how I came up with the "Australian Technical University in Melbourne"; every reference I can locate in the materials I prepared the review from points to the University of Technology, Sydney. My apologies to all.

Dan Wilder dan@gasboy.com

Perl Errata Sheet

I read your review of my book (*Teach Yourself Perl in 21 Days*, reviewed by David Flood on page 15 —ED) in the November 1995 *Linux Journal*. In the review, David noted that he was unable to get an errata sheet from CompuServe. In case you have not yet managed to pick up an errata list anywhere, I have taken the liberty of enclosing one. This contains every error I know of.

Thank you for taking the time to review my book. If you know of anyone else who wants an errata sheet, let me know and I will pass this on to them.

I am glad that you have found my book useful, and of course encourage you to tell everyone you know to buy it.

Thanks,

—Dave Tillauthor of *Teach Yourself Perl in 21 Days* davet@klg.com

***LJ* Responds:**

The entire errata sheet may be found at the *LJ* Home Page on the WWW. Just browse the contents for Issue #19, November 1995, and click on the title of David Flood's review of Till's book to see the complete list.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

From the Editor

Michael K. Johnson

Issue #21, January 1996

In a rare opinion column, our editor turns Socrates and asks: “What is success?”

As Linux approaches the fifth anniversary of its conception (in June), it is perhaps worthwhile to ask if it has been a success so far. It has certainly been far more successful than anyone originally thought it could be, as support for new hardware has increased tremendously and users all over the world number at least in the hundreds of thousands, and more likely in the millions. Is that success?

For those of us who use it every day to take care of all of our computing needs, Linux is a success regardless of the numbers of people and organizations who use it. Without even a magazine for its users, it would be a success. Without commercial applications, it would still be a success. Linux has nothing to prove.

But that doesn't mean that there is nothing to *im*prove.

Linux is a success with technical people, and has been for a long time. Linux comes with a huge, well-understood tool box of programs for data manipulation and services. And for basic, well-understood services, the Linux distributions provide (more or less) out-of-the-box solutions. FTP services, WWW services, NFS file services, LPD print services, SMB/Lan Manager file and print services, and more all work out-of-the-box, or with a little configuration. In part because of this, Linux is seeing growing personal, corporate, educational, and governmental use.

So what's missing? Fairly obviously, as Linus Torvalds himself points out, a wide choice of desktop applications. But that's being worked on (native applications, Wine, DOSEMU, Executor) and nothing I could say would speed up any of those projects. Instead, I'd like to present one particular challenge for growth: the market for pre-configured (“works out-of-the-box”) software that fills the needs of particular niche markets. If this challenge isn't met, Linux will still be a

success; Linux use won't shrink. This is just an area in which Linux has the potential to be very useful, but where important pieces are still missing.

I'll use the example of Point Of Sale (POS) systems, since I know a little bit about them. It is very definitely possible for a technically competent person to use a Linux system to create a POS system. The goal is essentially to piece together a database with a terminal or network of terminals in order to quickly look up the prices of individual items and compute the total cost of a sale, as well as manage inventory and do financial transactions.

The standard Linux techie (call him Jon Hacker) answer runs something like this: “Oh, that's easy. Just build a database (flat text, DBM, Postgres 95, or one of the commercial databases for Linux) and write a program (Tcl/Tk for X, curses for text terminals) for a user interface. I could do that in a week. Then add things like credit card validation on-line, inventory control, etc. That could take, um, a while longer.”

Jo Store Owner doesn't have a week. She isn't a guru, and she doesn't have a guru to write the system for her, either. And she's not going to hire Jon Hacker to write a POS system based on Linux, since she can buy a system that does meet her needs—though perhaps not as well as a customized system—which runs on DOS or Windows or Mac, and it will cost her less to get it up and running. It may not be as well customized for her business, and it may not even be flexible enough to customize, but it will work well enough for her, and to instead hire Jon Hacker would invoke the law of diminishing returns—it simply wouldn't be profitable.

However, if a POS package (free or commercial; it doesn't much matter) were available for Linux, and came configured intelligently, but used Linux tools to do the job and was therefore easily customized, it would be an attractive option. Joe might even hire Jon to customize it for him.

My point isn't really POS systems; there is already at least one complete POS system based on Linux. However, there are lots of niches like this that Linux is a great technology base for, but which don't have off-the-shelf solutions based on Linux yet, even though more free and commercial tools are available all the time. (Read comp.os.linux.announce and *LJ*'s own **New Products** if you need convincing.) Being able to start doing *something* after running a simple installation (like **a:setup** under DOS and Windows) is the basis of meeting this challenge.

As I see it, this challenge **is** being met to some degree, but sporadically and piecemeal. My goal is merely to help popularize the idea of making Linux a useful business solution, and encourage Jon Hacker to search for and support

niches in a way that Jo Store Owner can understand and trust. I'm not suggesting this to help Linux take over the world, but rather because I think that the technology available for Linux has lots of price/benefit potential for Jo and employment benefit for Jon, and because I think that the price pressure that Linux's low cost can provide will invigorate niche markets.

Perspective

There was a time when Linux existed, but there was no such thing as a distribution. You had to put a Linux system together from scratch—a few floppy images, including kermit for file transfer, or tar to pull more files off floppies, or maybe mtools to read DOS-format floppies. Individual binaries were available from tsx-11.mit.edu and ftp.funet.fi. Distributions weren't *necessary*; it really was possible to build your own. I've done it several times. I would even consider doing it again—for fun. But when I want it done right, I get one of the Linux distributions and install it in a matter of minutes, or at most hours, most of which is consumed by the computer quietly pulling files off a CD-ROM without my assistance.

What I'm suggesting is very much like these distributions: the basic problem already solved, ready for site-based customization, provided in a convenient format. If you think that is a simplistic view of the need, remember that Slackware was created by one person who customized and bug-fixed SLS for his friends and college professors. Although it evolved from there, and doesn't meet everyone's Linux needs, Slackware was useful from the start.

Many of the advertisements in *Linux Journal* are for CD-ROMs with new versions of Linux and Linux tools. That is important; an easily-available supply of new tools has helped Linux spread even faster than it could over the Internet alone. However, based on my belief that Linux is growing and evolving, I suggest that in five more years, we will see more and more advertisements touting Linux-based products intended to solve a business problem, rather than impress geeks like me.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

System Administation: Maximizing System Security, Part 1

Æleen Frisch

Issue #21, January 1996

A lot of UNIX security is based on passwords, and in this first part of a two-part article, Æleen helps explain many of thei issues involved in setting up and maintaining passwords on Linux systems. Next month's installment will cover other system security issues.

One of the most hackneyed cliches in all of UNIX culture is that UNIX security is a contradiction in terms. While things aren't quite as hopeless as this cynical view, it is important to realize that a secure system is something you create, not something you get automatically when you install any current Linux distribution (or any other UNIX operating system for that matter).

This article provides an overview of UNIX security issues, and discusses the resources and tools available to Linux system administrators or anyone responsible for administering a Linux system—which are not necessarily synonymous. It considers what the most important issues are and what exists to defend the system. And since many of the most egregious “UNIX” security problems are actually vulnerabilities in TCP/IP networking and its component protocols, we naturally consider network security issues, as well as those relevant to an isolated computer system.

What is Security?

General discussions of computer security traditionally focus on the types of losses that can result from inadequate security measures:

- Loss of equipment. The first or last threat to any computer system (depending on your point of view) is the loss of the computer itself. This can result from a variety of causes: theft, fire, water, earthquakes and other natural disasters, vandalism, and accidents (e.g., a user spilling coffee on it).

- Loss of data. This type of loss can also occur in a variety of ways: data could be obtained by someone who should not have it (for example, a competitor), files could be accidentally or deliberately damaged or destroyed, or information that should have remained private could become publically accessible or broadcast.
- Loss of use. A third type of loss can occur when neither the equipment nor its data is damaged, destroyed or removed, but the system is nevertheless unable to perform some or all of its normal functions. For example, an extended power outage could cause such a loss of use; the 1988 Internet worm incident is an example of software rendering a computer unusable.

Depending on your situation, some of these threats are obviously more potentially hazardous to you than others.

Effective thinking about security begins by considering potential losses rather than potential threats, because doing so allows you to place the threats in the context of your system and thereby make appropriate choices about how to prevent and address them. For example, every system has the potential of being broken into by an unauthorized person. However, the specific nature of that threat changes depending on the sort of loss that would be its most serious consequence—as do the corresponding measures to prevent the loss.

A successful intruder always has the potential to alter or destroy any file on the system, so every system needs to guard against and have a plan for recovering from that eventuality. In addition, for a system containing sensitive or proprietary data (customer credit card numbers, source codes for software products under development, and so on) one might need to consider ways of securing such data even from the root account. On the other hand, if loss of use is the primary loss against which a system needs to be protected, then devising ways of quickly identifying and neutralizing such an attack is much more important than providing extra security for any of the data on the system.

As these scenarios suggest, security involves more than just prevention against attacks. Equally important components of computer security are the recovery plans which specify what to do when something goes wrong. Computer security is not something you think about once in a while, but rather something that is an integral part of your thinking and actions in every administrative activity you perform. It includes the following concerns (not all of which will necessarily apply to any specific system):

- Physical system access
- Theft prevention—locks and so on
- Prevention of physical and electronic vandalism

- Ensuring continuous power via an uninterruptible power supply (UPS) unit
- Fire control systems, surge suppressors, and other devices to prevent damage from the external environment
- User authentication: passwords and other mechanisms
- Modem access (dialin and dialout)
- File ownership and protection
- Encryption of very sensitive or private data
- Network access policies and network software configuration
- NFS configuration
- Procedures and policies related to building, testing, installing and using public domain software
- Backup procedures
- Secure storage of backup media (including offsite copies)
- Storage of original operating system media
- Disaster recovery plans
- User training for good security practices

A thorough discussion of all of these topics would consume several entire issues of *Linux Journal*, so we focus on operating system-level protections and solutions useful for Linux systems, in terms of both "standard" features and useful additional packages. Security facilities offered by the various Linux distributions vary considerably, but no current distribution includes everything that a prudent system administrator would want to have and use.

Security Resources

Package	ftp Location
COPS	ftp.cert.org:/pub/tools/cops
Courtney	ftp.best.com:/pub/lat
Crack	ftp.cert.org:/pub/tools/crack
Gabriel	ftp.best.com:/pub/lat
Merlin	ciac.llnl.gov:/pub/ciac/sectools/unix/merlin
Netscape	ftp.netscape.com:/netscape/unix
npasswd	ee.utah.edu:/admin/passwd/npasswd
passwd+	ee.utah.edu:/admin/passwd/passwd+
Perl	prep.ai.mit.edu:/pub/gnu
Satan	ftp.win.tue.nl:/pub/security
shadow	sunsite.unc.edu:/pub/Linux/system/Admin
sudo	sunsite.unc.edu:/pub/Linux/system/Admin
swatch	sierra.stanford.edu:/pub/sources
TCP Wrappers	ftp.win.tue.nl:/pub/security/tcp_wrapper
Tripwire	coast.cs.purdue.edu:/pub/COAST/Tripwire

Passwords and User Authentication

Passwords are the primary way of securing user accounts on Linux systems. However, the protection offered by passwords is only as good as the passwords themselves. If a hacker decides to attack the accounts on your system, bad passwords are almost as bad as no passwords at all.

There are several things you can do to ensure that the password facility is providing the best protection it is capable of:

- Make sure all active accounts have passwords and that system accounts not intended for user logins (e.g. bin) are disabled (do this by placing an asterisk in the password field for that account).
- Secure the encoded versions of the system's passwords by using a shadow password file.
- Educate users about keeping passwords secret, selecting hard-to-crack passwords, changing passwords as necessary, using different passwords at different sites, and similar security practices. Institute password aging and/or new password obscurity checking if appropriate.

The first item is self-explanatory; we look at the others in detail.

Shadow Password Files

Shadow password files are designed to correct the security hole resulting from the normal password file being world-readable. Everyone needs to be able to view the contents of /etc/passwd so that things like file ownership displays properly (UIDs are translated into usernames). However, since the file is readable, anyone can make a copy of it. This means someone with legitimate or illegitimate access to an ordinary user account can copy it and attempt to crack the passwords of more powerful accounts at his leisure.

A shadow password file facility removes the encoded passwords from the normal password file and places them in another file, conventionally /etc/shadow, which can be read only by root. The shadow package provides shadow password file capabilities for a variety of UNIX systems including Linux. It is included in some Linux distributions by default. It includes replacements for the login, passwd, and su commands as well as many utilities for creating and manipulating the shadow password file and account entries within it.

Building the shadow package is quite straightforward. If you've retrieved a version that has been ported to Linux, you'll generally only have to modify the config.h file. I recommend the following settings (culled from various points within that file):

```
/* Use shadow password file.      */
#define SHADOWPWD
/* Use up to 16 char. passwords.  */
#define DOUBLESIZE
/* Enable password aging checks.  */
#define AGING
/* Log events to syslog facility. */
#define USE_SYSLOG
/* Support for remote logins.     */
#define RLOGIN
#define UT_HOST
```

```
/* Data file for most recent login time records */
#define LASTFILE "/var/adm/lastlog"
```

Once the package is built and installed, the **pwconv** command may be used to create an initial /etc/shadow file. It creates the files /etc/npasswd and /etc/nshadow. The former is an altered version of the original password file in which the password field in each entry has been replaced by an **x**; the latter is the corresponding shadow password file. In order to activate them, you must rename them by hand:

```
# cd /etc
# mv passwd passwd.prev
# cp npasswd passwd
# cp nshadow shadow
```

Password Aging

Users don't like to change their passwords, and left to their own devices they will literally never do so. The shadow package includes an optional password aging facility which enables a system administrator to specify how often users must change their passwords. Whether using these features is necessary or not depends on the needs of your site.

Entries in /etc/shadow have the following format:

```
username:password:change_date:min_change:\
 max_change:warn:inactive:expire:
```

where the first two fields are the username and encoded password for the account. The other fields relate to account expiration and password aging. [Note that in the shadow file, it cannot be continued across two lines, as we have done here to make it fit in the magazine—ED]

change_date encodes the date of the most recent password change. *min_change* and *max_change* indicate the minimum and maximum number of days between password changes, and ***warn*** indicates the number of days before a password expires that the user is warned of this fact. ***inactive*** specifies how many days after its password has expired that an account is automatically disabled, and ***expire*** encodes the date upon which the account itself will expire and be disabled.

Here is a sample entry from /etc/shadow:

```
chavez:XdleIqAert:9422:7:180:5:21::
```

User chavez can keep the same password for at most 180 days, and she will be warned 5 days before her password expires. When she does change her password, she must keep the new one for at least 7 days. If she doesn't use her

account for 21 days, it will be automatically disabled. No expiration date is set for user chavez's account.

Once /etc/shadow is installed, it can be edited directly. However, the shadow package also provides tools for manipulating entries within it. Its version of the passwd command updates passwords within the shadow password file, and the command also has additional options for modifying the other password settings. For example, the following command changes the minimum password lifetime to 2 days, the maximum password lifetime to 1 year, the warning period to 3 days, and the inactive period to two months for user chavez:

 # passwd -n 2 -x 365 -w 3 -i 60 chavez

If you wanted to remove all aging controls from an account, use this combination of options:

 # passwd -n 0 -x 99999 -i -1 angela

These are the default values (along with a warning period of 14 days, which is irrelevant when passwords essentially never expire).

Account expiration dates are set with the usermod command. The following command sets the account expiration date to January 1, 1999 for user chavez:

 # usermod -e 1/1/1999 chavez

Note that the useradd and usermod commands may also be used to create user accounts and specify or alter these and other account settings.

passwd's **-l** and **-u**options may be used to manually lock (i.e., prevent logins) and unlock an account respectively:

 # passwd -l badboy

Finally, the **chage -d** command may be used to force a user to change his password at his next login; this option sets the date of the last password change field. It may be used to force a password change at a user's next login, provided that a maximum password lifetime is also set. Here is a simple script which accomplishes this:

```
#!/bin/csh
# force_change username -- run as root
chage -l $1 >& /dev/null
if ($status == 1) then
   echo force_change: invalid user "\$1\"
   exit 1
endif
set max=`grep ^$1\: /etc/shadow | \
```

```
  awk -F: '{print $5}'`
chage -d `date +%D` $1
set today=`grep ^$1\: /etc/shadow | \
  awk -F: '{print $3}'`
set yest=`expr $today - 1`
if ($max >= $yest) set max=`expr $yest - 1`
set date=`expr $yest - $max`
chage -M $max -I 2 -W 7 -d $date $1
```

The script extracts the current maximum password lifetime setting, sets the password change date to today, and then extracts the equivalent integer value (the number of days since 1/1/1970). It then sets the password change date to yesterday, reducing the maximum lifetime if necessary so that a password expiration is possible on that date. It also sets the inactivity period to 2 days and the warning period to 1 week.

shadow's Configuration File

Default settings for password aging settings are defined in the configuration file for the shadow package, usually defined as /etc/login.defs. This file contains a variety of entries which control various aspects of how the package functions, all well-documented in its comment lines, and you should examine this file carefully and select settings which make sense for your system.

Here are some of the most important entries from this file, along with my suggestions for their values:

```
# Enable dialup passwords.
DIALUPS_CHECK_ENAB      yes

# Track login failures in /var/adm/faillog.
FAILLOG_ENAB            yes
LOG_UNKFAIL_ENAB        yes

# Track login times in /var/adm/lastlog.
LASTLOG_ENAB            yes

# Enable password obscurity checking.
OBSCURE_CHECKS_ENAB     yes

# Enable login time restrictions (/etc/porttime).
PORTTIME_CHECKS_ENAB    yes

# Specify the su log file.
SULOG_FILE      /var/adm/sulog

# Enable use of /etc/nologin file to prevent
# non-root logins. The contents of the file
# is displayed as an error message.
NOLOGINS_FILE   /etc/nologin

# Password aging settings.
PASS_MAX_DAYS   186
PASS_MIN_DAYS   7
PASS_WARN_AGE   14

# Set minimum password length.
PASS_MIN_LEN    12
```

Selecting Good Passwords

Making sure that all accounts have passwords that are changed regularly is only part of what is necessary to get the maximum protection from passwords. Passwords must also remain secret, and they must be hard to guess—for either a program or a human—to be most effective. The first of these can only be ensured by educating users about the importance of passwords to system security. It is possible to have a little more control over the second.

Bad password choices include all correctly-spelled words, proper names, and names or numbers significant to the person choosing the password, as well as simple transformations of any of these items: reversals, simple capitalization changes, rotations, adding a digit at the end, and the like.

Good passwords include a variety of character types—upper and lowercase letters, numbers and symbols, and control characters. Longer passwords are also better than shorter ones. I strongly recommend enabling the shadow package's double-length password capabilities, setting a minimum length of 10 or 12 characters and allowing up to 16.

The shadow package has only minimal capabilities for checking the qualities of the password that users choose. However, there are other packages which provide this function by substituting an alternate version of the passwd command. The npasswd package can check proposed passwords against words in online dictionaries. The passwd+ package checks proposed passwords against one or more dictionaries, and also tests transformations of the proposed passwords according to instructions provided by the configuration file. The file /etc/passwd.test can be customized by the system administrator. Listing 1 gives some sample entries from the file which will give you a sense of passwd+'s capabilities.

```
# sample passwd+ configuration file
# Test                          Error Message
#
(%#a==8)&((%#c==0)|(%#w==0))    Include a capital letter or numeral.
(%#l>5)|(%#c==0)                Must include a nonalphabetic character.
#
"%*p"=~"^%*u$"                  Can't use username as password.
"%*p"=~"^%-*u$"                 Can't use reversed username as password.
"%*p"=~"^%*f%*f$"               Can't use doubled first name as password.
#
{tr A-Z a-z < /usr/dict/words} =~ "%*p"   Password found in dictionary.
```

Each entry lists a type of unacceptable password and gives an appropriate error message to be displayed to the user if such a password is proposed. The following symbols are used in the sample rules defining unacceptable passwords (passwd+ offers many more as well):

```
<ul>
<li>%p  proposed password
```

```
<li>%a  alphanumeric character
<li>%l  lowercase letters
<li>%c  capital letters
<li>%w  numerals
<li>%u  username
<li>%f  first name
<li>%-x reversed version of x (e.g. %-p =
reversed proposed password)
<li>%*x lowercased version of x
<li>%#  number of x's in proposed password
(e.g. %#w = # of numerals)
<li>&   logical AND
<li>|   logical OR
<li>==  equals
<li>=~  matches pattern
<li>^   beginning of line
<li>$   end of line
</ul>
```

The first section of the sample file checks structure of the proposed password. The first entry rejects 8-character all-lowercase passwords, and the second entry says that passwords containing 6 or more lowercase letters must also contain a capital letter.

The second section of the file tests the password against items from the user's password file entry as well as some transformations of them. The third section performs a case-insensitive comparison of the password with the words in the system's dictionary file.

By default, passwd+ is designed to be used as a stand-alone replacement for the normal passwd command, and it is not aware of the shadow password file. However, it is not very difficult to modify it for use on systems with shadow passwords; one method involves modifying the obscure routine in the shadow package to call the verify routine in the passwd+ package.

You can also use the Crack facility to check the quality of users' existing passwords. (Note that it is unethical to run Crack without permission on the password file from systems where you are not the system administrator!) Crack is easy to build and use, and it includes a shadmrg utility (in its Scripts subdirectory) which can reconstruct a traditional /etc/passwd-style file on a system using the shadow package. For example:

```
# cd /usr/src/Crack*
# Scripts/shadmrg > passwd.test
# Crack passwd.test
```

If you choose to use Crack, it is extremely important to ensure that the program and all of the data and results files that it creates are protected against all non-root access.

Secondary Authentication

In some circumstances, you may want to use some means of determining that a user is who she says she is in addition to standard passwords. To address

such a need, the shadow package also supports entries like the following in /etc/shadow:

```
chavez:XdleIqAert;@/sbin/extra:9422:7:180:5:21::
harvey:<\@>/sbin/extra:9233:0:99999:0:-1::
```

When user chavez logs in, she is prompted for her password, and then the program /sbin/extra runs as a secondary authentication program. This program, supplied by the system administrator, can perform whatever sort of additional authentication is desired, returning a value of 0 or 1 depending on whether the user has passed or failed. The second entry indicates that user harvey runs /sbin/extra as his only authentication method. In the shadow file syntax, the @ sign introduces the alternate or additional authentication program, and a semicolon is used to separate it from the encoded password.

Dialup Passwords

The final feature of the shadow package we consider is its dialup password facility, which allows you to require an additional password from users who connect to the system via a dialup line. When this feature is enabled, two additional configuration files are used, /etc/dialups and /etc/d_passwd. /etc/dialups contains a list of special filenames for serial lines that are to be protected with an additional password whenever someone dials into the system (one per line), and /etc/d_passwd holds the encoded dialup passwords.

Dialup passwords are assigned on a shell-by-shell basis, and the dpasswd command is used to create and change them. For example, the following command will allow you to change the current dialup password for the shell /bin/bash:

 # dpasswd /bin/bash

Dialup passwords will not be required from users using shells that are not listed in /etc/d_passwd.

sudo: Selective Access to root

One of the biggest weaknesses with UNIX security in general is its all-or-nothing approach to system privilege: root is powerful, so it is only prudent to limit access to the root account as much as possible. The sudo facility enables non-root users to run specified commands as root without having to know the root password, allowing a system administrator to provide users with just the level of access they actually need.

A user uses the facility by prefacing the command he wants to run with the sudo command:

$ sudo mount hamlet:/data /mnt

:sudo will require the user to enter his own password before completing the command. Thus, in this case, using sudo allows this user to use the mount command without knowing the root password.

Access to sudo is controlled by its configuration file, /etc/sudoers. This file specifies which users can use sudo along with the commands they are allowed to execute. Here is a small extract from such a file:

```
chavez  ALL=ALL
harvey  ALL=/bin/mount,/bin/umount
nelson  ALL=!/sbin/shutdown
```

Usernames are the first field in each entry, followed by one or more access description strings which have the general form: ***host(s)=command(s)***. Based upon these entries, user chavez can use sudo to run any command on any system, user harvey can mount and unmount disks, and user nelson can run any command except shutdown. Note that these examples represent only the simplest form of this file; its actual syntax is very flexible and powerful, allowing you to define named groups of hosts and/or commands and thereby specify exact access for each user-host-command combination in as much detail as necessary.

For More Information About System Security

Books

Practical Internet and UNIX Security, (2nd edition of *Practical UNIX Security*), Simson Garfinkel and Gene Spafford (O'Reilly & Associates, late 1995 or early 1996). An excellent book-length treatment of system security and the associated system administrative concerns and tasks.

Essential System Administration, 2nd edition, Æleen Frisch (O'Reilly & Associates, 1995). A substantial chapter is devoted to system security, as are numerous additional sections throughout the book.

Firewalls and Internet Security, William R. Cheswick and Steven M. Bellovin, (Addison-Wesley, 1994); *Building Internet Firewalls*, D. Brent Chapman and Elizabeth D. Zwicky (O'Reilly & Associates, 1995). Two essential books for anyone considering setting up a firewall system.

Security Alerts Mailing Lists

The Computer Emergency Response Team (CERT) manages the primary UNIX-related security-alert system. Send mail to cert-advisory-request@cert.org to be

added to the mailing list. Past advisories and updates are available via anonymous ftp at info.cert.org:/pub/cert_advisories.

There is also a Linux-specific security advisory mailing list. Send mail to majordomo@linux.nrao.edu with **subscribe linux-alert** in the body of the message to be added to the list. You may also want to subscribe to the linux-security mailing list, which is a moderated discussion list for Linux-related security topics. (To subscribe, include **subscribe linux-security** in the body of a message to the same email address.) The archives for these mailing lists are available via anonymous ftp at linux.nrao.edu:/pub/linux/security/list-archive.

Æleen Frisch (aefrisch@lorentzian.com) manages a very heterogeneous network of Linux and other UNIX systems and PCs. Having recently finished second editions of two books, she looks forward to pursuing her true calling: pulling the string for her cats, Daphne and Sarah.

Archive Index Issue Table of Contents

Advanced search

New Products

LJ Staff

Issue #21, January 1996

SCSIserver SC16, Freedom Desktop for Linux and more.

SCSIserver SC16

ABLE Communications Inc. has announced SC16, a SCSI-based terminal server for Unix and Linux based platforms. The SCSIserver SC16 features full modem control and baud rates up to 115kbs for all channels. It comes with 16 ports in a single enclosure, including a universal power supply. The SC16 is ideally suited for multi-user applications or where system operators are controlling large banks of modems. Seven SC16s can be used together for up to 112 ports on a single system. Price: $1595.

Contact: ABLE Communications, 2823 McGaw, Irvine, CA 92714 Phone: 714-553-8825. Fax: 714-553-1320.

Freedom Desktop for Linux

Freedom Software, in partnership with Thinking Objects Software GmbH, has announced that the Freedom Desktop for Motif is now available for Linux. Freedom Desktop for Motif is an easy-to-use yet powerful desktop manager/ GUI integrated with the Unix environment. It combines ease of use and advanced features to help users interact with Unix quickly and efficiently. It also runs transparently in a variety of Unix environments, from desktop computers to enterprise workstations. A free evaluation copy may be retrieved from fsw.com in /pub.

Contact: Freedom Software, 9F Oliver Court, Pittsburgh, PA 15239. Phone: 412-327-4940. Fax: 412-327-6518. E-mail: support@freedom.lm.com or uhl@to.com . URL: www.fsw.com.

FairCom releases Linux Server

FairCom Corporation has released the first commercial version of the FairCom Server and SQL Server for Linux. According to Winston Atkisson, Senior Engineer at FairCom, “FairCom's Linux Server offers true client/server architecture and heterogeneous network support.” The Linux Server is priced from $495 to $2395, depending on the number of users.

Contact: FairCom Corporation, 4006 West Broadway, Columbia, MO 65203. Phone: 800-234-8180. Fax: 314-445-9698.

BOOT ROM for Workstations

If you want to make a “diskless” workstation truly diskless, you can use a BOOT ROM. This read-only memory chip with the boot code burned into it plugs into your Ethernet adapter and loads the kernel of your operating system via the network. BOOT ROM comes with a floppy disk containing all the software necessary to set up your Linux machine as a boot server for a network of diskless Linux workstations. Source code for all the software is available upon request on a second floppy disk. Currently, BOOT ROMs are available for 3c509 (3com) and NE2000 (generic) cards. Price: BOOT ROM, $14.00 + shipping; Source code, $2.00 + shipping.

Contact: bootrom@datawire.com for more information or to order.

Chat Server Available

The Chat Server is a continuous-stream, real-time, multimedia-capable, web-based communication server developed by Magma Communications Ltd. Developed in a Linux environment for Linux-based machines, it works with practically all Web browsers, but to take advantage of Chat Server's continuous-stream capabilities, the Netscape browser is required on the client end. The Chat Server is not a cgi-script but a specialized server designed for a chatting environment. The Linux version of the Chat Server is available now, with ports to BSDI, Sun, NT, and HP-UX in the works.

Further information on the Chat Server can be found at Magma Communications Ltd.'s website at www.magmacom.com/chatserver/index.html.

Volant Corporation Announces *htmlscript*

Volant Corporation has announced the availability of an easy-to-use language for Web Servers called *htmlscript*. Completely browser independent, *htmlscript* allows users to develop interactive web pages in a server-safe environment.

The software is available for most Unix and Unix work-alike systems. Access to reference documentation is available at http://htmlscript.volant.com/. Price: $99.00 for a 500-user license, which includes one year of free updates.

Contact: Volant Corporation, 2629 Ariane Drive, San Diego, CA 92117. Phone: 619-490-2570. E-mail: htsinfo@volant.com.

Archive Index Issue Table of Contents

Advanced search

The chmod Command

Eric Goebelbecker

Issue #21, January 1996

How to use the versatile command chmod.

Do you know how to rename a file you can't read? Better yet, do you know how other users can rename your files? Have you ever ftp'd a program from another host and been unable to run it?

The subject of file permissions, and how to manipulate them with the **chmod** command, is a good place to start learning about these situations.

First, let's create a file and examine its long listing. (In order to fit in the magazine, all the listings in this article are trimmed to fit.)

```
$ touch test_file
$ ls -l test_file
-rw-rw-r--   1 eric     users
```

Since I created this file, it makes sense that the third column shows my user name as the file's owner and that the fourth shows my group. (On some systems, the group name may be the same as the user name.) As you follow along in these examples, you will see your username in place of “eric”.

The leftmost column of the directory listing shows the file's **mode**. Mode is the term used to refer to a file's permissions. ls displays the file's type and mode together as a grouping of ten one-character fields:

Type	Owner	Group	World	-	rwx	rw-	r--

The type field has several valid values. For the sake of this tutorial, we are only concerned with two: empty (**-**) for a regular file, and **d** for directories.

The other three columns cover the three **classes** of access that are stored for each file in a Unix-like file system. Linux (and Unix) evaluates access in terms of user ownership, group ownership and world (or other).

For each of these classes, rights are evaluated in terms of three **operations**: reading (**r**), writing (**w**) and executing (**x**). The permissions above specify "full" access for the owner, reading and writing for group, and only reading for world (an unusual combination used for demonstration). Those permissions specify that

- The owner of the file is allowed to read, write and execute the file.
- Any user who is a member of the group that owns the file is permitted to write to the file.
- Any other user can only read the file.

Changing permissions

If test_file were a very important document that we did not want anyone to be able to modify or delete, we would need to remove write access from group:

```
$ chmod g-w test_file
$ ls -l test_file
-rw-r--r--   1 eric     users
```

We see that the **w** for group is now replaced with a **-**, signifying that write permission is denied to members of the group **users**.

If test_file contained sensitive information that only members of the group **users** should be able to review:

```
$ chmod o-r test_file
$ ls -l test_file
-rw-r-----   1 eric     users
```

Now we see that the last triplet of the mode field, which specifies permissions for world, are all dashes. This means that other users who do not belong to the **users** group have no permissions to do anything with test_file whatsoever.

The command line usage for chmod mode looks like this:

```
chmod [options] new-mode filename
```

The new mode is specified in **octal mode** or **symbolic mode**. We'll cover symbolic mode first. In the first example we used **g-w** to remove write permission for group. As you might be able to guess, **g** stood for group, **-** for remove and **w** represented write permission.

```
$ chmod g+wx test_file
$ ls -l test_file
-rw-rwx---   1 eric     users
```

This operation added permission for group to write and execute.

Let's look at an example of these permissions in action.

```
$ chmod u-rwx test_file
$ ls -l test_file
----rwx---   1 eric     users
$ cat test_file
cat: test_file: Permission denied
$ cat .profile > test_file
bash: test_file: Permission denied
```

We are not able to display the file's contents because we do not have read access to our own file. When we specified **u-rwx** to chmod, we removed all access for the user (the file's owner). We were also denied permission when we attempted to add the contents of another file to it since we removed write access. (I should note that **rm** would still be able to delete this file, although it will normally request confirmation.)

```
$ chmod u+rwx test_file
$ ls -l test_file
-rwxrwx---   1 eric     users
```

When we specify **u+rwx**, all permissions are restored. Removing permissions from a file we own does not affect our ability to restore the permissions, because the mode is **not** stored in the file. It is stored in a structure called an inode entry. Only the owner of the file (and root) may modify this.

Understanding chmod

Let's look at a summary of chmod's options, and then cover each option in depth:

User

> u user (owner)
>
> g group
>
> o other (world)
>
> a all (user, group, and other)

Operation

> + add

> - remove
>
> = set exactly

Mode

> r read
>
> w write
>
> x execute
>
> X conditionally set execute
>
> s Set UID or set GID
>
> t set “sticky” bit

```
$ chmod a+rwx test_file
$ ls -l test_file
-rwxrwxrwx   1 eric     users
```

This demonstrates the fourth possible symbol for user when using symbolic mode. We used **a** to set full permissions for all user classes at once. Let's delete the file and start over in order to demonstrate the difference between the **=** operator and the **+** and **-** operators. (From here on, we'll assume that you know how to get the directory listing, and won't list the ls command.)

```
$ rm test_file
$ touch test_file
-rw-rw-r--   1 eric     users
$ chmod g+x test_file
-rw-rwxr--   1 eric     users
```

This added execute permission for group.

```
$ chmod g=x test_file
-rw---xr--   1 eric     users
```

The **=** operators set group's permissions to execute, and in doing so removed read and write permission. While **+** and **-** set or unset the permissions specified, **=** will set *exactly* the mode specified and remove any others.

Read, write and execute modes are very straightforward when referring to files. Read and write allow a user to examine and modify/delete data from a file, respectively. Execute allows a user to execute a shell script or binary program. If you ftp a program from one host to another and then try to run it without setting execute permission, it will fail, since ftp does not set execute permission.

Directories

For directories, the rules can be a bit more complicated.

Read permission allows a user to examine the contents of a directory.

```
$ mkdir test_dir
$ touch test_dir/foo
$ ls test_dir
foo
$ chmod u-r test_dir
$ ls test_dir
ls: test_dir: Permission denied
```

Write permission allows a user to modify the contents **of the directory**. That means that lack of write permission on a directory does not prevent a user from modifying a file within the directory, if the file's permissions allow it. It **does** prevent the user from renaming, moving, deleting or creating any file in the directory. This is because a directory is a really a file that contains a list of filenames, and so read and write permission control access to that list.

```
$ chmod u=rx test_dir
dr-xrwxr-x   2 eric     users
$ touch test_dir/bar
touch: test_dir/bar: Permission denied
$ mv test_dir/foo ./foo
mv: cannot move `test_dir/foo' to `./foo':
Permission denied
```

This property also works the other way. Since write permission allows the modification of directory entries, a user can move or rename a file **without permission to examine the contents**. This is a very good reason for paying attention to write access for important directories.

To demonstrate:

```
$ ls -l test_dir
-rw-rw-r--   2 eric  users  foo
$ chmod u=rwx test_dir
$ chmod u=rx test_dir/foo
$ cat .bashrc > test_dir/foo
bash: test_dir/foo: Permission denied
$ mv test_dir/foo ./foo
$ ls test_dir
(It's empty)
$ ls foo
foo (It's in our present directory.)
```

Execute permission for directories (also referred to as search permission) is also very important. Execute permission is necessary for **accessing** a directory.

```
$ chmod u=rwx test_dir
$ cp ~/.bashrc test_dir
(any text file will do)
$ chmod u=rw test_dir
$ cd test_dir
bash: test_dir: Permission denied
$ cat test_dir/.bashrc
cat: test_dir/.bashrc: Permission denied
```

This copy of .bashrc does not do us a lot of good. However, setting execute permission for directory and not setting read or write can come in handy.

```
$ chmod u=x test_dir
$ cat test_dir/.bashrc
(we see the contents of the file)
$ ls test_dir
ls: test_dir: Permission denied
```

A directory that has execute permission only can be used to “hide” files. Only users who know the exact file name and path can access them; this includes both data files and programs.

Conditional execute

Let's return to test_file to examine the **X** option.

```
$ chmod u=rw,g=r,o=r test_file
-rw-r--r--   1 eric     users
$ chmod o+X test_file
-rw-r--r--   1 eric     users
$ chmod u+x test_file
-rwxr--r--   1 eric     users
$ chmod o+X test_file
-rwxr--r-x   1 eric     users
```

In the first command, we see that we can set options for more than one class at a time by using a comma to separate the mode specifications. Here, we set the mode so that no user has execute permission. In the second command, we try to set execute permission for other with **X**. This fails, because **X** only works when one of the classes already has execute permissions. When we add execute permissions for owner, **X** sets executable permission for other.

The **s** option sets or removes set UID (SUID) and set GID (SGID) mode. These modes are very important in terms of UNIX/Linux security. When a file has SUID mode set, the process executing it has the effective rights of the file's owner for the duration of the program's execution.

For example, the program **dip** is used to create SLIP network connections. This requires root access, because creating a network interface device requires root access. Instead of forcing users to become root in order to use dip, which would require that the users know the root password, the dip program can belong to root and have the SUID mode set.

```
$ ls -l /usr/sbin/dip
-r-s--x---   1 root     dip
```

The **s** in the spot for user's execute field indicates the SUID mode is set. Another example of a use for the SUID mode is the passwd program, which allows users to modify the passwd (or shadow) file.

For security reasons, the SUID bit can affect only binary programs; it has no effect on shell scripts in Linux.

The SGID mode sets the group instead of the owner, and is set with (for example) **g+s**. It also has another purpose.

When a user creates a new file the group ownership defaults to the user's default group, which is the one listed in the passwd file. Sometimes users belong to more than one group and want to share files. The SGID mode can provide a convenient method for this. If the SGID mode bit is set for a directory, new files created in that directory will belong to that group, regardless of the creator's default group. If you belong to more than one group, try this. (You can check what groups you belong to with the **id** command. The default group is listed first, and you can use the chgrp command to change the group ownership of a file to another group you are a member of.)

```
$ mkdir test_dir
$ chgrp nondefault test_dir
$ chmod g+s test_dir
$ touch test_dir/foo
$ ls -l test_dir/foo
-rw-rw-r--   1 eric     nondefault
```

The SUID and SGID modes can be a security hole. However, when used carefully, they are very valuable tools and actually enhance system security by providing an alternative to distributing important passwords.

Make it simple

Specifying user classes can be used to simplify copying permissions.

```
$ chmod g=u test_file
-rwxrwxr-x   1 eric     users
```

This copied the permissions from user to group. All of the classes can be used on the right side of the **+**, **-** or **=** operators in this way.

```
$ chmod o-u test_file
-rwxrwx---   1 eric     users
```

This cleared all of the permissions that user has from other.

The last mode listed above is the **t** option, known as the "sticky bit". This mode is actually supported on the command line for compatibility purposes with shell scripts from older operating systems. It is not needed for Linux. If an installation guide instructs you to use it, it actually does nothing.

Do your math

File access modes can also be set using octal notation. This syntax is built by adding the mode fields together. For each user class, the fields are calculated this way:

- 4 Read
- 2 Write
- 1 Execute

Full permissions for any class would be **7**, no permissions would be **0**.

```
$ chmod 754 test_file
-rwxr-xr-x   1 eric     users
```

The classes are passed to chmod in the same order ls displays them. The mode we set is broken down this way:

```
        Owner = 4 + 2 + 1 = 7
        Group = 4 + 1     = 5
        World = 4         = 4
```

Octal mode is convenient because other utilities, such as find, expect modes to be expressed this way.

In octal mode, SUID and SGID are set by specifying them in another column *before* the user mode. For SUID use 4, for SGID use 2, and use 6 for both:

```
$ chmod 4755 test_file
-rwsr-xr-x   1 eric     users
```

Power chmod

Chmod also provides a few command line options to simplify administrative tasks. For changing file permissions in directory trees use **-R**.

```
$ chmod -R g-w test_dir
```

This would remove write permission for group for all of the files in and below test_dir.

In order to control the output of messages from chmod use **-c**, **-v** and **-f**:

```
$ chmod -v 700 test_file
mode of test_file changed to 0700 (rwx------)
```

This option caused chmod to display how the permissions of test_file were set. The **-c** option causes chmod to display messages only when files are changed, and the **-f** option suppresses messages about files that can't be changed.

Chmod also provides a **--version** option to display the version and **--help** to see a short help message.

Summary

File permissions are an integral part of Linux. The same concepts also apply to other operating system objects such as semaphores, shared memory, and NIS+. This tutorial provides you with some of the basic knowledge necessary to protect your data and have more fun with your Linux system, and provides you with mental building blocks for learning more about Linux.

Eric Goebelbecker (eric@cnct.com) is a systems analyst for Reuters America, Inc. He supports clients (mostly financial institutions) who use market data retrieval and manipulation APIs in trading rooms and back office operations. In his spare time (about 15 minutes a week...), he reads about philosophy and hacks around with Linux.

Archive Index Issue Table of Contents

Advanced search

Linux on Alpha AXP—Milo, The Mini-loader

David Rusling

Issue #21, January 1996

Linux on Alpha AXP—Milo, The Mini-loader

Late in 1994, I was in the US visiting my home group in Hudson, Massachusetts, where the Alpha AXP processors are built. On a free morning, I followed up a rumour that I had heard a few weeks before. The rumour was that Jim Paradis was porting Linux to Alpha. At that time I did not know anything about Linux other than that it was a freeware version of Unix developed by a student in Finland. When I caught up with Jim, I was in for a surprise. Well, two surprises actually: Linux—running on an Alpha laptop. We chatted for a while and I soon became infected (if that's the right phrase) with Jim's enthusiasm.

One subject that we discussed was Linux/Alpha's need for a small loader. On an Intel PC system, the firmware that initialises the system when it is powered on is known as BIOS. There are several very well known providers of BIOS code, and PCs are built to conform to a very rigid set of rules, which means that PCs are very similar to each other in hardware terms. The equivalent software in an Alpha-based system from Digital (and even in a VAX-based system) is the **console**, and within Digital it is known as the **SRM console**, since its interface is described in the System Reference Manual.

So, just why did Linux need the SRM console? Firstly, Linux needs to be loaded from some media, and the SRM contains device drivers to do just that. Secondly, Linux needed the Digital Unix PALcode. PALcode can be thought of as a tiny software layer that tailors the chip to a particular operating system. It runs in a special mode (PALmode) and has certain restrictions, but it uses the standard Alpha instruction set. In this way, the Alpha chip can run such diverse operating systems as Windows NT, Open VMS, Digital Unix and, of course, Linux. Finally, Jim was using the SRM call backs in his prototype device drivers. From Linux's point of view, though, the SRM Console does too much. It contains call back procedures that allow the running operating system to write messages to the console or to write environment variables and so on. Linux

makes no use of these functions; in fact, the only part of the SRM console needed, once it loads Linux, is the PALcode.

I volunteered to write a loader that would be small and would do only those things that Linux needed. Like all good projects, my one-man effort had some straightforward project goals. First and foremost, the software would be under free licence, built and freely distributed as part of standard Linux distributions. Secondly, Linux drivers should be able to be used within Milo without modification or even re-compilation. Thirdly, it should maximize the amount of memory available to Linux. Little did I realise quite what I was letting myself in for.

Milo

Milo contains the following functional pieces:

- PALcode
- Device drivers
- Linux Kernel and pseudo-Kernel
- Linux Kernel interface code
- User interface code

Finding Digital Unix PALcode was no problem. Back in 1992 when Digital announced the Alpha processor, it also announced that it was moving into the merchant chip market. I joined a small group in the UK to provide engineering effort in Europe to further these aims. We are a small offshoot of the main group, which is based in the silicon factory in Hudson, Mass. We build evaluation boards for the Alpha processors and PCI peripheral chips. These systems included a very low level Evaluation Board Debug Monitor that uses Digital Unix PALcode. The sources for the Evaluation Board Debug Monitor and the PALcode are under free licence.

Although this PALcode is fully compliant with the interface described in the Alpha Architecture Manual, there are some differences between it and the SRM console's PALcode. One of the differences between Alpha based systems is the way interrupts are handled outside of the processor itself. There are a limited number of interrupt signals into the CPU itself, and the number varies from CPU to CPU, but there are typically three: timer, I/O and non-maskable interrupt.

The way in which real device interrupts are mapped onto CPU interrupts is system-specific. Most current Alpha systems include an ISA bus, whose interrupts are routed through a pair of 8259s in the same way as on x86 PCs. The SRM console PALcode handles these differences and interprets the

interrupt, passing it to the OS's interrupt handling code as an “SCB offset” (described by Jim Paradis in last month's Kernel Korner). The PALcode used in Milo does not do this interpretation, so the OS's interrupt handling code must do it instead. Particularly with PCI devices, there must be code in the PCI BIOS code and within the interrupt handler that understands how interrupts are routed in the system. One side effect of this is that when Linux has been loaded by Milo, the interrupt handler can handle more than one device's interrupt each time it is called.

One interesting and useful feature of the example PALcode I adopted from our Alpha evaluation boards is that it allows the Evaluation Board Debug Monitor to run in 1-to-1 physical addressing mode. Bit 0 of the virtual page table base register turns this feature on and off. When translation buffer misses occur, the PALcode builds a new page table entry and inserts it into the cache. Pretty much the first thing that Milo does when it is loaded is to swap to this PALcode in physical address mode. The last thing that the Mini loader does is to swap to this PALcode again, this time passing final control to the Linux kernel.

Milo must turn virtual memory mapping on as it passes control to the kernel, because Linux expects that a control structure called the Hardware Restart Parameter Block (HWRPB) is at the right virtual address. Amongst other things, this describes the type of system and how much memory is free, together with where the memory is. As Linux was first loaded via the SRM console, it naturally used the interface provided by the SRM, which was the HWRPB, as described in the Alpha Architecture Manual. I could see no reason to change this interface: there are enough interfaces in the world, so why invent yet another one?

In order for Milo to set up the memory mapping correctly, it must itself have a good idea of what memory is available and what it is being used for. It finds the amount of memory available because after the PAL reset code has been executed, the size of memory is put into the impure area, a data structure shared between the PALcode and the console or Evaluation Board Debug Monitor. Milo keeps a memory map describing what each page in the system is being used for. While device drivers are running, they allocate temporary memory and use it. Just before control is passed to Linux, Milo must build a correct memory cluster description in the HWRPB, and the memory map is used to do this. Pages are marked as “free”, “allocated” or “temporarily allocated” in the memory map. When Milo builds the memory cluster descriptions in the HWRPB, it treats all temporarily allocated memory as free, since they will be free once Linux starts to run. In this way, the only memory that is marked as allocated is the memory containing the PALcode (8 pages), the memory for the HWRPB (1 page) and the memory for the level 2 and level 3 page tables (2 pages): 11 pages in total. I think I've succeeded in my goal of maximizing the amount of memory available to Linux.

Device drivers

One of the really excellent things about Linux is the number of device drivers that have been developed for it. It seems as if any commercially-available card or chip set has a driver already written for it. It therefore seemed vital that Milo should be able to make use of those drivers. This allows companies building Alpha-based systems to differentiate their products by having a vast choice of possible devices.

To run the device drivers unmodified, I had to duplicate some services of the Linux kernel. Originally, I planned not to have real interrupts, but instead to poll the drivers. This was the way that Milo worked in Linux 1.1.68. However, once I started to try and get the NCR 53C810 SCSI driver working in Milo, I ended up needing proper interrupt handling, and it seemed best to take the interrupt handling directly from Linux, which I did.

I have tried to keep the number of Linux services that I have had to duplicate in Milo to a minimum. After all, as Linux progresses, these routines tend to need re-writing. A good example is the change from 1.1.68 and 1.2.8; the floppy driver changed its way of determining that it was running during kernel initialisation. This caused me headaches as I figured this out.

Maybe over time I will incorporate more of the real Linux kernel into Milo, but it is supposed to be the Mini loader, so I do not want to add the whole of the kernel into it. Right now, Milo includes the PCI BIOS code, the block device code, the interrupt handling and DMA code directly from the kernel. The scheduling services are mine and I cannot see them changing unless I add multi-threading support.

Using Milo

The final functional piece of Milo is the part that most users see, the user interface. Milo can operate via the serial port, but mainly people use it via the system console. For this reason, it must have some keyboard and VGA initialization code.

The keyboard code is very very simple and does just enough to take in commands correctly. Linux itself assumes that some BIOS code has initialized the VGA device and its console device drivers just use it; that meant that Milo had to initialize the VGA device. There are two ways of doing this. The first is to have very simple ISA VGA initialisation code, and this is how Milo first operated. The second way is to include BIOS emulation code that can run the on-board initialisation (which is Intel x86 object code) from the different video cards. David Mosberger-Tang pulled this part of Milo together, with the result that it can successfully initialise a number of common ISA and PCI graphics cards.

The Milo interface is meant to be very simple and do just enough to get the right kernel loaded and to pass the right boot arguments to it. Typing anything other than a legal command displays all of the commands available. Right now, Milo assumes that all devices that it can see are available to boot from and will attempt to use the EXT2 file system with them.

Loading the Loader

Milo was developed on an Alpha evaluation board (an EB66, which is a 21066 based system similar to the AxpPCI33). This meant that loading and testing Milo was easy, since the Evaluation Board Debug Monitor was running. However, for real systems like the AxpPCI33 (Noname), Milo needs to be loaded some other way.

Alpha-based systems boot in several steps. The first step, immediately after power on, is to clock the SROM code directly into the I-Cache stream and then to start executing it. This code does really basic system set up such as figuring out how many DRAM slots are occupied, and with what size of memory. The next step varies from system to system, but essentially that SROM code loads the firmware code (whatever it is) into memory and passes control to it in PALmode. This is the PALcode reset entry point for the image. Some firmware, notably Milo, has the address of the entry point to the user-mode firmware defaulted in the PALcode, and that is where control is passed to when the PALcode reset code has finished initialising the system. Other systems have this information stored in NVRAM or infer it from jumper settings.

For a variety of reasons, Milo can be loaded from a failsafe boot block floppy, from flash, and via the Windows NT ARC firmware. What varies most from system to system is where the SROM code is able to load firmware from. On the AxpPCI33, the SROM code is capable of loading from flash, from a serial line, or from a failsafe boot block floppy. On the AlphaPC64, the failsafe boot block floppy is not supported. All of this is controlled by jumpers and/or boot options saved in NVRAM (in the TOY clock in the AlphaPC64's case). There are systems that do not support flash and instead have ROMs. These are not easy for users to change without access to a ROM blower. and so yet another way must be found for Milo to be loaded. Paradoxically, you could load Milo via the SRM console, but a more fruitful approach is to load it via the Windows NT ARC firmware, since that is the firmware these boards ship with.

There are a number of ways to put an image into flash, and for this reason Milo supports running any image, so long as it is linked to where the Linux kernel usually is. In this way, I can build images that update the flash when loaded and not burden Milo with knowing about the flash requirements of every different system that it runs on.

Loading via the Windows NT ARC console is interesting. On Alpha, Windows NT runs in “super paged” mode, which does not support KSEG addressing—which is unfortunately exactly what Linux needs for fully 64-bit operations. However, all PALcode implementations must support the “Swap Pal” call, and this allows you to change from one mode to another. The Windows NT ARC console has within it the notion of running images and providing services to them so long as they are built to run in the appropriate addressing mode and run at a safe place in memory.

Thus, the Windows NT OS loader is in fact an executable image which gets loaded in order to load Windows NT using the appropriate call backs. I have written a very simple OS loader whose only function is to load Milo, which in turn loads Linux. It is this simple loader which makes the “Swap Pal” call which causes control to be passed to Milo and KSEG addressing turned on. From then on, Milo operates exactly as before with the addition that it can execute commands passed via the [cw]OSLOADOPTIONS[ecw] environment variable for this boot option, and thus boot directly without pausing at the Milo prompt.

I have tried to eliminate this need for an image that is built under the Windows NT firmware development tree; unfortunately this is not possible, so I have kept the functionality of this part as minimal as possible.

Of course, loading Milo via the Windows NT ARC console is one way to get Milo running so that Milo can run the flash update utility to put itself into flash. Alternatively, it can be a way of running either operating system without one interfering with the other.

The Future

Milo is in its infancy and I hope to see the Linux community add into it what they need. After all, that is part of the attraction of Linux itself—a community of able, enthusiastic programmers adding to the effort.

David Rusling (david.rusling@reo.mts.dec.com) lives in Wokingham, England with his wife, two children, 3 cats, and his 1977 MGB GT. He works for the Semiconductor Division of Digital Equipment Corporation, and he thinks that Linux on Alpha is the best thing that he's been involved with in his 10 years at Digital.

Archive Index Issue Table of Contents

Advanced search

Linux Universe

Christopher Boscolo

Issue #21, January 1996

The book is actually an installation guide with some reference material tacked on to the end.

Authors: Stefan Strobel and Thomas Uhl

Publisher: Springer-Verlag

ISBN: 0-387-94506-7

Price: $34.95

Reviewer: Christopher Boscolo

While perusing the Linux section of a local book store, I ran across *Linux Universe*. The teasers on the back of the book describe a 32-bit multi-user/ multitasking UNIX system that runs directly from a CD-ROM. They also mention: easy access to the Internet, a graphical administration tool, and ELF file format. At first, I thought this was yet another Linux book accompanied by existing Linux distributions on CD-ROM. However, I discovered that *Linux Universe* was a completely new distribution. The book is actually an installation guide with some reference material tacked on to the end.

It is not clear which type of user *Linux Universe* is targeting. The professional-looking install program and GUI administration tool would seem to indicate a target audience of beginner to intermediate users. For this reason, I paid special attention to ease of use and clarity of the documentation.

What Is Included

The book contains 7 chapters of installation instructions, a reference, and the *Linux Universe* CD-ROM. Chapters 1-3 contain an introduction and system

requirements information, while chapters 4-6 cover the installation and configuration. Chapter 7 describes how to use the GUI administration tool, and the purpose of system directories such as /etc and /var. The last section of the book is a reference containing UNIX command descriptions. However, the book was by no means a complete reference to configuring a Linux system.

Linux Universe uses System V style startup scripts, (i.e., rc1.d, rc2.d...). It also uses the new ELF format executables. The 1.2.0 kernel that is installed includes support for most hardware. It includes X11R6 and most of the popular utilities found in other distributions.

One area where the Linux Universe distribution seems to fall short is in telecommunications and Internet access. Although the book mentions easy Internet access, I could not find Netscape or Mosaic, and PPP support is not compiled into the kernel. [Licensing restrictions make it difficult to put Mosaic and Netscape on a CD—ED]

Two Linux Universe utilities make it shine as a potential commercial distribution: the Boot Manager and xadmin. Linux Universe uses its own OS loader instead of LILO. The Linux Universe boot manager is probably the best boot manager I have used. Its 3D looking text interface displays a countdown while booting, and allows you to interrupt it. You can also change what and how you want to boot on the fly. This allowed me to add a configuration to boot my previous Linux version without having to reboot with the new configuration.

The second great utility is xadmin. xadmin is a wishx application that allows you to configure almost every aspect of your Linux system. With xadmin, I added an account for myself, and configured the file system to mount my previous Linux version and my MS DOS partition. xadmin can also be used to configure network information, modem ports, printing, and to change system settings such as time/date. Another nice feature of xadmin is the package install/ uninstall. The Linux Universe distribution treats applications such as emacs or the Ada compiler as packages that can be installed and uninstalled through xadmin. One difficulty was determining which features, such as man pages, were in which packages.

Installation

I wish I could say that the installation was a breeze, but I ran into several snags. The Linux Universe distribution ships only on a CD-ROM, with no floppy disk to do a fresh install. This means you must have DOS or Linux already installed. Although this is common for most distributions, it is handy to have an install disk. First I tried to use the DOS application which starts the Linux install program. This attempt failed due to lack of conventional memory, even though I had over 500KB available, which is what the documentation indicates is

required. The only way I could free up some more conventional memory was to remove my DOS CD-ROM drivers... do you see the problem here? The alternative was to use rawrite.exe to write a Linux Universe installation boot floppy, which worked fine.

With the boot floppy made, I rebooted and was greeted with the Boot Manager, which then fired up Linux and the professional-looking install application. One nice feature of the installation process was the ability to tell the installer where to find Linux Universe installation sources. Along with the choices for different CD-ROM types was the choice of an NFS file system. The book describes the steps of installing Linux Universe from choosing the keyboard type to setting up X-Windows. One complaint about the documentation was that it says to create a swap partition but does not describe how. I also had a problem with the X-Windows installation. The installation program has you select a mouse and mouse port, but when X-Windows came up, the mouse was not configured properly.

Configuration

Configuring the system after it booted was fairly simple, again because of xadmin. The installation book suggests rebuilding the kernel after rebooting, which I needed to do anyway, as I wanted PPP support. However, the book did not mention which packages needed to be installed to build the kernel. After installing the compiler and the assembler, I had to fix some of the links to header files used by kernel sources. With the kernel rebuilt I set up PPP and used xadmin to configure my network information with no problems.

I configured Linux Universe to use XDM. When the system booted, the root window displayed a commercial looking Linux Universe Logo. The default xsession is also set up well. It uses fvwm, and comes up with a utility toolbar down the right side of the screen. Using Linux Universe with the defaults for the user I created went smoothly.

Summary

It is difficult to see exactly which sets of users benefit the most from *Linux Universe*. For beginners, it's is probably not the right choice. The snags during installation and the lack of hard bound documentation would be overwhelming. For beginners I would recommend a more “plug and play” distribution, such as Yggdrasil. For the intermediate to expert users, or users looking for easy Internet access, I would stick to other distributions as well, such as Slackware. *Linux Universe* seems best suited for an intermediate user who wants an easily administered Linux system, but is knowledgeable enough to handle problems when they arise.

Christopher Boscolo (chris@neopath.com) orks as a Lead Software Engineer for NeoPath, where he is working on the AutoPap 300 Automatic Pap Screener System. When he is not working, he enjoys spending time with his wife and son. Christopher has been using Linux for over two years as a development platform for network management package.

Archive Index Issue Table of Contents

Advanced search

Upcoming Events

Phil Hughes

Issue #21, January 1996

Linux Kongress in Berlin and more.

The 3rd International Linux Kongress will be held May 23 and 24, 1996 at the *Haus am Koellnischen Park* in Berlin (*Tagungszentrum Berlin Mitte*). It follows the tradition of the Linux/Internet conference series (Heidelberg 1994 and Berlin 1995) which has been since its inception one of the most important meetings for Linux experts and developers. The conference is a must for those who are interested in Linux technology and applications.

The main focus of the forthcoming conference is on current developments of Linux and its various components. However, since Linux has become a well-established and widespread system, the development of applications and usage in commercial environments—even for mission critical purposes—is another major conference topic. Moreover, the Internet in the context of current Linux developments will be covered by various talks and presentations.

Key speakers of the conference will be Linus Torvalds, Theodore T'so and Alan Cox. Because of the growing commercial interest in Linux, a trade show will be part of the conference, featuring companies that offer products based on or using Linux.

The conference will be organized by GUUG (Association of German Unix Users) and supported by several companies (ASKnet, Fachbuchhandlung Lehmanns, Lunetix, Thinking Objects) and publishers (Addison-Wesley, dpunkt, Thomson/ O'Reilly). Any profits from this event will be used to support Free Software Projects.

For information or registration contact Ms Tauchert in the registration office (Tel. +49-30-8207 406, Fax +49-30-8207 465, e-mail: info@linux-kongress.de ; www.linux-kongress.de).

First Conference on Freely Redistributable Software

The First Conference on Freely Redistributable Software (sponsored by the Free Software Foundation) will take place Friday to Monday, February 2-5, 1996 at the Cambridge Center Marriott in Cambridge, MA. Keynote speakers will be Linus Torvalds and Richard Stallman. The conference will feature two days of tutorials on Linux (Phil Hughes), Advanced Emacs and GCC (Richard Stallman) expect (Don Libes), PERL (Tom Christenson), and other topics, as well as refereed papers.

Peter Salus will give seminars entitled ''Linux: An Open System For Everyone'' and “Installing and Running Linux.” The first seminar will look at Linux from its beginnings through its current capabilities, including a look at what some companies are currently doing with Linux. The seminar will conclude with a look at the future of Linux. Peter's second seminar will consist of a ''Look Under the Hood'' covering what makes up a Linux system, what you need, how to install it and what to do when something goes wrong. Interconnectivity options will also be addressed. Requests for registration materials and full programs may be made by e-mail: conf96@gnu.ai.mit.edu); phone (617-542-5942) or fax (617-542-2652).

Archive Index Issue Table of Contents

Advanced search

Consultants Directory

> This is a collection of all the consultant listings printed in *LJ* 1996. For listings which changed during that period, we used the version most recently printed. The contact information is left as it was printed, and may be out of date.

ACAY Network Computing Pty Ltd
Australian-based consulting firm specializing in: Turnkey Internet solutions, firewall configuration and administration, Internet connectivity, installation and support for CISCO routers and Linux.

Address:
Suite 4/77 Albert Avenue, Chatswood, NSW, 2067, Australia
+61-2-411-7340, FAX: +61-2-411-7325
sales@acay.com.au
http://www.acay.com.au

Aegis Information Systems, Inc.
Specializing in: System Integration, Installation, Administration, Programming, and Networking on multiple Operating System platforms.

Address:
PO Box 730, Hicksville, New York 11802-0730
800-AEGIS-00, FAX: 800-AIS-1216
info@aegisinfosys.com
http://www.aegisinfosys.com/

American Group Workflow Automation
Certified Microsoft Professional, LanServer, Netware and UnixWare Engineer on staff. Caldera Business Partner, firewalls, pre-configured systems, world-wide travel and/or consulting. MS-Windows with Linux.

Address:
West Coast: PO Box 77551, Seattle, WA 98177-0551
206-363-0459
East Coast: 3422 Old Capitol Trail, Suite 1068, Wilmington, DE 19808-6192
302-996-3204
amergrp@amer-grp.com
http://www.amer-grp.com

Bitbybit Information Systems
Development, consulting, installation, scheduling systems, database interoperability.

Address:
Radex Complex, Kluyverweg 2A, 2629 HT Delft, The Netherlands
+31-(0)-15-2682569, FAX: +31-(0)-15-2682530
info@bitbybit-is.nl

Celestial Systems Design
General Unix consulting, Internet connectivity, Linux, and Caldera Network Desktop sales, installation and support.

Address:
60 Pine Ave W #407, Montréal, Quebec, Canada H2W 1R2
514-282-1218, FAX 514-282-1218
cdsi@consultan.com

CIBER*NET
General Unix/Linux consulting, network connectivity, support, porting and web development.

Address:
Derqui 47, 5501 Godoy Cruz, Mendoza, Argentina
22-2492
afernand@planet.losandes.com.ar

Cosmos Engineering
Linux consulting, installation and system administration. Internet connectivity and WWW programming. Netware and Windows NT integration.

Address:
213-930-2540, FAX: 213-930-1393
76244.2406@compuserv.com

Ian T. Zimmerman
Linux consulting.

Address:
PO Box 13445, Berkeley, CA 94712
510-528-0800-x19
itz@rahul.net

InfoMagic, Inc.
Technical Support; Installation & Setup; Network Configuration; Remote System Administration; Internet Connectivity.

Address:
PO Box 30370, Flagstaff, AZ 86003-0370

602-526-9852, FAX: 602-526-9573
support@infomagic.com

Insync Design
Software engineering in C/C++, project management, scientific programming, virtual teamwork.

Address:
10131 S East Torch Lake Dr, Alden MI 49612
616-331-6688, FAX: 616-331-6608
insync@ix.netcom.com

Internet Systems and Services, Inc.
Linux/Unix large system integration & design, TCP/IP network management, global routing & Internet information services.

Address:
Washington, DC-NY area,
703-222-4243
bass@silkroad.com
http://www.silkroad.com/

Kimbrell Consulting
Product/Project Manager specializing in Unix/Linux/SunOS/Solaris/AIX/ HPUX installation, management, porting/software development including: graphics adaptor device drivers, web server configuration, web page development.

Address:
321 Regatta Ct, Austin, TX 78734
kimbrell@bga.com

Linux Consulting / Lu & Lu
Linux installation, administration, programming, and networking with IBM RS/6000, HP-UX, SunOS, and Linux.

Address:
Houston, TX and Baltimore, MD
713-466-3696, FAX: 713-466-3654
fanlu@informix.com
plu@condor.cs.jhu.edu

Linux Consulting / Scott Barker
Linux installation, system administration, network administration, internet connectivity and technical support.

Address:
Calgary, AB, Canada
403-285-0696, 403-285-1399
sbarker@galileo.cuug.ab.ca

LOD Communications, Inc
Linux, SunOS, Solaris technical support/troubleshooting. System installation, configuration. Internet consulting: installation, configuration for networking hardware/software. WWW server, virtual domain configuration. Unix Security consulting.

Address:
1095 Ocala Road, Tallahassee, FL 32304
800-446-7420
support@lod.com
http://www.lod.com/

Media Consultores
Linux Intranet and Internet solutions, including Web page design and database integration.

Address:
Rua Jose Regio 176-Mindelo, 4480 Cila do Conde, Portugal
351-52-671-591, FAX: 351-52-672-431
http://www.clubenet.com/media/index.html/

Perlin & Associates
General Unix consulting, Internet connectivity, Linux installation, support, porting.

Address:
1902 N 44th St, Seattle, WA 98103
206-634-0186
davep@nanosoft.com

R.J. Matter & Associates
Barcode printing solutions for Linux/UNIX. Royalty-free C source code and binaries for Epson and HP Series II compatible printers.

Address:
PO Box 9042, Highland, IN 46322-9042
219-845-5247
71021.2654@compuserve.com

RTX Services/William Wallace
Tcl/Tk GUI development, real-time, C/C++ software development.

Address:
101 Longmeadow Dr, Coppell, TX 75109
214-462-7237
rtxserv@metronet.com
http://www.metronet.com/~rtserv/

Spano Net Solutions
Network solutions including configuration, WWW, security, remote

system administration, upkeep, planning and general Unix consulting. Reasonable rates, high quality customer service. Free estimates.

Address:
846 E Walnut #268, Grapevine, TX 76051
817-421-4649
jeff@dfw.net

Systems Enhancements Consulting
Free technical support on most Operating Systems; Linux installation; system administration, network administration, remote system administration, internet connectivity, web server configuration and integration solutions.

Address:
PO Box 298, 3128 Walton Blvd, Rochester Hills, MI 48309
810-373-7518, FAX: 818-617-9818
mlhendri@oakland.edu

tummy.com, ltd.
Linux consulting and software development.

Address:
Suite 807, 300 South 16th Street, Omaha NE 68102
402-344-4426, FAX: 402-341-7119
xvscan@tummy.com
http://www.tummy.com/

VirtuMall, Inc.
Full-service interactive and WWW Programming, Consulting, and Development firm. Develops high-end CGI Scripting, Graphic Design, and Interactive features for WWW sites of all needs.

Address:
930 Massachusetts Ave, Cambridge, MA 02139
800-862-5596, 617-497-8006, FAX: 617-492-0486
comments@virtumall.com

William F. Rousseau
Unix/Linux and TCP/IP network consulting, C/C++ programming, web pages, and CGI scripts.

Address:
San Francisco Bay Area
510-455-8008, FAX: 510-455-8008
rousseau@aimnet.com

Zei Software
Experienced senior project managers. Linux/Unix/Critical business software development; C, C++, Motif, Sybase, Internet connectivity.

Address:
2713 Route 23, Newfoundland, NJ 07435
201-208-8800, FAX: 201-208-1888
art@zei.com

Archive Index Issue Table of Contents

Advanced search