The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

PC Administration Tools: Using Linux to Manage Personal Computers

Jim Trocki
American Cyanamid Company

PC Administration Tools: Using Linux to Manage Personal Computers

Jim Trocki – American Cyanamid Company

ABSTRACT

Personal computers in a networked environment can provide users with access to a broad set of distributed resources. Unfortunately, the management overhead of maintaining PC clients can become overwhelming, especially with a large installed base. Popular PC operating systems do not provide system administrators with a set of efficient and flexible management tools that can take advantage of a networked environment. UNIX system administrators are accustomed to having such tools at their disposal to handle common administration tasks, such as software upgrades, initial machine installation, networked file transfer, and remote backup.

This paper describes the PC Administration (PCADM) tools developed to provide PCs with a UNIX environment and robust tool set for client administration purposes, without installing supporting software on individual clients. Linux, custom scripts and libraries, MD5 signatures, and freely available software including Perl [Schwartz], Bash, and SAMBA are used to accomplish this task – all made accessible from a single floppy disk.

Motivation

Managing PCs is time-consuming and cumbersome, partially due to how desktop PC software has evolved from a stand-alone environment into a larger, networked setting. In order to minimize time spent managing clients, support personnel need the ability to install and configure software on multiple machines simultaneously. This is both a technical and a logistic problem; most PC software installation programs are highly interactive, which limits the possibility of simultaneous installations for each PC support person, since physical attention is required for each client. The PCADM tools were designed to minimize, if not eliminate, interaction for software installations, and to provide a high level of remote accessibility to perform these tasks.

Not only is software installation and upgrading less than convenient, but the accessibility of the data stored on the local client is extremely limited. This complicates transferring data to alternate machines for analysis, an important need in a scientific R & D environment. Backup of data on a client PC directly to tape or other medium on a UNIX workstation is extremely difficult with tools supplied with popular PC operating systems. The PCADM tools provide this type of accessibility without expensive and proprietary PC software.

Design Goals

UNIX supplies an excellent environment for system administration compared to typical PC operating systems. The "tool set" approach is far more valuable for computer administration purposes than the "one big tool". There should be a set of extensible tools which can be easily applied to new situations which appear in PC administration. These tools should fit into the shell scripting paradigm in order to leverage from other shell-oriented tools.

User interaction should be limited whenever possible, and the input of redundant information should not be necessary. When performing maintenance tasks on a large number of clients, both speed and accuracy are important. Jobs performed in non-interactive mode complete faster than those which require persistent user interaction. If interaction is required at all, it should happen as near as possible to the initial invocation of a utility.

Minimal or no software should have to be installed on the client. If client software installation is a requirement, then it becomes an unwanted part of the overhead of administering PCs. This is what we are trying to reduce.

When possible, network configuration (IP address, etc.) should be gathered from already installed PC networking software, e.g., Sun PC-NFS, DEC Pathworks, and Microsoft Windows 95.

A Functional Linux System from a Floppy

Linux was chosen as the base system because of its broad PC hardware support, the support for the DOS FAT filesystem, and its availability.

All Linux installations provide bootable floppy disks which contain a kernel, libraries, utilities, and scripts used to perform the installation. The floppy normally contains a Minix filesystem which is loaded into memory by the ramdisk driver, and then mounted as root. The PCADM boot floppy is based on this technique for booting. Network drivers for several adapter models used on site are compiled into the kernel. In some cases, there are boot disks including alternate drivers for less common ethernet cards. This method has the advantage of portability between machines, and does not require software to be installed on the client disk.

Obviously, there is little room for a complete system on a single floppy, so the first priority during the boot process is to establish IP networking in order to NFS mount /usr from a file server, which will hold all additional non boot-specific software. Software is then limited to whatever the administrator wishes to make available. Shell utilities such as sort, find, expr, grep, head, etc. are in the implementation, in addition to a vi clone and networking utilities like telnet, ftp, ping, and rsh.

The floppy is constructed to hold the boot loader, a Minix filesystem containing a Linux 1.2.13 kernel, libc and libm, ifconfig, route, a reduced shell, Perl 4.036, and custom scripts. This will be referred to as the PCADM boot disk.

LILO As The Boot Loader

The LILO [Almesberger, 1995] boot loader serves the purpose of loading a kernel image from the floppy. It also allows the user to supply kernel parameters (usually for drivers), but any parameters which are not recognized by the kernel are set as environment variables. The PCADM boot scripts take advantage of this feature, and IP configuration hints can be passed via LILO parameters. Other user-supplied parameters are used to enable the SAMBA daemon for SMB (LanManager) protocol exporting of the local disk, to prevent IP information from being acquired from locally-installed software, and more.

The following example will tell the boot scripts to use the IP address of ‘‘10.0.0.1’’, set the hostname to ‘‘pc001’’, export the ‘‘/dosc’’ directory using the SMB protocol, to ignore the locally installed TCP/IP configuration, and to give the user one second to make changes to the network configuration settings:

```
LILO: param ip=10.0.0.1 hn=pc001 \
          samba=1 nogrep=1 wait=1
```

Currently recognized options are:

`ip`	IP address
`hn`	Hostname
`nm`	Netmask
`na`	Network address
`gw`	Default router
`ns`	Nameserver
`dm`	Domain name
`samba`	Start SMB server exporting /dosc if nonzero
`nogrep`	override local TCP/IP software configuration if nonzero
`wait`	give user time to change configuration if nonzero

Default configurations are stored both by LILO as kernel parameters, and also in a ‘‘defaults’’ file on the boot floppy.

Memory

The PCADM disk will work properly on a machine with eight or more megabytes of RAM. Mileage may vary with less memory, and less is definitely not recommended. No attempt is made to establish a swap partition or a swap file, so all operations to be done should be able to fit in physical RAM. For our purposes, eight megs has been suitable. Four megs will not work. Since nearly all networked PCs on our site have at least eight megs of RAM, this does not hinder the use of the PCADM disk.

Boot Scripts

The boot scripts perform the function of acquiring the network configuration for the current host from the user, from boot loader hints, or automatically from DOS TCP/IP software which is already installed. Because this task is easier accomplished in Perl rather than sh, the Perl interpreter must reside on the boot floppy.

The boot scripts mount the local DOS FAT (the native DOS filesystem) boot partition on to the root filesystem (which is located in the ramdisk) as /dosc.

If no IP parameters are supplied via the boot loader, scripts attempt to identify locally installed PC networking software, such as Sun PC-NFS or DEC Pathworks. Based on what is found, the network configuration (IP address, netmask, default gateway, and nameserver) is extracted from the respective configuration files for the product (/nfs/network.bat and /nfs/hosts for PC-NFS, /pw/cfg0001.tpl for Pathworks), and used for the host.

If none of the above suffices, the scripts prompt the user to confirm the configuration, and parameters may be interactively changed. Since we favor minimal interaction, a small timeout can be set via a LILO parameter, so that it is possible to insert the floppy into several machines and not need to attend to them, other than powering them on, and all further operations can be done from a remote location via Telnet.

The final stage in booting, once networking has been configured, is to mount /usr from a server. The rest of the necessary software is located on the server, including software installation packages, miscellaneous binaries, and daemons (telnetd, ftpd, rshd, etc.). /bin/sh is removed from the ramdisk filesystem (which was previously a minimal shell), and a symbolic link is made to /usr/bin/bash.

Remote Accessibility

Once the system has booted and /usr is mounted from the server, inetd is run. It is then possible to telnet, FTP, or rsh to the client PC, allowing full operation of administration tasks from any location via the network. Standard, open protocol, non-proprietary utilities are used to connect to the client to perform maintenance functions. Nearly any host which supports TCP/IP will have Telnet, FTP, or rsh client utilities installed, so the administrator need not install the "other half" of remote administration software packages on to the workstations where the maintenance is to be done from. This is the functionality that all current PC networking packages lack – especially the capability of executing commands on the client host.

Accounts reside only in /etc/passwd on the floppy. The password-protected "root" account and a password-less "user" account allow login. Only root may write to the local filesystem. We are not overly concerned with the security of the PC, since physical security of the typical PC is normally non-existent, and anything could be done to the PC with or without the use of the PCADM disk. DOS or Windows provide no effective security mechanisms, anyway. Any software which attempts to limit access to the PC can be overridden by booting a floppy disk, and a password-protected BIOS is ineffective without physical security.

The PCADM effectively allows anyone with a PC to be root on that machine, and it does make it easier to exploit certain vulnerabilities in the servers. NFS security is one vulnerability that is easy to exploit, as is rsh (with the appropriate .rhosts). Also, various denial-of-service attacks are possible because the local IP address is completely configurable. Password sniffing is also possible with tcpdump. However, these vulnerabilities exist with or without the PCADM floppy, an the system administrator should already be taking appropriate precautions.

PC Software Installation

Since Linux is able to read and write to a DOS FAT filesystem, it enables us to install and configure software from an arbitrary vendor using the PCADM disk. The challenge then becomes mimicking the behavior of the vendor's installation code. Using Rivest's MD5 message digest algorithm [Rivest, 1992] and some other techniques similar to Spafford and Kim's Tripwire [Spafford et al, 1994], we are able to identify the exact results of the vendor's software installation program. This includes discovering files which are added to or removed from the client disk. The MD5 checksums identify exactly the files which were modified during the installation.

A MD5 checksum of each file on a PC with a bare minimum setup (DOS and Windows only) is recorded using the PCADM disk and the "makemd5db" script, which recursively traverses the local filesystem. Files which are suspect to modification (CONFIG.SYS, AUTOEXEC.BAT, all Windows INI files, etc.) are saved to a temporary location on a file server, and the machine is then rebooted with DOS. The vendor's software is installed using the provided DOS/Windows setup program, and PCADM is again booted. Another MD5 database is formed with "makemd5db", and then a Perl script is used to compare the first and second checksum databases. The output is a list of files which were added, removed, or modified. Files which were changed (hopefully ASCII text files) are compared to the originals, which we previously preserved, using "diff -c", and the results of the software installation are identified.

Files which were added are collected from the client disk into a compressed TAR file, and moved to a location on the file server for later use. A MD5 database of checksums is also stored along with this image for individual file identification.

Scripts and Libraries

The PCADM disk provides a flexible environment with the proper tool set in order to complete the software installation. The next task is to generate a set of scripts which will reconstruct the installation on clients using the PCADM disk.

PC software installations can be separated into a few simple tasks: identifying dependencies, copying files to the client disk, and modifying several types of configuration files.

In the spirit of the "UNIX way", whenever possible and practical, we use existing tools which perform the tasks we need to, such as copying and modifying files. It would seem obvious to do such a thing, but in the world of PC software, each vendor seems to invent a wheel with a different diameter and a proprietary "axel interface", preventing it from being used on a someone else's (or particularly our own) cart. Only when existing utilities lack the functionality we need do we create our own tools. Kernighan and Pike's book [Kernighan, 1984] gives an excellent discussion of how the UNIX programming philosophy is based on the relationships between programs, and the method of using programs for building programs. This, combined with the expressive grammar of the shell, allows programmers to phrase their ideas in a flexible manner. Van Jacobson's keynote speech at USENIX '96 addressed this idea about what makes UNIX good, especially compared to popular PC operating systems.

Most dependencies for DOS or Windows software deal with particular versions of dynamic link libraries (DLLs), which can easily be identified using MD5 and compared to expected versions. Some PC software installation programs judge a DLL version by the date on the file, which is not unique enough criteria for identification. A priori knowledge about these types of dependencies from vendor documentation can be an advantage, however this will normally be discovered by trial and error.

The matter of copying files to the client disk is the simplest. However, it is not always adequate simply to copy files to the client, for fear of overwriting things which might already exist, and possibly losing valuable data. For this scenario, a utility is provided which will rename files which already exist, and creates a DOS batch file on the fly. In case of some misfortune, this batch file can be executed under DOS, and things the files will be restored to their state before the installation occurred.

PC software installation programs will usually modify the same types of files. CONFIG.SYS and AUTOEXEC.BAT usually get modified, as do WIN.INI and SYSTEM.INI. Since these are common types of ASCII files, we would like a set of tools to make editing them simpler. In the case of AUTOEXEC.BAT and CONFIG.SYS, sed may or may not be the right choice of tool, depending on the type of modification. These files are free-form and have no apparent structure.

However, in the case of the standard Windows INI files, sed will not work without some trickery. Since these files have a somewhat rigid structure, it would be easiest to perform an operation like "add the entry BorderWidth=3 to the [Windows] section of WIN.INI". To make this type of phrasing possible, a Perl library needed to be developed to handle this type of file. Using the library, the code which accomplishes what this example wants is the following:

```
&change_ini ("/dosc/windows/win.ini",
  "Windows & BorderWidth & 3") ||
      die "failure\n";
```

Other functions defined in the library will report the value of a variable in an INI file, since this type of operation is normal to the installation process. Utilities which will rename files to a unique name, perform logging functions, check free disk space, and write DOS batch files (among other things) are also included as part of the tool set for creating installation scripts.

Software Package Sets

Utilities are divided into sets specific for a given software package, e.g., Vendor A product, Vendor B product. Within a given software set there will be multiple scripts handling the logical stages of the installation of that package. A script set for a typical package will include the following:

- package.getdefs – "primes" the configuration by interpreting an existing configuration and supplying default hints to the following scripts. An example would be upgrading a version of networking software, and setting the parameters of the new version to be the same as the old version.
- package.getinfo – gets configuration information from the user. Here, interaction is minimized. The output of this stage will be a file holding "name=value" pairs, which will be interpreted by the configuration and setup scripts.
- package.setup – performs the operations of transferring the software files to the local machine and modifies the local configuration to include the appropriate entries, but does not insert the user-entered configuration data.
- package.configure – modifies local configuration files to reflect the configuration options entered by the user in the "package.getinfo" script.
- package.install – a shell script driver which controls the overall process of an in installation, from retrieving the defaults to configuring the package.

  By separating the installation into somewhat discrete steps, we are able to make separate scripts that will configure or re-configure a machine which already has the package installed.

Drawbacks

The largest drawback is in the difficulty in identifying all dependencies by looking at installed files. Hopefully, adequate vendor documentation can aid in this process.

Proprietary and visually indecipherable data formats can be extremely frustrating. For what reason vendors choose to store small or medium sizes of configuration data in non-ASCII formatted files is beyond my own and several of my co-workers' understanding, but nevertheless, this is an all too popular practice. The Windows 95 registry is a notable offender, in addition to Windows 3.1 .GRP files and OLE registry.

The newly created installation scripts can be painstaking to debug, but careful initial preparation in the identification phase can minimize problems. The time saved (when ammortized over a large number of PC installations) makes the debugging effort well worth it.

Timing Figures

By eliminating the user interaction, in some instances we can achieve some excellent speed increases. The example I cite is an installation of all standard software which is delivered to the client. This includes Microsoft Word, Excel, Lotus Freelance Graphics, Netscape, among other things.

Without the use of the PCADM utility, an installation would take approximately an hour and fifteen minutes or more to "get it right". This involved installing some software from floppy and some from distributions located on the network. The process is highly interactive, demanding constant attention from the person, so only one could be done at a time by a single person.

The same installation, when done with the PCADM utility, clocked in at less than ten minutes.

An additional scenario where the PCADM installation methods can be exploited is the setup and configuration of new PCs. Often an organization will receive a volume of identical newer model computers which need the ‘‘standard issue’’ software installation and individual network configuration (IP address, default network filesystem mounts, etc.). In this situation, a single PC can be installed from a single, preconfigured ‘‘clone’’ machine in ten minutes or less. In addition, multiple machines can be installed simultaneously, stacked together without keyboards or monitors – just ethernet connections.

Data Transfer and Backup

The PCADM utility can also be used to turn a client PC into an NFS or LanManager server. This enables us to transfer data between ‘‘clients’’ instead of using temporary disk space on an intermediate server. This capability is useful when transferring software or data files from an older PC over to a newly purchased replacement machine. The conventional means for doing this was to backup the old machine to a portable quarter inch tape via the parallel port, and restore it to the new machine. This is an extremely slow and arduous task. With the PCADM disk, we have even left the old machine (without a keyboard or monitor) under users' desks, and the clients have used Windows File Manager to transfer only the files that they need, nearly eliminating the need for the intervention of the administrator.

The situation when locally stored data (word processing, spreadsheet, database files, etc.) must be transferred to another location can be problematic. Not always do the operating system files or applications on the local disk need to be moved, so it is desirable to identify the user data files from other files. However, a complication is that users often misplace files. A simple ‘‘find’’ command can find files with common extensions (‘‘*.doc’’, ‘‘*.xls’’, etc.), but this technique alone is not enough to reliably select data. Users might possibly (and surprisingly often do) choose filenames that don't correspond with the conventions for the applications they use.

Because of the above problems, a filtering tool was developed that uses several criteria to identify user data. Filenames are first compared to regular expressions which attempt to match the normal filename extension conventions. If no match is found, then an additional comparison is used based on ‘‘magic numbers’’, which uses specific byte patterns in the beginning of the file to identify user data. Files which match any of the above criteria pass through the filter, which are usually fed as input to a copying utility such as ‘‘cpio -p’’.

Care must be taken in assembling the magic number list. If magic numbers are misinterpreted and omitted, it could possibly result in missing some important data during the transferral process. The method used for our purposes was to construct a list of the first five bytes of every file on one of our file servers (over 14,000 user data files). This list was then grouped based on file extension, resulting in output similar to this:

```
dba52d00 7683 doc(7115 92.61%)
     tmp(123 1.60%) bak(100 1.30%)
     drl(901.17%)
```

The first column is the hexadecimal pattern for the first four bytes of the files, followed by the number of files with this pattern, and percentages of how many files from the second column contained that extension, i.e., over 92% of files starting with the pattern ‘‘dba52d00’’ ended with the extension ‘‘.doc’’. We know that the ‘‘.doc’’ files are from our word processing package, and after inspecting other high percentage occurrences of the ‘‘doc’’ extension, it is reasonable to use the string ‘‘dba52d00’’ as part of our magic number criteria.

File extension regular expressions are stored in a separate file to make it simple to add patterns, and has the following format:

```
\.do[ct]    Word processing file
\.xl[cs]    Spreadsheet file
\.pre,\.flw Presentation pkg file
```

‘‘Magic numbers’’ are stored in a file with the following format:

```
dba52d00   Word processing file
01020304   Spreadsheet file
```

This technique has so far yielded acceptable results.

Conclusions

The PCADM boot disk and tools have been useful to the administration tasks of our organization. They provide enough flexibility to customize software installations and reduce installation times. Many convenient operations simply are not possible without PCADM, particularly in the area of remote access.

Availability

Source code and images of the PCADM boot disks will be made publicly available. However, at the time of writing, the author is still looking for a facility from which the packages can be distributed. Volunteers with ample disk space on their FTP servers may contact the author at ‘‘trockij@pt.cyanamid.com’’, and further arrangements can be made.

Bibliography

Rivest, R. ‘‘The MD5 Message Digest Algorithm,’’ RFC 1321, 1992.

Kernighan, B., Pike, R. ‘‘The UNIX Programming Environment’’, Prentice-Hall, Inc., 1984.

Schwartz, R., Wall, L. ‘‘Programming Perl’’, O'Reilly and Associates, 1991.

Spafford, E., Kim, G. ‘‘The Design and Implementation of Tripwire: A File System Integrity Checker’’, Dept. of Computer Sciences of Purdue University, 1994.

Almesberger, W. "Generic Boot Loader for Linux", User's Guide, 1995.