by RICHARD MORIN, Technical Editor

UNIX Administration Notes, Part 1

Looking through some lecture notes for a decade-old course on UNIX systems administration, I ran into some material that still seems timely today.

Systems Administration Mythology

All of us, at one time or another, encounter some rather peculiar mythology concerning systems administration. I know that you don't believe these myths, but you may be able to use some ammunition when discussing these issues with your peers (and/or management):

• Systems administration is optional. UNIX loves to create files: log files, temporary files, spool files and so on. And, unfortunately, it doesn't always clean up after itself. With all this activity, it is quite possible for things to break. If they don't get fixed, they can break worse. File system damage, in particular, tends to propagate over time.

• System administration is just "housekeeping" (and thus insignificant). No more (or no less!) insignificant than ordinary "housekeeping." A properly administered system runs efficiently and cleanly, and presents an image of order. A poorly administered system causes inefficiency and frustration for all of its users.

• Anyone can do it. No. A system manager must be a competent UNIX user and must have a reasonable set of attitudes (more on this later) about the task. There is also some basic knowledge of systems administration that is needed, but much of this can be picked up along the way.

• Only gurus can do it. No. Some parts of systems administration are very tricky, to be sure, but most systems administrators are not gurus. It is helpful to have a guru on call, however.

• Somebody else will do it. Not if you're the only user or the owner of the system. This makes you the systems administrator. Aside from occasional "help" from passing system crackers, you're on your own. Nor can you count on the vendor to automate everything or even to set things up right in the first place.

• Security isn't important here. OK, you run a nice, loose shop, and everyone is completely honest. Or perhaps this is your own private system, sitting in your den. Well, there are nasty people out there with modems and Internet

connections, for starters. Failing that, some of your employees or friends may not be as honest as you'd like.

The best reason for security, however, has to do with human error and damage control. A secure system will keep your users from shooting off each others' feet when they make mistakes.

Professional Attitude

An important component of effective systems administration, attitude is particularly crucial if the system is going to have more than one user. Some users, on receiving the root password, decide that they are now the local deity. Maybe so, but there are limits:

• Omniscience. Read the manual; don't just assume you know everything. Besides, vendors occasionally change things around. It is also useful to ask users about their needs and preferences. They have a totally different perspective from yours, so you may be surprised (and enlightened) by what you hear.

JACQUES COURNOYER

• Infallibility. Don't stay logged in as root all the time, and be very careful when you take on root authority. Fingers have been known to slip, and UNIX assumes that anyone who is logged in as root knows what s/he is doing.

• Omnipotence. As the systems administrator, you are pretty much in charge of things. You can take the system down whenever you like, throw away anybody's files, and generally do whatever you like. In a word, don't. Users have a right to expect courteous behavior on your part, and it is only proper to provide it.

Scheduled downtime, for instance, should be announced well in advance. It should also take place during times (e.g., weekends) when the system would normally have few users on-line and when you have time to fix anything you break.

• Immortality. You won't be around forever, so leave some notes around for your successor(s). Be sure to write the really critical stuff into a bound notebook. Remember, when the system is down, so is your on-line documentation.

Raison D'être

So much for political indoctrination. Now let's look at some specific reasons for performing systems administration:

• To protect the integrity of programs and data. You don't want things to get lost, stolen or sabotaged. Neither do your users. As the systems administrator, your task is to make sure that bad things either don't happen or can be resolved with minimal losses.

• To help the system run smoothly. UNIX isn't Windows, let alone MS-DOS or Mac OS. It is a big, hairy, complex operating system that requires both proper setup and continuing support to work properly.

• To integrate new devices and users into the system. As new devices and users are added to the system, the systems administrator must modify and/or create assorted files to reflect this. If this is done in a sloppy manner, the result will be security holes, inefficiency and possible loss of data.

• To add desired features to the system. The UNIX distribution can't possibly include every package you might want. Consequently, there are many freely redistributable utilities and proprietary packages you may want to add.

Levels of Systems Administration

There are several levels of systems administration:

• Elementary. These tasks constitute most of an administrator's duties. They involve adding and removing users, backing up files, moving files around and so forth.

• Intermediate. Now we get into trickier stuff, such as restoring files, installing new devices and versions of UNIX, setting up UUCP links and network connections, etc.

• Advanced. These activities should really be done by gurus, but ordinary systems administrators sometimes get pulled into them. They include hacking sendmail, modifying device drivers and repairing clobbered file systems.

• Specialized. Each site has certain subsystems it cherishes, and these involve a certain amount of administration. The skills may range from elementary to advanced, and the system manager simply gets to cope as best s/he can.

This column introduces most of the elementary tasks, touches on a couple of intermediate tasks, and leaves the rest alone. If you are running a small stand-alone system, making few changes to the configuration, this column may meet most of your needs. Regardless, I suggest that you peruse a good UNIX systems administration text, such as Evi Nemeth's *UNIX System Administration Handbook*, second edition (Prentice Hall, 1995, ISBN 0-13-151051-7).

Bringing It Up, Shutting It Down

• System start-up. UNIX can usually handle start-up by itself, unless something has gotten severely damaged. Let the normal start-up procedure take care of things, noting any peculiarities. If it has real trouble, it will let you know.

Leave some notes around for your successor(s). Be sure to write the really critical stuff into a bound notebook. Remember, when the system is down, so is your on-line documentation.

• Normal shutdown. The system will need to be shut down occasionally for preventive maintenance, adding devices and so on. Use `/etc/shutdown`, giving an explanation and a reasonable amount of warning, for example,

```
# shutdown -h 5 "Need to add a disk drive."
```

When the machine has successfully shut itself down, you may power it off.

• Rapid shutdown. Occasionally, something untoward will happen, making it is necessary to shut the system down at once. If possible, follow the above instructions, using a suitably short amount of time. Otherwise, use the following command:

```
# shutdown -h now "Disk drive is screeching."
```

• Panic shutdown. Smoke has just started to pour out of a critical piece of system hardware. Pull the plug! (You can clean up the mess later.) You may lose some files, but you should have backups for most of them. In any case, the hardware is a bit more critical at this point.

• Accidental shutdown. A large truck has just eaten your utility pole, and the power will be off until the utility company gets things put back together. Unplug the system and leave it that way until the power seems to have stabilized. Power companies have a tendency to switch things on and off a bit when they are getting the power back in service. Your computer system is very vulnerable when it is starting up; being interrupted could cause it to lose or damage files.

Security Basics

Security issues have to do with protection of resources from loss or harm. Computer equipment is expensive and should be protected. Data can be far more valuable (often irreplaceable) and deserves even more protection. I don't want to breed paranoia, but I do suggest that you treat security issues with respect. A modicum of caution now could prevent a great deal of anguish down the road.

Before I get into a discussion of software-related security issues, give some thought to the physical security of your installation. What would prevent someone from simply walking off with components or even entire systems? Is there anything to prevent a cracker from sniffing packets from your LAN? Do you have a regular system in place for performing system backups? Are your backups cataloged and well protected? Do you cycle through your media, retiring it at some point? Do you keep some of your backups off-site, in case of disaster?

UNIX systems assume that there will be a multitude of users, both local and networked. The system accepts responsibility, by and large, for keeping these users out of each others' way. It does have an administrative account (root) with complete authority, but access to the root account is, or at least should be, guarded with extreme care.

UNIX security is implemented via access permissions for all processes, directories and files on the system. Access permissions tell UNIX which users have what kinds of access; UNIX then enforces these restrictions.

As shipped, most UNIX systems have reasonably secure permissions. Let your vendor know if you find a problem. As the systems administrator, you are responsible for maintaining this security. If you open up the permissions on a directory or a file, make sure you haven't allowed any undesirable access to take place.

User and Group IDs

UNIX tracks a user ID (UID) and a group ID (GID) for each file and process. The UID identifies the owner of the item in question. In most cases, the owner is the only user who will have anything to do with the item.

In cases where more than one user (but not everyone on the system) needs to have access to an item, the GID comes into play. The GID identifies a specific group of users (listed in the `/etc/group` file) as having a special (usually increased) amount of access to the item. For instance, a file might be totally accessible by its owner, readable by members of its group, and not accessible at all by anyone else.

Here is a brief summary of key points about user and group IDs:

- Files get their UIDs from their creator.
- Files generally get their GIDs from the enclosing directory.
- Users get their (shell's) UID and GID from `/etc/passwd` at log-in time. They are:
  - Passed to all subprocesses
  - Overridden by setuid and setgid routines

– Checked against every file at “open” time
– User and group names are translated to/from IDs via `/etc/passwd` and `/etc/group`.

File Permissions

Access to files and directories is controlled by their permissions and by the permissions on the directories above. Note: To access a file, you must satisfy the restrictions not only on the file, but for each directory on the path to it.

UNIX supports three sets of permissions: user, group and other. These are checked, in order, with the first matching test controlling access. That is, if you are the file’s owner, only the first set of bits will be checked for you.

Each set contains three mode bits: read (`r`), write (`w`) and execute (`x`). The interpretation of these bits varies somewhat, depending on the nature of the item. For files, the bits are interpreted as follows:

• Read permission allows data to be read from a file.
• Write permission allows data to be written into an existing file.
• Execute permission allows a file to be run as a command.

In the case of shell scripts, read permission is also needed. For directories, the interpretations are analogous but a bit subtler:

• Read permission allows a directory to be read, by `ls` or shell wild-carding–use of pattern-matching metacharacters.
• Write permission allows a directory to be written, as in creating or removing files.
• Execute permission allows a directory to be used in a path name (passed through on the way to a file).

Note: It is a relatively common practice to remove read permission from a directory while retaining execute permission. This keeps stray users from snooping around and allows the owner to say “Pick up `~rdm/A123fW`” to a friend without much fear of any unauthorized party gaining access.

Next month, Part 2 will consider which files reside in the file system and how to deal with permissions. Note: The author would like to acknowledge Jim Joyce and Doug Merritt, whose material played a part in writing this column.

Richard Morin operates Prime Time Freeware (`ptf@cfcl.com`), which publishes mixed-media (book/CD-ROM) freeware collections. He also consults and writes on UNIX-related topics. He may be reached at Canta Forda Computer Laboratory, P.O. Box 1488, Pacifica, CA 94044 or by email at `rdm@cfcl.com`.