by RICHARD MORIN, Technical Editor

UNIX Administration Notes, Part 2

This column concludes my ramblings on systems administration, adapted from some decade-old lecture notes by Jim Joyce, Doug Merritt and myself. I hope you have found it useful.

Inspecting Files

UNIX allows many sorts of things to reside in the file system. The file `/dev/cua1`, for instance, could be a connection to the device driver for the modem serial port. The file `/dev/modem`, in contrast, could be a *symbolic link* to `/dev/cua1`. This provides great convenience and flexibility for both programmers and users. On occasion, however, it's nice to know exactly what a given item is. I can inspect any item in the file system by means of the `ls` command:

```
% ls -alg ~rdm/foo
-rwxr-x---  …  rdm  staff  …  /u/rdm/foo
```

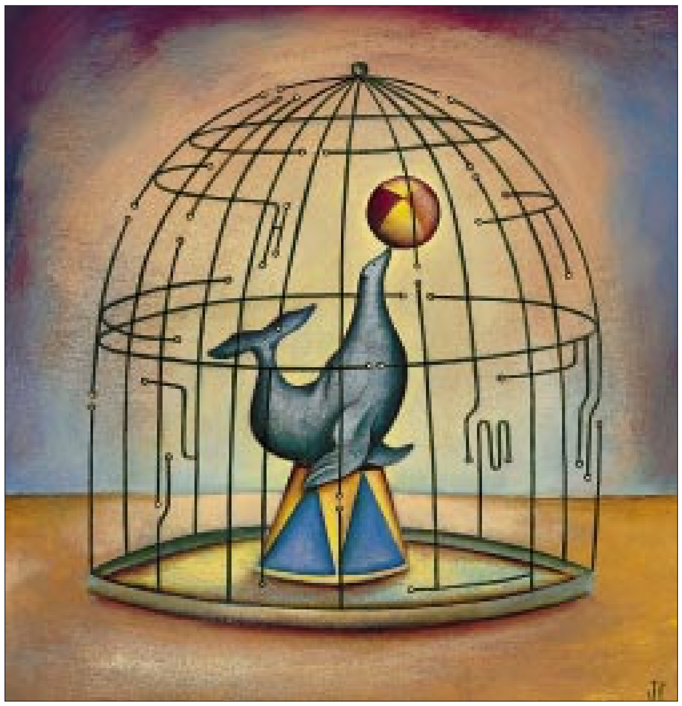
The first character in the output line is the *file type* flag, which indicates what kind of "file" this is. Here are some common file type flags:

-	Ordinary file
b	Block special file
c	Character special file
d	Directory
l	Symbolic link
p	Named pipe special file

In this case, the "file" is really a file. Further, it is owned by a user named `rdm`, has group `staff`, and has permissions `-rwxr-x---`. For more information on the flags and output of the `ls` command, try running `man ls`.

I discussed file mode bits last month. Now let's look at some. In the example above, the first set of file mode flags (`rwx`) allows the file's owner (`rdm`) to read, write and execute the file. The second set (`r-x`) allows any member of group `staff` to read and execute the file. Finally, the third set (`---`) prevents anyone else from accessing the file in any way.

The actual situation is a bit more complicated, of course. To gain access to `~rdm/foo`, a user must also satisfy the permissions on directories `/`, `/u` and `/u/rdm`. And, of course, `root` can almost entirely ignore permissions.


JACQUES COURNOYER

Permissions on symbolic links are even more convoluted. The link itself displays modes `rwxrwxrwx`, but this is misleading. In fact, a symbolic link has no real file modes of its own. Assuming that the system can get to the symbolic link (satisfying the permissions on any directories it encounters along the way), it must now start again at the `root` directory, walking down the complete path specified in the link.

Initial Permissions

The permissions acquired by a newly created file are determined by the creator's current *umask* (usage mask), as set by the `umask` command:

```
% umask  077  # create files as "rwx------"
% umask  027  # create files as "rwxr-x---"
% umask  002  # create files as "rwxrwxr-x"
```

The argument to `umask` is interpreted as an octal value.

Each bit in the value masks (by means of an "exclusive OR") the default permission (777), yielding the permission bits for the resulting file. Thus, a mask value of 27 results in permissions of `rwxr-x---` (i.e., 750 or 027 XOR 777).

Changing Permissions

The owner of a file (or `root`) may change the permissions on a file, using the `chmod` command. Permissions may be specified either symbolically or numerically. For example,

```
% chmod go+x foo
% chmod 750  foo
```

See the `chmod` manual page for more detailed coverage of its usage. A file's group may also be changed by means of the `chgrp` command:

```
% chgrp staff foo
```

The file's owner can change a file's group to any group of which the owner is a member. Group membership is controlled by the `/etc/passwd` and `/etc/group` files.

Predictably, `root` may change a file's group to any group, even if it isn't listed in `/etc/group`:

```
# chgrp fubar foo
# chgrp 12345 bar
```

Finally, `root` may change a file's ownership, using the `chown` command:

```
# chown root foo
```

Editing /etc/group

Each user has a default group, as specified in his or her `/etc/passwd` entry. Although it is not necessary to enter default groups into the `/etc/group` file, your `ls` output will be prettier if you do. It is also a good idea to list the membership of default groups in `/etc/group`, as a bit of useful documentation.

There are several ways of handling default group IDs. On a small system, each account can have its own group, with all joint groups being explicitly defined. A dedicated group (and possibly a dedicated directory) is then created whenever users need to share a set of files. A `staff` group, for example, can let certain users do mundane systems management functions without becoming `root`.

On larger systems, default groups are often useful for project teams or user classes. Note, however, that the `umask` should be set appropriately to avoid accidental security holes.

Each line in `/etc/group` has four colon-separated fields:

1. Group name	alphanumeric
2. Group password	encrypted (see below)
3. Group ID	numeric
4. Members	comma-separated login names

Note: Group passwords are poorly supported and are generally a bad idea. Users don't tend to care for their own passwords very well, and they take no care whatsoever of group passwords. Fill this field with an asterisk, then ignore it.

Backing Up and Restoring Files

There are several reasons for doing backups, but the essential reason is risk avoidance. If you have an adequate backup system, few problems can do serious harm to your data; if you do not, almost anything can cause you grief.

Here is a sampling of the risks your data faces:

• Human error – UNIX is quite willing to let users damage or delete their own files. More files are damaged by slipping fingers than by any other cause. And, of course, the `root` account can remove anything on the system.

• Software problems – If a data structure gets damaged, UNIX may get confused and lose one or more files. Alternatively, a program could crash, damaging working files.

• Hardware problems – Hard drives sometimes start getting read errors; less frequently, they simply jam and start smoking.

• Environmental problems – Mother nature can cause real problems for your data. How well are you prepared for an earthquake, fire, flood or other natural disaster?

• Intentional damage – Arson is always a possibility, as are theft and malicious tampering. Physical security reduces these threats but cannot entirely eliminate them.

It is quite possible to do complete system backups on a daily basis. It is a lot of work, however, and is probably inappropriate for most UNIX systems. I recommend a mixed backup strategy, which should meet your needs at a lower cost than a brute-force approach.

Here is a list of backup approaches:

• Full backups – Every so often, you should make a full backup of your system. This ensures that you don't overlook something that may later turn out to have been critically important.

• Partial backups – By selecting important parts of your system for backup, you can reduce the amount of work involved. How many copies of the `/usr` partition do you really need?

• Delta backups – Some backup programs allow you to save only files that have been modified since the last full backup. If your data is relatively stable, this can greatly reduce the backup effort.

• Disk-to-disk backups – Many risks (human error, hardware and software problems) can be reduced by automated disk-to-disk backups. Have `cron` run your favorite backup program during the middle of the night, compress the output, and save it on a different disk drive. To save space, you can do mostly delta backups.

Media Considerations

Your backups are important; handle them with care. Treat your backup media with care. You don't want it to be stolen, damaged, examined or modified by malicious intruders.

Be sure to employ backup media. It is possible that your system could crash during the backup process. Alternatively, your backup tape (disk, etc.) could develop an error and become unreadable. If you only have one backup, you could be in big trouble. Use sets of at least three tapes, cycling through them as you perform your backups.

You should make use of off-site storage. If your building burns down, will you still have copies of your data? It is very easy to save copies of your backups in another location; the hard part is remembering to do it.

Many sites are beginning to use write-once media for archival purposes. These devices are physically robust, have long shelf lives, and are quite resistant to tampering.

Even with occasional use, your tapes will eventually wear out. Sheer age will also cause your tapes to degrade. Keep track of the age and usage of your tapes; after a few dozen uses or a couple of years of use, take them out of service.

Note: Many sites are beginning to use write-once media (recordable CD-ROMs) for archival purposes. These devices are physically robust, have long shelf lives, and are quite resistant to tampering.

Finally, if you are making frequent backups and saving them forever, you could easily run out of storage space. On the other hand, how important is data from several months or years ago? In most cases, I recommend that you save some snapshots forever, but refrain from trying to save *everything.*

Backup Frequency

How often, when and how should backups be done? These aren't easy questions; several interrelated factors are involved:

• How much data can you afford to lose? Your data is changing constantly; anything short of instantaneous backup (e.g., mirrored disk drives or a RAID system) may lose some amount of data.

• How often can you take your system down? Backups are best done while the system is quiescent, but kicking off your users too often has a cost as well. You may want to take your chances on a few files being in transition during the occasional backup.

• How many resources do you want to allocate to doing backups? Your time has a value, even if you aren't getting paid for it. If you are paying operators to hang tapes, there is a real cost involved.

Like any other form of insurance, backups must be evaluated on a cost-benefit basis. Decide how much risk you are willing to sustain and how much effort you are willing to spend in risk reduction. Just don't let the decision get made by default; you may not like the results.

Backup Methods

UNIX has several programs that can be used for backups, including:

• `cpio` – The `cpio` utility will archive a specified list of files. It combines well with the `find` command, which can create file lists according to assorted criteria.

• `dd` – The `dd` utility is able to copy a bit-for-bit image of a disk partition, or even an entire disk drive. `dd` is seldom a useful tool for general-purpose backups, but it can be quite useful for, say, copying one hard disk (partition) to another. On the other hand, the created images aren't too portable: The destination operating system must be able to read the format used on the initial partition.

• `dump` – The `dump` utility is able to make a logical copy of a disk partition. It is also able to create "delta" backups, containing material modified since a previous backup. `dump` is a useful tool for general-purpose backups, but it is somewhat inflexible–it only dumps entire partitions. With a bit of advance planning, however, this needn't be a problem. Divide your system's disks into a number of partitions, based on expected use and `dump` frequency.

• `tar` – The `tar` utility is very facile at archiving specified directory subtrees. It is also the tool of choice for exchanging subtrees between disparate systems. The standard UNIX `tar` has two problems worth noting. First, it is unable to deal with path names longer than 100 characters. Second, it cannot archive "special" files, such as devices. So, get a copy of GNU `tar` from `ftp://prep.ai.mit.edu`.

Setting Up Disks

UNIX allows disks to be added, but a number of things have to be done correctly, and in the right order. First, wiring it up. A UNIX system will have one or more SCSI buses, each of which can support several devices. Each device is addressed by means of its SCSI ID. Devices on the bus are "daisy-chained," usually by means of external cables. Termination must be provided at each end of the bus.

Each device on a given SCSI bus must have its own unique ID (the CPU almost always has ID 7). An external SCSI device will normally have a switch that sets its ID to a value ranging from 0 to 7. Internal devices use removable jumpers for this purpose.

SCSI devices are commonly daisy-chained, with several units being linked in series. Each end of the bus must be terminated by a set of resistors or an active (solid-state) terminator. The inboard (CPU) end of the SCSI bus is normally terminated by the vendor; terminating the outboard end is the responsibility of the user.

Some SCSI devices are shipped with removable packs of resistors. I suggest that you remove any such "internal" termination, as it is a frequent source of confusion. Use an external terminator (preferably an active one) attached to the last device in the chain.

Next, new disks need to be formatted. A newly acquired disk is a bit of an unknown. It may have bad tracks, inappropriate sectoring, or any number of other problems. Fortunately, Sun and other vendors provide tools for dealing with these issues.

Disk formatting sets up the disk correctly for the system, scans for bad spots, and generally readies the disk for use. On UNIX systems, disk labeling is then performed. The label allows the system to know what kind of disk is being used, how it is laid out (partitioned) and so on.

UNIX wants to find all of its devices in `/dev`, even the disks themselves. In general, however, any needed entries will be present in the standard UNIX system.

If you need a device entry that is not present, try using the `MAKEDEV` script, located in `/dev`. If this does not meet your needs, you can use the `mknod` command to make a file system node for a device, and the `chmod` command to set appropriate permissions.

If a disk partition is to be used for storing UNIX files, it must first be partitioned into UNIX file systems. The `mkfs` and `newfs` commands do this, setting up the assorted control blocks and directories that UNIX needs. `newfs` is easier to use.

Creating a Mount Point

When a file system is “mounted” on a directory, the original contents of the directory disappear. The contents reappear when the disk is dismounted, but you might want to look at them before then. So, it is generally appropriate to create an empty directory to be used as the *mount point* for the new file system. UNIX provides the `/mnt` directory for the temporary mounting of file systems.

The mount point directory can have any desired name, but short names are preferable. Some administrators prefer to name mount points for the SCSI device ID and partition involved, `/1a`, for example; others like to name them topically, `/wombats`, for example. Be sure to set the mode of the directory to 777; other modes can cause the mounted file system to behave weirdly:

```
# mkdir /1a
# chmod 777 /1a
```

Use the `mount` command to mount file systems, and the `umount` command to unmount them. If the `umount` command fails, check for a shell session whose current directory lies within the mounted file system. UNIX will not unmount a file system that is in use.

If a file system is going to be mounted frequently, you should add a line for it in `/etc/fstab`. Use the `noauto` flag if the file system should *not* be mounted automatically at system boot.

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`.