386 User's Log

Moving from DOS to Unix: backup options

BY SIMSON L. GARFINKEL

The BACKUP and RESTORE utilities on MS-DOS systems leave a lot to be desired as a hard disk backup system. Because these programs store information from the MS-DOS file system as they copy the files, they are slower than other disk backup programs that access the floppy disk directly. There is also no easy way to get a list of the files on a BACKUP floppy disk, and it is not possible to RESTORE a file to a directory other than the one from which the BACKUP took place.

On the other hand, when an experienced MS-DOS user sits down at a 386-based Unix system for the first time and wants to copy files onto a floppy disk, a menagerie of choices present themselves, each with its own special-purpose utility program and syntax. In this article, I will discuss the process of mounting a floppy disk as a Unix file system and will then give you an in-depth tour of the two most popular Unix backup utilities: `tar` and `cpio`.

It is possible to put an actual Unix file system on a floppy disk — indeed, this is the format in which Unix boot disks are distributed. To do so requires first formatting the floppy disk and then using the `/etc/mkfs` command to create the file system. (Some versions of Unix have an `/etc/newfs` command that acts as a front end to `/etc/mkfs` and creates the `/lost+found` directory in the new file system.) File systems on floppy disks must be mounted with the `/etc/mount` command before they are accessed and unmounted with the `/etc/umount` command when you are finished with them. Instead of being given prefixes like A: and B:, files stored on the floppy disk are mounted into an already extant directory on the hard disk's root file system, usually `/mnt`. Once your floppy disk has been mounted as a file system, you can simply copy files to it with the Unix `cp` command.

Fortunately, Unix provides ways of using floppy disks for file storage that are faster and more straightforward than this floppy file system approach.

A variety of Unix utility programs have evolved over the years for archiving files to and restoring files from magnetic tape drives. Because the Unix programmer's interface to the tape drive is very similar to the interface for most other Unix mass storage devices, these programs work equally well when used with a floppy disk. Indeed, they treat floppy disks as if they were small magnetic tapes: reads and writes are performed consecutively from the first block on the disk to the last. While this may seem inefficient, it has the advantage of minimizing the time that the floppy disk's head spends seeking.

The two most popular programs used for transferring files between the tape or floppy drive and the hard disk are `tar` (tape archiver) and `cpio` (copy file archives in and out). Another set of programs available on Berkeley-Unix-based systems are `dump` and `restore`. In line with the general Unix philosophy, these programs all provide different ways of accomplishing more or less the same things: copy a file or files to a tape archive, restore them, or print an archive's table of contents.

Whether it is "better" to use `tar` or `cpio` is a subject of long-standing debate among Unix gurus. The programs are actually quite similar; the key difference is that `tar` reads the list of files to be stored from the command line while `cpio` reads the file list from standard input.

Saving Files with tar

To copy a directory of files named `/u/simsong` to a floppy disk or tape drive with `tar`, you might type:

```
$ tar cv /u/simsong
```

The `c` option tells `tar` that you want to create a new `tar` archive. The `v` following the `c` tells `tar` to operate in "verbose" mode. In its normal, silent operation, `tar` prints only error messages.

The name `/u/simsong` is the name of the file or directory to copy. If the name provided is a directory, `tar` automatically copies all the files in that directory. If there are any subdirectories within the named directory, `tar` automatically copies them too, recursively.

Notice that `/u/simsong` is an absolute path name — that is, it begins with a forward slash. Tar files made this way can be restored only to the exact place that they were saved from. A far more flexible approach is to create the `tar` archive using relative path names; more on this later.

If I type the `tar cv /u/simsong` command on my system, I might get a response like this:

```
a /u/simsong/article  1 blocks
a /u/simsong/mbox  32 blocks
```

The `a` at the beginning of each line indicates that `tar` is adding to the file archive. The second field on each line is the name of the file being added, followed by the number of "tape" blocks taken up by the file.

If `tar` encountered an error in copying the file, it might have generated an error message like:

```
tar: tape  write  error
```

This can be confusing to the novice user until you remember that `tar` is a tape archive program and, as such, assumes that it is writing to a magnetic tape even if it isn't. As a reward for your troubles, remember that you won't have to learn the syntax of a new archiving system if you buy a streaming tape drive for your system — `tar` will work equally well with any device, even baroque ones like automatic card punches.

A list of the files in the archive can be made by using the `t` (for titles) option:

```
$ tar  t
/u/simsong
/u/simsong/article
/u/simsong/mbox
```

If you also specify the `v` option, `tar` will print additional information on each file (e.g., the length, permission mode, and time stamp).

Files on an archive can be restored by using the `x`, or "extract" option. Again, the `v` is specified so that you can see what `tar` is doing:

```
$ tar  xv
/u/simsong/article, 321 bytes,
  1  tape  blocks
/u/simsong/mbox, 16000 bytes,
  32  tape  blocks
```

The option `f` can be used to change the name of the device (or file) that `tar` uses to save the archives in. Thus:

```
$ tar  fcv /dev/fd1 /u/simsong
```

would save `/u/simsong` on a $5^1/4$-inch, 1.2-Mbyte disk in the second floppy disk drive (the first floppy drive is `/dev/fd0`).

Most versions of `tar` are compiled so that they default to a reasonable mass storage device (e.g., most Ultrix versions of `tar` on VAXes use the 9-track tape drive at 1600 bpi (bits per inch), while most 80386-based versions of `tar` use the PC's $5^1/4$-inch disk in high-density mode). Alternatively, some versions of Unix have `tar` use a default device named `/dev/tar` (which is a link to the particular device that the site administrator wants to use for defaults). Other versions of `tar` read their default device from a file in the `/etc` directory.

dec 89

Options for `tar` and `cpio`

These are just some of `tar`'s and `cpio`'s options. Check your manual for a detailed listing.

`tar [options] [files]`

Options:

c	Create a new archive.
e	Do not split files to archive across floppy disks (or tapes).
f	Causes `tar` to use a file (or device) for the archive other than the default.
o	Extract files with the owner and group ID of the user running the program, rather than using owner and group that is stored in the archive.
r	Files are written to the end of the `tar` archive.
t	List the names of each file (the "titles") of the archive.
u	Files are added to the archive if they are not present or if they have been modified since they were written to the archive.
v	Verbose mode.
w	Causes `tar` to prompt the user before writing or reading each file to or from the archive.
x	Files are extracted from the archive.
0...9	Specifies floppy disk (or tape) to use.

`cpio <action>[options] [files]`

Action:

-i	Copy files in.
-o	Copy files out.
-p	(Pass) Reads file names from standard input and copies them to the directory given as an argument.

Options:

a	After copying the file, change its access time.
B	I/O is blocked 5 Kbytes to the record.
d	Create directories.
c	Write ASCII headers (for portability).
t	Display a table of contents (use with `-i`).
u	Unconditionally copy the file.
v	Verbose mode.
l	Preserve links.
m	Preserve file modification time.
f	Copy in all files except those given as arguments.

Moving Files with `tar`

In the above examples, `tar` was always given a full path name (`/u/simsong`) as a command-line argument. It is also possible, and usually preferable, to give `tar` a relative path name, which allows the archive to be restored anywhere in the Unix file system. The following two examples illustrate this:

```
$ cd /u/simsong
$ tar cv .
a ./article  1 blocks
a ./mbox  32  blocks

$ cd  /u/joconnor
$ tar  xv
./article, 321 bytes, 1  tape
 blocks
./mbox, 16000 bytes, 32  tape
 blocks
```

The two files archived from `/u/simsong` have now been restored in the `/u/joconnor` directory (assuming that the person performing the extract operation has write access to `/u/joconnor`).

If your intention is simply to move or copy files from one directory hierarchy to another, you can dispense with the tape or floppy disk entirely and pipe the output of one `tar` command into the input of another:

```
$ (cd /u/simsong;  tar  cf - .) |
 (cd /u/joconnor; tar  xpf -)
```

`tar` takes the dash (-), when presented as the name of an archive file, to mean standard input or standard output. The `p` option in the second `tar` command means "preserve," an option that assures that the files will be restored with the same file permissions (read, write, and execute) as they were saved. If the person doing the restore is the superuser, the files will be restored with their user ID intact (i.e., the ownership of the files will be unchanged); otherwise the restorer will become the owner of the relocated files.

Although this may seem a contorted way of moving files when commands such as `mv`, `mvdir`, and `cp` already exist, using `tar` has the advantage of preserving owner and modification times of files, as well as preserving links, when moving files across devices. (When `mv` moves a file across a file system, it is forced to copy the file to the new file system and then delete the old.)

The `tar` example also has the advantage that it can be readily extended, in a networked environment, to move files between networked computers:

```
$ (cd /u/simsong; tar cf - .) |
 rsh  unix2 "(cd /u/joconnor;
 tar xpBf - )"
```

The command `rsh` specifies that the second `tar` command is to be executed on the computer named `unix2`. The second `cd` and `tar` commands are enclosed in quotes so they get sent to `unix2` as a complete command; the added `B` in the second `tar` command assures that blocking will be properly performed over the network connection.

Although commands such as `rcp` and `ftp` exist to transfer files between networked computers, using `tar` is one of the few ways of assuring that the files will be sent with their permissions and owner IDs intact and that files that are linked (with either soft or hard links) are sent as such, rather than expanding each link into a copy of the same file.

`tar` can also be used to gather together many files into a single "`tar` file." These files are typically given names like `emacs.tar` or `uucp.tar` and are frequently used to distribute projects that are made up of lots of small files, like source code distributions or the individual chapters, drawings, and associated files of a large document. To create a `tar` file for a project that is kept in a directory `/u/simsong/Book` you might execute the command:

```
$ cd /u/simsong
$ tar  cfv  book.tar  Book
a  Book/ch-01
a  Book/ch-02
...
```

Using other options to `tar`, it is possible to add files to a tape only if they are not there (or if they have been modified since last writing), to create `tar` archives that span more than one tape (or disk), and to prompt the user for a yes/no response before each file is saved or restored. You should check your Unix manual for details.

`cpio`

Like `tar`, `cpio` bundles a group of files together in a single archive. Unlike tar, however, cpio defaults to writing its archive to standard output: the output stream of cpio must be redirected to whatever device or file you want to save the data on.

The second major difference between tar and cpio is that tar takes the names of the files to be saved on the command line, while cpio reads the list from standard input. Thus, to save the files in /u/simsong with cpio, it is first necessary to get a list of them. The find command constructs this list nicely, as in this example:

```
$ find ./u/simsong -print |
 cpio -oBv > /dev/fd0
```

The -o option to cpio indicates that you are "outputting" data to an archive. The B option causes output to be blocked with 5 Kbytes to the record, which dramatically improves efficiency on devices like tapes and disks. The -v option lets you see what's going on, just as it does with tar.

If you were planning to restore these files on some other machine, you would also want to use the c option (i.e., cpio -ocBv), which makes cpio write ASCII headers, for portability.

To read the archive back, use cpio with the -i option:

```
cpio -iv < /dev/fd0
```

By default, cpio will read all the files in the archive. To restrict the restore to a single file, specify its name on the command line, as with tar.

Moving Files with cpio

Let's see what it's like to move the files between the /u/simsong and /u/joconnor directories using cpio. Consider the following example:

```
$ cd /u/simsong
$ find . -print | cpio -ovB
 >/dev/fd0
$
$ cd /u/joconnor
$ cpio -idmvB </dev/fd0
.
Mail
Mail/mbox
article
33 blocks
```

The -d option, used on the input side, tells cpio to create subdirectories if it needs to. In this example, cpio would have issued an error message if we hadn't used the -d switch.

The -m switch instructs cpio to preserve the modification time of each file.

Since our intention is simply to move files from one directory to another, we can use cpio's -p switch (for "pass") and forget about the floppy disk altogether:

```
$ cd /u/simsong
$ find . -print | cpio -pdmvl
 /u/joconnor
/u/joconnor/article
/u/joconnor/Mail/mbox
33 blocks
```

Again, this example assumes that the person issuing the cpio command has write access to /u/joconnor.

Because cpio reads the list of files to save from standard input, it allows greater flexibility in actually choosing these files. Using the find command, it is possible to select all the files in a file system that belong to a given user, are larger than a particular size, or haven't been accessed for more than a given number of days.

For example, you can use the following cpio command to find all the files owned by user simsong and save them on a floppy disk:

```
$find .-user simsong -print |
 cpio -oB >/dev/fd0
```

Of course, this could also be done with tar, using the command:

```
$tar cf 'find .-user simsong
 -print'
```

However, the Unix backquote notation fails when the command contained within the backquotes expands to more than 4096 characters.

It should also be noted that there are some versions of tar (i.e., the System V and Xenix versions) that will not archive or restore empty directories or nonregular files (e.g., the block-special and character-special files in the /dev directory or the FIFO file in /usr/spool/lp). So, at least on these Unix systems, cpio will be able to completely back up your root file system and your /usr/spool file system (assuming you have one), while tar will not.

Explore the Options

The Unix toolkit of utilities frequently provides you with more than one way to get a job done. This is certainly the case when the job at hand is moving files or backing up your system. As a newcomer to Unix, you might want to take the time to use both tar and cpio for a while and explore their options (see the summary in the sidebar on page 90) as you archive files and move them around in your system. With some practice, you'll find the backup tool that suits the occasion and feels comfortable to use. ■

Simson L. Garfinkel is a freelance writer and computer consultant living in Cambridge, MA.

Next month in *MIPS*...

Since we last tested RISC workstations based on Motorola's 88000, faster versions of the processor have hit the market. Next month we'll see how much of a boost the increased clock speed gives to new systems from Opus and Everex and to Tektronix's RP88, an 88000 coprocessor board for the Macintosh. We'll also report on how these new systems compare to 486 systems.

Also featured next month are new 80486 systems from Olivetti and Cheetah, the former enhanced with the EISA bus and the latter with a new memory interface. A review of 386SX computers rounds out our system coverage.

On the software scene, *MIPS* editors examine typical uses of computer-aided software engineering (CASE), test several CASE applications under DOS and Windows and on the Mac, and explore the likely future of CASE. Other articles include a comparison of hard disk controllers, a review of the QNX operating system, and more.