What's GNU, Part Six: tar

By Jerry Peek

This month — in the penultimate article of a series on new features added to utilities by GNU hackers and others — let's look at changes to the somewhat-under-named "tape archiver," or *tar*. *tar* handles a lot more than tapes: It packs files and their metadata (permissions, owner, links and more) into an archive format that can be compressed and transferred across a network, fed through a pipe to another *tar* running in a different directory, stored on a public server for people to download — and yes, written to a tape.

Let's look at GNU *tar* version *1.13.93* from the *Debian stable* distribution.

Keys Versus Options

Some of the original *Unix* utilities didn't accept options starting with a dash (`-`). Instead, their first argument was one or more *key* characters, like options without the dashes. One of these utilities was *dump* and another was *tar*. Some keys had corresponding arguments — a filename or the blocking factor, for instance — and the arguments had to come in the same order as their keys.

To learn all about GNU *tar*, get a *gallon* of coffee and read its *info* file (type *info tar*)

For example, to add a *tar* file on the no-rewind tape device */dev/nrst8* and use a blocking factor of 20, you'd use the add flag `r`, the file flag `f`, the blocking factor flag `b`, then the arguments for the `f` and `b` keys, respectively, and finally the filenames to add. You'd type:

```
$ tar rfb /dev/nrst8 20 dir1 dir2
```

GNU *tar* still accepts those old-time flags, but it also understands normal options starting with single dashes and GNU-style options starting with two dashes. So, now you can type:

```
$ tar -r -f /dev/nrst8 -\
b 20 dir1 dir2
```

tar Archive Formats

Like a lot of early Unix utilities that ran on small machines with (now) tiny disks, early versions of *tar* were limited. For instance, the *Version 7 tar* limited filename length to 100 characters. This wasn't just a limitation of the utility; it was designed into the format of a *tar* archive.

Since then, various versions have added additional formats. The "Formats" section of the GNU *tar info* page shows that it can handle five formats, including two different GNU formats: one pre-*version 1.12*, the other after. To top that, version 1.13.93 creates archives in the (new) GNU format, but later versions use the *POSIX* format instead.

However, even if you have the same version of *tar* described here, don't assume that yours renders the same format. The default format is chosen at compile time. You can find the default format by reading the end of the output from `tar --help`. Luckily, GNU *tar* reads five different input formats automatically (and can also handle compressed archives, as you'll see in the next section).

If you're trying to make portable archives, the best format might be POSIX. But your guess is as good as any! Just be aware of the differences, and, if someone else will be opening your archive, consider including a *README* about the format. There's advice in the *tar info* file section titled "Making 'tar' Archives More Portable."

Compressed Archives

Compression — with a program like *gzip* — typically reduces the size of a file. If you're saving a *tar* archive to disk or are sending it across a slow network, compressing it can make sense. (Compressing archives you write to tape, however, can be a bad idea. Because of the way compression algorithms work, a single bad block in the tape may make the whole archive unrecoverable.)

Figure One has an example. The first command packs a directory into a *tar* file. The second command tries to compress the archive with *gzip –v*, which indicates the reduction (`83.7%`) on the standard error after compression has finished. (Feeding the file to *gzip*'s standard input with the shell's `<` operator keeps *gzip* from overwriting the uncompressed file; also, saving *gzip* 's standard output in a different file makes it easy to compare the original and compressed *tar* files.) Finally, *du* shows the size savings: the original directory took 80 kilbytes (taking disk block size into account), the uncompressed tar file took 64 KB, and the compressed *tar* file takes 12 KB.

If you're running out of disk space, consider *tar*ing, compressing, and deleting your unused files.

Compressing archives is so common that GNU *tar* can run *gzip* — and other compression utilities, too — as part of the archiving operation. To create or extract a *gzip*-ped archive, add the `z` flag to the list of *tar* flags in its first argument. For example, `tar czf archive.tar.gz` creates a *gzip*-ped archive automatically. (If you like options starting with dashes, use `-z`, `--gzip`, `--gunzip`, or `--ungzip`, as appropriate.)

GNU *tar* also supports the older *compress* format. The `Z` (uppercase "z") flag handles that. If you have an old compressed archive (often with an uppercase `Z` in its name), but you don't have the *uncompress* utility, don't panic: use a `z` (lowercase) flag instead. *tar* will feed the archive to *gzip*, which can detect and handle the old *compress* format with no problem.

The *bzip2* compression format often (though not always) does more compression than *gzip*. In GNU *tar* 1.13, the `j` flag specifies *bzip2*. Unfortunately, some other versions use `I` (uppercase "i") instead. The long option `--bzip2` is more portable.

You can also specify an arbitrary compression program with the option `--use-compress-program=/path/to/program`. The program must accept the `-d` ("decompress") option; if it doesn't, create a shell script front-end to it.

Pathname Problems Solved

The original *tar* would restore a file to exactly the pathname it was stored with. In other words, if you archived the file */etc/somefile*, the absolute pathname (with the leading slash) meant that *somefile* could only be restored to */etc/*. If you didn't want that, you had a couple of ugly choices. You could run *tar* under *chroot* to temporarily change the location of the root directory while extracting the files. (You'd also need to make a copy of the *tar* binary, plus any shared libraries and other tools, and the tape device, under this temporary root.) Or you could copy the archive into a file, then use a binary editor to change the stored pathnames. (There were probably other ways, too.)

By default, GNU *tar* strips the leading slash (`/`) from pathnames when writing and reading archives. To restore an archive into the root directory, just type `cd /` before you run GNU *tar*. To make GNU *tar* work like other implementations of the utility *tar*, use the option `P`, `-P`, or `--absolute-names`.

FIGURE ONE: Testing the compression of a *tar* archive

```
$ tar cf /tmp/sortcol.tar 0602_gnus5_sort
$ gzip -v < /tmp/sortcol.tar > /tmp/sortcol.tar.gz
  83.7%
$ du 0602_gnus5_sort /tmp/sortcol.tar*
80       0602_gnus5_sort
64       /tmp/sortcol.tar
12       /tmp/sortcol.tar.gz
```

Avoiding Overwriting

Old *tar* extracted an archive unconditionally: if a file existed, it would be overwritten. The `w` option made *tar* ask before extracting each file, but this was a pain if you were extracting a lot of files and only wanted to prevent overwriting.

GNU tar gives you some better choices. The options `k`, `-k` and `--keep-old-files` don't overwrite existing files.

The options `-K` *pathname* and `--starting-file` *pathname* tell *tar* to begin reading the archive from the start but not to extract any files until it finds *pathname*. (To see the order of files in the archive, use `tar t`, `-t`, or `--list`.)

If someone else will be opening your archive, consider including a *README* about the format

The `--backup` option makes a backup of files before overwriting them. You can add one of the types of backup: `simple`, `existing`, or `numbered`. (An earlier article in this series, from August 2005, explains GNU backup schemes in detail. It's available online at http://www.linux-mag.com/2005-08/power_01.html.)

As as example, `tar -x --backup=simple somedir` extracts the directory `somedir`. If it's going to extract a file named *foo* and that file already exists, *tar* renames *foo* to *foo~* before extracting *foo* from the archive.

You can change the default suffix from ~ to something else with `--suffix='X'`, where *X* is the suffix you want to use.

Choosing Files

Original *tar* would extract the files or directories you gave as command-line arguments. If you named a directory, it would extract all entries from the directory (though you could also specify individual files from a directory, like *dir/file1 dir/file2*). Choosing certain types of files — for instance, all filenames ending in *.c* — wasn't trivial. Original *tar* couldn't match wildcards like `*.c` against the contents of an archive. You could use a fairly ugly hack, though: filter a listing of the archive's contents through *grep*, then use the *grep* output as command-line arguments to another *tar*.

For example, if your tape was mounted on the default tape drive (so you didn't need the `f` option), you could extract all *.c* files this way:

```
tar xv `tar t | grep '\.c$'`
```

The first *tar*, run by command substitution (inside the backquotes), yields a complete table of contents; the *grep* yields all names ending with `.c`. The next *tar* would extract those files and list their names as it did. (If there were too many *.c* files, the command line could become too long; you'd need to use another method.)

Some later versions of *tar* accepted an `X` option with an argument of a filename containing a list of pathnames to *exclude* from the archive. The similar `I` option (called `T` in GNU *tar*) requires a list of files and directories to *include* — in addition to any files named on the command line. This "include" listing gets around the problem of naming too many files on the command line (command-lines had length limits on older systems).

GNU tar has the options `d`, `-d`, `--diff`, and `--compare`. These options compare an archived file to the current version on disk

GNU *tar* supports inclusionary and exclusionary lists. Furthermore, options like `--wildcards` and `--no-wildcards` control wildcard matching in the exclude file, and `--exclude=`*`pattern`* lets you specify exclusionary wildcards on the command-line.

Also, check out options like `--after-date` and `--newer` to choose files by date.

Which Occurrence?

In the days before big hard disks, archives were usually written to reels of tape. A tape could be appended to, so a tape might have multiple copies of the same file. As an example, the following series of commands add two versions of *foo* to the same (disk) archive:

```
$ echo test file > foo
$ tar cf foo.tar foo
$ echo more data >> foo
$ tar rf foo.tar foo
$ tar tvf foo.tar
-rw ... 10 2005-12-04 17:08:34 foo
-rw ... 20 2005-12-04 17:08:57 foo
```

If you tell *tar* to extract a file — for instance, `tar x` *`somefile`* — it reads the whole archive, extracting every occurrence of *somefile*. In that case, you'll end up with the last occurrence of *somefile*.

On a tape drive, you could choose which occurrence by using utilities like *mt* and *dd*. You could also use the dirty hack of a command like `tar xv somefile`, watching the verbose output until *tar* had extracted the occurrence you wanted, then killing *tar* with Control-C.

GNU *tar* has made this a lot easier. The option `—occurrence=`*`num`* extracts the *`num`th* occurrence of a file. Or, by default, `—occurrence` extracts the first occurrence.

Comparing Disk Files to Archive Files

With the original *tar*, if you wondered whether a disk file was different than an archived file, you'd need to extract the archive somewhere safe and then run *diff*.

GNU *tar* has the options `d`, `-d`, `--diff`, and `--compare`. These options compare an archived file to the current version on disk, and can tell you whether a file is longer or shorter, whether the file's contents have changed (it's the same size but the contents are different), and if its last-modification time has changed.

In the next example, two of the files in the archive *chk.tar* are different than the disk files in the current directory:

```
$ tar df chk.tar
chk: Mod time differs
chk: Size differs
chk.1: Mod time differs
chk.1: Contents differ
```

If you need to know more, you can use the *tar* options `O`, `-O` (both an uppercase letter "o") or `--to-stdout` to extract an archived file to a pipe, then compare it with *diff*. For instance, here line 5 of the archived file *chk* ends with `$subj` but the disk file ends with `$subject`:

```
$ tar xOf chk.tar chk | diff - chk
5c5
< echo "$file $size $subj"
—-
> echo "$file $size $subject"
```

There's Much More

For a quick summary of *tar* options and operation, type `tar --help`. To learn all about GNU *tar*, get a *gallon* of coffee and read its *info* file (type *info tar*). As you'll see here, there's much more than anyone could cover in three pages.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.