Your Short Cut to Knowledge

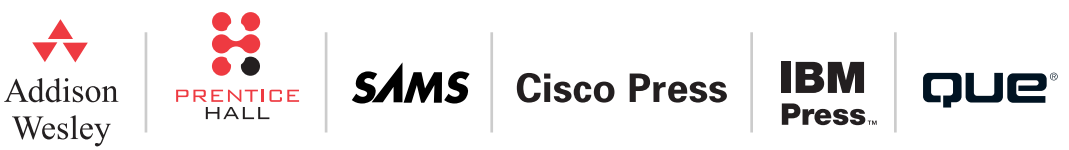
The following is an excerpt from a Short Cut published by one of the Pearson Education imprints.

Short Cuts are short, concise, PDF documents designed specifically for busy technical professionals like you.

We've provided this excerpt to help you review the product before you purchase. Please note, the hyperlinks contained within this excerpt have been deactivated.

Tap into learning—NOW!

Visit **www.informit.com/shortcuts** for a complete list of Short Cuts.



CHAPTER 31

Making and Melting tarballs

This is important. We're talking about ***tar***, that sticky software that bonds files together into one big ball. It's one of the most important programs you'll learn to use from the Linux CLI.

Why? Because most free/open source software makes use of tar in distributing itself, that's why. So unless you know—at a minimum—how to untar a tarball, you will remain cut off from a vital flow, nay, from the heartbeat of free/open source software.

That's why tar is a necessary skill for you to learn. That, and the fact it's required to earn your CLI merit-badge, grasshopper.

Introduction to tar

The name comes from Tape Archive. It's no wonder it's so widely used by the open source community, especially in the distribution of software. ***Tar*** allows developers to create a "tarball" containing everything needed, including directory structures, to build and install complex applications from their source code.

Note that tar does not actually do any compression. It simply sticks all the files together into a single tarball, or archive. Normally the tarball itself is then compressed by another program. But tar is smart enough to handle both the compression and the decompression—using those other programs—if you tell it to do so.

The help found by entering ***man tar*** is a bit overwhelming for noobies. Not for you, of course, but for some. In fact, the first item in the Bugs section of the man pages for ***tar*** on my system is a recommendation not to rely on them, but to use the info command instead. I kid you not. Here it is:

> *The GNU folks, in general, abhor man pages, and create info documents instead. The maintainer of tar falls into this category. This man page is neither complete, nor current, and was included in the Debian Linux packaging*

of tar entirely to reduce the frequency with which the lack of a man page gets reported as a bug in our defect tracking system.

Keep in mind that you won't get all there is to grok from ***man anything*** the first time you use it. But as you learn more about "anything," you can always go back to ***man anything*** and learn a bit more. As always, we'll follow the KISS principle with tar and try to "Keep It Simple, Stupid."

Decompressing tarballs

Note from a reader: If typing those ludicrously long file names gets you frustrated, be sure to let the bash shell help you with that. Type just the first few characters of the file name (like "kool" in the example above) and then press Tab. It will attempt to autocomplete, pausing for you to add more characters to the name if needed. That's much easier than typing long file names with lots of punctuation marks in them.

My approach is to download everything to a directory reserved for the purpose, then to decompress the tarball in that directory. That usually results in a new subdirectory with a name similar to the downloaded file itself.

For example, if I downloaded *kool-new-app-1.0-3.tgz* it would appear in my download directory by that same name. When I decompressed the tarball, then normally—depending entirely on how the tarball was built—a new sub-directory would appear with the name of *kool-new-app-1.0-3*.

Sometimes contrarians will put the tarball together in such a way that it doesn't create a new subdirectory. Then it's a mess to untangle which files were part of the tarball and which were pre-existing. In those cases, I usually create a subdirectory myself, and then decompress the tarball inside of it.

Let's look at three different tar commands that might be used on a downloaded tarball, depending on whether or not it's compressed, and if it is, the compression scheme used.

In each case, the arguments you provide in the command line tell tar what to do, what compression format to use if any, and the file name of the tarball. For example, to decompress and untar the example above, you would enter:

```
linux~>tar xzf kool-new-app-1.0-3.tgz
```


The ***x*** tells tar to extract data rather than to create a tarball. The ***z*** tells tar that the archive was compressed in the gzip format. The ***f*** says "Listen up, tar: what follows is the File name of the archive I want you to extract and decompress." That's followed, of course, by the file name.

You can normally count on the last three letters of the downloaded tarball to tell you whether or not it has been compressed, and if it has been, the format used to do so. A simple ".tar" extension indicates no compression at all. To untar it, you would simply type:

```
linux~>tar xf filename
```

Just as in the first example, ***x*** tells tar to extract data from the tarball and the ***f*** tells tar that the filename to work on follows immediately. Compression is not mentioned since it's not needed.

If the tarball filename ends in ".bz2", it was compressed with ***bzip2***. The command you would give tar in that case is:

```
linux~>tar xjf filename
```

The ***x*** and ***f*** mean the same as always, and the ***j*** says "This tarball has been compressed using ***bzip2***.

Creating a tar Archive

Naturally, ***tar*** is used to create archives as well as to untar/decompress them. Let's take a look at how easy it is to use. One thing that it's a good idea to back up from time to time is your home directory. We'll use that for our example. Here's how to make a tarball containing everything in your home directory, including any hidden files and sub-directories that may live there.

Now, we don't want to be like the contrarians I mentioned earlier, those rogues who create tarballs that spill their contents into the current directory when they are untarred.

To avoid that, we'll specify the full path name for the name of the archive and for the data to be archived.

First, make sure you are in your home directory. If you're not, enter cd all by itself to get there. Then enter this tar command:

```
linux~>tar cjf ~/backup-home.tar.bz2 .
```

Note: Don't forget the blank and the . after the filename. That's what tells tar where to start archiving things.

In the example above, the "cjf" tells tar to create an archive using bzip2 for compression and to name it "backup-home.tar.bz2".

If you're paranoid about backups, it might not be a bad idea to make sure your newly created tarball contains everything that it should. Easy enough to do. Enter this command:

```
linux~>tar tjf backup-home.tar.bz2
```

The ***t*** in ***tjf*** tells ***tar*** to list the contents of the archive. Since the archive is compressed, you need to tell ***tar*** that too, so he can decompress it to get the file names.

But What If You Don't Want to Archive It All?

What if you only want to select specific files to be archived? That's pretty easy too. Say that you want to get all the digital photos you've taken together in one spot so you can burn a CD to store them on.

Assuming that the images are in the JPG format, you can create an archive containing only your JPG files like this:

```
linux~>tar cjf digital-pix.tar.bz2 *.jpg
```

Note that the "." from the first example has been changed to "*.jpg". That's the difference between getting everything and in getting selective files in the archive.

Alright, there you have it. Another important tool in your kit to allow you to create or untar archives whenever you have a need.