gzip, bzip2 and tar

EXPERT PACKING

A short command is all it takes to pack your data or extract it from an archive.

BY HEIKE JURZIK

Archiving provides many benefits: packed and compressed files occupy less space on your disk and require less bandwidth on the Internet. Linux has both GUI-based programs, such as File Roller or Ark, and command-line tools for creating and unpacking various archive types. This article examines some shell tools for archiving files and demonstrates the kind of expert packing that clever combinations of Linux commands offer the command line user.

Nicely Packed with "gzip"

The gzip (GNU Zip) program is the default packer on Linux. Gzip compresses simple files, but it does not create complete directory archives. In its simplest form, the gzip command looks like this:

```
gzip file
```

gzip replaces the original file with a compressed version, adding the *.gz* extension. File properties such as access and modification time stamps are retained by the packing process. If you prefer to use a different extension, you can set the *-S* (suffix) flag to specify your own instead. For example, the command

```
gzip -S .z image.bmp
```

creates a compressed file titled *image.bmp.z*.

The size of the compressed file depends on the distribution of identical strings in the original file. gzip will compress a file more efficiently if it contains patterns that repeat frequently. gzip does not do a good job of compressing files that use compressed formats for the original, such as MP3 or JPEG. Listing 1 shows the difference in performance between a bitmap and a JPEG file.

You can additionally specify a compression factor between 1 and 9 to influence the compression: *gzip -1* is quicker and *gzip -9* is slower but has a far higher compression rate. The default is *-6*. If you prefer to change the default, you can do so by setting the gzip environment variable in the *~/.bashrc* file

```
export GZIP="-9"
```

A gzip file can be unpacked using either gunzip or *gzip -d*. If the tool discovers a file of the same name in the working directory, it prompts you to make sure that you know you are overwriting this file:

```
$ gunzip screenie.jpg.gz
gunzip: screenie.jpg ➥
```

Command Line

Although GUIs such as KDE or Gnome are useful for various tasks, if you intend to get the most out of your Linux machine, you will need to revert to the good old command line from time to time. Apart from that, you will probably be confronted with various scenarios where some working knowledge will be extremely useful in finding your way through the command line jungle.

Listing 1: Compression Compared

```
01 $ <B>ls -l<B>
02 -rw-r--r--  1 daisy daisy
   2313894 Sep  3 22:47 screenie.
   bmp
03 -rw-r--r--  1 daisy daisy
   169862 Sep  5 12:41 screenie.
   jpg
04 $ <B>gzip screenie*<B>
05 $ <B>ls -l<B>
06 -rw-r--r--  1 daisy daisy
   9547 Sep  3 22:47 screenie.
   bmp.gz
07 -rw-r--r--  1 daisy daisy
   130524 Sep  5 12:41 screenie.
   jpg.gz
```

```
already exists; do you wish↵
 to overwrite (y or n)?
```

If you say [n] at this point, gzip aborts the operation. If you object to the safety check, you can disable the check by setting the *-f* (for "force") option. The parameter has an additional side effect: by default, gzip refuses to compress **Symlinks**. But if you set the *-f* parameter before pointing gzip at a symlink, the tool will compress the file the link points to and assign the name of the symlink (plus the normal extension). When you unpack the file, this does not give you a symlink but a normal file (Figure 1).

There is no need to unpack compressed text files, such as the HOWTOs below */usr/share/doc/*, before using the less or more pager to view them. Running gzip with the *-c* option (which gives you output on **stdout**), and piping the output to a pager, gives you a much quicker approach, for example:

```
gzip -dc /usr/share/doc/↵
iptables/README.Debian.gz | less
```

You can abbreviate this even further using *zless file.gz*: This command is in fact a short script that has a similar result but with less typing.

Newer, Quicker, Better: bzip2

The bzip2 tool uses a different compression algorithm, and thus it compresses most files far more effectively. Listing 2 shows gzip and bzip2 in comparison. Additionally, bzip2 has a recovery mode. On compressing, the tool splits files into individual blocks; if one file is damaged, it may still be possible to rescue the other, undamaged files. To do this, run bzip2recover to unpack the undamaged elements.

Apart from a few minor differences, most bzip2 options are like their gzip counterparts. Again, you simply specify a filename to compress that file:

```
bzip2 file
```

The compressed file will have a *.bz2* extension after the event, and it keeps the original file properties – just like compressing with gzip. In contrast to gzip, bzip2 has an option for creating a copy of the original. To do this, simply specify the *-k* (for "keep") parameter:

```
bzip2 -k file
```

(To do the same thing with gzip, you could output the compressed file to standard output, and then redirect the output to a new file: *gzip -c file > file.gz*. However, the compressed version of the file will not keep the properties of the

Listing 2: gzip and bzip2 Compared

```
01 $ <B>ls -l bild.bmp*<B>
02 -rw-r--r--  1 daisy daisy
   2313894 Sep  5 13:35 image.bmp
03 -rw-r--r--  1 daisy daisy
   2534 Sep  5 13:35 image.bmp.
   bz2
04 -rw-r--r--  1 daisy daisy
   9547 Sep  5 13:35 image.bmp.gz
```

original file in this case.)

Just like with gzip, the *-1* through *-9* (default) parameters tell bzip2 what level of compression to use. To change the default, again set the *BZIP2* environment variable by adding the following to *~/.bashrc*

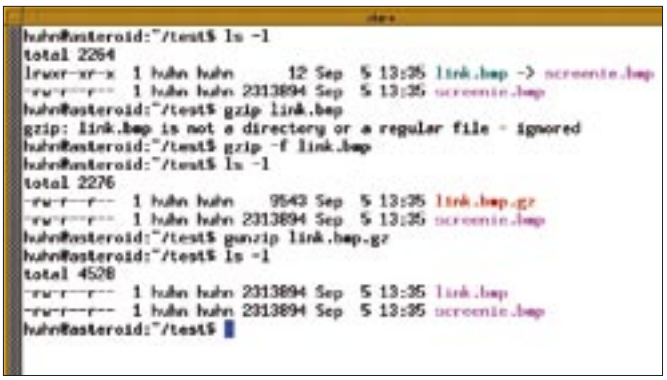

Figure 1: Be careful how you compress symlinks. You can use the -f parameter to force this operation, but unpacking the file will not give you a symlink.

```
export BZIP2="-6"
```

To unpack a compressed file, set bzip2's *-d* flag, or use the special bunzip2 unpacking tool. Like gzip, bzip2 gives you some protection against inadvertently overwriting existing files, but in contrast to gzip, the tool does not give you freedom of choice. Instead it displays the following message and quits the operation:

```
$ bunzip2 peggy.jpg.bz2
bunzip2: Output file ➥
peggy.jpg already exists.
```

The *-f* parameter disables this behavior.

Into the Archive

The tar program stores multiple files in an archive. The program's name (*tar* is short for "tape archiver") indicates its original use for tape archive management. But tar can do more than just concatenate files; it has options for compressing archives directly using gzip or bzip2 – and this is what gives you the common *tar.gz* and *tar.bz2* archive names.

To put multiple files in an archive, enter

```
tar -cvf archive.tar file1 file2
```

for example. The *-c*, *-v*, and *-f* options, which are in shorthand in this command, tell tar to create the archive ("c" for "create"), tell us what is going on behind the scenes ("v" for "verbose"), and interpret the first argument (*archiv.tar*) as the archive name ("f" for "file").

If you notice that you have forgotten a file at a later date, you don't have to redo the whole archive; instead you can simply append the file using the *-r* option:

```
tar -rf archive.tar file
```

Unpack an archive with the *-x* ("x" for "extract") option:

```
tar -xvf archive.tar
```

To ensure than tar does not overwrite existing files with content from the archive, it is a good idea to unpack the archive in a temporary directory, or use *-t* instead of *-x* for a trial run, just to see which files and directories the archive contains.

You can handle complete directories and subdirectories with the same approach – instead of specifying individual filenames, just give tar the name of the folder:

```
$ tar -cvf archive.tar folder/
test/
test/screenie.bmp
test/link.bmp
test/new/
test/new/screenie.jpg
test/new/new/
...
```

It can make sense to exclude some directories from the archive, especially if you are using tar for backups. The *--exclude* parameter gives you this ability. The *--rsh-command* parameter is also useful in this context, as you can tell tar to use SSH to send the backup to another machine. The complete command looks like this:

```
tar -cvf user@host:/scratch➥
/tmp/backup_$➥
(date '+%Y_%m_%d').tar ➥
--rsh-command=/usr/bin/ssh ➥
--exclude=/proc /
```

This fairly lengthy command gives the following instructions: use SSH to create a backup copy in the */scratch/tmp/* directory on the *host* machine, basing the archive name on the *backup_* prefix, the current date, and the *.tar* extension (for example, *backup_2005_10_05.tar*). This backup will include all directories from the root */*, but without the */proc* directory, which does not contain any real data.

Altogether Now!

As I mentioned previously, tar has a number of parameters for creating gzip or bzip2 compressed archives all at once. The syntax for the gzip variant of the tar command is:

```
tar -cvzf archive.tar.gz file(s)
```

If you prefer to use bzip2 instead of gzip with tar, replace the *-z* option in the preceding command with *-j*:

```
tar -cvjf ➥
archive.tar.bz2 file(s)
```

You can set the *-z* and *-j* parameters appropriately when unpacking the archive to save an extra step; for example, instead of

```
bunzip2 archive.tar.bz2
tar -xvf archive.tar
```

just say

```
tar -xvjf archive.tar.bz2
```

to invoke the *-j* option for unpacking a bzip2 archive. ■

THE AUTHOR

Heike Jurzik studied German, Computer Science and English at the University of Cologne, Germany. She discovered Linux in 1996 and has been fascinated with the scope of the Linux command line ever since. In her leisure time you might find Heike hanging out at Irish folk sessions or visiting Ireland.