Lab: File Processing Utilities: Compressing Data

In this lab, you will learn how to put a group of files together into a single archive. You will also learn how to compress an archive file using various compression methods.

Creating an archive

Let's create a backup for all the bash scripts in the [/home/elliot] directory. As the [root] user, create a directory named [backup] in [/root]:

```
root@ubuntu-linux:~# mkdir /root/backup
```

Now let's create an archive named [scripts.tar] in [/root/backup] for all the bash scripts in [/home/elliot]. To do that, we first change to the [/home/elliot] directory:

```
root@ubuntu-linux:~# cd /home/elliot
root@ubuntu-linux:/home/elliot#
```

Then we run the command:

```
root@ubuntu-linux:/home/elliot# tar -cf /root/backup/scripts.tar *.sh
```

This will create the archive file [scripts.tar] in [/root/backup], and there will be no command output:

```
root@ubuntu-linux:/home/elliot# ls -l /root/backup/scripts.tar
-rw-r--r-- 1 root root 20480 Nov 1 23:12 /root/backup/scripts.tar
```

We could have also added the verbose option [-v] to see the files that are being archived:

```
root@ubuntu-linux:/home/elliot# tar -cvf /root/backup/scripts.tar *.sh
3x10.sh
detect.sh
empty.sh
filetype.sh
fun1.sh
game.sh
hello20.sh
hello2.sh
hello3.sh
hello.sh
math.sh
mydate.sh
noweb.sh
numbers.sh
rename.sh
size2.sh
size3.sh
size.sh
```

Viewing archive contents

To view the contents of the archive [scripts.tar] that we just created, you can run the command:

```
root@ubuntu-linux:/home/elliot# tar -tf /root/backup/scripts.tar
3x10.sh
detect.sh
empty.sh
filetype.sh
fun1.sh
game.sh
hello20.sh
hello2.sh
hello3.sh
hello.sh
math.sh
mydate.sh
noweb.sh
numbers.sh
rename.sh
size2.sh
size3.sh
size.sh
```

As you can see, it listed all the files in the [scripts.tar] archive.

Extracting archive files

You may also want to extract files from an archive. To demonstrate, let's create a directory named [myscripts] in [/root]:

```
root@ubuntu-linux:/# mkdir /root/myscripts
```

To extract all the files in the [scripts.tar] archive to the [/root/myscripts] directory, you can run the following command:

```
root@ubuntu-linux:/# tar -xf /root/backup/scripts.tar -C /root/myscripts
```

The [-x] option is the shorthand notation of [--extract], which extracts the files from the archive. We also used the [-C] option, which basically changes to the [/root/myscripts] directory before carrying out any operation, and thus the files are extracted to [/root/myscripts] instead of the current directory.

Now let's verify that the files were indeed extracted to the [/root/myscripts] directory:

```
root@ubuntu-linux:/# ls /root/myscripts
3x10.sh
empty.sh
fun1.sh
hello20.sh
hello3.sh
math.sh
noweb.sh
rename.sh
size3.sh
```

```
detect.sh
filetype.sh
game.sh
hello2.sh
hello.sh
mydate.sh
numbers.sh
size2.sh
size.sh
```

And sure enough, we see all our bash scripts in the [/root/myscripts] directory!

Compressing with gzip

To compress the [scripts.tar] archive into a [gzip]-compressed archive named [scripts.tar.gz], you first need to change to the [/root/backup] directory and then run the following command:

```
root@ubuntu-linux:~/backup# tar -czf scripts.tar.gz scripts.tar
```

Now if you list the contents of the [backup] directory, you will see the newly created [gzip]-compressed archive [scripts.tar.gz]:

```
root@ubuntu-linux:~/backup# ls
scripts.tar scripts.tar.gz
```

The magic happened by using the [-z] option, which compressed the archive with the [gzip] compression method. And that's it! Notice how it's very similar to creating an archive: we just added the [-z] option -- that's the only difference.

Now let's run the [file] command on both archives:

```
root@ubuntu-linux:~/backup# file scripts.tar
scripts.tar: POSIX tar archive (GNU)
root@ubuntu-linux:~/backup# file scripts.tar.gz
scripts.tar.gz: gzip compressed data, last modified: Sat Nov 2 22:13:44 2019,
from Unix
```

As you can see, the [file] command detects the type of both archives. Now let's compare the size (in bytes) of both archives:

```
root@ubuntu-linux:~/backup# du -b scripts.tar scripts.tar.gz
20480 scripts.tar
1479 scripts.tar.gz
```

The compressed archive [scripts.tar.gz] is way smaller in size as we expected compared to the uncompressed archive [scripts.tar]. If you want to extract the files in the compressed archive [scripts.tar.gz] to [/root/myscripts], you can run:

```
root@ubuntu-linux:~/backup# tar -xf scripts.tar.gz -C /root/myscripts
```

Notice it is exactly the same as the way that you would extract the contents of an uncompressed archive.

Compressing with bzip2

To compress the [scripts.tar] archive into a [bzip2]-compressed archive named [scripts.tar.bz2], you first need to change to the [/root/backup] directory and then run the following command:

```
root@ubuntu-linux:~/backup# tar -cjf scripts.tar.bz2 scripts.tar
```

Now if you list the contents of the [backup] directory, you will see the newly created [bzip2]-compressed archive [scripts.tar.bz2]:

```
root@ubuntu-linux:~/backup# ls
scripts.tar scripts.tar.bz2 scripts.tar.gz
```

Let's run the [file] command on the [bzip2]-compressed archive [scripts.tar.bz2]:

```
root@ubuntu-linux:~/backup# file scripts.tar.bz2
scripts.tar.bz2: bzip2 compressed data, block size = 900k
```

It correctly detects the type of compression method used for the archive [scripts.tar.bz2]. Awesome -- now let's compare the size (in bytes) of the [gzip]-compressed archive [scripts.tar.gz] and the [bzip2]-compressed archive [scripts.tar.bz2]:

```
root@ubuntu-linux:~/backup# du -b scripts.tar.bz2 scripts.tar.gz
1369 scripts.tar.bz2
1479 scripts.tar.gz
```

Notice that the [bzip2]-compressed archive [scripts.tar.bz2] is smaller than the [gzip]-compressed archive [scripts.tar.gz]. If you want to extract the files in the compressed archive [scripts.tar.bz2] to [/root/myscripts], you can run:

```
root@ubuntu-linux:~/backup# tar -xf scripts.tar.bz2 -C /root/myscripts
```

Notice it is exactly the same as the way that you would extract the contents of a [gzip]-compressed archive.

Compressing with xz

The [xz] compression method is yet another popular compression method used on Linux. On average, [xz] compression does the best job out of all three compression methods in reducing (compressing) the file sizes.

You can compress an archive with [xz] compression by using the [-J] option with the [tar] command as follows:

```
tar -cJf compressed_name archive_name
```

Notice here we use the uppercase letter [J] with [xz] compression. So to compress the [scripts.tar] archive into an [xz]-compressed archive named [scripts.tar.xz], you first need to change to the [/root/backup] directory and then run the following command:

```
root@ubuntu-linux:~/backup# tar -cJf scripts.tar.xz scripts.tar
```

Now if you list the contents of the [backup] directory, you will see the newly created [xz]-compressed archive [scripts.tar.xz]:

```
root@ubuntu-linux:~/backup# ls
scripts.tar scripts.tar.bz2 scripts.tar.gz scripts.tar.xz
```

Let's run the file command on the [xz]-compressed archive [scripts.tar.xz]:

```
root@ubuntu-linux:~/backup# file scripts.tar.xz
scripts.tar.xz: XZ compressed data
```

It correctly detects the type of compression method used for the archive [scripts.tar.xz].

Measuring performance

You can use the [time] command to measure the time it takes a command (or a program) to finish executing. The general syntax for the [time] command is as follows:

```
time command_or_program
```

For example, to measure how long it takes for the [date] command to finish executing, you can run the following command:

```
root@ubuntu-linux:~# time date
Sun Nov 3 16:36:33 CST 2019

real 0m0.004s
user 0m0.003s
sys 0m0.000s
```

It just took four milliseconds to run the [date] command on my system; this is quite fast!

The [gzip] compression method is the fastest of all three compression methods; well, let's see if I am lying or telling the truth! Change to the [/root/backup] directory:

```
root@ubuntu-linux:~# cd /root/backup
root@ubuntu-linux:~/backup#
```

Now let's see how long it takes to create a [gzip]-compressed archive file for all the files in [/boot]:

```
root@ubuntu-linux:~/backup# time tar -czf boot.tar.gz /boot
real 0m4.717s
user 0m4.361s
sys 0m0.339s
```

On my system, it took [gzip] 4.717 seconds to run! Now let's measure the time it takes to create a [bzip2]-compressed archive of the same directory [/boot]:

```
root@ubuntu-linux:~/backup# time tar -cjf boot.tar.bz2 /boot
real 0m19.306s
user 0m18.809s
sys   0m0.359s
```

It took [bzip2] an enormous [19.306] seconds to run! You can see how [gzip] compression is much faster than [bzip2]. Now let's see the time it takes to create an [xz]-compressed archive of the same directory [/boot]:

```
root@ubuntu-linux:~/backup# time tar -cJf boot.tar.xz /boot
real 0m53.745s
```

```
user 0m52.679s
sys   0m0.873s
```

It almost took [xz] a full minute! We can conclude that [gzip] is definitely the fastest of all three compression methods we have discussed.

Finally, let's check the size (in bytes) of the three compressed archives:

```
root@ubuntu-linux:~/backup# du -b boot.*
97934386 boot.tar.bz2
98036178 boot.tar.gz
94452156 boot.tar.xz
```

As you can see, [xz] did the best job of compressing the files. [bzip2] claimed second place, and [gzip] came in last.

Knowledge check

For the following exercises, open up your Terminal and try to solve the following tasks:

1. Create a [gzip] archive named [var.tar.gz] in [/root] for all the files in [/var].
2. Create a [bzip2] archive named [tmp.tar.bz2] in [/root] for all the files in [/tmp].
3. Create an [xz] archive named [etc.tar.xz] in [/root] for all the files in [/etc].