Module 9: Archiving and Compression

Objectives

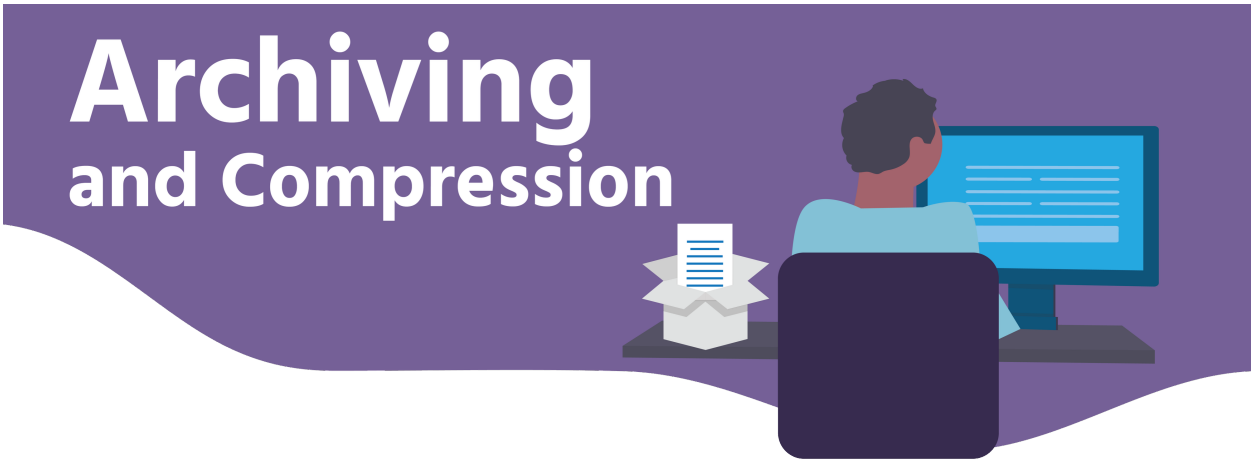
Upon completion of this module, you will be able to:

- Create, list, and extract archive files from the command line.
- Apply compression to reduce the size of files and archives.
- Differentiate between the concepts of archiving and file compression.
- Manage files and **directories** for archiving purposes within a user's home directory.

Key Knowledge Areas:

- Files, directories
  Section 9.1
- Archives
  Section 9.3
- Compression
  Section 9.2

Topics



9.1 Introduction

In this chapter, we discuss how to manage archive files at the command line. *File archiving* is used when one or more files need to be transmitted or stored as efficiently as possible. There are two fundamental aspects which this chapter explores:

- *Archiving*: Combines multiple files into one, which eliminates the overhead in individual files and makes the files easier to transmit.
- *Compression*: Makes the files smaller by removing redundant information.

Consider This

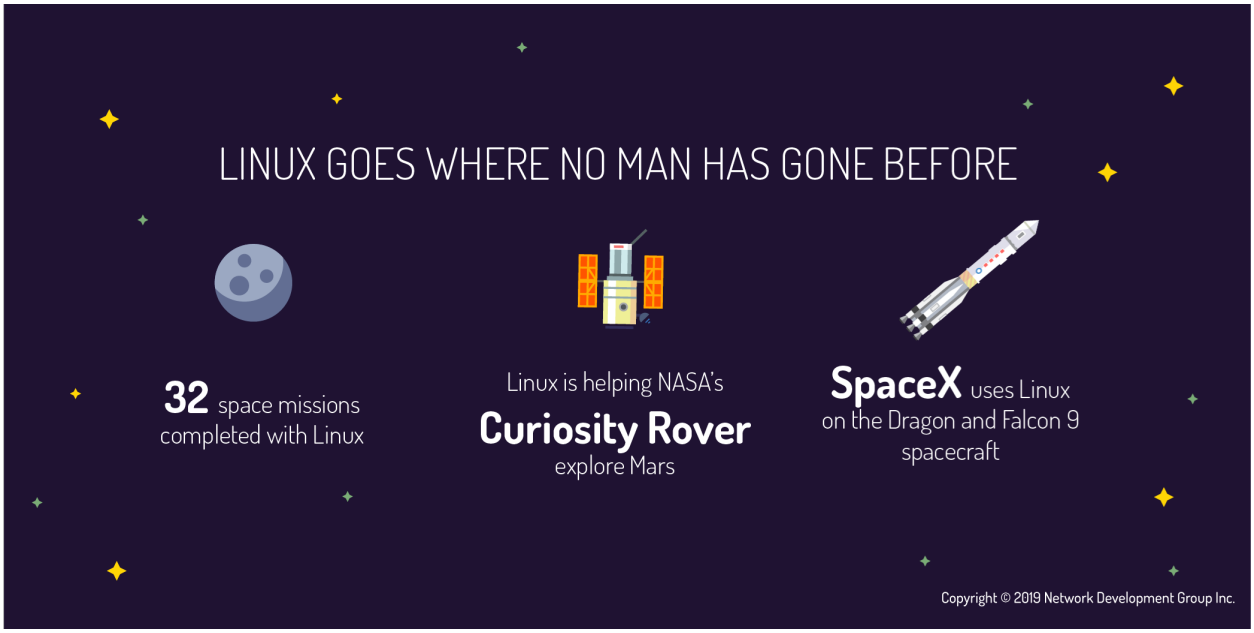
Files can be compressed individually, or multiple files can be combined into a single archive and then subsequently compressed. The latter is still referred to as archiving.

When an archive is *decompressed*, and one or more files are extracted, this is called *un-archiving*.

Even though disk space is relatively cheap, archiving and compression still have value:

- When making a large number of files available, such as the source code to an application or a collection of documents, it is easier for people to download a compressed archive than it is to download files individually.
- Log files have a habit of filling disks, so it is helpful to split them by date and compress older versions.
- When backing up directories, it is easier to keep them all in one archive than it is to version (update) each file.
- Some streaming devices such as tapes perform better if you're sending a stream of data rather than individual files.
- It can often be faster to compress a file before sending it to a tape drive or over a slower network and decompress it at the other end than it would be to send it uncompressed.

It is essential for Linux administrators to become familiar with the tools for archiving and compressing files.



9.2 Compressing Files

Compression reduces the amount of data needed to store or transmit a file while storing it in such a way that the file can be restored. A file with human-readable text might have frequently used words replaced by something smaller, or an image with a solid background might represent patches of that color by a code. The compressed version of the file is not typically viewed or utilized, instead, it is decompressed before use.

The *compression algorithm* is a procedure the computer uses to encode the original file, and as a result, make it smaller. Computer scientists research these algorithms and come up with better ones that can work faster or make the input file smaller.

When talking about compression, there are two types:

- *Lossless*: No information is removed from the file. Compressing a file and decompressing it leaves something identical to the original.
- *Lossy*: Information might be removed from the file. It is compressed in such a way that uncompressing a file will result in a file that is slightly different from the original. For instance, an image with two subtly different shades of green might be made smaller by treating those two shades as the same. Often, the eye can't pick out the difference anyway.

Generally, human eyes and ears don't notice slight imperfections in pictures and audio, especially as they are displayed on a monitor or played over speakers. Lossy compression often benefits media because it results in smaller file sizes and people can't tell the difference between the original and the version with the changed data. For things that must remain intact, such as documents, logs, and software, you need lossless compression.

Most image formats, such as GIF, PNG, and JPEG, implement some form of compression. JPEGs use lossy compression, while GIFs and PNGs are compressed but lossless. You can generally decide how much quality you want to preserve. A lower quality results in a smaller file, but after decompression, you may notice artifacts such as rough edges or discolorations. High quality will look much like the original image, but the file size will be closer to the original.

Compressing an already compressed file will not make it smaller. This fact is often forgotten when it comes to images since they are already stored in a compressed format. With lossless compression, this multiple compression is not a problem, but if you compress and decompress a file several times using a lossy algorithm, you will eventually have something that is unrecognizable.

Linux provides several tools to compress files; the most common is `gzip`. Here we show a file before and after compression:

```
sysadmin@localhost:~$ cd Documents
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 66540 Dec 20  2017 longfile.txt
sysadmin@localhost:~/Documents$ gzip longfile.txt
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 341 Dec 20  2017 longfile.txt.gz
```

In the preceding example, there is a file called `longfile.txt` that is `66540` bytes. The file is compressed by invoking the `gzip` command with the name of the file as the only argument. After that command completes, the original file is gone, and a compressed version with a file extension of `.gz` is left in its place. The file size is now `341` bytes.

The `gzip` command will provide this information, by using the `-l` option, as shown here:

```
sysadmin@localhost:~/Documents$ gzip -l longfile.txt.gz
         compressed        uncompressed  ratio uncompressed_name
                341               66540  99.5% longfile.txt
```

The compression ratio is given as `99.5%`, an impressive reduction helped by the repetitive information in the original file. Additionally, when the file is decompressed, it will be called `longfile.txt` again.

Compressed files can be restored to their original form using either the `gunzip`command or the `gzip -d` command. This process is called *decompression*. After `gunzip` does its work, the `longfile.txt` file is restored to its original size and file name.

```
sysadmin@localhost:~/Documents$ gunzip longfile.txt.gz
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 66540 Dec 20  2017 longfile.txt
```

Consider This

The `gunzip` command is just a script that calls `gzip` with the right parameters.
There are other commands that operate virtually identically to `gzip` and `gunzip`. These include `bzip2` and `bunzip2`, as well as `xz` and `unxz`.

The `gzip` command uses the **Lempel-Ziv** data *compression algorithm*, while the `bzip` utilities use a different compression algorithm called **Burrows-Wheeler** block sorting, which can compress files smaller than `gzip` at the expense of more CPU time. These files can be recognized because they have a `.bz` or `.bz2` extension instead of a `.gz` extension.

The `xz` and `unxz` tools are functionally similar to `gzip` and `gunzip` in that they use the **Lempel-Ziv-Markov (LZMA)** chain algorithm, which can result in lower decompression CPU times that are on par with `gzip` while providing the better compression ratios typically associated with the `bzip2` tools. Files compressed with the `xz` command use the `.xz` extension.

9.3 Archiving Files

If you had several files to send to someone, you could choose to compress each one individually. You would have a smaller amount of data in total than if you sent uncompressed files, however, you would still have to deal with many files at one time.

Archiving is the solution to this problem. The traditional UNIX utility to archive files is called `tar`, which is a short form of *TApe aRchive*. It was used to stream many files to a tape for backups or file transfer. The `tar` command takes in several files and creates a single output file that can be split up again into the original files on the other end of the transmission.

The `tar` command has three modes that are helpful to become familiar with:

- *Create:* Make a new archive out of a series of files.
- *Extract:* Pull one or more files out of an archive.
- *List:* Show the contents of the archive without extracting.

Remembering the modes is key to figuring out the command line options necessary to do what you want. In addition to the mode, remember where to specify the name of the archive, as you may be entering multiple file names on a command line.

9.3.1 Create Mode

```
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
```

Creating an archive with the `tar` command requires two named options:

Option	Function
`-c`	Create an archive.
`-f ARCHIVE`	Use archive file. The argument `ARCHIVE` will be the name of the resulting archive file.

All the remaining arguments are considered as input file names, either as a wildcard, a list of files, or both.

The following example shows a *tar file*, also called a *tarball*, being created from multiple files. The first argument creates an archive called `alpha_files.tar`. The wildcard option `*` is used to include all files that begin with `alpha` in the archive:

```
sysadmin@localhost:~/Documents$ tar -cf alpha_files.tar alpha*
sysadmin@localhost:~/Documents$ ls -l alpha_files.tar
-rw-rw-r-- 1 sysadmin sysadmin 10240 Oct 31 17:07 alpha_files.tar
```

The final size of `alpha_files.tar` is `10240` bytes. Normally, tarball files are slightly larger than the combined input files due to the overhead information on recreating the original files. Tarballs can be compressed for easier transport, either by using `gzip` on the archive or by having `tar` do it with the `-z` option.

Option	Function
`-z`	Compress (or decompress) an archive using the `gzip` command.

The next example shows the same command as the prior example, but with the addition of the `-z` option.

```
sysadmin@localhost:~/Documents$ tar -czf alpha_files.tar.gz alpha*
sysadmin@localhost:~/Documents$ ls -l alpha_files.tar.gz
-rw-rw-r-- 1 sysadmin sysadmin 417 Oct 31 17:15 alpha_files.tar.gz
```

The output is much smaller than the tarball itself, and the resulting file is compatible with `gzip`, which can be used to view the compression details. The uncompressed file is the same size as it would be if you tarred it in a separate step:

```
sysadmin@localhost:~/Documents$ gzip -l alpha_files.tar.gz
         compressed        uncompressed  ratio uncompressed_name
                417               10240  96.1% alpha_files.tar
```

While file extensions don't affect the way a file is treated, the convention is to use `.tar` for tarballs, and `.tar.gz` or `.tgz` for compressed tarballs.

The `bzip2` compression can be used instead of `gzip` by substituting the `-j` option for the `-z` option and using `.tar.bz2`, `.tbz`, or `.tbz2` as the file extension.

Option	Function
`-j`	Compress (or decompress) an archive using the `bzip2` command.

For example, to archive and compress the `School` directory:

```
sysadmin@localhost:~/Documents$ tar -cjf folders.tbz School
```

9.3.2 List Mode

```
tar -t [-f ARCHIVE] [OPTIONS]
```

Given a `tar` archive, compressed or not, you can see what's in it by using the `-t`option. The next example uses three options:

Option	Function
`-t`	List the files in an archive.
`-j`	Decompress with an `bzip2` command.
`-f ARCHIVE`	Operate on the given archive.

To list the contents of the `folders.tbz` archive:

```
sysadmin@localhost:~/Documents$ tar -tjf folders.tbz
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
School/Art/linux.txt
School/Math/
School/Math/numbers.txt
```

In the example, the directory `School/` is prefixed to the files. The `tar` command will recurse into subdirectories automatically when compressing and will store the path info inside the archive.

Consider This

To show that this file is still nothing special, we will list the contents of the file in two steps using a pipeline, the `|` character.

```
sysadmin@localhost:~/Documents$ bunzip2 -c folders.tbz | tar -t
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
School/Art/linux.txt
School/Math/
School/Math/numbers.txt
```

The left side of the pipeline is `bunzip2 -c folders.tbz`, which decompresses the file, but the `-c` option sends the output to the screen. The output is redirected to `tar -t`. If you don't specify a file with `-f` then tar will read from the standard input, which in this case is the uncompressed file.

Pipes and standard inputs will be covered in detail later in the course.

Note

The examples above are designed to demonstrate the `tar` command's ability to recurse into subdirectories. The files displayed in the example are not available within the virtual machine environment of this course.

9.3.3 Extract Mode

```
tar -x [-f ARCHIVE] [OPTIONS]
```

Creating archives is often used to make multiple files easier to move. Before extracting the files, relocate them to the `Downloads` directory:

```
sysadmin@localhost:~/Documents$ cd ~
sysadmin@localhost:~$ cp Documents/folders.tbz Downloads/folders.tbz
sysadmin@localhost:~$ cd Downloads
```

Finally, you can extract the archive with the `-x` option once it's copied into a different directory. The following example uses a similar pattern as before, specifying the operation, the compression, and a file name to operate on.

Option	Function
`-x`	Extract files from an archive.

Option	Function
`-j`	Decompress with the `bzip2` command.
`-f ARCHIVE`	Operate on the given archive.

```
sysadmin@localhost:~/Downloads$ tar -xjf folders.tbz
sysadmin@localhost:~/Downloads$ ls -l
total 8
drwx------ 5 sysadmin sysadmin 4096 Dec 20  2017 School
-rw-rw-r-- 1 sysadmin sysadmin  413 Oct 31 18:37 folders.tbz
```

The original file is untouched, and the new directory is created. Inside the directory, are the original directories and files.

```
sysadmin@localhost:~/Downloads$ cd School
sysadmin@localhost:~/Downloads/School$ ls -l
total 12
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:45 Art
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:47 Engineering
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:46 Math
```

Add the `-v` flag and you will get a verbose output of the files processed, making it easier to keep track of what's happening:

Option	Function
`-v`	Verbosely list the files processed.

The next example repeats the prior example, but with the addition of the `-v` option:

```
sysadmin@localhost:~/Downloads$ tar -xjvf folders.tbz
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
School/Art/linux.txt
School/Math/
```

```
School/Math/numbers.txt
```

It is important to keep the `-f` flag at the end, as `tar` assumes whatever follows this option is a file name. In the next example, the `-f` and `-v` flags were transposed, leading to `tar` interpreting the command as an operation on a file called `v`, which does not exist.

```
sysadmin@localhost:~/Downloads$ tar -xjfv folders.tbz
tar (child): v: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

If you only want some files out of the archive, add their names to the end of the command, but by default, they must match the name in the archive exactly, or use a pattern.

The following example shows the same archive as before, but extracting only the `School/Art/linux.txt` file. The output of the command (as verbose mode was requested with the `-v` flag) shows only the one file has been extracted:

```
sysadmin@localhost:~/Downloads$ tar -xjvf folders.tbz School/Art/linux.txt
School/Art/linux.txt
```

The `tar` command has many more features, such as the ability to use patterns when extracting files, excluding certain files, or outputting the extracted files to the screen instead of disk. The documentation for `tar` has in-depth information.

9.4 ZIP Files

The de facto archiving utility in Microsoft is the ZIP file. ZIP is not as prevalent in Linux but is well supported by the `zip` and `unzip` commands. Albeit, with `tar` and `gzip`/`gunzip` the same commands and options can be used interchangeably to do the creation and extraction, but this is not the case with `zip`. The same option has different meanings for the two different commands.

The default mode of `zip` is to add files to an archive and compress it.

```
zip [OPTIONS] [zipfile [file…]]
```

The first argument `zipfile` is the name of the archive to be created, after that, a list of files to be added. The following example shows a compressed archive called `alpha_files.zip` being created:

```
sysadmin@localhost:~/Documents$ zip alpha_files.zip alpha*
  adding: alpha-first.txt (deflated 32%)
  adding: alpha-second.txt (deflated 36%)
  adding: alpha-third.txt (deflated 48%)
  adding: alpha.txt (deflated 53%)
  adding: alpha_files.tar.gz (stored 0%)
```

The output shows the files and the compression ratio.

It should be noted that `tar` requires the `-f` option to indicate a filename is being passed, while `zip` and `unzip` require a filename and therefore don't need you to inform the command a filename is being passed.

The `zip` command will not recurse into subdirectories by default, which is different behavior than the `tar` command. That is, merely adding `School` will only add the empty directory and not the files under it. If you want `tar` like behavior, you must use the `-r` option to indicate recursion is to be used:

```
sysadmin@localhost:~/Documents$ zip -r School.zip School
updating: School/ (stored 0%)
updating: School/Engineering/ (stored 0%)
updating: School/Engineering/hello.sh (deflated 88%)
updating: School/Art/ (stored 0%)
updating: School/Art/linux.txt (deflated 49%)
updating: School/Math/ (stored 0%)
updating: School/Math/numbers.txt (stored 0%)
  adding: School/Art/red.txt (deflated 33%)
  adding: School/Art/hidden.txt (deflated 1%)
  adding: School/Art/animals.txt (deflated 2%)
```

In the example above, all files under the `School` directory are added because it uses the `-r` option. The first lines of output indicate that directories were added to the archive, but otherwise the output is similar to the previous example.

The `-l` *list* option of the `unzip` command lists files in `.zip` archives:

```
sysadmin@localhost:~/Documents$ unzip -l School.zip
Archive:  School.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2017-12-20 16:46   School/
        0  2018-10-31 17:47   School/Engineering/
      647  2018-10-31 17:47   School/Engineering/hello.sh
        0  2018-10-31 19:31   School/Art/
       83  2018-10-31 17:45   School/Art/linux.txt
        0  2018-10-31 17:46   School/Math/
       10  2018-10-31 17:46   School/Math/numbers.txt
       51  2018-10-31 19:31   School/Art/red.txt
       67  2018-10-31 19:30   School/Art/hidden.txt
       42  2018-10-31 19:31   School/Art/animals.txt
```

```
---------                     -------
      900                     10 files
        0  2018-10-31 17:46   School/Math/
       10  2018-10-31 17:46   School/Math/numbers.txt
---------                     -------
      740                     7 files
```

Extracting the files is just like creating the archive, as the default operation of the `unzip` command is to extract. It gives several options if unzipping files will overwrite existing ones:

```
sysadmin@localhost:~/Documents$ unzip School.zip
Archive:  School.zip
replace School/Engineering/hello.sh? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Art/linux.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Math/numbers.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Art/red.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Art/hidden.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Art/animals.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

This can be avoided by copying the zip file into a new directory:

```
sysadmin@localhost:~/Documents$ mkdir tmp
sysadmin@localhost:~/Documents$ cp School.zip tmp/School.zip
sysadmin@localhost:~/Documents$ cd tmp
sysadmin@localhost:~/Documents/tmp$ unzip School.zip
Archive:  School.zip
   creating: School/
   creating: School/Engineering/
  inflating: School/Engineering/hello.sh
   creating: School/Art/
  inflating: School/Art/linux.txt
   creating: School/Math/
 extracting: School/Math/numbers.txt
  inflating: School/Art/red.txt
  inflating: School/Art/hidden.txt
  inflating: School/Art/animals.txt
```

Here, we extract all the files in the archive to the current directory. Just like `tar`, you can pass filenames on the command line. The examples below show three different attempts to extract a file.

First, just the name of the file is passed without the directory component. Like `tar`, the file is not matched.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip linux.txt
Archive:  School.zip
caution: filename not matched:  linux.txt
```

A second attempt passes the directory component along with the file name, which extracts just that file.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip School/Math/numbers.txt
Archive:  School.zip
 extracting: School/Math/numbers.txt
```

The third version uses a wildcard, which extracts the four files matching the pattern, just like `tar`.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip School/Art/*t
Archive:  School.zip
  inflating: School/Art/linux.txt
  inflating: School/Art/red.txt
  inflating: School/Art/hidden.txt
  inflating: School/Art/animals.txt
```

The `zip` and `unzip` man pages describe the other things you can do with these tools, such as replace files within the archive, use different compression levels, and even use encryption.

Key Terms

bzip2
: Command used to compress and decompress files using the Burrows-Wheeler compression algorithm.
Section 9.2

common tar options
: Section 9.3.1 | Section 9.3.2 | Section 9.3.3

gzip
: Command used to compress and decompress files using the Lempel-Ziv compression algorithm.
Section 9.2

tar
: Command used to create, extract, and view archive files. The traditional UNIX archiving utility.
Section 9.3

unzip
: Command used to list and extract files from ZIP archives.
Section 9.4

xz
: Command used to compress and decompress files using the Lempel-Ziv-Markov compression algorithm.
Section 9.2

zip

Command used to archive and compress files. The primary archiving utility in Microsoft.
Section 9.4