2.3 - File interaction

Overview

When testing, you will often encounter situations where you need to interact with, or create archives. This will typically be to move large quantities of files easily between attacker and target.

Archive Tools

Please understand that this list below is not exhaustive and there is undoubtedly different compression technologies you will encounter as you work towards your pen-testing goals.

Gzip

As you will soon see, `tar` will replace most usage cases of `gzip`. For example, if you compress a `tar` archive, it will use `gzip`. `gzip` uses Lempel-Ziv (LZ77) encoding to reduce size of named files

Tar

Tar will likely be your day to day archive management tool on Linux, it is fast, configurable, and will be installed on just about every Linux distribution.

Zip

Zip is used to package and compress files. You will encounter zip files on both Windows and Linux Distributions. As you will soon see, Powershell may also be used to create and decompress `zip` archives from the command line.

Powershell

`Powershell` on Windows is more than capable of both creating, and extracting zip archives. Handy for those times you have a console connection to a Windows machine and need to move a significant amount of data while being unable to open WinZip.

How is it used

Compression utilities are used to package either files, or folders into manageable archives for ease of storage and transfer. You can alter the ratio of compression by sacrificing compression speed (the computer works harder to make a file end up smaller), and optionally password protect the archive.

Why is this important

Notwithstanding the real-world scenario which will be discussed shortly, archiving files is important for at a minimum, it will reduce the duration of a transfer by reducing the size of the transferred files.

Real-word applications

Consider the following example

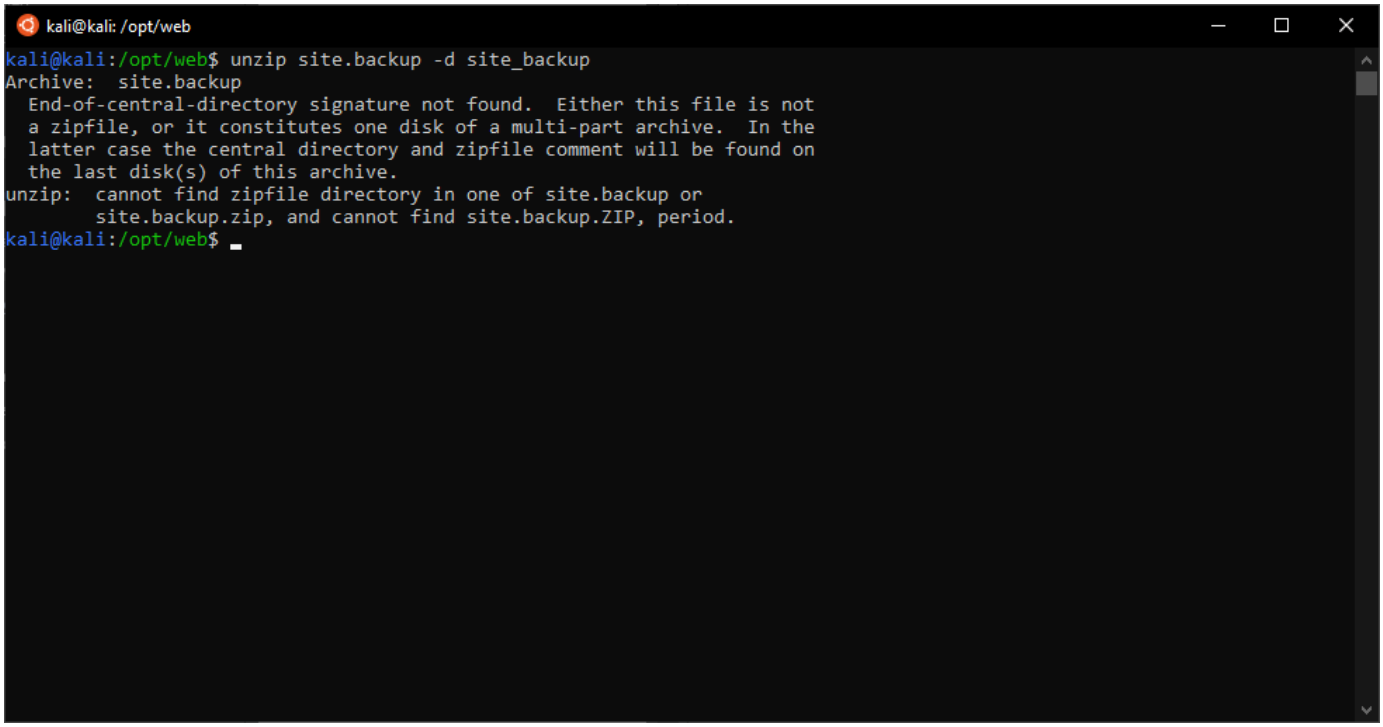
> You have achieved RCE on a target machine which enables you to execute single commands on the target. You are using these commands to work through the configuration files for a web server in hope for identifying useful credentials or database information. It may be more beneficial to instruct the server to archive the entire folder, enabling you to download it in one go for review on your attacking machine.

Potential Issues

There is not much that can go wrong with compressing and uncompromising archives. However, in some instances you may encounter an archive that is pretending to be something it's not. Let's look at an example.

Consider the following scenario:

> You have gained access to a web server and identified a file in a hidden directory called `site.backup`. You have a suspicion that it is an archive (makes sense, it's a site backup), so you try a simple unzip command (don't worry if you don't know the flags, we will get into that) and use `unzip site.backup -d site_backup` and see the error below.



Seems we need some further investigation.

> With `unzip` failing, you use the `file` command to determine the file type along with a `cat site.backup | head` to print the first `10 lines` of the file (as is the default for the `head` command). The image below shows that it is being reported as `ASCII text` and the contents of the file look at awful lot like `base64` encoded text.

```
Select kali@kali: /opt/web
kali@kali:/opt/web$ cat site.backup | head
UEsDBBQDAAAAAKQzR1IAAAAAAAAAAAAAAAAAEAAAAd2ViL1BLAwQUAwAACABInlVR+OV/Gu0GAACx
FAAACwAAAHdlYi8xNTAyMy5jrVd7b9MwEP8bJL7DrTBwSllbGGiiDFSxDiYeq7rxEqDITZzWLC/s
pA9gfHbu4mRpuhSEIGpT+3z+3e/Od5e03bx2FZogFrEfyQS8SMFi74H9YBd8GaYLOBMqFD5Ifu+u
XmqH+z6IIPV5IqMQGJ9wGVqEcEUJV2onmgkcwHgJYxHClM/PhM6W5zKZwlT4MXgqCkBFY6ES0HMp
nDMJPHQh4TOpIVIBTpa0h7441lxMZch9mqR8KxcfSl8kSqrbeRV8cBJrCMFXOt2Woz+QlgMJF1M/k
V0aGwklO4ZGhZPPEzlnthCJ5bHSH6psIltrnczhUXKehOINHMxFOMEi0w1NCjLW746djlOzEfrGP
zzF4Q6k58cI9MQlohwyCSO84UXChTLci3LGSMxAUbhNoJ3IFjCOlorlwTQR1LEJXqMLPNt6uXb0u
Q8dPUfcRHlY7WcZC70wfr8vnXCY14jhR3BFrCzJMNsEoMVmTpqHUibsmRImMLst8Oa7BDPD0L+kq
GRpL164SF1d4gLTAxjji2jhNhG0zhnRirgJ2z7IsYE2wMbiBTGwHk0RbLA21nIQYPT8KJ0BCq1fi
VVb/iBwrgQJhm9PKLGwyUGEBqxNcrAOCGlmPXCeXjSzAbKCMYLNIutD0pC9aYMazgGM5fr929cqq
kali@kali:/opt/web$ file site.backup
site.backup: ASCII text
kali@kali:/opt/web$
```

Our next step is to use the `base64` program with the `-d` flag to decode it, and store it as a new file. The image below shows that the `file` command now correctly identifies it as a zip file, so we could unzip it and start hunting for sensitive information.

```
kali@kali: /opt/web
kali@kali:/opt/web$ cat site.backup | base64 -d > site.file
kali@kali:/opt/web$ file site.file
site.file: Zip archive data, at least v2.0 to extract
kali@kali:/opt/web$
```

No file command? No problem

In the example above, we have used the `file` command to determine the type of file we are working with. Consider the following scenario:

> You have accessed a target machine, and the file command has been intentionally removed to make your life more difficult. However, you recall that you are able to use `File Signatures` (also known as `Magic Bytes`) to determine the type of file.

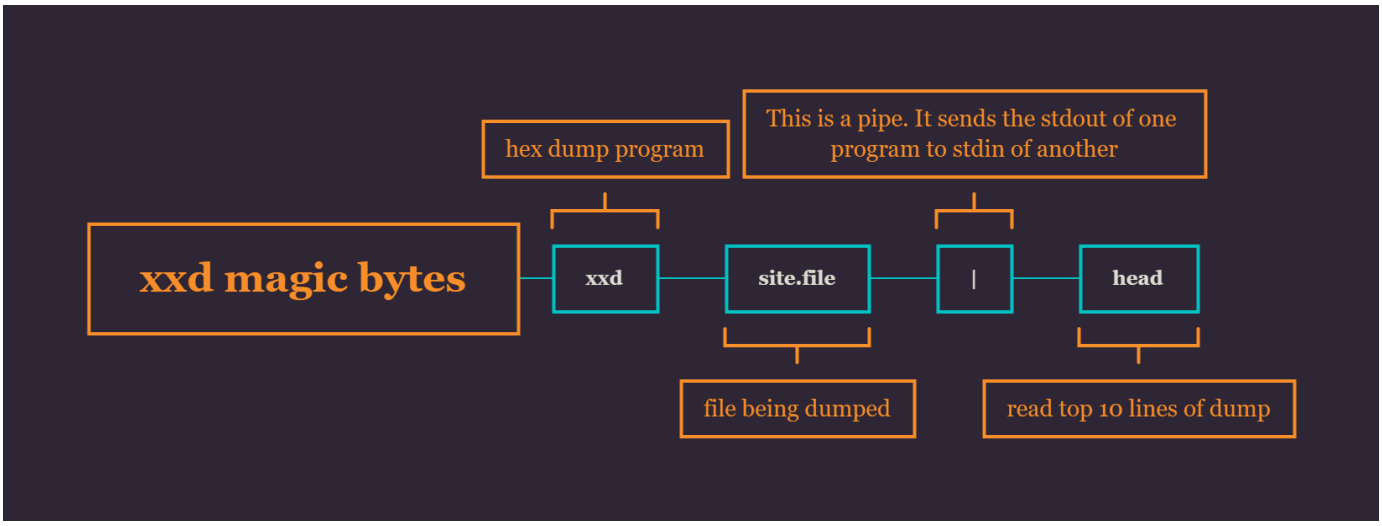
What are `Magic Bytes` I hear you ask?

> Magic bytes are `hex signatures` that appear at the start of the file. They allow programs (and humans) to identify what type of file they are attempting to open. They are not only useful for figuring out that a hidden file is in fact, an archive, but you can modify them to bypass some file upload restrictions as your php shell could instead be recognized as a harmless image file.

Viewing the Magic Bytes

To view the `magic bytes` of a file, we will use the tool `xxd`. `xxd` is used for making hex dumps.

The image below explains the syntax:



For ease of copy and paste:

```
xxd site.file | head
```

This image is the result. I have highlighted the `Magic Bytes`. You can look them up at the link below and it will show you that this is a `zip file format.`

https://en.wikipedia.org/wiki/List_of_file_signatures

```
Select kali@kali: /opt/web
kali@kali:/opt/web$ xxd site.file | head
00000000: 504b 0304 1403 0000 0000 a433 4752 0000  PK.........3GR..
00000010: 0000 0000 0000 0000 0000 0400 0000 7765  ..............we
00000020: 622f 504b 0304 1403 0000 0800 489e 5551  b/PK........H.UQ
00000030: f8e5 7f1a ed06 0000 b114 0000 0b00 0000  ................
00000040: 7765 622f 3135 3032 332e 63ad 577b 6fd3  web/15023.c.W{o.
00000050: 3010 ff1b 24be c3ad 3070 4a59 5b18 68a2  0...$...0pJY[.h.
00000060: 0c54 b10e 261e abba f112 a0c8 4d9c d62c  .T..&.......M..,
00000070: 2fec a40f 607c 76ee e264 69ba 1484 206a  /...`|v..di... j
00000080: 53fb 7cfe ddef ce77 97b4 ddbc 7615 9a20  S.|....w....v..
00000090: 16b1 1fc9 04bc 48c1 62ef 81fd 6017 7c19  ......H.b...`.|.
kali@kali:/opt/web$
```

Exercise

Installation

The command below will install the latest version of `gzip`, `zip`, and `tar` on your system.

```
sudo apt update && sudo apt install gzip zip tar -y
```

Usage

Gzip

`gzip` operates differently to most other archiving programs. It's a tool for compressing, not making an archive and then compressing (as tar is which we will see soon).

First, I'm using the command below to create a collection of text files. I'll end up with files called `file1.txt, file2.txt, etc.`

```
touch file{1..30}.txt
```

```
kali@kali: ~/working
kali@kali:~/working$ touch file{1..30}.txt
kali@kali:~/working$ ls
file10.txt  file13.txt  file16.txt  file19.txt  file21.txt  file24.txt  file27.txt  file2.txt   file4.txt  file7.txt
file11.txt  file14.txt  file17.txt  file1.txt   file22.txt  file25.txt  file28.txt  file30.txt  file5.txt  file8.txt
file12.txt  file15.txt  file18.txt  file20.txt  file23.txt  file26.txt  file29.txt  file3.txt   file6.txt  file9.txt
kali@kali:~/working$
```

Next, I'll have `gzip` compress each file in the directory using the `-r` flag

```
Select kali@kali: ~/working
kali@kali:~/working$ touch file{1..30}.txt
kali@kali:~/working$ ls
file10.txt  file13.txt  file16.txt  file19.txt  file21.txt  file24.txt  file27.txt  file2.txt   file4.txt  file7.txt
file11.txt  file14.txt  file17.txt  file1.txt   file22.txt  file25.txt  file28.txt  file30.txt  file5.txt  file8.txt
file12.txt  file15.txt  file18.txt  file20.txt  file23.txt  file26.txt  file29.txt  file3.txt   file6.txt  file9.txt
kali@kali:~/working$ gzip -r .
kali@kali:~/working$ ls
file10.txt.gz  file14.txt.gz  file18.txt.gz  file21.txt.gz  file25.txt.gz  file29.txt.gz  file4.txt.gz  file8.txt.gz
file11.txt.gz  file15.txt.gz  file19.txt.gz  file22.txt.gz  file26.txt.gz  file2.txt.gz   file5.txt.gz  file9.txt.gz
file12.txt.gz  file16.txt.gz  file1.txt.gz   file23.txt.gz  file27.txt.gz  file30.txt.gz  file6.txt.gz
file13.txt.gz  file17.txt.gz  file20.txt.gz  file24.txt.gz  file28.txt.gz  file3.txt.gz   file7.txt.gz
kali@kali:~/working$
```

The files have 0 byte sizes, but I hope you get the idea. If they were actual files, they would be compressed, but I'd still have 30 separate `.gz` files. Not super handy to move around. This is where `tar` comes in.

Tar

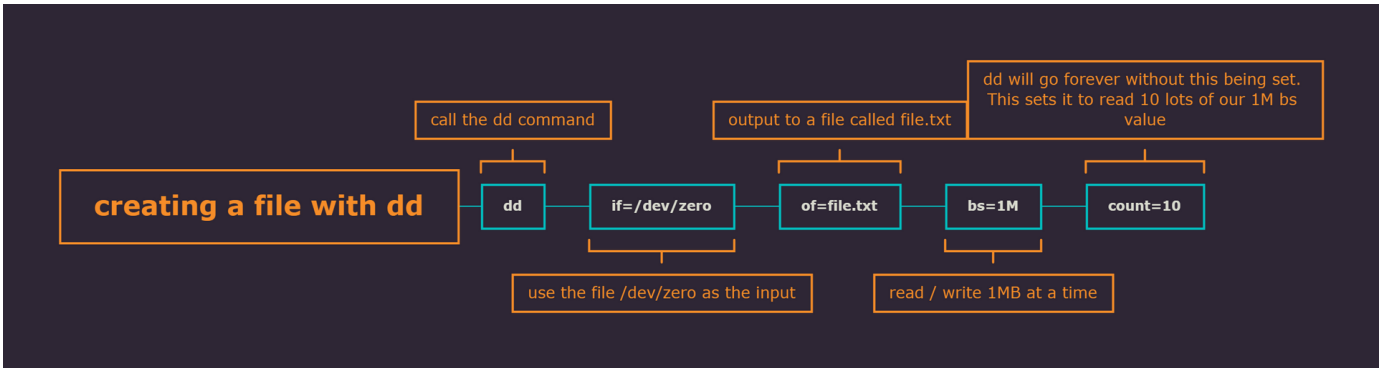
With the disadvantages of `gzip` in mind (being that it compresses discrete files, not folders). We are going to use `tar`. Tar will let us create an archive, and then use gzip to compress the archive. Let's look at why the compression is important.

Creating files to work with

The command below will create a file called `file.txt` which is populated from `/dev/zero`. `/dev/zero` is a file on Linux which provides `null` characters which will will use to fill up our file.

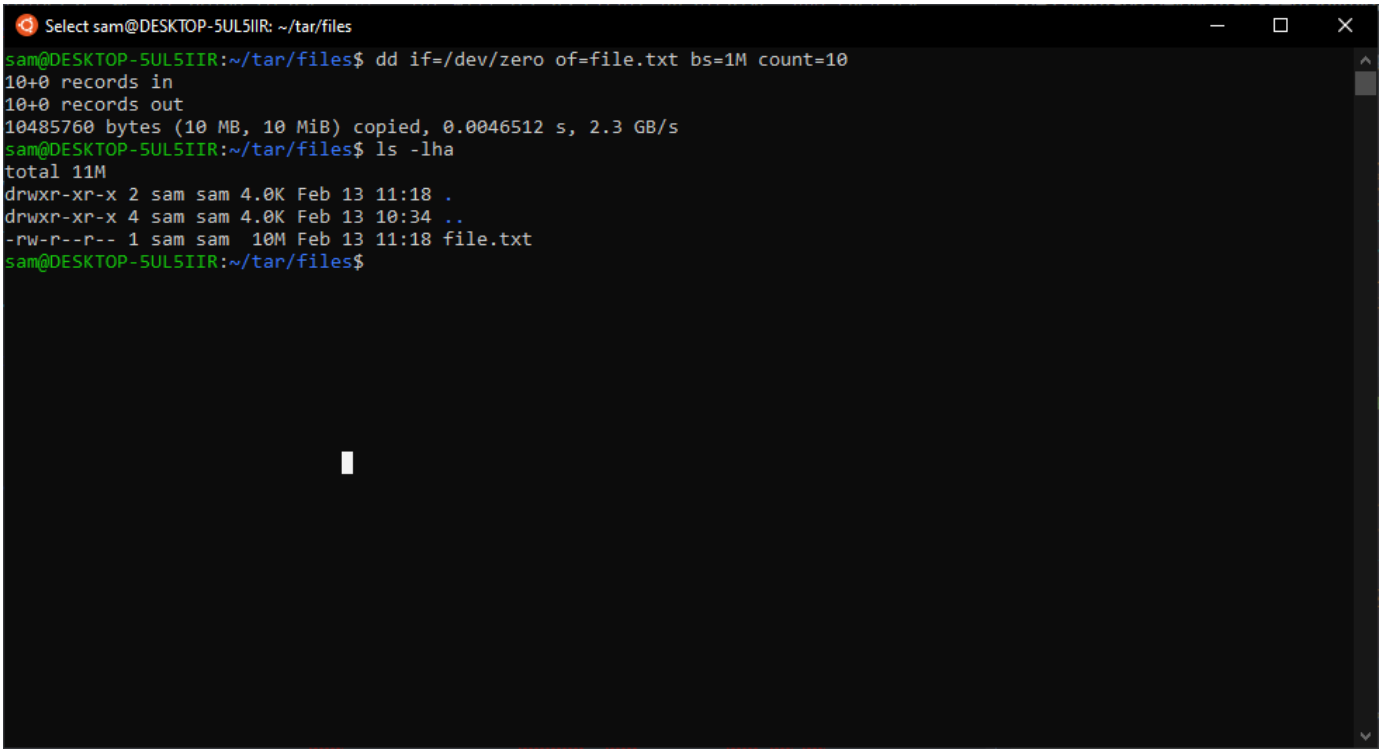
> Warning: `dd` is dangerous. You can overwrite your own drive and important files when used incorrectly. When used correctly, it is extremely powerful as will be seen when you start working on `forensic` type tasks.

> This first command will create a single file that is 10MB in size




```
dd if=/dev/zero of=file.txt bs=1M count=10
```

As you ca see below, we now have a single file.

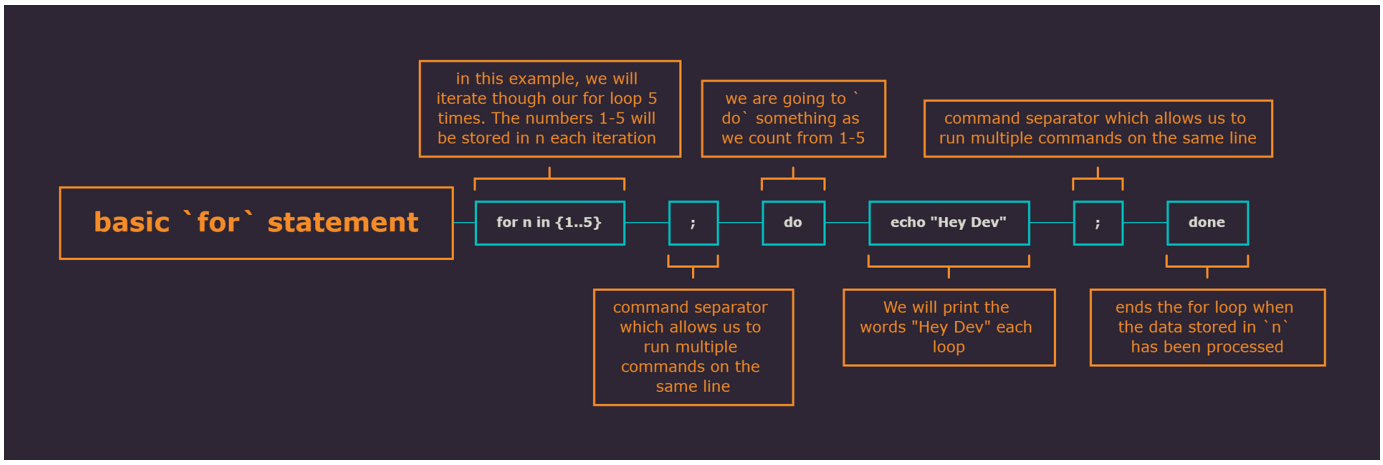


Hold on, but would it make sense that if we are going to make an `archive`, we need multiple files? Time to introduce `for loops`

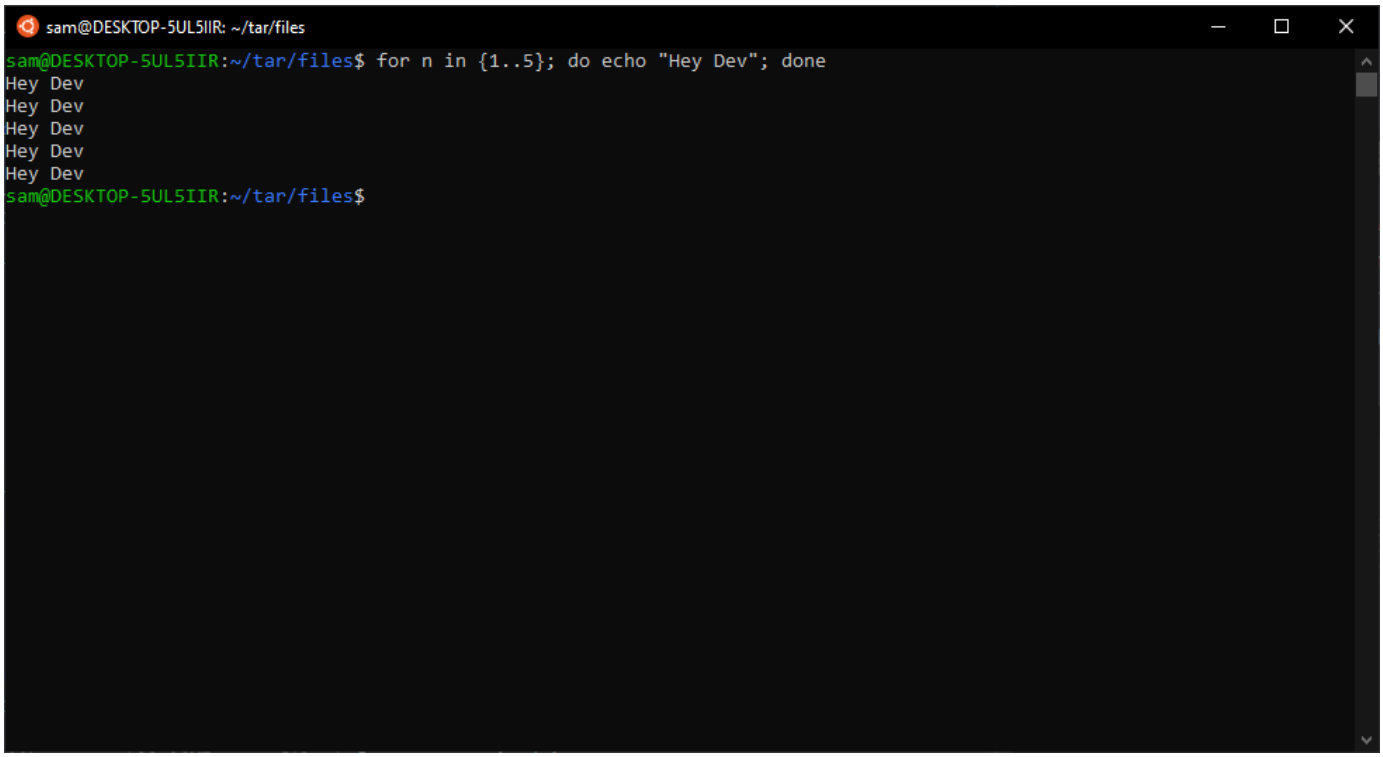
For Loop

A `for loop` allows a command to be repeatably executed. For example say we wanted to print "Hey Dev" five times on the screen.

> I'm going to use a value of `n` directly after the `for` statement. This is just a variable and you can call it largely whatever you want.




```
for n in {1..5}; do echo "Hey Dev"; done
```



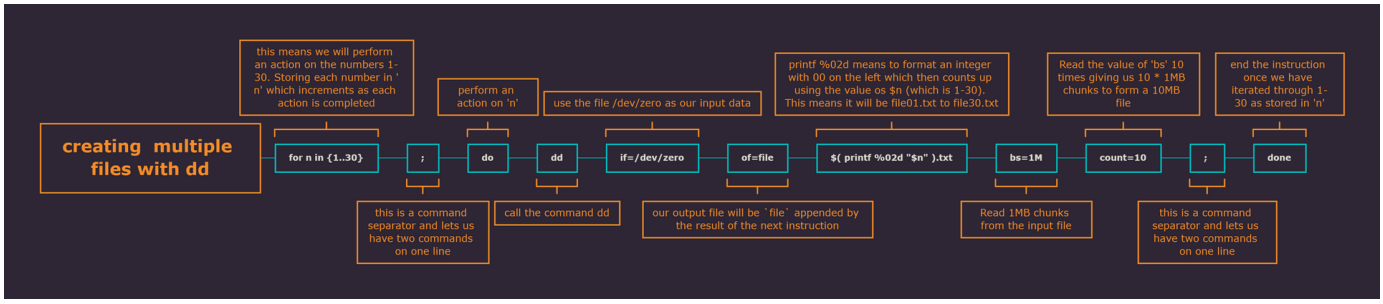
> Why don't you try swapping out `echo "Hey Dev"` for `echo $n` and you will see the value of `n` incrementing each time it runs through the loop.
> You could take it a step further and replace `echo "Hey Dev"` with `touch file{1..5}.txt` and you will end up creating five empty files throughout the loop as shown in the image below.

```
sam@DESKTOP-5UL5IIR: ~/tar/files
sam@DESKTOP-5UL5IIR:~/tar/files$ for n in {1..5}; do touch file{1..5}.txt; done
sam@DESKTOP-5UL5IIR:~/tar/files$ ls -lha
total 8.0K
drwxr-xr-x 2 sam sam 4.0K Feb 13 11:49 .
drwxr-xr-x 4 sam sam 4.0K Feb 13 10:34 ..
-rw-r--r-- 1 sam sam    0 Feb 13 11:49 file1.txt
-rw-r--r-- 1 sam sam    0 Feb 13 11:49 file2.txt
-rw-r--r-- 1 sam sam    0 Feb 13 11:49 file3.txt
-rw-r--r-- 1 sam sam    0 Feb 13 11:49 file4.txt
-rw-r--r-- 1 sam sam    0 Feb 13 11:49 file5.txt
sam@DESKTOP-5UL5IIR:~/tar/files$
```

Combining dd and a for loop

Now that we have seen that `dd` can make files and fill them with data, and that a `for loop` can automate that process for us, let's combine the two and have a `for loop of dd` create `30 10MB text files` for us to work with.

> The block below may seem intimidating, but we are combining a `for loop` with the `dd` file creation capability to create a collection of `10MB` files we can use for our archive activities.




```
for n in {1..30}; do dd if=/dev/zero of=file$( printf %02d "$n" ).txt
bs=1M count=10; done
```

That did it, we now have 30 * 10MB files.

```
sam@DESKTOP-5UL5IIR: ~/tar/files
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file01.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file02.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file03.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file04.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file05.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file06.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file07.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file08.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file09.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file10.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file11.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file12.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file13.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file14.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file15.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file16.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file17.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file18.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file19.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file20.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file21.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file22.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file23.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file24.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file25.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file26.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file27.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file28.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file29.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file30.txt
sam@DESKTOP-5UL5IIR:~/tar/files$
```

This has given us a total folder size of around `300MB`

```
sam@DESKTOP-5UL5IIR: ~/tar/files
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file03.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file04.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file05.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file06.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file07.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file08.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file09.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file10.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file11.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file12.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file13.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file14.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file15.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file16.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file17.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file18.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file19.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file20.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file21.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file22.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file23.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file24.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file25.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file26.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file27.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file28.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file29.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file30.txt
sam@DESKTOP-5UL5IIR:~/tar/files$ du -h
301M    .
sam@DESKTOP-5UL5IIR:~/tar/files$
```

Creating a tar archive

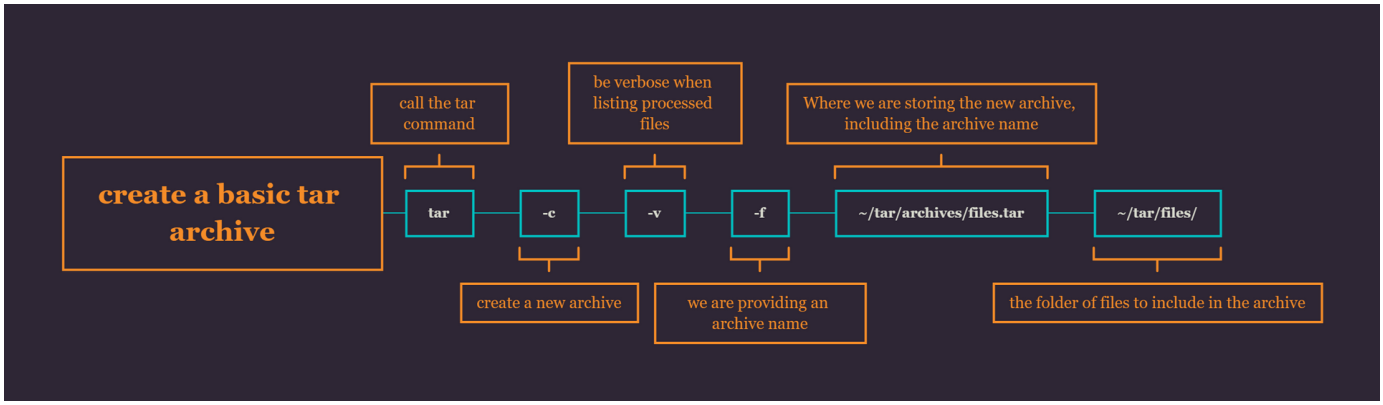
Our first attempt will be to just create an archive using our `30 10MB files` we have generated. My folder structure is as below.

> The `files` folder contains the 30 x 10MB files, and the `archive` folder will contain the resultant archives.

```
sam@DESKTOP-5UL5IIR: ~/tar
sam@DESKTOP-5UL5IIR:~/tar$ tree -d
.
├── archives
└── files

2 directories
sam@DESKTOP-5UL5IIR:~/tar$
```

Now we have our basic folder structure, we can use that to determine our command.

> Remember, `~/tar/files/` contains our `30 * 10MB` files.




```
tar -c -v -f ~/tar/archives/files.tar ~/tar/files/
```

> I've broken up the flags for the command above for ease of reading, but you may pass them as `tar cvf` instead.

The archive is the same size as the individual files

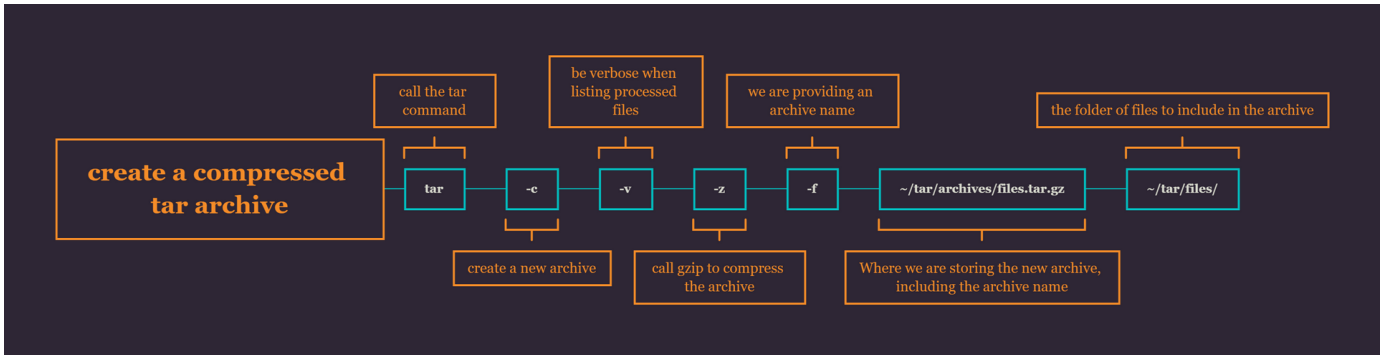
If you take a look at your newly created archive as in the image below, you will notice that the archive is the same size (if not slightly larger) than the files contained within it.

```
sam@DESKTOP-5UL5IIR:~/tar/archives$ ls -lha
total 301M
drwxr-xr-x 2 sam sam 4.0K Feb 13 11:59 .
drwxr-xr-x 4 sam sam 4.0K Feb 13 10:34 ..
-rw-r--r-- 1 sam sam 301M Feb 13 12:39 files.tar
sam@DESKTOP-5UL5IIR:~/tar/archives$
```

> This is because we have asked `tar` to make an archive, but not compress it. Let's make a new one, but compress it this time.

Creating a compressed tar archive

This time, we will use the `-z` flag to call gzip and compress our archive.




```
tar -c -v -z -f ~/tar/archives/files.tar.gz ~/tar/files/
```

> You will see a significant improvement in our new `files.tar.gz` archive with `301MB vs 301k`
> As with compressing, I split up -c -v -z -f for ease of reading. Normally you would use `-cvzf`

```
sam@DESKTOP-5UL5IIR: ~/tar/archives
sam@DESKTOP-5UL5IIR:~/tar/archives$ ls -lha
total 601M
-rw-r--r-- 1 sam sam 301M Feb 13 12:52 -z
drwxr-xr-x 2 sam sam 4.0K Feb 13 12:53 .
drwxr-xr-x 4 sam sam 4.0K Feb 13 10:34 ..
-rw-r--r-- 1 sam sam 301M Feb 13 12:39 files.tar
-rw-r--r-- 1 sam sam 301K Feb 13 12:53 files.tar.gz
sam@DESKTOP-5UL5IIR:~/tar/archives$
```

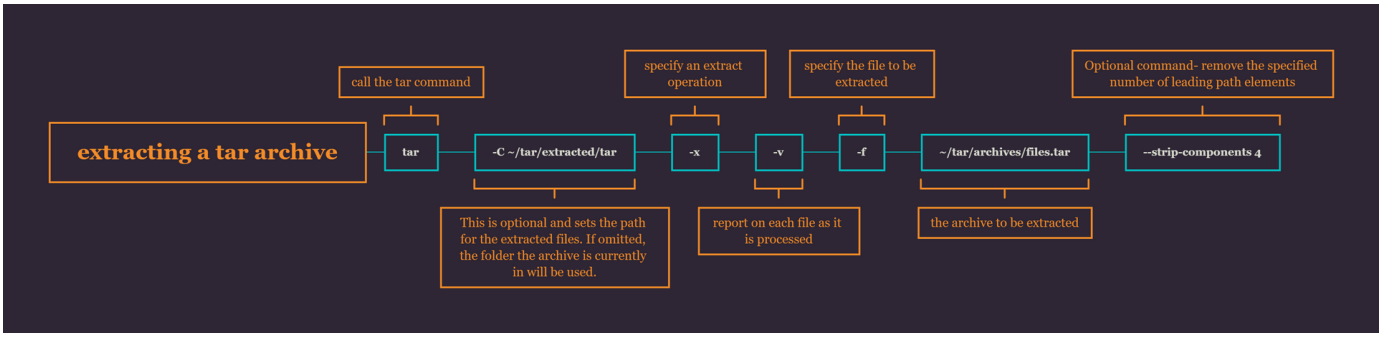
Extract a tar file

> The instructions for extracting a `tar` and a `tar.gz` file are slightly different. This first section will focus on a `tar` file.
> I've added a second set of folders for extracting the `tar` and `tar.gz` as shown in the image below. Please keep in mind the `-C` parameter which selects the extraction folder is `optional` omitting it will extract files to the folder the archive resides in.

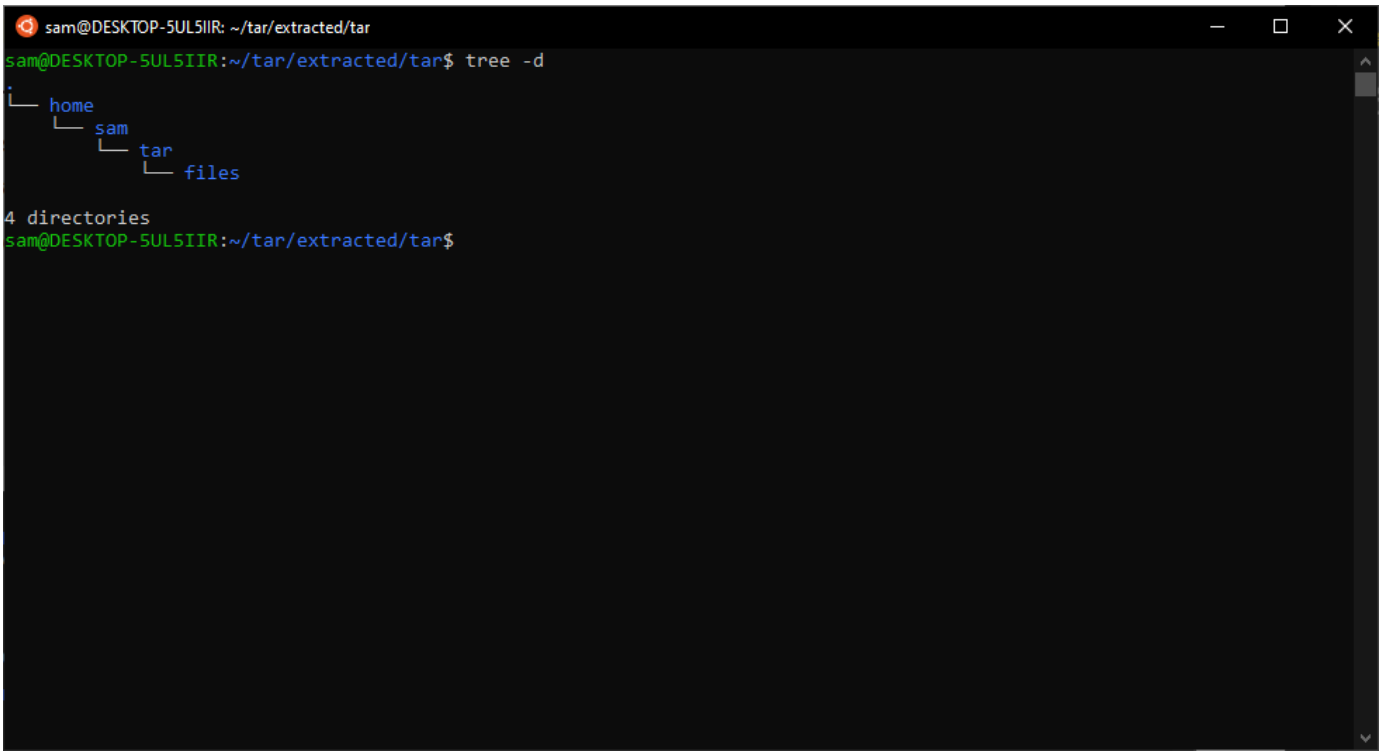
```
sam@DESKTOP-5UL5IIR: ~/tar
sam@DESKTOP-5UL5IIR:~/tar$ tree -d
.
├── archives
├── extracted
│   ├── tar
│   └── tar.gz
└── files

5 directories
sam@DESKTOP-5UL5IIR:~/tar$
```

> We can look at this two ways. We can extract the archive as it is, and it will include the files, along with the full path to where they where when the archive was made. For example, as the

> command that compressed the archive was `~/tar/files/` there are four `path elements` in front of the files we are interested in. These being `home;sam;tar;files`. The files you are after are within the last folder.




```
tar -C ~/tar/extracted/tar -x -v -f ~/tar/archives/files.tar
```



> You may use the `--strip-components option` to remove these path elements. As there are four, you could set the value to four and they will be extracted to the specified directory

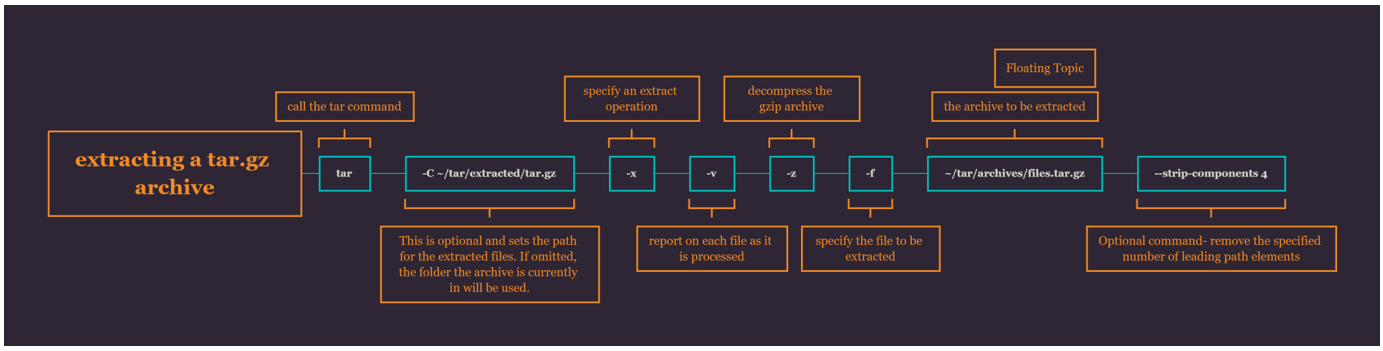
```
tar -C ~/tar/extracted/tar -x -v -f ~/tar/archives/files.tar --strip-components 4
```

```
sam@DESKTOP-5UL5IIR: ~/tar/extracted/tar
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file01.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file02.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file03.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file04.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file05.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file06.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file07.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file08.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file09.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file10.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file11.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file12.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file13.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file14.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file15.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file16.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file17.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file18.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file19.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file20.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file21.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file22.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file23.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file24.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file25.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file26.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file27.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file28.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file29.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file30.txt
sam@DESKTOP-5UL5IIR:~/tar/extracted/tar$
```

Extract a tar.gz file

The options are only slightly different for extracting a `tar.gz` compressed version of a tar archive.

> As before, the -C for the extraction path and `--strip-components` are optional




```
tar -C ~/tar/extracted/tar.gz -x -v -z -f ~/tar/archives/files.tar.gz --strip-components 4
```

```
sam@DESKTOP-5UL5IIR: ~/tar/extracted/tar.gz
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file01.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file02.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file03.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file04.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file05.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file06.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file07.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file08.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file09.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file10.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file11.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file12.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file13.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file14.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file15.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file16.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file17.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file18.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file19.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file20.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file21.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file22.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file23.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file24.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file25.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file26.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file27.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file28.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file29.txt
-rw-r--r-- 1 sam sam  10M Feb 13 11:55 file30.txt
sam@DESKTOP-5UL5IIR:~/tar/extracted/tar.gz$
```

Zip

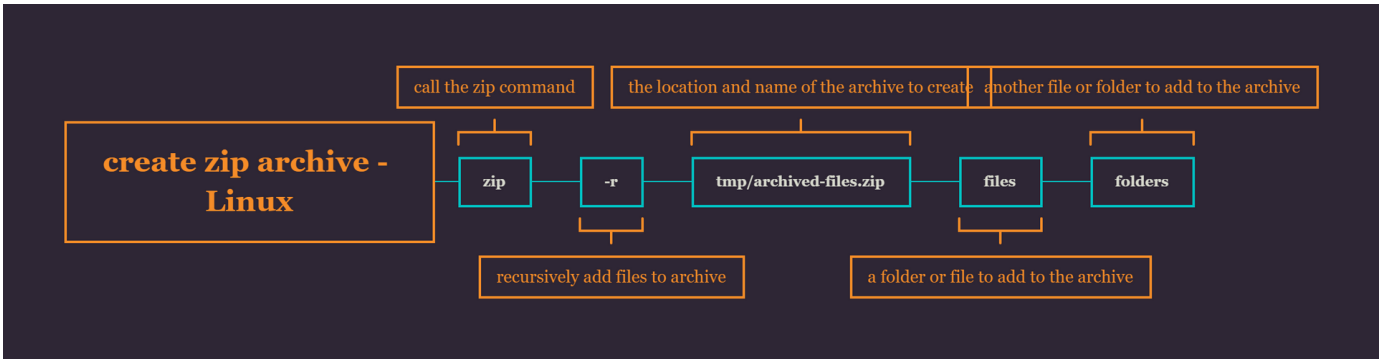
Zip is found just about anywhere, across windows and linux. In this example, I have gone ahead and created a folder called `files` which contains `30 * 10MB` files generated the same method as contained within the tar section. I have also created a folder called `folders` which containers `five` empty folders created with the command below

```
for n in {1..5}; do mkdir folder_$n; done
```

> The image below shows my folder structure, the folders_1-5 are empty, but the files folder has 30 * 10MB files.

```
sam@DESKTOP-5UL5IIR: ~/zip
sam@DESKTOP-5UL5IIR:~/zip$ tree -d
.
├── archives
├── files
└── folders
    ├── folder_1
    ├── folder_2
    ├── folder_3
    ├── folder_4
    └── folder_5

8 directories
sam@DESKTOP-5UL5IIR:~/zip$ ls files/
file01.txt  file04.txt  file07.txt  file10.txt  file13.txt  file16.txt  file19.txt  file22.txt  file25.txt  file28.txt
file02.txt  file05.txt  file08.txt  file11.txt  file14.txt  file17.txt  file20.txt  file23.txt  file26.txt  file29.txt
file03.txt  file06.txt  file09.txt  file12.txt  file15.txt  file18.txt  file21.txt  file24.txt  file27.txt  file30.txt
sam@DESKTOP-5UL5IIR:~/zip$
```

Below explains the basic zip command. We will make a folder called `archived-files.zip` with the tmp folder in our local directory. This archive will contain the recursive contents of the folders `files` and `folders`




```
zip -r tmp/archived-files.zip files folders
```

```
sam@DESKTOP-5UL5IIR: ~/zip/tmp
sam@DESKTOP-5UL5IIR:~/zip/tmp$ ls -lha
total 316K
drwxr-xr-x 2 sam sam 4.0K Feb 13 14:19 .
drwxr-xr-x 6 sam sam 4.0K Feb 13 13:44 ..
-rw-r--r-- 1 sam sam 306K Feb 13 14:19 archived-files.zip
sam@DESKTOP-5UL5IIR:~/zip/tmp$
```

Decompressing a Zip archive

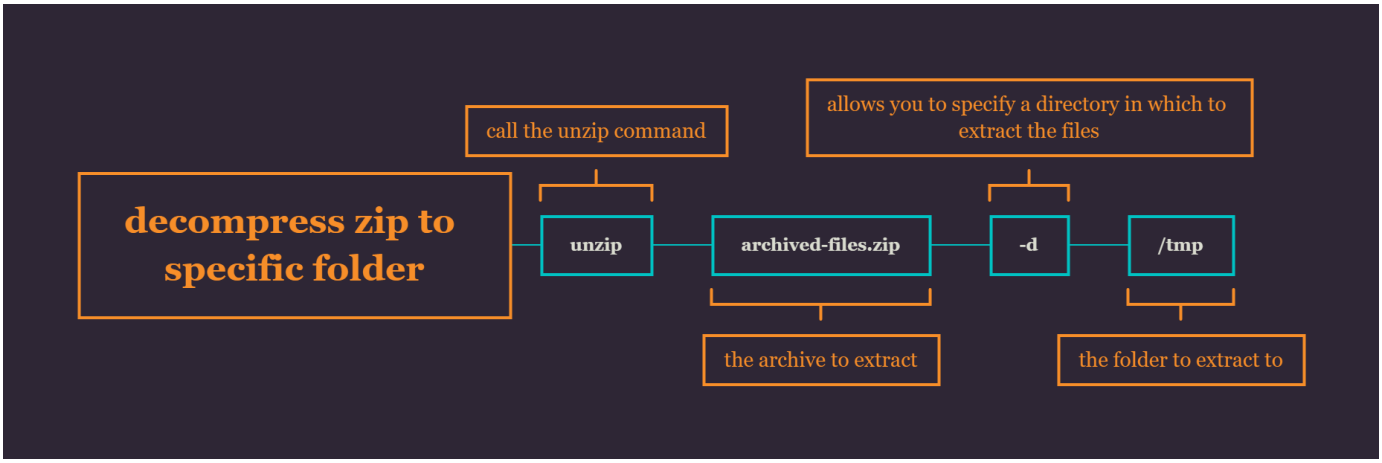
The simplest command is below

> This command will unzip the contents to the current folder

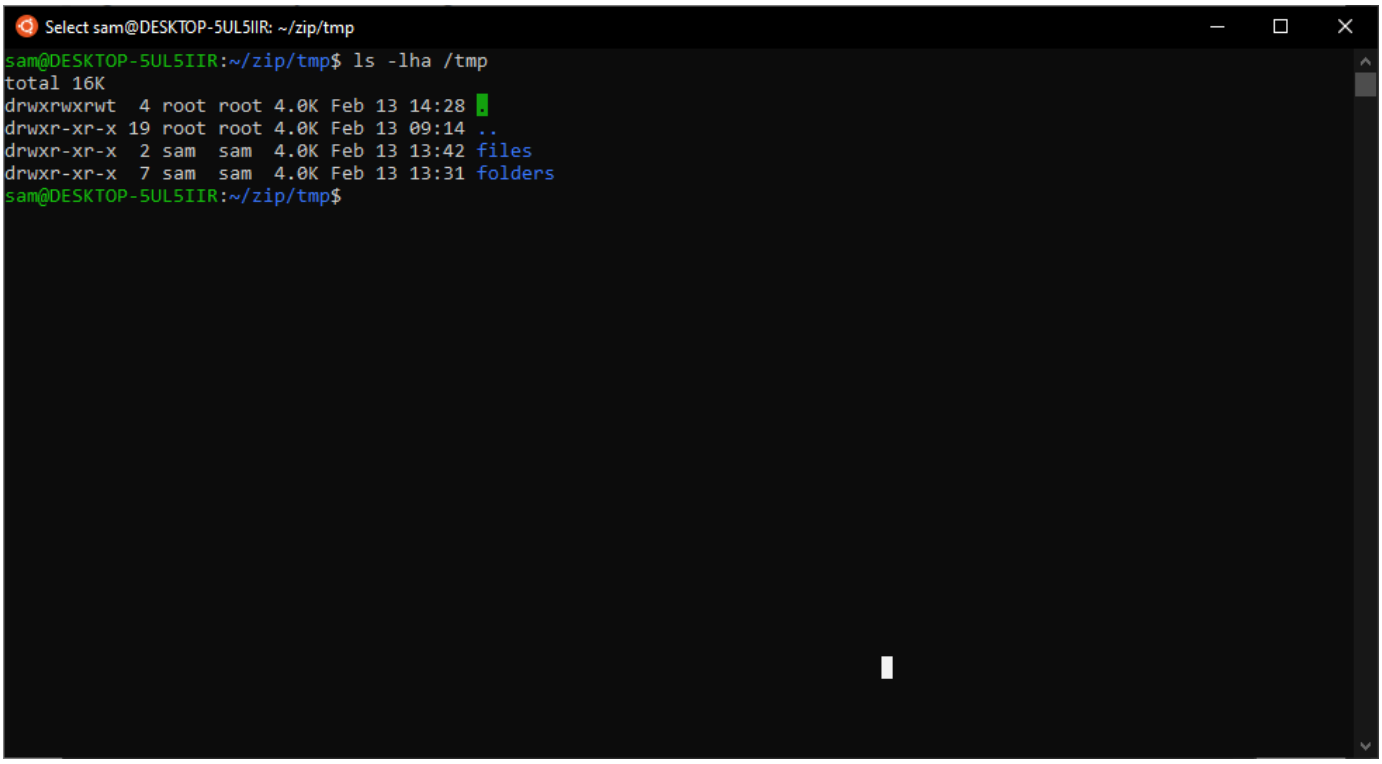
```
unzip archived-files.zip
```

```
sam@DESKTOP-5UL5IIR: ~/zip/tmp
sam@DESKTOP-5UL5IIR:~/zip/tmp$ ls -lha
total 324K
drwxr-xr-x 4 sam sam 4.0K Feb 13 14:26 .
drwxr-xr-x 6 sam sam 4.0K Feb 13 13:44 ..
-rw-r--r-- 1 sam sam 306K Feb 13 14:19 archived-files.zip
drwxr-xr-x 2 sam sam 4.0K Feb 13 13:42 files
drwxr-xr-x 7 sam sam 4.0K Feb 13 13:31 folders
sam@DESKTOP-5UL5IIR:~/zip/tmp$
```

> What about unzipping to a specific folder? perhaps you don't have write access to the entire machine but can only upload to a specify a folder. Below let's you specific a directly to extract to



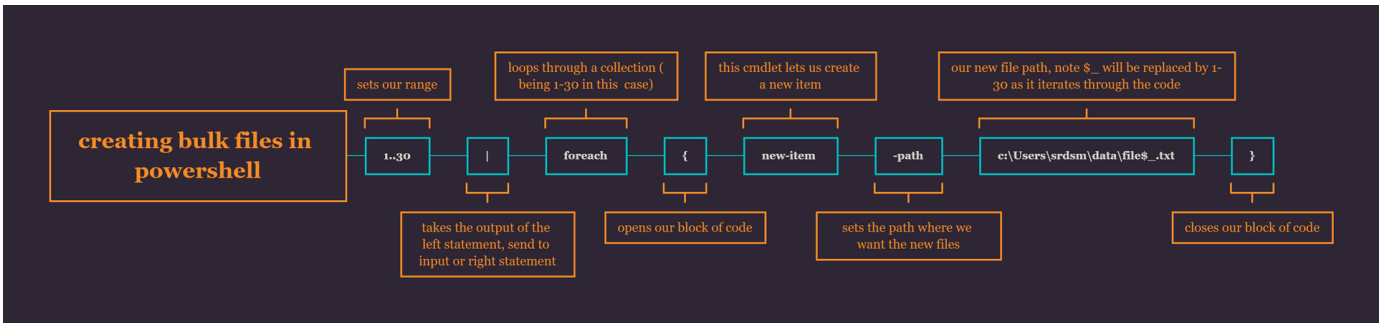

```
unzip archived-files.zip -d /tmp
```



Powershell

Powershell will likely be your go-to for almost everything on Windows from reverse shells, to decoding credentials, to compressing files; Powershell can do it all.

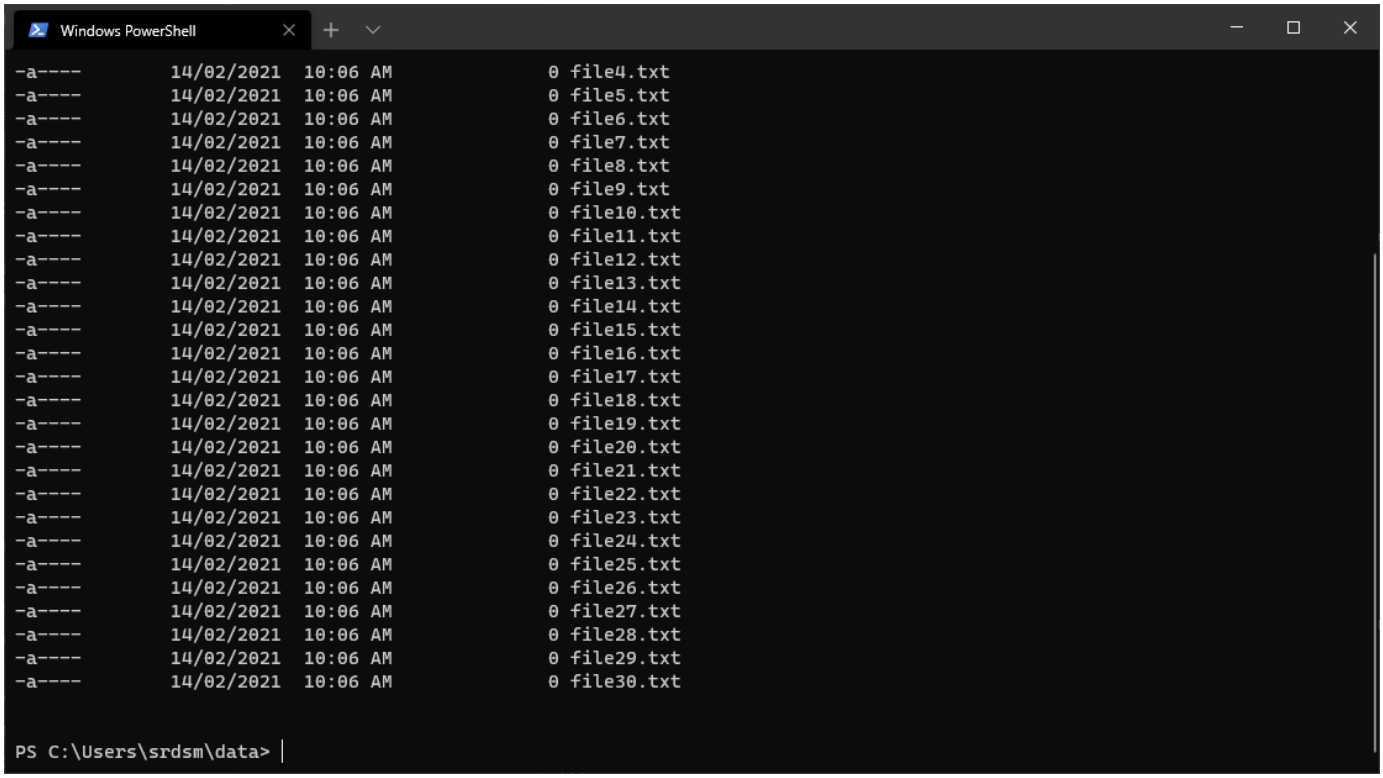
As we are looking at making archives, the first thing we are going to do is use a loop statement to create our 30 text files within a folder.



> This command needs to be run within Powershell

```
1..30 | foreach { new-item -path c:\Users\srdsm\powershell-zip\files\file$_.txt }
```

With that run, our files for archiving are created.



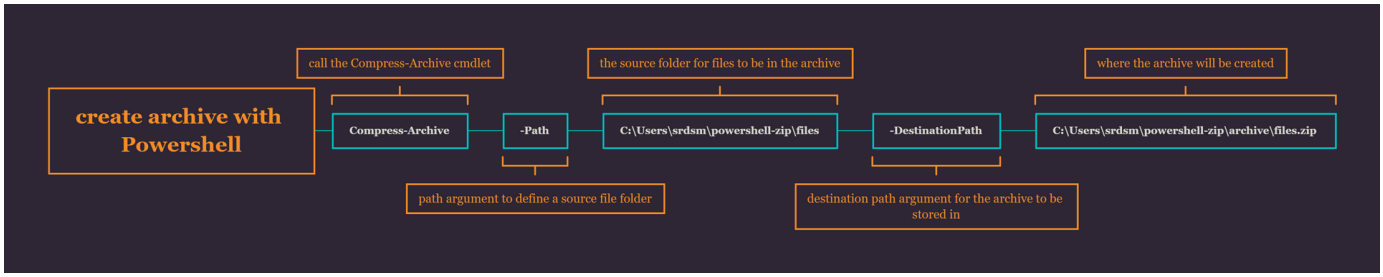
For this activity, I have the following folder structure.

> **archive is where I'll store the compressed archive**
>
> **extracted is where I will extract the compressed files**
>
> **files will hold the thirty (30) text files to be compressed**

```
PS C:\Users\srdsm\powershell-zip> tree
Folder PATH listing
Volume serial number is 96FF-5200
C:.
├───archive
├───extracted
└───files
PS C:\Users\srdsm\powershell-zip>
```

Compress a folder of files

The image below explains the arguments required to create the archive




```
Compress-Archive -Path C:\Users\srdsm\powershell-zip\files -
DestinationPath C:\Users\srdsm\powershell-zip\archive\files.zip
```

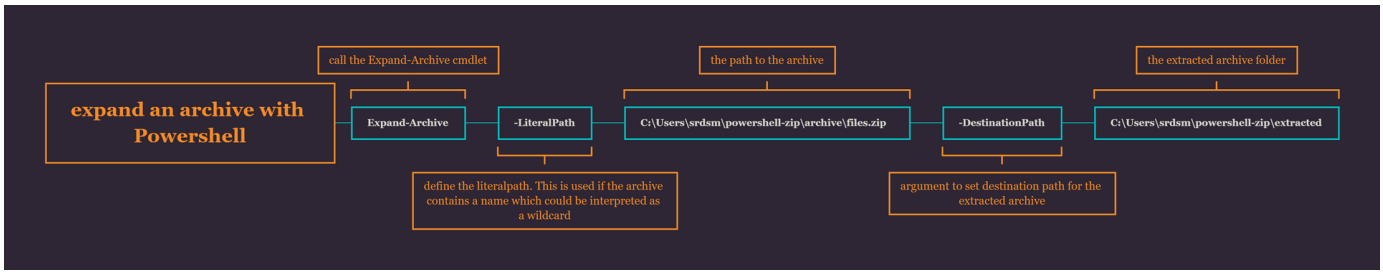
With the command run, our archive is created

```
Windows PowerShell
PS C:\Users\srdsm\powershell-zip> Compress-Archive -Path C:\Users\srdsm\powershell-zip\files -DestinationPath C:\Users\s
rdsm\powershell-zip\archive\files.zip
PS C:\Users\srdsm\powershell-zip> dir .\archive\


    Directory: C:\Users\srdsm\powershell-zip\archive


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        14/02/2021  10:20 AM           3244 files.zip


PS C:\Users\srdsm\powershell-zip>
```

Decompress an archive with Powershell

> The example below will use the `-LiteralPath` parameter instead of the `Path` parameter. This would be used if the archive contained components in the name which could be interpreted as wildcards (e.g. `[v1]`)

The image below explains the command used




```
Expand-Archive -LiteralPath C:\Users\srdsm\powershell-zip\archive\files.zip -DestinationPath C:\Users\srdsm\powershell-zip\extracted
```

Once executed, the archive is extracted and the files are available.

```
Windows PowerShell

PS C:\Users\srdsm\powershell-zip\extracted> Expand-Archive -LiteralPath C:\Users\srdsm\powershell-zip\archive\files.zip
-DestinationPath C:\Users\srdsm\powershell-zip\extracted
PS C:\Users\srdsm\powershell-zip\extracted> dir


    Directory: C:\Users\srdsm\powershell-zip\extracted


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        14/02/2021  10:30 AM                files


PS C:\Users\srdsm\powershell-zip\extracted> cd .\files\
PS C:\Users\srdsm\powershell-zip\extracted\files> dir


    Directory: C:\Users\srdsm\powershell-zip\extracted\files


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        14/02/2021  10:07 AM              0 file1.txt
-a----        14/02/2021  10:07 AM              0 file10.txt
-a----        14/02/2021  10:07 AM              0 file11.txt
-a----        14/02/2021  10:07 AM              0 file12.txt
-a----        14/02/2021  10:07 AM              0 file13.txt
-a----        14/02/2021  10:07 AM              0 file14.txt
-a----        14/02/2021  10:07 AM              0 file15.txt
-a----        14/02/2021  10:07 AM              0 file16.txt
```

Assessment