Introduction

Unix treats every object in the operating system as a file. We can find the files associated with every action performed and can make use of them for different system or process-related manipulations. For example, the command terminal that we use is associated with a device file. We can write to the terminal by writing to the corresponding device file for that specific terminal. Files take different forms such as directories, regular files, block devices, character-special devices, symbolic links, sockets, named pipes, and so on. Filename, size, file type, modification time, access time, change time, inode, links associated, and the filesystem the file is on are all attributes and properties that files can have. This chapter deals with recipes that handle operations or properties related to files.

Generating files of any size

For various reasons, you may need to generate a file filled with random data. It may be for creating a test file to perform tests, such as an application efficiency test that uses a large file as input, or to test the splitting of files into many parts, or to create loopback filesystems (**loopback files** are files that can contain a filesystem itself and these files can be mounted similarly to a physical device using the `mount` command). It takes effort to create such files by writing specific programs. So we use general utilities.

How to do it...

The easiest way to create a large-size file with the given size is to use the `dd` command. The `dd` command clones the given input and writes an exact copy to the output. Input can be `stdin`, a device file, a regular file, or so on. Output can be `stdout`, a device file, a regular file, or so on. An example of the `dd` command is as follows:

```
$ dd if=/dev/zero of=junk.data bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.00767266 s, 137 MB/s
```

The preceding command will create a file called `junk.data` that is exactly 1 MB in size. Let's go through the parameters: `if` stands for the `- input` file, `of` stands for the `- output` file, `bs` stands for bytes for a block, and `count` stands for the number of blocks of `bs` specified to be copied.

Be careful while using the `dd` command, it operates on a very low level with the devices. If you make a mistake, you might end up wiping your disk or corrupting data otherwise. So, always double check your `dd` command syntax, especially your `of=` parameter for correctness.

In the previous example, we are only creating a file, which is 1 MB in size, by specifying `bs` as 1 MB with a count of 1. If `bs` was set to `2M` and `count` to `2`, the total file size would be 4 MB.

We can use various units for **block size** (**BS**) as follows. Append any of the following characters to the number to specify the size in bytes:

Unit size	Code
Byte (1 B)	`c`
Word (2 B)	`w`
Block (512 B)	`b`
Kilobyte (1024 B)	`k`
Megabyte (1024 KB)	`M`
Gigabyte (1024 MB)	`G`

We can generate a file of any size using this. Instead of MB we can use any other unit notations, such as the ones mentioned in the previous table.

`/dev/zero` is a character special device, which infinitely returns the zero byte (`\0`).

If the input parameter (`if`) is not specified, it will read the input from `stdin` by default. Similarly, if the output parameter (`of`) is not specified, it will use `stdout` as the default output sink.

The `dd` command can also be used to measure the speed of memory operations by transferring a large quantity of data and checking the command output (for example, `1048576 bytes (1.0 MB) copied, 0.00767266 s, 137 MB/s` as seen in the previous example).

The intersection and set difference (A-B) on text files

Intersection and set difference operations are commonly used in mathematical classes on set theory. However, similar operations on strings are also very helpful in some scenarios.

Getting ready

The `comm` command is a utility to perform a comparison between the two files. It has many good options to arrange the output in such a way that we can perform intersection, difference, and set difference operations.

- **Intersection**: The intersection operation will print the lines that the specified files have in common with one another