6. However, there is a way to redirect data to a file, as well as provide a copy of redirected data as `stdin` for the next set of commands. This can be done using the `tee` command. For example, to print `stdout` in the terminal as well as redirect `stdout` into a file, the syntax for `tee` is as follows:

    ```
    command | tee FILE1 FILE2
    ```

    In the following code, the `stdin` data is received by the `tee` command. It writes a copy of `stdout` to the `out.txt` file and sends another copy as `stdin` for the next command. The `cat -n` command puts a line number for each line received from `stdin` and writes it into `stdout`:

    ```
    $ cat a* | tee out.txt | cat -n
    cat: a1: Permission denied
         1a1
         2a1
    ```

    Examine the contents of `out.txt` as follows:

    ```
    $ cat out.txt
    a1
    a1
    ```

    Note that `cat: a1: Permission denied` does not appear because it belongs to `stderr`. The `tee` command can read from `stdin` only.

    By default, the `tee` command overwrites the file, but it can be used with appended options by providing the `-a` option, for example, `$ cat a* | tee -a out.txt | cat -n`.

    Commands appear with arguments in the format: `command FILE1 FILE2` ... or simply `command FILE`.

7. We can use `stdin` as a command argument. It can be done by using `-` as the filename argument for the command as follows:

    ```
    $ cmd1 | cmd2 | cmd -
    ```

    For example:

    ```
    $ echo who is this | tee -
    who is this
    who is this
    ```

    Alternately, we can use `/dev/stdin` as the output filename to use `stdin`.

    Similarly, use `/dev/stderr` for standard error and `/dev/stdout` for standard output. These are special device files that correspond to `stdin`, `stderr`, and `stdout`.

## How it works...

For output redirection, `>` and `>>` operators are different. Both of them redirect text to a file, but the first one empties the file and then writes to it, whereas the later one adds the output to the end of the existing file.

When we use a redirection operator, the output won't print in the terminal but it is directed to a file. When redirection operators are used, by default, they operate on standard output. To explicitly take a specific file descriptor, you must prefix the descriptor number to the operator.

`>` is equivalent to `1>` and similarly it applies for `>>` (equivalent to `1>>`).

When working with errors, the `stderr` output is dumped to the `/dev/null` file. `./dev/null` is a special device file where any data received by the file is discarded. The null device is often known as a **black hole** as all the data that goes into it is lost forever.

## There's more...

A command that reads `stdin` for input can receive data in multiple ways. Also, it is possible to specify file descriptors of our own using `cat` and pipes, for example:

```
$ cat file | cmd
$ cmd1 | cmd
```

### Redirection from a file to a command

By using redirection, we can read data from a file as `stdin` as follows:

```
$ cmd < file
```

### Redirecting from a text block enclosed within a script

Sometimes we need to redirect a block of text (multiple lines of text) as standard input. Consider a particular case where the source text is placed within the shell script. A practical usage example is writing a logfile header data. It can be performed as follows:

```
#!/bin/bash
cat<<EOF>log.txt
LOG FILE HEADER
This is a test log file
Function: System statistics
EOF
```