The lines that appear between `cat <<EOF >log.txt` and the next `EOF` line will appear as the `stdin` data. Print the contents of `log.txt` as follows:

```
$ cat log.txt
LOG FILE HEADER
This is a test log file
Function: System statistics
```

Custom file descriptors

A file descriptor is an abstract indicator for accessing a file. Each file access is associated with a special number called a file descriptor. 0, 1, and 2 are reserved descriptor numbers for `stdin`, `stdout`, and `stderr`.

We can create our own custom file descriptors using the `exec` command. If you are already familiar with file programming with any other programming language, you might have noticed modes for opening files. Usually, the following three modes are used:

- Read mode
- Write with truncate mode
- Write with append mode

`<` is an operator used to read from the file to `stdin`. `>` is the operator used to write to a file with truncation (data is written to the target file after truncating the contents). `>>` is an operator used to write to a file by appending (data is appended to the existing file contents and the contents of the target file will not be lost). File descriptors can be created with one of the three modes.

Create a file descriptor for reading a file, as follows:

```
$ exec 3<input.txt # open for reading with descriptor number 3
```

We could use it in the following way:

```
$ echo this is a test line > input.txt
$ exec 3<input.txt
```

Now you can use file descriptor `3` with commands. For example, we will use `cat <&3` as follows:

```
$ cat<&3
this is a test line
```

If a second read is required, we cannot re-use the file descriptor `3`. It is required that we reassign the file descriptor `3` for read using `exec` for making a second read.

Create a file descriptor for writing (truncate mode) as follows:

```
$ exec 4>output.txt # open for writing
```

For example:

```
$ exec 4>output.txt
$ echo newline >&4
$ cat output.txt
newline
```

Create a file descriptor for writing (append mode) as follows:

```
$ exec 5>>input.txt
```

For example:

```
$ exec 5>>input.txt
$ echo appended line >&5
$ cat input.txt
newline
appended line
```

Arrays and associative arrays

Arrays are a very important component for storing a collection of data as separate entities using indexes. Regular arrays can use only integers as their array index. On the other hand, Bash also supports associative arrays that can take a string as their array index. Associative arrays are very useful in many types of manipulations where having a string index makes more sense. In this recipe, we will see how to use both of these.

Getting ready

To use associate arrays, you must have Bash Version 4 or higher.

How to do it...

1. An array can be defined in many ways. Define an array using a list of values in a line as follows:

   ```
   array_var=(1 2 3 4 5 6)
   #Values will be stored in consecutive locations starting from
   index 0.
   ```