We can check whether a command terminated successfully or not by using the following script:

```
#!/bin/bash
#Filename: success_test.sh
CMD="command" #Substitute with command for which you need to test the
exit status
$CMD
if [ $? -eq 0 ];
then
    echo "$CMD executed successfully"
else
    echo "$CMD terminated unsuccessfully"
fi
```

Passing arguments to commands

Arguments to commands can be passed in different formats. Suppose `-p` and `-v` are the options available and `-k N` is another option that takes a number. Also, the command takes a filename as argument. It can be executed in multiple ways as shown:

- `$ command -p -v -k 1 file`
- `$ command -pv -k 1 file`
- `$ command -vpk 1 file`
- `$ command file -pvk 1`

Reading the output of a sequence of commands in a variable

One of the best-designed features of shell scripting is the ease of combining many commands or utilities to produce output. The output of one command can appear as the input of another, which passes its output to another command, and so on. The output of this combination can be read in a variable. This recipe illustrates how to combine multiple commands and how its output can be read.

Getting ready

Input is usually fed into a command through `stdin` or arguments. Output appears as `stderr` or `stdout`. While we combine multiple commands, we usually use `stdin` to give input and `stdout` to provide an output.

In this context, the commands are called **filters**. We connect each filter using pipes, the piping operator being `|`. An example is as follows:

```
$ cmd1 | cmd2 | cmd3
```

Here we combine three commands. The output of `cmd1` goes to `cmd2` and output of `cmd2` goes to `cmd3` and the final output (which comes out of `cmd3`) will be printed, or it can be directed to a file.

How to do it...

We typically use pipes and use them with the subshell method for combining outputs of multiple files. Here's how:

1. Let us start with combining two commands:

   ```
   $ ls | cat -n > out.txt
   ```

   Here the output of `ls` (the listing of the current directory) is passed to `cat -n`, which in turn puts line numbers to the input received through `stdin`. Therefore, its output is redirected to the `out.txt` file.

2. We can read the output of a sequence of commands combined by pipes as follows:

   ```
   cmd_output=$(COMMANDS)
   ```

   This is called **subshell method**. For example:

   ```
   cmd_output=$(ls | cat -n)
   echo $cmd_output
   ```

   Another method, called **back quotes** (some people also refer to it as **back tick**) can also be used to store the command output as follows:

   ```
   cmd_output=`COMMANDS`
   ```

   For example:

   ```
   cmd_output=`ls | cat -n`
   echo $cmd_output
   ```

   Back quote is different from the single-quote character. It is the character on the ~ button in the keyboard.