There's more...

There are multiple ways of grouping commands. Let us go through a few of them.

Spawning a separate process with subshell

Subshells are separate processes. A subshell can be defined using the `( )` operators as follows:

```
pwd;
(cd /bin; ls);
pwd;
```

When some commands are executed in a subshell, none of the changes occur in the current shell; changes are restricted to the subshell. For example, when the current directory in a subshell is changed using the `cd` command, the directory change is not reflected in the main shell environment.

The `pwd` command prints the path of the working directory.

The `cd` command changes the current directory to the given directory path.

Subshell quoting to preserve spacing and the newline character

Suppose we are reading the output of a command to a variable using a subshell or the back quotes method. We always quote them in double quotes to preserve the spacing and newline character (`\n`). For example:

```
$ cat text.txt
1
2
3

$ out=$(cat text.txt)
$ echo $out
1 2 3 # Lost \n spacing in 1,2,3

$ out="$(cat tex.txt)"
$ echo$out
1
2
3
```

Reading n characters without pressing the return key

`read` is an important Bash command to read text from the keyboard or standard input. We can use `read` to interactively read an input from the user, but `read` is capable of much more. Most of the input libraries in any programming language read the input from the keyboard; but string input termination is done when *return* is pressed. There are certain critical situations when *return* cannot be pressed, but the termination is done based on a number of characters or a single character. For example, in a game, a ball is moved upward when + is pressed. Pressing + and then pressing *return* every time to acknowledge the + press is not efficient. In this recipe we will use the `read` command that provides a way to accomplish this task without having to press *return*.

How to do it...

You can use various options of the `read` command to obtain different results as shown in the following steps:

1. The following statement will read *n* characters from input into the `variable_name` variable:

   ```
   read -n number_of_chars variable_name
   ```

   For example:

   ```
   $ read -n 2 var
   $ echo $var
   ```

2. Read a password in the nonechoed mode as follows:

   ```
   read -s var
   ```

3. Display a message with `read` using:

   ```
   read -p "Enter input:"  var
   ```

4. Read the input after a timeout as follows:

   ```
   read -t timeout var
   ```

   For example:

   ```
   $ read -t 2 var
   #Read the string that is typed within 2 seconds into variable var.
   ```