To check whether the given word is a dictionary word, use the following script:

```
#!/bin/bash
#Filename: checkword.sh
word=$1
grep "^$1$" /usr/share/dict/british-english -q
if [ $? -eq 0 ]; then
  echo $word is a dictionary word;
else
  echo $word is not a dictionary word;
fi
```

The usage is as follows:

```
$ ./checkword.sh ful
ful is not a dictionary word

$ ./checkword.sh fool
fool is a dictionary word
```

How it works...

In `grep`, `^` is the word-start-marker character and the `$` character is the word-end marker.

`-q` is used to suppress any output and to be silent.

Or, alternatively, we can use the spell check, `aspell`, to check whether a word is in a dictionary or not, as follows:

```
#!/bin/bash
#Filename: aspellcheck.sh
word=$1

output=`echo \"$word\" | aspell list`

if [ -z $output ]; then
        echo $word is a dictionary word;
else
        echo $word is not a dictionary word;
fi
```

The `aspell list` command returns output text when the given input is not a dictionary word, and does not output anything when the input is a dictionary word. A `-z` command checks whether `$output` is an empty string or not.

List all words in a file starting with a given word as follows:

```
$ look word filepath
```

Or alternately, use:

```
$ grep "^word" filepath
```

By default, if the filename argument is not given to the `look` command, it looks up into the default dictionary (`/usr/share/dict/words`) and returns an output:

```
$look word
# When used like this it takes default dictionary as file
```

For example:

```
$ look android
android
android's
androids
```

Automating interactive input

Automating interactive input for command-line utilities are extremely useful for writing automation tools or testing tools. There will be many situations when we deal with commands that read input interactively. An example of executing a command and supplying the interactive input is as follows:

```
$ command
Enter a number: 1
Enter name : hello
You have entered 1,hello
```

Getting ready

Creating utilities that can automate the acceptance of input are useful to supply input to local commands, as well as for remote applications. Let us see how to automate them.

How to do it...

Think about the sequence of an interactive input. From the previous code, we can formulate the steps of the sequence as follows:

```
   1[Return]hello[Return]
```