Using awk for advanced text processing

`awk` is a tool designed to work with data streams. It is very interesting, as it can operate on columns and rows. It supports many built-in functionalities, such as arrays and functions, in the C programming language. Its biggest advantage is its flexibility.

Getting ready...

The structure of an `awk` script is as follows:

```
awk ' BEGIN{  print "start" } pattern { commands } END{ print "end" }
file
```

The `awk` command can read from `stdin` also.

An `awk` script usually consists of three parts—`BEGIN`, `END`, and a common statements block with the pattern match option. The three of them are optional and any of them can be absent in the script.

How to do it...

Let's write a simple `awk` script enclosed in single quotes or double quotes, as follows:

```
awk 'BEGIN { statements } { statements } END { end statements }'
```

Or, alternately, use the following command:

```
awk "BEGIN { statements } { statements } END { end statements }"
```

For example:

```
$ awk 'BEGIN { i=0 } { i++ } END{ print i}' filename
```

Or:

```
$ awk "BEGIN { i=0 } { i++ } END{ print i }" filename
```

How it works...

The `awk` command works in the following manner:

1. Execute the statements in the `BEGIN { commands }` block.
2. Read one line from the file or `stdin`, and execute `pattern { commands }`. Repeat this step until the end of the file is reached.
3. When the end of the input stream is reached, execute the `END { commands }` block.

The `BEGIN` block is executed before `awk` starts reading lines from the input stream. It is an optional block. The statements, such as variable initialization and printing the output header for an output table, are common statements that are written in the `BEGIN` block.

The `END` block is similar to the `BEGIN` block. It gets executed when `awk` completes reading all the lines from the input stream. The statements, such as printing results after analyzing all the values calculated for all the lines or printing the conclusion are the commonly-used statements in the `END` block (for example, after comparing all the lines, print the maximum number from a file). This is an optional block.

The most important block is of the common commands with the pattern block. This block is also optional. If this block is not provided, by default `{ print }` gets executed so as to print each of the lines read. This block gets executed for each line read by `awk`. It is like a `while` loop for lines read, with statements provided inside the body of the loop.

When a line is read, it checks whether the provided pattern matches the line. The pattern can be a regular expression match, conditions, range of lines match, and so on. If the current read line matches with the pattern, it executes the statements enclosed in `{ }`.

The pattern is optional. If it is not used, all the lines are matched and the statements inside `{ }` are executed.

Let's go through the following example:

```
$ echo -e "line1\nline2" | awk 'BEGIN{ print "Start" } { print } END{
print "End" } '
Start
line1
line2
End
```

When `print` is used without an argument, it will print the current line. There are two important things to be kept in mind about it. When the arguments of the print are separated by commas, they are printed with a space delimiter. Double quotes are used as the concatenation operator in the context of `print` in `awk`.

For example:

```
$ echo | awk '{ var1="v1"; var2="v2"; var3="v3"; \
print var1,var2,var3 ; }'
```

The preceding statement will print the values of the variables as follows:

```
v1 v2 v3
```