There is a flexible alternate method to pass many variable values from outside `awk`. For example:

```
$ var1="Variable1" ; var2="Variable2"
$ echo | awk '{ print v1,v2 }' v1=$var1 v2=$var2
Variable1 Variable2
```

When an input is given through a file rather than standard input, use the following command:

```
$ awk '{ print v1,v2 }' v1=$var1 v2=$var2 filename
```

In the preceding method, variables are specified as key-value pairs, separated by a space and `(v1=$var1 v2=$var2 )` as command arguments to `awk` soon after the `BEGIN`, `{ }`, and `END` blocks.

Reading a line explicitly using getline

Usually, `awk` reads all the lines in a file by default. If you want to read one specific line, you can use the `getline` function. Sometimes, you may need to read the first line from the `BEGIN` block.

The syntax is `getline var`. The variable `var` will contain the content for the line. If `getline` is called without an argument, we can access the content of the line by using `$0`, `$1`, and `$2`.

For example:

```
$ seq 5 | awk 'BEGIN { getline; print "Read ahead first line", $0 } {
print $0 }'
Read ahead first line 1
2
3
4
5
```

Filtering lines processed by awk with filter patterns

We can specify some conditions for lines to be processed. For example:

```
$ awk 'NR < 5' # first four lines
$ awk 'NR==1,NR==4' #First four lines
$ awk '/linux/' # Lines containing the pattern linux (we can specify
regex)
$ awk '!/linux/' # Lines not containing the pattern linux
```

Setting delimiter for fields

By default, the delimiter for fields is a space. We can explicitly specify a delimiter by using `-F "delimiter"`:

```
$ awk -F: '{ print $NF }' /etc/passwd
```

Or:

```
awk 'BEGIN { FS=":" } { print $NF }' /etc/passwd
```

We can set the output fields separator by setting `OFS="delimiter"` in the `BEGIN` block.

Reading the command output from awk

In the following code, `echo` will produce a single blank line. The `cmdout` variable will contain the output of the command `grep root /etc/passwd`, and it will print the line containing the root:

The syntax for reading out the command in a variable `output` is as follows:

```
"command" | getline output ;
```

For example:

```
$ echo | awk '{ "grep root /etc/passwd" | getline cmdout ; print cmdout
}'
root:x:0:0:root:/root:/bin/bash
```

By using `getline`, we can read the output of external shell commands in a variable called `cmdout`.

`awk` supports associative arrays, which can use the text as the index.

Using loop inside awk

A `for` loop is available in `awk`. It has the following format:

```
for(i=0;i<10;i++) { print $i ; }
```

Or:

```
for(i in array) { print array[i]; }
```