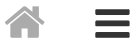


🕒 Sunday, March 05, 2017



LIKE GEEKS



LINUX

Regex Tutorial For Linux

📅 February 23, 2017 👤 admin 💬 2 Comments

In order to successfully working with the **Linux sed** editor and the **awk command** in your shell scripts you has to understand **regular expressions** or in short regex and to be accurate in our case it is bash regex, since there are many engines for regex you can use and we here in this regex tutorial will use the shell regex and see the bash power in working with regex.

First, we need to understand what regex is then we will dive deep into using it

Our main points are:

What is regex

Types of regex

Define BRE Patterns

Special characters

Anchor characters

The dot character

Character classes

Negating character classes

Using ranges

Special character classes

The asterisk

Extended Regular Expressions

Grouping expressions

Practical examples

What is regex

For some people when they see the regular expressions for the first time they said what are those ASCII pukes !! well. A regular expression or regex, in general, is a pattern of text you define that a Linux program (in our case) like sed or awk uses to filter text.

The regex pattern makes use of wildcard characters to represent one or more characters in the data stream. We've seen some of those wildcard characters when introducing basic Linux

commands and see how ls command use wildcard characters to filter output.

Types of regex

There are many different applications use different types of regex in Linux. These include programming languages (Java, Perl, Python,...) and Linux programs like (sed, awk, grep,) and many other applications

A regex is implemented using a regular expression engine. A regular expression engine is the underlying software that interprets regular expression patterns and uses those patterns to match the text.

Linux has two regular expression engines:

- The **POSIX Basic Regular Expression (BRE)** engine
- The **POSIX Extended Regular Expression (ERE)** engine

Most Linux programs at a minimum conform to the POSIX BRE engine specifications, recognizing all the pattern symbols it defines. Unfortunately, some utilities (such as the sed) conform only to a subset of the BRE engine specifications. This is due to speed constraints because the sed attempts to process text quickly as possible.

The POSIX ERE engine is often found in programming languages. It provides advanced pattern symbols as well as special symbols for common patterns, such as matching digits, and words. The awk command uses the ERE engine to process its regular expression patterns

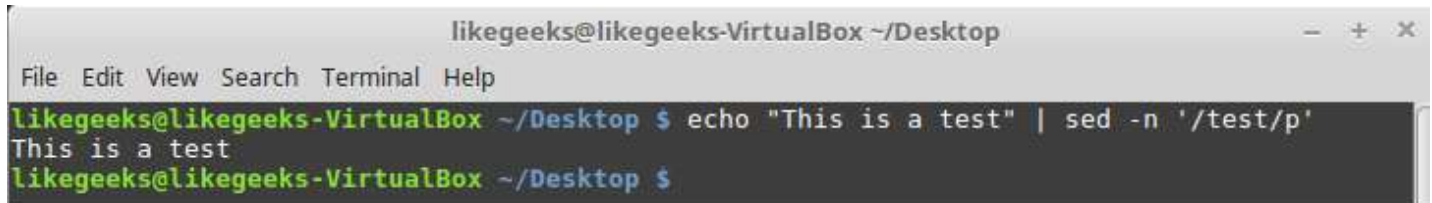
And because there are so many different ways to implement regex, it's hard to write patterns that work on all engines. Hence we will focus on the most commonly found regex and demonstrate how to use them in the sed and awk.

Define BRE Patterns

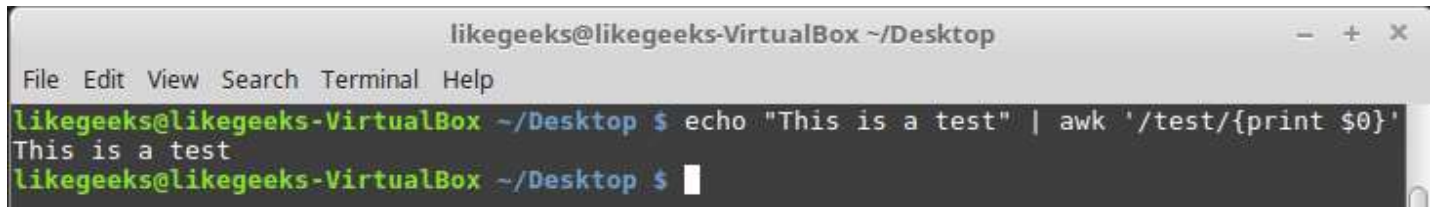
The most basic BRE pattern is matching text characters in a data stream and we've seen that using sed and awk but let's refresh our memory

```
$ echo "This is a test" | sed -n '/test/p'
```

```
$ echo "This is a test" | awk '/test/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | sed -n '/test/p'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```



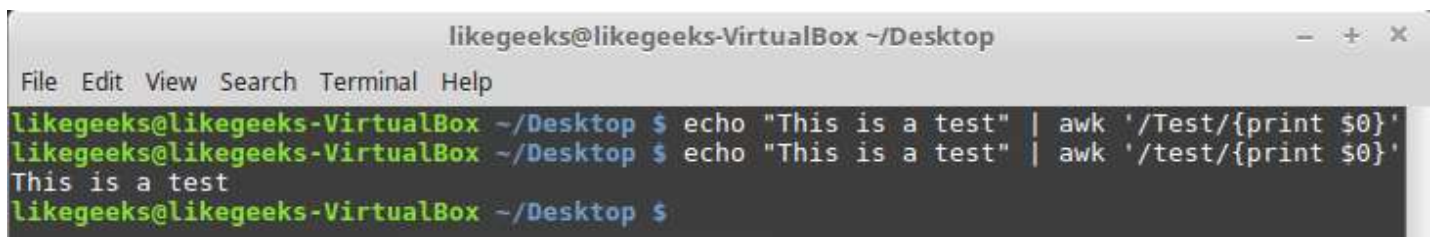
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/test/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

You may notice that the regex doesn't care where in the data stream the pattern occurs. It also doesn't matter how many times the pattern occurs. After the regex can match the pattern anywhere in the text string, it passes the string along to the Linux program that's using it.

The first rule to remember is that regular expression patterns are case sensitive.

```
$ echo "This is a test" | awk '/Test/{print $0}'
```

```
$ echo "This is a test" | awk '/test/{print $0}'
```

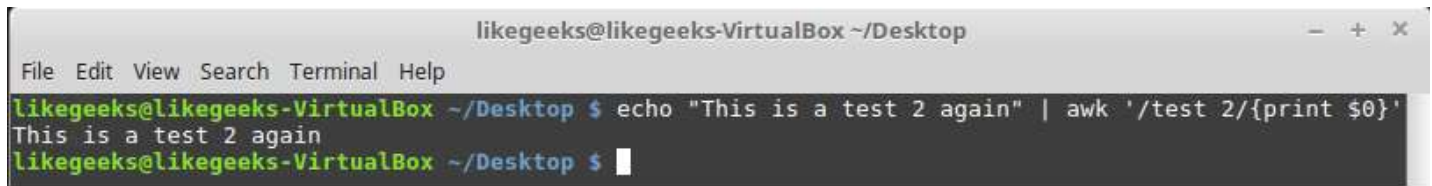


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/Test/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/test/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The first regex found no match because the word "test" doesn't appear in uppercase in the text string, while the second line, which uses the lowercase letter in the pattern, worked just fine

You also don't have to limit yourself to single text words in the regular expression. You can include spaces and numbers in your text string as well

```
$ echo "This is a test 2 again" | awk '/test 2/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test 2 again" | awk '/test 2/{print $0}'
This is a test 2 again
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Spaces are treated just like any other character in regex.

Special characters

There are a few exceptions when defining text characters in a regex.

regex patterns assign a special meaning to a few characters. If you try to use these characters in your text pattern, you won't get the results you were expecting

These special characters are recognized by regex

```
. * [ ] ^ $ { } \ + ? | ( )
```

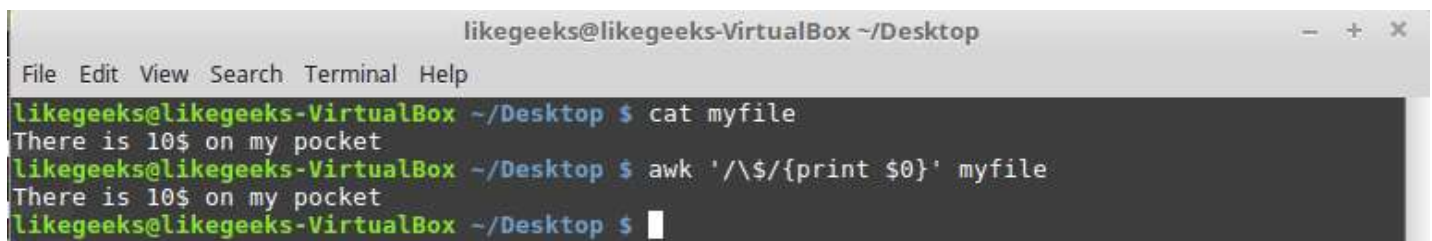
If you want to use one of the special characters as a text character, you need to escape it

The special character that does this is the backslash character (\).

For example, if you want to search for a dollar sign in your text, just precede it with a backslash character like this

```
$ cat myfile
There is 10$ on my pocket
```

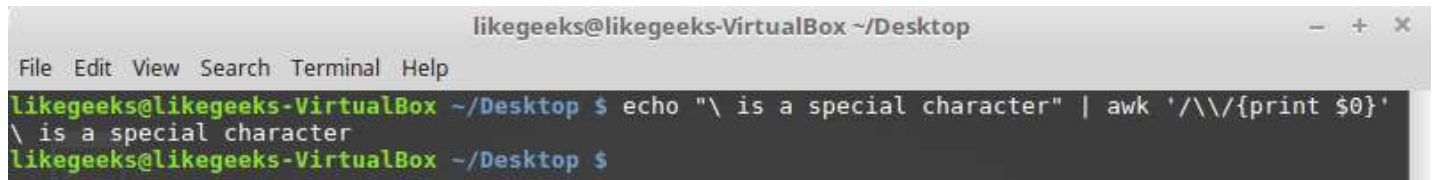
```
$ awk '/\$/ {print $0}' myfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
There is 10$ on my pocket
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/\$/ {print $0}' myfile
There is 10$ on my pocket
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Also, backslash itself is a special character, if you need to use it in a regex pattern, you need to escape it as well, producing a double backslash

```
$ echo "\ is a special character" | awk '/\\/ {print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "\ is a special character" | awk '/\\/ {print $0}'
\ is a special character
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Although the forward slash isn't a regular expression special character, if you use it in your regular expression pattern in the sed or the awk, you get an error

```
$ echo "3 / 2" | awk '/// {print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "3 / 2" | awk '/// {print $0}'
awk: cmd. line:1: /// {print $0}
awk: cmd. line:1: ^ syntax error
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

So you need to escape it like this

```
$ echo "3 / 2" | awk '/\/ {print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "3 / 2" | awk '/\/ {print $0}'
3 / 2
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Anchor characters

You can use two special characters to anchor a pattern to either the beginning or the end of lines in the text

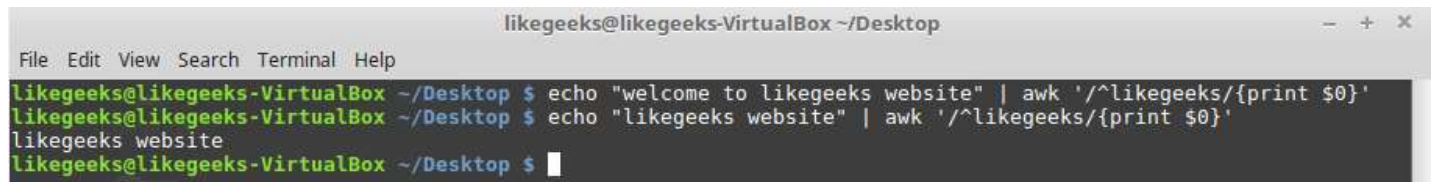
The caret character (^) defines a pattern that starts at the beginning of a line of text in the text

If the pattern is located any place other than the start of the line of text, the regex pattern fails

You can use it like this

```
$ echo "welcome to likegeeks website" | awk '/^likegeeks/{print $0}'
```

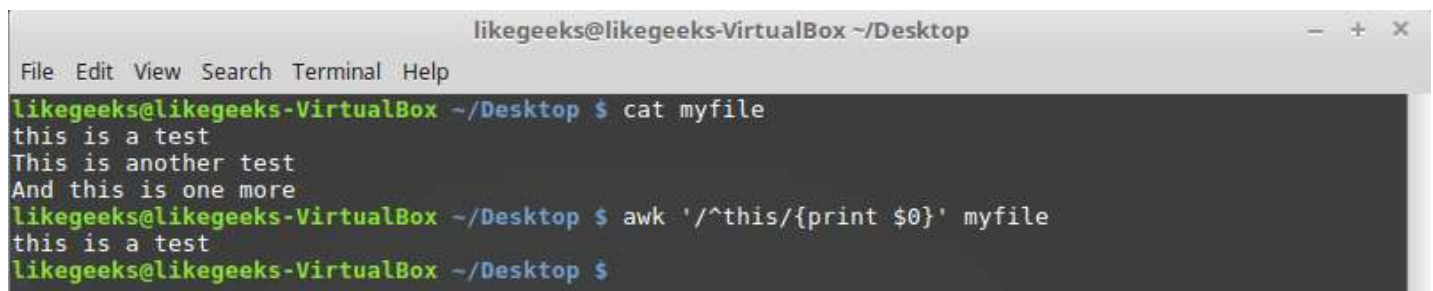
```
$ echo "likegeeks website" | awk '/^likegeeks/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "welcome to likegeeks website" | awk '/^likegeeks/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "likegeeks website" | awk '/^likegeeks/{print $0}'
likegeeks website
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The caret anchor character checks for the pattern at the beginning of each new line of data

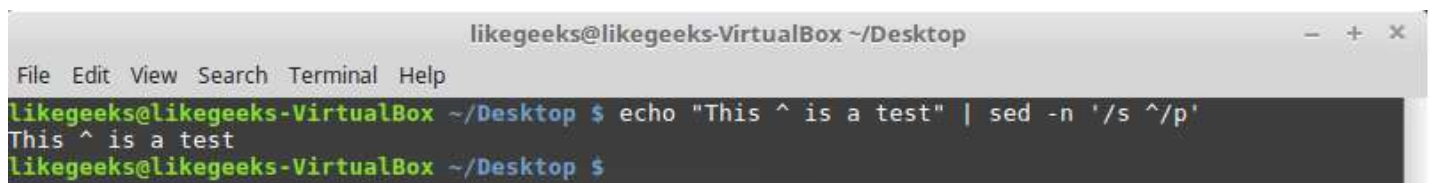
```
$ awk '/^this/{print $0}' myfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/^this/{print $0}' myfile
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Great!! When using sed, if you position the caret character in any place other than at the beginning of the pattern, it acts like a normal character and not as a special character

```
$ echo "This ^ is a test" | sed -n '/s ^/p'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This ^ is a test" | sed -n '/s ^/p'
This ^ is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

But if you use awk you have to escape it like this

```
$ echo "This ^ is a test" | awk '/s \^/{print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This ^ is a test" | awk '/s \^/{print $0}'
This ^ is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

This about looking at the beginning of text what about looking at the end

The dollar sign (\$) special character defines the end anchor

```
$ echo "This is a test" | awk '/test$/{print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/test$/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

You can combine both the start and end anchor on the same line like this

```
$ cat myfile
this is a test
This is another test
And this is one more
```

```
$ awk '/^this is a test$/{print $0}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/^this is a test$/{print $0}' myfile
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

As you can see it prints only the line that has the matching pattern only

You can filter blank lines with the following pattern

```
$ awk '!/^$/{print $0}' myfile
```

Here we introduce the negation which is done by the exclamation mark !

The pattern looks for lines that have nothing between the start and end of the line and negates that to print only the lines have text.

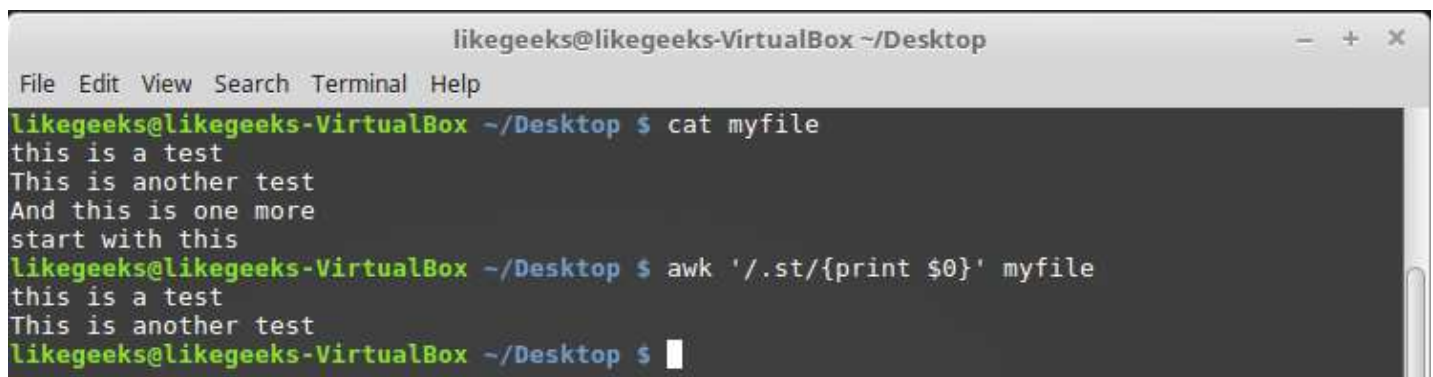
The dot character

The dot character is used to match any single character except a newline character

Look at the following example to get the idea

```
$ cat myfile  
this is a test  
This is another test  
And this is one more  
start with this
```

```
$ awk '/.st/{print $0}' myfile
```



The screenshot shows a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal displays the output of the 'cat myfile' command, which is the same text as in the previous code block. Below that, the 'awk '/.st/{print \$0}' myfile' command is executed, resulting in only the first two lines of the file being printed: 'this is a test' and 'This is another test'. The prompt '\$' is visible at the end of the line.

You can see from the result that it prints only the first two lines because they contain the st pattern while the third line does not have that pattern and fourth line start with st so that also doesn't match our pattern.

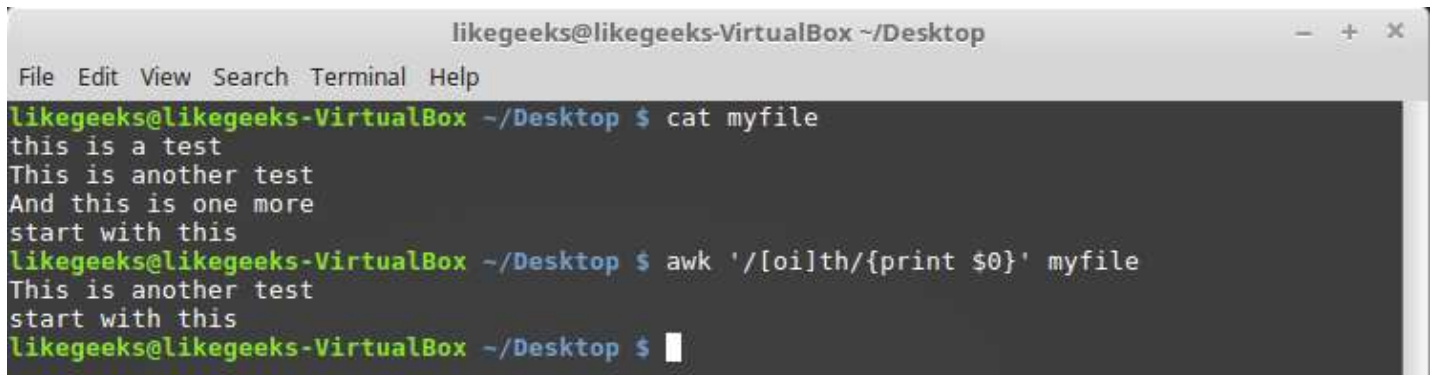
Character classes

You can match any character with the dot special character but what if you want to limit what characters to match. This is called a character class

You can define a set of characters that would match a position in a text pattern. If one of the characters from the character set is in the text, it matches the pattern

To define a character class, you use square brackets [] like this

```
$ awk '/[oi]th/{print $0}' myfile
```



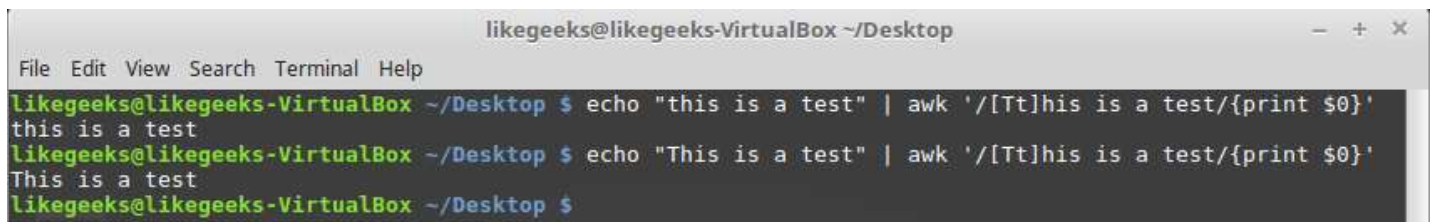
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[oi]th/{print $0}' myfile
This is another test
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here we search for any th character that has o character or l before it.

This comes handy when you are searching for words that may contain upper or lower case and you are not sure about that.

```
$ echo "this is a test" | awk '/[Tt]his is a test/{print $0}'
```

```
$ echo "This is a test" | awk '/[Tt]his is a test/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "this is a test" | awk '/[Tt]his is a test/{print $0}'
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/[Tt]his is a test/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Of course, it is not limited to characters; you can use numbers or whatever you want. You can employ it as you want as long as you got the idea.

Negating character classes

You can also reverse the effect of a character class. Instead of looking for a character contained in the class, you can look for any character that's not in the class. To do that, just place a caret character at the beginning of the character class range.

```
$ awk '/[^oi]th/{print $0}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[^oi]th/{print $0}' myfile
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

By negating the character class, the regex pattern matches any character that's neither o nor an i.

Using ranges

You can use a range of characters within a character class by using the dash symbol like this.

```
$ awk '/[e-p]st/{print $0}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[e-p]st/{print $0}' myfile
this is a test
This is another test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

This matches all characters between e and p then followed by st as shown

You can also use ranges for numbers

```
$ echo "123" | awk '/[0-9][0-9][0-9]/'
```

```
$ echo "12a" | awk '/[0-9][0-9][0-9]/'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "123" | awk '/[0-9][0-9][0-9]/'
123
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "12a" | awk '/[0-9][0-9][0-9]/'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

You can also specify multiple, non-continuous ranges in a single character class

```
$ awk '/[a-fm-z]st/{print $0}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[a-fm-z]st/{print $0}' myfile
this is a test
This is another test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The character class allows the ranges a through f, and m through z to appear before the st text.

Special character classes

The BRE contains special character classes you can use to match against specific types of characters

And this is the list

<code>[[:alpha:]]</code>	Matches any alphabetical character, either upper or lower case
<code>[[:alnum:]]</code>	Matches any alphanumeric character 0–9, A–Z, or a–z
<code>[[:blank:]]</code>	Matches a space or Tab character
<code>[[:digit:]]</code>	Matches a numerical digit from 0 through 9
<code>[[:lower:]]</code>	Matches any lowercase alphabetical character a–z

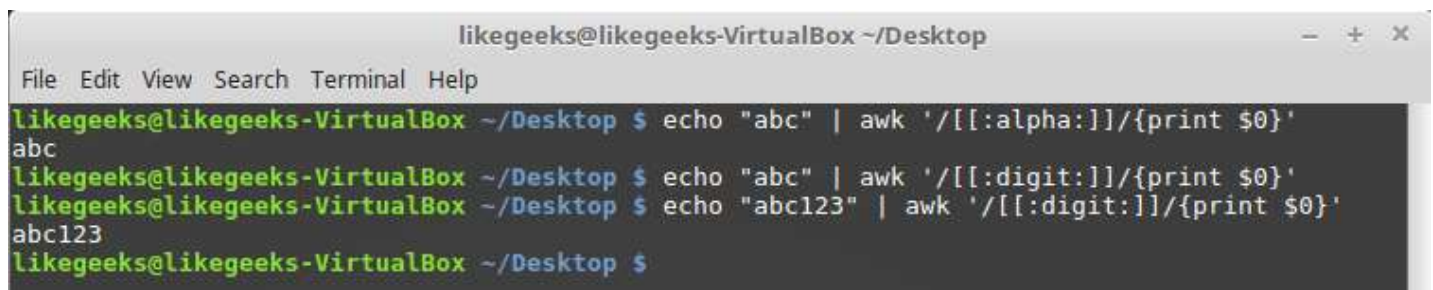
<code>[:print:]</code>	Matches any printable character
<code>[:punct:]</code>	Matches a punctuation character
<code>[:space:]</code>	Matches any whitespace character: space, Tab, NL, FF, VT, CR
<code>[:upper:]</code>	Matches any uppercase alphabetical character A–Z

You can use them like this

```
$ echo "abc" | awk '/[[:alpha:]]/{print $0}'
```

```
$ echo "abc" | awk '/[[:digit:]]/{print $0}'
```

```
$ echo "abc123" | awk '/[[:digit:]]/{print $0}'
```



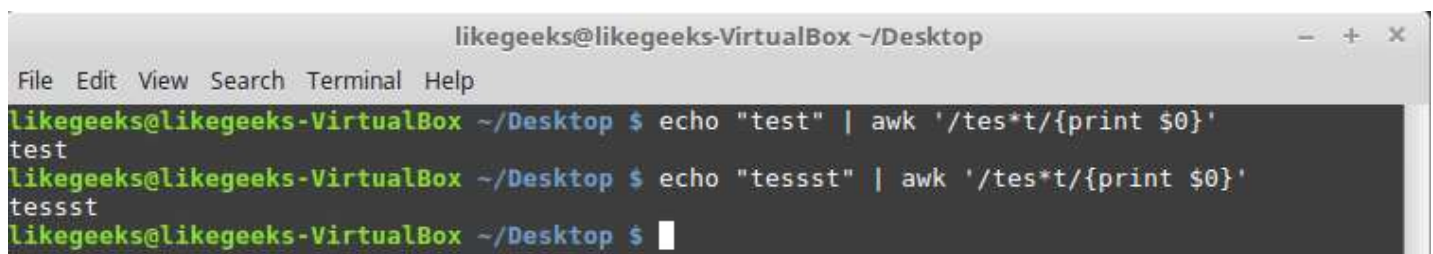
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "abc" | awk '/[[:alpha:]]/{print $0}'
abc
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "abc" | awk '/[[:digit:]]/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "abc123" | awk '/[[:digit:]]/{print $0}'
abc123
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The asterisk

Placing an asterisk after a character signifies that the character must appear zero or more times in the text to match the pattern

```
$ echo "test" | awk '/tes*t/{print $0}'
```

```
$ echo "tessst" | awk '/tes*t/{print $0}'
```

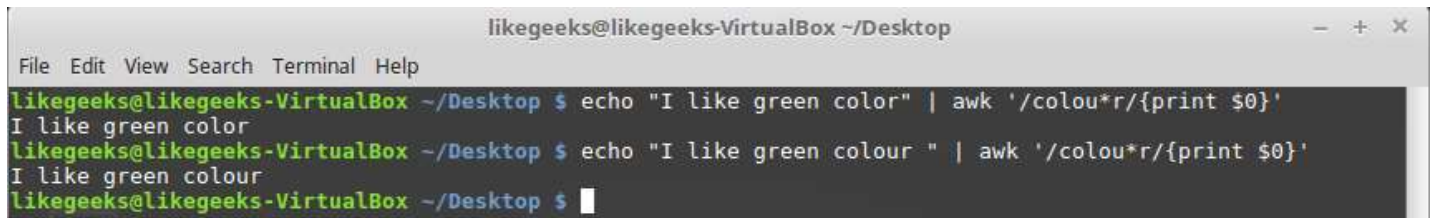


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/tes*t/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tessst" | awk '/tes*t/{print $0}'
tessst
likegeeks@likegeeks-VirtualBox ~/Desktop $
```


This pattern symbol is commonly used for handling words that have a common misspelling or variations in language spellings

```
$ echo "I like green color" | awk '/colou*r/{print $0}'
```

```
$ echo "I like green colour " | awk '/colou*r/{print $0}'
```

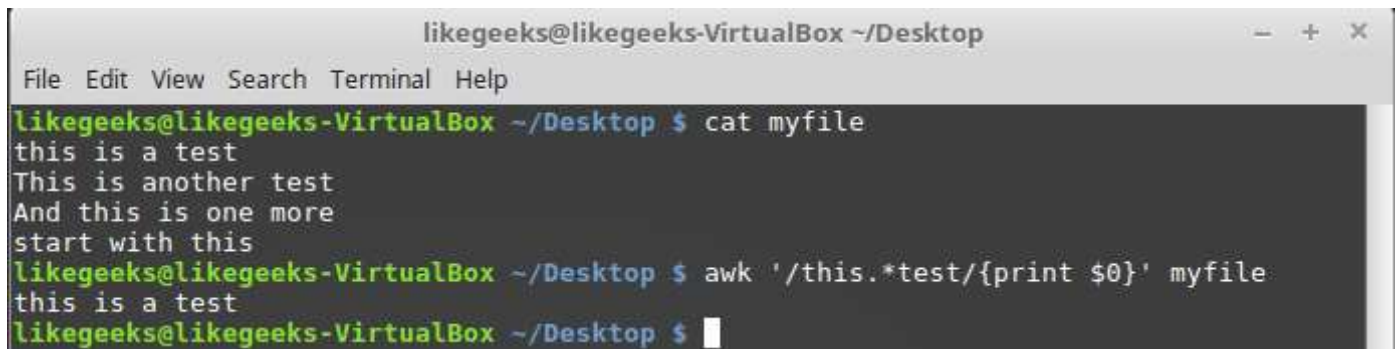


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "I like green color" | awk '/colou*r/{print $0}'
I like green color
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "I like green colour " | awk '/colou*r/{print $0}'
I like green colour
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Here in those examples whether you type it color or colour it will match because the asterisk means if the u character existed many time or zero time that will match.

Another handy feature is combining the dot character with the asterisk character. This combination provides a pattern to match any number of any characters.

```
$ awk '/this.*test/{print $0}' myfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/this.*test/{print $0}' myfile
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

It doesn't matter how many words between the words this and test, any line will match will be printed.

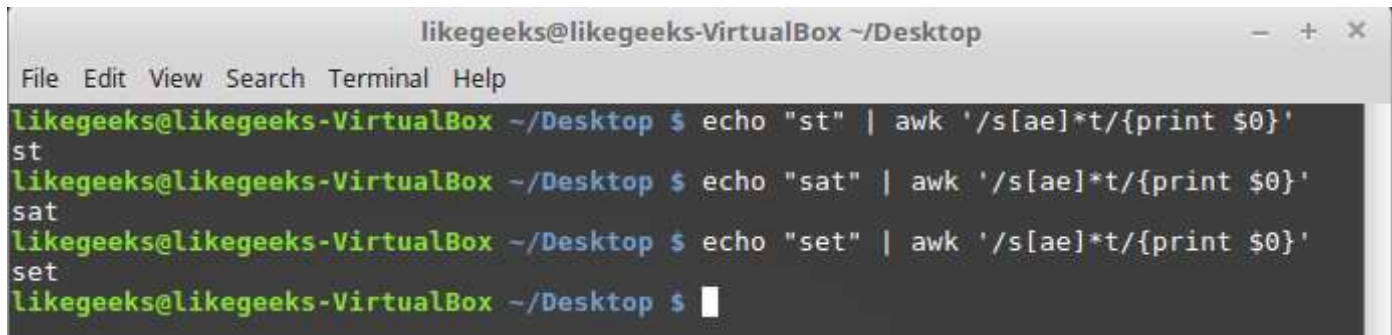
The asterisk can also be applied to a character class.

```
$ echo "st" | awk '/s[ae]*t/{print $0}'
```

```
$ echo "sat" | awk '/s[ae]*t/{print $0}'
```



```
$ echo "set" | awk '/s[ae]*t/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "st" | awk '/s[ae]*t/{print $0}'
st
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "sat" | awk '/s[ae]*t/{print $0}'
sat
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "set" | awk '/s[ae]*t/{print $0}'
set
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

All three examples match because the asterisk means if you find zero times or more of a character or e print it.

Extended Regular Expressions

The POSIX ERE patterns include a few additional symbols that are used by some Linux applications and utilities. The awk command recognizes the ERE patterns, but sed doesn't.

We will discuss the commonly used ERE pattern symbols that you can use in your awk program scripts.

The question mark

The question mark indicates that the preceding character can appear zero or one time so no repeating here

```
$ echo "tet" | awk '/tes?t/{print $0}'
```

```
$ echo "test" | awk '/tes?t/{print $0}'
```

```
$ echo "tesst" | awk '/tes?t/{print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tet" | awk '/tes?t/{print $0}'
tet
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/tes?t/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tesst" | awk '/tes?t/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

You can use the question mark symbol along with a character class

```
$ echo "tst" | awk '/t[ae]?st/{print $0}'
```

```
$ echo "test" | awk '/t[ae]?st/{print $0}'
```

```
$ echo "tast" | awk '/t[ae]?st/{print $0}'
```

```
$ echo "taest" | awk '/t[ae]?st/{print $0}'
```

```
$ echo "teest" | awk '/t[ae]?st/{print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/t[ae]?st/{print $0}'
tst
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/t[ae]?st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tast" | awk '/t[ae]?st/{print $0}'
tast
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "taest" | awk '/t[ae]?st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teest" | awk '/t[ae]?st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

If zero or one character from the character class appears, the pattern match passes.

But if both characters appear, or if one of the characters appears twice, the pattern match fails

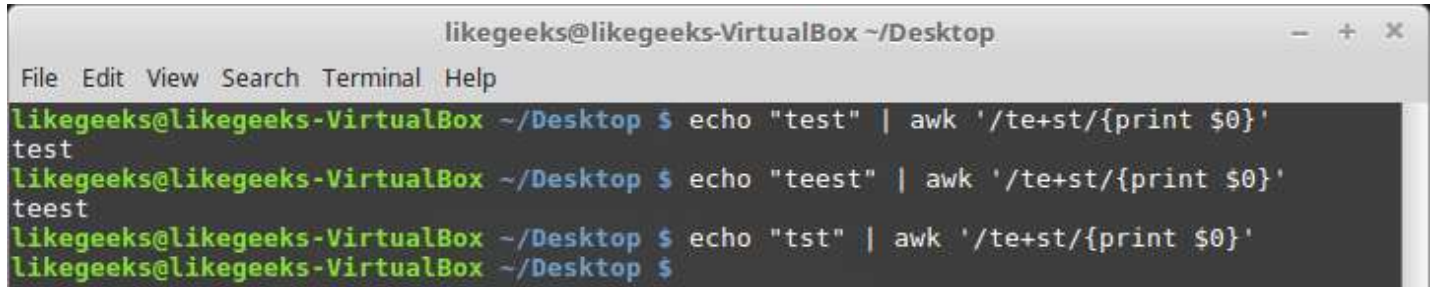
The plus sign

The plus sign indicates that the preceding character can appear one or more times, but must be present at least once

```
$ echo "test" | awk '/te+st/{print $0}'
```

```
$ echo "teest" | awk '/te+st/{print $0}'
```

```
$ echo "tst" | awk '/te+st/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/te+st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teest" | awk '/te+st/{print $0}'
teest
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/te+st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

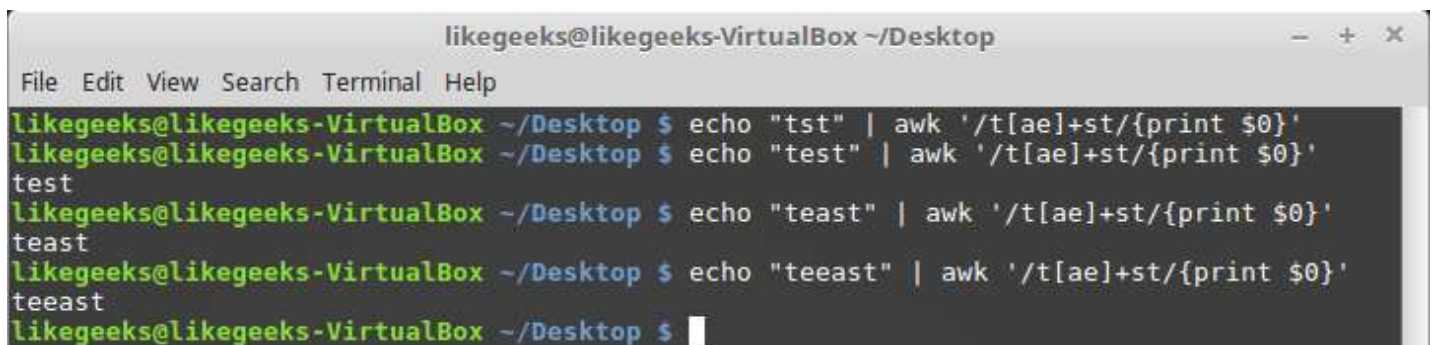
If the e character is not present, the pattern match fails. The plus sign also works with character classes, the same way as the asterisk and question mark

```
$ echo "tst" | awk '/t[ae]+st/{print $0}'
```

```
$ echo "test" | awk '/t[ae]+st/{print $0}'
```

```
$ echo "teast" | awk '/t[ae]+st/{print $0}'
```

```
$ echo "teeast" | awk '/t[ae]+st/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/t[ae]+st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/t[ae]+st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teast" | awk '/t[ae]+st/{print $0}'
teast
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teeast" | awk '/t[ae]+st/{print $0}'
teeast
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

This time if either character defined in the character class appears, the text matches the specified pattern.

Curly braces

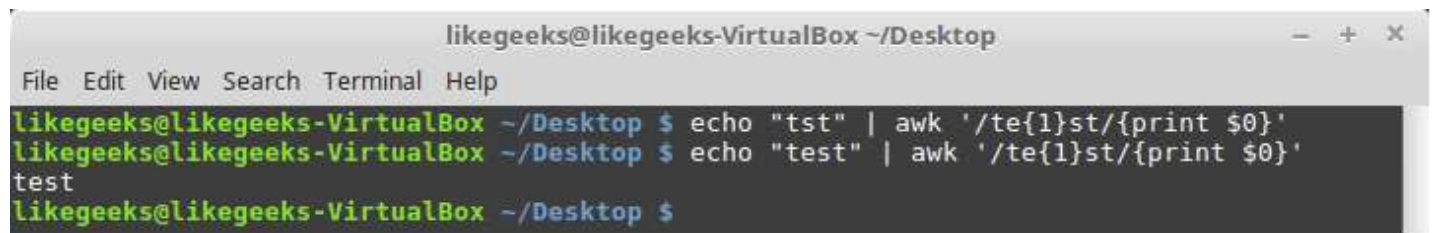
Curly braces are available in ERE to allow you to specify a limit on a repeatable regex, it has two formats

n: The regex appears exactly n times.

n,m: The regex appears at least n times, but no more than m times

```
$ echo "tst" | awk '/te{1}st/{print $0}'
```

```
$ echo "test" | awk '/te{1}st/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/te{1}st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/te{1}st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

In old versions of awk, you should use `-re-interval` command line option for the awk command to recognize regular expression intervals but now you don't need it

```
$ echo "tst" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "test" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "teest" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "teeest" | awk '/te{1,2}st/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/te{1,2}st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/te{1,2}st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teest" | awk '/te{1,2}st/{print $0}'
teest
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teeest" | awk '/te{1,2}st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

In this example, the e character can appear once or twice for the pattern match to pass; otherwise, the pattern match fails

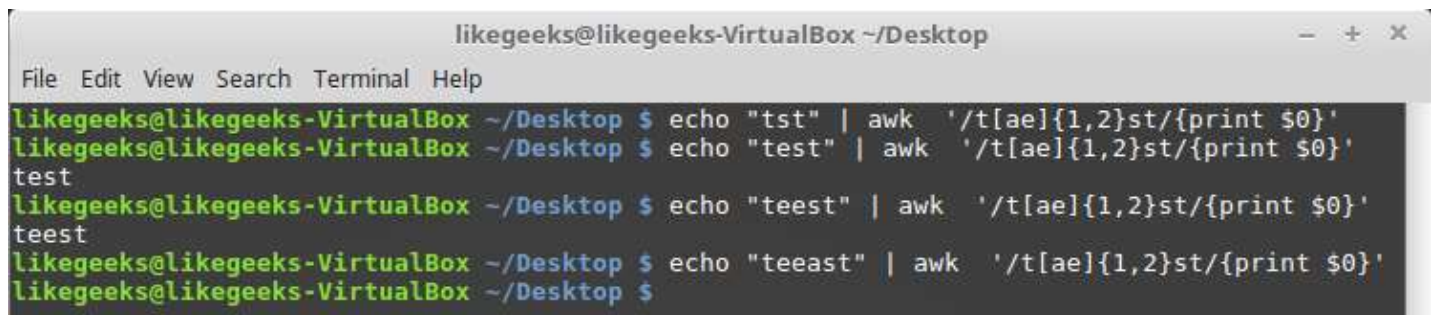
The interval pattern match also applies to character classes the same way as we did so we don't have to do it again

```
$ echo "tst" | awk '/t[ae]{1,2}st/{print $0}'
```

```
$ echo "test" | awk '/t[ae]{1,2}st/{print $0}'
```

```
$ echo "teest" | awk '/t[ae]{1,2}st/{print $0}'
```

```
$ echo "teeast" | awk '/t[ae]{1,2}st/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tst" | awk '/t[ae]{1,2}st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/t[ae]{1,2}st/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teest" | awk '/t[ae]{1,2}st/{print $0}'
teest
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "teeast" | awk '/t[ae]{1,2}st/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

This regex pattern matches if there are exactly one or two instances of the letter a or e in the text pattern, but it fails if there are any more in any combination

The pipe symbol

The pipe symbol allows to you to specify two or more patterns that the regex engine uses in a logical OR formula when examining the data stream. If any of the patterns match the text, the text passes. If none of the patterns matches, the pattern will fail, here is an example

```
$ echo "This is a test" | awk '/test|exam/{print $0}'
```

```
$ echo "This is an exam" | awk '/test|exam/{print $0}'
```

```
$ echo "This is something else" | awk '/test|exam/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/test|exam/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is an exam" | awk '/test|exam/{print $0}'
This is an exam
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is something else" | awk '/test|exam/{print $0}'
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

This example looks for the regular expression test or exam in the text. Keep in mind that you can't place any spaces within the regular expressions and the pipe symbol.

Grouping expressions

Regex patterns can also be grouped by using parentheses. When you group a regex pattern, the group is treated like a standard character. You can apply a special character to the group just as you would to a regular character

```
$ echo "Like" | awk '/Like(Geeks)?/{print $0}'
```

```
$ echo "LikeGeeks" | awk '/Like(Geeks)?/{print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "Like" | awk '/Like(Geeks)?/{print $0}'
Like
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "LikeGeeks" | awk '/Like(Geeks)?/{print $0}'
LikeGeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

The grouping of the "Geeks" ending along with the question mark allows the pattern to match either the full day name LikeGeeks or the word Like only.

Practical examples

We've seen some simple demonstrations of using regular expression patterns, it's time to put that in action, just for practicing and I always say more practicing more power

Counting directory files

Let's look at a bash script that counts the executable files that are present in the directories defined in your PATH environment variable. To do that, you need to parse out the PATH variable into separate directory names.

```
$ echo $PATH
```

To get a listing of directories that you can use in a script, you must replace each colon with space.

```
$ echo $PATH | sed 's/:/ /g'
```

Now let's iterate through each directory using for loop like this

```
mypath=$(echo $PATH | sed 's/:/ /g')
for directory in $mypath
do
done
```

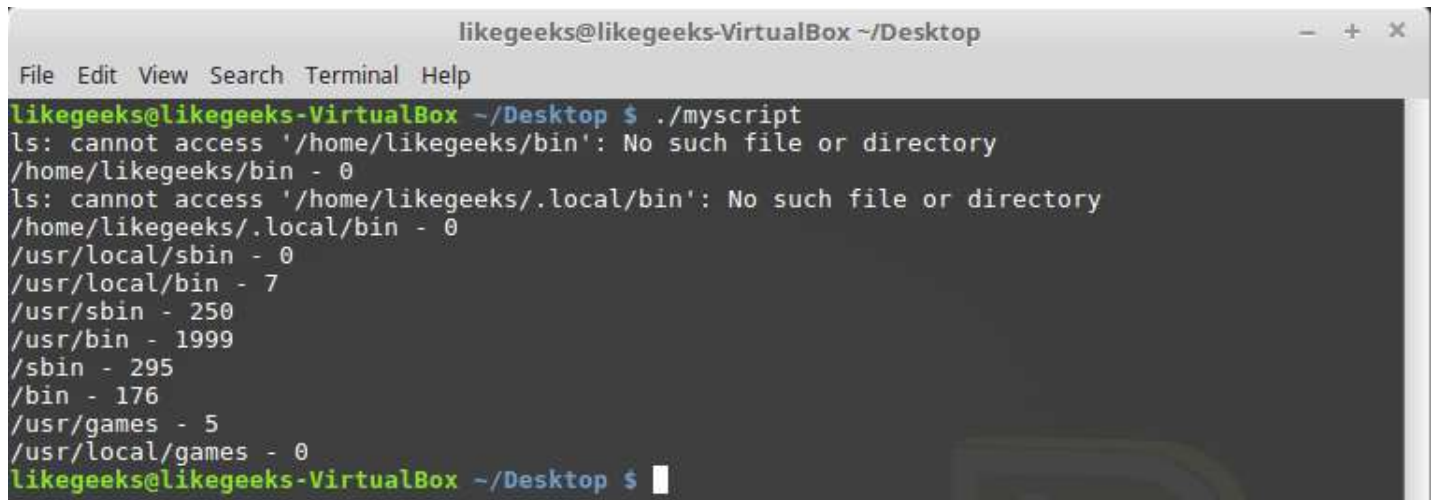
Great!!

Now we can use the ls command to list each file in each directory and save the count in a variable

```
#!/bin/bash
mypath=$(echo $PATH | sed 's/:/ /g')
count=0
for directory in $mypath
do
check=$(ls $directory)
for item in $check
do
count=$((count + 1))
done
```

```
echo "$directory - $count"
count=0
done
```

You may notice some directory not existed, no problem with this



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
ls: cannot access '/home/likegeeks/bin': No such file or directory
/home/likegeeks/bin - 0
ls: cannot access '/home/likegeeks/.local/bin': No such file or directory
/home/likegeeks/.local/bin - 0
/usr/local/sbin - 0
/usr/local/bin - 7
/usr/sbin - 250
/usr/bin - 1999
/sbin - 295
/bin - 176
/usr/games - 5
/usr/local/games - 0
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Cool!! This is the power of regex. Those few lines of code count all files in all directories. Of course, there is a Linux command to do that very easy but here we introduce how to employ regex in something you can use and with some brain ideas you can come up with some more useful.

Validating e-mail address

There are a ton of websites that offers read to use regex patterns for everything e-mail, phone number and much more this is handy but we want to understand how it works

username@hostname.com

The username can use any alphanumeric characters combined with dot, dash, plus sign, underscore

The hostname can use any alphanumeric characters combined with dot and underscore

Let's start building the regular expression pattern from the left side. We know that there can be multiple valid characters in the username. This should be very easy

```
^([a-zA-Z0-9_\-\.]+)@
```

This grouping specifies the allowed characters in the username and the plus sign to indicate that at least one character must be present or more then the @ sign

Then the hostname pattern should be like this

```
([a-zA-Z0-9_\-\.]+)
```

There are special rules for the top-level domain. Top-level domains are only alphabetic characters, and they must be no fewer than two characters (used in country codes) and no more than five characters in length. The following is the regex pattern for the top-level domain

```
\.([a-zA-Z]{2,5})$
```

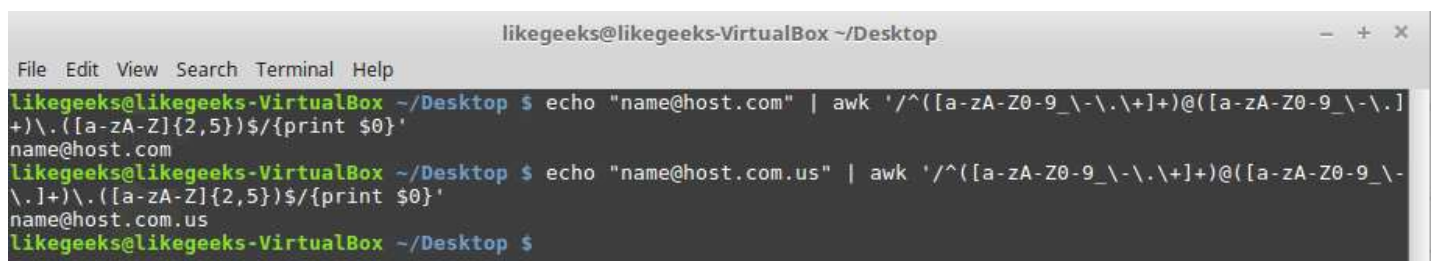
Now we put them all together

```
^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$
```

Let's test that regex against an email

```
$ echo "name@host.com" | awk '/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/ {print $0}'
```

```
$ echo "name@host.com.us" | awk '/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/ {print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "name@host.com" | awk '/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/ {print $0}'
name@host.com
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "name@host.com.us" | awk '/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/ {print $0}'
name@host.com.us
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Awesome!! Works great

This was just the beginning for regex world that never ends I hope after this post you understand these ASCII pukes J and use it more professionally

This is for now, hope you like the post

Thank you.

16

Admin

<https://likegeeks.com>

RELATED ARTICLES



LINUX

Linux bash scripting the awesome guide part3

📅 February 12, 2017 👤 admin

So far you've seen how to write Linux bash scripts that interact with data, variables, and files and how to control the flow of the bash script. Today we will continue our



LINUX

Shell scripting the awesome guide part4

📅 February 13, 2017 👤 admin

On the previous post we've talked about parameters and options in detail and today we will talk about something is very important in shell scripting which is input &



LINUX

Expect command and how to automate shell scripts like magic

📅 February 27, 2017 👤 admin

In the previous post we've talked about writing practical shell scripts and we've seen how it is easy to write a shell script and we've used most of

series about Linux bash scripting. I recommend you to review the previous posts if you want to know what we are talking about. [...]

output & redirection. So far, you've seen two methods for displaying the output from your shell scripts Displaying output on the screen Redirecting output to a [...]

our knowledge we've discussed on the previous posts, today we are going to talk about a tool that does magic to our shell scripts, that tool is expect command [...]

10

9

7

[◀ 30 Examples for awk command in text processing](#)[How to write practical shell script ▶](#)**2 Comments** **likegeeks****1 Login** ▼ **Recommend**  **Share****Sort by Best** ▼

Join the discussion...

killaklik • 8 days ago




Please correct " The first regex found no match because the word "this" doesn't appear in uppercase in the text string, while the second line, which uses the lowercase letter in the pattern, worked just fine " it should read the word "Test". Keep up the good work !

1 ^ | v • Reply • Share ›

likegeeks Mod ➔ killaklik • 8 days ago

Sorry for this mistake I've corrected it
Thanks for you attention.

^ | v • Reply • Share ›

 **Subscribe**  **Add Disqus to your site** **Add Disqus** **Add**  **Privacy**

SEARCH

Search ...

Search