

Bash-скрипты, часть 9: регулярные выражения

<https://likegeeks.com/regex-tutorial-linux/>

- Блог компании RUVDS.com,
- Настройка Linux,
- Системное администрирование
- [Перевод](#)

[Bash-скрипты: начало](#)

[Bash-скрипты, часть 2: циклы](#)

[Bash-скрипты, часть 3: параметры и ключи командной строки](#)

[Bash-скрипты, часть 4: ввод и вывод](#)

[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](#)

[Bash-скрипты, часть 6: функции и разработка библиотек](#)

[Bash-скрипты, часть 7: sed и обработка текстов](#)

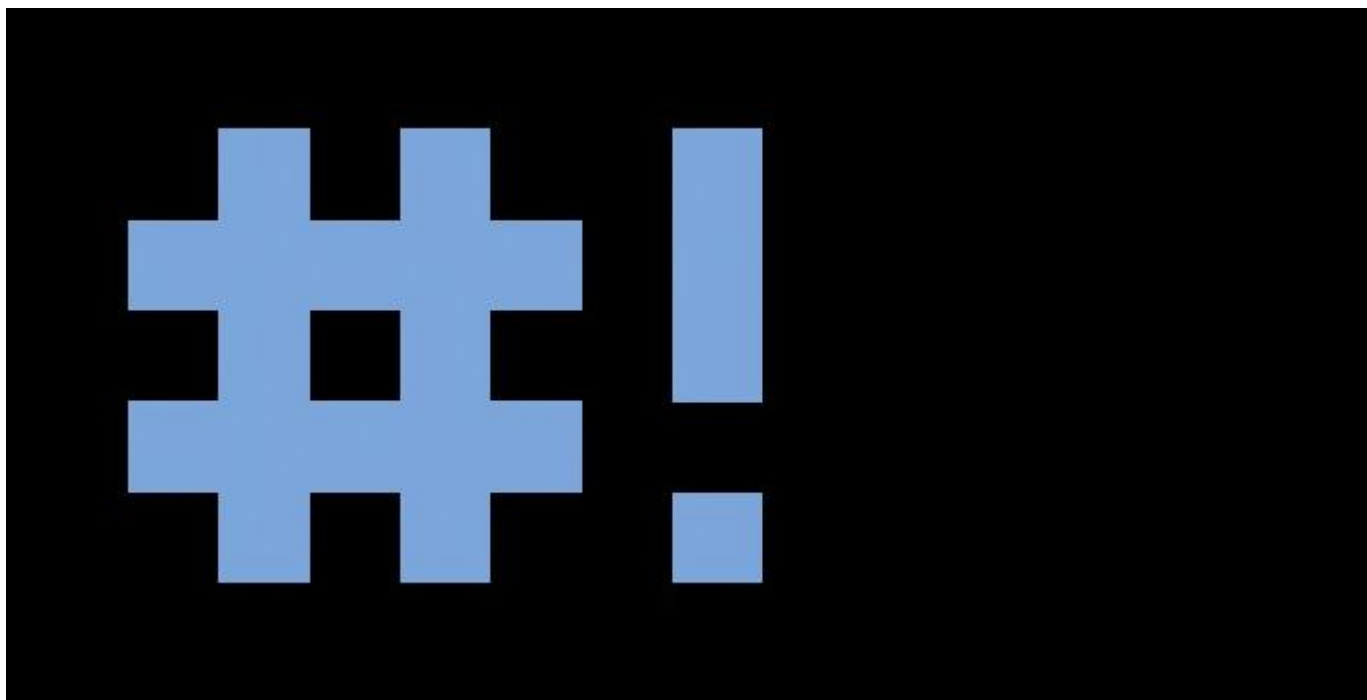
[Bash-скрипты, часть 8: язык обработки данных awk](#)

[Bash-скрипты, часть 9: регулярные выражения](#)

[Bash-скрипты, часть 10: практические примеры](#)

[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](#)

Для того, чтобы полноценно обрабатывать тексты в bash-скриптах с помощью sed и awk, просто необходимо разобраться с регулярными выражениями. Реализации этого полезнейшего инструмента можно найти буквально повсюду, и хотя устроены все регулярные выражения схожим образом, основаны на одних и тех же идеях, в разных средах работа с ними имеет определённые особенности. Тут мы поговорим о регулярных выражениях, которые подходят для использования в сценариях командной строки Linux.



Этот материал задуман как введение в регулярные выражения, рассчитанное на

тех, кто может совершенно не знать о том, что это такое. Поэтому начнём с самого начала.

Habrhabr10

Промо-код для скидки в 10% на наши виртуалы

Что такое регулярные выражения

У многих, когда они впервые видят регулярные выражения, сразу же возникает мысль, что перед ними бессмысленное нагромождение символов. Но это, конечно, далеко не так. Взгляните, например, на это регулярное выражение

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

На наш взгляд даже абсолютный новичок сходу поймёт, как оно устроено и зачем нужно :) Если же вам не вполне понятно — просто читайте дальше и всё встанет на свои места.

Регулярное выражение — это шаблон, пользуясь которым программы вроде `sed` или `awk` фильтруют тексты. В шаблонах используются обычные ASCII-символы, представляющие сами себя, и так называемые метасимволы, которые играют особую роль, например, позволяя ссылаться на некие группы символов.

Типы регулярных выражений

Реализации регулярных выражений в различных средах, например, в языках программирования вроде `Java`, `Perl` и `Python`, в инструментах `Linux` вроде `sed`, `awk` и `grep`, имеют определённые особенности. Эти особенности зависят от так называемых движков обработки регулярных выражений, которые занимаются интерпретацией шаблонов.

В `Linux` имеется два движка регулярных выражений:

- Движок, поддерживающий стандарт POSIX Basic Regular Expression (BRE).
- Движок, поддерживающий стандарт POSIX Extended Regular Expression (ERE).

Большинство утилит `Linux` соответствуют, как минимум, стандарту POSIX BRE, но некоторые утилиты (в их числе — `sed`) понимают лишь некое подмножество стандарта BRE. Одна из причин такого ограничения — стремление сделать такие утилиты как можно более быстрыми в деле обработки текстов.

Стандарт POSIX ERE часто реализуют в языках программирования. Он позволяет пользоваться большим количеством средств при разработке регулярных выражений. Например, это могут быть специальные последовательности символов для часто используемых шаблонов, вроде поиска в тексте отдельных

слов или наборов цифр. Awk поддерживает стандарт ERE.

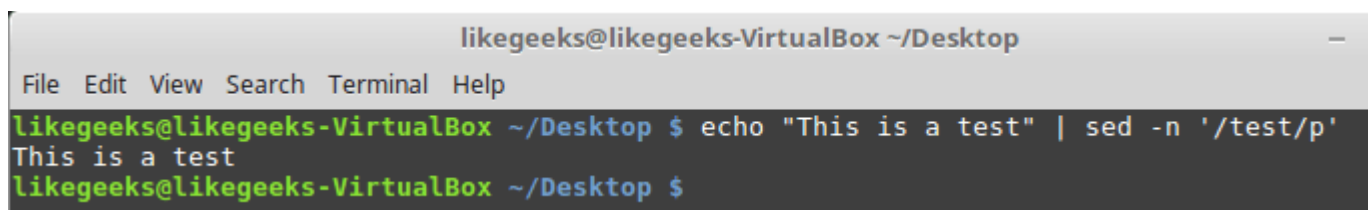
Существует много способов разработки регулярных выражений, зависящих и от мнения программиста, и от особенностей движка, под который их создают. Непросто писать универсальные регулярные выражения, которые сможет понять любой движок. Поэтому мы сосредоточимся на наиболее часто используемых регулярных выражениях и рассмотрим особенности их реализации для sed и awk.

Регулярные выражения POSIX BRE

Пожалуй, самый простой шаблон BRE представляет собой регулярное выражение для поиска точного вхождения последовательности символов в тексте. Вот как выглядит поиск строки в sed и awk:

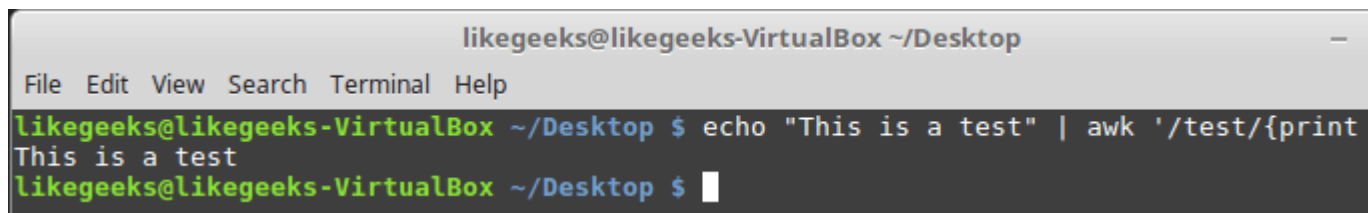
```
$ echo "This is a test" | sed -n '/test/p'
```

```
$ 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 $
```

Поиск текста по шаблону в sed



```
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 $
```

Поиск текста по шаблону в awk

Можно заметить, что поиск заданного шаблона выполняется без учёта точного места нахождения текста в строке. Кроме того, не имеет значение и количество вхождений. После того, как регулярное выражение найдёт заданный текст в любом месте строки, строка считается подходящей и передаётся для дальнейшей обработки.

Работая с регулярными выражениями нужно учитывать то, что они чувствительны к регистру символов:

```
$ 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 $
```

Регулярные выражения чувствительны к регистру

Первое регулярное выражение совпадений не нашло, так как слово «test», начинающееся с заглавной буквы, в тексте не встречается. Второе же, настроенное на поиск слова, написанного прописными буквами, обнаружило в потоке подходящую строку.

В регулярных выражениях можно использовать не только буквы, но и пробелы, и цифры:

```
$ 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 $
```

Поиск фрагмента текста, содержащего пробелы и цифры

Пробелы воспринимаются движком регулярных выражений как обычные символы.

Специальные символы

При использовании различных символов в регулярных выражениях надо учитывать некоторые особенности. Так, существуют некоторые специальные символы, или метасимволы, использование которых в шаблоне требует особого подхода. Вот они:

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

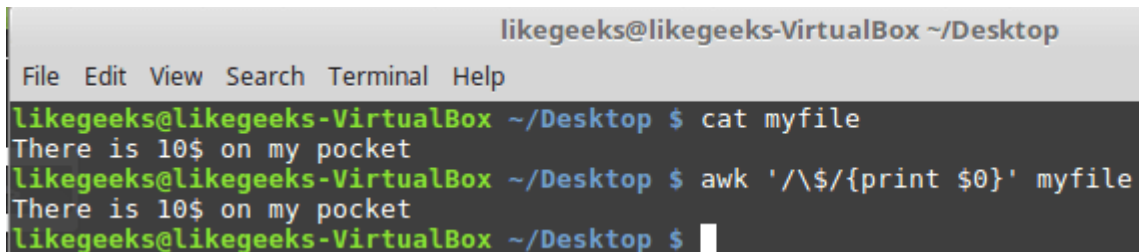
Если один из них нужен в шаблоне, его нужно будет экранировать с помощью обратной косой черты (обратного слэша) — \.

Например, если в тексте нужно найти знак доллара, его надо включить в шаблон, предварив символом экранирования. Скажем, имеется файл `myfile` с таким текстом:

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

Знак доллара можно обнаружить с помощью такого шаблона:

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

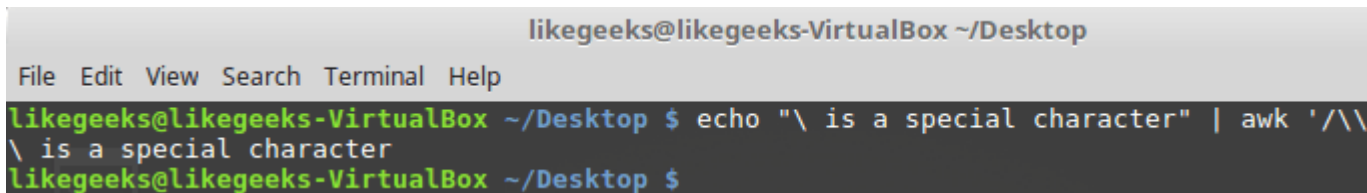


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
There is 10$ on my pocket
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/\$/ {print $0}' myfile
There is 10$ on my pocket
likegeeks@likegeeks-VirtualBox ~/Desktop $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

Использование в шаблоне специального символа

Кроме того, обратная косая черта — это тоже специальный символ, поэтому, если нужно использовать его в шаблоне, его тоже надо будет экранировать. Выглядит это как два слэша, идущих друг за другом:

```
$ 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 $
```

Экранирование обратного слэша

Хотя прямой слэш и не входит в приведённый выше список специальных символов, попытка воспользоваться им в регулярном выражении, написанном для `sed` или `awk`, приведёт к ошибке:

```
$ 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 $
```

Неправильное использование прямого слэша в шаблоне

Если он нужен, его тоже надо экранировать:

```
$ 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 $
```

Экранирование прямого слэша

Якорные символы

Существуют два специальных символа для привязки шаблона к началу или к концу текстовой строки. Символ «крышка» — ^ позволяет описывать последовательности символов, которые находятся в начале текстовых строк. Если искомый шаблон окажется в другом месте строки, регулярное выражение на него не отреагирует. Выглядит использование этого символа так:

```
$ 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 $
```

Поиск шаблона в начале строки

Символ ^ предназначен для поиска шаблона в начале строки, при этом регистр символов так же учитывается. Посмотрим, как это отразится на обработке текстового файла:

```
$ 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 $
```

Поиск шаблона в начале строки в тексте из файла

При использовании sed, если поместить крышку где-нибудь внутри шаблона, она будет восприниматься как любой другой обычный символ:

```
$ 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 $
```

Крышка, находящаяся не в начале шаблона в sed

В awk, при использовании такого же шаблона, данный символ надо экранировать:

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

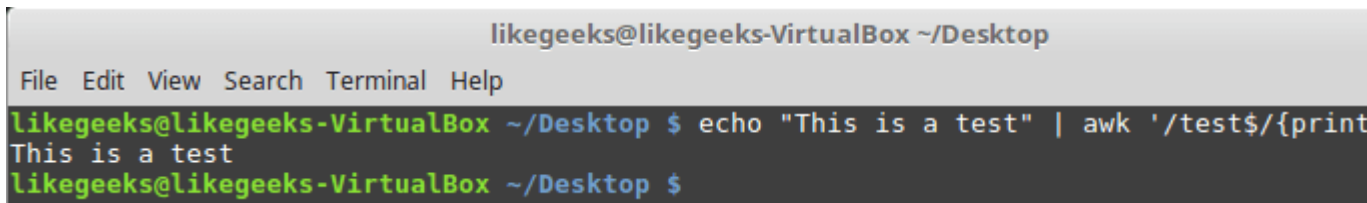
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This ^ is a test" | awk '/s \^/{prin
This ^ is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Крышка, находящаяся не в начале шаблона в awk

С поиском фрагментов текста, находящихся в начале строки мы разобрались. Что, если надо найти нечто, расположенное в конце строки?

В этом нам поможет знак доллара — \$, являющийся якорным символом конца строки:

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

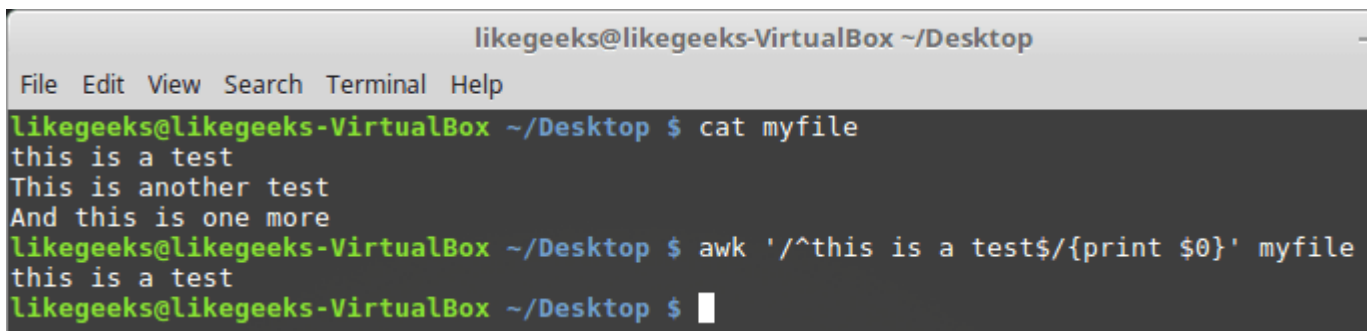


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/test$/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Поиск текста, находящегося в конце строки

В одном и том же шаблоне можно использовать оба якорных символа. Выполним обработку файла `myfile`, содержимое которого показано на рисунке ниже, с помощью такого регулярного выражения:

```
$ 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 $
```

Шаблон, в котором использованы специальные символы начала и конца строки

Как видно, шаблон среагировал лишь на строку, полностью соответствующую заданной последовательности символов и их расположению.

Вот как, пользуясь якорными символами, отфильтровать пустые строки:

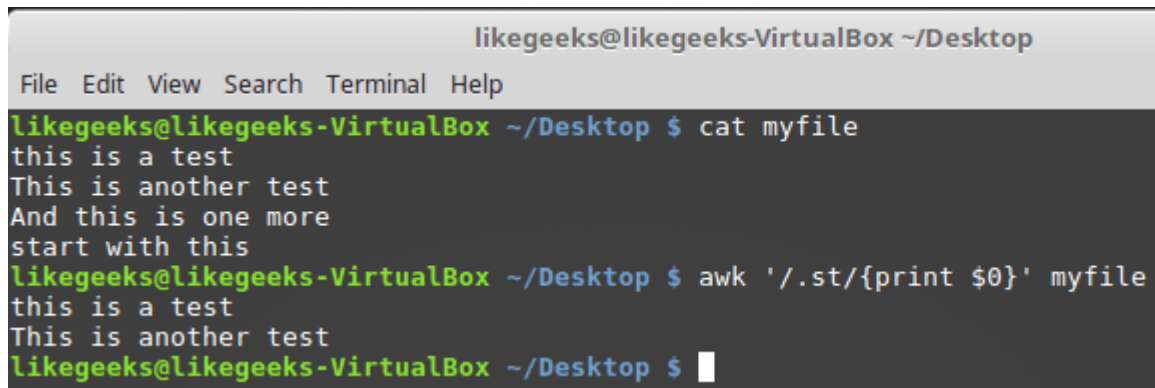
```
$ awk '!/^$/{print $0}' myfile
```


В данном шаблоне использовал символ отрицания, восклицательный знак — `!`. Благодаря использованию такого шаблона выполняется поиск строк, не содержащих ничего между началом и концом строки, а благодаря восклицательному знаку на печать выводятся лишь строки, которые не соответствуют этому шаблону.

Символ «точка»

Точка используется для поиска любого одиночного символа, за исключением символа перевода строки. Передадим такому регулярному выражению файл `myfile`, содержимое которого приведено ниже:

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



The screenshot shows a terminal window titled "likegeeks@likegeeks-VirtualBox ~/Desktop". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The prompt is "likegeeks@likegeeks-VirtualBox ~/Desktop \$". The user enters "cat myfile", and the output is:

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

The user then enters "awk '/.st/{print \$0}' myfile", and the output is:

```
this is a test
This is another test
```

The prompt returns to "likegeeks@likegeeks-VirtualBox ~/Desktop \$".

Использование точки в регулярных выражениях

Как видно по выведенным данным, шаблону соответствуют лишь первые две строки из файла, так как они содержат последовательность символов «`st`», предварённую ещё одним символом, в то время как третья строка подходящей последовательности не содержит, а в четвёртой она есть, но находится в самом начале строки.

Классы символов

Точка соответствует любому одиночному символу, но что если нужно более гибко ограничить набор искомых символов? В подобной ситуации можно воспользоваться классами символов.

Благодаря такому подходу можно организовать поиск любого символа из заданного набора. Для описания класса символов используются квадратные скобки — `[]`:

```
$ 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 $
```

Описание класса символов в регулярном выражении

Тут мы ищем последовательность символов «th», перед которой есть символ «o» или символ «i».

Классы оказываются очень кстати, если выполняется поиск слов, которые могут начинаться как с прописной, так и со строчной буквы:

```
$ 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
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/[Tt]his is a
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Поиск слов, которые могут начинаться со строчной или прописной буквы

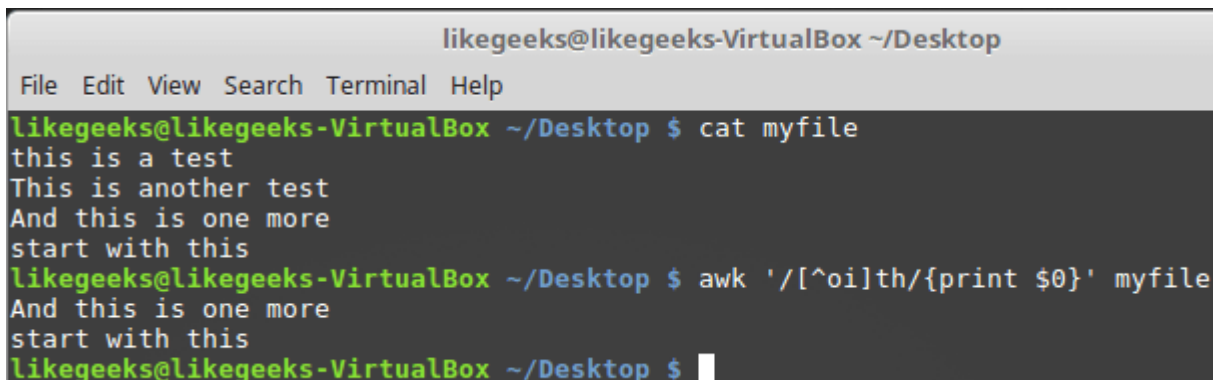
Классы символов не ограничены буквами. Тут можно использовать и другие символы. Нельзя заранее сказать, в какой ситуации понадобятся классы — всё зависит от решаемой задачи.

Отрицание классов символов

Классы символов можно использовать и для решения задачи, обратной описанной выше. А именно, вместо поиска символов, входящих в класс, можно организовать поиск всего, что в класс не входит. Для того, чтобы добиться такого

поведения регулярного выражения, перед списком символов класса нужно поместить знак ^. Выглядит это так:

```
$ 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 $
```

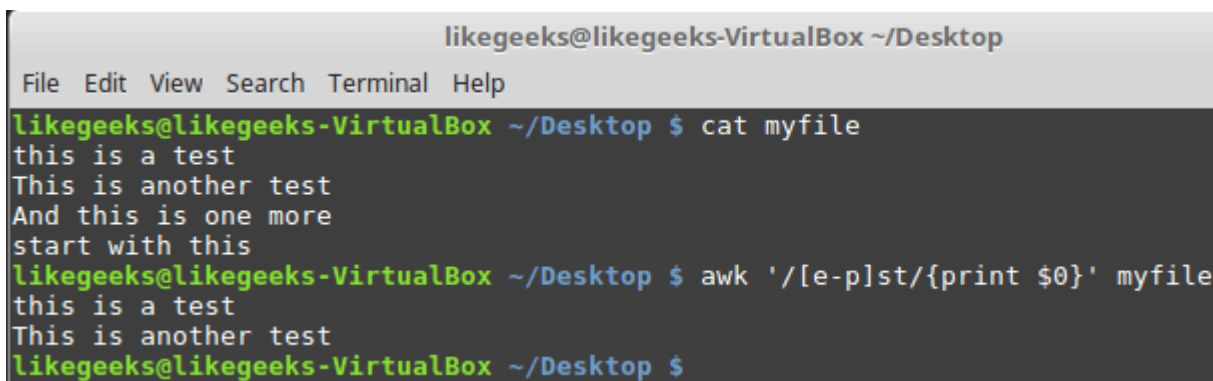
Поиск символов, не входящих в класс

В данном случае будут найдены последовательности символов «th», перед которыми нет ни «o», ни «i».

Диапазоны символов

В символьных классах можно описывать диапазоны символов, используя тире:

```
$ 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 $
```

Описание диапазона символов в символьном классе

В данном примере регулярное выражение реагирует на последовательность символов «st», перед которой находится любой символ, расположенный, в алфавитном порядке, между символами «e» и «p».

Диапазоны можно создавать и из чисел:

```
$ 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 $
```

Регулярное выражение для поиска трёх любых чисел

В класс символов могут входить несколько диапазонов:

```
$ 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 $
```

Класс символов, состоящий из нескольких диапазонов

Данное регулярное выражение найдёт все последовательности «st», перед которыми есть символы из диапазонов a-f и m-z.

Специальные классы символов

В BRE имеются специальные классы символов, которые можно использовать при написании регулярных выражений:

- `[[alpha:]]` — соответствует любому алфавитному символу, записанному в верхнем или нижнем регистре.

- `[[[:alnum:]]]` — соответствует любому алфавитно-цифровому символу, а именно — символам в диапазонах 0-9, A-Z, a-z.
- `[[[:blank:]]]` — соответствует пробелу и знаку табуляции.
- `[[[:digit:]]]` — любой цифровой символ от 0 до 9.
- `[[[:upper:]]]` — алфавитные символы в верхнем регистре — A-Z.
- `[[[:lower:]]]` — алфавитные символы в нижнем регистре — a-z.
- `[[[:print:]]]` — соответствует любому печатаемому символу.
- `[[[:punct:]]]` — соответствует знакам препинания.
- `[[[:space:]]]` — пробельные символы, в частности — пробел, знак табуляции, символы NL, FF, VT, CR.

Использовать специальные классы в шаблонах можно так:

```
$ 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 $
```

Специальные классы символов в регулярных выражениях

Символ «звёздочка»

Если в шаблоне после символа поместить звёздочку, это будет означать, что регулярное выражение работает, если символ появляется в строке любое количество раз — включая и ситуацию, когда символ в строке отсутствует.

```
$ 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 $
```

*Использование символа * в регулярных выражениях*

Этот шаблонный символ обычно используют для работы со словами, в которых постоянно встречаются опечатки, или для слов, допускающих разные варианты корректного написания:

```
$ 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/'
I like green color
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "I like green colour " | awk '/colou*'
I like green colour
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Поиск слова, имеющего разные варианты написания

В этом примере одно и то же регулярное выражение реагирует и на слово «color», и на слово «colour». Это так благодаря тому, что символ «u», после которого стоит звёздочка, может либо отсутствовать, либо встречаться несколько раз подряд.

Ещё одна полезная возможность, вытекающая из особенностей символа звёздочки, заключается в комбинировании его с точкой. Такая комбинация позволяет регулярному выражению реагировать на любое количество любых символов:

```
$ 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 $
```

Шаблон, реагирующий на любое количество любых символов

В данном случае неважно сколько и каких символов находится между словами «this» и «test».

Звёздочку можно использовать и с классами символов:

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

```
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 $
```

Использование звёздочки с классами символов

Во всех трёх примерах регулярное выражение срабатывает, так как звёздочка после класса символов означает, что если будет найдено любое количество символов «а» или «е», а также если их найти не удастся, строка будет соответствовать заданному шаблону.

Регулярные выражения POSIX ERE

Шаблоны стандарта POSIX ERE, которые поддерживают некоторые утилиты Linux, могут содержать дополнительные символы. Как уже было сказано, awk поддерживает этот стандарт, а вот sed — нет.

Тут мы рассмотрим наиболее часто используемые в ERE-шаблонах символы, которые пригодятся вам при создании собственных регулярных выражений.

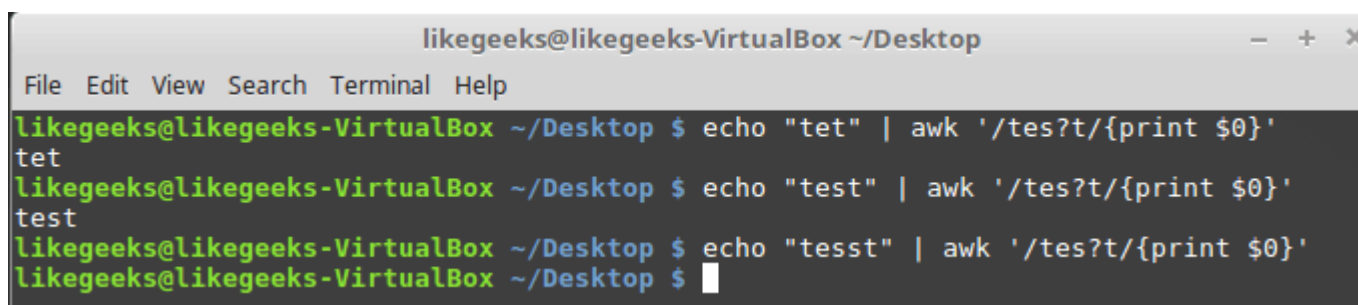
| Вопросительный знак

Вопросительный знак указывает на то, что предшествующий символ может встретиться в тексте один раз или не встретиться вовсе. Этот символ — один из метасимволов повторений. Вот несколько примеров:

```
$ 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 $
```

Вопросительный знак в регулярных выражениях

Как видно, в третьем случае буква «s» встречается дважды, поэтому на слово «tesst» регулярное выражение не реагирует.

Вопросительный знак можно использовать и с классами символов:

```
$ 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 $
```

Вопросительный знак и классы символов

Если символов из класса в строке нет, или один из них встречается один раз, регулярное выражение срабатывает, однако стоит в слове появиться двум символам и система уже не находит в тексте соответствия шаблону.

Символ «плюс»

Символ «плюс» в шаблоне указывает на то, что регулярное выражение обнаружит искомое в том случае, если предшествующий символ встретится в тексте один или более раз. При этом на отсутствие символа такая конструкция реагировать не будет:

```
$ 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 $
```

Символ «плюс» в регулярных выражениях

В данном примере, если символа «е» в слове нет, движок регулярных выражений не найдёт в тексте соответствий шаблону. Символ «плюс» работает и с классами символов — этим он похож на звёздочку и вопросительный знак:

```
$ 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 $
```

Знак «плюс» и классы символов

В данном случае если в строке имеется любой символ из класса, текст будет сочтён соответствующим шаблону.

Фигурные скобки

Фигурные скобки, которыми можно пользоваться в ERE-шаблонах, похожи на символы, рассмотренные выше, но они позволяют точнее задавать необходимое число вхождений предшествующего им символа. Указывать ограничение можно в двух форматах:

- n — число, задающее точное число искомых вхождений
- n, m — два числа, которые трактуются так: «как минимум n раз, но не больше чем m ».

Вот примеры первого варианта:

```
$ 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 $
```

Фигурные скобки в шаблонах, поиск точного числа вхождений

В старых версиях awk нужно было использовать ключ командной строки `--re-interval` для того, чтобы программа распознавала интервалы в регулярных выражениях, но в новых версиях этого делать не нужно.

```
$ 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 $
```

Интервал, заданный в фигурных скобках

В данном примере символ «e» должен встретиться в строке 1 или 2 раза, тогда регулярное выражение отреагирует на текст.

Фигурные скобки можно применять и с классами символов. Тут действуют уже знакомые вам принципы:

```
$ 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 $
```

Фигурные скобки и классы символов

Шаблон отреагирует на текст в том случае, если в нём один или два раза встретится символ «а» или символ «е».

Символ логического «или»

Символ `|` — вертикальная черта, означает в регулярных выражениях логическое «или». Обработывая регулярное выражение, содержащее несколько фрагментов, разделённых таким знаком, движок сочтёт анализируемый текст подходящим в том случае, если он будет соответствовать любому из фрагментов. Вот пример:

```
$ 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 $
```

Логическое «или» в регулярных выражениях

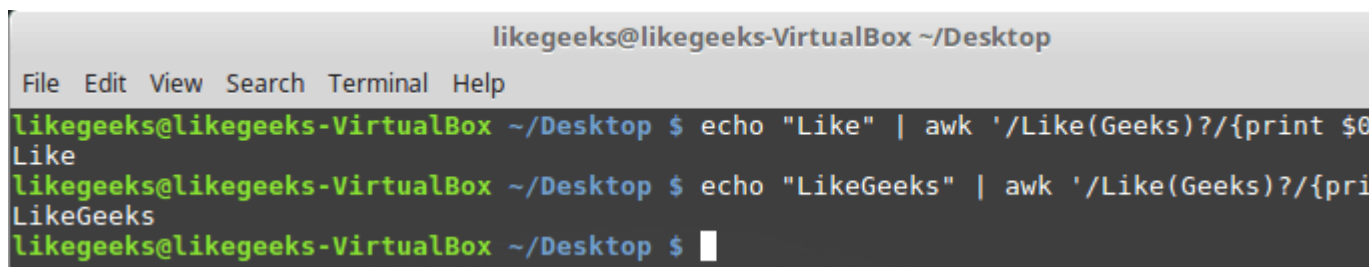
В данном примере регулярное выражение настроено на поиск в тексте слов «test» или «ехат». Обратите внимание на то, что между фрагментами шаблона и разделяющим их символом | не должно быть пробелов.

Группировка фрагментов регулярных выражений

Фрагменты регулярных выражений можно группировать, пользуясь круглыми скобками. Если сгруппировать некую последовательность символов, она будет восприниматься системой как обычный символ. То есть, например, к ней можно будет применить метасимволы повторений. Вот как это выглядит:

```
$ 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 $
```

Группировка фрагментов регулярных выражений

В данных примерах слово «Geeks» заключено в круглые скобки, после этой конструкции идёт знак вопроса. Напомним, что вопросительный знак означает «0 или 1 повторение», в результате регулярное выражение отреагирует и на строку «Like», и на строку «LikeGeeks».

Практические примеры

После того, как мы разобрали основы регулярных выражений, пришло время сделать с их помощью что-нибудь полезное.

Подсчёт количества файлов

Напишем bash-скрипт, который подсчитывает файлы, находящиеся в директориях, которые записаны в переменную окружения `PATH`. Для того, чтобы это сделать, понадобится, для начала, сформировать список путей к директориям. Сделаем это с помощью `sed`, заменив двоеточия на пробелы:

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

Команда замены поддерживает регулярные выражения в качестве шаблонов для поиска текста. В данном случае всё предельно просто, ищем мы символ двоеточия, но никто не мешает использовать здесь и что-нибудь другое — всё зависит от конкретной задачи.

Теперь надо пройти по полученному списку в цикле и выполнить там необходимые для подсчёта количества файлов действия. Общая схема скрипта будет такой:

```
mypath=$(echo $PATH | sed 's:/:/g')
```

```
for directory in $mypath
```

```
do
```

```
done
```

Теперь напомним полный текст скрипта, воспользовавшись командой `ls` для получения сведений о количестве файлов в каждой из директорий:

```
#!/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
```

При запуске скрипта может оказаться, что некоторых директорий из `PATH` не существует, однако, это не мешает ему посчитать файлы в существующих директориях.

```
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 $
```

Подсчёт файлов

Главная ценность этого примера заключается в том, что пользуясь тем же подходом, можно решать и куда более сложные задачи. Какие именно — зависит от ваших потребностей.

Проверка адресов электронной почты

Существуют веб-сайты с огромными коллекциями регулярных выражений, которые позволяют проверять адреса электронной почты, телефонные номера, и так далее. Однако, одно дело — взять готовое, и совсем другое — создать что-то самому. Поэтому напишем регулярное выражение для проверки адресов электронной почты. Начнём с анализа исходных данных. Вот, например, некий

адрес:

```
username@hostname.com
```

Имя пользователя, `username`, может состоять из алфавитно-цифровых и некоторых других символов. А именно, это точка, тире, символ подчёркивания, знак «плюс». За именем пользователя следует знак `@`.

Вооружившись этими знаниями, начнём сборку регулярного выражения с его левой части, которая служит для проверки имени пользователя. Вот что у нас получилось:

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

Это регулярное выражение можно прочесть так: «В начале строки должен быть как минимум один символ из тех, которые имеются в группе, заданной в квадратных скобках, а после этого должен идти знак `@`».

Теперь — очередь имени хоста — `hostname`. Тут применимы те же правила, что и для имени пользователя, поэтому шаблон для него будет выглядеть так:

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

Имя домена верхнего уровня подчиняется особым правилам. Тут могут быть лишь алфавитные символы, которых должно быть не меньше двух (например, такие домены обычно содержат код страны), и не больше пяти. Всё это значит, что шаблон для проверки последней части адреса будет таким:

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

Прочесть его можно так: «Сначала должна быть точка, потом — от 2 до 5 алфавитных символов, а после этого строка заканчивается».

Подготовив шаблоны для отдельных частей регулярного выражения, соберём их вместе:

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


Теперь осталось лишь протестировать то, что получилось:

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

Проверка адреса электронной почты с помощью регулярных выражений

То, что переданный `awk` текст выводится на экран, означает, что система распознала в нём адрес электронной почты.

Итоги

Если регулярное выражение для проверки адресов электронной почты, которое встретилось вам в самом начале статьи, казалось тогда совершенно непонятным, надеемся, сейчас оно уже не выглядит бессмысленным набором символов. Если это действительно так — значит данный материал выполнил своё предназначение. На самом деле, регулярные выражения — это тема, которой можно заниматься всю жизнь, но даже то небольшое, что мы разобрали, уже способно помочь вам в написании скриптов, которые довольно продвинуто обрабатывают тексты.

В этой серии материалов мы обычно показывали очень простые примеры `bash`-скриптов, которые состояли буквально из нескольких строк. В следующий раз рассмотрим кое-что более масштабное.

Уважаемые читатели! А вы пользуетесь регулярными выражениями при обработке текстов в сценариях командной строки?