## Регулярные выражения

Регулярные выражения - это шаблоны, которые могут интерпретировать только определенные команды. Они похожи на глобальные шаблоны, но имеют больше разнообразия и силы. Как и глобальные шаблоны, регулярные выражения могут быть расширены для соответствия определенным последовательностям символов в тексте.

Примеры, показанные в этой главе, будут использовать команду **grep** для демонстрации регулярных выражений. Команда **grep** может фильтровать текст из данных, обеспечивая наглядную демонстрацию работы регулярных выражений. Команда **grep**, которая будет использоваться, на самом деле является псевдонимом, который запускает **grep --color**, так что соответствующий текст будет отображаться красным цветом.

Регулярные выражения имеют две общие формы: базовую и расширенную. Большинство команд, которые используют регулярные выражения, могут интерпретировать основные регулярные выражения. Однако расширенные регулярные выражения доступны не для всех команд, и для их правильной работы обычно требуется параметр команды.

В следующей таблице приведены основные символы регулярных выражений:

| Базовый символ | Значение |
|---|---|
| . | Любой один символ |
| [ ] | Любой указанный символ |
| [^] | Ни один из указанных символов |
| * | Ноль или более предыдущих символов |
| ^ | Если первый символ в шаблоне, то шаблон должен находиться в начале строки, чтобы соответствовать, в противном случае просто литерал ^ |
| $ | Если последний символ в шаблоне, шаблон должен быть в конце строки, чтобы соответствовать, в противном случае просто литерал $ |

В следующей таблице приведены расширенные регулярные выражения, которые должны использоваться либо с командой **egrep**, либо с параметром **-E** с командой **grep**:

| Расширенные символы | Значение |
|---|---|
| + | Один или несколько из предыдущего шаблона |
| ? | Предыдущий шаблон не является обязательным |
| { } | Укажите минимальное, максимальное или точное соответствие предыдущего шаблона |
| \| | Чередование - логическое «или» |
| ( ) | Используется для создания групп |

Самое простое из всех регулярных выражений - просто использовать буквенные символы. Например, чтобы найти все строки файла **/etc/passwd**, которые содержат выражение root, используйте команду

```
grep 'root' /etc/passwd
```

***Якорные символы***

Якорные символы являются одним из способов использования регулярных выражений для сужения результатов поиска. Например, шаблон **root** много раз появляется в файле **/etc/passwd:**

```
grep 'root' /etc/passwd
root:x:0:0:root:/root:bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

Первый якорный символ ^ используется для того, чтобы шаблон появился в начале строки. Например, чтобы найти все строки в **/etc/passwd**, начинающиеся с **root,** вы можете использовать шаблон **^root**. Обратите внимание, что ^ должен быть первым символом в шаблоне, чтобы быть эффективным:

```
grep ^root /etc/passwd
root:x:0:0:root:/root:bin/bash
```

Второй якорный символ **$** может быть использован для обеспечения появления шаблона в конце строки, тем самым эффективно сокращая результаты поиска. Чтобы найти строки, заканчивающиеся на **bash** в файле **/etc/passwd**, используйте шаблон **bash$**:

```
grep bash$ /etc/passwd
```

Позиция этого символа важна, $ должен быть последним символом в образце, чтобы быть эффективным как якорь.

***Использование символа «.»***

Одним из самых полезных выражений является «.» Он будет соответствовать любому символу, кроме символа новой строки.

Шаблон **r..t** найдет любую строку, содержащую букву **r**, за которой следуют ровно два символа (которые могут быть любыми символами, кроме новой строки), а затем буква **t**:

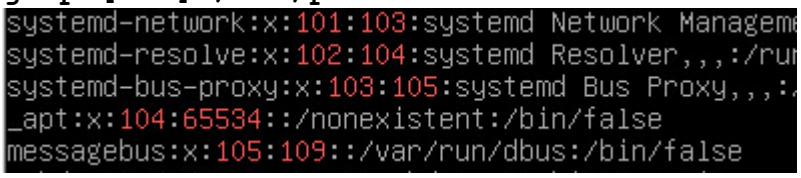
```
grep r..t /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:1000:37::/root:
```

***Совпадение одного символа с []***

Квадратные скобки [] работают в регулярных выражениях подобно тому, как они работают в глобальных выражениях; они соответствуют одному символу из списка или диапазона возможных символов, содержащихся в скобках.

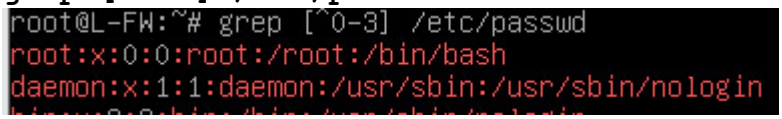
Чтобы найти все строки в файле **/etc/passwd**, которые имеют номер, используйте шаблон [0123456789] или [0-9]:

```
grep [0-9] /etc/passwd
```




С другой стороны, чтобы найти все строки, содержащие любые нечисловые символы, вставьте ^ в качестве первого символа в скобках. Этот символ отрицает перечисленные символы:

```
grep [^0-9] /etc/passwd
```




**Показывает строки, которые содержат не числа**

Когда другие символы регулярного выражения помещаются в квадратные скобки, они рассматриваются как буквенные символы. Например, обычно «.» соответствует любому одному символу, но помещается в квадратные скобки, тогда он будет соответствовать самому себе. В примере только строки, которые содержат «.» совпадают.

***Повторяющиеся символы символ или шаблон ****

Символ регулярного выражения * используется для соответствия нуль или нескольким вхождениям символа или шаблона, предшествующего ему. Например, **e*** будет соответствовать нуль или более вхождений буквы e:

```
cp /usr/share/dict/words
cat words | grep 're*d' | head
```

Также возможно сопоставить ноль или более вхождений списка символов, используя квадратные скобки:

```
cat words | grep 'r[oe]*d' | head
```

При использовании только с одним другим символом * не очень помогает. Любой из следующих шаблонов будет соответствовать каждой строке или строке в файле:. * e* b* z*.

Это потому, что * может соответствовать нулю вхождения шаблона. Чтобы сделать * полезным, необходимо создать шаблон, который включает в себя не только один символ, предшествующий *. Например, приведенные выше результаты могут быть уточнены путем добавления еще одного e, чтобы шаблон ee* эффективно соответствовал каждой строке, содержащей хотя бы одно e.

```
cat words | grep 'ree*d' | head
```

```
root@L-FW:~# cat words | grep 'ree*d' | head -30
Airedale
Airedale's
Airedales
Alfred
Alfreda
```

***Совпадение повторяющегося символа или шаблон с +***

Другое решение проблем использования символа * - вместо этого использовать расширенное регулярное выражение +. Это соответствует как минимум одному из предыдущих символов вместо нуля, так что это гораздо более избирательно. Например, **e+** будет соответствовать, только если текст содержит один или несколько символов **e**, что приводит к тем же результатам, что и шаблон **ee*** с предыдущего примера:

```
cat words | grep -E 'e+'
cat words | grep -E 're+d'
```

```
veered
veneered
ventured
volunteered
```

***Необязательный символ или шаблон с ?***

? расширенное регулярное выражение делает предыдущий символ (или шаблон) необязательным. Например, рассмотрим слово `yogurt`, которое также можно записать с дополнительным h как `yoghurt`. Используйте расширенное регулярное выражение **yogh?urt** для соответствия любому написанию:

```
echo 'Frozen yogurt is called Froyo' | grep -E 'yogh?urt'
echo 'Frozen yoghurt is called Froyo' | grep -E 'yogh?urt'
```

***Повторяющиеся символы или шаблоны с {}***

Фигурные скобки также изменяют предыдущий символ или шаблон. Их можно использовать вместо *, + или ? . Следующая таблица иллюстрирует некоторые примеры использования фигурных скобок:

| Пример с фигурной скобкой | Эквивалент регулярного выражения | Значение |
|---|---|---|
| `a{0,}` | `a*` | Ноль или более символа a |

| Пример с фигурной скобкой | Эквивалент регулярного выражения | Значение |
|---|---|---|
| a{1,} | a+ | Один или более символа a |
| a{0,1} | a? | Ноль или один символ a |
| a{5} | N/A | Пять символов a |
| a{,5} | N/A | Пять или менее символов a |
| a{3,5} | N/A | От 3 до 5 символов a |

***Совпадение шаблонов с |***

Символ | отделяет альтернативные выражения, которые могут совпадать; это также известно как чередование. Например, чтобы соответствовать слову «tire» или «tire», используйте выражение «tire | tire»:

```
echo 'Does a MINI need a tyre or a tire?' | grep -E 'tire|tyre'
Does a MINI need a tyre or a tire?
```

***Группировка шаблонов с ()***

Использование скобок для создания групп может быть полезно для нескольких целей. На самом базовом уровне они используются для группировки символов, которые могут быть изменены символом регулярного выражения, таким как *, +,? или фигурные скобки {}. Например, чтобы найти шаблон **pae**, повторенный один или несколько раз, используйте расширенное регулярное выражение **(pae)+**.

```
cat words | grep -E '(pae)+'
root@L-FW:~# cat words | grep -E '(pae)+'
encyclopaedia
encyclopaedia's
encyclopaedias
orthopaedic
```

Скобки также могут использоваться с чередованием. Предыдущий пример шин | шин может быть переписан как **t(i|y)re**. Когда круглые скобки используются в регулярных выражениях из командной строки, не забудьте заключить их в одинарные кавычки, чтобы оболочка не могла их интерпретировать:

```
echo 'Does a MINI need a tyre or a tire?' | grep -E 't(i|y)re'
Does a MINI need a tyre or a tire?
```

В некоторых командах круглые скобки могут даже использоваться для ссылки на то, что было найдено. То, что соответствовало первому набору скобок, может упоминаться как \1, второе как \2 и так далее. Например, если у вас есть файл, содержащий список людей с их именами и фамилиями, вы можете поменять их местами через запятую с помощью следующей команды **sed:**

```
cat names
Linus Torvalds
Dennis Ritchies
Ken Thompson
sed -r 's/(\w+) (\w+)/\2,\1/' names
Torvalds, Linus
Ritchies, Dennis
Thompson, Ken
```

Параметр **-r** команды **sed** указывает на использование расширенных регулярных выражений.

*Использование специальных последовательностей регулярных выражений*

Обычно, поставить обратную косую черту перед символом означает **буквально** соответствовать этому символу. Например, соответствие **.** символ с **\.** является целесообразным.

Однако есть и некоторые специальные комбинации символов обратной косой черты, как показано на следующей диаграмме.

| Backslash Sequence | Matches |
|---|---|
| \b | Граница слова |
| \B | Не граница слова |
| \w | Слово |
| \W | Не слово |
| \s | Пробельный символ |
| \S | Не пробельный символ |
| \\ | Символ обратной косой черты |

Слова не всегда имеют пробелы вокруг них, иногда они будут иметь запятые или другие знаки препинания до или после слова. Регулярное выражение **\b** может помочь идентифицировать края слов, которые не имеют пробелов вокруг них. Без использования границы слова могут возникнуть следующие проблемы:

```
echo 'This is useful' | sed 's/is/was/'
Thwas is useful
```

Обратите внимание, что sed произвел поиск по входу is и заменил его на was. Это не ограничивало его замену тем, что было отдельным словом, оно также затрагивало часть слова Это заменяя его на Thwas.

Было бы лучше использовать критерии поиска **\bis\b** или **is** с ограничением слов вокруг него, чтобы соответствовать слову **is**:

```
echo 'This is useful' | sed 's/\bis\b/was/'
This was useful
```