Регулярные выражения

Последнее изменение: Пт Ноя 27 09:44:51 2020

Содержание

Что такое регулярное выражение?
Структура регулярного выражения
Символы привязки: ^ и $
Сопоставление символа с набором символов
Сопоставьте любой символ с .
Указание диапазона символов с помощью [...]
Исключения в наборе символов
Повторяющиеся наборы символов с *
Сопоставление определенного количества наборов с помощью \{ и \}
Сопоставление слов с помощью \< и \>
Обратные ссылки - запоминание шаблонов с помощью \(, \) и \1
Потенциальные проблемы
Расширенные регулярные выражения
Наборы символов POSIX
Расширения Perl
Спасибо

Регулярные выражения и расширенное сопоставление шаблонов

Брюс Барнетт

Обратите внимание, что это было написано в 1991 году, до Linux. В 1980-х годах было обычным иметь разные наборы функций регулярных выражений с разными функциями. ed(1) отличался от sed(1), который отличался от vi (1) и т. Д. Обратите внимание, что Sun прошлась по каждой утилите и заставила каждую использовать одну из двух разных библиотек регулярных выражений - обычную или расширенную. Я написал это руководство для пользователей Sun, и некоторые из

обсуждаемых команд в настоящее время устарели. В Linux и других системах UNIX вы можете обнаружить, что некоторые из этих функций не реализованы. Ваш пробег может отличаться.

Оригинальная версия, написанная в 1994 году и опубликованная в Sun Observer

Что такое регулярное выражение?

Регулярное выражение - это набор символов, которые задают шаблон. Термин "обычный" не имеет ничего общего с диетой с высоким содержанием клетчатки. Это происходит от термина, используемого для описания грамматик и формальных языков.

Регулярные выражения используются, когда требуется выполнить поиск определенных строк текста, содержащих определенный шаблон. Большинство утилит UNIX работают с файлами ASCII по строке за раз. Регулярные выражения ищут шаблоны в одной строке, а не шаблоны, которые начинаются в одной строке и заканчиваются в другой.

Поиск определенного слова или строки символов прост. Это может сделать почти каждый редактор в любой компьютерной системе. Регулярные выражения более мощные и гибкие. Вы можете искать слова определенного размера. Вы можете искать слово с четырьмя или более гласными, которые заканчиваются на "s". Числа, знаки препинания, вы называете это, регулярное выражение может найти его. Другое дело, что произойдет, когда программа, которую вы используете, найдет это. Некоторые просто ищут шаблон. Другие выводят строку, содержащую шаблон. Редакторы могут заменить строку новым шаблоном. Все зависит от утилиты.

Регулярные выражения сбивают людей с толку, потому что они очень похожи на шаблоны сопоставления файлов, используемые оболочкой. Они даже действуют одинаково - почти. Квадратные скобки похожи, и звездочка действует аналогично, но не идентично звездочке в регулярном выражении. В частности, оболочка Bourne, оболочка C, *find* и *cpio* используют шаблоны сопоставления имен файлов, а не регулярные выражения.

Помните, что метасимволы оболочки расширяются до того, как оболочка передает аргументы программе. Чтобы предотвратить это расширение, специальные символы в регулярном выражении должны заключаться в

кавычки при передаче в качестве опции из командной строки. Вы уже знаете, как это сделать, потому что я рассматривал эту тему в учебном пособии в прошлом месяце.

Структура регулярного выражения

Регулярное выражение состоит из трех важных частей. **Привязки** используются для указания положения шаблона по отношению к строке текста. **Наборы символов** соответствуют одному или нескольким символам в одной позиции. **Модификаторы** указывают, сколько раз повторяется предыдущий набор символов. Простым примером, демонстрирующим все три части, является регулярное выражение "^#*". Стрелка вверх - это якорь, который указывает на начало строки. Символ "#" - это простой набор символов, который соответствует единственному символу "#". Звездочка - это модификатор. В регулярном выражении указывается, что предыдущий набор символов может появляться любое количество раз, включая ноль. Это бесполезное регулярное выражение, как вы скоро увидите.

Существует также два типа регулярных выражений: "базовое" регулярное выражение и "расширенное" регулярное выражение. Несколько утилит, таких как *awk* и *egrep*, используют расширенное выражение. Большинство использует "базовое" регулярное выражение. С этого момента, если я буду говорить о "регулярном выражении", оно описывает функцию в обоих типах.

Вот таблица команд Solaris (около 1991 года), которые позволяют задавать регулярные выражения:

Полезность	Тип регулярного выражения
vi	Базовые модели
sed	Базовые модели
grep	Базовые модели
csplit	Базовые модели
dbx	Базовые модели
dbxtool	Базовые модели
Еще	Базовые модели
изд	Базовые модели
выражение	Базовые модели
лексический	Базовые модели
pg	Базовые модели
NL	Базовые модели
rdist	Базовые модели
awk	Расширенные

nawk	Расширенные
egrep	Расширенные
EMACS	Регулярные выражения EMACS
PERL	Регулярные выражения PERL

Символы привязки: ^ и $

Большинство текстовых средств UNIX ориентированы на строки. Поиск шаблонов, охватывающих несколько строк, выполнить непросто. Видите ли, символ конца строки не включается в текстовый блок, в котором выполняется поиск. Это разделитель. Регулярные выражения проверяют текст между разделителями. Если вы хотите выполнить поиск шаблона, который находится на одном или другом конце, вы используете *якоря*. Символ "^" является начальным якорем, а символ "$" является конечным якорем. Регулярное выражение "^A" будет соответствовать всем строкам, начинающимся с заглавной буквы A. Выражение "A $" будет соответствовать всем строкам, которые заканчиваются заглавной буквой A. Если символы привязки не используются в правильном конце шаблона, то они больше не действуют как привязки. То есть "^" является привязкой только в том случае, если это первый символ в регулярном выражении. "$" является привязкой только в том случае, если это последний символ. Выражение "$1" не имеет привязки. Как и "1 ^". Если вам нужно сопоставить "^" в начале строки или "$" в конце строки, вы должны *экранировать* специальные символы обратной косой чертой. Вот краткое изложение:

Шаблон	Матчи
^A	"A" в начале строки
A$	"A" в конце строки
A^	"A^" в любом месте строки
$A	"$A" в любом месте строки
^^	"^" в начале строки
$$	"$" в конце строки

Использование "^" и "$" в качестве указателей начала или конца строки является соглашением, используемым другими утилитами. Редактор *vi* использует эти два символа в качестве команд для перехода к началу или концу строки. Оболочка C использует "!^" для указания первого аргумента предыдущей строки, а "! $" является последним аргументом в предыдущей строке.

Это один из тех вариантов, которые другие утилиты используют для поддержания согласованности. Например, "$" может ссылаться на последнюю строку файла при использовании *ed* и *sed*. *Cat -e* помечает

конец строк символом "$". Вы можете увидеть это и в других программах.

Сопоставление символа с набором символов

Самый простой набор символов - это символ. Регулярное выражение "the" содержит три набора символов: "t", "h" и "e". Он будет соответствовать любой строке со строкой "the" внутри нее. Это также соответствовало бы слову "другое". Чтобы предотвратить это, ставьте пробелы перед и после шаблона: "the". Вы можете объединить строку с привязкой. Шаблон "^From:" будет соответствовать строкам почтового сообщения, которые идентифицируют отправителя. Используйте этот шаблон с grep для печати каждого адреса в вашем почтовом ящике входящей почты:

```
grep '^From: ' /usr/spool/mail/$USER
```

Некоторые символы имеют особое значение в регулярных выражениях. Если вы хотите выполнить поиск по такому символу, экранируйте его обратной косой чертой.

Сопоставьте любой символ с .

Символ "." является одним из таких специальных мета-символов. Само по себе оно будет соответствовать любому символу, кроме символа в конце строки. Шаблон, который будет соответствовать строке с одним символом, является

```
^.$
```

Указание диапазона символов с помощью [...]

Если вы хотите сопоставить определенные символы, вы можете использовать квадратные скобки для определения точных символов, которые вы ищете. Шаблон, который будет соответствовать любой строке текста, содержащей ровно одно число, является

```
^[0123456789]$
```

Это многословно. Для указания диапазона можно использовать дефис между двумя символами:

```
^[0-9]$
```

Вы можете смешивать явные символы с диапазонами символов. Этот шаблон будет соответствовать одному символу, который является буквой, цифрой или символом подчеркивания:

[A-Za-z0-9_]

Наборы символов можно комбинировать, размещая их рядом друг с другом. Если вы хотите выполнить поиск по слову, которое

Начинается с заглавной буквы "T".
Было ли первое слово в строке
Вторая буква была строчной буквой
Было ровно три буквы длиной, и
Третья буква была гласной

регулярным выражением будет "^T[a-z][aeiou] ".

Исключения в наборе символов

Вы можете легко выполнить поиск по всем символам, кроме тех, что заключены в квадратные скобки, поставив "^" в качестве первого символа после "[". Для сопоставления всех символов, кроме гласных, используйте "[^aeiou]".

Подобно якорям в местах, которые нельзя считать якорем, символы "]" и "-" не имеют особого значения, если они непосредственно следуют за "[". Вот несколько примеров:

Регулярное выражение	Матчи
[]	Символы "[]"
[0]	Символ "0"
[0-9]	Любое число
[^0-9]	Любой символ, отличный от числа
[-0-9]	Любое число или "-"
[0-9-]	Любое число или "-"
[^-0-9]	Любой символ, кроме числа или "-"
[]0-9]	Любое число или "]"
[0-9]]	Любое число, за которым следует "]"
[0-9-z]	Любое число,
	или любой символ между "9" и "z".
[0-9\-a\]]	Любое число, или
	a "-", a "a" или a "]"

Повторяющиеся наборы символов с *

Третья часть регулярного выражения - это модификатор. Он используется, чтобы указать, сколько раз вы ожидаете увидеть предыдущий набор символов. Специальный символ "*" соответствует

нулю или более копий. То есть регулярное выражение "0*" соответствует **нулю или более нулей**, в то время как выражение "[0-9]*" соответствует нулю или более чисел.

Это объясняет, почему шаблон "^ # *" бесполезен, поскольку он соответствует любому количеству "#" в начале строки, включая **ноль**. Поэтому это будет соответствовать каждой строке, потому что каждая строка начинается с нуля или более "#'s".

На первый взгляд может показаться, что начинать отсчет с нуля глупо. Это не так. Поиск неизвестного количества символов очень важен. Предположим, вы хотите найти число в начале строки, и перед числом могут быть или не быть пробелы. Просто используйте "^ *", чтобы сопоставить ноль или более пробелов в начале строки. Если вам нужно сопоставить один или несколько символов, просто повторите набор символов. То есть "[0-9]*" соответствует нулю или более чисел, а "[0-9][0-9]*" соответствует одному или нескольким числам.

Сопоставление определенного количества наборов с помощью \{ и \}

Вы можете продолжить описанную выше технику, если хотите указать минимальное количество наборов символов. Вы не можете указать максимальное количество наборов с помощью модификатора "*". Существует специальный шаблон, который вы можете использовать для указания минимального и максимального количества повторов. Это делается путем помещения этих двух чисел между "\{" и "\}". Обратная косая черта заслуживает отдельного обсуждения. Обычно обратная косая черта **отключает** специальное значение символа. Точке соответствует символ "\.", а звездочке соответствует символ "*".

Если обратная косая черта помещается перед "<," ">," "{," "}," "(," ")," или перед цифрой обратная косая черта **приобретает** особое значение. Это было сделано потому, что эти специальные функции были добавлены в конце срока службы регулярных выражений. Изменение значения "{" привело бы к нарушению старых выражений. Это ужасное преступление, наказуемое годом каторжных работ по написанию программ на COBOL. Вместо этого, добавление обратной косой черты добавило функциональность, не нарушая работу старых программ. Вместо того, чтобы жаловаться на несимметричность, рассматривайте это как эволюцию.

Убедив вас, что "\{" - это не заговор, чтобы сбить вас с толку, приведу пример. Регулярное выражение, соответствующее 4, 5, 6, 7 или 8 строчным буквам, является

[a-z]\{4,8\}

Могут использоваться любые числа от 0 до 255. Второе число может быть опущено, что снимает верхний предел. Если запятая и второе число опущены, шаблон должен быть продублирован точное количество раз, указанное первым числом.

Вы должны помнить, что модификаторы, такие как "*" и "\{1,5 \}", действуют как модификаторы, только если они следуют за набором символов. Если бы они были в начале шаблона, они не были бы модификатором. Вот список примеров и исключений:

Регулярное выражение	Матчи
_	
*	Любая строка со звездочкой
*	Любая строка со звездочкой
\\	Любая строка с обратной косой чертой
^*	Любая строка, начинающаяся со звездочки
^A*	Любая строка
^A*	Любая строка, начинающаяся с "A*"
^AA*	Любая строка, если она начинается с одной "A"
^AA*B	Любая строка с одной или несколькими буквами "A", за которыми следует буква "B"
^A\{4,8\}B	Любая строка, начинающаяся с 4, 5, 6, 7 или 8 букв "A"
	за которым следует буква "B"
^A\{4,\}B	Любая строка, начинающаяся с 4 или более букв "A"
	за которым следует буква "B"
^A\{4\}B	Любая строка, начинающаяся с "AAAAB"
\{4,8\}	Любая строка с "{4,8}"
A{4,8}	Любая строка с "A{4,8}"

Сопоставление слов с помощью \< и \>

Поиск слова не так прост, как кажется на первый взгляд. Строка "the" будет соответствовать слову "other". Вы можете ставить пробелы перед и после букв и использовать это регулярное выражение: " the ". Однако это не соответствует словам в начале или конце строки. И это не соответствует случаю, когда после слова стоит знак препинания.

Есть простое решение. Символы "\<" и "\>" похожи на якоря "^" и "$", поскольку они не занимают позицию символа. Они "привязывают" выражение between к совпадению только в том случае, если оно находится на границе слова. Шаблоном для поиска слова "the" будет "\ <[tT]he\>". Символ перед "t" должен быть либо символом новой строки,

либо любым другим символом, кроме буквы, цифры или подчеркивания. Символ после "e" также должен быть символом, отличным от цифры, буквы или символа подчеркивания, или это может быть символ конца строки.

Обратные ссылки - запоминание шаблонов с помощью \(, \) и \1

Другой шаблон, который требует специального механизма, - это поиск повторяющихся слов. Выражение "[a-z][a-z]" будет соответствовать любым двум строчным буквам. Если вы хотите выполнить поиск строк с двумя соседними одинаковыми буквами, приведенный выше шаблон не поможет. Вам нужен способ запомнить, что вы нашли, и посмотреть, повторился ли тот же шаблон снова. Вы можете пометить часть шаблона с помощью "\(" и "\)". Вы можете вспомнить запомнившийся шаблон с помощью "\", за которым следует одна цифра. Поэтому для поиска двух одинаковых букв используйте "\([a-z]\)\1". У вас может быть 9 разных запоминаемых шаблонов. Каждое вхождение "\(" запускает новый шаблон. Регулярное выражение, которое соответствовало бы палиндрому из 5 букв (например, "радар"), будет

```
\([a-z]\)\([a-z]\)[a-z]\2\1
```

Потенциальные проблемы

На этом обсуждение базового регулярного выражения завершено. Прежде чем я начну обсуждать расширения, предлагаемые расширенными выражениями, я хотел бы упомянуть две потенциальные проблемные области.

Символы "\<" и "\>" были введены в редакторе *vi*. В то время у других программ не было такой возможности. Также модификатор "\{*min*, *max* \}" является новым, и более ранние утилиты не имели такой возможности. Это затрудняло работу начинающего пользователя регулярных выражений, потому что казалось, что каждая утилита имеет свое соглашение. Sun обновила новейшую библиотеку регулярных выражений для всех своих программ, поэтому все они обладают одинаковыми возможностями. Если вы попытаетесь использовать эти новые функции на компьютерах других производителей, вы можете обнаружить, что они работают по-другому.

Другой потенциальной причиной путаницы является степень совпадения шаблонов. Регулярные выражения соответствуют максимально возможному шаблону. То есть регулярное выражение

```
A.*B
```

соответствует "AAB", а также "AAAABBBBABCCCCBBBAAAAB". Это не вызывает много проблем при использовании *grep*, потому что ошибка в регулярном выражении просто приведет к совпадению большего количества строк, чем требуется. Если вы используете *sed*, и ваши шаблоны увлекаются, вы можете в конечном итоге удалить больше, чем хотели.

Расширенные регулярные выражения

Две программы используют расширенные регулярные выражения: *egrep* и *awk*. С этими расширениями специальные символы, которым предшествует обратная косая черта, больше не имеют особого значения: "\{" , "\}", "\<", "\>", "\(", "\)" так же, как и "*digit*". Для этого есть очень веская причина, которую я отложу с объяснением, чтобы усилить напряженность.

Символ "?" соответствует 0 или 1 экземпляру предыдущего набора символов, а символ "+" соответствует одной или нескольким копиям набора символов. Вы не можете использовать \{ и \} в расширенных регулярных выражениях, но если бы вы могли, вы могли бы считать, что "?" совпадает с "\{0,1 \}", а "+" совпадает с "\{1,\}".

К настоящему времени вы задаетесь вопросом, почему расширенные регулярные выражения вообще стоит использовать. За исключением двух сокращений, преимуществ нет, а недостатков много. Поэтому примеры были бы полезны.

Тремя важными символами в расширенных регулярных выражениях являются "(", "|" и ")". Вместе они позволяют сопоставлять **выбранные** шаблоны. В качестве примера, вы можете использовать *egrep* для печати всех строк *из:* и *Subject:* из вашей входящей почты:

```
egrep '^(Из|Subject): ' /usr/spool/mail/$USER
```

Будут напечатаны все строки, начинающиеся с "From:" или "Subject:". Не существует простого способа сделать это с помощью базовых регулярных выражений. Вы могли бы попробовать "^ [FS] [ru] [ob] [mj] e * c * t *: " и надеюсь, у вас нет строк, начинающихся с "Sromeet:". Расширенные выражения не содержат символов "\<" и "\>". Вы можете компенсировать это с помощью механизма чередования. Сопоставление слова "the" в начале, середине, конце предложения или в конце строки может быть выполнено с помощью расширенного регулярного выражения:

```
(^| )([^a-z]|$)
```

Перед словом можно выбрать два варианта: пробел или начало строки. После слова должно быть что-то, кроме строчной буквы или конца строки. Одним из дополнительных преимуществ расширенных регулярных выражений является возможность использования модификаторов "*", "+" и "?" после группировки "(...)". Следующее будет соответствовать "простой проблеме", "простой проблеме", а также "проблеме".

```
egrep "a[n]? (простой | легкий)?проблема " данные
```

Обратите внимание на пробел после слов "простой" и "легкий".

Я обещал объяснить, почему символы обратной косой черты не работают в расширенных регулярных выражениях. Ну, возможно, "\{... \}" и "\<...\>" можно было бы добавить к расширенным выражениям. Это новейшее дополнение к семейству регулярных выражений. Их можно было бы добавить, но это может сбить людей с толку, если эти символы будут добавлены, а "\(... \)" - нет. И нет никакого способа добавить эту функциональность к расширенным выражениям без изменения текущего использования. Вы понимаете, почему? Это довольно просто. Если "(" имеет особое значение, то "\(" должен быть обычным символом. Это противоположно основным регулярным выражениям, где "(" является обычным, а "\(" является специальным. Использование круглых скобок несовместимо, и любое изменение может привести к поломке старых программ.

Если расширенное выражение использовало "(..|...)" в качестве обычных символов, и "\(...\|...\)" для указания альтернативных шаблонов можно использовать один набор регулярных выражений, обладающий полной функциональностью. Кстати, это именно то, что делает GNU emacs.

Остальное - случайные заметки.

Регулярное выражение	Класс	Тип	Значение
_			
.	ВСЕ	Набор символов	Один символ (кроме новой строки)
^	ВСЕ	Привязка	Начало строки
$	ВСЕ	Привязка	Конец строки
[...]	ВСЕ	Набор символов	Диапазон символов
*	ВСЕ	Модификатор	ноль или более дубликатов
\<	Базовые модели	Привязка	Начало слова
\>	Базовые	Привязка	Конец слова

	модели		
\(..\)	Базовые модели	Обратная ссылка	Запоминает шаблон
\1..\9	Базовые модели	Ссылка	Напоминает шаблон
_+	Расширенные	Модификатор	Один или несколько дубликатов
?	Расширенные	Модификатор	Ноль или один дубликат
\{M,N\}	Расширенные	Модификатор	От M до N дубликатов
(...|...)	Расширенные	Привязка	Показывает изменение
_			
\(...|...\)	EMACS	Привязка	Показывает изменение
\w	EMACS	Набор символов	Соответствует букве в слове
\W	EMACS	Набор символов	Напротив \w

Наборы символов POSIX

В POSIX добавлены новые и более переносимые способы поиска наборов символов. Вместо использования [a-zA-Z] вы можете заменить 'a-zA-Z' на [:alpha:] или для большей полноты. замените [a-zA-Z] на [[:alpha:]] . Преимущество заключается в том, что это будет соответствовать международным наборам символов. Вы можете смешивать старый стиль и новые стили POSIX, такие как
grep '[1-9[:alpha:]]'
Вот список заполнения

Группа символов	Значение
[:alnum:]	Буквенно-цифровые
[:cntrl:]	Управляющий символ
[:нижний:]	Символ нижнего регистра
[:пробел:]	Пробелы
[:альфа:]	Алфавитный
[:цифра:]	Цифра
[:печать:]	Печатаемый символ
[:верхний:]	Символ верхнего регистра
[:пробел:]	пробелы, табуляции и т.д.
[:график:]	Печатные и видимые символы
[:punct:]	Пунктуация
[:xdigit:]	Расширенная цифра

Обратите внимание, что некоторые люди используют [[:alpha:]] в качестве обозначения, но внешний символ '[...]' задает набор символов.

Расширения Perl

Регулярное выражение		
Класс	Тип	Значение
\t	Набор символов	табуляция
\n	Набор символов	перевод строки
\r	Набор символов	Возврат
\f	Набор символов	форма
\a	Набор символов	тревога
\e	Набор символов	побег
\033	Набор символов	восьмеричные
\x1B	Набор символов	шестнадцатеричное
\c[	Набор символов	управление
\l	Набор символов	нижний регистр
\u	Набор символов	верхний регистр
\L	Набор символов	нижний регистр
\U	Набор символов	верхний регистр
\E	Набор символов	конец
\Q	Набор символов	Цитата
\w	Набор символов	Сопоставление символа "слово"
\W	Набор символов	Сопоставление символа, не являющегося словом
\s	Набор символов	Сопоставление пробельного символа
\S	Набор символов	Сопоставление символа, не содержащего пробелов
\d	Набор символов	Сопоставление символа цифры
\D	Набор символов	Сопоставление нецифрового символа

\b	Привязка	Сопоставьте границу слова
\B	Привязка	Сопоставление не- (границы слова)
\A	Привязка	Совпадение только в начале строки
\Z	Привязка	Совпадение только в EOS или перед новой строкой
\z	Привязка	Совпадение только в конце строки
\G	Привязка	Совпадают только с тем местом, где остановились предыдущие m / / g

Пример расширенного многострочного регулярного выражения PERL

```
m{ \(
  ( # Начать группу
  [^()]+ # что угодно, кроме '(' или ')'
  | # или
\( [^()]* \)
  )+ # конечная группа
  \)
}x
```

Спасибо

Спасибо следующим, кто заметил некоторые ошибки

Чарухас Мехендейл
Рунак Джайн
Питер Рензланд
Karl Eric Wenzel
Axel Schulze
Деннис отпугивает
Брайан Бергерт
Брэд Коэнвуд
Michael Siegel

Этот документ был переведен troff2html версии 0.21 27 июня 2001 года.