# Регулярные выражения в Perl

Стив Холзнер

***От редактора**: Когда мы публиковали это произведение, нам ничег Спасибо нашим читателям, благодаря им мы нашли фамилию автора Но по-прежнему остается неизвестным герой, который перевел его н нибудь именно об этом переводе (существуют другие), пожалуйста, Тихоном Тарнавским aka t.t.*

- Вывод результатов поиска

# Определения

Регулярные выражения в perl одна из самых мощных его возможност
указанным шаблоном, разбивать текст в массив по шаблону, производить з
другое. Так-же иногда регекспами называются операторы поиска и замены

Оператор `q(text)` заменяет строку text на строку, заключенную в один
поставить символ q`(text\n)`, то напечатает `text\n` , т.е. `\n` это два символ
`amam $file`). В данном случае почти все специальные символы не будут
`'\'`

```
$some=q(Don't may be);
```

Оператор `qq~text~;` (вместо значка `~` можно ставить например знак
многострочными текстами. пользуясь этим оператором можно выводить ц
имена скалярных переменных.

Оператор `qw("text")` разбивает строку на массив слов.

```
@mass=qw("я вышел погулять и увидел как через реку строят новый мо
#хотя с настроенной локалью будет работать и
@mass=qw(я вышел погулять и увидел как через реку строят новый мос
for(@mass){print $_,"\n"}
```

Оператор `qr/pattern/` ключи - `imosx` работает подобно регулярному выра

```
$rex=qr/my.STRING/is;
s#$rex#foo#;
#тоже самое, что и
s/my.STRING/foo/is;
```

Результат может использоваться подобно вызову подпрограммы(см perldoc

```
$re=qr/$pattern/;
$string=~/foo${re}bar/;
$string=~$re;
$string=~/$re/;
```

Ключи `imosx` стандартные(см. ниже)

Оператор `qx/STRING/` работает как системная команда, подобно иллюстрирующая использование данного оператора:

```
#!/usr/bin/perl
qx[dbfdump --fs="\x18" --rs="\x19" pdffile.dbf >pdffile.txt];
```

файл `pdffile.dbf` содержит memo-поля(memo-поле содержит ссылку, по расширением *.fpt), которые при помощи `DBI.pm` мне когда-то давно вы FoxBASE4 и дампит файлы со встроенными `memo`-полями в текстовый вид информацию из файла memo-типа `*.fpt`.

Допустим используя команду `$perl_info = qx(ps $$);` мы выводим инф скрипта(каждая запущенная программа в UNIX имеет свой собственн содержится во встроенной переменной `$$` - достаточно уникальное числ случайных чисел). Если сказать `$shell_info = qx'ps $$';` то вывед осуществляют своеобразное экранирование от двойной кавычки.

В перл есть три основных оператора, работающих со строками:

`m/.../` - проверка совпадений (matching),
`s/.../.../` - подстановка текста (substitution),
`tr/.../` - замена текста (translation).

Опертаор `m/.../` анализирует входной текст и ищет в нем подстроку совпа регулярным выражением). Оператор `s/.../.../` выполняет подстановку о при помощи регулярных выражений. Оператор `tr/.../.../` заменяет вых регулярные выражения, осуществляя замену посимвольно.

Оператор `m/шаблон/` - поиск подстроки по определенному шаблону. На `{4})!g` найдет и выведет все даты в переменной `$_`. В шаблоне не важно, ч поиске гиперссылок, которые зачастую содержат символы `/`, разумнее п символами ограничителями. В таком случае шаблон будет более прост дл немного короче. В perl оператор `m/.../` используется очень часто, и начальной буквы `m`. Если начальная буква есть, то в качетсве символов другой символ.

Для оператора `m/pattern/` есть 6 параметров: `gimsxo`

`m/foo/g` говорит компилятору найти все `foo` в тексте, в то время как подстроки `foo` в строке `$_`. В строке `$_` содержится обычный текст, переменная, только она существует всегда и вводится, когда не определена

Например можно сказать `for (@mass){print $_,"\n"}` или `for $elem (@m` делают одно и то-же, но в первом случае запись короче, да и зачастую бы например, когда нужно выделить при помощи регулярного выражения от массива(функция map):

```
@res=map{/(\d\d\d\d)/} split /\s/, $texts;
```

что эквививалентно коду

```
push @res, $1 while m!((\d){4})!g; #(в данном случае $_=$texts)
```

или что эквивалентно конструкции

```
foreach(split /\s/, $texts){
push @res, $1 if(/(\d\d\d\d)/g)
}
```

Следующий параметр `m/foo/i`, говорит о том, что не нужно учитывать рег

Параметр `m/foo/s` говорит от том, что строка, по которой производится по

Например нужно выцепить все url картинок из странички www.astronomy этой странички и пользователи могли с интересом читать последние новос

```
#!/usr/bin/perl -wT
use LWP::Simple;
$page=get "http://www.astronomynow.com";
&getlink($page);
sub getlink{
local $_=$_[0];
push(@res, "http://$2")
while m{SRC\s*=\s*(["'])http://(.*?)\1\s*(.*?)WIDTH="100" HEIGHT="
}
```

В подпрограмме заводится при помощи функции `local` переменна подпрограммы. Этой переменной присваивается значение переменно

выкачанной `Simple.pm` странички.

Можно сделать немного по другому, сохранить скачанную страничку в фай

```
$/="\001";
open F, "<page.html"; $page=<F>; close F;
&getlink($page); ...
```

Встроенная переменная `$/` содержит символ разделителя входных записей upload far'ом на сервер файлов в не ASCI виде, она приобретают на конце

Если `$/` переопределить, то можно свободно пользоваться дескрипт многострочного текста(`m/pattern/s`). Например когда открывается ф `<file.txt"; @mass=<F>`, то присваивая дескриптор `F` массиву в массиве содержащимся в `$/`.

Переопределив `$/` можно запросто написать:

```
open F, "<file.txt"; $mass=<F>
```

и в переменной `$mass` будет содержаться многострочный текст с точки зр этот текст как одну строку и по тексту можно будет запросто пройтис необходимые подстроки.

Параметр `m/foo/o` говорит от том, что шаблон нужно компилировать толь сочетании с операциями привязки `=~` и отрицание `!~`, то строкой, в кот стоящая слева от операции привязки. В противном случае поиск ведется в

Оператор `s!pattern!substring!` - поиск в строке по шаблону `pattern` и за и для оператора `m/.../`, косую черту можно не ставить, пригоден противореции с заданным выражением. Не рекомендуется использовать в

`s!/usr/local/etc/!/some/where/else!` - заменяет путь.
`s(/usr/local/etc/)(/some/where/else)g` - заменяет все встречающимеся

параметры: `egimsxo` `e` - указывает, что substring нужно вычислить.

например нужно переделать все escape последовательности, для этого вызы

```
$text =~ s/(&.*?;)/&esc2char($1)/egs;
```

т.е. из регулярного выражения происходит вызов подпрограммы.

`g` - заменить все одинаковые компоненты, а не один, как в отсутствии ключ
`i` - не учитывать регистр.
`m` - строка, в которой происходит поиск, состоит из множества строк.
`s` - строка, в которой происходит поиск, состоит из одной строки.
`x` - сложный шаблон, т.е. можно писать не в строчку, а для упрощения г
строк, примеры об этом ниже.
`o` - компилировать шаблон один раз.

Допустим нужно сделать поисковик, который ходит по директориям на сер
`bin/` и т.п. индексировать нельзя. Объявляем переменную, которая будет с
случае перечисление или `img` или `image` или `temp` или `tmp` или `cgi-bin`:

```
$no_dir = '(img|image|temp|tmp|cgi-bin)';
```

Ключи регулярного выражения `m#$no_dir$#io` говорят о том, что компили
один раз(ключ `o`) и также еще не учитывать регистр(ключ `i`).

Оператор `tr/выражение1/выражение2/`, ключи `cds`

Смысл: замена `выражения1` на `выражение2`. Если указан ключ `c`, то это инве
входят содержащиеся в нем символы. если указа ключ `d`, то значит стереть
то значит заменить многочисленные повторяющиеся символы на одиночны

Оператор `y/выражение1/выражение2/`(ключи `cds`), равносилен оператору `tr`

Например в поисковой системе нужно приводить запрос в нижний регистр

```
$CAP_LETTERS = '\xC0-\xDF\xA8';
$LOW_LETTERS = '\xE0-\xFF\xB8';
$code = '$html_text =~ ';
$code .= "tr/A-Z$CAP_LETTERS/a-z$LOW_LETTERS/";
$down_case = eval "sub{$code}";
```

## одиночные символы

В регулярном выражении любой символ соответствует самому себе, есл
специальным значением (такими метасимволами являются `\`, `|`, `(`, `)`, `[`, `{`
проверяется, не ввел ли пользователь команду "quit" (и если это так, то пре

```
while(<>){
if(m/quit/){exit;}
}
```

Правильнее проверить, что введенное пользователем слово "quit" не и
предложения. (Например, программа выполнит заведомо неверное действ
команду "Don't quit!".) Это можно сделать с помощью метасимволов
нечувствительно к разнице между прописными и заглавными буквами, исп

```
while (<>) {if (m/^quit$/i) {exit;} }
```

Кроме обычных символов perl определяет специальные символы. Они вы
(escape-последовательности) и также могут встречаться в регулярном выра

- `\077` - восьмеричный символ,
- `\a` - символ BEL (звонок),
- `\c[` - управляющие символы (комбинация Ctrl + символ, в данном слу
- `\d` - соответствует цифре,
- `\D` - соответствует любому символу, кроме цифры,
- `\e` - символ escape (ESC),
- `\E` - конец действия команд `\L`, `\U` и `\Q`,
- `\f` - символ прогона страницы (FF),
- `\1` - следующая литера становится строчной (lowercase),
- `\L` - все последующие литеры становятся строчными вплоть до коман
- `\n` - символ новой строки (LF, NL),
- `\Q` - вплоть до команды `\E` все последующие метасимволы становятся
- `\r` - символ перевода каретки (CR),
- `\s` - соответствует любому из "пробельных символов" (пробел, верт
  символ новой строки и т. д.),
- `\S` - любой символ, кроме "пробельного",
- `\t` - символ горизонтальной табуляции (HT, TAB),
- `\u` - следующая литера становится заглавной (uppercase),
- `\U` - все последующие литеры становятся заглавными вплоть до коман
- `\v` - символ вертикальной табуляции (VT),

- `\w` - алфавитно-цифровой символ (любая буква, цифра или символ под
- `\W` - любой символ, кроме букв, цифр и символа подчеркивания,
- `\x1B` - шестнадцатиричный символ.

Bat также можете "защитить" любой метасимвол, то есть заставить р символ, а не как команду, поставив перед метасимволом обратную косы типа `\w`, `\d` и `\s`, которые соответствуют не одному, а любому символу один такой символ, указанный в шаблоне, соответствует ровно одному с задания шаблона, соответствующего, например, слову из букв, цифр и си конструкцию `\w+`, как это сделано в следующем примере:

```
$text = "Here is some text."
$text =~ s/\w+/There/;
print $text;
There is some text.
```

## классы символов

Символы могут быть сгруппированы в классы. Указанный в шаблоне кл символов, входящим в этот класс. Класс - это совокупность символов, Можно указывать как отдельные символы, так и их диапазон (диапаз соединенными тире). Наример, следующий код производит поиск гласных

```
$text ="Here is the text.";
if ($text =" /[aeiou]/) {print "Vowels: we got 'em.\n";}
Vowels: we got 'em.
```

Другой пример: с помощью шаблона `[A-Za-z]+` (метасимвол `+` означа символов") ищется и заменяется первое слово:

```
$text = "What is the subject.";
$text =" s/[A-Za-z]+/Perl/;
print $text;
Perl is the subject;
```

Если требуется задать минус как символ, входящий в класс символов, п черту `\-`. Если сразу после открывающей квадратной скобки стоит символ А именно, этот класс сопоставляется любому символу, кроме перечисле примере производится замена фрагмента текста, составленного не из букв

```
$text = "perl is the subject on page 493 of the book.";
```

```
$text =- s/[a-Za-z\s]+/500/;
print $text;
perl is the subject on page 500 of the book.
```

## альтернативные шаблоны

Вы можете задать несколько альтернативных шаблонов, используя си шаблоны позволяют превратить процедуру поиска из однонаправленн подходит один шаблон perl подставляет другой и повторяет сравнение, возможные альтернативные комбинации. Например, следующий фрагмент "quit" или "stop":

```
while (<>){
 if(m/exit|quit|stop/){exit;}
}
```

Чтобы было ясно, где начинается и где заканчивается набор альтернати скобки - иначе символы, расположенные справа и слева от группы шаб шаблонами.

В следующем примере метасимволы `^` и `$` обозначают начало и альтернативных шаблонов с помощью скобок:

```
while (<>){
 if(m/^(exit|quit|stop)$/){exit;}
}
```

Альтернативные варианты перебираются слева направо. Как только на выполняется совпадение с шаблоном, перебор прекращается. Участки выполняют специальную роль при выполнении операций поиска и замен скобках, он интерпретируется как обычный символ. Поэтому если вы `[Tim|Tom|Tam]`, то она будет эквивалентна классу символов `[Tioam` метасимволов и команд, специфичных для регулярных выражений - символы, описанные в двух последующих разделах, - внутри квадратных или escape-последовательности текстовых строк.

## квантификаторы

Квантификаторы в регулярных выражениях

Квантификаторы указывают на то, что тот или иной шаблон в строке мо
раз. Например, можно использовать квантификатор `+` для поиска мест нео
буквы `e` и их замены на одиночную букву `e`:

```
$text = "Hello from Peeeeeeeeeeeeeeerl.";
$text =~ s/e+/e/:
print $text;
Hello from perl.
```

## мнимые символы

Мнимые символы в регулярных выражениях

В perl имеются символы (метасимволы), которые соответствуют не как
выполнение определенного условия (поэтому в английском языке их наз
можно рассматривать как мнимые символы нулевого размера, распол
символами в точке, соответствующей определенному условию:

- `^` - начало строки текста,
- `$` - конец строки или позиция перед символом начала новой строки, ра
- `\b` - граница слова,
- `\B` - отсутствие границы слова,
- `\A` - "истинное" начало строки,
- `\Z` - "истинный" конец строки или позиция перед символом начала нс
  конце строки,
- `\z` - истинный конец строки,
- `\G` - граница, на которой остановился предыдущий глобальный поиск,
- `(?= шаблон)` - после этой точки есть фрагмент текста, который
  выражению,
- `(?! шаблон)` - после этой точки нет текста, который бы соответствова
- `(?<= шаблон)` - перед этой точкой есть фрагмент текста, соответствую
- `(?<! шаблон)` - перед этой точкой нет фрагмента текста, сос
  выражению.

Например, вот как выполнить поиск и замену слова, используя метасимво

```
$text = "Here is some text.";
$text = s~/\b([A-Za-z]+)\b/There/;
print $text;
```

```
There is some text.
```

perl считает границей слова точку, расположенную между \w и \W, независ
символы . В следующем примере выводится сообщение о том, что польз
оно единственное, что ввел пользователь. Для этого шаблон включает мни

```
while (<>) {
 if (m/^yes$/) {
 print "Thank you for being agreeable.\n";
 }
}
```

Приведенный выше пример требует комментария. Прежде всего, бросаетс
для начала и конца строки. В большинстве случаев они означают одно и то
(то есть \n), встречающиеся внутри текстового выражения, не рассматрив
для команды m/.../ или s/.../.../ указан модификатор m, то текстов
многострочный текст, в котором границами строк выступают символы н
текста метасимвол ^ сопоставляется с позицией после любого симво
текстового выражения. Точно также метасимвол $ - это позиция
расположенным внутри текстового выражения, а не обязательно конец тек
концевым символом \n. Однако метасимвол \A - начало текстового выра
выра-жения или позиция перед концевым символом \n, даже если в т
символы \n и при выполнении операции поиска или йены указан
соответствует любому символу, кроме символа новой строки \n. Независи
будет сопоставляться ни с внутренними, ни с концевыми символами \
рассматривать \n как обычный символ - использовать модификатор s.

Отсюда понятна разница между метасимволами \Z и \z. Если в каче
результат чтения входного потока данных, то с большой вероятностью д
\n, за исключением того варианта, когда программа предусмотрительно "
или chomp. Метасимвол \Z игнорирует концевой символ \n если он слу
ситуации как "конец строки". В отличие от него метасимвол \z оказыва
концевой символ \n как неотъемлемую часть проверяемого текстового
позаботился об удалении этого символа.

Отдельно следует остановиться на метасимволе \G. Он может указыват
случае, если выполняется глобальный поиск (то есть если команда m/..
указанный в шаблоне, соответствует точке, на котброй остановилась предь

## ссылки на найденный текст

Иногда нужно сослаться на подстроку текста, для которой получено с
Например, при обработке файла, HTML может потребоваться выд
открывающими и закрывающими метками HTML (например, <A> и </
котором выделялся текст, ограниченнуй метками HTML <A> и <B>. След
расположенный между любыми правильно закрытыми метками:

```
$text = "<A>Here is an anct1or.</A>";
if($text=~m%<([A-Za-z]+)>[\w\s\.]+</\1>%i){
}
```

Вместо косой черты в качестве ограничителя шаблона использован дру
символ косой черты внутри шаблона без предшествующей ему обратной к
заключенному в круглые скобки, соответствует определенная внутренняя
так что на них можно ссылаться внутри шаблона, поставив перед номеро
значения переменных можно ссылаться внутри шаблона, как на обычный
если открываю-щей меткой служит<A>, и , если открывающей метк
переменные можно использовать и вне шаблона, ссылаясь на них как на ск

```
$text = "I have 4 apples.";
if ($text =- /(\(\d+)/) {
print "Here Is the number of apples: $1.\n";
Here is the number of apples: 4.
```

Каждой паре скобок внутри шаблона после завершения операции
переменная с соответствующим номером. Это можно 'использовать пр
работы фрагментов ана-лизируемой строки. В следующем примере мы
строке с помощью команды `s/.../.../`:

```
$text = "I see you.";
$text=-s/^(\w+) *(\w+) *(\w+)/$3 $2 $1/;
print $text;
you see I.
```

Переменные, соответствующие фрагментам шаблона, нуумеруются слев
Например, после следующей операции поиска будут проинициализирова
шести парам скобок:

```
$text = "ABCDEFGH";
$text =- m/(\w(\w)(\w))((\w)(\w))/;
print "$1/$2/$3/$4/$5/$6/";
ABC/B/C/DE/D/E
```

Кроме переменных, ссылающихся на найденный текст, можно использоват

## Функции, использующие регулярные выраже

Фактически, есть три функции, которые в качестве разделителя могут ис `grep`, `map` и еще можно воспользоваться специальными операторами `...` условиями `if`, `unless` и просто логическими операторами.

### split

Если необходимо разделить данные из `STDIN` по нужному разделителю, то

```
sub example_local{
local $/ = undef;
@mass= split /pattern/, <>;
return 1;
}
print scalar(@mass);
```

Можно разделять данные из файла и так:

```
undef $/;
@res=split /pattern/, <F>;
```

что эквивалентно:

```
while (<F>) {push @asdf, split}
```

После split можно ставить вместо запятой и стрелочку:

```
@mass = split /(\d){4}/ => $file;
```

В функции сплит можно воспользоваться макисмальным квантификатор позволит разделить строку на символы, которых там нет(в силу того, что `*`

```
@ruru = split /\001*/ => "lalalalalala";
#массив @ruru будет содержать элементы по одной букве.
```

Если строка состоит из нескольких строк, то можно поставить разделителе

```
 $str = "asdf\nghjk\nqwer\n";
 @lines = split /^/ => $str;
```

Вобщем, в split можно вставлять любой поиск по шаблону.

## grep

Функция grep так-же позволяет запонять массив значениями. Например ну
в заданной директории:

```
while(<$dir/*.*>){push @files, $_} #читаем директорию
@test = grep { s|.*/(.*?)\.(.*)|$2| } @files; #оставляем в директо
```

можно использовать признак четности для занесения в массив:

```
@test1=qw(1 2 3 4 5 6 7 8 9);
@evens = grep($_%2 == 1) @test1;
```

Или более сложное регулярное выражение для вытаскивания всех e-mail а

```
@mass=grep{s/(.*) ([\w+\-\.]+\@[\w\-\.]+\.\w{2,3})(.*)/$2/ig} spli
```

Здесь используется укороченная запись:

```
@mass=grep {/pattern/} split /\n/, $test;
```

которая эквивалента записи из двух сторчек:

```
@uuu=split /\n/, $test;
@mass=grep {/pattern/} @uuu;
```

## map

Функция map похожа по своей работе на обычное условие `if`, допу
разделенные четырьмя пробелами:

```
@probel = map m!\s{4}!, split /\n/, $test;
```

## other

Вывод строк из заданного интервала для данной строки:

```
if(/pattern1/i .. /pattern2/i){...}
```

```
#истинность первого оператора включает конструкцию, а второго е вы
if($nomer1 .. $nomer2){...}
```

`...` не возвратит истину, в отличии от `..`, если условия выполняются в одн

```
if(/pattern1/i ... /pattern2/i){...}
if($nomer1 ... $nomer2){...}
```

для многострочного файла

```
print -ne 'print if 3 .. 15' file.txt
```

выведет строки файла с 3 по 15 строчку, та-же самая опреация но немного

```
open F, "<file";
while(<F>){
 print if(3 .. 15)
}
```

или с какой нибудь начальной и конечно разметкой, например есть различные виды html, в зависимости от действия пользователя) для разн исходя из контекста программы:

```
open F, "<file";
while(<F>){
 print if(/<!--begin welcome-->/i ... /<!--end welcome-->/i)
}
```

Такая конструкция позволяет выводить куски многострочного html кода(д `..`). Условия в таких операторах можно ставить и разнотипными

```
$file=qr/2345/;
while(<F>){
 print if(/^$/ .. 10); #увидим, что находится от пустой до 10-й ст
 print if(/^\001/ .. /$file/); #выведет все, что после нуля и до т
}
```

Программа чтения почтовых адресов из mbox или sent-mail:

```
while(<F>){
 next unless /^From:?\s/i .. /^$/;
 while (/([^<>(,;)\s]+\@[^<>(,;)\s]+)/)g{
 print "$1\n" unless $test{$1}++;
 }
}
```

запускается `./regex.pl /root/mail/sent-mail` и выводит каждый емейл п

# Использование встроенных переменных

- `$'` - подстрока, следующая за совпадением.
- `$&` - совпадение с шаблоном поиска
- `$`` - подстрока, расположенная перед совпадением
- `$^R` - результат вычисления утверждения в теле шаблона
- `$n` - n-ный фрагмент совпадения
- `\n` - n-ный фрагмент совпадения вызываемый в самом шаблоне
- `$+` - фрагмент совпадения
- `$*` - разрешает выполнять поиск в многострочных файлах
- `@-` - спецмассив, который содержит начальную позицию найденного с
- `@+` - массив, содержащий позицю последнего найденного слова

`$&` - совпадение с шаблоном поиска, при последней операции поиска или
переменную переопределять как вздумается нельзя.

`$'` подстрока за совпадением с шаблоном поиска, е также можно только чи

`$`` - подстрока, расположенная перед совпадением, разрешается только е ч

`$^R` - результат вычисления утверждения в теле шаблона для последнего и
или вызывается внешняя программа:

```
$qwer="lala";
$qwer=~ /x(?{$var=5})/;
print $^R;
5
```

`$+` - фрагмент совпадения в шаблоне, который в нем был последним в кр
`$+`.

`$*` - разрешает выполнять поиск в многострочных файлах, булева перем
шаблона поиска `^` и `$` сопоставляются позициям перед и после внутрен
начала текста и до конца текста:

```
$kim="lala\nfa\eti\nzvuki...";
$kim=~~ /^eti/; #совпадение не нашлось
$*=1;
$kim=~~ /^eti/; #совпадение нашлось
```

`$n` - `n`-ный фрагмент совпадения:

```
print "$1 $2 $3\n" if(/^(\d)(\w)(\W)$/);
```

\n - n-ный фрагмент совпадения вызываемый в самом шаблоне, например

```
/a href=(['"])(.*?)\1>/
```

Например нужно занести в массив только цифры из строчки "12@#34@@#@@

```
$_='12@#34@@#@@###34@@##67#@@#@#@34';
s/@/#/g;
s/(#)\1+/$1/g;
print join /\n/, split /#/, $_;
```

Регулярное выражение s/(#)\1+/$1/g; использует повторение переменной заменяет все подряд идущие # между цифрами на одну #, содержащуюся шаблона или шаблон указать в круглых скобках).

Допустим нужно определить, все ли цифры числа различны. Попробуем на

```
if(/(\d).*(?=\1)/g){
print "по крайней мере одна цифра $1 различна\n";
}
```

Выражение берет 1-ю цифру и ищет е совпадения со всеми остальны заканчивает работу. Регулярное выражение берет первое число при помо остальными числами при помощи .*(?=\1). Если первое число в строке сопостовлять второе число со всеми восемью оставшимися числами. Ес берется третье число и сравнивается со всеми остальными. И т.д., если выражение возвращает true и заканчивает свою работу, даже если в стро можно было просмотреть все повторяющиеся числа, можно воспользовать

```
$_ = '2314152467';
my @a = m/(\d)(?=\d*\1)/g ;
if (@a){
 print join(',',@a)," - Repeat\n";
}
else{
 print "Ok\n" ;
}
```

Этот усовершенствованный код работает до тех пор, пока не будут найд есть.

В perl 5.6 вводятся переменные @- и @+, комбинация которых может

совпадения шаблона переменная `$-[0]` содержит начало соответсвия текс
конец соответсвия текста шаблону. В начале поиска обе являются нулями.
`$`` , `$&`, и `$'`:

```
$do = substr($stroka, 0, $-[0]);
$sovpalo = substr($stroka, $-[0], $+[0] - $-[0]);
$posle = substr($stroka, $+[0]);
```

Например:

```
$test="11-231234";
$test=~/\d{2}-\d{6}/;
print "$-[0], $+[0]";
0, 9
```

Соответственное переменные `$#-` и `$#-` указывают размерность массивов `@`

Переменная $^N.

## Как работают регулярные выражения

Регулярные выражения, использующие квантификаторы, могут порождат
возвратом (backtracking). Чтобы произошло совпадение текста с шабло
текстом и всем регулярным выражением, а не его частью. Начало шаблона
поначалу срабатывает, но впоследствии приводит к тому, что для части
несоответствие между текстом и шаблоном. В таких случаях perl во
соответствия между текстом и шаблоном с самого начала, ограничивая "ж
процесс и называется "перебор с возвратом"). Перечислим квантификатор

- `*` - ноль или несколько совпадений,
- `+` - одно или несколько совпадений,
- `?` - ноль совпадений или одно совпадение,
- `{n}` - ровно `n` совпадений,
- `{n,}` - по крайней мере `n` совпадений,
- `{n,m}` - от `n` до `m` совпадений.

Например квантификатор `+` соответствует фразе "один или несколько" и
принцип перебора с возвратом на примере квантификатора `+`:

```
'aaabc' =~/a+abc/;
```

a+ сразу в силу жадности совпадает с тремя a:

```
(aaa)bc
```

но после aaa не следует строка "abc", а следует "bc". Поэтому резул
откатиться назад и вернуть с помощью a+ два a: (aa)abc т.е. на втором шаг

Рассмотрим пример работы еще одного жадного квантификатора *(ноль и

```
 amxdemxg /.*m/
```

Сначала будет найдена вся строка abcdebfg в силу жадности .*, потом ква
с буквой m, произойдет ошибка. Квантификатор .* отдаст одну букву и ег
снова нет буквы m. Будет отдана еще одна буква и снова не будет найден
квантификатор .* будет содержать подстроку amxde, за которой уже стоит
смотря на то, что в строке amxdemxg содержится не одна буква m. Потому
жадностью, т.е. находят максимально возможное совпадение.

Допустим нужно найти совпадение:

```
$uu="How are you? Thanks! I'm fine, you are ok??";
$uu=~s/.*you//;
print $uu;
```

Квантификатор .* оставит текст " are ok??", а вовсе не "? Thanks! I'm
ограничитель ?, который вместе со знаком квантификатора означает макси

```
$uu="How are you? Thanks! I'm fine, you are ok??";
$uu=~s/.*you//;
print $uu;
```

то переменная $uu будет содержать текст "? Thanks! I'm fine, you are

Предположим нужно найти совпадения типа network workshop, т.е. перекр

```
$u='network';
$m='workshop';
print "перекрытие $2 найдено: $1$2$3\n" if("$u $m" =~/^(\w+)(\w+)
```

$1 сразу берет все слово в $u, но дальше идет еще один максимальный кв
надо и он забирает из переменной \1 букву k(причем только одну):

```
#!/usr/bin/perl
$uu="asdfg asdf";
$uu=/(\w+)(\w+)\s(\w+)(\w+)/;
print "$1 $2##$3 $4";
asdf g##asd f
```

далее пошаговая работа regex выглядит примерно так:

```
1: 'networ''k'=> '\sk' совпадает ли с '\sworkshop' falure
2: 'netwo''rk'=> '\srk' совпадает ли с '\sworkshop' falure
3: 'netw''ork'=> '\sork' совпадает ли с '\sworkshop' falure
4: 'net''work'=> '\swork' совпадает ли с '\sworkshop' ok
```

и в результате программа выдаст:

```
перекрытие work найдено: networkshop
```

Данный регексп не сработает, если

```
$u='networkwork';
$m='workshop';
```

шаблон найдет перекрытия `workwork`, а не `work`. Чтобы этого избежать, н
`(\w+) \2(\w+)$/`

Квантификатор действует только на предшествующий ему элемент шабл
будет соответствовать последовательности из одной или нескольких строч
цифр, а не последовательности, составленной из чередующихся цифр и б
которую действует квантификатор, используются круглые скобки: `(\d{2}(`

## Логические операции в регулярных выражен

В регулярных выражениях perl есть синтаксические выражение, позвол
логические конструкции:

- `(?= шаблон)` - после этой точки есть фрагмент текста, который
  выражению
- `(?! шаблон)` - после этой точки нет текста, который бы соответствова
- `(?<= шаблон)` - перед этой точкой есть фрагмент текста, соответствую
- `(?<! шаблон)` - перед этой точкой нет фрагмента текста, сос
  выражению.

- `(?#текст)` - комментарий. Текст комментария игнорируется.
- `(?:шаблон)` или `(?модификаторы:шаблон)` - группирует элементы ш скобок, не создает нумерованной переменной. Например, модифи строчными и заглавными буквами, однако область действия этог указанным шаблоном.
- `(?=шаблон)` - "заглядывание вперед". Требует, чтобы после текущей данному шаблону. Такая, конструкция обрабатывается как услов включается В результат поиска. Например, поиск с помощью коман следуют один или несколько "пробельных символов", однако сами они
- `(?!шаблон)` - случай, противоположный предыдущему. После т соотносимого с заданным шаблоном. Так, если шаблон `w+(?=\s)` - э символ", то шаблон `w+(?!\s)` - это слово, за которым мет "пробельног
- `(?<=шаблон)` - заглядывание назад. Требует, чтобы перед текущей точк шаблон `(?<=\s)w+` интерпретируется как слово, перед которым им заглядывания вперед, заглядывание назад может работать только символов).
- `(?<!шаблон)` - отрицание предыдущего условия. Перед текущей точк заданным шаблоном. Соответственно, от команды `/(?<!\s)w+/` тр пробельного символа.
- `(?{код})` - условие (мнимый символ), которое всегда выполняется фигурных скобках. Вы можете использовать эту конструкцию, тольк `use re 'eval'`. При последовательном соотнесении текста и шаблона выполняется указанный код. Если полного соответствия для остав возврате левее данной точки шаблона вычисления, проделанные с назад. (Условие является экспериментальным. В документации, пр детальное рассмотрение (с примерами) работы этого условия применения.)
- `(?>шаблон)` - "независимый" или "автономный" шаблон. Использу поскольку запрещает "поиск с возвратом". Такая конструкция соотве заданный шаблон, если его закрепить в текущей точке без учета пос шаблон `(?>a*)ab` в отличие от `a*ab` не может соответствовать никак шаблон `a*`, он съест все буквы `a`, не оставив ни одной шаблону `ab`. (Д `*` будет ограничен за счет работы поиска с возвратами: после то соответствие между шаблоном и текстом, perl сделает шаг назад и ум конструкцией `a*`.)
- `(?(условие)шаблон-да|шаблон-нет)` или `(?(условие)шаблон-да)` - ус или иной шаблон в зависимости от выполнения заданного условия. Бо
- `(?модификаторы)` - задает модификаторы, которые локальным обра отличие от глобальных модификаторов, имеют силу только для теку

круглых скобок, охватывающих конструкцию, Например, шаблон (
учета регистра.

Поиск повторяющихся слов в регулярном выражении осуществляется при
был приведен пример их использования для выбирания всех адресов рисун

```
m{SRC\s*=\s*(["'])http://(.*?)\1\s+(.*?)WIDTH="100" HEIGHT="100"(.
```

`(["'])` - найти либо `"` либо `'` либо ничего, т.к. `src=http://` может быть без
из этих трех позиций, через минимальное количество символов(регу
заносится в специальную переменную `\1`, которая вне `m/.../` может
вызывается в его левую половину как `$1`). Дальше после `*.gif|*.jpg|*.b`
один пробел `\s+`, т.к. броузеры воспримут подстроку `src=file.gifbor`
`gifborder=0`. Поэтому данное регулярное выражение вполне исправно раб
в `img src` ставится полный адрес, т.е. начинающийся с `http://` Для дру
пути в ссылках используя `base href`, если есть или его `url`. Если нужно на
в строке, то это реализуется примерно так:

```
while($str=~/WHAT/g){$n++}
$n++ while $str=~/WHAT/g;
$n++ while $str=~/(?=WHAT)/g;#для перекрывающихся совпадений
for($n=0; $n=~/WHAT/g; $n++){}
```

Каждое кратное совпадение

```
(++$n % 6) == 0;
```

Нужное Вам совпадение:

```
$n=($str=~/WHAT/gi)[6]; #допустим шестое
```

Или каждое четное совпадение

```
@mass=grep{$n++ %2==0} /WHAT/gi;
```

для нечетного нужно написать внутри `grep`: `$n++ %2==1` Логические опер
нужно найти последнее совпадение, то можно воспользоваться отрицанием

```
m#PATTERN(?!.*PATTERN)$#
```

т.е. нийти какой-то `PATTERN`, при этом не должно найтись что-то еще(
совпадение;

Минимальные квантификаторы `*?`, `+?`, `??`,`{}?`

допустим нужно найти двойку, перед которой не стоит 3 или пробел:

```
print "$1\n" while m%2(?![3\s])gm%;
```

используется условие по отрицанию, `A(?!B)`: найти `A`, перед которым не на
стоит 3 или пробле(`\s`), то можно воспользоваться:

```
print "$1\n" while m%2(?=[3\s])gm%;
```

или

```
print "$1\n" while m%2(?![^3\s])gm%;
```

где используется `^`, `[^3\s]`, который значит следущее: в класс символов, к
или другими словами найти все кроме `3` и `\s`.

Допустим существует HTML-документ, в котором произвольное число вл
Требуется "вырезать" по очереди самые вложенные таблицы (не соде
соответственно, выводить. И так - рекурсивно до конца вырезать из
программа, реализующая эту задачу при помощи логического оператора `(?`

```
#!/usr/bin/perl -wT
$file=qq|s<table>aaa bbb
<table>cc<table>ccc
<table> 2<table>bb</table> <table>cc</table> </table></table>cc
</table>
ddd</table>d
|;
print $file;
&req($file);
sub req {
if($file=~m%(<table>((?!.*<table>).*?)</table>)%igs){
$file=~s%(<table>((?!.*<table>).*?)</table>)%%igs;
print "Virezali --$1--";
&req($file);
}
return $file;
}
```

Продолжаем рассматривать логические операторы в регулярных выражени

Регексп истиннен, если `/AM|BMA/` или `/AM/ || /BMA/` и если есть перекрыти

```
/^(?=.*AM)(?=.*BMA)/s
```

Выражение истинно если `/AM/ и /BMA/` совпадают при перекрытии которо

```
/AM.*BMA|BMA.*AM/s
```

Выражение истинно, если шаблон `/ABC/` не совпадает:

```
!~/ABC/
```

или

```
/^(?:(?!ABC).)*$/s
```

Выражение истинно, если `ABC` не совпадает, а `VBN` совпадает:

```
/(?=^(?:(?!ABC).)*$)VBN/s
```

Несовпадение можно проверить несколькими способами:

```
unless($str =~ /MMM/){...}
if(!($str =~ /MMM/)){...}
if($str !~ /MMM/){...}
```

Для обязательного совпадения в двух шаблонах:

```
unless ($str !~ /MMM/ && $str !~ /BBB/){...}
#или
if ($str =~ /MMM/ && $str =~ /BBB/){...}
```

Хотя бы в одном

```
unless ($str !~ /MMM/ || $str !~ /BBB/){...}
#или
if ($str =~ /MMM/ || $str =~ /BBB/){...}
```

Регулярные выражения - основа работы с операторами `m/.../` и `s/.../.` качестве аргументов. Разберемся, как устроено регулярное выражение отдельных слов в строке:

```

$text = "Perl is the subject.";
$text =~/\b([A-Za-z]+)\b/;
print $1;
```

Выражение `\b([A-Za-z]+)\b` включает в себя группирующие метасимво класс всех латинских букв `[A-Za-z]` (он объединяет заглавные и строч

указывает на то, что требуется найти один или несколько символов рассма
выражения, как это было в предыдущем примере, могут быть очень слс
случае регулярное выражение состоит из следующих компонентов:

Совпадение с любым символом

В perl имеется еще один мощный символ - а именно, точка `(.)`. В шабл
символа новой строки. Например, следующая команда заменяет в строк
модификатор `g`, обеспечивающий глобальную замену):

```
$text = "Now is the time.";
$text =~ s/./*/g;
print $text;
********************
```

А что делать, если требуется проверить совпадение именно с точкой?
`[{^$*+?.`), играющие в регулярном выражении осббую роль) назы
метасимволами, и если вы хотите, чтобы они внутри шаблона инт
метасимволу должна предшествовать обратная косая черта. Точно так
символу, используемому в качестве ограничителя для команды `m/...`
встречается внутри шаблона и не должен рассматриваться как ограничите

```
$line = ".Hello!";
if ($1ine =- m/\./) {
 print "Shouldn't start a sentence with a perlod!\n";
}
Shouldn't start a sentence with a perlod!
```

Если нужно найти самый короткий текстовый фрагмент `/QQ(.*?)FF/` в `"QQ
ff"`. Шаблон всегда находит левую строку минимальной длины, которая
строка в этом примере. Для правильного шаблона нужно воспользоватьс
выражениях: `/QQ((?:(?!QQ).)*)FF/`, т.е. сначала `QQ`, потом `не QQ`, потом `FF`

Конструкции `(?<=шаблон)` и `(?<!шаблон)` работают только с шаблонами, с
символов. Иными словами, в шаблонах, указываемых для `(?<=...)` и `(?<!`

Эти условия полезны, если нужно проверить, что перед определенным фр
нужная строка, однако ее не требуется включать в результат поиска
используются спе-циальные переменные `$&` (фрагмент, для которого
регулярным выражением), `$`` (текст, предшествующий найденному фрагме
фрагментом). Более гибким представляется применение нумерованных пер

отдельные части найденного фрагмента.

В следующем примере ищется слово, за которым следует пробел, но сам п

```
$text = "Mary Tom Frank ";
while ($text =~ /\w+(?=\s)/g) {print $& . "\n";}
Mary
Tom
Frank
```

Того же результата можно добиться, если заключить в круглые скобки и использовать ее как переменную `$1`:

```
$text = "Mary Tom Frank ";
while ($text =~ /(\w+)\s/g) {
 print $1 . "\n";
}
Mary
Tom
Frank
```

Следует четко понимать, что вы имеете в виду, когда используете то и пример:

```

$text="Mary+Tom";
if($text=~m|(?!Mary\+)Tom|){
print "Tom is without Mary!\n";
}
else{
print "Tom is busy...\n";
}
```

Вопреки нашим ожиданиям, perl напечатает: `Tom is without Mary!` Это п различные начальные точки входной строки, от которой начинается сопо поздно доберется до позиции, расположенной прямо перед именем "`Tom`". текущей точки не находился текст *`Mary+`", и это условие для рассматри последовательно проверяет, что после текущей точки следуют буквы "T", (после проверки условия `(?!Mary\+)` текущая точка остается на месте). подстрокой "Tom" и шаблоном, поэтому команда поиска возвращает значе

Регулярное выражение `(?!Mary\+)....Tom`, резервирующее четыре симв выше случая выведет то, что требовалось, но выдаст ошибочный отве

символов:

```
$text="O, Tom! ";
if($text =~ m|(?!Mary\+)....Tom|){
print "Tom is without Mary!\n";
}
else{
print "Tom is busy...\n";
}
Tom is busy...
```

Наконец, если более точно сформулировать, чего требуется, получится нуж

```
$text="Mary+Tom";
if($text=~m|(?<!Mary\+)Tom|){
print "Tom is without Mary!\n";
}
else{
print "Tom is busy...\n";
}
Tom is busy...
```

Вспомнить и написать про строчку вида

```
push @mass, $li unless($li=~m/(([2 .. 12]).*?1995)|(([6 .. 12]).*?
```

; perldoc perlop [0-9.]

Модификаторы команд `m/.../` и `s/.../.../`

В perl имеется несколько модификаторов, используемых с командами `m/..`

- `i` - игнорирует различие между заглавными и строчными буквами.
- `s` - метасимволу "точка" разрешено соответствовать символам `\n`.
- `m` - разрешает метасимволам `^` и `$` привязываться к промежуточны влияет на работу метасимволов `\A`, `\Z` и `\z`.
- `x` - игнорирует "пробельные символы" в шаблоне (имеются в виду "и пробелы, созданные через escape-последовательности). Разрешает исп
- `g` - выполняет глобальный поиск и глобальную замену.
- `c` - после того как в скалярном контексте при поиске с модификатором не позволяет сбрасывать текущую позицию поиска. Работает тольк модификатором `g`.
- `o` - запрещает повторную компиляцию шаблона при каждом обра

замены, пользователь, однако, должен гарантировать, что шаблон фрагмента кода.

- `e` - показывает, что правый аргумент команды `s/.../.../` - это фрагме для подстановки будет использовано возвращаемое значение - возмож
- `ee` - показывает, что правый аргумент команды `s/.../.../` - это строк выполнить как фрагмент кода (через функцию `eval`). В качеств возвращаемое значение - возможно, после процесса интерполяции

Особенности работы команд `m/.../` и `s/.../.../`

До сих пор мы рассматривали регулярные выражения, используемые в `s/.../.../`, и не особо интересовались, как работают эти команды. Настал

Команда `m/.../` ищет текст по заданному шаблону. Ее работа и возвращ скалярном или списковом контексте она используется и имеется ли модиф

Команда `s/.../.../` ищет прототип, соответствующий шаблону, и, если н на новый текст. Без модификатора замена производится только д модификатором `g` выполняются замены для всех, совпадений во входно результата число успешных замен или пустую строку (условие ложь false) качестве анализируемого текста используется `$_` (режим по умолчанию) и с помощью оператора `=~` или `!~`. В случае поиска (команда `m/.../`) констр `=~` или `!~`, может и не быть переменной. В случае замены (команда `s` скалярная переменная, или элемент массива, или элемент хэша, или же объектов.

Вместо косой черты в качестве ограничителя для аргументов команд `m` любой символ, за исключением "пробельного символа", буквы или ци использовать символ комментария, который будет работать как ограничите

```
$text="ABC-abc";
$text =~ s#B#xxx#ig;
print $text;
AxxxC-axxxc
```

В качестве ограничителей не стоит использовать вопросительный знак и а с такими ограничителями обрабатываются специалиным образом. Если черты в качестве разделителя, то букву m можно опустить:

```
```

```
while (defined($text = <>)) { if ($text =~/^exit$/i) {exit;} }
```

Если в качестве ограничителя для команды `m/.../` используется вопрос
опустить. Однако шаблоны, ограниченные символом `?`, в случае поиска
наличия или отсутствия начальной `m`). А именно, они ведут себя как тр
потом выдают состояние ложь (false), пока их не взведут снова, вы
блокировки сразу всех конструкций `?...?`, локальных для данного п
сценария проверяет, есть ли в файле пустые строки:

```
while (<>)
if (?^$?) {print ."There is an empty line nere.\n";} continue {
reset if eof; #очистить для следующего файла
}
```

Диагностическое сообщение будет напечатано только один раз, даже есл
строк. Команда поиска с вопросительным знаком относится к подозрител
новые версии perl. 1 В качестве ограничителей можно также использовать

```
while (<>){
 if(m/^quit$/i){exit;}
 if(m(^stop$)i){exit;}
 if(m[^end$]i) {exit;}
 if(m{^bye$}i) {exit;}
 if (!1)<^exit$>i) {exit;}
}
```

В случае команды `s/.../.../` и использования скобок как ограничите
второго аргумента могут выбираться независимо:

```
$text =~ "Perl is wonderful";
$text =~ s/is/is very/;
$text =~ s[wonderful]{beautiful};
$text =~ s(\.)/!/;
print $text;
Perl is very beautiful!
```

Предварительная обработка регулярных выражений

Аргументами команд `m/.../` и `s/.../.../` являются регулярные выр
интерполируются подобно строкам, заключенным в двойные кавычки В о
выполняется интерполяция имен типа `$)`, `$|` и одиночного `$` - perl счит
метасимволу конца строки, а не специальной переменной. Если же в
оказался пустой строкой, perl использует последний шаблон, который прим

Если вы не хотите, чтобы perl выполнял интерполяцию регулярного в использовать апостроф (одиночную кавычку), тогда шаблон будет вести апострофы. Однако, например, в случае команды замены `s/.../.../` описывается чуть дальше) для второго аргумента будет выполняться и заключен в апострофы.

Если вы уверены, что при любом обращениик команде поиска или замень несмотря на интерполяцию, скалярные переменные внутри шаблона не задать модификатор `o`. Тогда perl компилирует шаблон в свое внутреннее п данной командой поиска или замены. При остальных обращениях к коман значение. Однако, если внезапно изменить значение переменных, задейс заметит.

Команда замены `s/.../.../` использует регулярное выражение, указанное текста. Поскольку оно обрабатывается (интерполируется) после того, как нем можно, в частности, использовать временные переменные, созданны мы последовательно заменим местами пары слов, заданных во входном пробелу:

```
$text = "One Two Three Four Five Six";
$text =- s/(\w+)\s*(\w+)/$2$1/g;
Two One Four Three Six Five
```

Однако perl допускает и более сложные способы определения замен `s/.../.../` указать модификатор `e`, то в качестве второго аргумента выполнить (например, вызвать функцию). Полученное выражение будет При этом после вычисления текстового значения, но пер д его подстановк аналогичный процессу интерполяции текстовых строк, заключенных в дв реализуется, если задан модификатор `ee`. В этом слу-чае второй аргум выражение, которое сперва надо вычислить (то есть интерполировать), встроенную функцию `eval`) и только после второй интерполяции пол найденного текста.

Работа команды `m/.../` в режиме однократного поиска В скалярном конте возвращает логическое значение - целое число `1` (истина (true)), если поис (ложь (false)), если нужный фрагмент текста найти не удалось. Если вну заключенные в круглые скобки, то после операции поиска создаются нуме содержится текст, соответствующий круглым скобкам. В частности, если то в случае успешного поиска переменная `$1` будет содержать текст, соо поиска можно также использовать специальные переменные `$&`, `$'`, `$'` и `$+`

```
$text = "---one---two---three---";
$scalar = ($text =' m/(\w+)/);
print "Result: $scalar ($1).";
Result: 1 (one).
```

Если вы используете команду `m/.../` в списковом контексте, то возвраща
ли группы из круглых скобок в вашем шаблоне. Если они есть (то есть ес
то после успешного поиска в качестве результата будет получен список, со
(`$1`, `$2`,...):

```
$text = "---one, two, three---";
 array = ($text ='m/(\w+),\s+(\w+),\s+(\w+)/);
print join "=", array;
one=two=three.
```

В отличие от ранних версий, perl 5 присваивает значения нумерованны
работает в списковом контексте:

```
$text = "---one, two, three--- ";
($Fa, $Fb, $Fc) = ($text=-m/(\w+),\s+(\w+),\s+(\w+)/);
print "/$Fa/$Fb/$Fc/\n";
print "$1=$2=$3.\n";
/one/two/three/
one=two::three.
```

Если же в шаблоне нет групп, выделенных круглыми скобками, то в случа
состоящий из одного элемента - числа `1`. При неудачном поиске незави
скобки, возвращается пустой список:

```
$text = "---one, two, three--- ";
@array = ($text=~ m/z\w+/);
print "Result: /", @array, "/\n";
print "Size: ", $#array+1, ".\n";
Result://
Size: 0.
```

Обратите внимание на разницу между пустым и неопределенным спискам

Работа команды `m/.../` в режиме глобального поиска

Команда `m/.../` работает иначе, если указан модификатор `g`, задающий г
по всему тексту. Если оператор используется в списковом контексте и в
случае удачного поиска возвращается список, состоящий из всех найденны

```
$text = "---one---two-~-three---";
@array = ($text =~m/(-(\w+))/);
print "Single: [", join(", ", array),"].\n";
@array = ($text =~m/(-(\w+))/g);
print "Global: [", join(", ", array),"].\n";
Single: [-one, one].
Global: [-one, one, -two, two, -three, three].
```

Если же в шаблоне нет групп круглых скобок, то оператор поиска возвр
шаблона, то есть ведет себя так, как если бы весь шаблон был заключен в к

```
$text = "---one---two---three--";
@array = ($text =~m/\w+/);
print "Result: (", join(", ", @array), ").\n";
Result: (one, two, three).
```

В случае неудачного поиска, как и в предыдущих вариантах, возвращается
модификатором `g` команда `m/.../` ведет себя сивершенно особым об
переменная, стоящая слева от оператора `=~` или `!~`, при поиске с мод
свойства - в нее записывается последнее состояние. При каждом послед
кода поиск будет продолжаться с того места, на котором он остановилс
команда подсчитывает количество букв x в заданной строке текста:

```
$text = "Here is texxxxxt.";
$counter = O;
while ($text =~ m/x/g){
print "Found another x.\n";
$conter++;
print "Total amount = $counter.\n";
Found another x. Found another x.
Found another x.
Found another x. Found another x.
Total amount = 5.
```

Состояние (точнее, позиция) поиска сохраняется даже в случае пер
имеющему модификатор `g`. Неудачный поиск сбрасывает значение в исхо
`m/.../` не указан модификатор `c` (то есть команда должна иметь вид `m/.`
которого выполняется поиск, также сбрасывает позицию поиска в исход
текстовой строки последовательно извлекаются и выводятся пары им
закончится:

```
$text = "X=5; z117e=3.14l6; temp=lQ24;";
```

```
$docycle = 1; $counter = 0;
while ($docycle) { undef $name; undef $value;
if ($text =~ m/(\w+)\s*=\s*/g) {$name = $1;} if ($text =~ m/([\d\.
if (defined($name) and defined($value)) { print "Name=$name, Value
$counter++,
}else{
$docycle = 0;
}
}
print "I have found $conter values.\n";
Name=X, Value=5.
Name=z117e, Value=3.1416. Name=temp, Value=1024.
I have found 3 values.
```

Позиция, на которой остановился поиск, может быть прочитана и даже функции perl `pos`. В шаблоне на текущую позицию поиска можно с следующем примере из строки последовательно извлекаются буквы `p`, `o` и

```
$index = 0;
$_ = "ppooqppqq";
while ($index++ < 2) {
print "1: '";
print $1 while /(o)/gc; print "', pos=", pos, "\n";
print "2: '";
print $1 if /\G(q)/gc; print "', pos=";' pos, "\n";
print "3: '";
print while /(p)/gc; print "', pos=",pos, "\n";
}

1: 'oo', pos=4;
2: 'q', pos=7;
3: 'pp', pos=4;
1: '', pos=7;
2: 'q', pos=8;
3: '', pos=8;
```

В документации perl приводится основанный на этом механизме лексического разбора текста. В нем каждая последующая команда поиска выполняться с того места, где завершила свою работу предыдущая. С примером (страница руководства `perlop`, раздел "`Regexp Quote-Uke Op`
если вы хотите расширить доступный вам инструментарий perl!

Замена строк с помощью команды `tr/.../.../`

Кроме команд `m/.../` и `s/.../.../` строки можно обрабатывать с помощь `y/.../.../`):

```
tr/список1/список2/модификаторы;
y/список1/список2/модификаторы;
```

В отличие от `m/.../` и `s/.../.../`, эта команда не использует шаблон посимвольную замену, подставляя в текст вместо литер из первого списк списка. Например, в следующем случае производится замена литер "`i`" н `=~ tr/i/o/; print $text; My name is Tom.`

В качестве списков используются идущие друг за другом символы, не р строки, чем списки). В отличие от шаблонов команд `m/.../` и `s/.../` интерполируются (то есть подстановки значений вместо имен пер последовательности, указанные внутри аргументов, обрабатываются пр команда `tr/.../.../` по умолчанию работает с переменной `$_`:

```
while (<>){
tr/iI/jJ/;
print;
```

В качестве списков можно указывать диапазоны символов - как, н заменяющем строчные буквы на заглавные: `$text = "Here is the text. HERE IS THE TEXT.`

Как и в случае `m/.../` и `s/.../.../`, команда `tr/.../.../` не требует и качестве ограничителей. Можно использовать практически любой символ, также парные скобочные конструкции.

Команда `tr/.../.../` возвращает число успешных замен. В частности, возвращает число ноль. Это позволяет, например, подсчитать с пом вхождений буквы `x` в строку `$text`, не меняя содержимого этой переменно `= ($text =~tr/x/x/); print $xcount; 1` Если у команды `tr/.../..` "Модификаторы команды `tr/.../.../`"), то ее аргументы при обычных у Если второй аргумент длиннее первого, то он усекается до длины перв эквивалентна команде `tr/abc/012/`. Если первый аргумент длиннее вт аргумента необходимое число раз повторяется его последний символ команде `tr/0123456789/abccccccccc/`. Если же второй, аргумент пуст, то него первый аргумент.

Как легко заметить, если второй аргумент пуст, то (при отсутствии производит никаких действий, а возвращаемое ею значение равно числу обрабатываемым текстом. Например, следующая команда подсчитывает ко

```
$text = "Pi=3.1415926536, e=2.7182";
$digit_counter=($text =~ tr/0-9//);
print $digit_counter;
16
```

Команда `tr/.../.../` работает без рекурсии, просто последовательно зам
для замены заглавных букв на строчные, и на-оборот, достаточно выполни

```
$text = "MS Windows 95/98/NT";
$text =" tr/A-Za-z/a-zA-Z/;
print $text;
ms WINDOWS 95/98/nt
```

Если в списке, указанном в качестве первого аргумента, есть повторяющ
первое вхождение символа:

```
$text = "Billy Gates";
$text =~ tr/ttt/mvd/;
print $text;
Billy Games
```

Модификаторы команды `tr/.../.../`

Команда `tr/.../.../` допускает использование следующих модификаторо

- `d` - удаляет непарные символы, не выравнивая аргументы по длине.
- `c` - в качестве первого аргумента использует полный список из 256 символов.
- `s` - удаляет образовавшиеся в результате замены повторяющиеся симв

Если указан модификатор `d`, а первый аргумент команды длиннее второ
имеющие соответствия со вторым списком, удаляются из обрабатыва
латинские буквы и заменяем пробелы на слэши:

```
$text = "Here is the text.";
$text =~ tr[ a-z][/]d;
print $text;
H///.
```

Наличие модификатора `d` - единственный случай, когда первый и вт
относительно друга, В остальных вариантах второй аргумент либо у
повторяется до тех пор, пока аргументы не сравняются, либо, если второй

берется копия первого.

Если указан модификатор `c`, то в качестве первого аргумента рассматр
Например, заменим на звездочки все символы, кроме строчных латинских

```
$text = "Here is the text,";
$text =' tr/a-z/*/c;
print $text;
*ere*is*the*text*
```

Если указан модификатор `s`, то в случае если замещаемые символы образ
сокращаются до одного. Например, заменим слова, состоящие из латинс
черты:

```
$text = "Here is the text.";
$text ="tr(A-Za-z)(/)s;
print $text;
/ / / /.
```

Без модификатора `s` результат был бы другим:

```
$text = "Here is the text.";
$text =' tr(A-Za-z)(/);
print $text;
//// // /// ////.
```

Примеры:
1. Заменить множественные пробелы и нетекстовые символы на одиночны

```
$text = "Here is the text."
$text =~ tr[\000-\040\177\377][\040]s;
print $text;
Here is the text.
```

2. Сократить удвоенные, утроенные и т.д. буквы;

```
$text = "Here is the texxxxxxt.";
$text =~ tr/a-zA-Z/s;
print $text;
Here is the text.
```

3. Пересчитать количество небуквенных символов:

```
$xcount=($text =~ tr/A-Za-z//c);
```

4. Обнулить восьмой бит символов, удалить нетекстовые символы:

```
$text =- tr{\200-\377}{\000-\l77};
$text =~ tr[\000-\037\177][]d;
```

5. Заменить нетекстовые и 8-битные символы на одиночный пробел:

```
$text =~ tr/\021-\176/ /cs;
```

Поиск отдельных слов

Чтобы выделить слово, можно использовать метасимвол \S соответствующ

```
$text = "Now is the time.";
$text =- /(\S+)/;
print $1;
Now
```

Однако метасимвол `\S` соответствует также и символам, обычно не ис
отобрать слова, составленные из латинских букв, цифр и символов подче
`\w`:

```
$text = "Now is the time.";
$text =~ /(\w+)/;
print $1;
Now
```

Если требуется включить в поиск только латинские буквы, надо использов

```
$text = "Now is the time.";
$text =~ /([A-Za-z]+)/;
print $1;
Now
```

Более безопасный метод состоит в том, чтобы включить в шаблон мнимые

```
$text = "How is the time.";
$text=~/\b([A-Za-z]+)\b/;
print $1;
Now
```

Привязка к началу строки

Началу строки соответствует метасимвол (мнимый символ) `^`. Чтобы ш
символ в начале регулярного выражения. Например, вот так можно провер

```
$line = ".Hello!";
if($line=~m/^\./){
```

```
 print "Shouldn't start a sentence with a period!\n";
}
Shouldn't start a sentence with a period!
```

Чтобы точка, указанная в шаблоне, не интерпретировалась как метасимво
косую черту.

Привязка к концу строки

Чтобы привязать шаблон к концу строки, используется метасимвол (м
используем привязку шаблона к началу и к концу строки, чтобы убедиться

```
while(<>){
 if(m/"exlt$/) {exit;}
}
```

Поиск чисел

Для проверки того, действительно ли пользователь ввел число, мож
Метасимвол `\D` соответствует любому символу, кроме цифр. Например, сл
введенный текст представляет собой целое значение без знака и паразитны

```
$test = "Hello!";
if($text =~ /\D/){
print "It is not a number.\n";
}
It is not a number.
```

То же самое можно проделать, использовав метасимвол \d:

```
$text = "333";
if($text =~ /^\d+$/){
print "It is a number.\n";
}
It is a number.
```

Вы можете потребовать, чтобы число соответствовало привычному ф
десятичную точку, перед которой стоит по крайней мере одна цифра и, воз

```
$text= "3,1415926";
if($text =~ /^(\d+\.\d*|\d+)$/){
print "It is a number.\n";
}
It is a number.
```

Кроме того, при проверке можно учитывать тот факт, что перед числом пустое место):

```
$text = "-2.7182";
if ($text =~ /^([+-]*\d+)(\.\d*|)$/) {
print "It is a number.\n";
```

Поскольку плюс является метасимволом, его надо защищать обратной скобок, то есть класса символов, он не может быть квантификаторам. Зна играет роль оператора диапазона и поэтому должен защищаться обратной шаблона он никак не может обозначать диапазон, и поэтому обратная к строгая проверка, требует, чтобы знак, если он присутствует, был только о

```
$text = "+0.142857142857142857";
if ($text =~ /^(+|-|)\d+(\.\d*\)$/) {
 print "It is a number.\n";
}
It is a number.
```

Альтернативные шаблоны, если они присутствуют, проверяются слева на только найдено соответствие между текстом и шаблоном. Поэтому, на `(\.\d*|)` мог бы стать критичным, если бы не привязка к концу стр проверку того, что текст является шестна-дцатеричным числом без знака и

```
$text = "1AO";
unless (ftext =~ m/^[a-fA-F\d]+$/) {
print "It is not a hex number, \n";
}
```

Проверка идентификаторов

С помощью метасимвола `\w` можно проверить, состоит ли текст только из те символы, которые perl называет словесными (word characters)):

```
$text="abc";
if($text=~/^\w+$/){
 print "Only word characters found. \n";
}
Only word characters found.
```

Однако, если вы хотите убедиться, что текст содержит латинские б подчеркивания, придется использовать другой шаблон:

```
$text = "abc";
if($text=~ /^[A-Za-z]+$/)
{ print "Only letter characters found.\n";}
Qnly letter characters found.
```

Наконец, для проверки, что текст является идентификатором, то есть начи
символы подчеркивания, можно испольровать команду:

```
$text = "X125c";
if($text=~ /^[A-Za-z]\w+$/)
{ print "This is identifier.\n";}
This is identifier.
```

Как найти множественные совпадения

Для поиска нескольких вхождений шаблона можно использовать модифи
уже видели ранее, использует команду `m/.../` с модификатором `g` для поис

```
$text="Here is texxxxxt";
while($text=~m/x/g){
 print "Found another x.\n";
}
Found another x.
Found another x.
Found another x.
Found another x.
Found another x.
```

Модификатор `g` делает поиск глобальным. В данном (скалярном) контекст
при предыдущем поиске. Следующий поиск продолжается с отложенной
будет упорно находить первое вхождение буквы `x`, и цикл будет продолжат

В отличие от команды `m/.../` команда `s/.../.../` с модификатором `g` в
работая так, будто внутри нее уже имеется встроенный цикл поиска, по
пример за один раз заменяет все вхождения `x` на `z`:

```
$text = "Here is texxxxxt.";
$text =~ s/x/z/g;
print $text;
Here is tezzzzzt.
```

Без модификатора g команда `s/.../.../` заменит только первую букву `x`. I
значения число сделанных подстановок, что может оказаться полезным:

```
$text= "Here is texxxxxt.";
print (text =~ s/x/z/g)
5
```

Поиск нечувствительных к регистру совпадений

Вы можете использовать модификатор `i`, чтобы сделать поиск нечувств строчными буквами. В следующем примере про-грамма повторяет на экр пор, пока не будет введено `Q`, или `q` (сокращение для QUIT или quit), после

```
while(<>){
 chomp;
 unless (/^q$/i){
 print
 }
 else {
 exit;
 }
}
```

Выделение подстроки

Чтобы получить найденную подстроку текста, можно использовать кругл удобно, можно также использовать встроенную функцию substr. В следу строки нужный нам тип изделия:

```
$record = "Product number:12345
 Product type: printer
        Product price: $325";
if($record=~/Product type:\s*([a-z]+)/i){
 print "The product's type Is^$1.\n";
}
product's type is printer.
```

Вызов функций и вычисление выражений при подстановке текста

Используя для команды `s/.../.../` модификатор `e`, вы тем самым пс подставляемый текст) - это то выражение perl, которое надо вычислить. Н perl `uc` (uppercase) можно заменить все строчные буквы слов строки на заг

```
$text = "Now is the time.";
$text=~ s/(\w+)/uc($1)/ge;
print $text;
NOW IS THE TIME.
```

Вместо функции uc($l) можно поместить произвольный код, включая вызо

Поиск n-го совпадения

С помощью модификатора g перебираются все вхождения заданного ш определенная точка совпадения с шаблоном, например, вторая или тре круглыми скобками, выделяющими нужный образец, поможет вам:

```
$text = "Name:Anne Nanie:Burkart Name:Glaire Name: Dan";
while ($text =~ /Name: \s*(\w+)/g){
 ++$match;
 print "Match number $match is $1.\n";
}

Match number 1 is Anne
Match number 2 is Burkart
Match number 3 is Claire
Match number 4 is Dan
```

Этот пример можно переписать, используя цикл for:

```
$text = "Name:Anne Name:Burkart Name:Ciaire Name:Dan";
for ($match = 0;
 $text =~ /Name:\s*(\w+)/g;
        print "Match number ${\match} is $1.\n")
{}
Match nuwber 1 Is Anne
Match number 2 is Burkart
Match number 3 is Claire
Match number 4 is Dan
```

Если же вам требуется определить нужное совпадение не по номеру, а по имени пользователя), то вместо счетчика `$match` можно анализировать сод каждом найденном совпадении. Когда требуется не найти, а заменить вт применить ту же схему, использовав в качестве тела цикла выражение perl строки:

```
$text = "Name:Anne Name:Burkart Name:Claire Name:Dan";
$match =0;
$text =~ s/(Name:\s*(\w+))/ # начинается код perl
 if (++$match == 2) # увеличить счетчик
        {"Name:John ($2)"}# вернуть новое значение
 else {$1} # оставить старое значение
 /gex;
print $text;
```

```
Name:Anne Name:John (Burkart) Name:ClaireName:Dan
```

В процессе глобального поиска при каждом найденном совпадении выч
второго операнда. При его вычислении увеличивается значение счетчика,
подставляется либо старое значение текста, либо новое. Модификато
комментарии, делая код более прозрачным. Обратите внимание, что нам п
скобки, чтобы получить значение найденного текста и подставить его на п

Как ограничить "жадность" квантификаторов

По умолчанию квантификаторы ведут себя как "жадные" объекты. Н
захватывают самую длинную строку, которой может соответствовать
квантификатором. Алгоритм перебора с возвратами, используемый
квантификаторов, возвращаясь назад и уменьшая длину захваченной ст
между текстом и шаблоном. Однако этот механизм не всегда работает так
пример. Мы хотим заменить текст "That is" текстом "That's". Однако в си
выражение "`.*is`" сопоставляется фрагменту текста от начала строки и до

```
$text = "That is some text, isn't it?";
$text =~ s/.*is/That's/;
print $texts;
That'sn't it?
```

Чтобы сделать квантификаторы не столь жадными, а именно заставит
которой сопоставимо регулярное выражение, после квантификатора ну
самым квантификаторы принимают следующий вид:

- `*?` - ноль или несколько совпадений,
- `+?` - одно или несколько совпадений,
- `??` - ноль совпадений или одно совпадение,
- `{n}?` - ровно `n` совпадений,
- `{n,}?` - по крайней мере `n` совпадений,
- `{n,m}?` - совпадений по крайней мере `n`, но не более, чем m.

Оратите внимание, что смыслквантификатора от этого не меняется; мен
Если в процессе сопоставления шаблона и текста прототип определ
возвратами увеличит "жадность" такого квантификатора точно так же, ка
если выбор неоднозначен, то результат поиска будет другим:

```
$text = "That is some text, isn't it?";
$text =~ s/.*?is/That's/;
```

```
print $texts;
That's some text, isn't it?
```

Как удалить ведущие и завершающие пробелы

Чтобы отсечь от строки начальные "пробельные символы", можно использ

```
$text = " Now is the time.";
$text =~ s/^\s+//;
print $texts;
Now is the time.
```

Чтобы отсечь "хвостовые" пробелы, годится команда:

```
$text = "Now is the time. ";
$text =~ s/\s+$//;
print $texts;
Now is the time.
```

Чтобы отсечь и начальные, и хвостовые пробелы лучше вызвать последов
шаблон, делающий отсечение ненужных пробелов за один раз. Поскол
текста достаточно сложна, на эту простую операцию может уйти гораздо б

Например в тексте нужно найти текст, находящийся между открывающим

```
$text="<a>blah-blah</a>";
if($text=~m!<([a|b])>(.*?)/\1!ig){
print "$2\n";
}
```

найдет все слова, стоящие между тегами <a></a> и .

В регулярных выражениях пристутствует своя семантика: быстрота, торо
совпадает во многих случаях, то в результате быдет выведен наиболь
Быстрота: поиск старается найти как можно быстрее. `"Text"=~/m*/`, по см
возвращено значение `0`. Т.е. формально `0` и более символов.

```
$test="aaooee ooaao";
$test=~s/o*/e/;
print $test;
eaaooee ooaao
```

потому что 1 элемент сторки - `0` и более символов.

Если добавить квантификатор `g`, то результат будет таким:

```
eaeaeeeeee eeaeaee
```

т.к строка содержит 13 мест, где может встречатся `o`, в том числе и пустых.

Модификаторы:

- `/i` игнорировать регистр
- `/x` игнорировать пропуски в шаблоне и разрешить комментарии.
- `/g` модификатор разрешающий выполнение поиска/замены везде, где
- `/gc` не сбрасывается позиция при неудачном поиске.
- `/s` разрешается совпадение `.` с `\n`, игнорируется `$*`.
- `/m` разрешить совпадение `^` и `$` для начала и конца строки во внутренн
- `/o` однократная компиляция
- `/e` правая часть `s///` представляет собой выполняемый код
- `/ee` правая часть `s///` выполняется, после чего возвращаемое значени

при вызове `use locale` учитываются локальные настройки. Модификатор
`= m/(\d+)/g;` но это сработает для ненакладывающихся совпаден
воспользоваться оператором `?=...` Если ширина `= 0`, то механизм поис
данные остаются внутри скобок. Если есть модификатор `/g`, то текущая
перемещение на один символ вперед. `$numbers="123456789"; @one=$num
(\d\d\d))/g; print "@one \n"; print "@two \n";`

Модификаторы `m` и `s` нужны для поиска последовательностей символов,
совпадает с `\n` и игнорируется `$*`. m делает совпадающими `^` и `$` до и
программный код: `perl -i -n -p -e 's/(.)/lc($1)/g' *.html` приводит
директории к нижнему регистру.

Встроенные переменные в regex.

`$1, $2, $3, $4, ..., $n ...` содержат ссылки на найденный текст, толь
скобках:

```
s%<f(.*?)><(.*?)"><(.*?)">%$1 $2 $3%g;
```

внутри regex можно использовать переменные типа `\1, \2, \3, \4, ...`

```
s/a href=(["'])(.*?)\1>/$2/g
```

найдет все урл, заключенные в двойные, одинарные и вообще без кавычек,

для `/(a.*b)|(mumu)/` в переменной `$+` содержится `$1` или `$2`.

`$&` содержит полный текст совпадения при последнем поиске.

`$'` и `$`` содержатся строки до и после совпадения

Если нужно скопировать и сделать подстановку, то нужно действовать при

```
($at = $bt) =~ s!m(.*?)o!! #для строк
for(@mass1 = @mass2){s/umka/maugli/} #для массивов


$u = ($m=~s/a/b/g); #поменять $m и занести в $u число замен.
```

Если нужно выцепить только алфавитные символы, с учетом настр
`/^[^\W\d_]+$/` в нем учитываются все не алфавитные символы, не цифр
встанька"), симвлол отрицания в группе `[]` - `^`, т.е. найти все, что не `[\`
`!~m/^(\W|\d|_)*/`.

Для упрощения понимания сложных регулярных выражений можно восп
правда можно только по виду регулярного выражения определить зачем он

```
$mmm{$1} = $2 while ($nnn =~ /^([^:]+):\s+(.*)$/m);
```

читаем регулярное выражение:

```
нужно найти в файле все что до двоеточия не двоеточие и все
повторения после первого : .*?: .*?: .*?:, потому что была найден
есть двоеточие до первого двоеточия)
```

Что это может быть, вполне вероятно, что оно нужно для составления
письма и его названия из mbox в хеш. По крайней мере это регулярное выр

## Рабочие программы, использующие регуляр

В принципе регулярные выражениея это вовсе не вещь в себе, хотя иног
полностью реализуемая при помощи regex. Ниже приведены прогр

реглуярных выражений:

## Выделение чисел в математической записи

Пример использования логических условий для нахождения любых математической записи:

```
#!/usr/bin/perl
$_=qq~
1234
34 -4567
3456
-0.35e-0,2
56grf45
-.034 E20
-.034 e2,01 -,045 e-,23
-,034 e201 3e-.20
-,045 e-,23 e-0.88
4 E-0.20
22 E-21
-0.2 w 4 3
345
2 ^-,3
~;
print "$1\n" while m%(([+-]?(?=\d|[\.,]\d)\d*([\.,]\d*)?((\se|e|\s
([-+]?\d*[,\.]?)\d+)?)|([+-]?e[+-]?\d*[,.]?\d+))%gxi;
```

программа исправно выводит все числа. Разберем регулярное выражение

```
 m%(([+-]?(?=\d|[\.,]\d)\d*([\.,]\d*)?((\se|e|\s?\^)
 ([-+]?\d*[,\.]?)\d+)?)|([+-]?e[+-]?\d*[,.]?\d+))%gxi;
```

в переменной `$1` содержится то, что регулярное выражение находит в р `([+-]?e[+-]?\d*[,.]?\d+))%gmi` нужно для того, чтобы находить чис обозначают десятку в какой-то степени, например `e-0,20 = 10`$^{-0,20}$ или `E` выражение "что-то" для чисел вида не `e20` или `E21`:

```
([+-]?(?=\d|[\.,]\d)\d*([\.,]\d*)?((\se|e|\s?\^)([-+]?\d*[,\.]?)\d
```

`[+-]?` - есть ли в перед числом знак `+` или `-`. `?` - если вообще есть что- `[...]`. Выкинем проверку знака, регексп сократится до

```
(?=\d|[\.,]\d)\d*([\.,]\d*)?((\se|e|\s?\^)([-+]?\d*[,\.]?)\d+)?
```

рассмотрим regex `(?=\d|[\.,]\d)\d*` логический оператор `(?=B)` требуе
случае `B` представляет из себя regex `\d|[\.,]\d` Regex `\d|[\.,]\d` значит
что-то либо просто число, либо число, перед которым стоит либо запятая,
`.2` или просто числа `2`(`2` выбрано для примера, может быть и 3). Далее ско
`,2` точно пройдет(например `,2 e-,23` где перед запятой забыли поставить
это предусмотреть. Вообще когда пишешь программу, надо предпола
склеротический чайник, правда не всегда возможно предугадать что учуд
число вида `,223` не пройдет. Да и regex `(?=\d|[\.,]\d)` говорит о том, ч
запятой. Для остальных цифр и нужен квантификатор `\d*`, который знач
ноль, т.е. оно работает и для числе вида `.2` или `,2` Далее идет регулярное
том, есть ли вообще точка и запятая(здесь всю полную строчку в принцип
том числе и его отсутствие, ведь квантификатор `*` значит любой символ
было выше от этого большого регулярного выражения остается строчка:

```
((\se|e|\s?\^)([-+]?\d*[,\.]?)\d+)?
```

Эта строчка отвечает за поиск в строке `$_` математических обозначений
`0,20` например $a^{-0,20}$) и т.д. но только для подстрок вида `-,034 e201`. Зам
если степенное обозначение вообще существует. `(\se|e|\s?\^)` есть ли
числа в "компьютерной" записи вида `2 ^-,3 =` $2^{-0,3}$, т.е. этим регекспом м
ставить пробел при указании степени и разрешили писать значек `^` с пр
выражение `([-+]?\d*[,\.]?)`, которое говорит о том, что степень может
поставь нолик, а на самом деле хотел написать $a^{-0,23}$). Дальше ид
квантификатор то `*`). Потом идет либо точка либо запятая(причем
использование запятой/точки, после `e`, если степень дробная или вообі
иными словами не имеет смысла написать `-2,34e-,23`, хотя юзер на сам
Наконец мы добрались до конца: идет `\d+`, но тут уж, пользователь, бу
квантификатор `+`, а не `*` после `\d`. Т.е. наложили своего рода ограничения з
а можно написать и `2e,-` что суть бессмыленно. И еще, `m%(что-то)%igm` ст
быть и заглавным и квантификатор `x`, который разрешает разносить регуля

Прошу прошения что не ставил иногда знаки препинания, которые есть т
что-то лишнее написно и не подсечено как спецсимвол при помощи бэксл

Итак, регулярным выражением

```
m%(([+-]?(?=\d|[\.,]\d)\d*([\.,]\d*)?((\se|e|\s?\^)
```

```
 ([-+]?\d*[,\.]?)\d+)?)|([+-]?e[+-]?\d*[,.]?\d+))%gxi;
```

были предусмотрены числа степенного порядка, просто числа, числа со з
0,3 или 0.3), ошибки пользователя при вводе чисел( типа `-.034 e2,01` хо
`-.034 e2.01` хотя по смыслу перед точками и запятыми нужно ставить ну
"компьютерном" представлении.

Конечно, данное регулярное выражение не претендует на абсолютную
подстроках вида `-,045 e -,23 e-0.88` считая `-,045` отдельным числом,
идее должно было бы быть два числа `-,045 e -,23` и `e-0.88`, в таком сл
если хочется, чтобы степенные числа понимались корректно(для этой пр
степенью `e`.

## Облегчение поиска работы

Допустим Вы оказались без работы, развалилась ваша фирма или еще ка
новую. Для упрощения этой задачи естьь следующий скрипт, которы
программирование, зарплата от 200$ и т.д.) с www.job.ru все заявки за п
нужно слать резюме, что значительно убыстряет поиск работы(имея ба
резюме, используя нехитрый список рассылки):

```
#!/usr/bin/perl -wT
$url0="http://www.job.ru/cgi/list1.cgi?GR_NUM=";
$url1="%31&TOPICID=9&EDUC=2&TP=&Gr=&SEX=&AGEMIN=23&AGEMAX=&MONEY=2
$url2="&LDAY=99&ADDR=%ED%CF%D3%CB%D7%C1&KWORD=&KW_TP=AND";
use LWP::Simple;
foreach($i=1; $i<=57; $i++){#57 число листаемых страниц
$plus.="%31%2B";
$test=$url0.$plus.$url1.$url2,"\n";
@mass=grep{s/(.*) ([\w+\-\.]+\@[\w\-\.]+\.\w{2,3})(.*)/$2/ig} spli
$test.=join "\n", @mass;
$test.=\n";
}
@un=grep{!$test{$_}++} split /\n/, $test;
print join "\n", @un;
print "\nВы можете отправлять по вашей специальности $#un резюме\n
```

Что делает эта программа, она составляет GET запрос из параметров, кот
результатам запроса на www.job.ru. Программа при помощи `Simple.pm` отп
странички с поиском. Критерий ваших профессиональных навыков сос
разослать почту(для этого можно написать список рассылки) по адреса
регулярное выражение для вытаскивания почтового адреса из текуще

`\.]+\.\w{2,3})(.*)/$2/ig`.

`[\w+\-\.]\@` - найти все что содержит буквы, тире и точки до символа может быть вида a`a.ss-ss@chto-to.ru`. Тоже самое после символа `@` - `[\w` буква от 2 до 3 символов `\w{2,3}`, т.е. окончание, самый верхний доме выражение состоит из трех классов скобок `(.*)` - переменная `$1`, `([\w+\-` и все остальное в `(.*)` - `$3`. Пробел перед `$2` стоит потому, что так устрое по базе предложений о работе www.job.ru. Нам нужно содержимое `$2`, Пишем его во вторую часть `s/наш regex/$2/ig`. Квантификатор `i` нуж `Vasya@pupkin.ru` и `vasya@pupkin.ru`, квантиикатор g задействова на то адреса, по которым нужно высылать резюме. На 23 августа 2001 года на mail адресов(пролистав за 3-4 минуты 57 страниц), где вас ждут, как потен

Остается написать скрипт почтовой рассылки по e-mails, выданным данны

Примером выше был получен спсиок email адресов. Теперь необходимо домены, на которых заведены такие пользщователи(примитивная - но прое

```
#!/usr/bin/perl
use Socket; #загрузить inet_addr
s{ #
 ( #Сохранить имя хоста в $1
 (?: #Группирующие скобки
        (?! [-_] ) #ни подчеркивание, ни дефис
        [\w-] + #кусок имени хоста
        \. #и точка домена
        )+ #повторить несколько раз
        [A-Za-z] #следующий символ - буква
        [\w-]+ #домен верхнего уровня
 ) #конец записи $1
}{ #Заменить следующим:
 "$1" . #исходн часть + пробел
 (($addr = gethostbyname($1)) #Если имеется адрес
        ? "[" . inet_ntoa($addr). "]"#отформатировать
        : "[???]" #иначе пометить как сомнительный
        )
}gex
```

Переписываем исходную программу с учетом вышеприведенного кода

```
#!/usr/bin/perl -wT
$url0="http://www.job.ru/cgi/list1.cgi?GR_NUM=";
$url1="%31&TOPICID=9&EDUC=2&TP=&Gr=&SEX=&AGEMIN=23&AGEMAX=&MONEY=2
$url2="&LDAY=99&ADDR=%ED%CF%D3%CB%D7%C1&KWORD=&KW_TP=AND";
use Socket;
```

```
use LWP::Simple;
foreach($i=1; $i<=57; $i++){
 $plus.="%31%2B";
 $test=$url0.$plus.$url1.$url2,"\n";
 @mass=grep{s/(.*) ([\w+\-\.]+\@[\w+\-\.]+\.\w{2,3})(.*)/$2/ig} sp
 $test1.=join "\n", @mass;
 $test1.="\n";
}
@res=split /\n/, $test1;
@un=grep{!$test{$_}++} @res;
foreach $file(@un){
 $file=~s/(.*)\@(.*)/www\.$2/;
=pod
 $file=~s{((?:(?![-_])[\w-]+\.)+[A-Za-z][\w-]+)}
 {"$1".(($addr=gethostbyname($1))?"[".inet_ntoa($addr)."]":"[???]"
 print $file,"\n" if($file !~/\?\?\?/);
=cut
$file=~s{
 (
 (?:
 (?![-_])
 [\w-]+
 \.
 )+
 [A-Za-z]
 [\w_]+
 )
}{
 "$1".
 (($addr = gethostbyname($1))
 ? "[".inet_ntoa($addr)."]"
 : "[???]"
 )
}gex;
print $file,"\n" if($file !~/\?\?\?/);
}
```

Между строчками можно комментировать целые куски кода.

```
=pod
 $file=~s{((?:(?![-_])[\w-]+\.)+[A-Za-z][\w-]+)}
 {"$1".(($site=gethostbyname($1))?"[".inet_ntoa($site)."]":"[???]"
 print $file,"\n" if($file !~/\?\?\?/);
=cut
```

Эта программа успешно удалила некторые из адресов, которые Socket.p
какую-никакую, а проверку существования e-mail адресс окольными п
Автору сего текста все-таки больше нравится вариант, заключенный в ком

Да и если научится читать сложные регулярные выражения, то можно
который занимается тем, что выделяет адреса в точности с соответ
выражение несколько страгниц). Но впрочем ниже будет подглава, пс
первый взгляд, регекспов налету, со множеством примеров, выше же мы
регулярного выражения только по его виду.

Ключи, которые использовались в вышеприведенном регулярном выражен

`g` - глобальная замена
`e` - выполнение
`x` - улучшенное форматирование.
Если написать это регулярное выражение в одну строчку, то оно врядли та

```
s{((?:(?![-_])[\w-]+\.)+[A-Za-z][\w-])}#здесь силовой перевод каре
{"$1".(($addr=gethostbyname($1))?"[".inet_ntoa($addr)."]":"[???]")
```

Разберем один интересный момент в данном регекспе:

```
s/regex/условие?да:иначе/
```

Тут проявляется пожалуй одна из действительно сильнейших особенносте
выражении избежать многострочных условий с циклом. В приведенном
`$addr=gethostbyname($1)` - да, то ставить ip-адрес(`inet_ntoa($addr)`),
линии и пр) то метить этот урл как подозрительный `[???]`. В принцип
нужно, т.к. подозрительные отметаются условием `print $file,"\n" if(`
программы 10-15 минут.

## Очень простое решение для зеркала новостной ленты

Допустим нужно сделать зеркало какой-либо зарубежной новостной ленты
сервера, чтобы не ждать по несколько минут отображения содержимого
Приведенный скрипт запускается при помощи crontab каждые 5 часов:

```
#!/usr/bin/perl -w
$/="\001";
print "content-type: text/html\n\n";
$dir="/var/www/docs/html/news/images";
$imgurl="http://www.qwerty.ru/news/images";
use LWP::Simple;
use LWP::UserAgent;
```

```
$page=get "http://www.astronomynow.com";
$page=~s/face="(.*?)"//igs;
&getimg($page);
$page=~s!/images/grafix/listdot.gif!../../listdot.gif!igs;
$page=~s!/images/grafix/spacer.gif!../../spacer.gif!igs;
$page=~s!images/grafix/spacer.gif!../../spacer.gif!igs;
if($page=~m!<TABLE WIDTH="400" BORDER="0" CELLPADDING="0" CELLSPAC
 $file=$1;
 &getlink($page);
 foreach $names(@res){
 $names=~s|.*/||ig;
 $file=~s|src="http://(.*?)$names"|src=$imgurl/$names|igs;
 }
 $html=qq~
 <TABLE BORDER="0" CELLPADDING="0"
 CELLSPACING="0">
 $file
 </TD></TR></TABLE>~;
}
open F, ">$dir/news.txt";
print F $html or die "\n\n\n ERROR: $!\n\n\n";
close F;
sub getimg{
 &getlink($_[0]);
 foreach $img(@res){
 my $res = LWP::UserAgent->new->request(new HTTP::Request GET => $
 if ($res->is_success) {
 $img=~s|.*/||;
 open (ABC, ">$dir/$img") or die "\n\n\nERROR: $!\n\n\n";
 binmode(ABC);
 print ABC $res->content; close ABC or die "\n\n\nERROR: $!\n\n\n"
 } else {
 print $res->status_line;
 }
 }
 return @res;
}
sub getlink{
 local $_=$_[0];
 push(@res, "http://$2")
 while m{SRC\s*=\s*(["'])http://(.*?)\1\s*(.*?)WIDTH="100" HEIGHT=
 return @res;
}
```

## Вывод результатов поиска

Предположим есть необходимость подсветить результаты поиска в файла

aport. Данное регулярное выражение позволяет влоб реализовать эту кр
имеет очень большой минус, при обработке текста машина начинает неим
этот регексп из общих соображений:

```
$sn=4;
{
local $_=$description1;
print "...$1<font color=red>$3</font>$4..."
while(m/(([\s,\.\n^]*\w*){$sn})(\s*$query\s*)(([\s,\.\n^]*\w+){$sn
}
$_="";
```

Исходная задача состоит в следующем: вывести по 4 слова спереди и сз
если слово находится первым, то будет видно 4 слова позади него. В точ
слова.

Соответственно из вида регекспа понятно, что разделителями слов могут
символ перевода каретки `^`. Комбинация `(\d\d\d){$sn}` значит что нужно и