Операторы

В Perl ассоциативность и приоритетность операторов аналогична языку C. Ниже перечислены все операторы в порядке уменьшения приоритета, в начале строки указана ассоциативность.

ассоц.	операторы
левая	термы и левосторонние списковые операторы
левая	->
-	++ --
правая	**
правая	! ~ \ унарные + и -
левая	=~ !~
левая	* / % x
левая	+ - .
левая	<< >>
-	именованные унарные операторы
-	< > <= >= lt gt le ge

-	== != <=> eq ne cmp
левая	&
левая	| ^
левая	&&
левая	||
-	..
правая	?:
правая	= += -= *= и т.д.
левая	, =>
-	правосторонние списковые операторы
левая	not
левая	and
левая	or xor

Термы и левосторонние списковые операторы.

Любой терм имеет самый высокий приоритет. К терму относятся переменные, кавычки и их операторы, арифметические и логические выражения в скобках, любые функции с параметрами в скобках. Фактически таких функций нет, так как это просто унарные и списковые операторы. Просто они ведут себя подобно функциям с параметрами в скобках. Подробно смотри главу Функции.

Если после любого спискового оператора (print(), и т.д.) или унарного оператора (chdir(), и т.д.) следует левая круглая скобка, то операторы внутри скобок имеют наивысший приоритет. Так же как и обычные функции.

Если скобки отсутствуют, то приоритет списковых операторов или наивысший или наименьший в отношении операторов справа или слева от него.

Например:

```
@i = ('a ','b ', print 'c ', 'd ');

print "\n",@i,"\n";
```

Результат:

```
c d

a b 1
```

Здесь мы имеем списковый оператор print. Для запятых слева от него он имеет наименьший приоритет, но повышает приоритет правой запятой. Поэтому правая запятая воспринимается как параметр для print и печатается 'c d', а левая просто записывает код завершения операции в массив @i и последний print показывает это.

Оператор '->'

Так же как в C или C++ это инфиксный оператор переадресации. Если справа от него стоит [...] или {...} выражение, то правая часть может быть непосредственной или символической ссылкой на массив или хеш. В противном случае правая сторона это метод или простой скаляр, содержащий имя метода, а правая - или объект или имя класса. Подробно смотри главу Классы.

Операторы ++ (инкремент) и -- (декремент).

Эти операторы работают так же как и в C. Если оператор стоит перед переменной, то значение переменной изменяется на 1 и полученное значение используется. Если после переменной - то ее величина изменяется после применения.

Употребление инкремента к строковым переменным в Perl имеет одну особенность. Каждый символ остается в своем классе (большие, малые, цифры) и учитывается перенос предыдущего символа. Таким образом строковые переменные с цифрами работают как числовые переменные.

Пример:

```
print ++($i = "09");     # Результат "10"


print ++($i = "a9");     # "b0"


print ++($i = "az");     # "ba"

print ++($i = "aZ");     # "bA"
```

Оператор ** (возведение в степень)

Пример:

```
print 4**2     # Результат 16
```

```
print -4**2     # Результат -16 т.е. -(4**2)
```

Унарные операторы.

'!'	- логическое отрицание
'-'	- арифметический минус
'~'	- побитная инверсия (дополнение до 1)
'+'	- арифметический плюс
'\'	- получение ссылки на переменную (как & в C)

Операторы "привязки" =~ и != .

Эти "необычные", я бы даже сказал, оригинальные операторы имеют очень широкое применение в Perl. Можно даже назвать их "оригинальным" решением. Оператор =~ логически связывает левую часть выражения с патерном (pattern - образец, шаблон, модель) в правой. По умолчанию поиск или изменение по патерну выполняется в переменной $_ Операторы привязки позволяют делать это с любой переменной, указанной в левой части. Логическим результатом будет успех операции. Если в правой части вместо патерна присутствует выражение, то результат этого выражения воспринимается как патерн. Однако это не очень эффективно, т.к. патерн будет компилироваться во время исполнения программы, что заметно снизит быстродействие. Оператор != аналогичен =~, только результат совпадения инвертируется (логическое "нет"). Подробное применение этих операторов приводится в главе Патерны.

Мультипликативные операторы.

'*'	- арифметическое умножение
'/'	- арифметическое деление
'%'	- арифметический модуль
'x'	- оператор повторения

В скалярном контексте возвращает строку левой части, повторенную величиной, указанной в правой части. В списковом контексте, если в левой части список, то в круглых скобках - повторенный список.

Пример:

```
print '*' x 5;     # Результат '*****'


print (1,2) x 3; # Результат 121212
```

Аддитивные операторы.

'+'	- арифметический плюс
'-'	- арифметический минус
'.'	- конкатенация (объединение) строк

Операторы сдвига.

'<<'	- Сдвигает побитно влево значение выражения в левой части
	на количество бит указанное в правой.
'>>'	- Сдвигает побитно вправо значение выражения в левой части
	на количество бит указанное в правой.

Именованные унарные операторы.

Фактически это функции с одним аргументом. Круглые скобки можно опускать.

Операторы отношений.

'<'	- арифметическое меньше
'>'	- арифметическое больше
'<='	- арифметическое меньше или равно
'>='	- арифметическое больше или равно
'lt'	- строковое меньше
'gt'	- строковое больше
'le'	- строковое меньше или равно
'ge'	- строковое больше или равно

Операторы равенства.

'=='	результат true если левая часть равна правой (равно)
'!='	- не равно
'<=>'	- -1 если левая часть меньше правой, 0 если равна,
	1 если больше.
'eq'	- строковое равно
'ne'	- строковое не равно
'cmp'	- как и '<=>' применительно к строкам

Операторы работы с битами.

'&'	- побитное AND
'|'	- побитное OR
'^'	- побитное XOR

Логические операторы && (AND) и || (OR).

'&&'	- если левое выражение возвращает false, правое не выполняется.
'||'	- если левое выражение возвращает true, правое не выполняется.

Отличие от подобных операторов в C заключается в том, что в C возвращаемое значение либо 0, либо 1, тогда как в Perl возвращается результат выражения.

Оператор диапазона '..'

Результат работы данного оператора зависит от контекста. В списковом контексте результат есть список с элементами, первый элемент которого это левое выражение и последнее - правое. Значение каждого элемента внутри списка увеличивается на 1. Данный оператор удобен для небольших циклов, т.к. память отводится для всего списка целиком. Поэтому будьте внимательны и не задавайте слишком большой диапазон.

Пример:

```
for $i (1..4)
   { print "$i ";
   }
```

Результат: 1 2 3 4

В скалярном контексте результат - логическое значение. Каждая '..' операция устанавливает свое собственное состояние. Это false до тех пор пока левый операнд false. Как только левый операнд стал true результат - true до тех пока правый true, после чего результат опять - false. Если вы не хотите проверять правый операнд, то используйте оператор '...'.

Правый операнд не вычисляется пока результат false и левый операнд не вычисляется пока результат true. Приоритетность оператора '..' немного ниже чем '&&' и '||'. Возвращаемое значение если flase - нулевая строка, если true - порядковый номер начиная с 1. Порядковый номер обнуляется для каждого нового диапазона. Последний порядковый номер добавляется строкой "E0" которая не изменяет его значение, но позволяет фиксировать последнее значение.

Пример:

```
@алфавит = ('a'..'z'); # Массив малых букв латинского алфавита

@цифры = (0..9);     # Массив цифр
```

Условный оператор '?:'

Этот оператор работает так же как и в C. Если выражение перед '?' истинно, то выполняется аргумент перед ':' - иначе после ':'.

Пример:

```
$i = 1;

$i > 1 ? print "больше" : print "меньше";
```

Результат: меньше

Операторы присваивания.

'=' - обычный оператор "присвоить" правое значение переменной слева
Вся эта группа операторов подобна операторам C, т.е.

```
$i += 2;
```

эквивалентно

```
$i = $i + 2;
```

Остальные операторы этой группы работают аналогично. Допустимы следующие операторы:

```
**=
+=, -=, .=
*=, /=, %=, x=
&=, |=, ^=
<<=, >>=
&&=, ||=
```

Приоритет всей этой группы операторов равен приоритету '='.

Оператор ',' (запятая)

В скалярном контексте выполняется левый аргумент, результат игнорируется, затем правый и его результат есть результат действия оператора. В списковом контексте это разделитель элементов списка, который включает указанные элементы в список.

Операторы not, and, or, xor

Оператор логическое not (отрицание).
Унарный not возвращает противоположное значение, полученное выражением справа. Он эквивалентен '!', но имеет очень низкий приоритет.

Оператор логическое and (И).
Выполняет логическую конъюнкцию двух выражений. Эквивалентен '&&', но имеет очень низкий приоритет и "краткость" действия, т. е. если левое выражение равно false - левое не выполняется.

Логическое or (ИЛИ).
Выполняет логическую дизъюнкцию двух выражений. Эквивалентен '||', но имеет очень низкий приоритет и "краткость" действия, т. е. если левое выражение равно true - левое не выполняется.

Логическое xor (исключающее ИЛИ).
Выполняет логическое исключающие или. Всегда выполняются оба правое и левое выражение.

В Perl отсутствуют операторы языка C такие как:

унарное &	- получить адрес. Для этого применяется '\'.
унарный *	- переадресация.
(TYPE)	- совмещение типов.

Операторы ограничители строк.

Обычно ограничителями строк мы считаем литералы, но в Perl это операторы выполняющие разного рода интерполяцию и поиск по шаблону. Вы можете сами задавать удобные для вас ограничители. В следующей таблице приведен полный перечень вариантов. Фигурные скобки '{}' обозначают любой символ, используемый для ограничителя. В случае использования скобок (круглых '()', квадратных '[]', фигурных '{}', угловых '<>') в начале ставится открывающаяся скобка, а в конце закрывающая.

По умолчанию	Полное	Функция	Интерполяция
''	q{}	Literal	нет
""	qq{}	Литерал	да
``	qx{}	Команда	да
	qw{}	Список слов	нет
//	m{}	Шаблон	да
	s{}{}	Подстановка	да
	tr{}{}	Трансляция	нет

В строках допускающих интерполяцию имена переменных, начинающиеся с символов '$' или '@' - интерполируются, т.е. в строку вставляется значение строки или массива. Данные последовательности символов имеют специальное значение:

\t	символ табуляции
\n	символ новой строки

\r	возврат
\f	перевод формата
\v	вертикальная табуляция
\b	backspace (забой)
\a	звонок
\e	escape
\034	восьмеричный символ
\x1a	шестнадцатеричный символ
\c[	символ управления
\l	нижний регистр следующего символа
\u	верхний регистр следующего символа
\L	нижний регистр для всех символов до \E
\U	верхний регистр для всех символов до \E
\E	ограничитель смены регистра
\Q	отмена действия метасимволов до \E

Шаблоны интерполируются как регулярные выражения. Это выполняется вторым проходом после интерполяции переменных, поэтому в шаблоны можно вставлять переменные. Для отмены интерполяции используйте '\Q'. Если вы применяете вложенные ограничители, то внутренние ограничители работать не будут.

?PATERN?

Действие этого оператора аналогично /шаблон/, но выполняется до первого совпадения. Это удобно для поиска наличия какой-нибудь строки в одном или множестве файлов. Это не очень удачный оператор, поэтому в следующих версиях Perl его возможно не будет.

m/PATERN/gimosx
/PATERN/gimosx

Поиск в строке по патерну (шаблону). В скалярном контексте возвращает логическое значение true (1) или false (''). Если строка не указана с помощью операторов '=~' или '!~', поиск ведется в строке $_ Опции:

g	- Глобальный поиск. Поиск всех вхождений.
i	- Сравнение не зависит от регистра (верхний или нижний)
m	- Строка многострочная.
o	- однопроходная компиляция
s	- однострочная строка
x	- используются расширенные регулярные выражения.

Если '/' - ограничитель, то начальное 'm' можно опустить. С помощью него в качестве ограничителя может быть любой символ кроме пробела.

PATTERN может содержать переменные, которые будут интерполироваться (перекомпилироваться) каждый раз в момент вычисления. Переменные $) и $| не интерполируются. Если вы хотите, что бы такой шаблон интерполировался один раз - добавьте /o. Это необходимо делать в циклах поиска для увеличения быстродействия, однако, если вы измените значение переменной, Perl этого даже не заметит.

Если PATERN - нулевая строка, то используется последнее регулярное выражение.

В скалярном контексте возвращается список элементы которого - результаты выполнения выражений в скобках патерна ($1, $2, $3...). Обратите внимание что первый элемент $1.

Пример:

```
$a = "/usr/local/perl/perl.bin"; # Анализируемая строка
```

Цель: Создать массив @dirs с именами директорий.
Решение: Самый простой способ воспользоваться split('\/') но в качестве примера используем скобки.

```
@dirs =~ m[/(\w*)/(\w*)/(\w*)/(\w*)]
```

Здесь 'm[' - использовать квадратные скобки как ограничители. (\w*)- шаблон алфавитно-цифровой последовательности.
В результате @dirs равен ('usr', 'local', 'perl')

q/строка/
'строка'

Строка литералов. Не интерполируется. Внутри строки разрешается использовать \' или \\ для обозначения символов ' и \ .

Пример:

```
print q#Привет.#;     # Результат Привет.


print 'O\'K';          # O'K
```

qq/строка/
"строка"

Интерполируемая строка.

Пример:

```
$var = 13;

print "\$var = $var";
```

Результат: $var = 13

qx/строка/
`строка`

Сначала строка интерполируется, а потом выполняется как системная команда.

Пример:

```
print `date`;
```

Результат: Thu Nov 14 13:36:49 MSK 1996

qw/строка/

Возвращает список, элементы которого - слова строки, разделенные пробелами.

Пример:

```
print qw/Построимся и спасемся!/; # ('Построимся','и','спасемся!')
```

Результат:

```
Построимсяиспасемся!
```

Часто применяется как:

```
use POSIX qw( setlocale localeconv )

@EXPORT = qw( proc1 var );
```

s/шаблон/подстрока/egimosx

Поиск по шаблону и в случае успеха замена подстрокой. Возвращает количество произведенных подстановок, иначе false (0). Если строка в которой ведется поиск не указана (операторы =~ или !=), то используется переменная $_ . Если в качестве разделителя '/' использовать

одинарную кавычку ('), то интерполяции не будет, иначе можно применять переменные в шаблоне или подстроке.

Опции:

e	- Рассматривать правую часть как выражение.
g	- Глобальный поиск.
i	- Без различия регистра букв
m	- многострочная переменная
o	- компилировать шаблон один раз
s	- однострочная переменная
x	- расширенное регулярное выражение

Разделитель '/' можно заменить на любой алфавитно-цифровой символ кроме пробела.

Пример:

```
$var = "12345"; # исходная строка

$var =~ s/1/0/; # Заменить '1' на '0'. Результат 02345

$var =~ s(5)(.); # Заменить '5' на '.' Результат 0234.
```

Здесь в качестве разделителя применены скобки, поэтому подстрока взята в две скобки.

```
$var =~ s/\d*/каламбур/; Заменить все цифры. Результат 'каламбур.'


$var =~ s/a/o/g; # Заменить все 'a' на 'o'. Результат 'коломбур.'


$var = "12 34"; # Новое значение


$var =~ s/(\d\d) (\d\d)/$2 $1/; # Поменять местами числа. Результат '34 12'.
```

tr/таблица1/таблица2/cds
y/таблица1/таблица2/cds

Замена всех символов из "таблица1" на соответствующий символ из "таблица2". Результат - количество замен или стираний. Без оператора =~ или != операция выполняется со строкой

$_. Для совместимости с программой sed вместо tr можно писать 'y'.

Опции:

c	- дополнение "таблица1"
d	- стереть найденные, но не замененные символы.
s	- "сжать" повторяющиеся замененные символы.

Если указана опция /d таблица2 всегда интерпретируется как положено. Другими словами, если таблица2 короче, чем таблица1, то символ из таблицы1 интерпретируется всегда. Если таблица2 - null, то все символы строки остаются неизменными. Это удобно для подсчета количества символов в строке определенного класса или для сжатия повторяющихся символов, например, пробелов.

Пример:

```
$s = "hello";     # Исходная строка



$s =~ tr/a-z/A-Z/;     # Заменить малые буквы на большие. Результат
                # 'HELLO'



$s = 'Hel....lo';
$s =~ tr/a-zA-z/_/c;     # Заменить все не буквы на '_'
                # Результат 'Hel____lo'
$s =~ tr/_/ /s;          # Заменить '_' на ' ' и сжать.
                # Результат 'Hel lo'
$s =~ tr/a-zA-Z /a-zA-Z/d; # Удалить все не буквы. Результат 'Hello'
```

Если один и тот же символ несколько раз указан в таблице1, то применяется только первая замена.

Операторы ввода-вывода.

В Perl существует несколько операторов ввода-вывода. Первый это скобки из символа '`' - акцента. Строка в этих скобках воспринимается как системная команда и результат ее действия возвращается как "псевдо" литерал. В скалярном контексте это строка содержащая

весь результат, а в списковом - список, элементы которого - строки результата. Статус выполненной команды хранится в переменной $? .

Следующая команда ввода вывода выглядит как '<файл>'. Вычисление <файл> приводит к чтению строки из файла. Обратите внимание, что 'файл' здесь не имя файла, а указатель файла, который создается функцией open(). В скалярном контексте читается одна строка вместе с символом '\n' - перевода строки, а в списковом весь файл читается в список, элементы которого суть строки файла. В случае обнаружения конца файла результат оператора не определен и воспринимается как false. Если не указана переменная результата, то по умолчанию это $_. Указатель файла по умолчанию STDIN - стандартный ввод.

Пример:

```
while(<>) { print; }; # Прочитать и вывести весь файл STDIN
```

У оператора '<>' есть одна отличительная особенность. Если в командной строке нет никаких аргументов, то читается стандартный ввод, если есть аргументы, то они считаются именами файлов, которые последовательно читаются.

Если в угловых скобках записана переменная, то содержимое этой переменной считается именем указателя файла или ссылкой на указатель файла. Если такого указателя не существует, то содержимое переменной воспринимается как шаблон имен файлов и результат - имена файлов на диске, подходящих по шаблону.

Пример:

```
while(<*.pl>) { print;}; # То же что и ls *.pl



@files = <*>;     # Массив @files содержит имена файлов в директории
```

но лучше сделать: @files = glob("*"); т.к. внутри скобок можно использовать переменные.

Слияние констант.

Как и C Perl выполняет возможные вычисления в период компиляции. Так подстановка символов после '\' , операция конкатенации строк, арифметические выражения, содержащие только одни константы, все это делается в момент компиляции, что существенно увеличивает скорость выполнения программы.

Целочисленная арифметика.

По умолчанию Perl выполняет арифметику с плавающей запятой, но если вы укажете:

```
use integer;
```

то компилятор будет использовать целочисленную арифметику до конца текущего блока, хотя вложенный блок может это и отменить в своих пределах с помощью:

```
no integer;
```