

*Знакомство с языком*

# Perl 6

Первый российский воркшоп

# Perl Today

Москва, 26 октября 2007

*Знакомство с языком*

# Perl 6

[www.perlrussia.ru](http://www.perlrussia.ru) — [www.perl6.ru](http://www.perl6.ru) — 2007

## **Знакомство с языком Perl 6.**

Бумажное приложение к воркшопу “Perl Today”.

Ориентировано на тех, кто хотя бы поверхностно знаком с Perl 5.

Автор текста — *Андрей Шитов*  
andy@shitov.ru | <http://shitov.ru>

2007

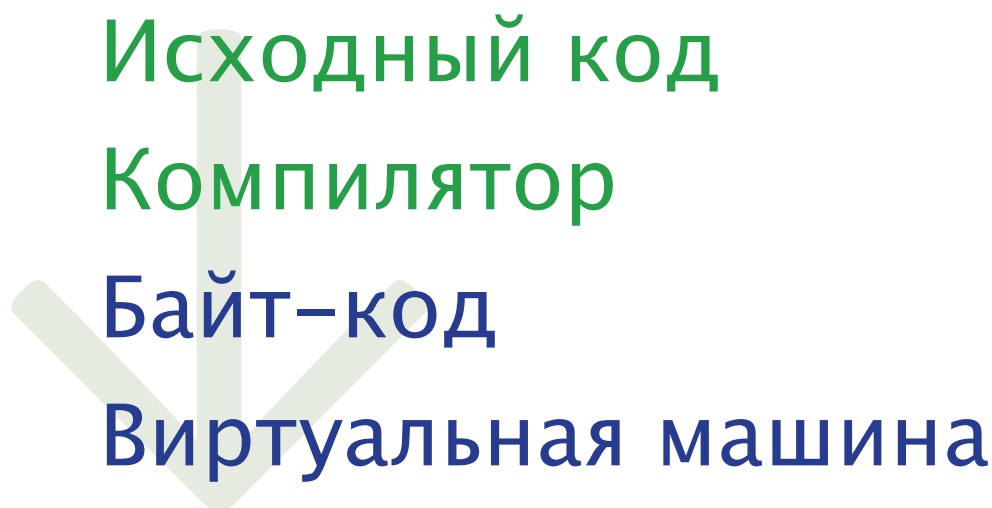
5 != 6  
4 != 5

Perl шестой версии — совсем другой язык. Чтобы начать изучать его, не нужно предварительно учить Perl 5. Про новый язык начали говорить в 2000 году, тогда же стали появляться документы, описывающие дизайн нового языка.

Отправная точка — набор трех серий документов: *Apocalypses*, *Exegeses* и *Synopses* (все они доступны на сайте [perl.org](http://perl.org)). В первой серии, написанной Ларри Уоллом, собраны основные идеи, которые присылали пользователи-программисты, и мнение автора языка. *Apocalypses* на сегодня следует считать историческим документом. Серия *Synopses* фактически является документацией по языку Perl 6.

Переход на новую версию — не причина для паники. Несколько лет назад многие испытывали трудности при переходе от четвертой версии к пятой. Сегодня про эти времена вспоминают совсем нечасто, а молодые программисты даже не знают, что были сложности.

o



Программы на Perl 6 компилируются в байт-код. С самого начала разработки языка была принята идеология, согласно которой любая программа предварительно компилируется в байт-код, а затем исполняется виртуальной машиной. Название *скрипт* формально неприменимо к исходным текстам на Perl 6, хотя по-прежнему допустимо делать однострочные программы, которые будут выполняться налету.

«Родная» виртуальная машина шестой версии перла называется Parrot ([www.parrotcode.org](http://www.parrotcode.org)); она способна выполнять байт-код, скомпилированный из программ на других языках. Дистрибутив содержит экспериментальные версии компиляторов более чем десятка языков.

Сегодня для изучения Perl 6 удобно пользоваться компилятором Pugs ([www.pugscode.org](http://www.pugscode.org)), в котором спецификация реализована наиболее полно, хотя и отсутствует возможность получить байт-код.

# Parrot

```
use v6;  
Perl6::*
```

# PUGS

Сегодня основной инструмент — компилятор Pugs. Первая экспериментальная версия компилятора Perl 6 была включена в состав первой версии виртуальной машины Parrot. В 2004 году развитие этого компилятора было остановлено, но появился новый инструмент — Pugs (Perl 6 User’s Golfing System).

Pugs написан на языке Хаскель, и требует для своей сборки установленного компилятора ghc, поэтому самостоятельная сборка обеих систем оказывается утомительной, однако проходит гладко как под Windows, так и в UNIX-подобных системах.

В разное время появилось еще несколько решений для того, чтобы начать экспериментировать с синтаксисом Perl 6. Например, на CPANе опубликованы модули Perl6::\* и v6, которые эмулируют некоторые особенности Perl 6 в программе, исполняемой обычным компилятором пятой версии.

```
print "5\n";  
say 6;  
6.say;
```

В Perl 6 появился новый оператор печати — встроенная функция `say`.

Функция `print` продолжает работать так же, как и в пятой версии, включая способность интерполировать переменные в строках, заключенные в двойные кавычки. Однако в Perl 6 часто (особенно во время отладки) удобнее пользоваться функцией `say`. Она выводит на печать (на экран) свои аргументы, после чего делает перевод строки.

Более существенное отличие — в Perl 6 возможно обратиться к `say` не только как к встроенной функции, но и как к методу переменной, например: `$x.say` (то же самое теперь возможно и с `print`). Любители C++ могут поставить круглые скобки: `$x.say()`.

Такая двойственная натура многих встроенных функций позволяет изменить порядок слов, когда нужно последовательно вызвать несколько операторов, например, вместо `print (join(':', @values), "\n")` написать `@values.join(':').say`.

```
my $str
    = '俄罗斯新闻网';
say $str.chars;
say $str.bytes;
say $str.graphs;
```

Исходный код Perl 6 записан в юникоде. Компилятор Perl 6 ожидает получить исходный код в юникоде (UTF-8). Поэтому можно свободно использовать любые символы за пределами ASCII. Дополнительных инструкция вроде `use utf8` записывать не нужно.

Вместе с приходом юникода потребовалось «расслоить» функцию вычисления длины строки `length` — теперь их три: `chars`, `bytes` и `graphs`. Все их возможно вызвать как функцию или как метод. `bytes` возвращает число байтов, которые занимает строка, а `chars` — число символов в ней. Те, кто знаком с юникодом и с тем, как в нем формируются композитные символы, может сделать предположение о результате работы метода `graphs`.

Чтобы не было путаницы, от функции с именем `length` решено отказаться полностью. Например, длину массива теперь возвращает метод (или функция) `elems`: `say @values.elems`.

```
say $scalar;  
say @array[1];  
say @array[3..7];  
say %hash{'key'};
```

Сигилы переменных в Perl 6 остаются неизменными. Сигилы (символы `&`, `$`, `@`, `%` и `*`) перед именами переменных в Perl 5 всегда были особенностью, на которой требовалось заострять внимание при изучении языка. Теперь независимо от того, как используется имя переменной и какой структурный тип (скаляр или список) имеет результат, сигил остается неизменным. Можно упрощенно считать, что сигил — одна из букв в названии переменной.

Обращение к элементу массива теперь записывается в виде `@array[$n]` вместо прежнего `$array[$n]`. При выборке среза массива как и ранее используется сигил массива: `@array[1, 2, 3]`.

Доступ к элементам хеша возможен с помощью традиционных фигурных скобок (`%hash{'key'}`) и с помощью нового синтаксиса: `%hash<key>`. Второй вариант удобен, когда ключи записаны строковым литералом. Срез хеша в таком варианте записывается компактнее: `%h<alpha beta>`.

```
if $x < $y {  
    say 'less';  
}  
  
say 1 if $x < $y < $z;
```

Операторы сравнения допустимо объединять в цепочку. Скобки вокруг условия необязательны. Условный оператор «пятой версии» неудобен прежде всего тем, что всегда требует круглых скобок, если условие стоит перед инструкцией, в то время как постфиксный вариант скобок не требует. В Perl 6 скобки допускается опускать в обоих случаях: `say 'OK' if $true; if !$false {say 'true'}`. (Фигурные скобки вокруг выполняемого кода опускать нельзя.)

Кроме того, оператор сравнения допускает объединение условий в цепочку, делая запись похожей на традиционную нотацию, принятую в математике: `if 0 < $x == $y > 10 {...}`. Попарные сравнения выполняются слева направо, поэтому будет ошибочно считать, что в приведенном примере сравниваются два булевых выражения `0 < $x` и `$y > 10`, как было бы в таком случае: `if (0 < $x) == ($y > 10) {...}`.

Аналогично ведет себя функция `unless`.

```
my @array = 1..10;
say @array;
say ~@array;
say +@array;
say int @array;
say ?+@array;
say ~hash @array;
```

В Perl 6 возможно явно управлять контекстом. Программист может самостоятельно указывать, в каком контексте следует рассматривать переменную или выражение, записав функцию смены контекста. Синтаксически преобразователи контекста — это префиксные функции.

Тильда требует преобразования в строковый контекст (массивы, например, преобразуются в разделенную пробелом последовательность значений элементов).

Плюс и эквивалентная ему функция `int` указывают числовой контекст. Массив в числовом контексте — это размер массива.

Знак вопроса — булевый контекст.

Контекст хеша (`hash`) преобразует массив в последовательность пар «ключ — значение».

Возможно «нанизывать» друг на друга несколько преобразований.

```
sub power ($x, $y) {  
    return $x ** $y;  
}  
say power(2, 10);
```

Синтаксис объявления, определения и вызова функций стал богаче.

Возможностей при объявлении функции стало намного больше: теперь возможно явно объявить все аргументы, сделать их именованными, пометить необязательность и присвоить значения по умолчанию.

*Perl традиционно называет обычные функции подпрограммами (subroutines), их поведение в Perl 6 описано в документе Synopses 6 "Subroutines" (сокращенно называют S6). В то же время существует недописанный документ Synopses 29 "Functions", который описывает совсем другое.*

В объявлении функции допустимо сразу перечислить ее формальные аргументы: `sub power ($x, $y)`; указанные переменные доступны по этим именам внутри функции. При вызове можно явно указывать именованные переменные, причем порядок их следования в этом случае произволен: `say power(y => 10, x => 2)`.

```
say power(  
    y => 10,  
    x => 2  
);  
say power(:x<2>, :y<10>);
```

Передача именованных параметров функции допускает несколько вариантов. Во-первых, традиционный с позиционными аргументами — то есть с аргументами, которые должны идти в том же порядке, в каком они перечислены в объявлении: `power(2, 10)`.

Во-вторых, можно передать аргументы по их именам, при этом порядок следования не имеет значения, как показано на предыдущей странице.

В-третьих, доступен альтернативный вариант записи при передаче именованных аргументов: `power(:x<2>, :y<10>)`. Формально такой вид записи — один из вариантов доступа по именам к элементам обычных хешей.

Наконец, возможно упаковать именованные параметры в отдельный хеш и передать его целиком. При этом компилятору необходимо сообщить о том, что в хеше содержатся пары «имя переменной — ее значение»: `my %h; %h<x> = 2; %h<y> = 10; say power(%h)`.

```
sub ret ($arg is rw) is rw {  
    return $arg  
}  
ret($y) = $new_value;
```

И аргументам, и самой функции возможно присвоить свойство `is rw`.

По умолчанию переданные в функцию аргументы изменить нельзя; при попытке это сделать возникает ошибка.

Perl 6 позволяет управлять поведением аргументов с помощью свойств (traits). Для этого служит ключевое слово `is`, за которым следует необходимое имя, например: `$arg is rw`.

Свойство `rw` (read, write) позволяет изменять значение аргумента в теле функции: `sub increment ($arg is rw)`.

Более того, свойство `is rw` возможно применить не только к аргументу, но и к самой функции, сделав ее таким образом *lvalue*, что позволит присвоить значение результату функции, как показано на примере вверху страницы.

Существует около десятка различных свойств.

```
sub inc ($arg, $step = 1);  
inc($x);  
inc($x, 2);  
  
sub dec ($arg, $step?);
```

В Perl 6 можно указывать необязательные параметры и аргументы по умолчанию. При объявлении функции допустимо указывать необязательные аргументы или устанавливать значение по умолчанию. Необязательные элементы можно передавать как позиционно, так и поименно.

Необязательный параметр помечают вопросительным знаком. При вызове этот аргумент может быть опущен, и ему по умолчанию присвоится значение undef.

Кроме того, для необязательных аргументов допустимо указать значение по умолчанию, записав его после знака равенства непосредственно в заголовке функции.

В дополнение к вопросительному знаку синтаксис предусматривает и восклицательный знак, чтобы объявить аргумент обязательным:  
sub power (\$x!, \$y!).

```
sub some (@a, @b) {  
    say join ', ', @a;  
    say join '; ', @b;  
}
```

По умолчанию функция не объединяет переданные аргументы в один список. Это означает, что если объявить функцию с двумя аргументами списочного типа, то оба они доступны в теле функции как два отдельных списка. В Perl 5 для реализации такого поведения необходимо передавать ссылки.

Полноценные аргументы могут пользоваться всеми остальными преимуществами нового синтаксиса, например быть объявленными по умолчанию или обладать атрибутом `is rw`.

При необходимости свернуть все позиционные аргументы в общий массив следует объявить его свертывающим (slurpy): `sub f (*@all)`. Теперь любые неименованные аргументы попадут в этот массив, например: `f(1..19, @b)`.

Аналогично объединяются в общий хеш именованные аргументы: `sub f (%all); f(alpha => 'a', beta => 'b')`.

```
my $square = -> $x {  
    $x * $x;  
}  
  
say $square(2);
```

Анонимные функции определяют непосредственно блоком кода или с помощью стрелки `->`. Анонимные (безымянные) функции существовали и в Perl 5, однако теперь для их объявления необязательно записывать ключевое слово `sub`: `my $func = {say 'Hi!'}`. Для вызова такой функции нужно записать круглые скобки: `$func()`.

В Perl 6 анонимные функции могут иметь именованные аргументы. Чтобы записать список аргументов, нужно воспользоваться другим синтаксисом для создания анонимной функции — с помощью оператора `->` как показано вверху страницы.

При использовании стрелки круглые скобки вокруг списка аргументов становятся необязательными. Однако скобки нельзя опускать при вызове такой функции, иначе компилятор встретит бессмысленную последовательность, состоящую из ссылки на функцию и некоторого значения или переменной.

```
for @list -> $x, $y {  
    say $x + $y;  
}
```

```
loop (my $c = 0;  
      $c != 10; $c++);
```

Цикл `for` принимает два параметра — список и анонимный блок.

Показанный выше вариант записи цикла `for` печатает попарную сумму элементов из списка `@list`. Синтаксис подсказывает, как будет выполняться цикл: из списка выбираются по два значения и их сумма выводится на печать.

Фактически у оператора (функции) `for` два аргумента: один из них — переменная `@list`, второй — безымянная функция с двумя аргументами `$x` и `$y`, печатающая их сумму.

Цикл в стиле C также продолжает существовать, однако под новым именем `loop`.

Вместо блока `do {...} while` теперь следует писать `repeat {...} while`. В теле циклов (в том числе и блока `repeat`) работают операторы `next`, `last` и `redo`. Цикл `foreach` упразднен, вместо него используют `for`.

```
my @sum
    =
    (1, 3, 5, 7)
    >> + <<
    (2, 4, 6, 8)
;
```

Гипероператоры выполняют попарные действия над элементами списков.

В Perl 6 есть группа predefined функций, которые называются гипероператорами. Все они имеют необычный синтаксис, например, `>> + <<`, причем вместо пары символов допускается использовать типографские кавычки (для Perl 6 это возможно прежде всего из-за того, что исходный текст программы записан в юникоде): `» + «`.

«Стрелки» гипероператора направлены от двух его аргументов, а знак указывает на выполняемое действие. Например, в показанном примере происходит попарное сложение элементов двух списков, а результат помещается в массив `@sum`.

И аргументы, и символ операции могут быть самыми разными, если выбранная комбинация имеет смысл. Например, чтобы создать новый массив с квадратами значений, можно написать `@sum >> ** << 2` или `@sum »**« 2`.

```
say "не февраль"  
  if $day == 30 | 31;
```

В Perl 6 появились объединения (*junctions*). Объединения — удобный способ компактно записать серию логических выражений, в которых одна и та же переменная сравнивается со множеством значений.

Объединение, показанное в примере, имеет альтернативную функциональную форму записи: `any (30, 31)`. Соответственно, все сравнение можно было бы записать в виде `if $day == any (30, 31)`.

Сравнение с участием `any` успешно в том случае, если сравнение успешно хотя бы с одним из значений, имеющихся в списке.

Противоположное по действию объединение — `none()`. Выражение `$day == none (30, 31)` истинно только тогда, когда переменная `$day` не равна ни одному из перечисленных значений.

Важно понимать, что `any()` и `none()` — вовсе не пара функций, которые всегда возвращают противоположные результаты.

```
multi sub f ($scalar);  
multi sub f (@arr);  
multi sub f ($a, $b);
```

В Perl 6 допустимо перегружать функции. Перегруженные функции следует предварить ключевым словом `multi`. Все варианты имеют одно и то же имя, однако число или тип аргументов должны отличаться.

При вызове функции (в показанном примере — `f`) компилятор решает неоднозначность на основе списка фактически переданных аргументов, поэтому записи `f(2)`, `f(1..10)` и `f(3, 4)` приведут к вызову разных функций.

Формально поведение перегруженных функций ничем не ограничивается. Однако, на практике нужно руководствоваться здравым смыслом и давать одинаковые имена лишь тем функциям, которые выполняют схожие действия.

Стоит заметить, что, например, оператор сложения `(+)` — это фактически семейство перегруженных функций, чьими аргументами могут быть переменные разных типов.

```
multi sub
infix:<минус> ($a, $b) {
    return $a - $b;
}
```

```
say 1 минус 2;
```

**Операторы тоже возможно перегрузить.** Операторы в Perl 6 — это функции, имеющие особый синтаксис. Существует около 20 различных грамматических категорий, к которым принадлежат все существующие операторы.

Самые простые категории — префиксы, инфиксы и постфиксы. Например, «плюс» — префиксный оператор, когда он переводит свой аргумент в числовой контекст, и инфиксный — при суммировании.

Синтаксис переопределения существующих и определения новых операторов похож на синтаксис определения функций и использует ключевое слово с названием грамматической категории оператора. Сам оператор записывают аналогично тому, как записывают ключи хеша.

Среди грамматических категорий, например, есть группы, которые позволяют переопределять поведение парных скобок или кавычек, в которые заключены строки.

```
given $x {  
    when 1 {say 'one'}  
    when $y {say 2}  
    when Int {say 'integer'}  
    when /\d+/ {say 'r&w'}  
    default {say '?'}  
}
```

В Perl 6 появился оператор выбора **given**. Новый оператор выбора, во-первых, дополнил перл механизмом, который в других языках реализован блоками `switch/case`, и во-вторых, разнообразил традиционное поведение такого блока новыми возможностями.

Используются три ключевых слова: **given** для объявления блока, **when** для начала ветви с условием и **default** для действия по умолчанию. После слова **given** следует выражение, которое участвует в сравнениях, производимых в наборе блоков **when**.

Условие **when** не обязательно должно быть простым сравнением с константой или другой переменной (как в первых двух строках показанного примера). Возможно, например, проверить тип переменной, записав `when Int`. Аналогично записываются сопоставления с регулярным выражением: `when /\d+ /`.

Блок **default** выполняется при неудачах сопоставлений в блоках **when**.

```
class Alphabet {  
}  
  
my $abc = new Alphabet;
```

**В Perl 6 полностью пересмотрен синтаксис классов.** Создание и использование классов в Perl 6 значительно изменилось и теперь намного ближе к подходам, используемым в других языках. Чтобы создавать класс, теперь не нужно вызывать функцию `bless`.

Класс объявляют с помощью ключевого слова `class`, за которым следует имя класса и его определение. Определение может быть заключено в блок кода (как в примере), либо, если объявление `class` стоит в начале файла, следовать сразу после заголовка (и простирается до конца файла).

Создание экземпляра класса осуществляет метод `new`; возможны две записи: `new Class` и `Class.new`.

Определения классов допустимо вкладывать друг в друга, но при этом имя внутреннего класса оказывается в том же пространстве имен, что и внешний класс. Управлять областью видимости возможно с помощью ключевых слов `our` и `my` перед словом `class`.

```
class Alphabet {  
    has $.Name;  
    has $Length;  
    has $!i;  
}
```

Переменные класса объявляются с помощью ключевого слова **has**. Класс может содержать переменные (они же атрибуты, они же члены-данные), которые объявляют аналогично обычным переменным, но вместо слова `my` пишут `has`. Тут же управляют областью видимости переменной, используя вторичный сигил (сигил, который стоит непосредственно за основным): открытые (и наследуемые) переменные содержат вторичный сигил-точку (`$.Name`), закрытые — восклицательный знак (`!Length`), который допустимо опускать (`Length`).

Доступ к переменным объекта некоторого класса осуществляется через вторичный сигил, использованный при объявлении: `$abc.Name` или `$abc!Length`. При объявлении переменной без вторичного сигила автоматически создается ее синоним с восклицательным знаком.

Переменная класса в свою очередь может быть объектом другого класса, в том числе определенного внутри того же класса, что и сама переменная.

```
class Alphabet {
  method say_name() {
  }
  submethod BUILD {}
  submethod DESTROY {}
}
```

Класс может содержать методы, объявленные как `method` и `submethod`. Для создания методов (функций-членов) класса предусмотрены ключевые слова `method` и `submethod`. Различие в том, что метод, объявленный как `submethod`, не наследуется.

Методы аналогичны обычным функциям (`sub`) за тем исключением, что получают в качестве первого аргумента ссылку на объект, для которого совершен вызов. Как и у функций, у методов могут быть позиционные или именованные аргументы: `method set_name ($new_name, $delay)`.

В теле метода доступ к текущему объекту (инвоканту) осуществляется через объект `self` (записывается без сигила), например: `say self.$!Name`. В объявлении функции этот объект можно явно указать среди других аргументов, поставив после него двоеточие: `method doit ($self: $x, $y)`.

Для конструкторов и деструктора предусмотрены ключевые слова `BUILD` и `DESTROY`.

```
class Latin
  is Alphabet
  is Ancient;
```

```
class Latin does Chars;
```

Классы Perl 6 поддерживают множественное наследование и роли. Чтобы унаследовать один класс от другого, нужно при объявлении указать имя базового класса после ключевого слова `is`. Допускается множественное наследование: для этого нужно повторить `is` необходимое число раз.

Кроме того, Perl 6 поддерживает механизм ролей. Роли синтаксически похожи на классы и объявляются с ключевым словом `role`. Как и классы, роли могут содержать методы и атрибуты (переменные класса), но основное назначение ролей — быть базовыми для других классов.

Если класс выполняет некоторую роль, используется ключевое слово `does` вместо `is`. В минимальном варианте, когда роль содержит только объявления методов, роли являются интерфейсами, — такими же, как в некоторых других языках.

Методы, объявленные без тела (используется синтаксис с многоточием: `method name {...}`), должны быть реализованы в производном классе.

# Документация

Основные документы, описывающие дизайн языка Perl 6, собраны на сайте <http://dev.perl.org/perl6/>. Порядок нумерации совпадает с главами в книге Camel Book, поэтому встречаются пропуски. Информация из части разделов этой книги устарела по отношению к шестой версии языка, а часть новой документации еще не написана.

На этой странице перечислены документы, которые доступны на октябрь 2007 года. Здесь использованы традиционные сокращения: A — Apocalypse, E — Ekegesis и S — Synopsis.

- 1 Overview [A1, S1](#)
- 2 Bits and Pieces [A2, E2, S2](#)
- 3 Summary of Perl 6 Operators [A3, E3, S3](#)
- 4 Blocks and Statements [A4, E4, S4](#)
- 5 Regexes and Rules [A5, E5, S5](#)
- 6 Subroutines [A6, E6, S6](#)
- 7 Formats [A7, E7](#)
- 9 Data Structures [S9](#)
- 10 Packages [S10](#)
- 11 Modules [S11](#)
- 12 Objects [A12, S12](#)
- 13 Overloading [S13](#)
- 16 IPC / IO / Signals [S16](#)
- 26 Documentation [S26](#)
- 29 Builtin Functions [S29](#)

## [perl6.ru](http://perl6.ru)

Информация о том, как работает Perl 6, как его установить под разные системы, о новых компиляторах и о событиях, связанных с Perl 6.

## [perlussia.ru](http://perlussia.ru)

Сайт о мероприятиях на русском языке, посвященных программированию на перле.

## [2008.perlussia.ru](http://2008.perlussia.ru)

Прием тезисов докладов на российский воркшоп 2008 года.

## [mail@perlussia.ru](mailto:mail@perlussia.ru)

Присылайте предложения на публикацию ваших статей по языкам Perl 5 и Perl 6.