PRAGMATIC PERL

28

06/2015

pragmaticperl.com

Pragmatic Perl 28

pragmaticperl.com

Выпуск 28. Июнь 2015

Другие выпуски и форматы журнала всегда можно загрузить с pragmaticperl.com. С вопросами и предложениями пишите на почту editor@pragmaticperl.com.

Комментарии к каждой статье есть в html-версии. Подписаться на новые выпуски можно по ссылке pragmaticperl.com/subscribe.

Авторы статей: Андрей Шитов, Наталья Савенкова, Дмитрий Шаматрин, Илья Чесноков, Владимир Леттиев

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2015-06-04 12:30

Оглавление

1. От редактора

Этим летом намечается еще два русскоязычных Perl-мероприятия. Читайте анонсы в этом выпуске.

Также предоставляем полный перевод изменений в недавно вышедшем perl5.22.

Друзья, журнал ищет новых авторов. Не упускайте такой возможности! Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2. Анонс рязанского Perl-воркшопа — 2015

Рязанский Perl Workshop 2015 — бесплатная конференция, посвященная языку программирования Perl, которая впервые проводится в Рязани.

Этот формат конференции призван объединить всех тех, кому язык Perl необходим в их профессиональной деятельности: программистов, системных администраторов и всех тех, кому язык помогает автоматизировать возникающие задачи.

Конференция будет проводится в субботу, 15 августа 2015, в офисе EPAM Systems по адресу: г. Рязань, ул. Гоголя, д. 16. Приглашаются участники любого уровня подготовки. Для участия нужно зарегистрироваться на сайте, участие в конференции бесплатно.

К нам приедут докладчики из Москвы, Санкт-Петербурга и других городов, работающие в ведущих IT-компаниях страны и зарубежья. Если вам есть что рассказать, мы будем рады вас выслушать, подать заявку на доклад можно здесь: http://event.yapcrussia.org/ryazan2015/newtalk

На 16 августа запланирован совместный выезд на природу — мы планируем снять домик за городом и съездить туда, чтобы отдохнуть, пожарить шашлыки и пообщаться с разработчиками мирового уровня :-) Для участия в этой поездке добавьте это мероприятие в персональное расписание (непосредственно перед поездкой будет организован сбор средств с участников, чтобы покрыть расходы).

Если вы представляете организацию, которая заинтересована в развитии языка Perl, и хотите оказать спонсорскую поддержку мероприятию, пишите по адресу: ryazan@yapcrussia.org.

До встречи на конференции!

■ *Илья Чесноков*

3. Анонс «Perl Mama» — 2015

Таки первая одесская конференция по Perl — Perl Mama 2015.

В субботу и воскресенье, 25-26 июля 2015 года, в Одессе пройдет небольшая конференция, посвященная программированию на Perl. Солнце, море и тонны позитива в комплекте. Участие бесплатное.

Программа конференции выглядит примерно следующим образом:

24 июля, пятница

День приезда. В 19:00 начинается препати, подробная информация о том, где это самое препати будет, появится ближе к началу июля на сайте конференции.

25 июля, суббота

С 10:00 до 18:00 будут доклады, кофе-брейки и основная нагрузка конференции. С 18:00 и пока не надоест будет афтерапати в замечательном одесском ресторане средневековой кухни.

26 июля, воскресенье

С 10:00 до 17:00 — конференция. С 17:30 и опять-таки, пока не надоест, еще одно афтерпати.

На данный момент регистрация открыта, ищем доклады. Спонсоров пока не ищем, но если вдруг они появятся, их список будет тут. Сайт конференции: http://perlmama.od.ua.

По любым вопросам, а также с любыми предложениями обращайтесь к Дмитрию Шаматрину, Вячеславу Тихановскому или Андрею Шитову в любое время.

Информационные партнеры:

- Educator.io — smart classroom in the cloud.

■ *Дмитрий Шаматрин*

4. Perl Golf на YAPC::Russia 2015

Рассказ про соревнование Perl Golf, проходившее в течение двух дней на майской YAPC::Russia в Москве

Perl Golf — это конкурс на написание самой короткой программы на перле (подробнее об этом можно прочитать в ранних выпусках журнала Pragmatic Perl). Задание для гольфа YR2015 озвучил Вадим Пуштаев в начале первого дня конференции.

Решения, присланные участниками, размещены в репозитории YAPC_-Russia_2015_perl_golf на гитхабе.

Задача. Скрипт принимает два параметра — длину и ширину матрицы — и по порядку печатает в заданном поле числа, закручивая их по спирали внутрь:

```
$ perl script.pl 7 6
1  2  3  4  5  6  7
22 23 24 25 26 27 8
21 36 37 38 39 28 9
20 35 42 41 40 29 10
19 34 33 32 31 30 11
18 17 16 15 14 13 12
```

Колонки должны быть выровнены влево и отделены друг от друга ровно одним пробелом.

Минимальное решение `bronton.pl`, которое шло в комплекте с заданием, содержало 338 символов, и сабмитить более длинное не было смысла. А за длиной остальных было интересно наблюдать, обновляя страницу новостей на сайте, где публиковались промежуточные результаты, но, конечно, без самих решений. В какой-то момент было три решения в 244, 245 и 246 символов, и казалось, что эти кандидаты могут побороться еще за один-два символа, чтобы обогнать друг друга. Но почти перед дедлайном конкурса Денис Ибаев запостил решение в 205 символов, которое уже само по себе никому не оставило шанса, и потом улучшил свой рекорд до 202 символов. Для меня самое удивительное, что Денис написал свой код на айпаде.

```
202 dionys.pl
208 agrishaev.pl
213 andrey_shitov.pl
```

```
229 nikolas_shulyakovskiy_s.pl
243 kyusev.pl
246 andrey_fedorov.pl
258 khripunov.pl
294 bor_vas.pl
303 orlenko_aleksandr.pl
326 evgeniy_kim.pl
338 bronton.pl
```

Я расскажу о своем решении (и предлагаю читателям самостоятельно разобраться в коде других участников :-).

Первым делом надо было понять, как оптимальнее закручивать матрицу. Я хотел придумать или найти формулу, вычисляющую значение клетки по ее координатам. В интернетах описана спираль Улама — она такая же, как в задании, но раскручивается из центра, — и даже находится не очень внятная формула, которая позволяет вычислить значение по координатам, но я не пошел этим путем, потому что было неясно, как же рассчитывать число пробелов между колонками. Имея формулу, мне хотелось сразу печатать значения, не создавая два цикла — один для генерации данных, а второй для их печати.

Но в итоге пришлось делать два этапа: генерацию и печать. ОК, чтобы закрутить спираль, надо понять, как поворачивать на углах и как не испортить уже заполненные клетки. Я порисовал матрицы на бумажке (здесь очень пригодились блокноты со страницами «в точечку», выдаваемые организаторами конференции) и заметил, что каждый поворот спирали уменьшает длину следующего прямолинейного отрезка на единицу. То есть для матрицы размером 7×5 первые два поворота будут через 7 и 4 шага, следующие два через 6 и 3, затем через 5 и 2 и так далее.

Этот алгоритм отражен в первом варианте решения.

```
#!/usr/bin/perl

($w,$h)=@ARGV; # ширина и высота

exit if !($w*$h);
#$n=0; # величина, записываемая в клетку
#$x=0; # начальное положение по горизонтали

$y=1; # начальное положение по вертикали

```

```
$v=1; # вектор движения по горизонтали
$u=1; # и по вертикали

# основной цикл – по ширине
while ($wi=$w--) { # здесь упомянутое выше уменьшение длины
    while ($wi--) { # прямолинейного отрезка
        $x+=$v; # перемещение на одну клетку вправо или влево
        $a[$x][$y]=++$n; # фиксирование результата
        $L[$x]=$n; # @L -- одномерный массив по числу строк,
                   # по мере генерации в него записывается
                   # текущее число, поэтому к моменту печати
                   # в каждом элементе будет максимальное число,
                   # которое возможно в соответствующей колонке,
                   # а значит известна и нужная ширина колонки.
    };

    $v=-$v; # подготовка для следующего горизонтального участка,
            # где смещение будет -1, то есть справа налево.

    $hi=$h--;
    while(--$hi) { # а теперь идем по вертикали
        $y+=$u;    # вверх (-1) или вниз (+1)
        $a[$x][$y]=++$n;
        $L[$x]=$n;
    }
    $u=-$u;
    last if !$h;
}

($w,$h)=@ARGV; # исходные переменные уже испорчены, читаем заново

# нехитрые циклы для печати @a
for $y (1..$h) {
    for $x (1..$w) {
        # хитрость только в выводе через
        # форматную строку типа "%-5i" с длиной из @L
        $s = $x != $w ? 1+length $L[$x] : '';
        printf "%-${s}i", $a[$x][$y];
    }
    print "\n";
}
```

Кстати, решение `dionys.pl` не содержит двумерной матрицы, а данные читаются из одного массива.

По ходу решения выяснилось, что просто посчитать ширину колонки как 1

`+ length $w*$h` нельзя, потому что может оказаться так, что даже если самое длинное число будет только одно, оно сразу сделает одну из колонок шире всех остальных.

```
$ perl script.pl 10 10
1  2  3  4  5   6  7  8  9  10
36 37 38 39 40  41 42 43 44 11
35 64 65 66 67  68 69 70 45 12
34 63 84 85 86  87 88 71 46 13
33 62 83 96 97  98 89 72 47 14
32 61 82 95 100 99 90 73 48 15
31 60 81 94 93  92 91 74 49 16
30 59 80 79 78  77 76 75 50 17
29 58 57 56 55  54 53 52 51 18
28 27 26 25 24  23 22 21 20 19
```

Прежде всего можно немного сэкономить на повторном извлечении размеров таблицы:

```
($p,$q)=($w,$h)=@ARGV;
```

Проверка на правильность введенных данных тоже не нужна, потому что таких входных параметров нет в тесте, проверяющем правильность решения :-) При нулевых и отрицательных значениях программа может уйти в бесконечный цикл, но это неважно.

```
exit if !($w*$h);
```

Ширина колонки определяется по длине числа в соответствующем элементе массива `@L`. Чтобы избавиться от пробелов в конце строк, нужно сделать длину последнего элемента нулевой:

```
$L[$p]='';
```

Это дает возможность исключить исходную проверку при вычислении ширины колонки:

```
$s = $x != $w ? 1+length $L[$x] : '';
```

Теперь хочется избавиться от проверки условий для завершения. Цикл для генерации спирали выглядит примерно так:

```
while ($wi=$w--) {
   . . .
   last if !$h;
}
```

Вместо этого можно написать так:

```
while ($h * ($wi=$w--)) {
    . . .
}
```

Инициализацию переменных (тех, которые не получилось начать с нуля) можно сделать в одну строку:

```
$y=$v=$u=1;
```

То же самое можно сделать для следующих двух строк, и вместо

```
$a[$x][$y]=++$n;
$L[$x]=$n;
```

записать

```
$L[$x]=$a[$x][$y]=++$n;
```

Теперь небольшая, но полезная оптимизация цикла печати с использованием постфиксной записи `for`:

```
for $y (1..$q) {
    printf "%-".(1+length$L[$_])."i", $a[$_][$y] for 1..$p;
    print "\n";
}
```

Потом наступило понимание, что изменение переменных `$u` и `$v` всегда происходит друг за другом, и они никогда не используются вместе, то есть к тому моменту, когда они нужны для вычислений, они всегда содержат одинаковое значение. Поэтому обе я заменил одной переменной `$d`, и ее знак меняется в конце тела внешнего цикла:

```
$d=1;

while ($h * ($i=$w--)) {
    . . .
    $d=-$d;
}
```

Вложенные циклы `while` тоже следует заменить постфиксной записью, но сначала можно попровать вынести повторяющийся код в функцию, и это действительно помогло заметно сэкономить:

```
sub p {
```

```
    $L[$x]=$a[$x][$y]=++$n
}
```

Вместо

```
while($wi--) {
    $x+=$v;
    $L[$x]=$a[$x][$y]=++$n;
}
```

и

```
while(--$hi) {
    $y+=$u;
    $L[$x]=$a[$x][$y]=++$n;
}
```

теперь так:

```
$x+=$d, p while $i--;

. . .

$y+=$d, p while --$i;
```

Было сильное желание разобраться с тем, как привести декременты `$i--` и `--$i` к одному виду, чтобы убрать повторяющийся код, но быстро это сделать не получилось. Вместо этого я потратил время на неудачную попытку создать еще одну функцию, которая будет менять либо переменную `$x`, либо `$y`:

```
#sub e {
#   eval "\$$_[0]+=$d, p while $_[1]"
#}

. . .

#e 'x','$i--';
. . .
#e 'y','--$i';
```

Но при таком подходе код оказался длиннее исходного.

Зато получилось избавиться от инициализации `$y = 1`. Исходное значение теперь 0, а запись `$y = 0` для гольфа не нужна.

Итоговый цикл вычисления спирали теперь такой:

```
$d=1;
while ($h * ($i=$w--)) {
    $x+=$d, p while $i--;
    $i=$h--;
    $y+=$d, p while --$i;
    $d=-$d;
}
```

Еще один подход к этапу печати. Сначала экономлю один символ, заменяя `\n` настоящим переводом строки в коде:

```
print "
"
```

Но потом, почитав perldoc perlop и Perl Golf 101, понимаю, что гольфисты уже все давно придумали и вместо `"\n"` пишут `$/`.

И небольшой, но удачный эксперимент по замене конкатенации строк

```
printf "%-".(1+length$L[$_])."i"
```

интерполяцией с выполнением кода в строке `"@{[]}"`:

```
printf "%-@{[1+length$L[$_]]}i"
```

Готово. 213 символов:

```
sub p{$L[$x]=$a[$x][$y]=++$n}($p,$q)=($w,$h)=@ARGV;$d=1;while($h
   *($i=$w--))
{$x+=$d,p while$i--;$i=$h--;$y+=$d,p while--$i;$d=-$d}$L[$p]='';
for$y(0..$q-1){printf"%-@{[1+length$L[$_]]}i",$a[$_][$y]for 1..$p
   ;print$/}
```

И еще о нескольких интересных приемах, которыми воспользовались другие участники.

Александр Орленко сразу продублировал входные параметры:

```
($w,$h,$y,$z)=(@ARGV)x2;
```

Хотя эта запись на два символа менее выгодна аналогичной:

```
($p,$q)=($w,$h)=@ARGV;
```

В авторском примере `bronton.pl` есть еще один интересный подход к начальному присваиванию:

```
($c,$m,$n)=(1,@ARGV);
```

Хотя он длиннее, чем если делать все по-отдельности:

```
$c=1;($m,$n)=@ARGV;
```

Николай Шуляковский применил редкий оператор-«качель» `...`:

```
do{$r[$j][$v+($i+=$v)]+1}...do{$r[$v+($j+=$v)][$i]+1and$v*=-1}
```

Андрей Федоров меняет направление, используя умножение на -1 или на 0, хитро вычисляя его из текущей позиции:

```
$X*=!!$Y--;$k*=-1;
```

Кстати, вариант с умножением ничем не отличается от явной смены знака:

```
$k*=-1;
$k=-$k;
```

Сергей Хрипунов использовал полезный прием для экономии на слове `length`:

```
$l=y///c;
```

В решении Евгения Кима места поворотов спирали вычисляются с помощью матрицы (предвычесленные значения всегда хорошо):

```
@r=([1,0],[0,1],[-1,0],[0,-1]);
. . .
($x,$y)=@{$r[++$d%4]}
```

Анатолий Гришаев сразу создает матрицу-заготовку и затем заполняет ее числами:

```
($a,$b)=@ARGV;$_=('1 'x$a++."0\n")x$b;@s=split;s/\d/$z++/ge;
```

■ *Андрей Шитов*

5. Рефакторинг Legacy

Рефакторинг устаревшего кода в примерах

В отличие, пожалуй, от многих программистов, которые очень не любят легаси-код, для меня рефакторинг такого кода это какое-то особенное удовольствие. Возможно, женская натура любит убирать бардак.

Рефакторинг бывает двух видов: невменяемый и вменяемый. Невменяемый рефакторинг это процесс ради процесса. Увидел новый подход и сразу: «давайте перепишем!» Но если ты работаешь в коммерческой структуре, это недопустимо. Никто не будет платить тебе за то, чтобы ты «поигрался с новым кодом». К тому же, гипер-рефакторинг рискует вообще не перейти в продакшн по причине своей глобальности. Поэтому он должен быть разумен и обоснован. Например, если нужно добавить или изменить функционал, а ты смотришь на древний код, который итерационно приобретал все больше и больше костылей и ветвлений, и решаешь, что так больше продолжаться не может. Но как всегда тот кусок кода, который нужно заменить, это функция-мутант ростом в три-четыре сотни строк, в лучшем случае.

Пусть у нас будет такой искуственный пример:

```
1  package Human;
2  use strict; use warnings;
3  sub do {
4      my ($logger, $text, $key) = @_;
5      if ($key eq 'rree') {
6          # ... some code
7          # normalize
8          $text =~ s/[!\—\+]/ ! /g;
9          $text =~ s/\(\(/ ( /g;
10         $text =~ s/\)\)/ ); /g;
11         if ($text =~ /superpower/) {
12             $logger—>log('warn', 'superpower used');
13             $text =~ s/superpower/secret weapon/g;
14         }
15         # ... more normalize rules !!!
16         # ... some code
17         # prepare
18         $text = sprintf '%s %s', $text, $key;
19         unless ($text) {
20             $logger—>log('warning', 'no text');
21         }
```

```
22      }
23      elsif ($key eq 'jjii') {
24          # ...
25      }
26      return $text;
27  }
28  1;
```

Допустим, в блок кода, где мы нормализуем переменную `$text`, мне нужно добавить еще одну регулярку, например, такую:

```
1 $text =~ s/gg|rr|e|t|q//g;
```

В моем примере блок регулярок не очень большой, потому что он сильно упрощен. Казалось бы, что вполне можно еще добавлять и добавлять. Но все равно наступит тот момент времени, когда это выйдет за рамки разумного.

Лучший друг рефакторинга это **TDD — Test-driven development** (разработка через тестирование), потому что мы хотим не просто сделать красиво, но чтобы оно еще и работало как раньше. Начнем с написания теста. Классически имя теста формируется по шаблону `t–<MODULE>.t`, поэтому наш тест назовем `t–human.t`.

И вот так он будет выглядеть:

```
1  #!/usr/bin/perl
2  use strict;
3  use Test::More;
4  use_ok('Human');
5  done_testing();
```

Запустив в консоли:

```
1 $ prove t–human.t
```

Увидим такой результат:

```
1 t–human.t .. ok
2 All tests successful.
3 Files=1, Tests=1,  0 wallclock secs ( 0.01 usr  0.00 sys +  0.01
    cusr  0.00 csys =  0.02 CPU)
4 Result: PASS
```

Конечно тесты пройдены, там же проверяется только `use`, а фрагмен с рабочего проекта, с чего бы ему не работать.

Дальше выносим нашего пациента в отдельную функцию. Здесь мы получаем сразу несколько плюсов: код становится чище и понятнее, а отдельно вынесенную функцию можно нормально тестировать, не ударяясь в тестирование функции do (допустим, тестов для нее никогда не было, и функционал ее безграничен).

Теперь у нас такой код:

```
#... code
    if ($key ne 'rree') {
        # ... some code
        # normalize
        my $text = _normalize($logger, $text);
        # ... some code
        # prepare
        $text = sprintf '%s %s', $text, $key;
        unless ($text) {
            $logger->log('warning', 'no text');
        }
    }
#... code
sub _normalize {
    my ($logger, $text) = @_;
    $text =~ s/[!\-\+]/ ! /g;
    $text =~ s/\(\(/ ( /g;
    $text =~ s/\)\)/ ); /g;
    if ($text =~ /superpower/) {
        $logger->log('warn', 'superpower used');
        $text =~ s/superpower/secret weapon/g;
    }
    # ... more normalize rules !!!
    return $text;
}
```

И проверяем:

```
$ prove t-human.t
```

Тест до сих пор работает, показывая, что синтаксически все ОК.

Наша задача — не просто вынести в фукцию, но еще и дописать ее, сохранив прежний функционал. Поэтому пишем тест для текущей логики:

```
my $tests = {
    '!abc+' => ' ! abc ! ',
    '((bbb))' => ' ( bbb ); ',
    'i want to use superpower' => 'i want to use secret weapon',
```

```
};

for my $t (keys %$tests) {
    is Human::_normalize($logger, $t), $test->{$t};
}
```

Но вот чертовщина! Функция имеет зависимость от неведомого `$logger`, у которого еще и методы вызывает. Просто передать `undef` не сработает. Здесь нужно себя перебороть и не удариться в невменяемый рефакторинг, пытаясь избавиться от этой зависимости. Пока будет достаточно просто вынести в функцию и на этом успокоиться. Но я не хочу писать сообщения в непонятный лог, я хочу их просто вывести в консоль и только в режиме `-v`.

`"ref $logger"` подсказывает, что это объект класса `Logger`. Ок, будем использовать его, но заменив метод `log`.

Делаем Mock с помощью `Test::MockModule` (сначала я использовала `Test::Mock::Simple`, но первый есть в libtest-mockmodule-perl, это определило выбор), и теперь наш фрагмент теста выглядит вот так:

```
# mock
my $module = Test::MockModule->new('Logger');
$module->mock('log', sub { shift; note explain ['IN MOCK', @_] })
    ;
my $logger = Logger->new();
for my $t (keys %$tests) {
    is Human::_normalize($logger, $t), $tests->{$t};
}
```

Здесь мы заменяем метод `log` объекта `Logger` простым выводом в консоль в режиме `-v` (метод `note`), а `explain` отдает человекопонятный дамп стека.

Запускаем тест:

```
$ prove t-human.t
t-human.t .. ok
All tests successful.
Files=1, Tests=5,  0 wallclock secs ( 0.01 usr  0.00 sys +  0.01
   cusr  0.00 csys =  0.02 CPU)
Result: PASS
```

И вот так будет выглядеть вывод при использовании режима verbose:

```
$ prove t-human.t -v
t-human.t ..
```

```
ok 1 – use Human;
ok 2 – use Logger;
# [
#   'IN MOCK',
#   'warn',
#   'superpower used'
# ]
ok 3
...
```

Теперь, когда мы знаем, что наша функция нормально работает на старых данных, можно расширять ее функционал. Конечно, тестовый массив в общем случае значительно больше, чем в этом примере. Тут нужно руководствоваться просто здравым смыслом.

Сначала дописываем тесты для нового функционала:

```
my $tests = {
    '!abc+' => ' ! abc ! ',
    '((bbb))' => ' ( bbb ); ',
    'i want to use superpower' => 'i want to use secret weapon',
    'gg' => '',
    '!ggabcrr))' => ' ! abc ); ',
};
```

Затем я всегда проверяю, что мой тест действительно перестал работать. Это паранойя? Но вдруг...

```
t–human.t .. 1/?
#   Failed test at t–human.t line 23.
#          got: ' ! ggabcrr ); '
#     expected: ' ! abc ); '

#   Failed test at t–human.t line 23.
#          got: 'gg'
#     expected: ''
# Looks like you failed 2 tests of 7.
```

Дописываем логику в код:

```
sub _normalize {
    my ($logger, $text) = @_;
    $text =~ s/[!\–\+]/ ! /g;
    $text =~ s/\(\(/ ( /g;
    $text =~ s/\)\)/ ); /g;
    $text =~ s/gg|rr|e|t|q//g; # вот наше новое условие
    if ($text =~ /superpower/) {
```

```
8         $logger->log('warn', 'superpower used');
9         $text =~ s/superpower/secret weapon/g;
10    }
11    # ... more normalize rules
12    return $text;
13 }
```

И наслаждаемся тем, что тесты проходят и все отлично! Ура.

```
1 $ prove t-human.t
2 t-human.t .. 1/?
3 #   Failed test at t-human.t line 23.
4 #          got: 'i wan o us suprpowr'
5 #     expected: 'i want to use secret weapon'
6 # Looks like you failed 1 test of 7.
```

Ого, что-то пошло не так. Конечно! Наша новая логика вырезает слишком много всего. Немного подправим код:

```
1 if ($text =~ /superpower/) {
2     $logger->log('warn', 'superpower used');
3     $text =~ s/superpower/secret weapon/g;
4 }
5 else {
6     $text =~ s/gg|rr|e|t|q//g; # перенесли сюда
7 }
```

```
1 $ prove t-human.t
2 t-human.t .. ok
3 All tests successful.
4 Files=1, Tests=7,  0 wallclock secs ( 0.02 usr  0.00 sys +  0.01
    cusr  0.00 csys =  0.03 CPU)
5 Result: PASS
```

Теперь все ОК!

Идеально, если тест покрывает 100% кода. Это, конечно, светлое и недостижимое будущее, но по крайней мере мы можем хотя бы стремиться к этому. Покрытие можно замерить с помощью `Devel::Cover`.

```
1 $ perl -MDevel::Cover t-human.t
2 Devel::Cover: Writing coverage database to /home/wwax/Desktop/
    pragmatic/1/cover_db/runs/1431793338.4403.02108
3 ---------------------------------- ------ ------ ------ ------
    ------ ------
4 File                               stmt   bran   cond    sub
      time  total
```

```
5 ------------------------------------------ ------ ------ ------ ------
    ------ ------
6 Human.pm                                   62.5   25.0    n/a   75.0
       1.3   55.5
7 Logger.pm                                 100.0    n/a    n/a  100.0
       0.0  100.0
8 t-human.t                                 100.0    n/a    n/a  100.0
      98.6  100.0
9 Total                                      81.2   25.0    n/a   90.0
     100.0   75.7
10 ------------------------------------------ ------ ------ ------ ------
    ------ ------
```

Нам важна только первая строка. Неплохие результаты, учитывая, что до этого никакого тестирования там вообще не было. Если в модуле `Human.pm` оставить только единственную функцию `_normalize`, ради которой это все затевалось, то `cover` показывает 100% =). Это значит, что все функции, операторы и ветвления были пройдены в результате нашего теста.

```
1 ------------------------------------------ ------ ------ ------ ------
    ------ ------
2 File                                       stmt   bran   cond    sub
      time  total
3 ------------------------------------------ ------ ------ ------ ------
    ------ ------
4 Human.pm                                  100.0  100.0    n/a  100.0
       1.2  100.0
```

Успешного рефакторинга! И я надеюсь, будет продолжение =)

■ *Наталья Савенкова*

6. Что нового в Perl 5.22

Краткий обзор наиболее заметных изменений в свежем стабильном релизе перла

Версия 5.22, в отличие от нескольких предыдущих, содержит некоторые интересные новшества, о которых стоит рассказать.

Регулярные выражения

...

Новый перл поставляется с новым юникодом версии 7.0. Одновременно появилось несколько выражений, которые можно использовать для поиска границ букв, слов и предложений:

- `\b{gcb}` или `\b{g}` — границы букв (точнее, графем: grapheme cluster boundary);
- `\b{wb}` — границы слов (word boundary);
- `\b{sb}` — границы предложений (sentence boundary).

Пробуем делить по графемам (важно не забыть дописать инструкции про utf8).

```
use v5.22;
use utf8;
use open qw(:std :utf8);

my @str = ('Über', "U\x{308}ber");

for (@str) {
    my @ch = split //;
    say scalar @ch;
    say join ',', @ch;

    my @gl = split /\b{g}/;
    say scalar @gl;
    say join ',', @gl;
```

```
15 }
```

Ожидаемо, первый `split //` делит строку на байты (4 для первой строки и 5 для второй), а второй `split /\b{gcb}/` на буквы.

Выражение `\b{wb}` совпадает на границе юникодных слов. Причем речь идет не только о составных символах (как на примере Ü выше). Например, слова с апострофом принимаются за слово целиком вместе с апострофом.

```
1 my $book = "Perl pour l'impatient";
2 say for grep /\S/, split /\b{wb}/, $book;
```

Этот пример напечатает три слова, а если разделить строку по шаблону `/\b/`, то пять слов (включая апостроф, выделенный в отдельное слово).

Наконец, выражение `\b{sb}` совпадает с границами предложений. Границей предложения могут стать и кавычки с цитатой, и точки после инициалов и даже перевод строки (точнее, все, что совпадает с `/\R/`). То есть реальной пользы от `\b{sb}`, видимо, не очень много.

```
1 my $text = <<TEXT;
2 Ultimately, it's Larry's choice. I haven't been able to convince
    Larry, he hasn't been able to convince me, but he has good
    reasons for the choices that he's made and he's a really smart
    guy. It might be that having Perl 5 and Perl 6 will actually
    turn out to be an advantage in some way that most of the rest
    of the community doesn't realize until we look back and say: “
    You know what? Larry was right again!”
3 TEXT
4
5 my @s = $text =~ /(.+?) *\b{sb}/g;
6 say "[$_]" for @s;
```

Подробные правила, принятые в юникоде для членения текста на значимые части, описаны в документе Unicode Text Segmentation.

/n для запрета захвата

В дополнение к `(?:...)` появился модификатор `/n`, который действует сразу на все выражение.

```
1 #'2015-06-02' =~ /(\d+)-(\d+)-(\d+)/; # захватит
```

```
'2015-06-02' =~ /(\d+)-(\d+)-(\d+)/n; # не захватит

say $1;
say $2;
say $3;
```

use re ‘strict’

В качестве экспериментальной добавлена возможность дополнительной проверки правильности регулярных выражений:

```
no warnings 'experimental::re_strict';
use re 'strict';
```

В документации [показан пример](http://search.cpan.org/~rjbs/perl-5.22.0/ext/re/re.pm#‘s
mode), когда такая проверка выдает предупреждение:

```
 qr/\xABC/
```

Без strict-режима все произойдет молча, а при наличии `use re 'strict';` появится сообщение:

```
Use \x{...} for more than two hex characters in regex;
```

Более явные ошибки, тем не менее, удается отследить и в обычном режиме. Например, попытка использовать несуществующий режим для поиска границ `\b{ww}`:

```
'ww' is an unknown bound type in regex;
```

Многое, по видимому, упирается в обратную совместимость. Было бы неплохо в будущем включать строгий режим по умолчанию, как минимум вместе с директивой `use v5.22;`, как это сейчас происходит с обычным `use strict;` при указании, например, `use v5.12;`.

Флаги по умолчанию

Инструкция вида `use re '/ax'` автоматически подключает все указанные в ней модификаторы ко всем регулярным выражениям до конца текущей обла-

сти видимости. Соответственно, `no re '/ax'` отключает перечисленные флаги.

```
use re '/i';

my @p = "ABC" =~ /([a–z])/g;
say for @p;
```

В этой программе к регулярному выражению добавится модификатор `/i`, поэтому программа напечатает три строки с прописными буквами.

Обратите внимание, что сразу записать `use re '/g'` невозможно:

```
Unknown regular expression flag "g" at regex–flags.pl line 3.
```

Впрочем, это невозможно и в записью вида `(?i:...)`:

```
my @p = "ABC" =~ /(?ig:[a–z])/;
```

Здесь сразу на помощь приходит `use re 'strict'`:

```
Useless (?g) – use /g modifier in regex;
```

Операторы

Битовые операторы

Экспериментальная фича `bitwise` активирует набор операторов `|.`, `&.`, `^.`, `~.` и, соответственно, `|.=`, `&.=`, `^.=`, `~.=`, которые выполняют побитовые действия, рассматривая аргументы как строки.

```
use feature 'bitwise';
no warnings 'experimental::bitwise';

say 400 |  142; # 414
say 400 |. 142; # 542

say 40 |  12;   # 44
say 40 |. 12;   # 52
```

В примерах с `|` выполняется операция над числами: например, `40|12` это `0x28|0x0C` или `0b00101000|0b00001100`, что дает `0b00101100 = 44`.

С новым оператором побитовые операции происходят посимвольно: `ord('4')|ord('1') = 52|49 = 0b110100|0b110001 = 0b110101 = 53`, то есть символ 5, а `ord('0')|ord('2') = 48|50 = 0b110000|0b110010 = 0b110010` (символ 2).

В операторах с точкой наличие или отсутствие пробела после нее решается в пользу более длинного оператора, сравните:

```
say 40 | .12; # 40
say 40 |. 12; # 52
say 40 |.12;  # 52
```

Аналогично работают и другие операторы.

Строковое побитовое сложение наверное удобно использовать для совмещения двух ASCII-строк, в которых символы из одной встают на пробельные позиции в другой (правда, заодно происходит преобразование в нижний регистр):

```
say "Hello,      !" |. "       World"; # hello, world!
```

С юникодом такие фокусы не имеют смысла:

```
use utf8;
say "Привет,    !" |. "        мир"; # пЫивеЂ, миЫ!
```

Тем не менее, введение новых операторов — вполне в пределах картины мира перла, в которой часть операторов не полиморфна, а разделяется на строковые и числовые (ср. с операторами сравнения). Кстати, изначально предлагалась сделать новые операторы более последовательными, а именно, для строковых операций использовать строковые же названия `band`, `bor`, `bxor` и `bnot`.

<<>>

Двойной ромб (double diamond) — более безопасная версия оператора <>. Отличие проявляется, когда он используется для чтения из файлов, имена которых указаны в командной строке.

Если файлов нет, то чтение происходит из стандартного потока ввода, и никакой разницы нет:

```
my $str = <>;
say $str;
```

При чтении из STDIN второй вариант делает то же самое:

```
my $str = <<>>;
say $str;
```

Если же при запуске программы в командной строке были указаны аргументы, то оператор-ромб последовательно открывает все файлы и читает из них, причем если аргумент содержит специальные символы типа > или |, то они ведут себя так же как в функции open, вызванной с двумя аргументами:

```
open my $f, 'ls |';
```

Следующий пример печатает содержимое текущего каталога:

```
open my $f, 'ls |';
print while <$f>;
```

Если такой аргумент передать в командной строке и при этом использоавть традиционный ромб, то происходит ровно то же:

```
print while <>;
```

Запуск:

```
perl diamond3.pl  'ls |'
```

Двойной ромб в таких случаях открывает файл вариантом функции open с тремя аргументами, и особой интерпретации таких символов не происходит:

```
open my $f, '<', 'ls |';
print while <$f>;
```

Эта программа будет пытаться открыть файл с именем `ls |`. Аналогично, чтобы открыть этот файл из командной строки:

```
perl ddiamond3.pl 'ls |'
```

достаточно воспользоваться двойным ромбом:

```
print while <<>>;
```

Новая многосимвольная форма оператора, судя по всему, потребовалась только для обратной совместимости.

Прочее

Из интересных возможностей, которые здесь не упомянуты, я бы обратил внимание на следующее:

- псевдонимы через ссылки,
- атрибут `:const`,
- `x` в списках при присваивании.

Подробности можно прочитать в переводе perldelta, опубликованном в этом номере журнала.

■ *Андрей Шитов*

7. Полный список изменений в Perl 5.22.0

Перевод документа perl5220delta.pod на русский язык

Название

perl5220delta - что нового в perl v5.22.0

Описание

Этот документ описывает изменения между релизом 5.20.0 и 5.22.0.

Если вы обновляетесь с более раннего релиза, как например, 5.18.0, сначала прочтите perl5200delta, который описывает отличия между 5.18.0 и 5.20.0.

Базовые улучшения

Новые битовые операторы

Была добавлена новая экспериментальная возможность, которая позволяет четырём стандартным битовым операторам (`& | ^ ~`) всегда воспринимать свои операнды как числа, а также представляет четыре новых оператора с точкой (`&. |. ^. ~.`), которые соответственно трактуют операнды как строки. Тоже самое справедливо и для их вариаций с присвоением (`&= |= ^= &.= |.= ^.=`).

Для активации этой возможности необходимо подключить возможность “bitwise” и отключить категорию предупреждений “experimental::bitwise”. Смотрите подробности в “Bitwise String Operators” in perlop и [perl #123466].

Новый оператор двойной ромб

<<>> подобен <>, но использует трёхаргументный вызов open для открытия файлов из @ARGV. Это также означает, что каждый элемент @ARGV будет рассматриваться как имя файла, и, например, "|foo" не будет трактоваться как открытие канала.

Новые границы в b для регулярных выражений

qr/b{gcb}/ gcb — сокращение от Grapheme Cluster Boundary (граница кластера графемы). Это свойство Юникода, которое ищет границы между последовательностью символов, которые являются представлением одного символа для носителя языка. В Perl давно поддерживалась такая возможность с использованием последовательности экранирования \X. Теперь же появился альтернативный способ. Смотрите подробности в “_,␣ -,’’ in perlrebackslash

qr/b{wb}/ wb — сокращение от Word Boundary (граница слова). Это свойство Юникода, которое ищет границы между словами. Это полностью совпадает с обычным \b (без угловых скобок), но больше подходит при обработке текстов на естественных языках. Свойство, к примеру, знает, что апостроф может находиться в середине слова. Подробнее смотрите в “_,␣ -,’’ in perlrebackslash.

qr/b{sb}/ sb — сокращение от Sentence Boundary (граница предложения). Это свойство Юникода, которое помогает при разборе предложений на естественных языках. Смотрите подробности в “_,␣ -,’’ in perlrebackslash.

Флаг регулярных выражений для отключения захвата

Регулярные выражения теперь поддерживают флаг /n, который отключает захват и заполнение переменных $1, $2, и прочих внутри группировки:

```
1 "hello" =~ /(hi|hello)/n; # $1 не устанавливается
```

Это эквивалентно установке ?: в начале каждой захватывающей группы.

Смотрите подробности в “n” in perlre.

`use re 'strict'`

Это выражение включает строгие синтаксические правила к шаблонам регулярных выражений, скомпилированным в области видимости данной прагмы. Это позволит предупредить вас об опечатках и других нежелательных последствиях, о которых нет возможности сообщить при обычной компиляции регулярных выражений из-за соображений обратной совместимости. Поскольку поведение подобной прагмы может измениться в будущих версиях Perl, по мере накопления опыта её применения, её использование приводит к выводу предупреждения категории `experimental::re_strict`. Смотрите подробнее в [‘strict’ в re](https://metacpan.org/pod/re#‘strict’ mode).

Поддерживается Юникод 7.0 (с корректировками)

Подробнее детали о том, что есть в этом релизе смотрите в http://www.unicode.org/versions/Unicode7.0.0/. Версия Юникод 7.0, поставляемая с Perl, включает корректировки, имеющие отношения к формированию Арабских глифов (смотрите http://www.unicode.org/errata/#current_errata).

`use locale` может ограничивать затронутые категории локалей

Теперь появилась возможность передавать параметр в `use locale` для указания списка категорий локали, для которых будут применяться правила локали, оставляя остальные категории незатронутыми. Подробности смотрите в [“The”use locale” pragma” in perllocale](https://metacpan.org/pod/perllocale#The%20“use%20

Perl теперь поддерживает дополнения локальной валюты POSIX 2008

На платформах, которые поддерживают стандарт POSIX.1-2008, хеш, возвращаемый `POSIX::localeconv()`, включает дополнительные поля национальной валюты, определённые этой версией POSIX стандарта. Это следующие

поля:

`int_n_cs_precedes`, `int_n_sep_by_space`, `int_n_sign_posn`, `int_p_cs_precedes`, `int_p_sep_by_space`, и `int_p_sign_posn`.

Лучшая эвристика на старых платформах для определения является ли локаль UTF-8

На платформах, которые не реализовали ни стандарт C99, ни стандарт POSIX 2001, определение, является ли локаль UTF-8 или нет, зависит от эвристики. Которая была улучшена в этом релизе.

Псевдонимы через ссылки

Переменным и подпрограммам теперь можно задавать псевдонимы, присваивая к ссылке:

```
\$c = \$d;
\&x = \&y;
```

Псевдонимы также могут быть назначены с использованием обратного слеша перед переменной итерации `foreach`; это, вероятно, одна из наиболее полезных идиом, которые предоставляет данная возможность:

```
foreach \%hash (@array_of_hash_refs) { ... }
```

Эта возможность экспериментальная и должна включаться с помощью `use feature 'refaliasing'`. Она выводит предупреждение, если не отключена категория предупреждений `experimental::refaliasing`.

Смотрите “Assigning to References” in perlref.

`prototype` без аргументов

`prototype()` без аргументов теперь подразумевает использование значения `$_`. [perl #123514].

Новый атрибут функции `:const`

Атрибут `const` может быть применён к анонимным функциям. Это приводит к тому, что функция немедленно исполняется всякий раз, когда создаётся (то есть когда обрабатывается выражение `sub`). Её значение запоминается и используется для создания новой функции-константы, которая и возвращается. Это возможность экспериментальная. Смотрите “Constant Functions” in perlsub.

`fileno` теперь работает и с дескрипторами директории

Если подобная возможность поддерживается в операционной системе, то встроенный `fileno` поддерживает работу с дескрипторами директорий, обрабатывая указанный файловый дескриптор, как файловый хендлер. На операционных системах без подобной поддержки `fileno` на дескрипторе директории продолжит возвращать неопределённое значение, как раньше, но также будет устанавливать `$!` для указания того, что операция не поддерживается.

На данный момент используется поле `dd_fd` в структуре ОС `DIR` или функция `dirfd(3)`, как определено в стандарте POSIX.1-2008.

Списочная форма записи для открытия канала реализована на Win32

Списочная форма записи для открытия канала:

```
1 open my $fh, "-|", "program", @arguments;
```

теперь реализована на Win32. Она имела те же ограничения, как и `system LIST` на Win32, т.к. Win32 API не поддерживает передачу аргументов программе в виде списка.

Присвоение списку с повторением

`(...)x ...` может быть теперь использован внутри списка, которому производится присвоение, пока левая сторона является lvalue. Это позволяет записывать `(undef,undef,$foo)= that_function()` как `((undef)x2, $foo)= that_function()`.

Улучшена обработка Infinity (бесконечность) и NaN (не-число)

Значения с плавающей запятой способны хранить специальные значения бесконечности, отрицательной бесконечности и NaN (не-число). Теперь мы быстрее распознаём и передаём результаты вычисления, а на выводе нормализуем их в строковые значения `Inf`, `-Inf`, and `NaN`.

Смотрите также улучшения в POSIX.

Улучшен разбор значений с плавающей запятой

Был улучшен разбор и вывод значений с плавающей запятой.

Как совершенно новая возможность, теперь поддерживается запись в виде шестнадцатеричных литералов (например, `0x1.23p-4`) и они могут быть выведены с помощью `printf "%a"`. Смотрите подробности в “Scalar value constructors” in perldata.

Упаковка бесконечности или не-числа в символ теперь фатальная ошибка

Раньше, при попытке упаковки бесконечности или не-числа в (знаковый) символ, Perl предупреждал об этом и подразумевал, что вы пытаетесь упаковать `0xFF`; если вы передавали это как аргумент к `chr`, то возвращался `U+FFFD`.

Но сейчас все подобные действия (`pack`, `chr` и `print '%c'`) приведут к фатальной ошибке.

Экспериментальное C API трассировки вызовов

Perl теперь поддерживает (через API C-уровня) получение трассировки вызовов уровня C (также как работают символьные отладчики, например, gdb).

Бэктрейс содержит трассировку стека фреймов C-вызовов с именами символов (именами функций), именами объектов (как, например, "perl"), а также, если возможно, расположение в исходном коде (файл:строка).

Поддерживаются платформы Linux и OS X (на некоторых *BSD также может работать, по крайне мере частично, но они ещё не тестировались).

Данная возможность должна быть включена с помощью `Configure -Dusecbacktrace`.

Смотрите подробнее в "C backtrace" in perlhacktips.

Безопасность

Perl теперь компилируется с опцией `-fstack-protector-strong`, если она доступна

Perl компилировался с опцией, предотвращающей переполнение буфера стека `-fstack-protector`, начиная с 5.10.1. Теперь Perl использует более новый вариант `-fstack-protector-strong`, если он доступен.

Модуль Safe допускал замещение внешних пакетов

Критическое исправление: внешние пакеты могли быть замещены. Safe был исправлен в версии 2.38.

Perl теперь всегда компилируется с опцией `-D_FORTIFY_SOURCE=2`, если она доступна

Опция ‘закалки кода’, называемая `_FORTIFY_SOURCE`, доступная в gcc 4.*, теперь всегда используется для компиляции Perl, если доступна.

Следует отметить, что это не обязательно значительное изменение, так как на многих платформах эта опция уже используется несколько лет: большинство Linux дистрибутивов (например, Fedora) используют эту опцию для Perl, а также в OS X она устанавливается уже много лет.

Несовместимые изменения

Сигнатура функции перемещена перед атрибутами

Экспериментальная возможность сигнатуры функции, появившаяся в 5.20, обрабатывала сигнатуру после атрибутов. В данном релизе, следуя отзывам пользователей этой экспериментальной возможности, позиция была перемещена так, чтобы сигнатура располагалась после имени подпрограммы (если есть) и перед списком атрибутов (если есть).

Прототипы `&` и `&` допускают только функции

Символ прототипа `&` теперь принимает только анонимные функции (`sub {...}`), начинающиеся с `\&`, или явный `undef`. Прежде они ошибочно допускали использование ссылок на массивы, хеши и списки. [perl #4539]. [perl #123062]. [perl #123062].

Дополнительно, прототип `\&` допускал вызов функции, тогда как сейчас он допускает только функции: `&foo` по-прежнему допустимый аргумент, в то время как `&foo()` и `foo()` больше нет. [perl #77860].

`use encoding` теперь лексическая

Прагма encoding теперь ограничена лексической областью видимости. Эта прагма устарела, но между тем, она может вредно воздействовать на другие модули, которые подключены в той же программе. Теперь это исправлено.

Срезы списков, возвращающие пустые списки

Срезы списков теперь возвращают пустой список, только если оригинальный список был пустым (или если нет индексов). Прежде срез списка возвращал пустой список если все индексы оказывались вне оригинального списка; теперь в этом случае возвращается список со значениями `undef`. [perl #114498].

`\N{}` с последовательностью пробелов теперь является фатальной ошибкой

Например, `\N{ОЧЕНЬ МНОГО ПРОБЕЛОВ}` или `\N{ЗАВЕРШАЮЩИЙ ПРОБЕЛ }`. Это являлось устаревшей конструкцией, начиная с v5.18.

`use UNIVERSAL '...'` теперь фатальная ошибка

Импортирование функций из `UNIVERSAL` является устаревшей операцией, начиная с v5.12 и теперь это фатальная ошибка. `use UNIVERSAL` без аргументов по-прежнему допускается.

В выражении `c_X_`, *X* должен быть печатным ASCII символом

В предыдущих релизах, невыполнение этого условия приводило к сообщению об устаревшей конструкции.

Разделение лексем `(?` и `(*` в регулярных выражениях теперь фатальная ошибка при компиляции.

Это является устаревшей конструкцией, начиная с v5.18.

`qr/foo/x` теперь игнорирует все Юникод-шаблоны пробельного символа

Модификатор регулярного выражения `/x` разрешал использование в шаблонах пробельных символов и комментариев (которые игнорировались) для удобства чтения. До текущего момента не все пробельные символы, которые стандарт Юникод обозначил для этих целей, поддерживались. Теперь также поддерживаются символы:

```
U+0085 NEXT LINE
U+200E LEFT-TO-RIGHT MARK
U+200F RIGHT-TO-LEFT MARK
U+2028 LINE SEPARATOR
U+2029 PARAGRAPH SEPARATOR
```

Использование этих символов вместе с `/x` вне классов символов, обособляемых скобками, и не предваряемых обратным слешем раньше, начиная с v5.18, выводило предупреждение об устаревшей конструкции. Теперь же они игнорируются.

Строки комментариев внутри `(?[ ])` теперь могут завершаться только символом `\n`

`(?[ ])` — это экспериментальная возможность, появившаяся в v5.18. Она действует, как если бы был активирован модификатор `/x`. Но есть отличие: строки комментариев (начинающиеся с символа #) экранируются любым символом, который совпадает с `\R`, что включает в себя все вертикальные пробелы, такие как разрыв страницы. Теперь, для согласованности, это поведение было изменено и для завершения строк комментариев вне `(?[ ])` используется `\n` (даже если экранирован), что является тем же символом, который завершает строку встроенного документа и формата.

Операторы `(?[...])` теперь следуют стандарту приоритетов в Perl

Это экспериментальная возможность, которая разрешает операции с наборами в регулярных выражениях. До этого оператор пересечения имел тот же приоритет, что и другие бинарные операции. Теперь он имеет более высокий приоритет. Это может привести к отличному поведению, чем ожидает существующий код (хотя документация всегда отмечала, что это изменение может произойти и рекомендовала полностью брать выражение в скобки). Смотрите в “Extended Bracketed Character Classes” in perlrecharclass.

Опускание % и @ перед именами хешей и массивов больше не допускается

Очень старые версии Perl допускали пропуск @ для имён массивов и % у имён хешей в некоторых местах. Это приводило к предупреждению об устаревшей конструкции, начиная с Perl 5.000, и теперь больше не разрешено.

Текст `$!` теперь на английском языке вне области действия `use locale`

Раньше текст, в отличии от всего другого, всегда базировался на текущей локали программы (также действовало на некоторых системах на `"$^E"`). Для программ, которые неготовы обрабатывать отличия в локалях, это может приводить к отображению мусорного текста. Гораздо лучше отображать текст, который может быть переведён через определённые утилиты, чем мусор, который труднее распознать.

Текст `$!` будет возвращён в UTF-8, если это приемлимо

Приведение к строковому значению `$!` и `$^E` установит флаг UTF-8, если текст является не-ASCII UTF-8. Это позволит программам, которые настроены на локале-зависимую работу, корректно выводит сообщения на естественном языке пользователя. Код, которому требуется оставить поведение 5.20 и более ранних версий, может приводить к строке в области действия прагм `use bytes` и `use locale ":messages"`. Внутри подобных областей никакие

Perl операции не будут затронуты локалью, кроме приведения к строке значений `$!` как `$^E`. Прагма `bytes` предотвратит установку флага UTF-8, также, как и в предыдущих Perl релизах. Это исправляет баг [perl #112208].

Удалена поддержка записи `?ШАБЛОН?` без явного оператора

Конструкция `m?ШАБЛОН?`, которая допускает только одно совпадение в регулярном выражении, раньше имела альтернативную форму записи, которая позволяла опускать явный оператор `m`. Подобное использование выдавало предупреждение об устаревшей конструкции, начиная с 5.14.0. Теперь это синтаксическая ошибка, освобождая символ вопроса для использования в новых операторах в будущем.

`defined(@array)` и `defined(%hash)` теперь фатальные ошибки

Эти конструкции устарели, начиная с v5.6.1, и выводили предупреждение об устаревшей конструкции, начиная с v5.16.

Использование хеша или массива как ссылки теперь фатальная ошибка

Например, `%foo->{"bar"}` теперь приводит к фатальной ошибке при компиляции. Это устарело, начиная с v5.8 и с тех пор выводило предупреждение об устаревшей конструкции.

Изменения в прототипе *

Символ * в прототипе подпрограммы раньше давал приоритет для bareword над большинством имён подпрограмм (но не над всеми). Это никогда не было последовательным и приводило к ошибочному поведению.

Теперь это было изменено и подпрограммы всегда получают приоритет над bareword, что приводит к согласованному поведению со встроенными функциями с аналогичными прототипами:

```
sub splat(*) { ... }
sub foo { ... }
splat(foo); # теперь всегда splat(foo())
splat(bar); # по–прежнему splat('bar'), как и раньше
close(foo); # close(foo())
close(bar); # close('bar')
```

Устаревшие конструкции

Установка `${^ENCODING}` в что-либо отличное от `undef`

Эта переменная позволяет писать Perl скрипты в кодировку отличную от ASCII или UTF-8. Однако она затрагивает все модули глобально, приводя к некорректным результатам и ошибкам сегментации. Новые скрипты должны писаться в UTF-8; старые должны быть сконвертированы в UTF-8, что может быть легко сделано с помощью утилиты piconv.

Использование нерафических символов в односимвольном имени переменной

Синтаксис для одно-буквенного имени переменной более снисходителен, чем для более длинных имён переменных, позволяя использовать односимвольные имена с символами пунктуации или даже невидимыми (нерафическими). Perl v5.20 объявил устаревшим использование контрольных символов ASCII для подобных имён. Теперь все нерафические символы, которые формально были разрешены теперь считаются устаревшими. Практический эффект от этого происходит, только когда не действует `use utf8`, и затрагивает только контрольные символы C1 (кодовые точки от 0x80 до 0xFF), NO-BREAK SPACE и SOFT HYPHEN.

Встраивание `sub (){ $var }` с заметным побочным эффектом

Во многих случаях Perl превращает `sub (){ $var }` во встраиваемую функцию-константу, захватывая значение `$var` во время интерпретации выражения `sub`. Это может нарушить замыкание в тех случаях, когда `$var`

в последствии модифицируется, поскольку подпрограмма не вернёт изменённое значение (это применимо только к анонимным подпрограммам с пустым прототипом (`sub ()`)).

Подобное использование теперь считается устаревшим в тех случаях, когда переменная модифицируется. Perl фиксирует подобные случаи и выводит предупреждение об устаревшей конструкции. Подобный код в будущем скорее всего изменит своё поведение и перестанет возвращать константу.

Если ваша переменная изменяется только в месте, где она определена, то Perl продолжит создавать встраиваемую подпрограмму без каких-либо предупреждений.

```
sub make_constant {
    my $var = shift;
    return sub () { $var }; # приемлимо
}

sub make_constant_deprecated {
    my $var;
    $var = shift;
    return sub () { $var }; # устаревшая конструкция
}

sub make_constant_deprecated2 {
    my $var = shift;
    log_that_value($var);   # может изменить $var
    return sub () { $var }; # устаревшая конструкция
}
```

Выше, во втором примере, достаточно сложно вычислить, что присвоение `$var` делается только один раз. Это происходит в месте отличном от декларации `my`, что достаточно для Perl, чтобы считать это подозрительным.

Это предупреждение об устаревшей конструкции происходит только для простых переменных в теле подпрограммы (блок `BEGIN` или оператор `use` внутри подпрограмм игнорируются, поскольку они не становятся частью тела подпрограммы). Для более сложных случаев, таких как `sub (){ do_something()if 0; $var }`, поведение изменилось и встраивания не происходит, если переменная изменяется где-либо ещё. Подобные случаи встречаются редко.

Использование нескольких подификаторов регулярных выражений /x

Теперь считается устаревшим использование подобных записей:

```
qr/foo/xx;
/(?xax:foo)/;
use re qw(/amxx);
```

Теперь x должен встречаться только один раз в строке, содержащей модификаторы регулярного выражения. Мы не склонны считать, что подобные случаи встречаются на CPAN. Это подготовка к будущему релизу Perl, который будет иметь /xx для разрешения пробелов для удобства чтения в классах символов заключённых в скобки (которые обособляются квадратными скобками: [...]).

Использование неразрывного пробела в псевдонимах символа \N{...} считается устаревшим

Этот неграфический символ практически неотличим от обычного пробела и поэтому не должен допускаться. Смотрите “CUSTOM ALIASES” in charnames.

Литерал { теперь должен экранироваться в шаблоне

Если вам требуется литерал левой фигурной скобки в шаблоне регулярного выражения, вы должны экранировать его с помощью предшествующего обратного слеша, или включив его в квадратные скобки, или используя \Q ; в противном случае будет выведено предупреждение об устаревшей конструкции. Впервые об этом было анонсировано в релизе v5.16; это позволит в будущем расширить синтаксис языка.

Не рекомендуется установливать все предупреждения фатальными

Документация для фатальных предупреждений отмечает, что использование use warnings FATAL => 'all' не рекомендуется и даёт общие пояснения по рискам использования фатальных предупреждений.

Улучшения в производительности

- Если метод или имя класса известны во время компиляции, то предварительно рассчитывается хеш для ускорения поиска методов во время исполнения. Также составные имена методов, такие как `SUPER::new`, разбираются во время компиляции, что снимает необходимость их обработки во время исполнения.
- Поиск элемента в массивах и хешах (особенно во вложенных), при котором используется константа или простая переменная в качестве индекса/ключа, теперь работает гораздо быстрее. Смотрите подробнее в “Internal Changes”.
- `(...)x1`, `("constant")x0` и `($scalar)x0` теперь оптимизированы в списочном контексте. Если правый аргумент константа 1, оператор повтора исчезает. Если правый аргумент это константа 0, то всё выражение оптимизируется в пустой список, если левый аргумент это простой скаляр или константа (таким образом, `(foo())x0` не оптимизируется).
- Присвоение `substr` в конце подпрограммы (или как аргумент `return`) теперь оптимизируется в 4-х аргументный `substr`. Раньше это происходило только в пустом контексте.
- В `"\L..."`, `"\Q..."` и подобных случаях дополнительная операция приведения к строке теперь оптимизируется, делая их такими же быстрыми, как `lcfirst`, `quotemeta`, и так далее.
- Присвоение пустому списку теперь иногда работает быстрее. В частности, теперь никогда не вызывается `FETCH` на связанные аргументы на правой стороне, в то время как раньше это иногда происходило.
- Производительность `length` увеличена на 20% если применяется к немагической, несвязанной строке, или, если находится в области видимости прагмы `use bytes`, или если строка не использует внутри UTF-8.
- На большинстве сборок perl с 64-битными целыми использование памяти для немагических, несвязанных скалярах, содержащих только значение с плавающей точкой было уменьшено от 8 до 32 байт в зависимости от ОС.
- В выражении `@array = split` присвоение может быть оптимизировано так, что `split` будет записывать непосредственно в массив. Эта оптимизация раньше происходила только для массивов пакета, отличных от `@_` и только иногда. Теперь эта оптимизация происходит практически всегда.
- `join` теперь может выполнять свёртку констант. Например, `join`

`"-", "a", "b"` конвертируется во время компиляции в `"a-b"`. Более того, `join` со скаляром или константой в качестве разделителя с одноэлементным списком упрощается до операции приведения к строке, а разделитель даже не обрабатывается.

- `qq(@array)` реализован с использованием двух операций: операция приведения к строке и операция объединения. Если `qq` не содержит ничего кроме одного массива, то операция приведения к строке опускается.
- `our $var` и `our($s,@a,%h)` в пустом контексте больше не вычисляются во время выполнения. Даже вся последовательность выражений `our $foo;` будет просто пропущена. Тоже самое относится к переменным `state`.
- Множество внутренних функция было переработано, чтобы улучшить производительность и снизить их потребление памяти. [perl #121436] [perl #121906] [perl #121969]
- Файловые тесты `-T` и `-B` завершаться раньше, если будет выявлено, что файл пустой. [perl #121489]
- Поиск в хеше по ключу-константе происходит быстрее.
- Подпрограммы с пустыми прототипами и телом, содержащим только `undef` теперь могут встраиваться. [perl #122728]
- Подпрограммы в пакетах больше не требуется сохранять в тайпглобах: объявление подпрограммы теперь помещает простую ссылку на подпрограмму непосредственно в стэш, если это возможно, экономя память. Тайпглоб по-прежнему номинально существует, попытка доступа к нему приведёт к модернизации стэша в тайпглоб (это просто детали внутренней реализации). Эта оптимизация на данный момент не применяется для XSUB или экспортированных подпрограмм, и вызов методов устранит её, поскольку они сохраняют кэш в тайпглобах. [perl #120441]
- Функции `utf8::native_to_unicode()` и `utf8::unicode_to_native()` (смотрите utf8) теперь оптимизируются на ASCII платформах. Теперь нет ни малейшего снижения производительности при написании кода, переносимого между ASCII и EBCDIC платформ.
- Win32 Perl использует меньше на 8 Кбайт памяти на каждый процесс, чем раньше, поскольку некоторые данные теперь отображаются в память с диска и разделяются между процессами одного исполняемого файла perl.

Модули и прагмы

Обновлённые модули и прагмы

Многие библиотеки, распространяемые вместе с perl были обновлены, со времени выхода v5.20.0. Для получения полного списка изменений можно выполнить:

```
corelist --diff 5.20.0 5.22.0
```

Вы также можете подставить нужную вам версию вместо 5.20.0.

Некоторые заметные изменения:

- Archive::Tar был обновлён до версии 2.04.

  Тесты теперь могут выполняться параллельно.

- attributes был обновлён до версии 0.27.

  Использование `memEQs` в XS было исправлено. [perl #122701]

  Предотвращено чтение за пределами границы буфера. [perl #122629]

- B был обновлён до версии 1.58.

  Он предоставляет новую функцию `B::safename`, основанную на существующей `B::GV->SAFENAME`, которая конвертирует `\cOPEN` в `^OPEN`.

  Обнулённые COP'ы теперь имеют класс `B::COP`, вместо `B::OP`.

  Объекты `B::REGEXP` теперь предоставляют метод `qr_anoncv` для доступа к неявным CV, ассоциированным с `qr//`, содержащими кодовые блоки, а метод `compflags`, который возвращает соответствующие флаги, относящиеся к операции `qr//blahblah`.

  `B::PMOP` теперь предоставляет метод `pmregexp`, возвращающий объект `B::REGEXP`. Были созданы два новых класса: `B::PADNAME` и `B::PADNAMELIST`.

  Была исправлена ошибка, когда после создания треда или псевдофорка, специальные или бессмертные SV в потомке треда/псевдофорка не имеют корректного класса `B::SPECIAL`. Были добавлены методы PADLIST `id` и `outid`.

- B::Concise был обновлён до версии 0.996.

  Нулевые операции, которые являются частью цепи исполнения теперь получают соответствующие порядковые номера.

  Закрытые флаги для нулевых операций теперь выводятся с мнемоникой, как и у ненулевых операций.

- B::Deparse был обновлён до версии 1.35.

  Теперь он корректно депарсит `+sub : attr { ... }` с начала выражения. Без начального `+`, `sub` стал бы меткой оператора.

  Блоки `BEGIN` теперь появляются в нужных местах в большинстве случаев, но это изменение, к сожалению, приводит к регрессии, при которой блоки `BEGIN`, находящиеся прямо перед концом блока могут оказаться после него.

  `B::Deparse` больше не помещает некорректные `local` тут и там, как например, в `LIST = tr/a//d`. [perl #119815]

  Соседние операторы `use` больше не вкладываются, если один из них содержит блок `do`. [perl #115066]

  Массивы в скобках в списке, передаваемом с `\` теперь корректно депарсятся со скобками (то есть, `\(@a, (@b), @c)` теперь сохранит скобки вокруг @b), предотвращая от выравнивания массивов в списке по ссылке. Раньше это работало только для одного массива: `\(@a)`.

  `local our` теперь корректно депарсится, с включённым `our`.

  `for($foo; !$bar; $baz){...}` депарсился без `!` (или `not`). Это было исправлено.

  Базовые ключевые слова, которые конфликтуют с лексическим подпрограммами теперь депарсятся с префиксом `CORE::`.

  `foreach state $x (...){...}` теперь корректно депарсится со `state`, а не с `my`.

  `our @array = split(...)` теперь депарсится корректно с `our` в тех случаях, когда выражение оптимизируется.

  Теперь депарсятся корректно `our(_LIST_)` и типизированное лексическое (`my Dog $spot`).

  `$#_` теперь депарсится корректно, вместо `$#{_}`. [perl #123947]

  Блоки BEGIN в конце замыкающей области видимости теперь депарсятся в нужном месте. [perl #77452]

Блоки BEGIN иногда депарсились, как __ANON__, но теперь всегда называются BEGIN.

Лексические подпрограммы теперь полностью депарсятся. [perl #116553]

`Anything =~ y///r` с `/r` больше не пропускают левый операндю

Дерево операций, которые формируют кодовый блок regexp теперь депарсятся по-настоящему. Раньше использовалась оригинальная строка, которая создавала регулярное выражение. Это приводило к проблемам с `qr/(?{<<heredoc})/` и многострочным блокам кода, которые депарсились некорректно. [perl #123217] [perl #115256]

`$;` в конце оператора больше не теряет точку с запятой. [perl #123357]

Некоторые случаи объявления подпрограмм, сохраняемые в стеше в краткой форме пропускались.

Не-ASCII символы теперь последовательно экранируются в строках, без пропусков. (По-прежнему есть проблемы с регулярными выражениями и идентификаторами, которые ещё не исправлены.)

Когда прототип вызова подпрограммы депарсится с `&` (например, с опцией **-P**), теперь добавляется `scalar`, когда это требуется, чтобы принудительно установить скалярный контекст, подразумеваемый прототипом.

`require(foo())`, `do(foo())`, `goto(foo())` и схожие конструкции с контролем циклов теперь корректно депарсятся. Внешние скобки не опциональные.

Пробельные символы больше не экранируются в регулярных выражениях, из-за того, что они ошибочно были экранированы внутри секции `(?x:...)`.

`sub foo { foo()}` теперь депарсится с этими обязательными скобками.

`/@array/` теперь депарсится как регулярное выражение, а не просто `@array`.

`/@{-}/`, `/@{+}/` и `$#{1}` теперь депарсятся с фигурными скобками, которые обязательны в этих случаях.

При депарсинге связок фич, `B::Deparse` выводил сначала `no feature;`, вместо вывода `no feature ':all';`. Это было исправлено.

`chdir FH` теперь депарсится без кавычек.

`\my @a` теперь депарсится без скобок. (Скобки выравняют массив).

`system` и `exec` с последующим блоком теперь депарсятся корректно. Раньше появлялся ошибочный `do` перед блоком.

`use constant QR => qr/.../flags` с последующим `"" =~ QR` больше не появляются без флагов.

Депарсинг `BEGIN { undef &foo }` с включённой опцией **-w** начало выводить предупреждение о неинициализированности в Perl 5.14. Теперь это было исправлено.

Вызов депарсинга к подпрограммам с прототипом `(;+)` приводил к бесконечному циклу. Прототипы `(;$)`, `(_)` и `(;_)` получали неправильный порядок приоритетов, приводя к депарсингу `foo($a<$b)` без скобок.

Deparse теперь предоставляет определённую state подпрограмму во внутренних подпрограммах.

- Был добавлен B::Op_private

  B::Op_private предоставляет детальную информацию о флагах, используемых в поле `op_private` опкодов Perl.

- bigint, bignum, bigrat были обновлены до версии 0.39.

  Задокументировано в секции CAVEATS (предостережение), что использование строк как чисел не всегда приводит к перегрузке больших чисел, а также информацию о том как вызвать её. [rt.perl.org #123064]

- Carp был обновлён до версии 1.36.

  `Carp::Heavy` теперь игнорирует несовпадение версий с Carp, если Carp новее, чем 1.12, так как начинка `Carp::Heavy` была внесена в Carp в этой версии. [perl #121574]

  Carp теперь лучше поддерживает не-ASCII платформы.

  Исправлена ошибка на единицу для Perl < 5.14.

- constant был обновлён до версии 1.33.

  Он теперь поддерживает полные имена констант, позволяя определять константы в пакетах отличных от пакеты вызывающего кода.

- CPAN был обновлён до версии 2.11.

  Была добавлена поддержка `Cwd::getdcwd()` и внесено обходное решение для ошибочного поведения, которое наблюдалось на Strawberry Perl 5.20.1.

Исправлена ошибка с `chdir()` после сборки зависимостей.

Введена экспериментальная поддержка для плагинов/хуков.

Интегрированы исходники `App::Cpan`.

Не проверяется рекурсия для опциональных зависимостей.

Проверка на вменяемость, что `META.yml` содержит хеш. [cpan #95271]

- CPAN::Meta::Requirements был обновлён до версии 2.132.

  Обход ограничения в `version::vpp` для определения магии v-строк и добавление поддержки грядущему бутстрапу `version.pm` в ExtUtils::MakeMaker для Perl старее чем 5.10.0.

- Data::Dumper был обновлён до версии 2.158.

  Исправлена уязвимость CVE-2014-4330 путём добавления переменной конфигурации или опции для ограничения рекурсии, при выводе глубоко-вложенных структур.

  Изменения для исправления ошибок найденных сканером безопасности Coverity. XS выводит некорректно сохранённое имя кодовой ссылки, хранимое в GLOB. [perl #122070]

- DynaLoader был обновлён до версии 1.32.

  Удалена глобальная переменная `dl_nonlazy`, если не используется в Dynaloader. [perl #122926]

- Encode был обновлён до версии 2.72.

  `piconv` теперь лучше отлавливает ошибки, когда имя кодировки не существует и исправлен разлом сборки при обновлении Encode в perl-5.8.2 и более ранних версий.

  Теперь работает сборка в C++ режиме на Windows.

- Errno был обновлён до версии 1.23.

  Добавлена опция `-P` для препроцессора в GCC 5. GCC добавил дополнительную строковую директиву, которая сломала разбор определений кодов ошибок. [rt.perl.org #123784]

- experimental был обновлён до версии 0.013.

  Зашиты возможности для Perl старше 5.15.7.

- ExtUtils::CBuilder был обновлён до версии 0.280221.

  Исправлена регрессия на Android. [perl #122675]

- ExtUtils::Manifest был обновлён до версии 1.70.

  Исправлена ошибка с обработкой файловых имён с кавычками в `maniread()` и улучшен `manifind()`, теперь следующий по ссылкам. [perl #122415]

- ExtUtils::ParseXS был обновлён до версии 3.28.

  Объявляет `file` неиспользуемым только если его действительно определили. Улучшена генерация кода `RETVAL`, чтобы избежать повторения ссылок на `ST(0)`. [perl #123278] Расширена и задокументирована typemap оптимизация `/OBJ$/` в `/REF$/` для метода `DESTROY`. [perl #123418]

- Fcntl был обновлён до версии 1.13.

  Была добавлена поддержка работы с размером буфера канала в Linux в командах `fcntl()`.

- File::Find был обновлён до версии 1.29.

  `find()` и `finddepth()` будут предупреждать, если передана некорректная или опция с опечаткой.

- File::Glob был обновлён до версии 1.24.

  `SvIV()` избегает расширения для вызова `get_sv()` три раза в нескольких местах. [perl #123606]

- HTTP::Tiny был обновлён до версии 0.054.

  `keep_alive` теперь безопасен для вызова fork или запуска новых тредов.

- IO был обновлён до версии 1.35.

  XS реализация была исправлена для поддержки работы на старых Perl.

- IO::Socket был обновлён до версии 1.38.

  Задокументированы ограничения метода `connected()`. [perl #123096]

- IO::Socket::IP был обновлён до версии 0.37.

  Более лучшее исправление для субкласса `connect()`. [cpan #95983] [cpan #97050]

  Реализован таймаут для `connect()`. [cpan #92075]

- Коллекция модулей libnet была обновлена до версии 3.05.

  Поддержка для IPv6 и SSL для `Net::FTP`, `Net::NNTP`, `Net::POP3` и `Net::SMTP`. Улучшения аутентификации в `Net::SMTP`.

- Locale::Codes был обновлён до версии 3.34.

  Исправлен баг в скриптах, используемых для извлечения данных из электронных таблиц, которые препятствовали поиску SHP коду валюты. [cpan #94229]

  Были добавлены новые коды.

- Math::BigInt был обновлён до версии 1.9997.

  Синхронизированы изменения в POD вместе со CPAN выпуском. `Math::BigFloat->blog(x)` иногда возвращает `blog(2*x)`, когда точность больше 70 цифр. Результат `Math::BigFloat->bdiv()` в списочном контексте теперь удовлетворяет `x = quotient * divisor + remainder`.

  Исправлена обработка субклассов. [cpan #96254] [cpan #96329]

- Module::Metadata был обновлён до версии 1.000026.

  Поддержка инсталляции на старых perl вместе с версиями ExtUtils::MakeMaker более ранних чем 6.63_03

- overload был обновлён до версии 1.26.

  Ненужная проверка `ref $sub` была удалена.

- Коллекция модулей PathTools была обновлена до версии 3.56.

  Избегаются предупреждения компилятора **gcc** теперь при сборке XS.

  `//` в начале строки не заменяется на `/` на Cygwin. [perl #122635]

- perl5db.pl был обновлён до версии 1.49.

  Отладчик вызывал нарушения предположения. [perl #124127]

  `fork()` в отладчике в `tmux` теперь создаёт новое окно для созданного процесса. [perl #121333]

  Отладчик теперь сохраняет текущий рабочий каталог при запуске и восстанавливает его, когда вы перезапускаете вашу программу с помощью `R` или `rerun`. [perl #121509]

- PerlIO::scalar был обновлён до версии 0.22.

Чтение из позиции за пределами скаляра теперь корректно возвращает конец файла. [perl #123443]

Смещение в отрицательную позицию по-прежнему приводит к ошибке, но больше не оставляет позицию на отрицательном смещении.

`eof()` на `PerlIO::scalar` дескрипторе теперь корректно возвращает истину, когда файловая позиция превышает планку в 2 Гбайта на 32-битных системах.

Попытка записи в файловую позицию недопустимую для платформы теперь сразу приводит к ошибке, а не падает после выделения 4 Гбайт.

- Pod::Perldoc был обновлён до версии 3.25.

  Файловые дескрипторы открытые для чтения или записи теперь имеют установленный `:encoding(UTF-8)`. [cpan #98019]

- POSIX был обновлён до версии 1.53.

  Были добавлены математические функции и константы C99 (например, `acosh`, `isinf`, `isnan`, `round`, `trunc`; `M_E`, `M_SQRT2`, `M_PI`).

  `POSIX::tmpnam()` теперь выводит предупреждение об устаревшей конструкции. [perl #122005]

- Safe был обновлён до версии 2.39.

  `reval` не распространял корректно пустой контекст.

- Scalar-List-Utils был обновлён до версии 1.41.

  Был добавлен новый модуль Sub::Util, содержащий функции, относящиеся к кодовым ссылкам, включая `subname` (по мотивам `Sub::Identity`) и `set_subname` (скопированная и переименованная из `Sub::Name`). Использование `GetMagic` в `List::Util::reduce()` также было исправлено. [cpan #63211]

- SDBM_File был обновлён до версии 1.13.

  Упрощён процесс сборки. [perl #123413]

- Time::Piece был обновлён до версии 1.29.

  При печати отрицательных `Time::Seconds`, “минус” больше не теряется.

- Unicode::Collate был обновлён до версии 1.12.

Улучшенные несмежные сокращения версии 0.67 сделаны недействительными по умолчанию и поддерживаются как параметр `long_contraction`.

- Unicode::Normalize был обновлён до версии 1.18.

  Реализация XSUB была удалена, предпочтение отдано реализации на чистом Perl.

- Unicode::UCD был обновлён до версии 0.61.

  Была добавлена новая функция property_values() возвращающая возможные значения для заданного свойства.

  Была добавлена новая функция charprop(), возвращающая значение данного свойства по заданной кодовой точке.

  Была добавлена новая функция charprops_all(), возвращающая значения всех свойств Юникода для заданной кодовой точки.

  Исправлен ошибка, когда propaliases() возвращает корректное короткое и длинные имена для Perl расширений, в которых они некорректны.

  Исправлена ошибка, когда prop_value_aliases() возвращает `undef` вместо некорректного результата для свойств, которые являются Perl расширениями.

  Модуль теперь работает на EBCDIC платформах.

- utf8 был обновлён до версии 1.17

  Было исправлено несоответствие между документацией и кодом в `utf8::downgrade()` в соответствии с документацией. Опциональный второй аргумент теперь корректно интерпретируется как логическое значение (семантика истина/ложь), а не как целое число.

- version был обновлён до версии 0.9909.

  Многочисленные изменения. Смотрите файл `Changes` в CPAN дистрибутиве.

- Win32 был обновлён до версии 0.51.

  `GetOSName()` теперь поддерживает Windows 8.1 и сборку в режиме C++.

- Win32API::File был обновлён до версии 0.1202

  Работает сборка в режиме C++.

- XSLoader был обновлён до версии 0.20.

  В XSLoader разрешена загрузка модулей их другого пространства имён. [perl #122455]

Удалённые модули и прагмы

Следующие модули (и связанные модули) были удалены из базового дистрибутива perl:

- CGI
- Module::Build

Документация

Новая документация

perlunicook Этот документ, созданный Томом Христиансеном, содержит примеры работы с Юникодом в Perl.

Изменения в существующей документации

perlaix

- Было добавлено замечание о long double.

perlapi

- Замечание, что `SvSetSV` не устанавливает магию.
- `sv_usepvn_flags` - исправление документации с упоминанием использования `Newx` вместо `malloc`.

[perl #121869]

- Уточнение, где `NUL` может быть встроен или требуется для завершения строки.
- Некоторая документация, которая раньше отсутствовала из-за ошибок форматирования теперь включена.
- Элементы теперь организованы по группам, а не файлам, в которых они находятся.
- Сортировка по алфавиту элементов теперь выполняется согласованно (автоматически POD-генератором), чтобы быстрее найти данные при поиске.

perldata

- Синтаксис одно-буквенных имён переменных был обновлён и более детально объяснён.
- Описаны шестнадцатеричные числа с плавающей запятой, а также infinity и NaN.

perlebcdic

- Это документ был значительно обновлён в свете последних улучшений поддержки платформы EBCDIC.

perlfilter

- Добавлена секция LIMITATIONS.

perlfunc

- Упоминается, что `study()` на данный момент пустая операция.

- Вызов `delete` или `exists` на значения массива теперь описана как “решительно неприемлемым” вместо “устаревшим”.
- Улучшена документация `our`.
- Для `-l` теперь упоминается, что он возвращает ложь, если символьные ссылки не поддерживаются файловой системой.

  [perl #121523]
- Замечание, что `exec LIST` и `system LIST` могут откатываться в вызов командной строки на Win32. Только неявный объектный синтаксис `exec PROGRAM LIST` и `system PROGRAM LIST` гарантировано позволит избежать запуск командной оболочки.

  Это также было добавлено в perlport.

  [perl #122046]

perlguts

- Пример с OOK был обновлён с учётом изменений COW и изменений в хранении смещения.
- Добавлены подробности о символах C-уровня и libperl.t.
- Добавлена информация о работе с Юникодом.
- Добавлена информация о поддержке EBCDIC.

perlhack

- Была добавлена информация о запуске на платформах с набором не-ASCII символов.
- Было добавлено замечание о тестировании производительности

perlhacktips

- Была добавлена документация, иллюстрирующая риски самонадеянности, что нет изменений в содержимом статической памяти, на которую

указывает возвращаемое значение Perl-обёртки для библиотечных C-функций.

- Замены для `tmpfile`, `atoi`, `strtol` и `strtoul` теперь рекомендуются.
- Обновлена документация для цели `make test.valgrind`.

  [perl #121431]
- Дана информация о написании файлов тестов переносимых на не-ASCII платформы.
- Была добавлено заметка о том, как получить трассировку стека вызовов языка C.

perlhpux

- Замечание о том, что сообщение “Повторное объявление”sendpath” с отличающимся описанием класса хранения” безобидно.

perllocale

- Обновлено данными об улучшениях, появившихся в v5.22, вместе с некоторыми пояснениями.

perlmodstyle

- Вместо указания на список модулей, мы теперь указываем на PrePAN.

perlop

- Обновлено данными об улучшениях, появившихся в v5.22, вместе с некоторыми пояснениями.

perlpodspec

- Спецификация языка pod изменена, кодировка по умолчанию для pod, которые не в UTF-8 (или иначе обозначенные), теперь CP1252 вместо ISO 8859-1 (Latin1).

perlpolicy

- Теперь у нас есть правила поведения для почтового списка рассылки *p5p*, задокументированные в “STANDARDS OF CONDUCT” in perlpolicy.
- Указаны условия, при которых экспериментальные возможности перестают быть экспериментальными.
- Были сделаны уточнения по типам изменений, которые допустимы в релизах сопровождения.

perlport

- Устаревшая специфичная для VMS информация была исправлена и/или упрощена.
- Были добавлены замечания о EBCDIC.

perlre

- Было уточнено описание модификатора `/x`, чтобы отметить, что комментарии не могут быть продолжены на следующей строке путём экранирования их; кроме того теперь есть список всех символов, которые рассматриваются этим модификатором как пробельные.
- Описан новые модификатор `/n`.
- Было добавлено заметка о том, как сделать классы символов, обособленные скобками, переносимыми на не-ASCII машины.

perlrebackslash

- Добавлена документация по `\b{sb}`, `\b{wb}`, `\b{gcb}` и `\b{g}`.

perlrecharclass

- Были добавлены пояснения в “Character Ranges” in perlrecharclass об эффекте `[A–Z]`, `[a–z]`, `[0–9]` и любых поддиапазонов в обособленных скобками классах символов в регулярных выражениях, который гарантирует, что произойдёт совпадение только с тем, что ожидает неподготовленный англоговорящий пользователь, даже на платформах (таких как EBCDIC), где требуется специальные манипуляции для достижения этой цели.
- Документация обособленных скобками классов символов была расширена, осветив улучшения в `qr/[\N{named sequence}]/` (смотрите “Selected Bug Fixes”).

perlref

- Была добавлена новая секция Присвоение ссылкам

perlsec

- Были добавлены комментарии об алгоритмической сложности и связанных хешах.

perlsyn

- Была исправлена неоднозначность в документации, касаемая оператора `....` [perl #122661]
- Пустое условие в `for` и `while` теперь задокументировано в perlsyn.

perlunicode

- Документация была значительно переработана для соответствия с текущей поддержкой Юникода и, чтобы сделать её более читабельной. Заметно, что Юникод 7.0 изменил поведение в отношении не-символов. Perl сохранил старый способ работы по причинам обратной совместимости. Смотрите “Noncharacter code points” in perlunicode.

perluniintro

- Был обновлён совет о том, как быть уверенным, что строки и шаблоны регулярных выражений интерпретируются как Юникод.

perlvar

- `$]` больше не считается устаревшей. Вместо этого была добавлено обсуждение о преимуществах и недостатках использования её вместо `$^V`.
- `${^ENCODING}` теперь считается устаревшей.
- Была уточнена запись о `%^H` для указания, что она может хранить только простые значения.

perlvms

- Устаревший и/или некорректный материал был удалён.
- Обновлена документация об окружении и взаимодействии с оболочкой на VMS.

perlxs

- Добавлено обсуждение о проблемах с локалью в XS коде.

Диагностика

Следующие добавления и изменения были сделаны в диагностическом выводе, включая предупреждения и фатальные сообщения об ошибках. Для полного перечная диагностических сообщений смотрите perldiag.

Новая диагностика

Новые ошибки

- Bad symbol for scalar

  (P) Внутренний запрос попросил добавить скалярную запись к чему-то, что не является записью в таблице символов.

- [Can't use a hash as a reference](https://metacpan.org/pod/perldiag#Can't use a hash as a reference)

  (F) Вы попытались использовать хеш как ссылку, например, `%foo ->{"bar"}` или `%$ref->{"hello"}`. Версии perl <= 5.6.1 разрешали подобный синтаксис, хотя этого не следовало делать.

- [Can't use an array as a reference](https://metacpan.org/pod/perldiag#Can't use an array as a reference)

  (F) Вы попытались использовать массив как ссылку, например, `@foo ->[23]` или `@$ref->[99]`. Версии perl <= 5.6.1 разрешали подобный синтаксис, хотя этого не следовало делать.

- [Can't use 'defined(@array)' (Maybe you should just omit the defined()?)](https://metac
  use 'defined(@array)' (Maybe you should just omit the defined()?))

  (F) `defined()` бесполезен для массивов поскольку он проверяет на неопределённость значение *скаляра*. Если вы хотите проверить является ли массив пустым просто используйте, например, `if ( @array){ # не пустой }`.

- [Can't use 'defined(%hash)' (Maybe you should just omit the defined()?)](https://metacp
  use 'defined(%hash)' (Maybe you should just omit the defined()?))

(F) `defined()` не совсем то, что нужно для хешей.

Хотя `defined %hash` является ложью на простом неиспользованном хеше, это становится истиной в некоторых неочевидных случаях, включая использования итераторов, слабых ссылок, имён стэшей и даже остаётся истиной после `undef %hash`. Это делает `defined %hash` довольно бесполезным на практике, поэтому сейчас он вызывает фатальную ошибку.

Если вам требуется проверка на то, что хеш не пустой, просто поставьте его в логический контекст (смотрите “Scalar values” in perldata):

```
if (%hash) {
   # not empty
}
```

Если вы использовали `defined %Foo::Bar::QUUX` для проверки, что существует подобная переменная пакета, то это никогда не было надёжно и не является правильным путём запроса о возможностях пакета или проверки того, что он загружен и так далее.

- Cannot chr %f

  (F) Вы передали некорректное число (например, бесконечность или не-число) в `chr`.

- Cannot compress %f in pack

  (F) Вы пытались конвертировать бесконечность или не-число в беззнаковый символ, что является бессмысленным.

- Cannot pack %f with ‘%c’

  (F) Вы пытались конвертировать бесконечность или не-число в символ, что является бессмысленным.

- Cannot print %f with ‘%c’

  (F) Вы попытались напечатать бесконечность или не-число как символ (`%c`), что не имеет смысла. Возможно вы имели ввиду `'%s'` или просто привести его к строковому виду.

- charnames alias definitions may not contain a sequence of multiple spaces

(F) Вы определили имя символа, который имеет несколько пробельных символов подряд. Замените их на один пробел. Обычно эти имена определены в аргументе импорта `:alias` для `use charnames`, но они также могут быть определены переводчиком и располагаться в `$^H{charnames}`. Смотрите “CUSTOM ALIASES” in charnames.

- charnames alias definitions may not contain trailing white-space

  (F) Вы определили имя символа, который завершается на пробельный символ. Удалите завершающий пробел. Обычно эти имена определены в аргументе импорта `:alias` для `use charnames`, но они также могут быть определены переводчиком и располагаться в `$^H{charnames}`. Смотрите “CUSTOM ALIASES” in charnames.

- :const is not permitted on named subroutines

  (F) Атрибут `const` используется, чтобы анонимная подпрограмма запускалась и захватывала значение в тоже время, в которое оно клонируется. Именованная подпрограмма не может быть клонирована подобным образом, поэтому атрибут для неё не имеет смысла.

- Hexadecimal float: internal error

  (F) Произошло нечто ужасное в обработке шестнадцатеричного значения с плавающей запятой.

- Hexadecimal float: unsupported long double format

  (F) Вы сконфигурировали Perl для использования long double, но внутренняя реализация формата long double неизвестна, поэтому вывод шестнадцатеричного значения с плавающей запятой невозможен.

- Illegal suidscript

  (F) Запуск скрипта под suidperl уже давно нелегален.

- [In ‘(?...)’, the ‘(’ and ‘?’ must be adjacent in regex; marked by <– HERE in m/%s/](https://metacpan.org/pod/perldiag#In ‘(?...)’, the ‘(’ and ‘?’ must be adjacent in regex; marked by <– HERE in m/%s/)

(F) Двухсимвольная последовательность `"(?"` в этом контексте в шаблоне регулярного выражения должно быть неделимой лексемой и ничего не может находиться между `"("` и `"?"`, но вы разделили их.

- [In '(*VERB...)', the '(' and "* must be adjacent in regex; marked by <– HERE in m/%s/](https://metacpan.org/pod/perldiag#In '(*VERB...)', the '(' and "* must be adjacent in regex; marked by <– HERE in m/%s/)

  (F) Двухсимвольная последовательность `"(*"` в этом контексте в шаблоне регулярного выражения должно быть неделимой лексемой и ничего не может находиться между `"("` и `"*"`, но вы разделили их.

- Invalid quantifier in {,} in regex; marked by <– HERE in m/%s/

  (F) Шаблон выглядит как квантор {min,max}, но min или max не могут быть распарсены как валидные числа: либо они имеют нули, стоящие в начале, либо представляют слишком большие числа, чтобы можно работать с ними. <– HERE показывает где в регулярном выражении найдена эта проблема. Смотрите perlre.

- ['%s' is an unknown bound type in regex](https://metacpan.org/pod/perldiag#'%s' is an unknown bound type in regex; marked by <– HERE in m/%s/)

  Вы использовали `\b{...}` или `\B{...}`, но `...` неизвестен Perl. Текущие допустимые варианты описаны в "_,␣ -, ²" in perlrebackslash.

- Missing or undefined argument to require

  (F) Вы пытались вызвать `require` без аргумента или с неопределённым значением в качестве аргумента. `require` ожидает имя пакета или указание на файл в качестве аргумента. Смотрите "require" in perlfunc.

  Раньше `require` без аргумента или с `undef` предупреждала о пустом имени файла.

Новые предупреждения

- `C` is deprecated in regex

  (D deprecated) Класс символов `/\C/` был объявлен устаревшим в v5.20 и теперь выводит предупреждение. Предполагается, что это станет ошибкой в v5.24. Этот класс символов совпадает с единичным байтом, даже если он появляется внутри многобайтовой последовательности, что нарушает инкапсуляцию и может повреждать UTF-8 строки.

- ["%s" is more clearly written simply as "%s" in regex; marked by <– HERE in m/%s/](https://metacpan.org/pod/perldiag#"%s" is more clearly written simply as "%s" in regex; marked by <– HERE in m/%s/)

  (W regexp) (только при действии `use re 'strict'` или внутри `(?[...])`).

  Вы указали символ, который имеет указанный более простой способ записи и который переносим между платформами, работающими с разными наборами символов.

- [Argument "%s" treated as 0 in increment (++)](https://metacpan.org/pod/perldiag#Arg
  "%s" treated as 0 in increment (++))

  (W numeric) Указанная строка была передана как аргумент оператору `++`, который ожидает либо число, либо строку, совпадающую с `/^[a–zA–Z]*[0–9]*\z/`. Смотрите подробности в "Auto-increment and Auto-decrement" in perlop.

- Both or neither range ends should be Unicode in regex; marked by <– HERE in m/%s/

  (W regexp) (только при действии `use re 'strict'` или внутри `(?[...])`).

  В классе символов, обособленных скобками, в шаблоне регулярного выражения у вас используется диапазон, который имеет одну из границ, указанной с использованием `\N{}`, а другая граница указана с помощью непереносимого механизма. Perl рассматривает этот диапазон, как Юникод-диапазон, в котором все символы являются Юникод-символами и которые могут иметь различные кодовые значения на некоторых платформах, на которых работает Perl. Например, `[\N{U+06}–\x08]` рассматривается как если бы написали `[\N{U+06}–\N{U+08}]`, который совпадает с символами, чьи кодовые значения в Юникоде соответственно 6, 7 и 8. Но `\x08` может означать что-то другое, поэтому выводится предупреждение.

- [Can't do %s("%s") on non-UTF-8 locale; resolved to "%s".](https://metacpan.org/pod/per
  do %s("%s") on non-UTF-8 locale; resolved to "%s".)

  (W locale) Вы 1) работаете при действии "`use locale`"; 2) текущая локаль не является UTF-8; 3) вы пытались сделать операцию изменения регистра на указанном Юникод символе; и 4) результат этой операции приведёт к смешиванию правил Юникода и локали, которые вероятно конфликтуют.

- :const is experimental

  (S experimental::const_attr) Атрибут `const` является экспериментальным. Если вы хотите использовать эту возможность, подавите предупреждения с помощью `no warnings 'experimental::const_attr'`, но имейте ввиду, что вы принимаете на себя риск, что ваш код может сломаться в будущих версиях Perl.

- gmtime(%f) failed

  (W overflow) вы вызвали `gmtime` с числом, которое невозможно обработать: слишком большое, слишком маленькое или NaN. Возвращаемое значение - `undef`.

- Hexadecimal float: exponent overflow

  (W overflow) Шестнадцатеричное значение с плавающей запятой имеет экспоненту бóльше, чем поддерживается форматом плавающей запятой.

- Hexadecimal float: exponent underflow

  (W overflow) Шестнадцатеричное значение с плавающей запятой имеет экспоненту меньше, чем поддерживается форматом плавающей запятой.

- Hexadecimal float: mantissa overflow

  (W overflow) Шестнадцатеричное значение с плавающей запятой имеет больше бит в мантиссе (часть между `0x` и экспонентой, также известную как значащие цифры), чем поддерживается форматом плавающей запятой.

- Hexadecimal float: precision loss

  (W overflow) Шестнадцатеричное значение с плавающей запятой имеет внутреннее представление с большим числом цифр, чем может быть выведено. Это может быть вызвано неподдерживаемым форматом long

double или 64-битными целыми, которые недоступны (требуемые для получения цифр не некоторых конфигурациях).

- [Locale ‘%s’ may not work well.%s](https://metacpan.org/pod/perldiag#Locale ‘%s’ may not work well.%s)

  (W locale) Вы используется именованную локаль, которая не является UTF-8, которую Perl определил как не полностью совместимую с Perl. Второй `%s` указывает на причину.

- localtime(%f) failed

  (W overflow) Вы вызвали `localtime` с числом, которое не может быть обработано: слишком большое, слишком маленькое или NaN. Возвращаемым значением является `undef`.

- Negative repeat count does nothing

  (W numeric) Вы пытались выполнить оператор повторения `x` меньше чем 0 раз, что не имеет смысла.

- NO-BREAK SPACE in a charnames alias definition is deprecated

  (D deprecated) Вы определили имя символа, содержащий неразрывный символ пробела. Замените его на обычный пробел. Обычно подобные имена определены в аргументе импорта `:alias` для `use charnames`, но они могут быть определены переводчиком в `$^H{charnames}`. Смотрите “CUSTOM ALIASES” in charnames.

- Non-finite repeat count does nothing

  (W numeric) Вы попытались выполнить оператор повторения `x Inf` (или `-Inf`) раз, что не имеет смысла.

- [PerlIO layer ‘:win32’ is experimental](https://metacpan.org/pod/perldiag#PerlIO layer ‘:win32’ is experimental)

  (S experimental::win32_perlio) Слой PerlIO `:win32` является экспериментальным. Если вы готовы принять риск его использования просто отключите предупреждение:

  ```
  1 no warnings "experimental::win32_perlio";
  ```

- [Ranges of ASCII printables should be some subset of “0-9”, “A-Z”, or “a-z” in regex; marked by <– HERE in m/%s/](https://metacpan.org/pod/perldiag#Ranges of ASCII printables should be some subset of “0-9”, “A-Z”, or “a-z” in regex; marked by <– HERE in m/%s/)

(W regexp) (только при действии `use re 'strict'` или внутри `(?[...])`).

Строгие правила позволяют найти опечатки и другие ошибки. Возможно вы даже и не планировали использовать здесь диапазон, если под "–" понимался другой символ или он должен был быть экранирован ("\–"). Если вы подразумевали диапазон, то то, что было использовано не является переносимым между ASCII и EBCDIC платформами и не имеют очевидного смысла для случайного читателя.

```
1 [3–7]    # OK; Очевидно и переносимо
2 [d–g]    # OK; Очевидно и переносимо
3 [A–Y]    # OK; Очевидно и переносимо
4 [A–z]    # Неправильно; Непереносимо; непонятно, что имелось
      ввиду
5 [a–Z]    # Неправильно; Непереносимо; непонятно, что имелось
      ввиду
6 [%–.]    # Неправильно; Непереносимо; непонятно, что имелось
      ввиду
7 [\x41–Z] # Неправильно; Непереносимо; непонятно для не–гика
```

(Вы можете улучшить переносимость, указав Юникод-диапазон, что означает, что точки указываются с использованием `N{...}`, но смысл может быть по-прежнему неочевиден. Строгие правила требуют, чтобы диапазоны, которые начинаются или заканчиваются ASCII символом, которые не являются контрольными, чтобы все крайние значения были литеральными символами, а не экранирующими последовательностями (как например, "\x41") и диапазоны должны состоять только из всех цифр, всех букв в верхнем регистре или всех букв в нижем регистре.

- Ranges of digits should be from the same group in regex; marked by <– HERE in m/%s/

  (W regexp) (только при действии `use re 'strict'` или внутри `(?[...])`)

  Строгие правила помогают найти опечатки или ошибки. Вы включили диапазон и одна из его границ является десятичной цифрой. При строгих правилах, когда такое происходит, обе границы должны быть цифрами в той же самой группе из 10 последовательных цифр.

- Redundant argument in %s

  (W redundant) Вы вызвали функцию с большим числом аргументов, чем требовалось, как показывает информация внутри других аргументов,

которые вы передали (например, формат printf). На данный момент выводится, когда printf-подобный формат требует меньше аргументов, чем было передано, но может использоваться в будущем для, например, .

Категория предупреждений `redundant` (излишний) является новой. Смотрите также [perl #121025].

- Replacement list is longer than search list

  Это не новое диагностическое сообщение, но в ранних релизах оно не отображалось, если транслитерация содержала широкие символы. Теперь это исправлено, поэтому вы можете увидеть это сообщение в местах, в которых раньше вы его не видели (но должны были бы).

- Use of _ for non-UTF-8 locale is wrong. Assuming a UTF-8 locale

  (W locale) Вы используете регулярное выражение с правилами локали, произошло совпадение с Юникод-границей, при этом локаль не является Юникодной. Это не имеет смысла. Perl продолжит, подразумевая Юникод (UTF-8) локаль, но результаты могут быть некорректными за исключением если локаль оказалась ISO-8859-1 (Latin1), где это сообщение ложное и может быть проигнорировано.

  Категория предупреждений `locale` является новой.

- [Using /u for ‘%s’ instead of /%s in regex; marked by <– HERE in m/%s/](https://metacpa
  /u for ‘%s’ instead of /%s in regex; marked by <– HERE in m/%s/)

  (W regexp) Вы использовали Юникод границу (`\b{...}` или `\B{...}`) в части регулярного выражения, где действуют модификаторы `/a` или `/aa`. Эти два модификатора указывают на ASCII интерпретацию и это лишено смысла для Юникод определения. Сгенерированное регулярное выражение будет собрано так, что все границы будут использовать Юникод. Другие части регулярного выражения затронуты не будут.

- The bitwise feature is experimental

  (S experimental::bitwise) Это предупреждение выводится, если вы использовали битовые операторы (`& | ^ ~ &. |. ^. ~.`) с включённой возможностью “bitwise”. Просто подавите это предупреждение, если вы хотите использовать эту возможность, но учтите, что в этом случае вы берёте на себя риск использования экспериментальной возможности, которая может измениться или быть удалённой в будущих версиях Perl:

```
no warnings "experimental::bitwise";
use feature "bitwise";
$x |.= $y;
```

- Unescaped left brace in regex is deprecated, passed through in regex; marked by <– HERE in m/%s/

  (D deprecated, regexp) Вы использовали литеральный символ `"{"` в шаблоне регулярного выражения. Вы должны заменить его на `"\{"`, поскольку в будущих версиях Perl (ориентировочно v5.26) это будет расцениваться как синтаксическая ошибка. Если разделитель шаблона также является фигурной скобкой, то любая совпадающая правая скобка (`"}"`) также должна быть экранирована, чтобы не запутывать парсер, например:

  ```
  qr{abc\{def\}ghi}
  ```

- Use of literal non-graphic characters in variable names is deprecated

  (D deprecated) Использование литеральных неграфических символов (включая контрольные) в исходном коде для ссылки на *ˆFOO* переменные, например `$^X` и `${^GLOBAL_PHASE}` теперь считается устаревшим.

- Useless use of attribute “const”

  (W misc) Атрибут `const` действует только на анонимные прототипы замыканий. Вы применил их к подпрограмме через attributes.pm. Это имеет смысл только внутри атрибутов анонимной подпрограммы.

- Useless use of /d modifier in transliteration operator

  Это не новое диагностическое сообщение, но в ранних релизах оно не отображалось, если транслитерация содержала широкие символы. Теперь это исправлено, поэтому вы можете увидеть это сообщение в местах, в которых раньше вы его не видели (но должны были бы).

- [“use re ‘strict’ ” is experimental](https://metacpan.org/pod/perldiag#“use re ‘strict’ ” is experimental)

  (S experimental::re_strict) Отличия в поведении, когда шаблон регулярных выражений компилируется при действии `'strict'`, могут меняться в будущих релизах Perl несовместимом образом; есть также предложения изменить способ включения строгой проверки вместо использования субпрагмы. Это означает, что шаблон, который компилируется сегодня, не будет компилироваться в будущих версиях Perl. Это предупреждение сигнализирует вам об этом риске.

- Warning: unable to close filehandle properly: %s

  Warning: unable to close filehandle %s properly: %s

  (S io) Раньше Perl молча игнорировал любые ошибки, когда делал неявное закрытие файлового дескриптора, то есть там, где счётчик ссылок файлового дескриптора достигал нуля и код пользователя ещё не вызывал `close()`; например:

  ```
  {
      open my $fh, '>', $file  or die "open: '$file': $!\n";
      print $fh, $data  or die;
  } # здесь неявное закрытие
  ```

  В ситуациях, когда диск полон, из-за буферизации, ошибка могла быть обнаружена только во время финального закрытия, таким образом, отсутствие проверки результатов закрытия может быть опасно.

  Теперь Perl предупреждает о подобных ситуациях.

- Wide character (U+%X) in %s

  (W locale) При использовании однобайтовойлокали (*то есть*, не UTF-8), был обнаружен многобайтовый символ. Perl рассматривает этот символ, как указанная кодовая точка Юникод. Комбинация не-UTF-8 локалей и Юникода опасна. Почти наверняка некоторые символы будут иметь разное представление. Например, в локали ISO 8859-7 (Греческая), кодовая точка 0xC3 представляет заглавную гамму. Но тем же самым является 0x393. Это делает сравнение строк ненадёжным.

  Вы должны найти как этот многобайтовый символ смешался с вашей однобайтной локалью (возможно вы думали, что у вас UTF-8 локаль, но Perl с этим не согласился).

  Категория предупреждений `locale` является новой.

Изменения в существующей диагностике

- <> должны быть кавычки

  Это предупреждение изменилось на <> at require-statement should be quotes, чтобы сделать сообщение более идентичным.

- [Argument “%s” isn’t numeric%s](https://metacpan.org/pod/perldiag#Argument “%s” isn’t numeric%s)

  В записи perldiag для этого предупреждения было внесено пояснение:

```
Обратите внимание, что для Inf и NaN (бесконечность и не-
    число) опередление
"числовой" нечто необычное; строка сами по себе (например, "
    Inf")
рассматриваются как числовые, а всё, что следует за нами
    рассматривается как
нечисловое значение.
```

- [Global symbol “%s” requires explicit package name](https://metacpan.org/pod/perldiag
  symbol “%s” requires explicit package name (did you forget to declare “my %s”?))

  К этому сообщению было добавлено ‘(did you forget to declare “my %s”?)’ (не забыли ли вы сделать объявление “my %s”?), чтобы сделать его более полезным для начинающих Perl программистов. [perl #121638]

- Сообщение ‘переменная “my” &foo::bar не может быть в пакете’ было переработано, используя слово ‘подпрограмма” вместо ’переменная’.

- `N{}` in character class restricted to one character in regex; marked by <– HERE in m/%s/

  В этом сообщении *character class* (класс символа) был изменён на *inverted character class or as a range end-point is* (инвертированный символьный класс или как граница диапазона), чтобы отметить улучшение в `qr/[\N{named sequence}]/` (смотрите в “Selected Bug Fixes”).

- panic: frexp

  К этому сообщению было добавлено ‘: `%f`’, чтобы показать виновное число с плавающей запятой.

- *Possible precedence problem on bitwise %c operator* перефразировано Possible precedence problem on bitwise %s operator.

- Unsuccessful %s on filename containing newline

  Это предупреждение теперь выводится, только если есть перевод строки в конце имени файла.

- “Variable `%s` will not stay shared” (переменная `%s` перестанет быть разделённой) было изменено, чтобы сказать “Subroutine” (подпрограмма), когда используется лексическая подпрограмма, которая перестанет быть разделённой.

- Variable length lookbehind not implemented in regex m/%s/

  Запись в perldiag для этого предупреждения имеет информацию о добавленном поведении Юникода.

Удалённые диагностические сообщения

- “Ambiguous use of -foo resolved as -&foo()”

  Здесь нет никакой неопределённости и это мешает использовать отрицательные значения констант, например, `-Inf`.

- “Constant is not a FOO reference”

  Было удалено разыменование констант (например, `my_constant->()`) во время компиляции, поскольку не учитывалась возможная перегрузка. [perl #69456] [perl #122607]

Изменения в утилитах

Удаление `find2perl`, `s2p` и `a2p`

- Директория `x2p/` была удалена из базового Perl.

  Это привело к удалению find2perl, s2p и a2p. Они все выпущены на CPAN как отдельные дистрибутивы (`App::find2perl`, `App::s2p`, `App::a2p`).

h2ph

- `h2ph` теперь поддерживает шестнадцатеричные константы в предопределённых компилятором макро-определениях, как видимые в `$Config{cppsymbols}`. [perl #123784].

encguess

- Больше не зависит от не базовых модулей.

Конфигурация и компиляция

- `Configure` теперь проверяет наличие `lrintl()`, `lroundl()`, `llrintl()` и `llroundl()`.
- `Configure` с `–Dmksymlinks` теперь должен быть быстрее. [perl #122002].
- Библиотеки `pthreads` и `cl` будут слинкованы, если присутствуют при сборке. Это позволит XS модулям, которым требуются треды для работы на perl без поддержки тредов. Обратите внимание, что вам по-прежнему требуется передавать аргумент `–Dusethreads`, если требуется поддержка тредов в perl.
- Для long double (чтобы получить больше точности и диапазона чисел с плавающей запятой) можно использовать библиотеку GCC quadmath, которая реализует четырёхкратную точность чисел с плавающей запятой на платформах x86 и IA-64. Смотрите подробности в `INSTALL`.
- MurmurHash64A и MurmurHash64B могут быть сконфигурированы как внутренняя хеш-функция.
- `make test.valgrind` теперь поддерживает параллельное тестирование.

  Например:

  ```
  1 TEST_JOBS=9 make test.valgrind
  ```

  Смотрите подробности в “valgrind” in perlhacktips.

  [perl #121431]

- Опция сборки MAD (различные украшения атрибутов) была удалена

  Это ныне не поддерживаемая попытка сохранения распарсеного дерева Perl более точным, чтобы была возможность сделать автоматическую конверсию Perl 5 в Perl 6 более простой.

  Эта опция конфигурации не сопровождается уже много лет и вероятно уже серьёзно отклонилась от актуальности с обеих сторон: как Perl 5, так и Perl 6.

- Доступен новый флаг компиляции `-DPERL_OP_PARENT`. Больше информации в дискуссии выше в “Internal Changes”.
- Pathtools больше не пытаются загрузить XS на miniperl. Это значительно ускоряет сборку perl.

Тестирование

- Был добавлен `t/porting/re_context.t` для проверки, что utf8 и его зависимости являются только подмножество переменных захвата `$1..$n`, которые зашиты в `Perl_save_re_context()` для локализации, поскольку в этой функции нет эффективной возможности во время исполнения определить какие переменные локализованы.
- Тесты на производительность были добавлены в файл `t/perf/taint.t`.
- Некоторые тесты регулярных выражений написаны так, что они будут работать очень медленно, если какие-то определённые оптимизации окажутся сломанными. Эти тесты были перенесены в новые файлы `t/re/speed.t` и `t/re/speed_thr.t`, и они запускаются с использованием `watchdog()`.
- `test.pl` теперь разрешает `plan skip_all => $reason`, чтобы сделать его более совместимым с `Test::More`.
- Был добавлен новый тестовый скрипт `op/infnan.t` для тестирования, что бесконечность и не-числа работают корректно. Смотрите “Infinity and NaN (not-a-number) handling improved”.

Поддержка платформ

Восстановленные платформы

- Платформы IRIX и Tru64 снова работают.

  Остаются несколько ошибок во время `make test`: [perl #123977] и [perl #125298] для IRIX; [perl #124212], [cpan #99605], и [cpan #104836 для Tru64.

- Работа z/OS с кодовой страницей 1047 EBCDIC

  Базовый perl теперь работает на этой EBCDIC платформе. Ранние версии также работали, но несмотря на то, что официально поддержка не

была отозвана, последние версии Perl не компилировались и не работали нормально. Perl 5.20 в целом работал, но имел множество багов, который теперь были исправлены. Множество CPAN модулей, которые поставляются с Perl по-прежнему не проходят тесты, включая и `Pod::Simple`. Однако версия `Pod::Simple` на CPAN должна работать: она была исправлено слишком поздно для включения в Perl 5.22. Ведётся работа по исправлению множества по-прежнему сломанных CPAN модулей, которая появится на CPAN, как только она будет завершена. Поэтому вам не придётся ждать Perl 5.24, чтобы получить работающие версии.

Платформы, для которых прекращена поддержка

- NeXTSTEP/OPENSTEP

  NeXTSTEP была проприетарной платформой, которая поставлялась с рабочими станциями NeXT в ранние 90-е; OPENSTEP была спецификацией API, которая предоставляла NeXTSTEP-подобное окружение для не-NeXTSTEP систем. Обе уже долгое время мертвы, поэтому поддержка сборки Perl на них была удалена.

Платформо-специфичные заметки

- EBCDIC

  Требуется специальная обработка для perl интерпретатора на EBCDIC платформах, чтобы `qr/[i-j]/` совпадало только с `"i"` и `"j"`, поскольку существует 7 символов между кодовыми точками `"i"` и `"j"`. Эта специальная обработка срабатывала только для диапазона, обе границы которого являлись литералами. Теперь это также срабатывает для любых форм `\N{...}`, указывающих символ по имени кодовой точки Юникод вместо литерала. Смотрите “Character Ranges” in perlrecharclass.

- HP-UX

  archname теперь отличает use64bitint от use64bitall.

- Android

  Поддержка сборки была улучшена для кросс-компиляции в целом и в частности для Android.

- VMS

  - При создании субпроцесса без ожидания возвращаемое значение теперь корректное значение PID.
  - Исправлен прототип, поэтому линковка не ломается при использовании компилятора C++ VMS.
  - Определение `finite`, `finitel` и `isfinite` было добавлено в `configure.com`, поддержка окружения имеет небольшие изменения и исправления для проверки статуса устаревших возможностей.

- Win32

  - `miniperl.exe` теперь собирается с `-fno-strict-aliasing`, позволяя 64-битным сборкам завершаться на GCC 4.8. [perl #123976]
  - `nmake minitest` теперь работает на Win32. Из-за проблем с зависимостями вам необходимо сначала выполнить `nmake test-prep` и некоторое небольшое число тестов упадёт. [perl #123394]
  - Perl теперь может быть собран с C++ режиме на Windows установкой макроса мэйкфайла `USE_CPLUSPLUS` в значение "define".
  - Списочная форма открытия канала была реализована для Win32. Обратите внимание: вместо `system LIST` это больше не откатывается на шелл. [perl #121159]
  - Новые опции конфигурации `DebugSymbols` и `DebugFull` были добавлены к мейкфайлам Windows.
  - Ранее сборка XS модулей (включая модули со CPAN) с использованием Visual C++ для Win64 приводило к большому числу предупреждений на каждый фал из `hv_func.h`. Эти предупреждения были заглушены.
  - Поддержка сборки без PerlIO была удалена из мэйкфайлов Windows. He-PerlIO сборки считаются устаревшими, начиная с Perl 5.18.0 и уже не поддерживаются в `Configure` на POSIX системах.
  - От 2 до 6 миллисекунд и семь I/O вызовов было сохранено на каждую попытку открыть perl модуль для каждого путя в `@INC`.
  - Сборка с Intel C теперь всегда проходит с установленной опцией C99.
  - `%I64d` теперь используется вместо `%lld` для MinGW.
  - Был исправлен крах в `open` в экспериментальном слое `:win32`. Также открытие `/dev/null` (которое работало по умолчанию в `:unix`

слое в Перле Win32) было реализовано для `:win32`. [perl #122224]
  - Была добавлена новая опция мэйкфайла `USE_LONG_DOUBLE` в dmake мэйкфайле Windows только для сборки в gcc. Установите её в значение “define”, если вы хотите, чтобы perl использовал long double для большей точности и расширенного диапазона чисел с плавающей запятой.

- OpenBSD

  На OpenBSD, Perl по умолчанию используется системный `malloc`, чтобы задействовать возможности безопасности, которые он предоставляет. Собственная обёртка malloc использовалась, начиная с v5.14 по соображениям производительности, но OpenBSD проект верит, что безопасность стоит того и считает, чтобы пользователи, которым нужна скорость специально потребуют этого. [perl #122000].

- Solaris

  - Мы ищем компилятор Sun Studio в двух местах: `/opt/solstudio*` и `/opt/solarisstudio*`.
  - Сборки на Solaris 10 с опцией `–Dusedtrace` сразу упадут, так как make проигнорирует предполагаемую зависимость для сборки `perldtrace.h`. Добавлена явная зависимость в `depend`. [perl #120120]
  - Опции C99 были удалены; поиск как `solstudio`, так и `SUNWspro`; а также была добавлена поддержка нативного `setenv`.

Внутренние изменения

- Была добавлена экспериментальная поддержка разрешить операциям в дереве опреаций на ходить своих родителей, если они есть. Это включается с помощью опции сборки `–DPERL_OP_PARENT`. Рассматривается возможность, что когда-нибудь это станет опцией, включённой по умолчанию, таким образом XS код, который напрямую запрашивает поле `op_sibling` должен быть обновлён, чтобы работать нормально в будущем.

  На сборках с `PERL_OP_PARENT` поле `op_sibling` было переименовано в `op_sibparent` и был добавлен новый флаг `op_moresib`. На последней

операции в цепи потомков op_moresib является ложью, а op_sibparent указывает на родителя (если есть), а не равен NULL.

Чтобы существующий код прозрачно работал с использованием PERL_OP_PARENT или без, было добавлено несколько новых макросов и функций, которые должны использоваться, вместо того, чтобы напрямую манипулировать с op_sibling.

Для случая, когда требуется прочесть op_sibling, только чтобы определить следующего потомка, были добавлены два макроса. Для простого сканирования через цепь потомков:

```
for (; kid->op_sibling; kid = kid->op_sibling) { ... }
```

теперь должно быть написано так:

```
for (; OpHAS_SIBLING(kid); kid = OpSIBLING(kid)) { ... }
```

Для дополнения дерева операций была добавлена общая функция op_sibling_splice(), которая позволяет манипулировать с цепью операций-потомков. По аналогии с Perl функцией splice() она позволяет удалить одну или больше операций из цепи потомков и заменить их нулём или несколькими новыми операциями. Она прозрачно поддерживает все изменения в потомков, родителей и op_last указателей и прочего.

Если требуется манипулировать с операциями на низком уровне, то есть три новых макроса OpMORESIB_set, OpLASTSIB_set и OpMAYBESIB_set , которые предназначены для низкоуровневой переносимой установки op_sibling / op_sibparent, а также обновления op_moresib. Первый устанавливает указатель потомка на нового потомка, второй устанавливает операцию последним потомком, а третий по условию выполняет первое или второе действие. Обратите внимание, что в отличии от op_sibling_splice() эти макросы не поддерживают согласованное состояние родителя в то же самое время (то есть обновление op_first и op_last, когда это необходимо).

Были добавлены функция C-уровня Perl_op_parent() и метод Perl-уровня B::OP::parent(). C-функция существует только в сборках с PERL_OP_PARENT (её использование на ванильных сборках это ошибка во время компиляции). B::OP::parent() присутствует всегда, но на ванильных сборках он всегда возвращает NULL. При наличии PERL_OP_PARENT они вернут родителя текущей операции, если она есть. Переменная $B::OP::does_parent позволяет вам определить поддерживает ли B получение предка операции.

PERL_OP_PARENT была добавлена в 5.21.2, но интерфейс изменился значительно в 5.21.11. Если вы обновили ваш код до изменений в 5.21.11, потребуется дальнейшая переработка. Основные изменения после 5.21.2:

  - Макросы OP_SIBLING и OP_HAS_SIBLING были переименованы в OpSIBLING и OpHAS_SIBLING для согласованности с другими макросам манипулирующими с операциями.
  - Поле op_lastsib было переименовано в op_moresib и его значение было инвертировано.
  - Макрос OpSIBLING_set был удалён и был заменён на OpMORESIB_set .
  - Функция op_sibling_splice() теперь поддерживает нулевой аргумент parent, при котором срез не затрагивает первую и последнюю операции в цепи потомков.

- Был создан макрос, чтобы позволить XS коду лучше манипулировать с POSIX категорией локали LC_NUMERIC. Смотрите “Locale-related functions and macros” in perlapi.

- Функция полной замены atoi grok_atou теперь замещена grok_atoUV. Смотрите подробности в perlclib.

- Была добавлена новая функция Perl_sv_get_backrefs(), которая позволяет получать слабые ссылки, если они доступны, которые указывают на SV.

- Функция screaminstr() была удалена. Несмотря на то, что она отмечена как публичная, она не была задокументирована и не использовалась в модулях на CPAN. Вызов функции вызывал фатальную ошибку, начиная с 5.17.0.

- Функции newDEFSVOP(), block_start(), block_end() и intro_my() были добавлены в API.

- Внутренняя функция convert в op.c была переименована в op_convert_list и добавлена в API.

- Функция sv_magic() больше не запрещает “ext” магию на доступных только для чтения значениях. В конце концов perl не может узнать будет ли заданная магия модифицировать SV или нет. [perl #123103].

- Доступ к “CvPADLIST” in perlapi в XSUB теперь запрещён.

  Поле `CvPADLIST` было задействовано для других внутренних целей для XSUB. Поэтому, в частности, вы больше не можете рассчитывать, что он имеет значение NULL, для проверки является ли CV XSUB. Используйте вместо этого `CvISXSUB()`.

- SV типа `SVt_NV` теперь иногда не имеют тела, если конфигурация и платформа поддерживают это: особенно, когда `sizeof(NV)<= sizeof(IV)`. “Безтелесность” означает, то NV значение сохраняется непосредственно в заголовке SV без необходимости выделения отдельного тела. Этот трюк уже использовался для IV, начиная с 5.9.2 (хотя в случае IV, это использовалось всегда, независимо от платформы и сборочной конфигурации).

- Переменные `$DB::single`, `$DB::signal` и `$DB::trace` теперь для set- и get-магии хранят свои значения как IV, и эти IV используются, когда тестируются их значения в `pp_dbstate()`. Это предотвращает бесконечную рекурсию в perl, если перегруженный объект присваивается к любой из этих переменных. [perl #122445].

- `Perl_tmps_grow()`, который был отмечен, как публичное API, но не документирован, был удалён из публичного API. Это изменение не затрагивает XS-код, который использует макрос `EXTEND_MORTAL` для предрасширения mortal стека.

- Внутренности Perl больше не устанавливают и не используют флаг `SVs_PADMY`. `SvPADMY()` теперь возвращает истинное значение для всего, что не отмечено `PADTMP`, а `SVs_PADMY` теперь определено как 0.

- Макросы `SETsv` и `SETsvUN` были удалены. Они больше не используется в базовом Perl, начиная с коммита 6f1401dc2a пять лет назад, и не были обнаружены на CPAN.

- Бит `SvFAKE` (неиспользуемый на HV) неофициально зарезервирован Дэвидом Митчеллом для будущей работы над vtables.

- Функция `sv_catpvn_flags()` принимает флаги `SV_CATBYTES` и `SV_CATUTF8`, которые указывают, что добавляемая строка является байтами или UTF-8 соответственно. (Эти флаги присутствуют начиная с 5.16.0, но раньше не упоминались как часть API.)

- Был представлен новый класс опкодов METHOP. Он хранит информацию, используемую во время исполнения для улучшения производительности вызовов методов класса/объекта.

  OP_METHOD и OP_METHOD_NAMED были изменены из UNOP/SVOP в METHOP.

- cv_name() это новая функция API, которой может быть передан CV или GV. Она возвращает SV, содержащий имя подпрограммы, для использования в диагностике.

  [perl #116735] [perl #120441]

- cv_set_call_checker_flags() это новая функция API, которая работает как cv_set_call_checker(), но в отличии от неё позволяет вызывающему указать требует ли проверщик вызова полного GV для вывода имени подпрограммы, или вместо него может быть передан CV. Какое бы значение не было передано оно будет приемлемо для cv_name(). cv_set_call_checker() гарантирует, что это будет GV, но ему может потребоваться создать его на лету, что неэффективно. [perl #116735]

- CvGV (который не является частью API) теперь более сложный макрос, который может функции и создавать GV. Для тех случаев, когда он используется как логическое значение, добавляется CvHASGV, который вернёт истину для CV, которое номинально имеет GV, но без создания самого GV. CvGV также теперь возвращает GV для лексических подпрограмм. [perl #120441]

- Функция “sync_locale” in perlapi была добавлена в публичное API. Нужно избегать изменений локали программы из XS кода. Однако это требуется некоторым не-Перл библиотекам, вызываемым из XS, как например Gtk. Когда это происходит, Перлу необходимо сообщить о том, что локаль была изменена. Используйте эту функцию, чтобы сделать это, до того как вернуть управление Перл.

- Определения и метки для флагов в поле op_private операций теперь автогенерируются из данных в regen/op_private. Эффект от этого можно заметить в том, что вывод некоторых флагов Concise может немного отличаться а вывод флагов perl −Dx может отличаться значительно (они оба используют сейчас тот же набор меток). Также, сборки с отладочной информацией теперь имеют несколько новых assert’ов в op_free(), чтобы быть уверенным, что операции не имеют неопознанных флагов, установленных в op_private.

- Устаревшая переменная PL_sv_objcount была удалена.

- Perl теперь пытается сохранить категорию локали LC_NUMERIC установленной в "C" за исключением операций, которым требуется установить её для заданной локали программы. Это защищает множество XS модулей, которые не могут справиться с ситуацией, когда разделитель целой и дробной части не является точкой. До этого релиза Perl устанавливал эту категорию в "C", но вызов POSIX ::setlocale() менял её. Теперь подобный вызов изменит локаль категории LC_NUMERIC для программы, но локаль сообщаемая XS коду останется в значении "C". Есть новые макросы для манипуляции с локалью LC_NUMERIC, включая STORE_LC_NUMERIC_SET_TO_NEEDED и STORE_LC_NUMERIC_FORCE_TO_UNDERLYING. Смотрите "Locale-related functions and macros" in perlapi.

- Был написан новый макрос isUTF8_CHAR, который эффективно определяет является ли строка, переданная параметром, начинается с правильно сформированным кодированным в UTF-8 символом.

- В следующих функциях приватного API удалён параметр контекста: Perl_cast_ulong, Perl_cast_i32, Perl_cast_iv, Perl_cast_uv, Perl_cv_const_sv, Perl_mg_find, Perl_mg_findext, Perl_mg_magical, Perl_mini_mktime, Perl_my_dirfd, Perl_sv_backoff, Perl_utf8_hop.

  Обратите внимание, что версии этих функций без префикса, которые являются частью публичного API, такие как cast_i32(), остались неизменными.

- Типы PADNAME и PADNAMELIST теперь раздельные типы и больше не могут быть просто приравнены к SV и AV. [perl #123223].

- Имена pad теперь всегда в UTF-8. Макрос PadnameUTF8 всегда возвращает истину. Ранее это уже было так, но поддержка для двух различных типов внутреннего представления имён pad была удалена.

- Был добавлен новый класс операций UNOP_AUX. Это субкласс UNOP с добавлением поля op_aux, который указывает на массив объединений UV, SV* и т.д. Он предназначен для случая, когда операции требуется сохранить больше данных, чем просто op_sv или подобное. На данный момент единственная подобная операция этого типа это OP_MULTIDEREF (смотрите следующий пункт).

- Была добавлена новая операция OP_MULTIDEREF, которая производит один или несколько поисков во вложенных массивах или хешах, где

ключ является константой иди простой переменной. Например, выражение `$a[0]{$k}[$i]`, которая ранее включало десять операций `rv2Xv`, `Xelem`, `gvsv` и `const`, теперь выполняется за одну операцию `multideref`. Это также работает для `local`, `exists` и `delete`. Сложные выражения для индекса, такие как `[$i+1]` по-прежнему выполняются с использованием `aelem`/`helem`, а просмотр одноуровнего массива с небольшим индексом-константой по-прежнему выполняется с использованием `aelemfast`.

Избранные исправления ошибок

- `close` теперь устанавливает `$!`

  Когда происходит ошибка ввода/вывода, факт подобной ошибки сохранялся в дескриптор. `close` возвращает ложь для подобных дескрипторов. Ранее значение `$!` оставлялось `close` нетронутым, поэтому общепринятый подход записи `close $fh or die $!` не работал полноценно. Теперь дескриптор сохраняет также и значение `$!`, а `close` его восстанавливает.

- `no re` теперь делает больше и лексически

  Раньше использование `no re` отключало лишь несколько опций. Теперь оно полностью отключает все включённые опции. Например, раньше вы не могли отключить отладку после её включения в том же самом блоке.

- `pack("D", $x)` и `pack("F", $x)` теперь обнуляют заполнение на x86 сборках с long double. При некоторых сборочных опциях на GCC 4.8 и старше они перезаписывали обнулённое заполнение или целиком игнорировали инициализированный буфер. Это приводило к ошибкам в `op/pack.t` [perl #123971]

- Расширение массива, клонированного из потока родителя могло привести к ошибке “Предпринята попытка модификации значения доступного только на чтение”, при попытке модифицировать новые элементы. [perl #124127]

- Ошибка предположения и последующий крах в выражении `*x=<y>` была исправлена. [perl #123790]

- Ошибка возможного краха/зацикливания, связанная с компиляцией лексических подпрограмм, была исправлена. [perl #124099]

- UTF-8 теперь работает корректно в именах функции, в нецитированных ограничителях встроенных документов, а также в именах переменных, используемых как индексы массива. [perl #124113]

- Повторяемый глобальный шаблон в скалярном контексте на крупных связанных строках выполнял поиск соответствия с экспоненциальным замедлением, в зависимости от текущей позиции в строке. [perl #123202]

- Были исправлены различные крахи, из-за того, что парсер смущали синтаксические ошибки. [perl #123801] [perl #123802] [perl #123955] [perl #123995]

- Был исправлен `split` в области лексической `$_` не падать из-за нарушения предположения. [perl #123763]

- Синтаксис `my $x : attr` внутри различных списочных операторов больше не приводит к падению из-за нарушения предположения. [perl #123817]

- Знак `@` в кавычках с последующей не-ASCII цифрой (что не является корректным идентификатором) приводил парсер к краху вместо того, чтобы просто попытаться использования `@` как литерал. Это было исправлено. [perl #123963]

- `*bar::=*foo::=*glob_with_hash` падали, начиная с Perl 5.14, но больше не делают этого. [perl #123847]

- `foreach` в скалярном контексте не размещал элемент на стеке, приводя к ошибкам. (`print 4, scalar do { foreach(@x){} } + 1` напечатает 5.) Это было исправлено и возвращает `undef`. [perl #124004]

- Было исправлено несколько случаев, когда данные, хранившие значения переменных окружения в базовом C коде, могли потенциально быть переписаны. [perl #123748]

- Некоторые шаблоны, начинающиеся с `/.*..../` при поиске совпадения в длинных строках были медленными, начиная с v5.8, а некоторые формы `/.*..../i` замедлились, начиная с v5.18. Теперь они снова быстрые. [perl #123743].

- Оригинальное видимое значение `$/` сохраняется, когда устанавливается в некорректное значение. Раньше, например, если вы устанавливали `$/` в ссылку на массив, то Perl выводил ошибку исполнения и не устанавливал `PL_rs`, и код, который проверял `$/` видел ссылку на массив. [perl #123218].

- В шаблонах регулярных выражений, POSIX класс, как например `[: ascii:]`, должен быть внутри класса символов, заключённый в скобки, как например `qr/[[:ascii:]]/`. Теперь выводится предупреждение, когда что-то напоминающее POSIX класс находится не внутри класса, заключённого в скобки. Это предупреждение не выводилось, если было отрицание POSIX класса: `[:^ascii:]`. Теперь это исправлено.

- В Perl 5.14.0 появился баг, когда `eval { LABEL: }` приводил к краху. Это было исправлено. [perl #123652].

- Были исправлены различные крахи, из-за того, что парсер смущали синтаксические ошибки. [perl #123617]. [perl #123737]. [perl #123753]. [perl #123677].

- Код, как например `/$a[/`, раньше читал следующую строку ввода и рассматривал её как если бы она шла сразу после открытой скобки. Некоторый некорректный код мог парситься и выполняться, но некоторый код приводил к краху, поэтому это теперь запрещено. [perl #123712].

- Исправлено переполнение аргумента для `pack`. [perl #123874].

- Исправлена обработка не строгого `\x{}`. Теперь `\x{}` является эквивалентом для `\x{0}` вместо того, чтобы падать.

- `stat –t` больше не считается стекируемым, также как и `–t stat`. [perl #123816].

- Следующее выражение больше не приводит к SEGV: `qr{x+(y(?0))*}`.

- Исправлен бесконечный цикл в разборе обратных ссылок в регулярных выражениях.

- Несколько небольших исправлений ошибок в поведении Infinity и NaN, включая предупреждения, когда производится приведение к числу строк, содержащих Infinity и NaN. Например, “NaNcy” больше приводится к NaN.

- Исправлена ошибка в шаблонах регулярных выражений, которая могла привести к сегфолту и другим крахам. Это происходит только в шаблонах, скомпилированных с `/i`, принимающих во внимание текущую POSIX локаль (что обычно означает, что они скомпилированы в области действия `use locale`), и это должна быть строка с по крайне мере 128 последовательными совпадающими байтами. [perl #123539].

- `s///g` теперь работает на очень больших строках (где больше 2 миллиардов итераций), вместо вылета с сообщением 'Зацикливание замены'. [perl #103260]. [perl #123071].

- `gmtime` больше не падает на значениях NaN. [perl #123495].

- `\()` (ссылка на пустой список) и `y///` с лексической `$_` в текущей области видимости может выполнять запись и в начало, и в конец стека. Теперь это исправлено и предварительно расширяет стек.

- `prototype()` без аргументов раньше читал предыдущий элемент на стеке, поэтому `print "foo", prototype()` выводило прототип foo. Это было исправлено и вместо этого использует `$_`.

  [perl #123514].

- Некоторые случаи, когда лексические state подпрограммы объявлены внутри предекларированных подпрограммах, приводили к краху. Например, при выполнении eval строки, включающей имя внешней переменной. Теперь такого не происходит.

- Некоторые случаи вложенных лексических подпрограмм внутри анонимных подпрограмм могли вызывать ошибку 'Bizarre copy' или возможно даже падать.

- При попытке выдать предупреждения, perl отладчик по умолчанию (`perl5db.pl`) иногда возвращал 'Undefined subroutine &DB::db_warn called'. Эта ошибка, которая стала происходить в Perl 5.18, теперь была исправлена. [perl #123553].

- Некоторые синтаксические ошибки в заменах, таких как `s/${<>{})//`, приводили к краху начиная с Perl 5.10. (В некоторых случаях крах не происходил до версии 5.16). Крах, естественно, был исправлен. [perl #123542].

- Исправлено несколько переполнений при увеличении длины строки, в частности, выражение повтора, как например, `33 x ~3`, приводило к

крупному переполнению буфера, так как размер нового буфера вывода неправильно обрабатывался в `SvGROW()`. Выражения, наподобие этого теперь корректно сигнализируют о панике при выделении памяти. [perl #123554].

- `formline("@...", "a");` приводит к краху. Случай `FF_CHECKNL` в `pp_formline()` не устанавливает указатель, который отмечал позицию среза, что приводило к тому, что случай `FF_MORE` приводил к краху в нарушением сегментации. Это было исправлено. [perl #123538].

- Было исправлено возможное переполнение буфера и крах при разборе литерального шаблона при компиляции регулярного выражения. [perl #123604].

- `fchmod()` и `futimes()` теперь устанавливают `$!`, когда у них происходит ошибка, при передаче закрытого файлового дескриптора. [perl #122703].

- `op_free()` и `scalarvoid()` больше не падают из-за переполнения стека при освобождении глубокого рекурсивного дерева операций. [perl #108276].

- В Perl 5.20.0, в `$^N` стал отключаться UTF-8 флаг, если к нему получали доступ из кодового блока внутри регулярного выражения, кодируя значение в UTF-8. Это было исправлено. [perl #123135].

- Завершившийся ошибкой вызов `semctl` больше не переписывает существующие элементы на стеке, что означает, что `(semctl(-1,0,0,0))[0]` больше не выдаёт предупреждение о “неициализированности”.

- `else{foo()}` без пробела перед `foo` теперь устанавливает правильный номер строки для этого выражения. [perl #122695].

- Иногда присвоение в `@array = split` оптимизируется так, что `split` пишет непосредственно в массив. Это приводило к ошибкам, нарушавшим возможность использовать это выражение для lvalue контекста. Поэтому `(@a=split//,"foo")=bar()` приводило к ошибке. (Эта ошибка появилась вероятно в Perl 3, Когда эта оптимизация была добавлена). Теперь это было исправлено. [perl #123057].

- Когда список аргументов проваливает проверку, заданную сигнатурой функции (которая по-прежнему экспериментальная возможность), результирующее сообщение об ошибке теперь даёт имя файла и номер строки вызывающей стороны, а не вызванной подпрограммы. [perl #121374].

- Флип-флоп операторы (`..` и `...` в скалярном контексте) раньше поддерживали отдельное состояние для каждого рекурсивного уровня (число раз, когда внешняя подпрограмма вызывалась рекурсивно), вопреки документации. Теперь каждое замыкание имеет внутреннее состояния для каждого флип-флопа. [perl #122829].

- Флип-флоп оператор (`..` в скалярном контексте) возвращал один и тот же скаляр каждый раз, пока окружающая подпрограмма вызывалась рекурсивно. Теперь он всегда возвращает новый скаляр. [perl #122829].

- `use`, `no`, метки оператора, специальные блоки (`BEGIN`) и pod теперь всегда разрешены, как первая часть в `map` или `grep` блоках, блоках после `print` или `say` (или других функций) возвращающих хендл, и внутри `${...}`, `@{...}` и подобных. [perl #122782].

- Оператор повтора `x` теперь распространяет lvalue контекст на свои аргументы с левой стороны, когда используется в контекстах, наподобие `foreach`. Это позволяет `for(($#that_array)x2){ ... }` ожидаемо работать, если цикл модифицирует `$_`.

- `(...)x ...` в скалярном контексте раньше портило стек, если на один из операндов действовала перегрузка “x”, вызывая ошибочное поведение. [perl #121827].

- Присвоение к лексическому скаляру часто оптимизируется; например, в `my $x; $x = $y + $z` оператор присвоения удаляется и оператор сложения пишет непосредственно в `$x`. Различные ошибки, связанные с этой оптимизацией были исправлены. Некоторые операторы на правой стороне иногда вообще не могут выполнить присвоение или присваивают неверное значение, или вызывают STORE дважды или вообще не разу на связанных переменных. Затронутыми операторами были `$foo++`, `$foo--` и `-$foo` при действии `use integer`, `chomp`, `chr` and `setpgrp`.

- Списочные присвоения иногда приводили к ошибкам, если тот же скаляр оказывался на обоих сторонах присвоения из-за использования `tied`, `values` или `each`. Результатом становилось присвоение ошибочного значения.

- `setpgrp($nonzero)` (с одним аргументом) по ошибке был изменён в 5.16 и стал значить `setpgrp(0)`. Это было исправлено.

- `__SUB__` возвращало ошибочное значение или портило память при работе в отладчике (опция `-d`) и в подпрограммах, содержащих `eval $string`.

- Когда `sub (){ $var }` становилась встраиваемой, то она теперь возвращает каждый раз разный скаляр, также как и невстраиваемая, хотя Perl по-прежнему оптимизирует копию в случаях, если это не имеет никакой заметной разницы.

- `my sub f (){ $var }` и `sub (): attr { $var }` больше не подлежат встраиванию. Первый приведёт к краху; последний просто выкинет атрибут. Исключение делается лишь для малоизвестного атрибута `:method`, который не делает ничего особенного.

- Встраивание подпрограмм с пустым прототипом теперь более последовательное, чем раньше. Раньше подпрограмма с множеством операторов, из которых все кроме последнего оптимизировались, могла встраиваться, только если это была анонимная подпрограмма, содержащая строковый `eval`, или объявление `state`, или замыкание внешней лексической переменной (или любой анонимной подпрограммой при работе отладчика). Теперь любая подпрограмма, которая приводится к простой константе после оптимизации операторов подлежит встраиванию. Это применяется для вещей наподобие `sub (){ jabber()if DEBUG; 42 }`.

  Некоторые подпрограммы с явным `return` делались встраиваемыми, вопреки документации. Теперь `return` всегда предотвращает встраивание.

- На некоторых системах, таких как VMS, `crypt` может вернуть не-ASCII строку. Если скаляр, которому идёт присвоение, содержал до этого UTF-8 строку, то `crypt` не будет убирать UTF-8 флаг, таким образом искажая возвращаемое значение. Это происходило с `$lexical = crypt ...`.

- `crypt` больше не вызывает `FETCH` дважды на связанном первом аргументе.

- Незавершённый встроенный документ на последней строке оператора цитирования (`qq[${ <<END }]`, `/(?{ <<END })/`) больше не приводит к двойному освобождению памяти. Этот дефект появился в 5.18.

- `index()` и `rindex()` больше не приводят к краху, когда используются на строках размером больше, чем 2 Гбайта. [perl #121562].

- Небольшая умышленная утечка памяти в PERL_SYS_INIT/PERL_SYS_INIT3 на сборках Win32 была исправлена. Это может затронуть встраиваемые сборки, которые регулярно создают и разрушают движок perl внутри одного процесса.

- POSIX::localeconv() теперь возвращает данные для текущей локали программы даже если вызывается вне области действия use locale.

- POSIX::localeconv() теперь корректно работает на платформах, которые не имеют LC_NUMERIC и/или LC_MONETARY, или на которых Perl был скомпилирован игнорировать обе или какую-либо одну из этих категорий локали. В подобных случаях в хеше, возвращаемом localeconv() нет полей для соответствующих значений.

- POSIX::localeconv() теперь маркирует соответственно возвращаемые значения как UTF-8 или нет. Ранее они всегда возвращались как байты, даже если должны были кодироваться как UTF-8.

- На Microsoft Windows внутри области видимости use locale следующие POSIX классы символов возвращают результаты для множества локалей, которые не соответствуют POSIX стандарту: [[:alnum:]], [[:alpha:]], [[:blank:]], [[:digit:]], [[:graph:]], [[:lower:]], [[:print:]], [[:punct:]], [[:upper:]], [[:word:]], и [[:xdigit:]]. Так происходило из-за того, что сама реализация Microsoft не следовала стандарту. Perl теперь принимает специальные меры, чтобы скорректировать это.

- Множество проблем было обнаружено и исправлено сканером Coverity.

- system() и друзья должны теперь работать правильно на Android сборках.

  По недосмотру, значение, указанное в –Dtargetsh для Configure, игнорировалось в части сборочного процесса. В итоге кросскомпилируемый perl для Android оказывался с дефектной версией system(), exec() и обратных кавычек: команды искали /bin/sh, вместо /system/bin/sh, что приводило к ошибке для большинства устройств, устанавливая $! в значение ENOENT.

- qr(...\(...\)...), qr[...\[...\]...], и qr{...\{...\}...} теперь работают. Ранее было невозможно экранировать эти три левых символа с обратной косой чертой внутри шаблона регулярного выражения,

где иначе они рассматривались как метасимволы, открывающий шаблон разделитель являлся тем символом, а закрывающий разделитель являлся его зеркальным отображением.

- `s///e` на заражённых UTF-8 строках портили `pos()`. Эта ошибка появилась в 5.20 и теперь исправлена. [perl #122148].

- Не-граница слова в регулярных выражениях (`\B`) не всегда совпадала с концом строки; в частности, `q{} =~ /\B/` не совпадало. Эта ошибка появилась в 5.14 и теперь исправлена. [perl #122090].

- `" P" =~ /(?=.*P)P/` должно совпадать, но не совпадало. Теперь это исправлено. [perl #122171].

- Ошибка компиляции `use Foo` в `eval` могло привести к ложному определению подпрограммы `BEGIN`, которое выводило предупреждение “Subroutine BEGIN redefined” на следующее использование `use` или другого блока `BEGIN`. [perl #122107].

- Синтаксис `method { BLOCK } ARGS` теперь корректно разбирает аргументы, если они начинаются с открывающей скобки. [perl #46947].

- Внешние библиотеки и Perl могут по разному воспринимать, что такое локаль. Это становится проблемой, при разборе строки версии, если разделитель цифр локали изменился. Разбор версии был исправлен, чтобы убедиться, что он обрабатывает локаль корректно. [perl #121930].

- Была исправлена ошибка, когда ошибка предположения на нулевую длину и кодовый блок внутри регулярного выражения, приводили к тому, что `pos` видел некорректное значение. [perl #122460].

- Разыменование константы теперь работает корректно для тайпглоб констант. Ранее глоб приводился к строковой форме и извлекалось его имя. Теперь используется сам глоб. [perl #69456]

- При разборе сигила (`$ @ % &`) с последующей фигурной скобкой, парсер больше не пытается угадать является ли это блоком или конструктором хеша (вызывая позже синтаксическую ошибку при таком предположении), так как дальше может идти только блок.

- `undef $reference` теперь освобождает ссылку немедленно, вместо того, чтобы ждать следующего выражения. [perl #122556]

- Различные случаи, когда имя подпрограммы используется (автозагрузка, перегрузка, сообщения об ошибках), раньше приводило к краху для лексической подпрограммы. Теперь это было исправлено.

- Поиск bareword теперь пытается избежать создания пакетов, если окажется, что bareword это не имя подпрограммы.

- Компиляция анонимных констант (например, `sub (){ 3 }`) теперь не удаляет подпрограмму с именем `__ANON__` в текущем пакете. Удалялось не только `*__ANON__{CODE}`, но также возникала утечка памяти. Этот баг появился в Perl 5.8.0.

- Объявление заглушки, наподобие `sub f;` и `sub f ();` больше не удаляло константу с тем же именем, определённую с помощью `use constant`. Этот баг появился в Perl 5.10.0.

- `qr/[\N{named sequence}]/` теперь работает правильно в большинстве случаев.

  Некоторые имена, известные в `\N{...}` ссылаются на последовательность из нескольких символов, вместо обычного единственного символа. Классы символов, заключённые в скобки обычно совпадают с одни символом, но теперь появилась специальная поддержка, которая позволяет им совпадать с именованной последовательностью, но не в случае, если класс инвертирован или последовательность указана в начале или конце диапазона. В этих случаях изменение поведения от изначального это несколько переработанное сообщение о фатальной ошибке, выводимое если класс является частью конструкции `?[...]`). Когда `[...]` находится в одиночестве, возникает такое же нефатальное предупреждение, как и раньше, и используется только первый символ последовательности, также как и раньше.

- Заражённые константы, которые вычисляются во время компиляции больше не приводят к тому, что не относящиеся выражения также становились заражёнными. [perl #122669]

- `open $$fh, ...`, который создавал хендл с именем наподобие `"main::_GEN_0"`, не давал хендлу правильный счётчик ссылок, поэтому могло происходить двойное освобождение памяти.

- При решении является ли bareword именем метода, парсер мог быть запутан, если `our` подпрограмма с тем же именем существует и пытается найти метод в пакете `our` подпрограммы, вместо пакета вызывающего.

- Парсер больше не путается при указании `\U=` внутри строки в двойных кавычках. Раньше это приводило к синтаксической ошибке, но теперь компилируется корректно. [perl #80368]

- Всегда подразумевалось, что файловые тестовые операторы `-B` и `-T` рассматривали кодированные в UTF-8 файлы как текст. (perlfunc был обновлён, чтобы уточнить это). Раньше, существовала возможность, что некоторые файлы рассматривались как UTF-8 но на деле они не были валидным UTF-8. Теперь это исправлено. Операторы теперь работают и на EBCDIC платформах.

- При некоторых условиях предупреждение возникающее при компиляции шаблона регулярного выражения выводилось несколько раз. Теперь это было исправлено.

- В Perl 5.20.0 появилась регрессия, при которой кодированный в UTF-8 шаблон регулярного выражения, который содержал одиночные ASCII буквы в нижнем регистре не совпадал с такой же буквой в верхнем регистре. Это было исправлено в 5.20.1 и 5.22.0. [perl #122655]

- Развёртка константы может подавить предупреждения, если не действуют лексические предупреждения (`use warnings` или `no warnings`) и `$^W` была ложью при компиляции, но стала истиной во время исполнения.

- Загрузка таблиц Юникода при поиске совпадения в регулярном выражении вызывала ошибки предположения в сборках с отладочной информацией, если предыдущий поиск совпадения был в таком же регулярном выражении. [perl #122747]

- Клонирование треда раньше работало некорректно для лексических подпрограмм, возможно вызывая крах или двойное освобождение при выходе.

- Начиная с Perl 5.14.0, удаление `$SomePackage::{__ANON__}` и затем присвоение значения undef анонимной подпрограмме приводило к внутреннему искажению данных, приводящему к краху Devel::Peek или выводу бессмысленных данных в B.pm. Это было исправлено.

- `(caller $n)[3]` теперь выдаёт имя лексической подпрограммы, вместо выдачи значения `"(unknown)"`.

- `sort subname LIST` теперь поддерживает использование лексических подпрограмм как подпрограмм сравнения.

- Создание псевдонимов (например, через `*x = *y`) может смутить списочное присвоение, где указываются два имени для одной и той же переменной на каждой стороне, приводя к присвоению некорректных значений. [perl #15667]

- Длинные ограничители встроенных документов вызывают ошибки чтения на коротких строках ввода. Это было исправлено. Сомнительно, что может возникнуть какой-либо крах. Этот баг относится ко времени, когда только появилсь встроенные документы в Perl 3.000 двадцать пять лет назад.

- Оптимизация в `split`, которая рассматривала `split /^/`, как `split /^/m` имела побочный эффект, когда `split /\A/` также рассматривался как `split /^/m`, чего не должно было происходить. Это было исправлено. (Однако надо заметить, что `split /^x/` не вёл себя как `split /^x/m`, что также рассматривается как баг и будет исправлен в будущей версии). [perl #122761]

- Малоизвестный синтаксис `my Class $var` (смотрите fields и attributes) мог быть сбит с толку в области действия `use utf8` если `Class` был константой, чьё значение содержит Latin-1 символы.

- Блокировка и разблокировка значений через Hash::Util или `Internals::SvREADONLY` больше не имеет никакого эффекта на значения, которые были доступны только на чтение с самого начала. Ранее разблокировка таких значений приводила к краху, зависаниям и другому некорректному поведению.

- Некоторые нетерминированные `(?(...)...)` конструкции в регулярных выражениях могут вызывать крах или выдавать неправильные сообщения об ошибках. `/(?(1)/` является одним из таких примеров.

- `pack "w", $tied` больше не вызывает FETCH дважды.

- Списочное присвоение как например, `($x, $z)= (1, $y)` теперь корректно работает если `$x` и `$y` созданные алиасы в `foreach`.

- Некоторые шаблоны, включающие кодовые блоки с синтаксическими ошибками, как например `/ (?{(^{})/`, могут зависать или падать с ошибками на сборках с отладочной информацией. Теперь они выводят ошибки.

- Ошибка предположения, при разборе `sort` с включённой отладочной информацией, была исправлена. [perl #122771].

- `*a = *b; @a = split //, $b[1]` может вызвать некорректное чтение и вывести мусорный результат.

- В выражении `()= @array = split`, `()=` в начале больше не смущает оптимизатор в предположении, что лимит 1.

- Фатальное предупреждение больше не предотвращает вывод синтаксических ошибок. [perl #122966].

- Исправлена ошибка NaN double в long-double конвертирование на VMS. Для чистых NaN (и только на Itanium, не Alpha) выводилась отрицательная бесконечность вместо Nan.

- Исправлена ошибка, которая приводила к тому, что `make distclean` ошибочно оставлял некоторые файлы. [perl #122820].

- AIX теперь устанавливает длину корректно в `getsockopt`. [perl #120835]. [cpan #91183]. [cpan #85570].

- Фаза оптимизации при компиляции регулярного выражения может работать вечно и израсходовать всю память при некоторых условиях. Теперь исправлено. [perl #122283].

- Тестовый скрипт `t/op/crypt.t` теперь использует SHA-256 алгоритм, если алгоритм по умолчанию отключён, вместо ошибки. [perl #121591].

- Исправлена ошибка на единицу при установке размера разделяемого массива. [perl #122950].

- Исправлена ошибка, которая могла привести ко входу perl в бесконечный цикл при компиляции. В частности, `while(1)` внутри субсписка, например

  ```
  1 sub foo { () = ($a, my $b, ($c, do { while(1) {} })) }
  ```

  Этот баг появился в 5.20.0 [perl #122995].

- На Win32, если переменная была локализована в псевдопроцессе, который позже производит fork, восстановление оригинального значения в потомке псевдо-процесса вызывает порчу памяти и крах в потомке псевдо-процесса (и соответственно в процессе ОС). [perl #40565].

- Вызов `write` на формате с полем `^**` может вызвать панику в `sv_chop()`, если не ыбло достаточно аргументов или если переменная, используемая для заполнения поля была пустой. [perl #123245].

- Не-ASCII лексическая подпрограмма теперь появляется без завершающего мусора, если она показывается в сообщении об ошибке.

- Прототип подпрограммы `\@` больше не выравнивает заключённые в скобки массивы (взятие ссылки на каждый элемент), но берёт ссылку на сам массив. [perl #47363].

- Блок, не содержащий ничего, вместо цикла `for` в C-стиле, вызывает повреждение стека, приводя список вне блока к потере элементов или перезаписи элементов. Это могло происходить в `map { for(...){...} } ...` и в списках, содержащих `do { for(...){...} }`. [perl #123286].

- `scalar()` теперь распространяет lvalue контекст, поэтому `for(scalar($#foo)){ ... }` может модифицировать `$#foo` через `$_`.

- `qr/@array(?{block})/` больше не умирает с сообщением “Bizarre copy of ARRAY”. [perl #123344].

- `eval '$variable'` во вложенных именованных подпрограммах может иногда искать глобальную переменную даже при наличии лексической переменной в области видимости.

- В Perl 5.20.0, `sort CORE::fake`, где ‘fake’ это что-угодно кроме ключевого слова, стало удалять последние 6 символов и рассматривать результаты, как имя подпрограммы сортировки. Предыдущее поведение, восприятия `CORE::fake`, как имени подпрограммы сортировки было восстановлено. [perl #123410].

- Вне `use utf8` односимвольная Latin-1 лексическая переменная запрещена. Сообщение об ошибке для этого случая “Can’t use global `$foo`...”, выдавало мусор, вместо имени переменной.

- `readline` на несуществующем хендле приводил к тому, что `${^LAST_FH}` выдавал ссылку на неопределённый скаляр (или падал из-за ошибки предположения). Теперь `${^LAST_FH}` неопределён.

- `(...)x ...` в пустом контексте теперь применяет скалярный контекст к аргументу с левой стороны, вместо контекста, в котором вызвана текущая подпрограмма. [perl #123020].

Известные проблемы

- `pack` значения NaN в perl, скомпилированным в Visual C 6 происходит неправильно, приводя к ошибкам в тесте `t/op/infnan.t`. [perl 125203]
- Преимущество Perl в способности быть собранным для работы правильно на любой версии Юникода. Однако в Perl 5.22 самая ранняя такая версия Юникода это 5.1 (текущая версия 7.0).
- Платформы EBCDIC
  - `cmp` (и соответственно `sort`) операторы не всегда дают корректный результат, когда оба операнда UTF-EBCDIC кодированные строки и есть смесь ASCII и/или контрольных символов вместе с другими символами.
  - Диапазоны, содержащие `\N{...}` в `tr///` (и `y///`) операторах транслитерации рассматриваются по-разному, чем в эквивалентном диапазоне в шаблоне регулярного выражения. Они должны, но не вызывают рассмотрение всех значений в диапазонах ни как Юникод кодовые точки, и ни как нативные. (“Version 8 Regular Expressions” in perlre даёт представление о том, как это должно работать).
  - Encode и кодирование в большинстве случаев сломаны.
  - Многие CPAN модули, которые поставляются с базовым Perl демонстрируют падающие тесты.
  - `pack`/`unpack` с форматом `"U0"` может не работать правильно.
- Известно, что следующие модули имеют ошибки в тестах с этой версией Perl. Во многих случаях патчи были отправлены, поэтому надеемся, что вскоре последуют новые релизы:
  - B::Generate версии 1.50
  - B::Utils версии 0.25
  - Coro версии 6.42
  - Dancer версии 1.3130
  - Data::Alias версии 1.18
  - Data::Dump::Streamer версии 2.38
  - Data::Util версии 0.63
  - Devel::Spy версии 0.07
  - invoker версии 0.34
  - Lexical::Var версии 0.009
  - LWP::ConsoleLogger версии 0.000018

- Mason версии 2.22
- NgxQueue версии 0.02
- Padre версии 1.00
- Parse::Keyword версии 0.08

Некролог

Брайн МакКоули умер 8 мая 2015. Он часто писал в Usenet, Perl Monks и другие Perl форумы, и отправил несколько CPAN дистрибутивов под ником NOBULL, а также внёс вклад в Perl FAQ. Он участвовал практически в каждой YAPC::Europe и помогал организовать YAPC::Europe 2006 и QA Хакатон 2009. Его сообразительность и радость работы с запутанными системами особенно проявлялись в его любви к настольным играм; многие Perl-монгеры помнят как играли во Fluxx и другие игры вместе с Брайном. Его будет не хватать.

Благодарности

Perl 5.22.0 представляет собой примерно 12 месяцев разработки, начиная с Perl 5.20.0 и содержит примерно 590,000 строк изменений в 2,400 файлах от 94 авторов.

Исключая автогенерируемые файлы, документацию и утилиты релиза, было сделано примерно 370,000 строк изменений в 1,500 .pm, .t, .c и .h файлах.

Perl продолжает бурно развиваться в своей третьей декаде благодаря активному сообществу пользователей и разработчиков. Известно, что следующие люди содействовали в улучшении того, что стало Perl 5.22.0:

Aaron Crane, Abhijit Menon-Sen, Abigail, Alberto Simões, Alex Solovey, Alex Vandiver, Alexandr Ciornii, Alexandre (Midnite) Jousset, Andreas König, Andreas Voegele, Andrew Fresh, Andy Dougherty, Anthony Heading, Aristotle Pagaltzis, brian d foy, Brian Fraser, Chad Granum, Chris ‘BinGOs’ Williams, Craig A. Berry, Dagfinn Ilmari Mannsåker, Daniel Dragan, Darin McBride, Dave Rolsky, David Golden, David Mitchell, David Wheeler, Dmitri Tikhonov, Doug Bell, E. Choroba, Ed J, Eric Herman, Father Chrysostomos, George Greer, Glenn D. Golden, Graham Knop, H.Merijn Brand, Herbert Breunung, Hugo van der Sanden, James E Keenan,

James McCoy, James Raspass, Jan Dubois, Jarkko Hietaniemi, Jasmine Ngan, Jerry D. Hedden, Jim Cromie, John Goodyear, kafka, Karen Etheridge, Karl Williamson, Kent Fredric, kmx, Lajos Veres, Leon Timmermans, Lukas Mai, Mathieu Arnold, Matthew Horsfall, Max Maischein, Michael Bunk, Nicholas Clark, Niels Thykier, Niko Tyni, Norman Koch, Olivier Mengué, Peter John Acklam, Peter Martini, Petr Písař, Philippe Bruhat (BooK), Pierre Bogossian, Rafael Garcia-Suarez, Randy Stauner, Reini Urban, Ricardo Signes, Rob Hoelz, Rostislav Skudnov, Sawyer X, Shirakata Kentaro, Shlomi Fish, Sisyphus, Slaven Rezic, Smylers, Steffen Müller, Steve Hay, Sullivan Beck, syber, Tadeusz Sośnierz, Thomas Sibley, Todd Rinaldo, Tony Cook, Vincent Pit, Vladimir Marek, Yaroslav Kuzmin, Yves Orton, Ævar Arnfjörð Bjarmason.

Список выше конечно неполон, так как был автоматически сгенерирован из истории системы контроля версий. В частности, он не включает имена (очень высоко ценимых) помощников, которые сообщали о проблемах в Perl багтрекер.

Множество изменений, включённых в этой версии, идут от CPAN модулей, включённых в ядро Perl. Мы благодарны всему CPAN сообществу за помощь в развитии Perl.

Полный список всех принимавших участие в разработке в истории Perl смотрите пожалуйста в файле `AUTHORS` в дистрибутиве исходного кода Perl.

Сообщения об ошибках

Если вы найдёте то, что как вы считаете является ошибкой, вы можете проверить ранее опубликованные статьи в новостной группе comp.lang.perl.misc и базе ошибок perl на http://rt.perl.org/perlbug/. Также может быть информация на http://www.perl.org/, домашней странице Perl.

Если вы уверены, что у вас ещё ни кем не сообщённая ошибка, пожалуйста запустите программу perlbug, включённую в ваш релиз. Убедитесь, что вы привели максимально краткий, но достаточный пример, для воспроизведения проблемы. Ваш отчёт по ошибке, вместе с выводом `perl -V`, будет отправлен на адрес perlbug@perl.org для анализа командой портирования Perl.

Если ошибка, о котором вы сообщаете, имеет отношение к безопасности, что

делает его неуместным для отправки в публичную архивируемую почтовую рассылку, пожалуйста отправьте его на perl5-security-report@perl.org. Это неархивируемая почтовая рассылка с закрытой подпиской, которая включает всех главных коммитеров, и позволит скоординировать выпуск патча для смягчения или исправления проблемы на всех платформах, на которых поддерживается Perl. Пожалуйста используйте этот адрес только для проблем безопасности в базовом Perl, а не для модулей, которые распространяются на CPAN.

Смотрите также

Файл `Changes` для просмотра исчерпывающей информации о том, что изменилось.

Файл `INSTALL` о том, как собирать Perl.

Файл `README` для общей информации.

Файлы `Artistic` и `Copying` для информации по правам.

■ *Владимир Леттиев* (перев.)

8. Обзор CPAN за май 2015 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц

Статистика

- Новых дистрибутивов — 179
- Новых выпусков — 730

Новые модули

Yandex::Metrika

`Yandex::Metrika` — это модуль для доступа к одноимённому API Яндекса с использованием OAuth-авторизации. API-вызовы соответствуют методам объекта `Yandex::Metrika`, полный их список есть в документации Яндекс.Метрики.

Cache::Reddit

`Cache::Reddit` — это довольно забавная идея использовать в качестве кеша новостной сайт reddit. Данные сериализуются с помощью `Storable` и постятся как текстовый пост на субреддите, задаваемого в переменной окружения `reddit_subreddit`. Извлекаются данные путём поиска, поэтому сложность метода O(n), зато удаление происходит за постоянное время O(1).

pluskeys

Прагма `pluskeys` — это ещё один подход к созданию атрибутов для объектов, по подобию `fields`.

```
package Some::Package;
use pluskeys qw(
    NAME
    SURNAME
    AGE
);

...

if ( $self->{+NAME} =~ /Larry/ ) { ... }
```

В данном примере прагма `pluskeys` просто создаёт константы `NAME`, `SURNAME` и `AGE`, которые можно использовать как ключи атрибутов объекта. По специальному соглашению используется знак `+`, чтобы парсер Perl понял, что это константа, а не строка. Такой подход позволяет избежать опечаток.

HTTP::Tinyish

`HTTP::Tinyish` — это обёртка для модулей HTTP-клиентов `LWP`, `HTTP::Tiny`, а также программ curl и wget. Модуль основан на коде из `App::cpanminus` и может быть полезен в ограниченных окружениях, когда поддержка https отсутствует во встроенной HTTP-библиотеке.

Crypt::NaCl::Sodium

`Crypt::NaCl::Sodium` — это обёртка к современной криптографической библиотеке libsodium для (де)шифрования, создания и проверок подписи, хеширования и прочих операций. API совместимо с библиотекой NaCl.

Sub::Disable

Модуль `Sub::Disable` позволяет удалять методы и подпрограммы из скомпилированного кода. Например:

```
use Sub::Disable 'debug';

sub debug { warn "DEBUG INFO: @_" }

```

```
__PACKAGE__->debug(some_heavy_debug()); # no-op
debug(even_more(), heavier_debug());    # no-op
```

В процессе работы вызов метода/подпрограммы `debug()` превращается в пустую операцию вместе со всеми возможно тяжёлыми вычислениями в её аргументах. Это бывает полезно, чтобы отладочный код не замедлял работу приложения в рабочем окружении.

JSV

`JSV` — это реализация валидатора JSON-схемы, позволяющая проверять структуры данных, пересылаемых в формате JSON.

Disque

`Disque` — это Perl-клиент для Disque, распределённой системы очереди заданий. Модуль использует Redis.

Обновлённые модули

Hash::Ordered 0.09

Обновлён модуль `Hash::Ordered`, который позволяет создавать упорядоченные хеши. В новых релизах улучшена производительность при удалении из хеша с большим числом элементов, а также при извлечении элементов.

Marpa::R2 3.000000

Вышел новый мажорный релиз парсера `Marpa::R2`. Единственное изменение в релизе — это исправление проблемы со странным поведением вызова метода `$recce->value()` в случае, если передавался blessed аргумент. Поскольку

чей-то код мог использовать подобную особенность, то изменение ломало обратную совместимость. По этой причине и состоялся мажорный релиз.

Clustericious 1.00

Состоялся первый мажорный релиз `Clustericious` — веб-фреймворка для создания RESTful-сервисов, работающих в составе кластера, где каждый сервис выполняет только одну задачу и делает это идеально. Релиз содержит небольшое обновление документации и очевидно был предназначен только для того, чтобы зафиксировать стабильное состояния фреймворка.

Mail::GPG 1.0.9

После длительного перерыва обновился модуль `Mail::GPG` для работы с зашифрованными или заверенными цифровой подписью письмами. В новой версии появилась поддержка GnuPG 2.x.

Protocol::HTTP2 1.00

Вышла первая мажорная версия реализации новой версии протокола интернета HTTP2 `Protocol::HTTP2`. Выпуск приурочен к выходу официальной спецификации протокола RFC 7540.

XML::WBXML 0.09

Обновился модуль `XML::WBXML`, предназначенный для конвертации между форматами XML и WBXML. Модуль поменял майнтейнера, теперь это Михаил Иванов, который написал интересный пост о том, как это произошло.

Data::Printer 0.36

Вышел новый релиз модуля `Data::Printer`, который позволяет легко и быстро получить красивый вывод с подсветкой на терминал различных переменных, объектов и других структур данных. В новой версии появилась функция `np()`, которая вместо печати вывода на экран возвращает строку. В свою очередь, `p()` теперь всегда печатает свой вывод и возвращает параметр, который был передан функции. Это несовместимое изменение, будьте внимательны при обновлении.

■ *Владимир Леттиев*

9. Интервью с Рене Беккером

Рене Беккер (Renée Bäcker) — немецкий Perl-программист, выпускал журнал `$foo` *, активно участвует в немецком Perl-сообществе*

Когда и как научился программировать?

Моей первой программой был простой калькулятор, написанный на QBasic где-то в середине 90-х. С компьютером моего отца шла документация, к которой прилагался справочник команд для QBasic. Поэтому я заинтересовался, что же можно сделать с этим QBasic. Было очень просто решать базовые задачи. Но в то время меня это не сильно захватило.

Через несколько лет я изучил в школе Pascal. В основном базовые вещи, и все еще программирование меня не сильно очаровывало. После школы — в 2001 — я пошел на курсы ИТ-специалистов, где мы изучали Java в компании и C++ на курсах. Я проходил обучение в небольшой компании, занимающейся биоинформатикой, и много писал на Java. Через полгода я сменил работу и попал в департамент биоинформатики одной большой фармацевтической компании. Люди там были математиками, химиками, биологами, но не программистами. И они использовали Perl…

Именно в это время я и научился программировать. Я всегда учился путем проб и ошибок и с помощью Google ;-) У меня не очень получается читать книги по информатике, мне хочется больше что-то делать, чем читать. Когда мне нужно что-то реализовать, я думаю, как это можно сделать и пробую. Если ничего не получается, я читаю документацию, форумы и блоги по конкретной тематике и пытаюсь понять что (и почему) происходит.

Таким способом у меня лучше происходит обучение и я получаю более глубокие знания по сравнению с чтением книг.

Какой редактор используешь?

Я не привязан к какому-то конкретному редактору. Я использую то, что доступно… Когда я на Windows, я использую Kephra или Padre, на текущем проекте я использую Notepad++ и Komodo. Когда я на *NIX, я использую vi/vim.

Поэтому я не тот человек, чтобы участвовать в холиварах.

Когда и как познакомился с Perl?

Я уже упоминал в ответе не первый вопрос, что мое первое знакомство с Perl было с департаменте биоинформатики. Это было в 2002-м. Непрограммисты занимались программированием. Они писали довольно простые и небольшие Perl-скрипты.

Они не могли научить меня Perl, но они хотели, чтобы я писал на Perl, чтобы они как минимум могли читать мой код. Поэтому мне пришлось самому научиться. Это было несложно, документация у Perl была очень хорошей, и были такие форумы как http://perl.de (сообщество этого форума несколько лет назад основало http://perl-community.de) и Perlmonks.

Поэтому это было что-то вроде JDIWP (Just do it - with perl, Просто Сделай Это — На Перле) без какого либо руководства. Наверное я до сих пор и пользуюсь перлом, потому что на нем можно писать, особо не разбираясь в языке. Конечно, в скриптах было полно глобальных переменных, не было `strict` и `warnings` (но с этим я познакомился через неделю), я не знал всех идиом, но скрипты работали.

Я до сих пор учу что-то новое каждый день…

С какими другими языками интересно работать?

Очень редко я программирую на C/C++ и PHP, и мне это совсем не нравится. Также мне приходится часто писать на JavaScript, а в последнее время я начал посматривать на Rust. С такими фреймворками как jQuery JavaScript вполне ок. Можно довольно быстро делать многие вещи. И Rust выглядит многообещающе…

Что, по-твоему, является самым большим преимуществом Perl?

Сообщество, CPAN и обратная совместимость.

Сообщество, возможно, не для всех является преимуществом. Я знаю множество программистов, которые не участвуют в сообществе (в не зависимости от языка программирования). Они используют язык как инструмент и не более. По-моему, сообщество особенно важно для технологий, за которыми не стоит большая компания.

Без CPAN я бы не сделал столько всего, сколько сделал. Я хотел бы сказать «Спасибо» всем авторам отличных модулей на CPAN.

А обратная совместимость это то, что я часто слышу от своих клиентов. Им нравится Perl, они используют его достаточно давно, и обновления проходят без проблем.

Что, по-твоему, является самым важным свойством языков будущего?

Это непростой вопрос — я не знаю, что будет важным через 10 лет, но на сегодняшний день важным аспектом современных приложений является хостинг в облаках. И объемы данных, которые приходится обрабатывать, все растут и растут. Не думаю, что один язык должен занять обе ниши, но язык должен быть хорош хотя бы в одной из них.

Что думаешь о Perl 6?

Я не следил внимательно на разработкой Perl 6 до недавнего момента и использую Perl 6 только для небольших скриптов, я думаю у языка много отличных особенностей. В моей работе Perl 6 не заменит Perl 5 в течение ближайших лет, но для моих личных проектов я все больше и больше использую Perl 6.

Мы должны прекратить волноваться о влиянии Perl 6 на Perl 5 (например, было ли правильным назвать язык Perl 6, хотя он максимум является родственным языком; задержала ли работа над Perl 6 развитие Perl 5; ...). Сейчас нам стоит наслаждаться Perl 6 и его фишками.

Как и когда начал выпускать журнал `$foo`? Почему решил прекратить?

В 2005 я разговаривал со своим другом на немецком Perl-воркшопе о журнале на немецком языке. Мы решили, что такого журнала не хватает, и было бы хорошо, если бы кто-то этим занялся. Но так все и осталось в разговорах.

В 2006 на очередном немецком Perl-воркшопе мы поговорили на эту тему еще раз, но снова ничего не случилось.

Где-то в ноябре 2006 я подумал, что раз никто не начинает журнал, я должен это сделать. Я составил список тем и начал писать статьи. Я загрузил Scribus и начал оформлять первый номер... Я ничего не знал об издательском деле

и еще меньше о «дизайне» журнала. Но я все равно сделал это. Я не хотел тратить много денег на подобный эксперимент, поэтому все делал сам.

В феврале 2007 был напечатан первый номер журнала `$foo`. Я отправил его нескольким Perl-программистов, которых я знал, и один из них работал в компании, которая занималась дизайном, и они указали мне на все ошибки, которые я допустил. Создание шаблона журнала не так просто! В конце концов они сделали мне шаблон, и я отплатил им программированием.

Я сделал небольшой сайт и начал продавать через него журнал.

Я выпускал журнал в течение восьми лет и прекратил потому, что это занимало очень много времени. Публикация журнала совсем непростое дело. Вам нужны авторы, а поскольку журнал небольшой, вы не можете платить гонорары, вам нужны волонтеры. Вам нужно связаться с авторами, попросить написать статьи. Иногда может случитья так, что за неделю до выпуска автор говорит, что не сможет написать статью. И это нормально, потому что он пишет это ради забавы. Но для меня это означает, что нужно найти за неделю статью.

Хочу поблагодарить Герберта Бройнунга (Herbert Breunung), который был постоянным автором и написал множество статей.

После того как появились статьи, их нужно вычитать. Затем нужно составить номер, отправить его в печать, забрать из типографии и отправить читателям.

С 2007 года после первого выпуска моя жизнь сильно изменилась. Я женился, у меня сейчас двое сыновей. Вначале было просто писать множество статей, так как мои проекты затрагивали самые разнообразные темы. Но в последние годы становилось все сложнее и сложнее. Мне приходилось думать о новых темах, читать о них, тестировать код и писать статьи. И у меня была обычная работа. Все это не срасталось.

Мне нравится моя семейная жизнь, и я хочу проводить больше времени с моей женой и детьми. Поэтому я решил прекратить выпуск журнала. Это было тяжелое решение, потому что `$foo` был тоже моим ребенком.

Почему сделал perlybook.org?

В течение многих лет и использую Pod в качестве основного языка разметки. Для статей в `$foo`, для документации, для обучающих материалов и т.д. Это очень просто, и можно конвертировать в другие форматы. Во время поездок в командировки я хочу готовиться к тренингам, читать документаци и т.п., но я не хочу делать это со своего лаптопа. Для этого больше подходят электронные книги. Поэтому мне нужно было сконвертировать свои тексты и документаци CPAN-модулей в ebook-форматы.

Поэтому я написал небольшую консольную утилиту для конвертации. После того как я показал ее своим друзьям, они сказали, что было бы неплохо иметь такой онлайн-сервис, где можно было бы создавать электронные книги из CPAN-документации.

К счастью, есть MetaCPAN с отличным API, поэтому я переписал консольную утилиту в надлежащий модуль с поддержкой плагинов для разных источников, таких как локальные файлы, MetaCPAN, GitHub и т.п. и для целевых форматов, таких как текст, PDF и EPub. Этим и занимается модуль EPublisher. Затем написал небольшое веб-приложение вокруг этого модули и perlybook был готов.

Так что этот вебсайт начался как небольшая утилита, которая решала мою задачу…

Что такое perl-academy.de?

В прошлом многие клиенты хотели, чтобы я не только занимался программированием, но и обучал их работников самым разнообразным Perl-темам (начиная от таких базовых вещей как правильно использовать DBI и ссылки до ООП с использованием Moose). И также я делал несколько докладов на Perl- и не-Perl-мепроприятиях. Мне нравилось делать доклады и обучать людей.

Также я провел несколько курсов в университете, где учился, для нескольких компаний. Весь этот опыт привел к решению начать собственный бизнес. На http://perl-academy.de можно найти различные Perl-курсы, которые нельзя найти в других компаниях, занимающихся тренингами (в Германии).

В начале я предоставлял материалы на субдомене Perl-Services.de, но потом понял, что так их сложнее найти. Поэтому я зарегистрировал Perl-Academy.de. Это сделало курсы более заметными.

Проведение тренингов вносит интересные перемены в жизнь программиста.

Ты спонсировал или помогал организовывать Perl-стенды на различных компьютерных выставках в Германии. Что мотивирует этим заниматься?

Мне нравится Perl и мне нравится говорить о Perl. Я начал заниматься стендами в 2010 году во время CeBIT, большой IT-выставки. Я не помню, кому пришла в голову идея принять участие в спонсировании стенда в зале open source-проектов, но в конце концов у нас был стенд, и мы отлично пообщались с 400 людьми о перле.

Мы узнали, что множество компаний используют Perl, но используют его также как и в 2000-м или 1990-м. Они не знают, насколько Perl и его экосистема эволюционировали. Поэтому мы им рассказывали про современные практики, новые модули и о многом другом.

С тех пор я был на многих мероприятиях, где присутствовал Perl-стенд, и общался со многими людьми.

Мне кажется, что очень важно, что Perl-сообщество открыто к разговору с людьми вне своей песочницы. Это помогает понимать, что другие люди думают о Perl, и в чем они нуждаются.

Но если быть честным до конца, то моей мотивацией не всегда был альтруизм. Perl-стенд был хорошей маркетинговой площадкой для рекламы журнала `$foo` и моего собственного бизнеса.

Спонсирование же — совсем другая тема. Я спонсирую мероприятия, потому что хочу что-то отдать сообществу. В 2004 году я посетил свой первый немецкий Perl-воркшоп, и он помог мне стать частью сообщества. Эти встречи помогают узнать людей, стоящих за модулями, которые вы используете каждый день. Поэтому такие мероприятия как немецкий Perl-воркшоп должны присутсвовать. Спонсирование же помогает организаторам сохранять низкие цены на билеты.

По-моему, каждая компания, которая использует open source, должна что-то отдавать взамен. И спонсирование это самый простой способ это делать. К тому же, это маркетинг ;-)

Что делаешь для Frankfurt.pm?

На данный момент совсем немного… Я член совета директоров. Мы создали группу в 2011 году, когда начали организовывать YAPC::EU 2012. И в 2013 году мы взяли на себя ответственность помогать локальным группам организовывать немецкий Perl-воркшоп.

Так как я не могу посещать социальные встречи Frankfurt.pm, я занимаюсь только организационной частью.

Где сейчас работаешь, сколько времени проводишь за написанием Perl-кода?

Я до сих пор занимаюсь собственным Perl-бизнесом (плюс немного HTML, SQL и JavaScript). Я пишу на Perl от 8 до 10 часов в день, и мне это нравится. На сегодняшний день у меня есть долгосрочный проект с Deutsche Bahn (немецкая железная дорога), но также у меня много работы по доработке OTRS. OTRS это система обработки заявок (конкурент RT, с которым должны быть знакомы CPAN-авторы), написанная на Perl и особенно широко распространенная в Германии.

Стоит ли советовать молодым програмистам сейчас изучать Perl?

Да! Мы не должны принуждать никого изучать Perl, но должны советовать молодым программистам на него посмотреть. Они затем должны решить, что им нравится использовать.

Perl, возможно, старый язык, но не устаревший. Я провел несколько уроков в школе и спонсировал призы на соревновании «Jugend forsich» (исследовательская программа для молодежи). Некоторые подростки проявили талант при работе с перлом, и им это понравилось.

И у молодых людей есть новые идеи, у них нет лицеприятных мнений. Это может привести к отличным новым разработкам. Хорошо иметь свежие идеи в Perl-сообществе. И, возможно, однажды у нас появится следующее большое приложение, которое будут все использовать, и оно будет написано на Perl.

Вопросы от читателей

Будут ли опубликованы выпуски `$foo`?

Все выпуски `$foo` доступны в PDF-формате по адресу http://abo.perl-magazin.

de. Мы планируем опубликовать статьи в других форматах. Но это может занять некоторое время.

Почему Mojolicious?

Короткий ответ: это отличный веб-фреймворк.

Длинный ответ: я начал веб-программирование с простых CGI-скриптов. Затем я перешел на CGI::Application и CGI::Application::Dispatch. Некоторые мои программы до сих пор используют эти модули. Я также использовал Catalyst. Но для одного проекта — очень небольшого веб-приложения — я хотел попробовать что-то более современное, чем CGI или CGI::Application, а Catalyst был слишком громоздким.

У меня в голове было три альтернативы: Dancer, Mojolicious и голый Plack. Мой клиент не был программистом, и я не хотел оставлять его с проблемами с зависимостями. Этим мне и нравится в Mojolicious — никаких зависимостей, кроме Perl. И, @vti, это все твоя вина ;-) Ты написал о Mojolicious в своем блоге, и мне это показалось интересным.

Мне очень нравились все фишки, которые были включены в ядро Mojolicious, такие как поддержка Websocket. Я знаю, что поставлять все одним пакетом может считаться минусом. Только один человек и небольшая группа ответственна за фиксы багов.

Другой интересной стороной Mojolicious является Mojolicious::Lite. Я могу пользоваться одним фреймворком и для «больших» веб-приложений, и для небольших приложений на один экран. И мне не нужно создавать иерархию директорий для небольшого приложения. Достаточно просто написать один файл и его задеплоить…

В то же время количество плагинов постоянно растет, и легко разрабатывать веб-приложения любого калибра.

■ *Вячеслав Тихановский*