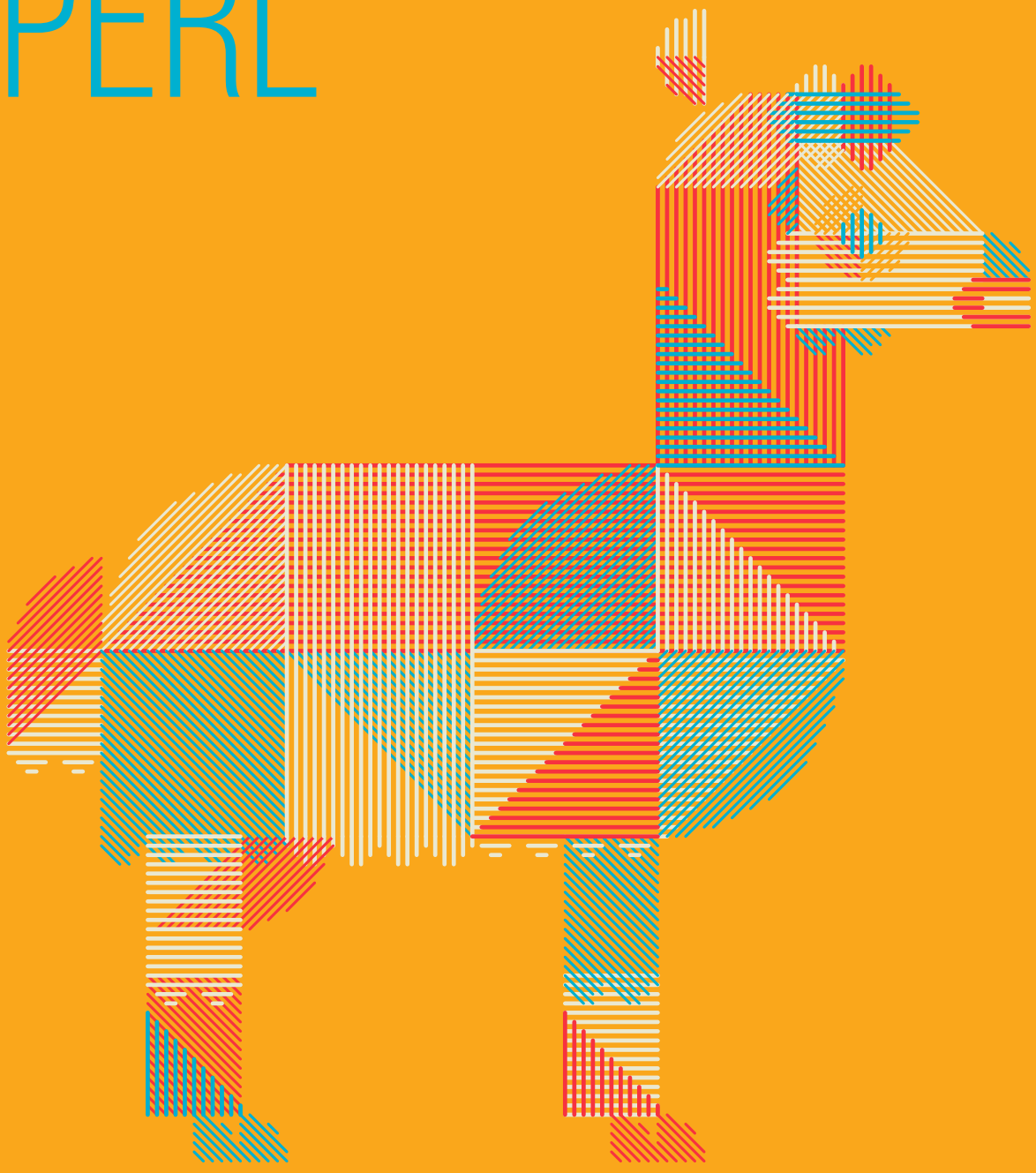
PRAGMATIC PERL

16



06/2014

pragmaticperl.com

Pragmatic Perl 16

pragmaticperl.com

Выпуск 16. Июнь 2014

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

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

Авторы статей: Владимир Леттиев, Андрей Шитов

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 23:30

© «Pragmatic Perl»

Оглавление

1. От редактора

Напоминаем, что конференция YAPC::Russia 2014 пройдет в в Санкт-Петербурге 13 и 14 июня (с пятницы по субботу). Спешите бронировать гостиницу и покупать билеты! С докладами можно ознакомиться на сайте.

Мы продолжаем искать авторов для следующих номеров. Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2. Синтаксические новинки в Perl 5.20

Рассказ о том, что нового появилось в синтаксисе.

27 мая вышел очередной стабильный релиз за номером 5.20. Подробное описание всех сделанных изменений доступно в файле perldelta, его перевод опубликован в этом номере журнала. В этой же статье я более подробно расскажу про изменения, которые были внесены в синтаксис языка.

Сигнатуры

(О сигнатурах журнал уже писал в одном из предыдущих выпусков.)

Формальные аргументы функций (подпрограмм sub) теперь возможно перечислять, причем вместе с именами, непосредственно в заголовке функции. Например:

```
1 sub fahrenheit2celsius($temperature) {
2     return 5 / 9 * ($temperature – 32);
3 }
```

Аргумент, переданный первым параметром, внутри тела функции становится доступным под именем `$temperature`. Дополнительно объявлять эту переменную не требуется (обратите внимание на отсутствие ключевого слова `my`).

Использование функции ничем не отличается от обычного способа:

```
1 say fahrenheit2celsius(100);
```

Важно помнить, что эта возможность считается экспериментальной, поэтому следует быть осторожным при ее использовании в ответственных приложениях. Экспериментальный статус вынуждает явно сообщать компилятору о своем намерении. Полный код работающей программы, которая компилируется и не выдает предупреждений, выглядит так:

```
1 use v5.20;
2
```

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

sub fahrenheit2celsius($temperature) {
   return 5 / 9 * ($temperature - 32);
}

say fahrenheit2celsius(100);
```

Директива `use feature 'signatures'` включает сигнатуры (несмотря на наличие `use v5.20`, автоматически этого не происходит), а `no warnings 'experimental::signatures'` подавляет сообщение `The signatures feature is experimental at signatures.pl line 6`.

При использовании сигнатур аргументы передаются по значению, поэтому внутри функции допустимо их изменять. Например, показанный пример возможно переписать следующим образом:

```
sub fahrenheit2celsius($temperature) {
   $temperature -= 32;
   return 5 / 9 * $temperature;
}

my $F = 100;
my $C = fahrenheit2celsius($F);
say "$F F = $C C";
```

Использование сигнатур никак не отражается на обработке массива `@_`. Поэтому вполне допустимо (хотя и непонятно, зачем) использовать его для доступа к переданным аргументам:

```
sub fahrenheit2celsius($temperature) {
   my ($t) = @_;
   return 5 / 9 * ($t - 32);
}
```

Использование оператора `shift` для чтения аргументов из массива `@_` тоже никак не затрагивает переменную `$temperature`, указанную в заголовке. Переданный аргумент исчезнет из `@_`, но останется в `$temperature`.

Однако, коль скоро функция объявлена с перечнем аргументов, все попытки ее использовать в коде будут проверяться на нали-

чие аргументов и их правильного числа. Несовпадение числа аргументов породит ошибку на этапе выполнения программы `Too many arguments for subroutine` или `Too few arguments for subroutine`.

Примечение. Ошибка возникает именно на этапе выполнения, а не при компиляции. В этом легко убедиться, напечатав что-нибудь до ошибочного вызова функции:

```
sub f($arg) {
    say "Never reached";
}

say "Start";
f(10, 20);
```

Аналогично, допустимо объявить и требовать при вызове пустой список параметров:

```
sub justcallme() {
    say "Called";
}
```

В этом случае безошибочным будет только вызов без аргументов: `justcallme();` или без скобок: `justcallme;`.

В списке параметров разрешено указывать значения по умолчанию (очевидно, что все они должны быть последними в списке, иначе компилятор в некоторых случаях не сможет разобраться, где переданы какие параметры). Синтаксис значений вполне очевидный:

```
sub print_envelope_label(
    $name, $last_name,
    $street, $house_number,
    $city = 'Moscow', $country = 'Russia') {

    say "$name $last_name\n$street, $house_number\n$city,
        $country";
}
```

Если в этом примере не указать город и страну, будет напечатана этикетка для конверта с адресом в Дефолт-сити:

```
print_envelope_label(
    'Konstantin', 'Konstantinopolsky',
```

```
    'Konstantinovskaya', 10,
    'Kiev', 'Ukraine'
);

print_envelope_label(
    'Ivan', 'Ivanov',
    'Tverskaya', 4
);
```

В качестве значений по умолчанию возможно использовать любые выражения (они будут вычисляться каждый раз в момент вызова функции), в том числе значения уже переданных аргументов.

В том случае, если необходимо передать произвольное число аргументов, допустимо воспользоваться переменной-списком или хешем:

```
sub numerate_list(@list) {
    my $c = 1;
    for my $value (@list) {
        say "$c. $value";
        $c++;
    }
}

numerate_list('alpha', 'beta', 'gamma');
```

Вариант с хешом:

```
sub dump_hash(%hash) {
    for my $key (sort keys %hash) {
        say "$key = $hash{$key}";
    }
}

dump_hash(a => 'alpha', b => 'beta', c => 'gamma');
```

Опять же, такие аргументы должны следовать в конце списка аргументов. Идеологически эти варианты похожи на традиционное чтение аргументов из массива `@_`.

Наконец, если в списке формальных аргументов окажутся переменные без имен (то есть голые сигилы `$`, `@` или `%`), то соответствующие аргументы окажутся обязательными, но внутри функции их значения по именам доступны не будут. Такая возможность окажется по-

лезной, например, при написании тестов или эмулировании некоего интерфейса:

```
sub mock_this($key, $, $value) {
    say "$key = $value";
}

mock_this(1, 2, 3);
```

Несмотря на то, что второй аргумент по имени недоступен, его по-прежнему возможно прочитать стандартным способом: `say $_[1]`.

Прототипы

Еще одно изменение, связанное с объявлением функций, — новое ключевое слово `prototype`. Синтаксис описания прототипа похож на синтаксис атрибутов функций:

```
use v5.20;

use feature 'signatures';

sub sum :prototype($$) {
    my ($x, $y) = @_;
    return $x + $y;
}

say sum(10, 20);
```

Похоже, что отдельное ключевое слово появилось из-за того, что включение в язык сигнатур сделало неоднозначным объявления типа `sub f($)`. С одной стороны, это функция, которая принимает один неиспользуемый скаляр (с точки зрения сигнатуры), а с другой — обязятельный скаляр (с точки зрения традиционных прототипов). В этом примере это одно и то же, но тем не менее, при наличии директивы `use feature 'signatures';` традиционный синтаксис описания прототипа (когда в скобках за именем функции без запятых или после точки с запятой перечисляются сигилы формальных параметров) перестает работать и компилятор сообщает об ошибке:

```
Parse error at prototype_error.pl line 6.
syntax error at prototype_error.pl line 6, near "$$) "
```

Прототипы в новом виде совместимы с сигнатурами. Показанный пример с функцией sum в этом случае будет выглядеть таким образом, что позволяет сэкономить одну строку кода:

```
sub sum :prototype($$) ($x, $y) {
    return $x + $y;
}
```

Именованные аргументы должны быть указаны только после объявления прототипа, непосредственно перед блоком с телом функции.

Казалось бы, указывать прототипы одновременно с сигнатурами избыточно. Однако, в некоторых случаях это может пригодиться. Например, функция с сигнатурой умеет разворачивать переданные ей списки. Поэтому, следующий пример полностью работоспособен, хотя функции передан массив вместо двух скаляров:

```
sub sum($x, $y) {
    return $x + $y;
}

my @values = (10, 20);
say sum(@values);
```

Уточнение `sub sum :prototype($$)($x, $y)` запрещает такое поведение. Если настаивать на передаче списка, то Perl сообщит об ошибке:

```
Not enough arguments for main::sum at prototype4.pl line
    15, near "@values)"
Execution of prototype4.pl aborted due to compilation
    errors.
```

Обратите внимание, что ошибка произошла во время компиляции (однако, безошибочный код в блоке BEGIN по-прежнему может быть выполнен).

Читателю предоставляется возможность самостоятельно разобраться в том, как изменились правила в описании прототипа, связанные с наличием пробелов, точек за запятой и звездочками.

Срезы с индексами

Массивы и хеши получили возможность рассказать о своей структуре, вернув список пар *ключ — значение* (для хешей) либо пар *индекс — значение* (для массивов). Документация описывает этот момент крайне скупо, а для того, чтобы заметить новинку, потребуется внимательность.

Новое заключается в том, что используется обычный синтаксис среза, но при этом на месте сигила стоит символ %. И для хешей, и для массивов. После сигила следует имя соответствующей переменной, а затем в фигурных или квадратных скобках перечисляются ключи или индексы. Например:

```
1 my %suffix = (pl => 'Perl', pm => 'Perl module', p6 => '
    Perl 6');
2 my @data = %suffix{'pl', 'p6'};
3 say Dumper(\@data);
4
5 my @greek = qw(alpha beta gamma delta epsilon);
6 @data = %greek[1, 2];
7 say Dumper(\@data);
```

Выражение `%suffix{'pl', 'p6'}` вернет список ключей и значений хеша `%suffix`:

```
1 $VAR1 = [
2           'pl',
3           'Perl',
4           'p6',
5           'Perl 6'
6         ];
```

А выражение `%greek[1, 2]` — индексы и соответствующие значения:

```
1 $VAR1 = [
2           1,
3           'beta',
4           2,
5           'gamma'
6         ];
```

Если намеренно или ошибочно поставить сигил списка: `@suffix{'`

`pl', 'p6'}` и `@greek[1, 2]`, то программа останется синтаксически верной, но работать будет совершенно иначе:

```
$VAR1 = [
          'Perl',
          'Perl 6'
        ];

$VAR1 = [
          'beta',
          'gamma'
        ];
```

(Впрочем, ошибочный сигил `$` в обоих случаях тоже будет успешно скомпилирован, даже включенный `use warnings` найдет не все.)

■ *Андрей Шитов*

3. Что нового в Perl 5.20.0

27 мая 2014 года была выпущена новая стабильная версия языка программирования Perl 5.20.0. Разработка велась примерно 12 месяцев, начиная с Perl 5.18.0, и содержит примерно 470 000 изменённых строк среди 2900 файлов от 124 авторов.

Ключевые изменения

Среди большого списка изменений можно выделить ключевые:

- Сигнатуры функции. Теперь появилась экспериментальная возможность указывать сигнатуру функции, которая инициализирует лексические переменные функции и присваивает им значения переданных аргументов, при этом осуществляется контроль числа переданных аргументов.

```
sub sum ($x, $y) {
    return $x + $y
}
```

Подробно о сигнатурах функции в Perl 5.20 было рассказано в 13 выпуске Pragmatic Perl.

- Постфиксное разыменование. Появилась экспериментальная возможность новой постфиксной записи для разыменования ссылок в Perl. Теперь следующие записи эквивалентны:

```
# Циркумфиксное разыменования
push @{ $href->{key}->{subkey} },
    keys %{ $aref->[$num] };

# Постфиксное разыменование
push $href->{key}->{subkey}->@*,
    keys $aref->[$num]->%* ;
```

Подробно о постфискном разыменовании было рассказано в 9 выпуске Pragmatic Perl.

- Новый синтаксис среза. Новый синтаксис позволяет получить срез ключей/значений для хеша или срез индекс/значение для массива:

```
# срез хеша
%hash{"key1", "key2"} # "key1", "value1", "key2", "
    value2"

# срез массива
%array[1,2,3]  # 1, "value1", 2, "value2", 3, "
    value3"
```

- Включён по умолчанию режим *COW* (копирование-при-записи) для строк. Это означает, что при присвоении одному скаляру значения другого, копирования буфера строки сразу не происходит. Это значительно повышает производительность и снимает необходимость передачи аргументов функций по ссылке (если они не будут изменяться).

- Unicode 6.3. Поддерживает последняя версия стандарта Юникод 6.3.

- Использование Perl-тредов (`use threads`) теперь официально *не* рекомендуется.

- Лучшая поддержка 64-битных плафторм. Теперь массивы могут содержать больше 2**31 элементов, а регулярные выражения работают со строками с более чем 2**31 символов.

- Официально поддерживается сборка на платформе Android.

- Генератор случайных чисел `rand()` теперь использует внутреннюю реализацию 48-битного генератора `drand48()` на всех поддерживаемых платформах. Это, однако, не означает, что генератор является криптографически безопасным.

- Множество других улучшений и исправлений

perldelta

Полный список изменений описан в perldelta.pod. Далее представлен перевод этого документа на русский язык. Оригинальный файл перевода в формате POD доступен на github.

Имя

perl5200delta — что нового в perl v5.20.0

Описание

Этот документ описывает изменения между релизом 5.18.0 и 5.20.0.

Если вы обновляетесь с более раннего релиза, как например, 5.16.0, сначала прочтите perl5180delta, который описывает отличия между 5.16.0 и 5.18.0.

Базовые улучшения

Экспериментальные сигнатуры функции

Декларативный синтаксис для разворачивания списка в лексические переменные. `sub foo ($a,$b){...}` проверяет число аргументов и помещает аргументы в лексические переменные. Сигнатуры не являются эквивалентом существующей идиоме `sub foo { my( $a,$b)= @_; ... }`. Сигнатуры доступны только при включении не используемой по умолчанию возможности и генерируют предупреждения о своей экспериментальности. Синтаксический конфликт с прототипами решён за счёт отключения короткой формы записи прототипа, когда сигнатуры включены.

Смотрите детали в “Signatures” in perlsub.

`sub` теперь может иметь атрибут `prototype`

При объявлении или создании `sub` прототип теперь может быть указан внутри атрибута `prototype` вместо записи его в скобках после имени.

Например, `sub foo($$){}` может быть записано как `sub foo : prototype($$){}`.

Более последовательный разбор прототипа

Раньше позволялось использовать несколько символов точки с запятой в прототипах подпрограмм и они рассматривались как одна точка с запятой. Но был один случай, в котором этого не происходило. Подпрограммы, чьи прототипы начинались с "*" или ";*" могли влиять на то, рассматривается ли простое слово именем метода или вызовом подпрограммы. Теперь тоже самое относится и к ";;;*".

Пробельные символы долгое время были разрешены внутри прототипов подпрограмм, таким образом `sub( $ $ )` было эквивалентно `sub($$)`, но до текущего времени они удалялись при лексическом разборе. Поэтому пробельные символы *не* были разрешены в прототипах задаваемых с помощью `Scalar::Util::set_prototype`. Теперь они разрешены и парсер больше не удаляет пробельные символы. Это означает, что `prototype &mysub` возвращает оригинальный прототип с пробелами и всем остальным.

`rand` теперь использует согласованный генератор случайных чисел

Ранее perl использовал специфичный для каждой платформы генератор случайных чисел, варьируясь между libc rand(), random() или drand48().

Это означало, что качество генератора случайных чисел менялось от платформы к платформе, начиная с 15-битного rand() на Windows

до 48-битного на POSIX платформах, таких как Linux с drand48().

Perl теперь использует свой собственную внутреннюю реализацию drand48() на всех платформах. Но это не делает `rand` перла криптографически безопасным. [perl #115928]

Новый синтаксис среза

Новый синтаксис `%hash{...}` и `%array[...]` возвращает список пар ключей/значений (или индексов/значений). Смотрите “Key/Value Hash Slices” in perldata.

Экспериментальное постфиксное разыменование

Когда включена возможность `postderef`, устанавливаются следующие синтаксические эквивалентности:

```
$sref->$*;  # то же, что и ${ $sref }  # интерполяция
$aref->@*;  # то же, что и @{ $aref }  # интерполяция
$href->%*;  # то же, что и %{ $href }
$cref->&*;  # то же, что и &{ $cref }
$gref->**;  # то же, что и *{ $gref }

$aref->$#*; # то же что и $#{ $aref }

$gref->*{ $slot }; # то же, что и *{ $gref }{ $slot }

$aref->@[ ... ];  # то же, что и @$aref[ ... ]  # интерполяция
$href->@{ ... };  # то же, что и @$href{ ... }  # интерполяция
$aref->%[ ... ];  # то же, что и %$aref[ ... ]
$href->%{ ... };  # то же, что и %$href{ ... }
```

Те, что отмечены, как интерполяция, интерполируют только при включении возможности `postderef_qq`. Эта возможность является **экспериментальной** и приводит к выводу предупреждений категории `experimental::postderef` при использовании, если только их вывод не подавлен.

Для большей информации, смотрите в the Postfix Dereference Syntax section of perlref.

Теперь поддерживается Юникод 6.3

Perl теперь поддерживает и поставляется с Юникодом 6.3 (хотя Perl может быть компилирован с поддержкой любого предыдущего релиза Юникода). Детальный список изменений в Юникоде 6.3 доступен в http://www.unicode.org/versions/Unicode6.3.0/.

Новое свойство шаблонов регулярных выражений `nicode\pU`

Это синоним для `\p{Any}` и совпадает с набором определённых в Юникоде кодов символом `0` – `0x10FFFF`.

Лучшая поддержка 64-битных платформ

На 64-битных платформах, внутренние функции массивов теперь используют 64-битные отступы, позволяя массивам Perl содержать больше 2**31 элементов, если у вас доступен такой объём памяти.

Движок регулярных выражений теперь поддерживает строки длиннее чем 2**31 символов. [perl #112790, #116907]

Функции PerlIO_get_bufsiz, PerlIO_get_cnt, PerlIO_set_cnt и PerlIO_set_ptrcnt теперь имеют тип SSize_t вместо int у возвращаемых значений и параметров.

`use locale` теперь работает на UTF-8 локалях

До этого выпуска поддерживались только однобайтные локали, такие как серия ISO 8859. Теперь всё более и более типичные мультибайтные UTF-8 локали также поддерживаются. UTF-8 локаль –

это такая локаль, которая имеет набор символов Юникод и кодировку UTF-8. POSIX-категория операций `LC_CTYPE` (изменение регистра (как `lc()`, `"\U"`) и классификация символов (`\w`, `\D`, `qr/[[:punct:]]/`)) под такой локалью работает как если бы была включена возможность `use feature 'unicode_strings'`, за тем исключением если выполняются правила taint. Сортировка остаётся в порядке возрастания кода символа в этом релизе. [perl #56820].

`use locale` теперь компилируется на системах без поддержки локалей

Ранее подобное выражение приводило к невозможности компиляции. В пределах действия прагмы программа ведёт себя как если была установлена локаль “C”. Таким образом, программы, написанные с поддержкой локалей, могут быть запущены на платформах без поддержки локалей без изменений в коде. Попытка поменять значение локали на отличное от “C” разумеется будет безуспешной.

Больше запасных вариантов для инициализации локали

Если происходила ошибка с локалями при запуске Perl, то он сразу сдавался и пытался использовать локаль `"C"`. Теперь он сначала пытается использовать другие локали, предоставленные переменными окружения, как описано в “ENVIRONMENT” in perllocale. Например, если обе `LC_ALL` и `LANG` установлены, и использовать локаль из `LC_ALL` не удалось, Perl теперь попытается использовать локаль из `LANG`, и если эта попытка тоже будет безуспешной, он попытается откатиться на `"C"`. На Windows машинах, Perl перед использованием `"C"`, попытается использовать системную локаль по умолчанию, если все локали из переменных окружения не заработали.

Опция запуска –DL была добавлена для отслеживания установок локали

Она предназначена для помощи разработчикам базового Perl в отладке ошибок, связанных с локалью.

-F теперь подразумевает -a, а -a подразумевает -n

Раньше **-F** без **-a** не давала никакого эффекта, также как и **-a** без **-n** или **-p**. С этим изменением если вы указываете **-F**, то обе **-a** и **-n** подразумеваются, а если вы указываете **-a**, то подразумевается **-n**.

Вы по-прежнему можете указывать **-p** для его дополнительного действия. [perl #116190]

Удаление предупреждения для $ab

Специальные переменные $ab, используемые в `sort`, теперь не приводят к предупреждениям “used once” (использованы один раз), даже если они использованы вне `sort`. Это позволит CPAN модулям предоставлять функции, использующие $ab для подобных же целей. [perl #120462]

Безопасность

Избежание возможного чтения освобождённой памяти при синтаксическом разборе

Раньше существовала возможность, что освобождённая память могла быть прочитана при синтаксическом разборе в необычных случаях, когда Perl программа заканчивалась встроенной документацией и последняя строка файла на диске не имела завершающего символа переноса строки. Это теперь исправлено.

Несовместимые изменения

`do` больше нельзя использовать для вызова подпрограммы

Форма вызова `do SUBROUTINE(LIST)` выводила предупреждение об устаревшей конструкции, начиная с Perl v5.0.0 и теперь это синтаксическая ошибка.

Изменения в экранировании в выражениях в кавычках

Символ после `\c` в строках с двойными кавычками ("..." или qq(...)) или в регулярных выражениях теперь должен быть печатным символом и не может быть `{`.

Литера `{` после `\B` или `\b` теперь фатальная ошибка.

Это являлось устаревшей конструкцией в perl v5.14.0.

"Заражение" теперь происходит в большем числе случаев; теперь соответствует документации

Это затрагивает регулярные выражения совпадающие и меняющие регистр строк (`lc`, `"\U"`, *etc.*) внутри области действия `use locale`. Результат теперь "заражён" (tainted) в соответствии с операцией независимо от содержимого там, где документация (perlsec, "SECURITY" in perllocale) указывает, что он должен. Раньше для операций изменения регистра, если строка содержала символы, чей регистр зависит от локали, результирующая строка не была "заражена". Например, результат `uc()` на пустой строке или содержащей только коды символов вне пределов Latin1 теперь "заражён", чего не было раньше. Это ведёт к более согласованным результатам "заражения". Шаблоны регулярных выражений "заражают" свой небинарный результат (как `$&`, `$2`) тогда и только тогда, когда шаблон содержит элементы, чьё совпадение зависит от текущей (возможно "заражённой") локали. Также как и в функциях изменяющих регистр, актуальное

содержимое строки, в которой ищется совпадение, теперь не имеет значение, в то время как раньше имело. Например, если шаблон содержит \w, результат будет "заражён", даже если поиск совпадения не использовал эту порцию шаблона для успешного или неуспешного поиска, поскольку совпадение \w зависит от локали. Однако, например, . в шаблонах не будет включать "заражение", поскольку точка совпадает любым одиночным символом и текущая локаль никак не может поменять результат поиска совпадения.

Поиск совпадений изменился для не Юникодных кодов символов

\p{} и \P{} определены в Юникоде только для определённых в Юникоде кодов символов (от U+0000 до U+10FFFF). Их поведение при поиске совпадений для легальных Юникод кодов символов не изменилось, но есть изменения для кодовых точек от 0x110000 и выше. Ранее Perl рассматривал результат совпадения с \p{} и \P{} с такими символами как undef, который интерпретировался как "ложь". Для \P{} это позже устанавливалось в "истину". Также предполагалось, что будет выдаваться предупреждение для таких случаев. Однако, различные оптимизации могли предотвращать появление предупреждения и часто приводили к нелогичным результатам как для поиска совпадения, так и кажущейся ложности обратного условия. Теперь все не Юникод коды символов рассматриваются как типичные коды символов, которые ещё не получили своё определение в Юникоде. Это в целом больше соответствует духу "делай то, что я имел ввиду". Предупреждение выводится только если результат может отличаться от строго следования стандарту Юникод. Код, который должен строго следовать стандарту Юникод может сделать это предупреждение фатальным и в таком случае Perl всегда будет выводить предупреждение.

Детали в "Beyond Unicode code points" in perlunicode.

`ll\pA` был расширен для совпадения со всеми возможными кодами символов

Определённый в Perl элемент шаблона регулярного выражения `\p{All}`, неиспользуемый на CPAN, раньше совпадал только с кодами символов Юникода, теперь он совпадает со всеми возможными кодами символов, таким образом он эквивалентен `qr/./s`. Следовательно `\p{All}` больше не является синонимом `\p{Any}`, который продолжает совпадать только с кодами символов Юникод, как и предписывает стандарт Юникод.

Вывод Data::Dumper может измениться

В зависимости от выводимой структуры данных и настройках Data::Dumper, вывод может отличаться от предыдущих версий.

Если у вас есть тесты, которые зависят от точного вывода Data::Dumper, то они могут упасть.

Чтобы избежать подобной проблемы в вашем коде, тестируйте структуру на совпадение с помощью вычисленного в eval дампа структуры, вместо самого дампа.

Символ десятичной запятой текущей локали больше не просачивается за пределы области видимости `use locale`

На самом деле это исправление бага, но есть код, который зависит от присутствия этого бага, поэтому это изменение указано здесь. Текущая локаль, под которой запущена программа, не должны быть видна для Perl кода за исключением области видимости `use locale`. Однако, до текущего момента при некоторых условиях символ, который используется для десятичной запятой (часто это запятая), просачивался вне этой области видимости. Если ваш код затронут этим изменением, просто добавьте `use locale`.

Присвоение кодов ошибки переменной `$!` для сокетов на Windows теперь отдаёт приоритет значениям `errno.h` над значениями WSAGetLastError()

В предыдущих версиях Perl, коды ошибок для Windows сокетов, возвращаемые WSAGetLastError(), присваивались `$!`, и некоторые константы, такие ка ECONNABORTED, отсутствующие в `errno.h` в VC++ (или различных Windows портах gcc) были определены в соответствующие WSAE* значения, чтобы иметь возможность протестировать `$!` на соответствие константам E*, экспортируемые Errno и POSIX.

Это работало пока в VC++ 2010 и более поздних, которые представили новые E* константы со значениями > 100 в `errno.h`, включая некоторые значения, которые были переопределены в perl к WSAE* значения. Это привело к проблемам при линковке XS кода с библиотеками, которые используют оригинальные определения констант из `errno.h`.

Чтобы избежать поломки существующего Perl кода, который присваивает WSAE* значения в `$!`, perl теперь перехватывает присвоение и производит такое же соответствие E* значениям, какие использует внутри при присвоении самой `$!`.

Однако, одна обратная несовместимость остаётся: существующий Perl код, который сравнивает `$!` с числовым значением кода ошибки WSAE*, которые раньше были присвоены в `$!`, теперь окажется сломанным в тех случаях, когда соответствующее E* значение было присвоено вместо него. Это становится проблемой для E* значений < 100, которые всегда экспортировались из Errno и POSIX с их оригинальными `errno.h` значениями, и поэтому не могли быть использованы для тестов кодов ошибок WSAE* (например, WSAEINVAL равен 10022, но соответствующее EINVAL равно 22). (E* значения > 100, при наличии, всё равно переопределяются в WSAE* значения, поэтому совместимость может быть достигнута при использовании E* констант, которые работали как до, так и после этого изменения, хотя и используя уже другие численные значения под капотом.)

Функции `PerlIO_vsprintf` и `PerlIO_sprintf` были удалены

Эти две функции, недокументированные, неиспользуемые на CPAN и имеющие проблемы, были удалены.

Устаревшие конструкции

Класс символов `/C/`

Класс символов `/\C/` в регулярных выражениях устарел. Начиная с perl 5.22 это будет выдавать предупреждение, а с perl 5.24 - станет ошибкой при компиляции регулярного выражения. Если вам требуется проверить отдельные байты, которые составляют UTF-8 символ, тогда используйте предварительно `utf8::encode()` на строке (или копии).

Управляющие символы в именах переменных

Устаревшими становятся вещи наподобие
cT,
$cT - -(`NAK``NEGATIVEACKNOWLEDGE`).,$, ^T, со знаком вставки добавленным только как альтернатива.

Форма управляющих символов становится устаревшей по двум основным причинам. Это невозможность исправления багов, таких как
cI^I, и их использование не переносимо на не-ASCII платформы: в то время как $^T будет работать везде, \cT является пробелом в EBCDIC. [perl #119123]

Ссылка на нецелые или отрицательные целые в `$/`

Присвоение `$/` ссылки на ноль или ссылки на отрицательное целое теперь считается устаревшим, и будет вести себя **в точности**

как присвоение ей значения `undef`. Если вам нужно slurp поведение устанавливайте `$/` в `undef` явно.

Присвоение `$/` ссылки на нецелочисленное значение теперь запрещено и будет приводить к ошибке. Perl никогда не документировал что будет происходить в данном контексте и поскольку раньше это соответствовало установке `$/` в адрес ссылки, а в будущем это может начать вести себя по-другому, мы установили запрет на подобное использование.

Функции поиска совпадения символов в POSIX

Использование следующих функций в `POSIX` модуле теперь считается устаревшим: `isalnum`, `isalpha`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper` и `isxdigit`. Функции кишат багами и не работают с UTF-8 кодированными строками. Смотрите их описание в POSIX для более подробной информации.

Будет выводиться предупреждение при первом вызове любой из них из любого места кода, в котором они вызываются. (Следовательно повторная инструкция в цикле будет выводить только одно предупреждение).

Треды на базе интерпретатора теперь *не рекомендуются*

“Треды на базе интерпретатора”, предоставляемые Perl не являются быстрой, легковесной системой для многозадачности как многие ожидают и надеются. Треды реализованы так, что очень легко их использовать неправильно. Немногие знают как использовать их правильно или способны помочь другим.

Использование тредов на базе интерпретатора в perl официально не рекомендуется.

Удаление модулей

Следующие модули будут удалены из базового дистрибутива в будущем релизе и должны будут устанавливаться со CPAN. Дистрибутивы на CPAN, которым требуются эти модули, должны указать их в своих списках зависимостей.

Версии модулей из базового дистрибутива теперь будут выдавать предупреждения категории "устаревшие", извещая вас об этом факте. Для подавления этих предупреждений установите эти модули со CPAN.

Обратите внимание, что запланированное удаление этих модулей из ядра не отражает суждение о качестве их кода и не намекает на остановку их разработки. Их удаление из ядра связано с необходимостью в бутстрапе для получения полнофункциональной CPAN-совместимой инсталляции Perl, и не касается их дизайна.

- CGI и связанные CGI:: пакеты
- inc::latest
- Package::Constants
- Module::Build и связанные Module::Build:: пакеты

Удаление утилит

Следующие утилиты будут удалены из базового дистрибутива в будущем релизе и должны будут устанавливаться со CPAN.

- find2perl
- s2p
- a2p

Улучшения в производительности

- Perl имеет новый механизм копирования-при-записи, который избегает необходимости копирования внутреннего буфера строки при присвоении одного скаляра другому. Это значительно ускорит наблюдаемый процесс копирования больших строк. Изменение одной из двух (или большего числа) строк после присвоения приведёт к вынужденному внутреннему копированию. Что делает ненужным передачу строк по ссылке для эффективности.

  Данная возможность уже присутствовала в 5.18.0, но она не была включена по умолчанию. Теперь она доступна по умолчанию, поэтому вам больше не требуется собирать perl с аргументом `Configure`:

  ```
  1 -Accflags=-DPERL_NEW_COPY_ON_WRITE
  ```

  Но может быть отключена (теперь) при сборке perl с:

  ```
  1 -Accflags=-DPERL_NO_COW
  ```

  На некоторых операционных системах Perl может быть собран так, что любая попытка модификации буфер строки разделённого с разными SV приведёт к краху. Таким образом авторы XS модулей могут проверить, что их модули могут работать с копируемых-при-записи скаляров корректно. Смотрите подробности в “Copy on Write” in perlguts.

- Perl имеет оптимизатор для шаблонов регулярных выражений. Он анализирует шаблоны для поиска такие вещи как минимальная длина строки для совпадения и т.д. Теперь он лучше поддерживает коды символов за пределами диапазона Latin1.

- Выполняя регулярное выражение, которое содержит якорь `^` (или его вариации при наличии флага `/m`), теперь работает значительно быстрее в некоторых ситуациях.

- Заранее вычисленные значения хеша теперь используются в больших местах при поиске метода.

- Поиск по ключу-константе хеша (`$hash{key}` в отличии от `$hash{$key}`) давно использует заранее вычисленное внут-

реннее значение хеша, чтобы ускорить поиск. Подобная оптимизация теперь применена и к срезам хеша.

- Комбинированные операторы `and` и `or` в пустом контексте, как например, в `unless ($a && $b)` и `if ($a || $b)`, теперь сразу переходят к концу выражения. [perl #120128]

- В тех ситуациях, когда `return` это последний оператор в основной области видимости подпрограммы, он оптимизируется. Это значить, что код вида:

  ```
  sub baz { return $cat; }
  ```

  будет вести себя как:

  ```
  sub baz { $cat; }
  ```

  что заметно быстрее.

  [perl #120765]

- Код вида:

  ```
  my $x; # or @x, %x
  my $y;
  ```

  теперь оптимизируется к:

  ```
  my ($x, $y);
  ```

  В комбинации с оптимизацией padrange появившейся в v5.18.0, это означает, что и более длинный список неинициализированных переменных оптимизируется:

  ```
  my $x; my @y; my %z;
  ```

  становится:

  ```
  my ($x, @y, %z);
  ```

  [perl #121077]

- Создание некоторых видов списков, включая массивы или срезы хешей, теперь выполняется быстрее.

- Оптимизация для массивов адресуемых небольшими постоянными целыми числами теперь применяется для целых в диапазоне -128..127, а не 0..255. Это должно ускорить Perl код, который использует выражения вида `$x[-1]`, в ущерб (значительно более редко используемому) коду, использующий выражения `$x[200]`.

- Первая итерация над длинным хешем (используя `keys` or `each`) теперь выполняется быстрее. Это достигнуто за счёт предварительного выделения состояния внутреннего итератора хеша, вместо его ленивого создания, когда он обходится в первый раз. (Для маленьких хешей итератор по-прежнему создаётся при первой необходимости. Предполагается, что маленькие хеши используются для объектов и потому никогда не создаются. Для больших хешей, это более вероятно, и цена создания итератора поглощается стоимостью выделения места под сам хеш.)

- При выполнении глобального поиска совпадений в регулярном выражении в строке, которая приходит от `readline` или оператора `<>`, данные больше бесполезно не копируются. [perl #121259]

- Разыменование (как в `$obj->[0]` или `$obj->{k}`) теперь происходит быстрее, когда `$obj` является экземпляром класса, который имеет перегруженные методы, но не перегружает ни одного из методов разыменования `@{}`, `%{}` и подобных.

- Оптимизатор Perl больше не пропускает оптимизацию кода, который следует за некоторыми `eval {}` выражениями (включая те, которые похожи на бесконечные циклы).

- Реализация теперь ещё лучше избегает бессмысленной работы в режиме исполнения. Внутренние бесполезные “null” операции (созданные как побочный эффект при синтаксическом разборе Perl программы) обычно удаляются во время компиляции. Это удаление теперь применяется в некоторых ситуациях, которые раньше не обрабатывались.

- Perl теперь меньше нагружает дисковую I/O подсистему, когда работает со свойствами Юникода, которые покрывают до трёх последовательных диапазонов кодов символов.

Модули и прагмы

Новые модули и прагмы

- experimental 0.007 был добавлен базовую поставку Perl.
- IO::Socket::IP 0.29 был добавлен в базовую поставку Perl.

Обновлённые модули и прагмы

- Archive::Tar был обновлён с версии 1.90 до 1.96.
- arybase был обновлён с версии 0.06 до 0.07.
- Attribute::Handlers был обновлён с версии 0.94 до 0.96.
- attributes был обновлён с версии 0.21 до 0.22.
- autodie был обновлён с версии 2.13 до 2.23.
- AutoLoader был обновлён с версии 5.73 до 5.74.
- autouse был обновлён с версии 1.07 до 1.08.
- B был обновлён с версии 1.42 до 1.48.
- B::Concise был обновлён с версии 0.95 до 0.992.
- B::Debug был обновлён с версии 1.18 до 1.19.
- B::Deparse был обновлён с версии 1.20 до 1.26.
- base был обновлён с версии 2.18 до 2.22.
- Benchmark был обновлён с версии 1.15 до 1.18.
- bignum был обновлён с версии 0.33 до 0.37.
- Carp был обновлён с версии 1.29 до 1.3301.
- CGI был обновлён с версии 3.63 до 3.65. Внимание: CGI устарел и может быть удалён из будущей версии Perl.
- charnames был обновлён с версии 1.36 до 1.40.
- Class::Struct был обновлён с версии 0.64 до 0.65.
- Compress::Raw::Bzip2 был обновлён с версии 2.060 до 2.064.
- Compress::Raw::Zlib был обновлён с версии 2.060 до 2.065.
- Config::Perl::V был обновлён с версии 0.17 до 0.20.
- constant был обновлён с версии 1.27 до 1.31.
- CPAN был обновлён с версии 2.00 до 2.05.
- CPAN::Meta был обновлён с версии 2.120921 до 2.140640.
- CPAN::Meta::Requirements был обновлён с версии 2.122 до 2.125.

- CPAN::Meta::YAML был обновлён с версии 0.008 до 0.012.
- Data::Dumper был обновлён с версии 2.145 до 2.151.
- DB был обновлён с версии 1.04 до 1.07.
- DB_File был обновлён с версии 1.827 до 1.831.
- DBM_Filter был обновлён с версии 0.05 до 0.06.
- deprecate был обновлён с версии 0.02 до 0.03.
- Devel::Peek был обновлён с версии 1.11 до 1.16.
- Devel::PPPort был обновлён с версии 3.20 до 3.21.
- diagnostics был обновлён с версии 1.31 до 1.34.
- Digest::MD5 был обновлён с версии 2.52 до 2.53.
- Digest::SHA был обновлён с версии 5.84 до 5.88.
- DynaLoader был обновлён с версии 1.18 до 1.25.
- Encode был обновлён с версии 2.49 до 2.60.
- encoding был обновлён с версии 2.6_01 до 2.12.
- English был обновлён с версии 1.06 до 1.09.
- Errno был обновлён с версии 1.18 до 1.20_03.
- Exporter был обновлён с версии 5.68 до 5.70.
- ExtUtils::CBuilder был обновлён с версии 0.280210 до 0.280216.
- ExtUtils::Command был обновлён с версии 1.17 до 1.18.
- ExtUtils::Embed был обновлён с версии 1.30 до 1.32.
- ExtUtils::Install был обновлён с версии 1.59 до 1.67.
- ExtUtils::MakeMaker был обновлён с версии 6.66 до 6.98.
- ExtUtils::Miniperl был обновлён с версии до 1.01.
- ExtUtils::ParseXS был обновлён с версии 3.18 до 3.24.
- ExtUtils::Typemaps был обновлён с версии 3.19 до 3.24.
- ExtUtils::XSSymSet был обновлён с версии 1.2 до 1.3.
- feature был обновлён с версии 1.32 до 1.36.
- fields был обновлён с версии 2.16 до 2.17.
- File::Basename был обновлён с версии 2.84 до 2.85.
- File::Copy был обновлён с версии 2.26 до 2.29.
- File::DosGlob был обновлён с версии 1.10 до 1.12.
- File::Fetch был обновлён с версии 0.38 до 0.48.
- File::Find был обновлён с версии 1.23 до 1.27.
- File::Glob был обновлён с версии 1.20 до 1.23.
- File::Spec был обновлён с версии 3.40 до 3.47.
- File::Temp был обновлён с версии 0.23 до 0.2304.
- FileCache был обновлён с версии 1.08 до 1.09.
- Filter::Simple был обновлён с версии 0.89 до 0.91.

- Filter::Util::Call был обновлён с версии 1.45 до 1.49.
- Getopt::Long был обновлён с версии 2.39 до 2.42.
- Getopt::Std был обновлён с версии 1.07 до 1.10.
- Hash::Util::FieldHash был обновлён с версии 1.10 до 1.15.
- HTTP::Tiny был обновлён с версии 0.025 до 0.043.
- I18N::Langinfo был обновлён с версии 0.10 до 0.11.
- I18N::LangTags был обновлён с версии 0.39 до 0.40.
- if был обновлён с версии 0.0602 до 0.0603.
- inc::latest был обновлён с версии 0.4003 до 0.4205. Внимание: inc::latest устарел и может быть удалён из будущей версии Perl.
- integer был обновлён с версии 1.00 до 1.01.
- IO был обновлён с версии 1.28 до 1.31.
- IO::Compress::Gzip и друзья были обновлены с версии 2.060 до 2.064.
- IPC::Cmd был обновлён с версии 0.80 до 0.92.
- IPC::Open3 был обновлён с версии 1.13 до 1.16.
- IPC::SysV был обновлён с версии 2.03 до 2.04.
- JSON::PP был обновлён с версии 2.27202 до 2.27203.
- List::Util был обновлён с версии 1.27 до 1.38.
- locale был обновлён с версии 1.02 до 1.03.
- Locale::Codes был обновлён с версии 3.25 до 3.30.
- Locale::Maketext был обновлён с версии 1.23 до 1.25.
- Math::BigInt был обновлён с версии 1.9991 до 1.9993.
- Math::BigInt::FastCalc был обновлён с версии 0.30 до 0.31.
- Math::BigRat был обновлён с версии 0.2604 до 0.2606.
- MIME::Base64 был обновлён с версии 3.13 до 3.14.
- Module::Build был обновлён с версии 0.4003 до 0.4205. Внимание: Module::Build устарел и может быть удалён из будущей версии Perl.
- Module::CoreList был обновлён с версии 2.89 до 3.10.
- Module::Load был обновлён с версии 0.24 до 0.32.
- Module::Load::Conditional был обновлён с версии 0.54 до 0.62.
- Module::Metadata был обновлён с версии 1.000011 до 1.000019.
- mro был обновлён с версии 1.11 до 1.16.
- Net::Ping был обновлён с версии 2.41 до 2.43.
- Opcode был обновлён с версии 1.25 до 1.27.
- Package::Constants был обновлён с версии 0.02 до 0.04. Внимание: Package::Constants устарел и может быть удалён из буду-

щей версии Perl.

- Params::Check был обновлён с версии 0.36 до 0.38.
- parent был обновлён с версии 0.225 до 0.228.
- Parse::CPAN::Meta был обновлён с версии 1.4404 до 1.4414.
- Perl::OSType был обновлён с версии 1.003 до 1.007.
- perlfaq был обновлён с версии 5.0150042 до 5.0150044.
- PerlIO был обновлён с версии 1.07 до 1.09.
- PerlIO::encoding был обновлён с версии 0.16 до 0.18.
- PerlIO::scalar был обновлён с версии 0.16 до 0.18.
- PerlIO::via был обновлён с версии 0.12 до 0.14.
- Pod::Escapes был обновлён с версии 1.04 до 1.06.
- Pod::Functions был обновлён с версии 1.06 до 1.08.
- Pod::Html был обновлён с версии 1.18 до 1.21.
- Pod::Parser был обновлён с версии 1.60 до 1.62.
- Pod::Perldoc был обновлён с версии 3.19 до 3.23.
- Pod::Usage был обновлён с версии 1.61 до 1.63.
- POSIX был обновлён с версии 1.32 до 1.38_03.
- re был обновлён с версии 0.23 до 0.26.
- Safe был обновлён с версии 2.35 до 2.37.
- Scalar::Util был обновлён с версии 1.27 до 1.38.
- SDBM_File был обновлён с версии 1.09 до 1.11.
- Socket был обновлён с версии 2.009 до 2.013.
- Storable был обновлён с версии 2.41 до 2.49.
- strict был обновлён с версии 1.07 до 1.08.
- subs был обновлён с версии 1.01 до 1.02.
- Sys::Hostname был обновлён с версии 1.17 до 1.18.
- Sys::Syslog был обновлён с версии 0.32 до 0.33.
- Term::Cap был обновлён с версии 1.13 до 1.15.
- Term::ReadLine был обновлён с версии 1.12 до 1.14.
- Test::Harness был обновлён с версии 3.26 до 3.30.
- Test::Simple был обновлён с версии 0.98 до 1.001002.
- Text::ParseWords был обновлён с версии 3.28 до 3.29.
- Text::Tabs был обновлён с версии 2012.0818 до 2013.0523.
- Text::Wrap был обновлён с версии 2012.0818 до 2013.0523.
- Thread был обновлён с версии 3.02 до 3.04.
- Thread::Queue был обновлён с версии 3.02 до 3.05.
- threads был обновлён с версии 1.86 до 1.93.
- threads::shared был обновлён с версии 1.43 до 1.46.

- Tie::Array был обновлён с версии 1.05 до 1.06.
- Tie::File был обновлён с версии 0.99 до 1.00.
- Tie::Hash был обновлён с версии 1.04 до 1.05.
- Tie::Scalar был обновлён с версии 1.02 до 1.03.
- Tie::StdHandle был обновлён с версии 4.3 до 4.4.
- Time::HiRes был обновлён с версии 1.9725 до 1.9726.
- Time::Piece был обновлён с версии 1.20_01 до 1.27.
- Unicode::Collate был обновлён с версии 0.97 до 1.04.
- Unicode::Normalize был обновлён с версии 1.16 до 1.17.
- Unicode::UCD был обновлён с версии 0.51 до 0.57.
- utf8 был обновлён с версии 1.10 до 1.13.
- version был обновлён с версии 0.9902 до 0.9908.
- vmsish был обновлён с версии 1.03 до 1.04.
- warnings был обновлён с версии 1.18 до 1.23.
- Win32 был обновлён с версии 0.47 до 0.49.
- XS::Typemap был обновлён с версии 0.10 до 0.13.
- XSLoader был обновлён с версии 0.16 до 0.17.

Документация

Новая документация

perlrepository Этот документ был удалён (на самом деле переименован в perlgit, получив тщательный пересмотр) в Perl v5.14, приведя к тому, что веб-сайты с документацией Perl показывали устаревшую версию из Perl v5.12 как последнюю версию. Теперь он был восстановлен в форме заглушки, направляя читателя к текущей информации.

Изменения в существующей документации

perldata

- В документ была добавлена новая секция о новом синтаксисе

идекса/значения среза массива и ключе/значении среза хэша.

perldebguts

- Подпрограммы отладчика `DB::goto` и `DB::lsub` теперь задокументированы. [perl #77680]

perlexperiment

- `\s`, совпадающее с `\cK`, отмечено как экспериментальное.
- ithreads были приняты в v5.8.0 (но не рекомендуются с v5.20.0).
- Числа с плавающей запятой выше двойной точности больше не рассматриваются экспериментальными.
- Код в регулярных выражениях, перебор с возвратом в регулярных выражениях и lvalue-подпрограммы больше не указываются как экспериментальные. (Это также затрагивает perlre и perlsub.)

perlfunc

- В `chop` и `chomp` теперь есть замечание, что они могут сбросить итератор хеша.
- Обработка аргументов в `exec` теперь задокументирована более чётко.
- `eval EXPR` теперь содержит предупреждение о расширении чисел с плавающей запятой в некоторых локалях.
- В `goto EXPR` теперь задокументировано, что поддерживаются выражения, которые вычисляются в ссылку на код, как если было записано `goto &$coderef`. Такому поведению уже как минимум десять лет.
- Начиная с Perl v5.10, присутствовала возможность для подпрограмм в `@INC` возвращать ссылку на скаляр, содержащий начальный фрагмент исходного кода для вставки в начало файла. Теперь это задокументировано.

- Была обновлена документация `ref` с рекомендацией использования `blessed`, `isa` и `reftype` при работе с ссылками на благословлённые объекты.

perlguts

- Было сделано несколько небольших изменений, отражающих внутренние изменения сделанные в этом релизе perl.
- Были добавлены новые секции: Значения только для чтения и Копирование при записи

perlhack

- Была обновлена секция Супер быстрое руководство по патчам.

perlhacktips

- Обновлена документация, включившая ещё несколько примеров использования `gdb`.

perllexwarn

- Документация perllexwarn раньше описывала иерархию категорий предупреждений, которые понимает прагма warnings. Это описание теперь перенесено в саму документацию warnings, оставляя perllexwarn как заглушку к ней. Это изменение объединяет всю документацию для лексчиеских предупреждений в одном месте.

perllocale

- Документация теперь упоминает `fc()` и `\F`, и включает множество общих пояснений и исправлений.

perlop

- Дизайн языка Perl всегда призывал к мономорфным операторам. Теперь это упоминается явно.

perlopentut

- Руководство по `open` было полностью переписано Томом Кристиансеном и теперь фокусируется на раскрытии только основ, вместо предоставления всеобъемлющей документации по всем вещам, которые можно открыть. Эта работа появилась вследствие активного обсуждения в perl5-porters начатое с набора улучшений, написанных Александром Хартмайером в существующий perlopentut. Документ “больше, чем вы когда-либо захотите узнать об `open`” может последовать в следующих версиях perl.

perlre

- Тот факт, что движок регулярных выражений не делает попыток вызова конструкций (?{}) и (??{}) любое указанное число раз (хотя он в целом DWIM в случае успешного совпадения) было задокументировано.
- `/r` модификатор (для недеструктивных замен) теперь задокументирован.[perl #119151]
- Документация для `/x` и `(?# comment)` была расширена и прояснена.

perlreguts

- Документация была обновлена в свете последних изменений в `regcomp.c`.

perlsub

- Теперь задокументирована необходимость предекларации рекурсивных функций прототипами для того, чтобы эти прототипы учитывались в рекурсивных вызовах. [perl #2726]
- Теперь включён список имён подпрограмм, используемые в реализации perl. [perl #77680]

perltrap

- Появилась секция JavaScript.

perlunicode

- Обновлена документация для отражения изменений в `Bidi_Class` в Юникоде 6.3.

perlvar

- Была добавлена новая секция, объясняющая проблемы с производительностью в , & и $', включая способы обхода и изменения в различных версиях Perl.
- Три English имени переменных, которые долго были задокументированы, но на самом деле не существовали, были удалены из документации. Это были `$OLD_PERL_VERSION`, `$OFMT` и `$ARRAY_BASE`.

perlxs

- Было исправлено несколько проблем в примере `MY_CXT`.

Диагностика

Следующие дополнения и изменения были сделаны в диагностическом выводе, включая предупреждения и сообщения о фатальных ошибках. Для полного списка диагностических сообщений смотрите perldiag.

Новая диагностика

Новые ошибки

- delete argument is index/value array slice, use array slice

  Вы использовали синтаксис индекс/значение среза массива (`%array[...]`) как аргумент для `delete`. Вероятно вы имели ввиду `@array[...]` с символом @.

- delete argument is key/value hash slice, use hash slice

  Вы использовали синтаксис среза ключ/значение (`%hash{...}`) как аргумент для `delete`. Вероятно вы имели ввиду `@hash{...}` с символом @.

- Magical list constants are not supported

  Вы присвоили магический массив элементу стэша и затем попытались использовать подпрограмму из этого же слота. Вы попросили сделать Perl то, что он не в состоянии, детали зависят от версии Perl.

- Добавлена Setting $/ to a %s reference is forbidden

Новые предупреждения

- %s on reference is experimental:

Возможность “авто-разыменования” является экспериментальной.

Начиная с v5.14.0 стало возможно использовать push, pop, keys и другие встроенные функции не только на соответствующих типах данных, но и ссылках на них. Эта возможность не была распространена согласно своей спецификации и теперь стала излишней в свете постфиксного разыменования. Она всегда относилась к экспериментальным возможностям и в v5.20.0 имеет соответствующее предупреждение.

Теперь выводится предупреждение во время компиляции, если обнаруживаются подобные операции.

```
no if $] >= 5.01908, warnings => "experimental::
    autoderef";
```

Тем не менее, рассмотрите возможность замены подобной возможности, так как её поведение может измениться после того, как она станет стабильной.

- A sequence of multiple spaces in a charnames alias definition is deprecated

  Trailing white-space in a charnames alias definition is deprecated

  Эти два предупреждения об устаревших конструкциях, касающиеся `\N{...}`, были неправильно реализованы. Они не выводили предупреждения (теперь выводят) и не могли быть сделаны фатальными через `use warnings FATAL => 'deprecated'` (теперь могут).

- Attribute prototype(%s) discards earlier prototype attribute in same sub

  (W misc) Подпрограмма была объявлена, например, как `sub foo : prototype(A): prototype(B){}`. Так как каждая подпрограмма может иметь только один прототип, ранние объявления отбрасываются и используется последнее.

- Invalid \0 character in %s for %s: %s\0%s

  (W syscalls) Встроенный символ \0 в файловых путях или в аргументах других системных вызовов, начиная с 5.20, приводит к предупреждениям. Часть после \0 раньше игнорировалась в системных вызовах.

- Matched non-Unicode code point 0x%X against Unicode property; may not be portable.

  Это заменяет сообщение “Code point 0x%X is not Unicode, all \p{} matches fail; all \P{} matches succeed”. (Код символа 0x%X не является Юникод-символом, поиск совпадения \p{} всегда неудачен, а \P{} всегда успешен.)

- [Missing ’\]‘in prototype for %s : %s](https://metacpan.org/pod/perldiag#Missing’\]in prototype for %s : %s)

  (W illegalproto) Группировка была начата с `[` но не была закрыта завершающим `]`.

- Possible precedence issue with control flow operator

  (W syntax) Возможна проблема при смешивании операторов управляющих исполнением (например, `return`) и низкоприоритетных операторов, таких как `or`. Рассмотрим:

  ```
  sub { return $a or $b; }
  ```

  Это разбирается как:

  ```
  sub { (return $a) or $b; }
  ```

  Что эквивалентно:

  ```
  sub { return $a; }
  ```

  Необходимо использовать либо скобки, либо более высокоприоритетный оператор.

  Предупреждение также может выводиться и для подобных конструкций:

  ```
  sub { 1 if die; }
  ```

- Postfix dereference is experimental

  (S experimental::postderef) Это предупреждение выводится, если вы используете синтаксис экспериментального постфиксного разыменования. Просто подавляйте это предупреждение, если вы хотите использовать эту возможность, но знайте, что делая так вы берёте на себя риск использования экспериментальной возможности, которая может измениться или может быть удалена в будущих версиях Perl:

```
no warnings "experimental::postderef"; use feature "
    postderef",
"postderef_qq"; $ref->$*; $aref->@*; $aref->@[
    @indices]; ... etc ...
```

- [Prototype ‘%s’ overridden by attribute ‘prototype(%s)’ in %s](https://metacpan.org
‘%s’ overridden by attribute ‘prototype(%s)’ in %s)

  (W prototype) Прототип был объявлен и в скобках после имени подпрограммы, и с использованием атрибута prototype. Прототип в скобках бесполезен, так как он был заменён прототипом из атрибута раньше чем будет использован.

- Scalar value @%s[%s] better written as $%s[%s]

  (W syntax) В скалярном контексте вы использовали срез индекс/значение массива (на что указывает %), чтобы получить один элемент массива. Обычно лучше запрашивать скалярное значение (указывая)., ‘foo[&bar]всегда ёведт себя как скаляр, как при возвращение значения, так и вычислении своего аргумента, в том время как%foo[&bar]ёсоздат списочный контекст для получателя, который может привести странным результатам, если вы ожидаете только одним элемент. При вызове в списочном контексте он также возвращает индекс (который возвращает&bar‘) в дополнении к значению.

- Scalar value @%s{%s} better written as $%s{%s}

  (W syntax) В скалярном контексте вы использовали срез ключ/значение хеша (на что указывает %), чтобы получить один элемент хеша. Обычно лучше запрашивать скалярное значение (указывая)., ‘foo{&bar}всегда ёведт себя как скаляр, как при возвращение значения, так и вычислении своего аргумента, в том время как@foo{&bar}‘ создаёт списочный контекст для получателя, который может привести странным результатам, если вы ожидаете только одним элемент. При вызове в списочном контексте он также возвращает ключ в дополнении к значению.

- Setting $/ to a reference to %s as a form of slurp is deprecated, treating as undef

- Unexpected exit %u

(S) Был вызван exit() или скрипт успешно завершился, если `PERL_EXIT_WARN` был установлен в `PL_exit_flags`.

- Unexpected exit failure %d

  (S) Неперехваченный die() был вызван при `PERL_EXIT_WARN` установленном в `PL_exit_flags`.

- Use of literal control characters in variable names is deprecated

  (D deprecated) Использование управляющих символов в исходном коде для ссылки на ^FOO переменные, такие как $^{\hat{}}$GLOBAL_PHASE}, теперь является устаревшим. Это затрагивает только код, подобный
  cT,
  $cT--(`SOH`)$:{"\cT"} и $^T продолжают работать.

- [Useless use of greediness modifier](https://metacpan.org/pod/perldiag#Useless use of greediness modifier '%c' in regex; marked by <-- HERE in m/%s/)

  Это исправляет ошибку [Perl #42957].

Изменения в существующей диагностике

- Предупреждения и ошибки движка регулярных выражений теперь содержат корректный UTF-8.

- Сообщение об ошибке "Unknown switch condition" (Неизвестное условие в переключателе) получило незначительное изменение. Это ошибка происходит, когда неизвестное условие в условном выражении `(?(foo))`. Раньше сообщение выглядело так:

  ```
  Unknown switch condition (?(%s in regex;
  ```

  Но значение %s зависело от удачи. Для `(?(foobar))`, вы могли видеть "fo" или "f". Для символов Юникод вы могли получить искажённую строку. Сообщение было изменено к следующему виду:

  ```
  Unknown switch condition (?(...)) in regex;
  ```

Дополнительно, маркер `'<-- HERE'` в сообщении об ошибке теперь указывает на правильную позицию в регулярном выражении.

- Предупреждение “%s”\x%X” does not map to Unicode” теперь корректно указано как строгое предупреждение, а не фатальная ошибка.

- В некоторых редких случаях можно было получить сообщение “Can’t coerce readonly REF to string” (не могу изменить ссылку, доступную только на чтение) вместо привычного “Modification of a read-only value” (модификация значения, доступного только на чтение). Это альтернативное сообщение было удалено.

- Сообщение “Ambiguous use of * resolved as operator *” (неоднозначное использование *, используется как оператор *) о “%” и “&” раньше появлялось при некоторых условиях, там где отсутствовал процитированный оператор, поэтому предупреждение было полностью некорректным. Это было исправлено [perl #117535, #76910].

- Предупреждения о некорректных прототипах функций теперь больше соответствуют тому, как они отображаются. Некоторые из этих предупреждений будут урезать прототипы, которые содержат нулевой символ. В других случаях одно предупреждение будет подавлять другое. Предупреждение о некорректных символах в прототипах больше не выводят “after ‘_’ ” (после), если плохой символ появился до подчёркивания.

- Perl folding rules are not up-to-date for 0x%X; please use the perlbug utility to report; in regex; marked by <– HERE in m/%s/

  Это сообщение теперь находится в категории регулярных выражений, а не в категории устаревших. Это по-прежнему предупреждение по умолчанию (т.е. строгое) [perl #89648].

- %%s[%s] in scalar context better written as $%s[%s]

  Это предупреждение теперь выводится для любого из `%array[$index]` или `%hash{key}`, если известно на этапе компиляции, что они в скалярном контексте. Ранее оно звучало как “Scalar value %%s[%s] better written as $%s[%s]”.

- Switch condition not recognized in regex; marked by <– HERE in m/%s/:

  Описание этого диагностического сообщения было расширено для отражения всех случаев, когда данное предупреждение возникает. Проблемы с положением индикатора стрелки также были исправлены.

Изменения в утилитах

a2p

- Возможный крах в результате ошибки на один байт, при попытке доступа перед началом буфера было исправлено. [perl #120244]

`bisect.pl` В утилите для git bisect `Porting/bisect.pl` было сделано множество исправлений.

Она распространяется как часть дистрибутива исходных кодов, но не устанавливается, так как она не является самодостаточной, так как она полагается на то, что запускается из директории с git checkout. Кроме того, она не пытается исправить тесты, исправить ошибки в работе или сделать что-либо полезное для инсталляции – её назначение сделать минимальное изменение, чтобы получить нужную версию из истории, чтобы собрать и запустить наиболее близко к состоянию “как было”, и соответственно упростить использование `git bisect`.

- Может опционально запустить тест с таймаутом.
- Может быть непосредственно запущенно в чистом git checkout.
- Может запустить тест под `valgrind`.
- Может применить пользовательские патчи и фиксы к исходному коду текущей ревизии перед сборкой.

- Имеет исправления, для возможности сборки немного большего числа исторических диапазонов bleadperl, что может быть полезно для поиска источника ошибки или изменения поведения.

find2perl

- find2perl теперь поддерживает символ `?` корректно. [perl #113054]

perlbug

- `perlbug` теперь имеет опцию `-p` для вложения патчей к отчётам об ошибках.
- Шаблон отчёта perlbug был исправлен для поддержки CRLF концов строк на Windows. [perl #121277]
- perlbug, насколько это возможно, делает меньше допущений о кодировке отчёта. Предположение об UTF-8 по умолчанию вероятно будет изменено в будущем, чтобы позволить пользователю переопределить кодировку.

Конфигурация и компиляция

- `Makefile.PL` для SDBM_File теперь производит гораздо лучший `Makefile`, который избегает условий гонки, при параллельной сборке, которые могли приводить к ошибкам сборки. Это последняя из известных проблем при параллельной сборке make (на *nix платформах) и, таким образом, мы верим, что параллельный make теперь всегда будет свободен от ошибок.
- Поддержка опций в `installperl` и `installman` была переработана на использование Getopt::Long. Обе используются целью `Makefile` `install` и не устанавливаются, поэтому эти изменения затронут вероятно только самодельные инсталляционные скрипты.

  - Однобуквенные опции теперь имеют и длинные имена.
  - Некорректные опции теперь отвергаются.
  - Аргументы командной строки, которые не являются опциями теперь отвергаются.
  - Каждые теперь имеют опцию `--help` для отображения сообщения об использовании.

  Поведение при всех корректных задокументированных вызовах не изменилось.

- Где это возможно, сборка избегает рекурсивных вызовов `make`, когда собирает чистые Perl расширения, не убирая параллельности сборки. На данный момент порядка 80 расширений могут быть обработаны непосредственно утилитой `make_ext.pl`, это означает, что больше не делается 80 вызовов `make` и 160 вызовов `miniperl`.

- Сборочная система работает корректно при сборке с включённым использованием оптимизации линковки (опция `-flto`) GCC или Clang. [perl #113022]

- Уникальные базовые имена библиотек с `d_libname_unique`.

  При компиляции perl с этой опцией библиотечные файлы XS модулей именуются уникально. Например, Hash/Util/Util.so становится Hash/Util/PL_Hash__Util.so. Это поведение соответствует тому, что на данный момент происходит в VMS и служит основой для работы над портом на Android.

- Опция `sysroot` для указания логического корневого каталога для gcc и clang.

  При сборке с этой установленной опцией, как Configure, так и компиляторы ищут заголовки и библиотеки в этом новом sysroot, вместо /.

  Это экономит значительное время при кросс-компиляции, а также может помочь при нативной сборке, если файлы вашего набор утилит сборки имеют нестандартное положение.

- Модель кросс-компиляции была обновлена. Появилось несколько новых опций и некоторые несовместимые изменения:

Сейчас мы собираем бинарные файлы для miniperl и generate_uudmap для использования на хосте, вместо запуска каждого вызова miniperl на целевой платформе; это означает более короткий ‘make test’, нам не требуется доступ к целевой системой после того как отработает Configure. Вы также может предоставить уже собранный бинарный файл через опции Configure `hostperl` и `hostgenerate`.

Дополнительно, если целевая платформа EBCDIC, а на хосте ASCII, или наоборот, вам потребуется запускать Configure с опцией –`Uhostgenerate`, указывающей на то, что generate_uudmap должен быть запущен на целевой платформе.

И, наконец, также есть возможность завершить Configure раньше, сразу после сборки бинарных файлов на хосте, запустив кросс-компиляцию без указания `targethost`.

Также к несовместимым изменениям относится отсутствие использования xconfig.h, xlib или Cross.pm, поэтому сохранённые файлы конфигурации и Makefile должны быть обновлены.

- Связанная с вышеописанным, появилась возможность указания расположения командной оболочки sh (или эквивалента) на целевой системе: `targetsh`.

  Например, Android имеет свой sh в /system/bin/sh, поэтому при кросс-компиляции из более привычного в Unix системах расположения sh в /bin/sh, “targetsh” получит значение /system/bin/sh, а “sh” - /bin/sh.

- По умолчанию, **gcc** 4.9 делает некоторые оптимизации, которые ломают perl. Опция **-fwrapv** отключает эти оптимизации (и возможно другие), поэтому для **gcc** 4.3 и более поздних (так как могут быть подобные проблемы в старых релизах, но **-fwrapv** был сломан до 4.3, и оптимизация вероятно никуда не денется), `Configure` добавляет **-fwrapv**, если только пользователь не запросил **-fno-wrapv**, который отключает **-fwrapv**, или **-fsanitize=undefined**, который игнорирует переполнения **-fwrapv** в ошибках исполнения. [perl #121505]

Тестирование

- Цель make `test.valgrind` теперь позволяет выполнение параллельных тестов. Эта цель разрешает запуск тестового набора Perl под Valgrind, который определяет некоторые виды ошибок в C-программировании ценой более длительного выполнения. На подходящем "железе" возможность параллельного запуска отыгрывает эти дополнительные затраты. [perl #121431]
- Различные тесты в `t/porting/` больше не пропускаются, если каталог `.git` находится вне дерева perl и задаётся переменной `$GIT_DIR` [perl #120505].
- Тестовый набор больше не завершается ошибкой, если пользовательский интерактивный командный интерпретатор поддерживает переменную окружения `$PWD`, а `/bin/sh` используемый для запусков тестов - нет.

Поддержка платформ

Новые платформы

- Android

  Perl теперь может быть собран для Android как нативно, так и через кросс-компиляцию, для всех трёх доступных архитектур (ARM, MIPS и x86) для широкого диапазона версий.

- Bitrig

  Поддержка компиляции была добавлена для Bitrig - форка OpenBSD.

- FreeMiNT

  Была добавлена поддержка для FreeMINT, свободной ОС с открытыми исходниками для системы Atari ST и её наследников, основанных на оригинальной MINT, которая ранее перешла к Atari.

- Synology

Synology поставляет свои NAS-устройства с урезанным Linux дистрибутивом (DSM) на относительно дешёвом CPU (как, например, Marvell Kirkwood mv6282 - ARMv5tel или Freescale QorIQ P1022 ppc - e500v2) не предназначенный для рабочих станций или разработки. Эти устройства теперь должны собираться. Основная проблема была в нестандартном расположении утилит.

Более неподдерживаемые платформы

- `sfio`

  Код, относящийся к поддержке системы ввода/вывода `sfio`, был удалён.

  В Perl 5.004 была добавлена поддержка родного API `sfio` - безопасной и быстрой библиотеки ввода/вывода AT&T. Этот код по-прежнему собирается с v5.8.0, хоть и с множеством падающих регрессивных тестов, но оказался нечаянно сломан перед релизом v5.8.1, что означает, что он больше не работал на всех последующих версиях Perl после него. На протяжении десятилетия мы не получили ни одного отчёта об ошибке об этом, отсюда становится понятно, что никто не использует эту функциональность на всех версиях Perl, которые в той или иной степени поддерживаются.

- AT&T 3b1

  Была удалена поддержка для 3b1 в Configure, также известной как AT&T Unix PC (и похожий AT&T 7300).

- DG/UX

  DG/UX был Unix'ом проданным Data General. Последний релиз был в апреле 2001. Он работает только на собственном оборудовании Data General.

- EBCDIC

  В отсутствии источника регулярных отчётов о сборке, код, отвечающий за поддержку платформ EBCDIC будет удалён в perl перед выпуском 5.22.0.

Платформо-специфичные замечания

- Cygwin
  - recv() на подключённом дескрипторе будет передавать возвращённый адрес отправителя, что бы не случилось с рабочим буфером. recv() теперь использует обходной манёвр, аналогичный обёртке Win32 recv() и возвращает пустую строку, если recvfrom(2) не изменяет переданную длину адреса. [perl #118843]
  - Исправлена ошибка сборки в cygwin.c на Cygwin 1.7.28.

    Тесты теперь обрабатывают ошибки, которые происходят, когда `cygserver` не запущен.
- GNU/Hurd

  Библиотека для совместимости с BSD `libbsd` больше не требуется для сборки.
- Linux

  Файл хинтов теперь ищет `libgdbm_compat` только если сам `libgdbm` также требуется. Первый бесполезен без последнего и, в некоторых случаях, включение его может привести к отказу сборки.
- Mac OS

  Сборочная система теперь учитывает настройку `ld`, переданную пользователем при запуске `Configure`.
- MidnightBSD

  `objformat` был удалён, начиная с версии 0.4-RELEASE MidnightBSD и являлся устаревшим в ранних версиях. Это приводило к тому, что сборочное окружение ошибочно конфигурировалось для `a.out`, вместо `elf`. Теперь это исправлено.
- Платформы со смешанным порядком байтов

  Код, поддерживающий операции `pack` и `unpack` на платформах со смешанным порядком байтов был удалён. Мы уверены, что Perl больше не собирается на архитектурах со смешанным порядком байт (таким как PDP-11s), поэтому мы не думаем,

что это изменение затронет какие-либо платформы, на которых собирался v5.18.0.

- VMS

  - Возможность PERL_ENV_TABLES для контроля заполнения %ENV при старте perl была сломана в Perl 5.16.0, но теперь была исправлена.
  - Пропуск проверок доступа в opendir() для внешних файлов. [perl #121002]
  - Проверка для метасимволов глоба для путей, возвращаемых оператором `glob()` была заменена проверкой для VMS символов подстановки. Это позволяет избегать большое число не нужных вызовов `lstat()`, что делает простые глоб-операции быстрее на 60-80%.

- Win32

  - `rename` и `link` на Win32 теперь устанавливают $! в ENOSPC и EDQUOT в соответствующих случаях. [perl #119857]
  - Опции makefile BUILD_STATIC и ALL_STATIC для линковки некоторых или (почти) всех расширений статически (в perl520.dll, и также в отдельный perl-static.exe) были сломаны для сборок MinGW. Теперь это исправлено.

    Опция ALL_STATIC также была улучшена и включает расширения Encode и Win32 (для обоих сборок VC++ и MinGW).
  - Была добавлена поддержка сборки с Visual C++ 2013. На данный момент есть два возможных падающих теста (смотрите “Testing Perl on Windows” in perlwin32), которые в скором времени будут благополучно исправлены.
  - Была добавлена экспериментальная поддержка сборки компилятором Intel C++ Compiler. Могут быть использованы nmake makefile (win32/Makefile) и dmake makefile (win32/makefile.mk). “nmake test” на данный момент не проходит из-за `cpan/CGI/t/url.t`.
  - Уничтожение дерева процессов с помощью “kill” in perlfunc и отрицательного сигнала было сломана, начиная с 5.18.0. При этой ошибке `kill` всегда возвращал 0

для негативного сигнала даже для корректных PID и процессы не уничтожались. Это было исправлено [perl #121230].

– Время, затрачиваемое на сборку perl на WIndows, было существенно сокращено (обычно отмечается экономия времени в диапазоне 30-40%) за счёт сокращения (обычно неудачных) вызовов ввода/вывода для каждого `require ()` (только для **miniperl.exe**). [perl #121119]

– Примерно 15 минут простоя было удалено при запуске `make test` из-за ошибки, в которой таймаут, используемый для тестов, не мог быть отменён после завершения теста, поэтому до запуска следующего теста проходил полный период таймаута. [perl #121395]

– На perl, собранного без псевдо-форка (сборка с псевдо-форком не затронута этой ошибкой), уничтожение дерева процессов с помощью `kill()` и негативного сигнала приводила к тому, что `kill()` инвертировал возвращаемое значение. Например, если `kill()` убивал один PID дерева процессов, тогда он возвращал 0 вместо 1, а если `kill()` получал два некорректных PID, тогда он возвращал 2, вместо 0. Это приводило проблемам, так как возможность уничтожения дерева процессов была реализована на Win32. Теперь это исправлено в соответствии с документацией. [perl #121230]

– При сборке 64-битного perl, не инициализированное чтение памяти в **miniperl.exe**, используемого в сборочном процессе, могло приводить к созданию 4GB **wperl.exe**. Теперь это было исправлено. (Обратите внимание, что **perl.exe** сам по себе не затронут, но **wperl.exe** был полностью сломан). [perl #121471]

– Perl теперь может быть собран с **gcc** версии 4.8.1 от http://www.mingw.org. Ранее он был сломан из-за некорректного определения DllMain() в одном из исходных файлов perl. Ранние версии **gcc** также были затронуты, если использовали версию 4 пакета w32api. Версии **gcc** доступные с http://mingw-w64.sourceforge.net/ не были затронуты. [perl #121643]

– Тесты теперь проходят без ошибок, если perl собран на разделе с FAT, а Windows OS находится на NTFS разделе.

[perl #21442]

- При клонировании стека контекста при эмуляции fork(), Perl_cx_dup() падал при попытке доступа к информации о параметрах для данных стека контекста, который не включал параметров, как, например, в `&foo;`. [perl #121721]
- Была исправлена, появившаяся в perl #113536, утечка памяти на каждый вызов `system` и апострофов (\` `) для большинства вариантов Perl на Win32, начиная с 5.18.0. Утечка происходила только если вы включали псевдофорк в вашей сборке Win32 Perl и запускали эту сборку на Server 2003 R2 или более новой ОС. Эта утечка не возникала на WinXP SP3. [perl #121676]

- WinCE
  - Сборка XS модулей большей частью восстановлена. Несколько по-прежнему пока не собираются, но на данный момент возможно собрать Perl на WinCE с небольшим числом патчей (для Socket и ExtUtils::MakeMaker), которые, мы надеемся, скоро будут приняты.
  - Perl может теперь быть собран в один заход, без прерывания пользователем на WinCE для запуска `nmake -f Makefile.ce all`.

    Была восстановлена поддержка сборки с помощью EVC (Embedded Visual C++) 4. Perl также может быть собран с использованием Smart Devices for Visual C++ 2005 or 2008.

Внутренние изменения

- Внутреннее представление было изменено для переменных поиска совпадений `$1`, `$2` и т.д., `$\``, `$&`, `$'`, `${^PREMATCH}`, `${^MATCH}` и `${^POSTMATCH}`. Они использует немного меньше памяти, избегают строкового сравнения и конвертации в численное при обращении и используют на 23 строчки C-кода меньше. Это изменение никак не затрагивает любой внешний код.

- Массивы теперь используют NULL внутренне для представления незанятых слотов, вместо &PL_sv_undef. &PL_sv_undef больше не рассматривается как специальное значение, поэтому av_store(av, 0, &PL_sv_undef) приведёт к тому, что элемент 0 в массиве будет содержать доступный только на чтение неопределённый скаляр. `$array[0] = anything` приведёт к фатальной ошибке, а `\$array[0]` при сравнении будет равно `\undef`.

- SV, возвращаемый HeSVKEY_force(), теперь корректно отображает наличие UTF8 у соответствующего ключа хеша, если этот ключ не сохранён в SV. [perl #79074]

- Некоторые редко используемые функции и макросы доступные для XS кода теперь считаются устаревшими. Это `utf8_to_uvuni_buf` (используйте `utf8_to_uvchr_buf` вместо неё), `valid_utf8_to_uvuni` (используйте `utf8_to_uvchr_buf` вместо неё), `NATIVE_TO_NEED` (это никогда не работало как следует) и `ASCII_TO_NEED` (это никогда не работало как следует)

  Начиная с этого релиза, приложению практически никогда не требуется делать различие между набором символов платформы и Latin1, на которых основаны нижние 256 символов Юникода. Новый код не должен использовать ни `utf8n_to_uvuni` (используйте `utf8_to_uvchr_buf` вместо неё), ни `uvuni_to_utf8` (используйте `uvchr_to_utf8` вместо неё).

- Сокращённые записи для цели в Makefile для большинства редко (или никогда не) используемых тестовых и профилирующих целей были удалены или объединены только те цели Makefile, которые используют их. В частности, данные цели были удалены вместе с документацией, которая ссылалась на них или объясняла как использовать их:

```
check.third check.utf16 check.utf8 coretest minitest
    .prep
minitest.utf16 perl.config.dashg perl.config.dashpg
    perl.config.gcov
perl.gcov perl.gprof perl.gprof.config perl.pixie
    perl.pixie.atom
perl.pixie.config perl.pixie.irix perl.third perl.
    third.config
perl.valgrind.config purecovperl pureperl quantperl
    test.deparse
```

```
6 test.taintwarn test.third test.torture test.utf16
    test.utf8
7 test_notty.deparse test_notty.third test_notty.
    valgrind test_prep.third
8 test_prep.valgrind torturetest ucheck ucheck.third
    ucheck.utf16
9 ucheck.valgrind utest utest.third utest.utf16 utest.
    valgrind
```

По-прежнему возможно запустить соответствующую команду “руками” - никакая связанная функциональность не удалена.

- Теперь стало возможно удерживать Perl от инициализации поддержки локали. По большей части Perl не обращает внимание на локаль. (Смотрите perllocale). Тем не менее, до текущего момента, при старте, он всегда пытался инициализировать поддержку системной локали просто на случай, если запускаемой программе потребуется использовать локаль. (Это первая вещь, о которой локале-зависимая программа должна позаботиться, задолго до того как Perl узнает нужна ему локаль или нет). Это работало хорошо, за исключением тех случаев, когда Perl встроен в другое приложение, которому нужна локаль, отличная от системной. Теперь устанавливается переменная окружения `PERL_SKIP_LOCALE_INIT` при старте Perl, то шаг инициализации пропускается. До этого, на Windows платформах, был только один способ обхода этого недостатка - использовать модифицированный Perl с переделанным внутренним кодом. Приложениям, которым требуется старый Perl, может обнаружить требуется ли ещё этот хак, по тестированию наличия символа C препроцессора `HAS_SKIP_LOCALE_INIT` [RT #38193].

- `BmRARE` и `BmPREVIOUS` были удалены. Они не использовались в API. Для XS-модулей они теперь определены как 0.

- `sv_force_normal`, которая обычно завершалась с фатальной ошибкой при получении доступных только на чтение значений, допускала модификацию таких значений на этапе компиляции. Теперь это было исправлено на генерацию ошибки при работе с доступными только на чтение значениями несмотря ни на что. Это изменение раскрыло несколько ошибок в ядре Perl.

- Новый механизм копирования-при-записи (который теперь включён по умолчанию), позволяет любому SvPOK скаляру автоматически обновиться до копируемого-при-записи при копировании. Число ссылок на строковой буфер сохраняется в самом строковом буфере.

  Например:

```
$ perl -MDevel::Peek -e'$a="abc"; $b = $a; Dump $a; Dump $b'
SV = PV(0x260cd80) at 0x2620ad8
  REFCNT = 1
  FLAGS = (POK,IsCOW,pPOK)
  PV = 0x2619bc0 "abc"\0
  CUR = 3
  LEN = 16
  COW_REFCNT = 1
SV = PV(0x260ce30) at 0x2620b20
  REFCNT = 1
  FLAGS = (POK,IsCOW,pPOK)
  PV = 0x2619bc0 "abc"\0
  CUR = 3
  LEN = 16
  COW_REFCNT = 1
```

  Обратите внимание, оба скаляра разделяют один и тот же PV буфер и имеют COW_REFCNT больше нуля.

  Это означает, что XS код, который желает модифицировать буфер SvPVX() SV, должен вызвать сначала SvPV_force() или подобное, чтобы убедиться, что буфер действующий (и не разделённый) и затем вызвать SvSETMAGIC(). Это, на самом деле, всегда было так (например, ключи хеша всегда были копируемыми-при-записи), это изменение просто распространило COW-поведение на широкий спектр SV.

  Одно важное отличие в том, что до 5.18.0 разделённые скаляры ключей хеша использовали установленный флаг SvREADONLY, теперь это больше не так.

  Это поведение может быть отключено при запуске Configure с параметром **-Accflags=-DPERL_NO_COW**. Эта опция возможно будет удалена в Perl 5.22.

- PL_sawampersand теперь является константой. Переключатель для этой переменной (для включения/отключения копирования в зависимости от того просмотрен ли был $&) был удалён

и заменён на копируемый-при-записи, исправляя несколько ошибок.

Старое поведение по-прежнему может быть включено при запуске `Configure` с **-Accflags=-DPERL_SAWAMPERSAND**.

- Функции `my_swap`, `my_htonl` и `my_ntohl` были удалены. Непонятно, почему эти функции были помечены как *A*, часть API. XS-код больше не может вызывать их напрямую, так как он не может рассчитывать на то, что они будут скомпилированы. Неудивительно, что на CPAN нет кода, который их использовал.

- Сигнатура regex-функции `Perl_re_intuit_start()` была изменена; указатель функции `intuit` в структуре плагинов движка регулярных выражений также соответственно изменился. Был добавлен новый параметр `strbeg`; он имеет такое же значение, как и также названный параметр в `Perl_regexec_flags`. Ранее intuit пытался угадать начало строки из переданного SV (при наличии) и иногда ошибался (например, перегруженный SV).

- Сигнатура regex-функции `Perl_regexec_flags()` была изменена; указатель функции `exec` в структуре плагинов движка регулярных выражений также соответственно изменился. Параметр `minend` теперь имеет тип `SSize_t` для лучшей поддержки 64-битных систем.

- XS-код может использовать различные макросы для изменения регистра символов или кодов символов (например, `toLOWER_utf8()`). Только часть из них была задокументирована до настоящего момента; и теперь им должно отдаваться предпочтение в использовании, вместо соответствующих функций. Смотрите “Character case changing” in perlapi.

- Код работал с uid и gid достаточно непоследовательно. В некоторых местах предполагалось, что они могут спокойно быть сохранены в UV, в других - в IV, в третьих - в int. Было создано четыре новых макроса: SvUID(), sv_setuid(), SvGID() и sv_setgid()

- `sv_pos_b2u_flags` был добавлен в API. Он схож с `sv_pos_b2u`, но поддерживает длинные строки на 64-битных платформах.

- `PL_exit_flags` теперь может быть использован для встраиваемого perl или другого XS-кода, чтобы иметь `warn` или `abort` из perl при попытке выхода [perl #52000].

- Компиляция с `-Accflags=-PERL_BOOL_AS_CHAR` теперь позволяет C99 и C++ компиляторам эмулировать альтернативное название `bool` в `char`, которое perl делает для C89 компиляторов. [perl #120314]

- `sv` аргумент в “sv_2pv_flags” in perlapi, “sv_2iv_flags” in perlapi, “sv_2uv_flags” in perlapi и “sv_2nv_flags” in perlapi и их более старые обёртки sv_2pv, sv_2iv, sv_2uv, sv_2nv, теперь не принимают NULL. Передача NULL приведёт к краху. Когда ненулевой маркер был в целом представлен в 5.9.3 функции были отмечены как ненулевые, но начиная с создания SV API в 5.0 alpha 2, если передавался NULL, функции возвращали 0 или ложное значение. Код, который поддерживал `sv` аргумент ненулевым, относился непосредственно к 5.0 alpha 2 и неявно к Perl 1.0 (pre 5.0 api). Отсутствие пометки в документации, что функции принимает NULL `sv`, было исправлено в 5.11.0, и между релизами 5.11.0 и 5.19.5 функции были помечены как NULLOK. Так как оптимизация NULLOK кода была удалена, функции стали помеченными как не-NULL снова, поскольку базовые макросы получающего типа никогда не передают NULL в эти функции и приводят к краху ещё до передачи NULL.

  Единственный способ передать NULL `sv` в функции sv_2*v* - это если XS-код напрямую вызовет sv_2*v*. Это маловероятно, так как XS-код использует макросы Sv*V* для получения соответствующего значения SV. Одна возможная ситуация, которая приведёт к передаче NULL `sv` в sv_2*v* функциях, это если XS-код определяет свой собственный макрос получения SV, который проверяет на NULL **до того**, как разыменует и проверит SV-флаги через макросы публичного API Sv*OK* или напрямую через приватное API `SvFLAGS`, и, если `sv` - это NULL, затем вызывать функции sv_2*v с NULL-литералом или передавать `sv`, содержащий значение NULL.

- newATTRSUB теперь является макросом

  newATTRSUB публичного API раньше был макросом к приватной функции Perl_newATTRSUB. Функция Perl_newATTRSUB

была удалена. newATTRSUB теперь макрос к другой внутренней функции.

- Изменения в предупреждениях, выводимых `utf8n_to_uvchr()`

  Эта низко-уровневая функция декодирует первый символ UTF-8 строки в код символа. Это доступно из кода XS-уровня, но не рекомендуется использовать её напрямую. Существуют более высоко-уровневые функции, которые могут быть вызваны вместо неё, например, “utf8_to_uvchr_buf” in perlapi. Для полноты, этот документ описывает изменения в ней. Теперь тесты на неправильность выполняются до всех других тестов поиска потенциальных проблем. Одна из подобных проблем, ключает коды символов настолько больших, что они никогда не появятся в официальном стандарте (текущий стандарт расширяется вплоть до наивысшего допустимого кода символа из ранних версий). Также возможно (но нет на CPAN), предупреждать или запрещать подобные коды, тем не менее принимая меньшие коды символов, которые по-прежнему выше легального максимального уровня Unicode. Предупреждение для этого случая включает код символа, если может быть представлено на данной машине. Ранее оно всегда показывало сырые байты, что по-прежнему делает для кодов символов без представления.

- Изменения в движке регулярных выражений, которые затрагивают интерфейс подключаемых движков регулярных выражений.

  Множество флагов, которые были видны через regexp.h и использовались для заполнения элемента extflags структуры регулярных выражений, были удалены. Эти поля были технически приватными для собственного движка регулярных выражений Perl и не должны быть видны здесь.

  Это затрагивает такие флаги:

```
RXf_NOSCAN
RXf_CANY_SEEN
RXf_GPOS_SEEN
RXf_GPOS_FLOAT
RXf_ANCH_BOL
RXf_ANCH_MBOL
RXf_ANCH_SBOL
```

```
RXf_ANCH_GPOS
```

Также как и маски для флагов:

```
RXf_ANCH_SINGLE
RXf_ANCH
```

Все они были переименованы в PREGf_ эквиваленты и перемещены в regcomp.h.

Поведение ранее достигаемое установкой одного или нескольких RXf_ANCH_ флагов (через маску RXf_ANCH) теперь было заменено на **единственный** бит флага в extflags:

```
RXf_IS_ANCHORED
```

подключаемые движки регулярных выражений, которые ранее устанавливали эти флаги должны теперь устанавливать только один этот флаг.

- Ядро Perl теперь правильно использует `av_tindex()` ("верхний индекс массива"), как наиболее удачно названный синоним для `av_len()`.

- Безвестная переменная интерпретатора `PL_timesbuf` ожидается к удалению в ранних версиях разрабатываемой ветки 5.21.x, поэтому Perl 5.22.0 не будет предоставлять её XS-авторам. Так как переменная по-прежнему существует в 5.20.0, мы надеемся, что предупреждающее сообщение об устаревании поможет всем ¸ кто использует эту переменную.

Избранные исправления ошибок

Регулярные выражения

- Исправлено небольшое количество regexp-конструкций, которые могли не дать совпадения или обрушить perl, если для строки, в которой искалось совпадение, было выделено более 2 Гб на 32-битных системах. [RT #118175]

- Были исправлены различные утечки памяти, затрагивающие разбор конструкции регулярного выражения `(?[...])`.

- `(?[...])` теперь допускает интерполяцию пред-компилируемого шаблона, состоящего из `(?[...])` с классами символов в квадратных скобках внутри (`$pat = qr/(?\[ \[a\] \])/; /(?\[ $pat \])/`). Ранее квадратные скобки смущали парсер регулярных выражений.

- Предупреждение “Quantifier unexpected on zero-length expression” (неожиданный квантор на выражении нулевой длины) могло появляться дважды, начиная с Perl v5.10 для регулярных выражений, содержащих альтернативы (например, “a|b”), вызванные оптимизацией префиксного дерева.

- В Perl v5.18 появилась ошибка, из-за которой интерполяция смешанных UTF-8 строк с разным внутренним представлением в регулярном выражении приводит к неправильно сформированному UTF-8 шаблону, особенно если строка с символом в байтовом представлении в диапазоне `\x80..\xff` следует за UTF-8 строкой, например:

```
1 utf8::upgrade(  my $u = "\x{e5}");
2 utf8::downgrade(my $d = "\x{e5}");
3 /$u$d/
```

  [RT #118297]

- В регулярных выражениях, содержащих множество кодовых блоков, значения `$1`, `$2`, и т.д. устанавливаемые вложенными вызовами регулярных выражений будут утекать из одного блока в другой. Теперь эти переменные всегда ссылаются на внешнее регулярное выражение в начале вложенного блока [perl #117917].

- `/$qr/p` был сломан в Perl 5.18.0; флаг `/p` игнорировался. Это было исправлено. [perl #118213]

- Начиная с Perl 5.18.0, конструкции вида `/[#](?{})/x` будут неправильно интерпретировать # как комментарий. Кодовый блок будет пропущен без разбора. Это было исправлено.

- Начиная с Perl 5.001, регулярные выражение вида `/[#$a]/x` или `/[#]$a/x` будут неправильно интерпретировать # как комментарий, поэтому переменная не будет интерполироваться. Это было исправлено. [perl #45667]

- Perl 5.18.0 нечаянно стал разыменовывать регулярное выражение (`${ qr// }`) как логическое значение ложь. Это было исправлено.

- Использование `\G` в регулярных выражениях, где он находится не в начале шаблона, теперь стало значительно менее глючным (тем не менее это по-прежнему проблемная вещь).

- Там, где регулярное выражение включает блоки кода (`/(?{...})/`) и где использование перегрузки констант вызывает перекомпиляцию блока кода, вторая компиляция не видит своего внешней лексической области видимости. Это является регрессией в Perl 5.18.0.

- Позиция в строке, задаваемая `pos` может сдвинуться, если строка меняет своё внутреннее представление в или из utf8. Это может происходить, например, со ссылками на объекты с перегрузкой строкового представления.

- Получение ссылок на возвращаемое значение двух вызовов `pos` с одинаковым аргументом и последующее присвоение одной ссылке и `undef` для другой может привести к ошибкам утверждения или утечкам памяти.

- Элементы @- и @+ теперь обновляются корректно, если они ссылаются на несуществующие захваты. Ранее элемент, на который ссылались (`$ref = \$-[1]`), мог ссылаться на неправильное совпадение после последовательных операций по поиску совпадений.

- Код, который разбирал ссылки на совпадения в регулярных выражениях (или код символа в восьмиричном представлении), такие как \123, делал простой atoi(), который мог приводиться к отрицательному значению для длинных цифровых строк и приводить к ошибке сегментации. Это было исправлено [perl #119505].

- Присвоение другого typeglob к `*^R` больше не приводит к краху движка регулярных выражений.

- Экранирование `\N` в регулярном выражении, при использовании без фигурных скобок (для обозначения `[^\n]`), игнорировало последующий `*`, если за ним следовал пробел при нали-

чии модификатора /x. Это работало так, как `\N` стал обозначать `[^\n]`, начиная с релиза 5.12.0.

- `s///`, `tr///` и `y///` теперь работают, если используется широкий символ как разделитель. [perl #120463]

- Некоторые случаи нетерминированной (?...) последовательности в регулярных выражениях (например, `/(?<!--`) были исправлены, чтобы выдавать корректное сообщение об ошибке, вместо “panic: memory wrap”. Другие случаи (например, `/(?(/`) ещё предстоит исправить.

  - Когда ссылка на ссылку перегруженного объекта возвращается из блока кода регулярного выражения (`??{...}`), может произойти некорректное скрытое разыменование, если внутренняя ссылка уже была возвращена из блока кода ранее.
  - Связанная переменная, возвращаемая из (`??{...}`) видит внутренние значения переменных поиска совпадений (например, $1 и т.д. из любого совпадения внутри блока) в своём FETCH методе. Но этого не происходит, если ссылка на перегруженный объект была последним элементом присвоенным связанной переменной. Вместо этого совпавшая переменная ссылается на внешний шаблон при выполнении вызова FETCH.
  - Исправлено неожиданное связывание через регулярное выражение с использованием локали. Раньше, при некоторых условиях использование классов символов могло приводить к связыванию, в то время как этого не должно было происходить. Некоторые классы символов являются локале-зависимыми, но до этого патча, иногда связывание происходило даже если классы символов не зависели от локали. [perl #120675]
  - При некоторых условиях Perl мог выдать ошибку, если в lookbehind операторе утверждения в регулярном выражении, оператор ссылается на именованный подшаблон, жалуясь, что lookbehind являлся переменной, в то время как он таковым не являлся. Это было исправлено. [perl #120600], [perl #120618]. Текущее исправление может быть улучшено в будущем.

- `$^R` не был доступен вне регулярного выражения, которое инициировало его. [perl #121070]
- Был включён большой набор исправлений и переработки для re_intuit_start(), включая:
  * Исправление паники при компиляции регулярного выражения `/\x{100}[xy]\x{100}{2}/`.
  * Исправлено регрессия в производительности при выполнении поиске глобальных совпадений в шаблоне для UTF-8 строки. [perl #120692]
  * Исправлена другая проблема с производительностью, где поиск совпадения в регулярном выражении вида `/ab.{1,2}x/` в длинной UTF-8 строке бесполезно вычисляет байтовый отступ для большой части строки. [perl #120692]
- Исправлена ошибка выравнивания при компиляции регулярного выражения, если был использована сборка с GCC на HP-UX 64 бит.
- На 64-битных платформах `pos` теперь может быть установлен в значение больше чем 2**31-1. [perl #72766]

Отладчик Perl 5 и -d

- Команда `man` отладчика была исправлена. Она была сломана в релизе v5.18.0. Команда `man` сделана синонимом имён `doc` и `perldoc` - теперь всё снова работает.
- `@_` теперь корректно видна в отладчике, исправляя регрессию, появившуюся в отладчике v5.18.0. [RT #118169]
- Для сборок с включённым копированием-при-записи (по умолчанию в 5.20.0) `${'_<-e'}[0]` больше не искажается. Это первая строка ввода, сохраняемая для использования отладчиком, при использовании однострочников [perl #118627].
- В сборках без поддержки тредов, установка `${"_<filename"}` ссылке или typeglob'у больше не приводит к тому, что `__FILE__` и некоторые сообщения об ошибках выводят испорченную строку и не препятствует директивам `#line` в строчных eval передавать строку исходников отладчику. Сборки с поддержкой тредов не затронуты.

- Начиная с Perl 5.12, номера строки были меньше на единицу, если переключатель **-d** был использован в строчке с #!. Теперь это исправлено.
- `*DB::DB = sub {} if 0` больше не препятствует режиму отладки Perl в поиске объявления подпрограммы `DB::DB` далее в программе.
- Хеши `%{'_<...'}` теперь устанавливают точки останова в соответствующие `@{'_<...'}` вместо произвольного массива, который сделан синонимом `@DB::dbline`. [perl #119799]
- Вызывается set-magic, при установке $DB::sub. [perl #121255]
- Команда отладчика “n” теперь учитывает lvalue подпрограммы и проходит через них [perl #118839].

Лексические подпрограммы

- Лексические константы (`my sub a(){ 42 }`) больше не приводят к краху при встраивании.
- Прототипы параметров, привязываемые к лексическим подпрограммам теперь учитывают при компиляции вызовы подпрограмм без скобок. Ранее прототипы учитывались только при вызове *со* скобками. [RT #116735]
- Синтаксические ошибки в лексических подпрограммах в комбинации с вызовами той же подпрограммы больше не приводят к краху во время компиляции.
- Предупреждения о глубокой рекурсии больше не приводят к краху лексических подпрограмм. [RT #118521]
- Проверка подпрограмм в dtrace теперь работает и с лексическими подпрограммами, вместо краха [perl #118305].
- Неопределённая и встроенная лексическая подпрограмма (`my sub foo(){ 42 } undef &foo`) приводила к краху, если были включены предупреждения.
- Неопределённая лексическая подпрограмма, используемая как наследуемый метод, больше не приводит к краху.
- Присутствие лексической подпрограммы с именем “CORE” больше не мешает работать суффиксу CORE::.

Всё остальное

- Код выделения OP теперь возвращает корректно выровненную память во всех случаях для `struct pmop`. Ранее он возвращал память выровненную по 4-байтной границе, что неправильно для ithreads сборок с 64-битным IV и на некоторых 32-битных платформах. Это приводило к полному краху сборки на sparc GNU/Linux. [RT #118055]
- Вычисление больших хешей в скалярном контексте теперь значительно быстрее, так как число используемых цепочек теперь кэшируется для больших хешей. Небольшие хеши продолжают не сохранять это значение, а вычислять когда требуется, так как это экономит один IV. Это также даст накладные расходы в 1 IV для любого объекта сделанного из хеша. [RT #114576]
- В Perl v5.16 появилась ошибка, в которой вызов XSUB, которая не была видна во время компиляции расценивалась как lvalue и, что к ней можно присваивать значения, даже если подпрограмма не являлась lvalue. Это было исправлено. [RT #117947]
- В Perl v5.18.0 двойные переменные, которые имели пустую строку для строковой части и ненулевое число для цифровой части стали расцениваться как истинное значение. В предыдущих версиях они трактовались как ложное значение, строковое представление имело более высокий прецедент. Было восстановлено старое поведение. [RT #118159]
- Начиная с Perl v5.12, встраивание констант, которые переназначали встроенные ключевые слова с тем же именем отменяли `use subs`, заставляя последующие упоминания констант использовать встроенное ключевое слово. Это было исправлено.
- Предупреждение, выводимое `-l $handle`, теперь относится к IO ссылкам и глобам, а не только ссылкам глобов. Предупреждение также теперь содержит корректный UTF-8. [RT #117595]
- `delete local $ENV{nonexistent_env_var}` больше не приводит к утечкам памяти.

- `sort` и `require` с последующим ключевым словом с префиксом `CORE::` теперь рассматривают его как ключевое слово, а не имя подпрограммы или модуля. [RT #24482]
- Через определённые головоломные трюки возможно вызвать освобождение текущего пакета. Некоторые операторы (`bless`, `reset`, `open`, `eval`) могут не обработать такую ситуацию и привести к краху. Они теперь сделаны более гибкими. [RT #117941]
- Создание псевдонима файлового дескриптора через присвоение глоб глобу не обновит внутренний метод для корректного кэширования, если существует пакет с тем же именем, что и файловый дескриптор, приводя не к вызову метода файлового дескриптора, а вызову метода пакета. Теперь это исправлено.
- `./Configure -de -Dusevendorprefix` не был по умолчанию. [RT #64126]
- Предупреждение `Statement unlikely to be reached` было указано в perldiag как предупреждение категории `exec`, но включалось и выключалось через категорию `syntax`. Теперь оно полностью поддерживается категорией `exec`.
- Предупреждение “Replacement list is longer that search list” для `tr///` и `y///` больше не происходит в присутствии `/c` флага. [RT #118047]
- Преобразование к строке NV больше не кэшируется, так как лексическая локаль контролирует преобразование к строке десятичной запятой. [perl #108378] [perl #115800]
- Было сделано несколько исправлений, относящихся к поддержке локалей в Perl. perl #38193 был описан выше в “Internal Changes”. Также был исправлен #118197, где символ десятичной запятой должен был быть ASCII символом (что не работает для некоторых незападных языков); и #115808, в котором `POSIX::setlocale()` при ошибке возвращала `undef` и не выдавала никаких предупреждений о неопределённости, даже если подобные предупреждения были включены.
- Компиляция оператора `split`, чей третий аргумент является именованной константой, вычисляемой в 0, больше

не вызывает изменение значения константы.

– Именованная константа, используемая как второй аргумент для `index` больше не приводится к строке если это ссылка, регулярное выражение, двойственная переменная и т.д.

– Именованная константа, вычисляемая в неопределённое значение, используемое как второй аргумент `index` больше не выводит “неинициализированное” предупреждение при компиляции. Но по прежнему выводит его во время выполнения программы.

– Если из подпрограммы в @INC был возвращён скаляр, указанный скаляр магически конвертировался в IO сущность, возможно приводя к ошибке “Bizarre copy”, если этот скаляр продолжал где-либо использоваться. Теперь Perl использует внутреннюю копию скаляра.

– Некоторые случаи использования оператора `sort` оптимизированы модифицировать массив на месте, как например, `@a = sort @a`. При этой сортировке массив становится доступным только на чтение. Если блок сортировки мог умереть, то массив оставался доступным только на чтение даже вне `sort`. Это было исправлено.

– `$a` и `$b` внутри блока sort являются алиасами актуальных аргументов `sort`, поэтому они могут быть изменены через эти две переменные. Это не всегда работало, например, для lvalue подпрограмм и `$#ary`, и, возможно, многих других операторов. Теперь это работает.

– Аргументы для `sort` теперь все в списочном контексте. Если `sort` сам по себе был вызван в пустом или скалярном контексте, тогда *некоторые*, но не все аргументы оказывались в пустом или скалярном контексте.

– Прототипы подпрограммы с символами Юникода выше U+00FF искажались при клонировании замыкания. Это происходило с подпрограммами замыкающимся на лексических переменных, определённых во вне, и с лексическими подпрограммами.

– `UNIVERSAL::can` теперь рассматривает свой первый аргумент также, как вызов метода do: typeglob’ы и глобссылки с непустыми IO слотами рассматриваются как

дескрипторы, а строки рассматриваются как файловый дескрипторы, а не пакет, если существует дескриптор с этим именем [perl #113932].

  - Вызов метода на typeglob'е (например, `*ARGV--->bar` вызывать метод пакета "foo" (как `foo->bar`). В некоторых случаях это приводило к тому, что метод вызывался на неверном дескрипторе. Теперь аргумент typeglob рассматривается как дескриптор (также как и `(\*foo)->bar`), или, если IO-слот пустой, выводится ошибка.

- Присвоение vstring к связанной переменной или к аргументу подпрограммы, который является алиасом к несуществующему элементу хеша или массива, теперь работает без приведения vstring к обычной строке.

- `pos`, `tie`, `tied` и `untie` не работали нормально на аргументах подпрограмм, которые являлись алиасами к несуществующим элементам хеша или массива [perl #77814, #27010].

- Оператор толстой запятой `=>` может теперь брать в кавычки встроенные ключевые слова, даже если они появляются на следующей строке, приводя в соответствие поведение с другими простыми словами.

- Автовивификация заглушки подпрограммы через `\&$glob` стало приводить к краху в Perl 5.18.0, если $glob был просто копией реального глоба, т.е. скаляр, которому был присвоен глоб. Это было исправлено. [perl #119051]

- Раньше Perl допускал утечку деталей реализации, когда происходила ссылка на возвращаемые значения у некоторых операторов. `for ($a+$b){ warn \$_; warn \$_ }` раньше отображал два различных адреса памяти, поскольку оператор `\` производил копирование переменной. В сборках с поддержкой тредов это также происходило для констант (`for(1){ ... }`). Теперь это исправлено. [perl #21979, #78194, #89188, #109746, #114838, #115388]

- Оператор диапазона `..` возвращал те же самые модифицируемые скаляры с каждым вызовом, за исключением случая если он был единственным внутри заголовка цикла `foreach`. Это означало, что изменения в значениях внутри возвращаемого

списка были видны при следующем выполнении оператора. [perl #3105]

- Развёртывание констант и встраивание подпрограмм больше не приводит к возвращению значений, доступных только на чтение, вместо обычного возврата новых значений модифицируемых скаляров.

- Замыкания вида `sub (){ $some_variable }` больше не встраиваются, что приводило к игнорированию изменения переменной, вызывающему подпрограмму коду. [perl #79908]

- Возвращаемые значения некоторых операторов, таких как `ref` иногда разделялось между рекурсивными вызовами к той же подпрограмме, приводя к тому, что внутренние вызовы модифицировали значение, возвращаемое `ref` во внешних вызовах. Это было исправлено.

- `__PACKAGE__` и константы, возвращающие имя пакета или ключ хеша теперь согласованно доступны только для чтения. В различных предыдущих релизах Perl они становились изменяемыми при некоторых условиях.

- Включение предупреждений “used once” больше не приводит к краху на циклических стэшах, созданных во время компиляции (`*Foo::Bar::Foo:: = *Foo::`).

- Неопределённые константы, используемые в ключах хэша (`use constant u => undef; $h{+u}`) больше не вызвают предупреждение о неинициализированном значении во время компиляции.

- Изменяя цель замены внутри подстановки замены больше не приводит к краху.

- Первый оператор внутри строкового eval раньше иногда использовал неверные настройки прагмы при изменении регистра константы. `eval 'uc chr 0xe0'` мог случайным образом выбирать между Юникодом, байтовым представлением или семантикой локали. Это было исправлено.

- Поддержка возвращенных значений @INC фильтров (подпрограммы, возвращённые подпрограммами в @INC) были исправлены. Ранее связанная переменная не поддерживалась и

установка $_ в ссылку на typeglob могла приводить к краху.

- XS-функция `SvPVbyte` была исправлена для работы со связанными скалярами, возвращающие нечто отличное от строки. Она возвращала utf8 в тех случаях, где должна была вернуть `SvPV`.

- В Perl 5.18.0 `--` и `++` приводили к краху при операции на разыменованном регулярном выражении, а `++` перестал приводить к строке vstrings.

- `bless` больше не умирает с “Can’t bless non-reference value”, если его первый аргумент является связанной ссылкой.

- `reset` с аргументом больше не пропускает копируемым-при-записи скаляры, регулярные выражения, копии typeglob и vstring. Также, когда он наталкивается на эти, или значения, доступные только на чтение, он больше не пропускает массивы или хеши с тем же самым именем.

- `reset` с аргументом теперь пропускает скаляры, которые являются алиасами на typeglob’ы (`for $z (*foo){ reset "z" }`). Ранее он мог повредить память или привести к краху.

- `ucfirst` и `lcfirst` не учитывали прагму bytes. Это была регрессия, начиная с Perl 5.12. [perl #117355]

- Изменения в `UNIVERSAL::DESTROY` теперь обновляют кэши DESTROY во всех классах, тогда как раньше классы, которые уже удаляли объекты, продолжали использовать старую подпрограмму. Это была регрессия в Perl 5.18. [perl #114864]

- Все известные ложные срабатывания предупреждения об устаревании “Useless use of ‘\’; doesn’t escape metacharacter ‘%c’ ”, добавленного в Perl 5.18.0, были удалены. [perl #119101]

- Значение $^E теперь сохраняется между обработчиками сигналов на Windows. [perl #85104]

- Лексические файловый дескриптор (такой как `open my $fh...`) обычно получает имя основанное на текущем пакете и имени переменной, например, “main::$fh”.”fh” часть имени. Это было исправлено.

- Неинициализированное значение, возвращаемое XSUBs больше не освобождается от предупреждения о неинициализированном значении. [perl #118693]

- `elsif ("")` больше не выводит ошибочное предупреждение о пустом контексте. [perl #118753]

- Передавая `undef` подпрограмме теперь приводит к тому, что @_ содержит такой же, доступный только на чтение, неопределённый скаляр, который возвращает `undef`. Более того, `exists $_[0]` теперь возвращает истину, если `undef` был первым аргументом. [perl #7508, #109726]

- Передача несуществующего элемента массива подпрограмме обычно не автовивифицирует его, если только подпрограмма не изменит свой аргумент. Это не работало корректно для отрицательных индексов и с несуществующими элементами внутри массива. Элемент автовивифицировался немедленно. Отложенная автовивификация была применена для работы с такими случаями. [perl #118691]

- Присваивая ссылки или глобы скаляру, возвращаемому $#foo после того, как массив @foo был освобождён не приводит к ошибкам утверждения на сборках с отладочными символами и утечкам памяти на обычных сборках.

- На 64-битных платформах, большие диапазоны, как например, 1..1000000000000 больше не приводят к краху, но пожирают всю вашу память. [perl #119161]

- `__DATA__` теперь размещает `DATA` дескриптор в правильный пакет, даже если текущий пакет был переименован через присвоение глобу.

- Когда `die`, `last`, `next`, `redo`, `goto` и `exit` перематывают область видимости, возможно, что `DESTROY` рекурсивно вызовет подпрограмму или формат, который уже завершился. В этом случае, иногда лексические переменные внутри подпрограммы получали значения из внешнего вызова, вместо неопределённых значений, которые они должны иметь. Это было исправлено. [perl #119311]

- $^{M}PEN\{\hat{}\text{MATCH}\}$.

- Perl теперь гораздо лучше пытается вернуть корректный номер строки в `(caller)[2]`. [perl #115768]

- Номера строк внутри многострочных операторов цитирования теперь выводятся корректно. [perl #3643]

- Директива `#line` внутри кода, встроенного в операторах цитирования, теперь учитывается.

- Номера строк теперь вычисляются корректно внутри второй встроенной документации, когда два маркера встроенной документации появляются на одной строке.

- Оптимизация в Perl 5.18 делала некорректные предположения, которые приводили к плохому взаимодействию со CPAN модулем Devel::CallParser. Если модуль загружался, то лексические переменные, объявленные в другом операторе, следующим за списком `my(...)`, могли не очищаться при выходе из области видимости.

- `&xsub` и `goto &xsub` вызовы теперь позволяют вызываемой подпрограмме автовивифицировать элементы @_.

- `&xsub` и `goto &xsub` больше не приводят к краху, если *_ становился неопределённым и не имел элемента ARRAY (т.е. @_ не существовал).

- `&xsub` и `goto &xsub` теперь работает со связанным @_.

- Слишком длинные идентификаторы больше не приводят к переполнению буфера (и краху). Как стало происходить в 5.18.

- Предупреждение “Scalar value @hash{foo} better written as $hash{foo}” теперь производит гораздо меньше ложных срабатываний. В частности, `@hash{+function_returning_a_list}` и `@hash{ qw "foo bar baz" }` больше не выводят предупреждений. То же самое относится к срезам массива . [perl #28380, #114024]

- `$! = EINVAL; waitpid(0, WNOHANG);` больше не входит во внутренний бесконечный цикл. [perl #85228]

- Возможная ошибка сегментации при дублировании файлового дескриптора была исправлена.

- Подпрограмма в @INC может вернуть ссылку на скаляр, содержащую начальное содержимое файла. Однако, этот скаляр мог быть преждевременно освобождён, если на него нигде не ссылались, приводя к произвольным результатам.

- `last` больше не возвращает значения, которые тот же оператор аккумулировал до сих пор, исправляя среди прочего давно известный баг, когда `push @a, last` попытается вернуть @a, копируя его как скаляр в процессе и заканчивая ошибкой, “Bizarre copy of ARRAY in last.” [perl #3112]

- В некоторых случаях, закрывая файловый дескриптор, открытый в канал в или из процесса, который был продублирован в стандартный дескриптор, вызовет внутреннюю обёртку `waitpid` perl’a с нулевым PID, вероятно блокируя процесс. Это ожидание нулевого процесса больше не происходит. [perl #119893]

- `select` раньше игнорировал магию на своём четвёртом аргументе (таймаут), приводя к эффекту, когда `select` блокировался бесконечно вместо ожидаемого времени сна. Это было исправлено. [perl #120102]

- Имя класса в `for my class $foo` теперь определяется верно. В случае, если второй символ имени класса был цифрой (например, ‘a1b’), раньше выдавалась ошибка “Missing $ on loop variable”. [perl #120112]

- Perl 5.18.0 случайно запретил `-bareword` при `use strict` и `use integer`. Это было исправлено. [perl #120288]

- `-a` в начале строки (или дефис с одиночной буквой, который не является оператором файлового теста) больше не приводит к ошибочному предупреждению ‘Use of “-a” without parentheses is ambiguous’. [perl #120288]

- lvalue контекст теперь правильно распространяется на простые блоки и блоки `if` и `else` в lvalue подпрограммах. Ранее, массивы и хеши иногда некорректно приводились к скаляру, когда возвращались в списочном lvalue-контексте или происходила ошибка “Bizarre copy”. [perl #119797]

- Lvalue контекст теперь распространяется на ветви `||` и `&&` (и их алфавитные эквиваленты `or` и `and`). Это значит, что `foreach (pos $x || pos $y){...}` теперь позволяет `pos` модифицироваться через `$_`.

- `stat` и `readline` помнят последний используемый дескриптор; первый для специального `_` файлового дескриптора, второй для `${^LAST_FH}`. `eval "*foo if 0"`, где `*foo` является последним дескриптором, переданным в `stat` или `readline`, приведёт к тому, что дескриптор будет забыт, если дескриптор ещё не был открыт. Это было исправлено.

- Различные случаи `delete $::{a}`, `delete $::{ENV}` и т.д., приводившие к краху были исправлены. [perl #54044]

- Установка `$!` в EACCESS перед вызовом `require` затронет поведение `require`. Это было исправлено.

- Предупреждение “Can’t use \1 to mean $1 in expression” теперь выводится только на правую часть (замена) замещения. Ранее это могла происходить в коде, встроенном в левую часть или в любом другом операторе цитирования.

- Благословение в ссылку (`bless $thisref, $thatref`) было давно запрещено, но магические скаляры для второго, например, `$/` и те, что были связаны, являлись исключениями. Теперь больше нет. [perl #119809]

- Благословение в ссылку было нечаянно разрешено в 5.18, если аргумент класса был благословенной ссылкой с устаревшим кэшем метода (т.е. чей класс имел определённые подпрограммы, начиная с последнего вызова метода). Теперь они запрещены ещё раз, так же как в 5.16.

- `$x->{key}`, где $x`myClass` больше не приводит к краху, если заглушка подпрограммы Class::FIELDS была объявлена.

- `@$obj{'key'}` и `${$obj}{key}` раньше исключались из проверки полей на этапе компиляции (“No such class field”; смотрите fields), но теперь больше нет.

- Несуществующий элемент массива с большим индексом, переданным подпрограмме, которая связывает массив и затем

пытается получить доступ к элементу, больше не приводит к краху.

- Объявление заглушки подпрограммы, названной NEGATIVE_INDICES, больше не приводит к краху для отрицательных индексов, когда текущий пакет является классом связанного массива.

- Объявление заглушек подпрограмм `require`, `glob` или `do` в пакете CORE::GLOBAL:: больше не приводит к краху при вызове соответствующих функций при компиляции.

- Создание алиасов для функций CORE::GLOBAL:: на константы перестало работать в Perl 5.10, но теперь было исправлено.

- Когда `` `...` `` или `qx/.../` вызывают переопределение `readpipe`, теперь происходит интерполяция как в двойных кавычках, также, как в случае без переопределения. Раньше, наличие переопределения могло заставить операторы работать как `q{}`, подавляя интерполяцию. [perl #115330]

- Встроенная документация `<<`...`` (с обратными кавычками в качестве разделителя) теперь вызывают замену `readpipe`. [perl #119827]

- `&CORE::exit()` и `&CORE::die()` теперь учитывают хинты vmsish

- Установка неопределённого значения глобу, которую вызывает метод DESTROY, который устанавливает неопределённое значение тому же глобу, теперь безопасно. Раньше это выводило предупреждение “Attempt to free unreferenced glob pointer” и приводило к утечке памяти.

- Если переопределение подпрограммы (`eval 'sub foo{}'` или `newXS` для XS-кода) вызывало метод DESTROY на подпрограмме, которая была переопределена и этот метод присваивает подпрограмму тому же слоту (`*foo = sub {}`), `$_[0]` больше не остаётся с указателем на освобождённый скаляр. Теперь DESTROY задерживается, пока новая подпрограмма не будет установлена.

- На Windows, perl больше не вызывает CloseHandle() на дескрипторе сокета. Это делает отладку на Windows проще,

удаляя некоторые неуместные исключения с плохими дескрипторами. Это также исправляет условие гонки, которая приводит функции над сокетами к случайным ошибкам в Perl процессе с несколькими тредами ОС и к возможным ошибкам тестов в `dist/IO/t/cachepropagate-tcp.t`. [perl #120091/118059]

- Форматы, включающие строки, кодированные в UTF-8, или странные переменные, такие как связанные, перегруженные или ссылки, преобразованные к строке (и в последних perl, чистые NOK-переменные), будут в целом делать неправильные вещи. В форматах, где переменная рассматривается как строка и повторно chop'ится, как в `^<<<~~` или подобных. Это теперь исправлено. [perl #33832/45325/113868/119847/119849/119851]

- `semctl(..., SETVAL, ...)` установит семафор на верхних 32 битах переданного целого, вместо нижних 32 бит на 64-битных big-endian системах. [perl #120635]

- `readdir()` теперь устанавливает `$!` только при ошибках. `$!` больше не устанавливается в `EBADF`, когда из директории читается завершающий `undef`, только если системный вызов не установит `$!`. [perl #118651]

- `&CORE::glob` больше не вызывает периодических крахов, когда стек perl'а становится повреждённым. [perl #119993]

- `open` со слоями, которые загружают модули (например, `<: encoding(utf8)`) больше не рискуют обрушиться из-за повреждения стека.

- Perl 5.18 сломал автозагрузку через вызов метода `->SUPER::foo` выполняя поиск AUTOLOAD из текущего пакета, а не суперкласса текущего пакета. Это было исправлено. [perl #120694]

- Давно известная ошибка, приводящая к тому, что `do {} until CONSTANT`, где константа содержит истинное значение, читает невыделенную память, была исправлена. Это обычно происходит после синтаксической ошибки. В предыдущих версиях Perl это приводило к периодическим крахам. [perl #72406]

- Исправлено падение `$!` на HP-UX. strerror() HP-UX возвращает пустую строку для неизвестного кода ошибки. Это вызывает ошибку утверждения в сборках с отладкой DEBUGGING. Теперь возвращаемая строка для "$!" содержит текст, который указывает, что это код для неизвестной ошибки.

- Индивидуально связанный элемент @INC (как в `tie $INC [0]...`) теперь поддерживает корректно. Раньше, будет ли подпрограмма, возвращённая таким связанным элементом, воспринята как подпрограмма, зависело от того был ли выполнен FETCH ранее.

- `getc` на байтно-размерном дескрипторе, после такого же `getc` оператора, используемого на utf8 дескрипторе, воспринимал байты как utf8, приводя к ошибочному поведению (например, вывод предупреждений о неправильно сформированном UTF-8).

- Начальный `{` в начале строки аргумента формата всегда интерпретировался как начало блока до v5.18. В Perl v5.18, он стал восприниматься как двусмысленный элемент. Парсер пытался угадать является ли это конструктором анонимного хеша или блок в зависимости от содержимого. Теперь предыдущее поведение было восстановлено. [perl #119973]

- В Perl v5.18 `undef *_; goto &sub` и `local *_; goto &sub` стали падать. Это было исправлено. [perl #119949]

- Обратные кавычки (`` `` или `qx//`), совместно с несколькими тредами на Win32 могли приводить к захвату вывода, отправленному на stdout одного треда, обратными кавычками внешней команды в другом треде. Это могло происходить также и для процессов псевдо-форка, так как на Win32 псевдо-форк реализован через треды. [perl #77672]

- `open $fh, ">+", undef` больше не теряет память, если TMPDIR установлена, но указывает на директорию, где временные файлы не могут быть созданы. [perl #120951]

- `for ( $h{k} || '' )` больше не автовивифицирует `$h{k}`. [perl #120374]

- На Windows машинах Perl теперь эмулирует POSIX использование окружения для инициализации локали. Ранее окружение игнорировалось. Смотрите “ENVIRONMENT” in perllocale.
- Исправлен крах при уничтожении само-ссылающегося GLOB. [perl #121242]

Известные проблемы

- IO::Socket известно, что не проходит тесты на AIX 5.3. Есть патч в request tracker, #120835, который может быть применён в последующих релизах.
- Следующие модули известны, что имеют проблемы с тестами с этой версией Perl. Патчи были отправлены, поэтому надеемся, что новые релизы появятся скоро:
  - Data::Structure::Util вресия 0.15
  - HTML::StripScripts версия 1.05
  - List::Gather версия 0.08.

Некролог

Диана Роза, 27, Рио-де-Жанейро, ушла на долгий покой 10 мая 2014 года вместе с плюшевым верблюдом, который она всегда держала рядом с монитором компьютера. Она была страстным Perl хакером, которая любила язык и своё сообщество, и, которая никогда не пропускала встречи Rio.pm. Она была настоящим мастером своего дела, энтузиастом по написанию кода, исполняла арии и расписывала стены граффити. Мы никогда не забудем тебя.

Грег МакКэрол умер 28 августа 2013.

Грег был хорошо известен по многим причинам. Он был одним из организаторов первой YAPC::Europe, которая закончилась незапланированным аукционом, где он неистово пытался собрать дополнительные средства, чтобы избежать убытка конференции. Это был Грег, который по ошибке прибыл на встречу london.pm на неделю

раньше; несколькими годами позже он был одним из тех, кто продал выбор даты официальной даты на аукционе YAPC::Europe и в конечном счёте как прекрасный лидер london.pm он получил в наследство непочтительную путаницу, которую он создал.

Всегда полезный, дружественный и весёлый оптимист, тебя будет не хватать, ты никогда не будешь забыт.

Благодарности

Perl 5.20.0 представляет собой примерно 12 месяцев разработки, начиная с Perl 5.18.0 и содержит примерно 470 000 изменённых строк среди 2 900 файлов от 124 авторов.

Исключая авто-генерированные файлы, документацию и утилиты для релиза, было изменено около 280 000 строк в 1 800 .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 5.20.0:

Aaron Crane, Abhijit Menon-Sen, Abigail, Abir Viqar, Alan Haggai Alavi, Alan Hourihane, Alexander Voronov, Alexandr Ciornii, Andy Dougherty, Anno Siegel, Aristotle Pagaltzis, Arthur Axel ‘fREW’ Schmidt, Brad Gilbert, Brendan Byrd, Brian Childs, Brian Fraser, Brian Gottreu, Chris ‘BinGOs’ Williams, Christian Millour, Colin Kuskie, Craig A. Berry, Dabrien ‘Dabe’ Murphy, Dagfinn Ilmari Mannsåker, Daniel Dragan, Darin McBride, David Golden, David Leadbeater, David Mitchell, David Nicol, David Steinbrunner, Dennis Kaarsemaker, Dominic Hargreaves, Ed Avis, Eric Brine, Evan Zacks, Father Chrysostomos, Florian Ragwitz, François Perrad, Gavin Shelley, Gideon Israel Dsouza, Gisle Aas, Graham Knop, H.Merijn Brand, Hauke D, Heiko Eissfeldt, Hiroo Hayashi, Hojung Youn, James E Keenan, Jarkko Hietaniemi, Jerry D. Hedden, Jess Robinson, Jesse Luehrs, Johan Vromans, John Gardiner Myers, John Goodyear, John P. Linderman, John Peacock, kafka, Kang-min Liu, Karen Etheridge, Karl Williamson, Keedi Kim, Kent Fredric, kevin dawson, Kevin Falcone, Kevin

Ryde, Leon Timmermans, Lukas Mai, Marc Simpson, Marcel Grünauer, Marco Peereboom, Marcus Holland-Moritz, Mark Jason Dominus, Martin McGrath, Matthew Horsfall, Max Maischein, Mike Doherty, Moritz Lenz, Nathan Glenn, Nathan Trapuzzano, Neil Bowers, Neil Williams, Nicholas Clark, Niels Thykier, Niko Tyni, Olivier Mengué, Owain G. Ainsworth, Paul Green, Paul Johnson, Peter John Acklam, Peter Martini, Peter Rabbitson, Petr Písař, Philip Boulain, Philip Guenther, Piotr Roszatycki, Rafael Garcia-Suarez, Reini Urban, Reuben Thomas, Ricardo Signes, Ruslan Zakirov, Sergey Alekseev, Shirakata Kentaro, Shlomi Fish, Slaven Rezic, Smylers, Steffen Müller, Steve Hay, Sullivan Beck, Thomas Sibley, Tobias Leich, Toby Inkster, Tokuhiro Matsuno, Tom Christiansen, Tom Hukins, Tony Cook, Victor Efimov, Viktor Turskyi, Vladimir Timofeev, YAMASHINA Hio, Yves Orton, Zefram, Zsbán Ambrus, 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 баг-трекер.

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.)*

4. Обзор CPAN за май 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 220
- Новых выпусков — 880

Новые модули

- Test::Text

Модуль для проверки орфографии и качества текста. Примечательна история появления этого модуля. Автор модуля *Juan Julián Merelo Guervós* пишет роман «Мануэль: Великолепный Механический человек», текст которого распространяется под свободной лицензией. Данный модуль стал частью системы непрерывной интеграции, которая позволяет в автоматическом режиме выявлять ошибки в тексте романа.

- YAPC::Russia

Модуль `YAPC::Russia` позволит вам узнать о дате и месте проведения очередной конференции YAPC::Russia. Возможно, в будущем модуль позволит регистрироваться на конференции, узнавать расписание и списки посетителей. Поживём, увидим.

- CGI::Fast

Старый добрый `CGI::Fast` обрёл самостоятельную жизнь, отделившись от дистрибутива `CGI`. Сделано это было для того, чтобы `CGI`

не имел обязательной зависимости от модуля FCGI. Поскольку, как только бы модуль CGI покинул базовый дистрибутив perl, установить его со CPAN стало бы нетривиальной задачей (поскольку CGI требует FCGI, а для сборки FCGI нужен компилятор).

- Test::Shadow

Модуль Test::Shadow главным образом предназначен для mock-тестирования. Близок по духу тестовому фреймворку RSpec-mocks из мира Ruby, но имеет более гибкий и ясный интерфейс. Позволяет переопределять некоторые методы тестируемого класса, задавать их поведение и возвращаемые значения.

- AnyEvent::MySQL

AnyEvent::MySQL — это реализация асинхронного клиента протокола MySQL на чистом Perl на базе AnyEvent. Модуль не использует ни DBI, ни DBD::mysql, самостоятельно реализуя интерфейс, близкий к DBI, но с особенностями, характерными для асинхронного кода. Автор заверяет, что модуль уже два года используется в боевом окружении, и данный релиз по сути является публикацией кода под открытой лицензией.

- Template::Jade

Модуль Template::Jade является портом движка шаблонов Jade из Node.js для Perl. Поддерживается только разметка HTML5, реализованы ещё не все возможности Jade.

- Router::R3

Router::R3 — это мощная и быстрая XS-библиотека для URL-роутинга. Реализована как обвязка к C-библиотеке libr3.

- Term::Drawille

Данный модуль позволяет вам рисовать различные геометрические фигуры в терминале, используя для этого символы из шрифта Брайля (символы состоят из комбинации точек). Идея честно украдена из аналогичного python-модуля drawille

- File::Slurp::Sane

`File::Slurp::Sane` — это замена для `File::Slurp`. `File::Slurp` может испортить ваши данные, если вы попытаетесь читать или писать файл в режиме открытия файла, отличном от `:raw`. `File::Slurp::Sane` решает проблему с чтением и записью текстовых файлов в любой заданной кодировке.

Обновлённые модули

- Minilla v1.0.0

Вышел первый мажорный релиз системы подготовки дистрибутивов для CPAN Minilla. В новом релизе произошёл переход на систему сборки `Module::Build::Tiny` по умолчанию, вместо уже не рекомендуемого `Module::Build`.

- File::LibMagic 1.01

Новый релиз `File::LibMagic` содержит исправление ошибки с утечкой файловых дескрипторов.

- IO::Socket::SSL 1.991

В новых майских релизах `IO::Socket::SSL` появилась поддержка протокола OCSP для онлайн-запросов состояния сертификатов без загрузки CRL-списков.

- LaTeXML 0.8.0

В новом релизе конвертера формата TeX/LaTeX в XML/HTML/MathML появилась поддержка HTML5 и ePub. Значительно ускорена генерация изображений для математических формул из TeX.

- Mojolicious 5.02

Вышел новый мажорный релиз веб-фреймворка Mojolicious с кодовым именем «Tiger Face», который последовал после первой конференции Mojoconf в Осло. Релиз содержит множество исправлений и изменений, среди которых редизайн страниц для ошибок 404 и 500, защита от CSRF для всех форм, использование опции сокета `SO_REUSEPORT`, для перезапуска приложения без простоя, ротация секретных фраз для постепенной инвалидации подписанных cookie и множество других изменений.

- Net::SSLeay 1.63

В майских релизах низкоуровневой обвязки к openssl `Net::SSLeay` появилась поддержка OSCP, возможность установки протокола TLSv1.1 и TLSv1.2 через переменную `ssl_version`.

- File::ShareDir 1.02

Модуль `File::ShareDir` для определения пути к общим файлам наконец обрёл нового сопровождающего — *Jens Rehsack*, который выпустил два новых релиза с незначительными исправлениями.

- DBD::Pg 3.3.0

Новая версия драйвера `DBI` для СУБД PostgreSQL содержит исправления ошибок в работе с UTF-8.

- perl 5.20.0

Вышел новый мажорный релиз интерпретатора Perl 5.20.0 с внушительным списком изменений и исправлений. Традиционно, с этого момента заканчивается поддержка ветки Perl 5.16.x и всем пользователям рекомендуется обновиться до 5.18.2 или 5.20.0.

- signatures 0.09

Новые релизы модуля сигнатур функций `signatures` включают исправление для работы с perl 5.20.0.

- Kelp 0.9001

Новый релиз веб-фреймворка Kelp исправляет потенциальную проблему безопасности при работе с заголовками `HTTP_X_*`, которая позволяла злоумышленнику обмануть приложение, подменив значения ip-адреса клиента или имя идентифицированного пользователя.

■ *Владимир Леттиев*

5. Интервью с Флорианом Рагвицом (Florian Ragwitz)

Флориан Рагвиц (rafl) — немецкий Perl-программист, автор и соавтор многих модулей на CPAN, разработчик ядра Perl.

Как и когда научился программировать?

Я все еще учусь и не думаю, что закончу в ближайшее время.

Говорят, что первый опыт нельзя забыть, но видимо это неправда. Я начал играться с программирования где-то с десяти лет, но точно не помню, что именно пробудило мой интерес.

Однако, у меня есть воспоминания о неком “детском обучающем компьютере”. Он выглядел как лаптоп, но был всего лишь детской игрушкой. У него был монохромный экран 4x20 символов, и в нем было пара обучающих программ в духе словарных или математических головоломок. Я долго искал в интернете и скорее всего это был “V-Tech Genius Leader 4004 Quadro L” (http://www.pinterest.com/pin/146930006562843060/) и очевидно он продавался только в Германии. В дополнение к разным развивающим играм в нем был интерпретатор BASIC и краткое руководство к этому языку. У меня нет никаких воспоминаний о том, что я программировал на этом устройстве, но я помню, что был очарован BASIC и проводил за ним много времени.

Не совсем приятные воспоминания у меня остались от i486, который моя семья использовала в своем деле некоторое время. Мне нравилось играть в игры, знакомиться с Windows 3.1, DOS и набором разных файловых команд, но мне несколько раз сильно попало, когда я ломал систему и моя семья вынуждена была звонить в сервисный центр, чтобы продолжить свою работу.

Другие воспоминания о программировании у меня со времен средней школы. Мы все использовали графические калькуляторы Casio модели CFX-9850G (http://en.wikipedia.org/wiki/Casio_9850_series). Его можно было программировать с помощью диалекта BASIC, это

было лучшим развлечением во время скучных уроков. Помню я написал несколько программ, которые использовал на уроках математики и физики для решения некоторых задач, но в большинстве случаев я писал игры. В то время я гордился тем, что написал графические шахматы для двух игроков. К сожалению оба игрока должны были играть на одном калькуляторе. До сего дня я виню в этом баг в моей модели калькулятора в реализации команд `Send()` и `Receive()`, что не позволило написать работоспособную версию для режима мультиплеера. Мои попытки написать искусственного противника также не увенчались успехом.

В школе также я познакомился с такими языками как Logo и Pascal, и пытался выучить Visual C++, Visual Basic и Java, как только у меня появился собственный компьютер. Но ни один из них так и не прижился. По-настоящему программировать на постоянной основе для решения реальных задач я начал после того, как перестал пользоваться Windows. Большинство задач в то время были небольшими и я писал их в шелле. Некоторые посложнее писались на awk, PHP и Perl.

Какой редактор используешь?

Оба!

Очень давно, где-то в 2002 году, я перешел с Windows на Linux как основную операционную систему. Частью этого перехода было мое знакомство с Vim. Через некоторое время, после того как я научился правильно выходить из редактора, у меня получилось прочитать документацию. Мне очень понравились его модальный режим и выразительность.

Vim отличный редактор и даже сегодня я регулярно его использую. Хотя, моим главным редактором стал Emacs.

Я хотел перейти с Vim на Emacs довольно продолжительное время. Меня привлекали некоторые его особенности и расширения. Много времени я провел в org-mode (http://orgmode.org) и вероятно не захотел бы переходить на другой редактор, который бы не предоставлял подобного функционала. Возможность использовать разный шрифт в одном буфере очень сильно помогает

при редактировании структурированного текста, как, например, LaTeX-документов или Perl POD — отображение `=head1` заголовков большим размером, чем `=head2`, делает текст для меня более понятным.

В дополнение к этому, у меня есть привычка проводить слишком много времени за тонкой настройкой утилит, которые я часто использую, включая текстовые редакторы. Мне больше нравится писать на Emacs Lisp, чем на Vim Script.

Переход с Vim на Emacs не был простым и потребовал нескольких попыток. Однако все получилось просто, когда я перестал использовать Vim для редактирования Emacs-конфигурации и заставил себя использовать Emacs для всего, включая алиас `alias vi=emacs` в конфиге моего шелла.

Периодически пробую другие редакторы и IDE и поражаюсь, как просто работают в них вещи, которые бы потребовали от меня нескольких часов реализации в Emacs. Я с нетерпением жду момента для перехода на другой редактор, но мне нужно найти такой редактор с такими отличительными особенности, которые бы перевешивали сам мучительный процесс перехода.

Когда и так познакомился с Perl?

Я смутно вспоминаю как заинтересовался Perl. Еще будучи в школе, где-то в 2003 году, я много времени проводил за веб-программированием на PHP. Возможно, я наткнулся на Perl читаю книгу по UNIX, или во время поиска альтернативы PHP, который мне не сильно нравился в то время.

Версия 5.8 была самой новой, когда я начал использовать Perl. Это был удивительный язык, по сравнению с тем, что я до сих пор использовал. Даже лексическая область видимости была достаточно весомой причиной для перехода. У PHP в то время не было ни областей видимости, ни замыканий, ни анонимных функций. У Perl все это было, и даже больше.

Я все еще писал изредка программы на PHP, так как его было легко запускать на сервере, в сравнении с Perl или другими языками. Perl

выделяется среди многих вещей, но простота развертывания до сих пор не является его сильной стороной, хотя и сильно улучшилась с того момента, как я начал им пользоваться. Это меня немного расстраивает.

С какими другими языками интересно работать?

С большинством.

Практически в каждом языке, с которым сталкивался, есть что-то уникальное и интересное. Мне кажется, что знакомясь с каждым новым языком, я становлюсь лучше как программист. Много языков навязывают свою картину мира своим пользователям, но редко, если вообще когда-нибудь, эта картина единственно правильная.

Доступ к разным идеям выраженным на разных языках программирования часто заставляет менять мое мнение об этих идеях. У таких языков как C и Pascal с их статической типизацией есть много практических проблем, но это не делает статическую типизацию плохой идеей. Потоки не плохие, потому что в Perl они плохо реализованы. Парадигмы функционального программирования могут решать многие задачи естественным способом, но также есть и множеств задач, которые более элегантно решаются с помощью объектно-ориентированного подхода.

Есть множество подобных противоречивых концепций в области программирования. Однако, в большинстве случаев они не взаимоисключаемые. Обычно у каждого подхода есть свои ограничения, который стоит учитывать в контексте решения задачи. Вы не будете способны это делать, если в запасе будет только один способ, который поддерживается вашим языком программирования.

Поэтому я стараюсь учить множество языков, когда мне предоставляется возможность. Среди наиболее интересных для меня в последнее время стали Haskell, F#, OCaml, Go и Scala.

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

Исторически у Perl в свое время было много преимуществ над другими языками: его “manipulexity” и “whipuptitude”, поддержка раз-

ных парадигм, его переносимость, отличная поддержка текстовой обработки, его культура тестирования и т.д. С тех пор, однако, многие другие языки его догнали.

Вначале я хотел ответить, что CPAN это самое большое преимущество Perl. CPAN действительно одна из основных причин, почему я продолжаю писать Perl-код, но даже в этой области другие языки шагнули далеко вперед, и, возможно, Perl уже давно не лидер, если брать за сравнение охват предметных областей.

В Perl все еще остается возможность, по моему мнению, удивительной гибкости. Многие вещи, обычно невозможные в других языках, вполне себе реализуемы в Perl, несмотря на то, что иногда приходится постараться. Perl мне бы нравился намного меньше, если бы было невозможно написать такие модули как `List::Gather`, или `Scope::Escape::Sugar`. С другой стороны, подобные модули добавляют функционал уже имеющийся в том или ином виде в других языках. Не знаю, хорошо это или плохо.

Что, по-твоему, является самой важной особенностью языков будущего?

Мне не приходит на ум одна единственная особенность, которую бы хотелось иметь в языке будущего, но я надеюсь, что эти языки позаимствуют возможности таких языков, как Lisp, Haskell и Scala. Мне кажется, что есть множество интересных идей в этих языках, которые еще не попали в массы.

Надеюсь, что мне не придется еще долго программировать на языках без pattern matching (сопоставление с образцом — *пер. с англ.*) и destructuring bind (деструктурирующая привязка — *пер. с англ.*). Я также нахожу интересным перспективу безопасного распараллеливания операций. Также жду, когда эволюционирует статическая типизация, так, что мне не придется часто отлаживать программы из-за моих глупых ошибок, и компилятор будет сразу сообщать мне о них.

Другой областью, на которую я надеюсь, это улучшения в совместимости разных языков программирования. Многие языки разделяют общую среду выполнения, например JVM или CLI. Это большой шаг

вперед к повторному использованию кода между разными языками, но мне кажется есть еще куда развиваться. Я не верю, что возможен один язык, который хорошо решает любой вид задач, но я бы хотел иметь возможность смешивать языки как мне хочется.

Ты пишешь CPAN-модули потому что нет такого функционала, или как доказательство, что такое можно реализовать на Perl?

Большинство написанных мною модулей или правок, внесенных в модули других людей, были мотивированы реальными задачами. Почти все время, которое я тратил на правку существующих модулей, было попытками чинить баги, на которые я наталкивался, или же сделать их более общими для решения определенных задач. И очень редко, как, например, в случае `List::Gather`, мне приходится писать новые модули, потому что существующие не удовлетворяют моим требованиям.

Большинство моих новых модулей реализуют функционал, которого вообще не было на CPAN. Многие из них это просто биндинги к существующим C-библиотекам, которые я хотел использовать.

Единственным модулем, который я написал как доказательство, что такое можно сделать в Perl, был мой эксперимент с `signatures.pm`.

Как получилось так, что ты начал выпускать perl-дистрибутивы?

Я не помню как конкретно это произошло, но свой первый Perl-релиз я выпустил в 2010 году, когда Jesse Vincent был в роли пампкина (pumpkin). В IRC он то ли попросил меня сделать релиз, то ли я сам напросился в добровольцы. В то время процедура релиза, которую заложил Jesse, была довольно хорошая и выпуск новой версии perl оказался на удивление простым.

Моими любимыми релизами остаются 5.15.4, который я выпустил сидя на диване Ларри Уолла, 5.17.5, который я выпустил во время доклада на YAPC::Brasil, и 5.14.2, единственную выпущенную мною версию не для разработчиков.

Как можно начать вносить свой вклад в развития ядра perl?

Я не могу порекомендовать как это сделать, но в моем случае я просто утолял свой собственный зуд.

Оказывается мой первый комит в ядро был в 2008 году. Тогда умное сравнение given/when было новым и интересным для меня, и, очевидно, B::Deparse, с которым я много работал, не очень хорошо поддерживал эту новую структуру. Мне это сильно досаждало и я поправил это. Это было очень просто — изменение всего лишь одной строчки кода в модуле B::Deparse.

Многие из моих ранних комитов следовали похожему шаблону. В какой-то момент я обнаружил, что нельзя просто совмещать флаги -E- и -p/-n в однострочниках perl -n'} END { ...'. Небольшой трехстрочный патч и все заработало. Где бы я ни находил опечатку или неточность в документации, я всегда отправлял патч. Это приносило внутреннее удовлетворение.

В общем, нет ничего магического в ядре perl (хотя, возможно, исключением является sv_magicext() и подобные функции). Большая часть ядра написана на самом Perl — обычном Perl с которым вы сталкиваетесь каждый день. Нет причин его бояться. Мне кажется, что каждый Perl-программист сможет поучаствовать в развитии своего языка программирования тем или иным образом, если захочет этого.

Достаточно ли у perl волонтеров для развития кодовой базы? В чем он сейчас нуждается?

На данный момент я не переживаю, что у Perl не хватит волонтеров для поддержания и развития языка, хотя, конечно, всегда есть место для улучшения.

Мне неизвестно в чем в данный момент нуждается Perl. Если кто-то хочет поучаствовать, стоит посмотреть на документацию perlhack и Porting/todo.pod в дистрибутиве, там находится множество отправных точек. Уверен, что сообщество разработчиков perl5 с удовольствием пообщается с вами о том, как можно помочь, если связаться с ними через список рассылки, IRC-канал #p5p или любым другим способом.

Лично я надеюсь увидеть альтернативные реализации Perl, которые можно запускать на различных окружениях, как, например, JVM, CLI или даже LLVM. Такие проекты как `Compiler::Lexer`, `Compiler::Parser` и `gperl` двигаются в правильном направлении, но, как по мне, есть еще куда расти.

Что делаешь для Debian?

Уже не очень многое. В свое время я поддерживал относительно большое количество пакетов. Большинство из них были Perl-ориентированными, но я также собирал и несколько не Perl вещей, как, например, музыкальный плеер XMMS2 и несколько клиентов к нему. Сейчас большинство моих пакетов уже нашли новых сопровождающих, которые отлично справляются с поддержанием их в обновленном и рабочем состоянии.

Где сейчас работаешь? Сколько времени проводишь за написанием Perl-кода?

Я работаю в небольшой консалтинговой IT-компании Infinity Interactive (http://iinteractive.com). Там я работаю со всем, что подходит для решения проблем наших партнеров. Иногда это Perl, иногда это что-то другое. Мне нравится техническое разнообразие в моей работе.

Последнее время я в основном работаю над задачами системной автоматизации. Приложения, написанные на Perl, занимают примерно 10% или 20% моего рабочего времени.

Стоит ли советовать молодым программистам учить сейчас Perl?

Я не думаю, что Perl это один из лучших языков для преподавания или изучения программирования. Я бы советовал другие языки для новичков в этой сфере.

Несмотря на это, я думаю, что есть множество хороших уроков, которые можно извлечь из изучения Perl, которые пригодятся в карьере программиста, будет ли он использовать Perl для решения своих задач или нет. В этом смысле рекомендовать Perl для начинающих

или опытных разработчиков, как правило, полезно.

Вопросы от читателей

Как ты понимаешь, что твой доклад слишком сложен для среднего разработчика?

К несчастью, я обычно этого не замечаю.

Откуда пошло `Acme::rafl::Everywhere`?

Это случилось во время YAPC::NA 2012 в Мэдисоне, Висконсине. В первый же день я присоединился к группе перловиков, организованной Дейвом Рольски для проведения (анти-) ужина по поводу приезда. Среди нас были Роберт Блэквелл и Sawyer X.

Я не сильно помню как разговор перешел на меня, но в какой-то момент Роберт заметил, что я принимаю участие во многих сообществах и/или проектах. Подозреваю, что он ссылался на большое количество разнообразных модулей на CPAN, который я поддерживал в то время, а также на то, что я участвовал во многих не Perl-проектах как Git, Linux, XMMS2 и Arduino и, возможно, также на тот факт, что я был на всех пяти YAPC и на многих международных Perl-воркшопах в том году.

В какой-то момент кто-то произнес “you are everywhere!” (“да ты везде!” — пер. с англ.) и Sawyer’а понесло. Большинство шуток из `Acme::rafl::Everywhere` были придуманы на том ужине. Позже, когда Sawyer после конференции выложил модуль, несколько других людей отправили pull-запросы на добавление других дополнительных фактов.

Есть даже команда “!rafl” на DuckDuckGo.com.

У меня двойственное отношение ко всему этому.

Часто выходишь на улицу?

Если небольшой загар на моем лбу и шее может быть индикатором — то слишком часто.

Есть множество интересных вещей, которые происходят в Бруклине и вообще в Нью-Йорке, и, к сожалению, большинство из них не происходят у меня дома. Чтобы посмотреть на все это, мне приходится заставлять себя выходить из дому. Эта та жертва, которую я готов принести. Также выбор пива на моей домашней полке сильно ограничен, поэтому часто приходится работать из разных баров и пивных на открытом воздухе на районе.

■ *Вячеслав Тихановский*