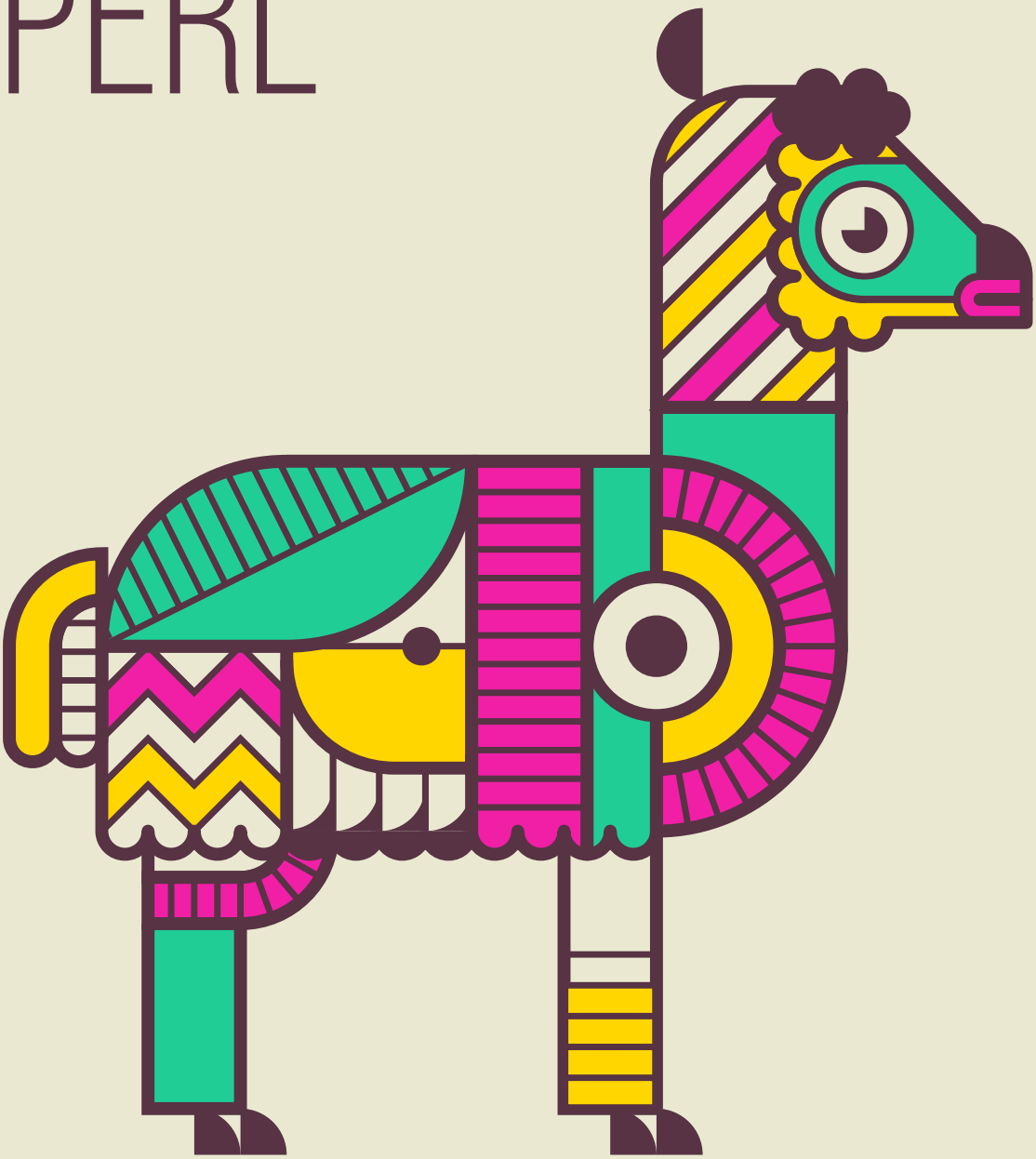
PRAGMATIC PERL

31



09/2015

pragmaticperl.com

Pragmatic Perl 31

pragmaticperl.com

Выпуск 31. Сентябрь 2015

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

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

Авторы статей: Андрей Шитов

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2015-09-21 19:25

Оглавление

1 От редактора

Читайте в этом номере отчет о конференции YAPC::Europe 2015. Если вы тоже были на конференции, присылайте свои отчеты в следующий выпуск!

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

Приятного чтения.

■ *Вячеслав Тихановский*

2 YAPC::Europe 2015

Рассказ о конференции

В этом году европейская конференция YAPC::Europe прошла в начале сентября в испанском городе Гранада. Мероприятие проводилось в одном из комплексов знаний университета, на «Факультете наук» (sic). Тема конференции: «Art + Engeneering».

Накануне

Традиционно накануне конференции проводится неформальная встреча (pre-conference meeting). В университетском дворике поставили небольшую печку для пиццы и кран с пивом.

В этом году расписание составлено так, что конференционный день начинался довольно рано по программистским меркам — в 9 или даже в 08:30.

Организаторы конференции договорились с десятком кафе около место проведения и выдали талоны на питание. Несмотря на крохотные порции тапасов, можно было использовать три талона за раз и не потратить ни копейки на еду за всю конференцию :-)

1-й день

На открытии конференции Антон Березин объявил о решении комитета YEF (YAPC Europe Foundation) о месте проведения YAPC::Eureop в 2016 году. Она пройдет в

румынском городе Клуж-Напока. Организаторы — группа Cluj.pm и компания Evozone — несколько лет назад начали активную работу по организации перл-мероприятий и с третьей попытки выиграли право проведения международного мероприятия.

Далее начался основной контент — доклады про перл и только. К сожалению, видеозапись велась только в главном зале, да и то выборочно, так что пересмотреть многие записи невозможно. Я выбирал для себя доклады, преимущественно связанные с Perl 6 или с Perl-сообществом.

Леон Тиммерманс. How to contribute to perl5 core

Леон кратенько рассказал о том, как устроена разработка Perl 5, о том, что есть списки рассылки, есть IRC, выпускающий релиз-менеджер, документация, которая нуждается в обновлении, гит-репозиторий с историей всех версий разработки, начиная с Perl 1, про существование Фонда поддержки ядра Perl 5, и про то, что слушать мнения пользователей не всегда обязательно (по этому поводу Рейни Урбан заметил в Твиттере: «This is really the perl community as we know it. A committer publicly calling a user asshole. Nothing changed»).

Доклад переродился в разговоры между слушателями в зале, эдакий междусобойчик среди p5p.

Бен Тайлер. Distributed Systems 102: CRDTs for Poets and Perl Hackers

Бен рассказывал о пробемах, возникающих при репликации распределенных баз данных, в которых обновления могут происходить в непредсказуемое время и в непредсказуемом месте, и как с этим бороться или жить, и в каких случаях даже на больших объемах данных не всегда важно иметь идеально верную копию данных (например, в счетчике лакйов на Ютубе).

Видимо из-за недостатка времени, доклад содержал лишь общие рассуждения о том, как работать с синхронизацией и реплицированием, и не содержал примеров из практики применения этого в жизни.

Рейни Урбан. Using the new compiler in production

Рейни в течение нескольких лет работает над новым бекендом для компилятора Perl. Его разработка используется в продакшне в продуктах компании cPanel. Основной посыл — новый компилятор позволяет сократить расходы на память и часто ускорить время компиляции программ. Более подробную информацию следует ожидать в ближайшее время на сайте perl11.org.

Блиц-доклады

Традиционно день завершается сессией пятиминутных блиц-докладов, модерируемых R Geoffrey Avery.

Среди блицев первого дня мне показался самым интересным выступление Дейва Кросса о том, что в мире еще существует много PERL-программистов (именно так, большими буквами). Это такие разработчики, которые используют Perl в духе скриптов из архива Матта, и это та категория, которая теоретически может перейти на сторону добра и начать использовать возможности современного перла, отказавшись от стиля прошлого века.

Кейноут Куртиса «Овида» Поу Turning Points

В этом году организаторы поставили в расписание конференции аж пять кейноутов, примерно по два в сутки: в начале и завершении дня.

В своем выступлении Куртис перечислил основные моменты, которые повлияли на то, чем Perl был раньше и чем стал сейчас, и что может ожиадать в будущем: O'Reilly, браузер Mosaic, HTML/CGI, Matt's Script Archive, анонс Perl 6 в июне 2000 года. Одновные достижения помимо этого: большое сообщество пользователей, обратная совместимость и CPAN Testers.

Система автоматического тестирования — лучшая из имеющихся среди всех языков, поэтому есть смысл не начинать стартапы на других языках программирования, многие из которых являются временно модными («JavaScript framework of the week»), а есть смысл выбирать Perl как технологию, которая проверена временем и стабильна.

Следующей важной поворотной точкой

может стать релиз Perl 6 в этом году. Но при этом неплохо бы подновить perl.com, где последняя статья была опубликована пару лет назад, сделать что-то с Perlmonks, который выглядит сайтом из далекого прошлого, а use.perl.org вообще закрыт.

И, наконец, есть смысл переименовать конференцию YAPC (Yet Another Perl Conference) обратно в The Perl Conference, потому что другой все равно нет.

2-й день

Coep Экс. State of the [Art] Velociraptor

Очередной открывающий день кейноут, который обычно давал Матт С. Траут. Coep упомянул, что список p5p полон негатива,

и надо по жизни уважать друг друга, тем более тех, кто работает над ядром Perl 5.

Далее он рассказал, что CPAN Pull Request Challenge оказался неожиданно успешным: вместо 10-20 человек, о которых мечтал Нил Бауэрс, подписалось 365.

Очень успешной была конференция YAPC::Asia. В прошлом году она собрала 1300 участников, в нынешнем более 2100, причем почти все из Японии (об этой конференции был отдельный рассказ от ее организатора).

Далее Соер напомнил, что на 2015 год выпало несколько юбилейных дат: 20 лет спану, 10 лет Perl::Critic, какая-то дата Perl::Lint, а еще был создан Фонд поддержки ядра Perl 5.

Последние минуты выступления были заполнены слащавыми словами о том, что мы семья и надо людить друг друга.

Джессика Роуз. What the Cult of Expertise Costs

Джессика устроила интерактивный рассказ, с привлечением аудитории о том, кого считать экспертами.

— Кто такой эксперт?

— Тот, кто сделал что-то более одного раза (Глория Уолл).

— Тот, кто может сделать что-то сам без посторонней помощи (Герберт Бреунинг).

— Тот, кто потратил на изучение не менее

10.000 часов (кто-то из зала).

Далее тема перешла на то, стоит ли доверять экспертам, которые называют экспертами сами себя. Среди посетителей нашлось много таких, кто готов был поспорить с докладчицей и с другими участниками (Илья, Коно, добавляйте свои комменты к статье!).

Мне эта дискуссия очень понравилось, хотя позже я слышал комментарии о том, как неискренне была докладчица.

Daisuke Maki (lestrrat). YAPC::Asia Tokyo Behind The Scenes, Or How We Organize A Conference For 2000 Attendees

Рассказ главного организатора конференций YAPC::Asia последних лет. Недавно он

писал у себя в блоге, что он прекращает заниматься организацией конференции в 2015 году.

Экспресс-статистика: 2130 участников, 90 докладов и блиц-докладов, 60 спонсоров, 400 записей в блогах, постоянный поток 100 Мб/с в сети во время конференции, 3500 устройств, подключенных к Wi-Fi, бюджет 26 миллионов йен (175.000 евро); 90 организаторов, в том числе 60 волонтеров и 22(!) человека, обеспечивающих работу Wi-Fi.

В этом году организаторы пользовались электронной почтой только для общения со спонсорами, а между собой общались в Слаке.

Спонсорам предлагали «спонсорское меню», где можно было набрать спонсорский

пакет из доступных опций, вместо традиционного деления на уровни спонсорства (золотые и пр.).

Junichi Ishida (uzulla). Great Tools Heavily Used In Japan, You Don't Know.

Небольшой доклад о том, что крутого делают бешеные японцы(TM) в виде перечня людей и модулей, которые они опубликовали.

Максим Вуец. Warm and fuzzy text matching

Рассказ о том, какие методы нечеткого поиска можно использовать при работе с текстом. Исходная задача Максима была

смерджить несколько списоков музыкальных записей, по максимуму используя компьютер для поиска того же названия, но записанного несколько иначе.

Panel Discussion: Growing the Perl community

Дискуссия о том, как вырастить Perl-сообщество. Эндрю Соломон организовал этот круглый стол, в котором приняло участие девять человек, из которых трое работают в booking.com, а один десять лет как не занимается перлом, а использует руби.

Модератор попросил участников рассказать, что они подразумевают под ростом комьюнити. Ответы были довольно вялыми и невнятными.

Я задал вопрос о том, о каком, собстсвенно, сообществе идет речь: Perl 5 или Perl 6. Ответы примерно поровну разделились на «оба», «Perl 5» и «да хз».

3-й день

Третий конференционный день начался аж в 08:30, причем в отличие от предыдущих дней доклады с самого утра были организованы в четыре потока. В расписании этого дня было собрано много выступлений про Perl 6.

Штефан Сайферт. Building bridges - bringing Perl 5 and Perl 6 back together

Интересный обзорный рассказ о том, как использовать Perl 5 в программах на Perl 6 и наоборот (или Perl 5 внутри Perl 6 внутри Perl 5).

Салве Нельсон. Perl Community Backchannels Session

Салве — движущая сила норвежских Perl-монгеров. В своем выступлении он перечислил некоторые неочевидные места, где можно общаться и сообщать миру о перле (например, Hackersplaces) и перечислил людей из Perl-сообщества, которые могут помочь с разными вопросами или просто занимаются интересными вещами.

Габор Сабо. Web development using Perl 6

Доклад о том, какие есть фреймворки, какие есть шаблонизаторы, модули для доступа к базам данных, показал простое приложение «Hello, World!» в браузере. Аплодисменты.

Алекс Мунтада. Writing tutorials for Perltuts.com

Рассказ о том, как устроен и выглядит сайт perltuts.com, каким образом сделаны страницы уроков на основе POD-файлов, как вставить в текст работающий пример или как нарисовать график с помощью jQuery и как вставить в урок сопровождающий тест.

Джон Халтинвангер. Whatever, or How I Stopped Worrying and Fell in Love with Perl 6 Operators

20-минутный рассказ об операторе Perl 6, который называется whatever (*). Наш журнал еще не писал об этом.

Тео ƒан Хуссел. qué les hablas mi lenguaje

Доклад появился в расписании в последнюю минуту, потому что накануне Джеф Гоф сломал ногу и не смог появиться на конференции в этот день.

Тео работает над переделкой Акта (фреймворка для сайтов конференций), под эту работу был выделен некий грант. Рассказ был о том, что существует HTTP-заголовок

`Accept-Language`, который можно использовать, чтобы отдавать сайт на разных языках :-o.

Элизабет Маттайсен. Parallelism, Concurrency, and Asynchrony in Perl 6

Лиз прочитала доклад, подготовленный Джонатаном Вортингтоном, который в этот раз не смог приехать (он переехал в Прагу, женился, вот все вот это).

Слайды как всегда очень подробные: их можно просто брать и читать как статью. Тема доклада касалась ключевых слов Perl 6 `start`, `await`, `race`, `hyper`, `supply`, `whenever`, `emit`, `react`, классах `Proc::Async` и `OO::Monitor` (он предлагает слова `monitor` и `actor`, аналогичные `class`) и встроенных функций типа `map` и `grep`

, которые могут работать параллельно без особых дополнительных усилий со стороны разработчика.

Ларри Уолл. Get Ready To Party!

Ларри повторил свой доклад, который он сделал в феврале этого года на конференции FOSDEM в Брюсселе. Наш журнал уже писал об этом. На YAPC::Europe велась видеозапись, так что этот доклад можно посмотреть в интернете.

Блиц-доклады

Третий день закончился третьей серией блиц-докладов. Антон Березин сообщил о возможных изменениях в деятельно-

сти YAPC Europe Foundation. Я рассказал про сайт с обложками книг по перлу и попросил японцев прислать мне парочку картинок. Через несколько минут lestrrat ответил со сцены скриншотом с японской книги на Амазоне и сделал смешной блиц на японском языке. Видео этого доклада есть на Ютубе.

■ *Андрей Шитов*

3 Прецизионные бенчмарки Perl

Какой Perl самый быстрый? Имеет ли смысл переходить на cperl или stableperl? Возможно ли провести точное сравнение производительности разных perl?

Бенчмарки по-настоящему сложны

Если вы смотрели доклад Питера Рэббитсона «Benchmarking is *REALLY* hard», то помните, что выполнение бенчмарков — зто нетривиальная задача. Основа большинства бенчмарков — это измерение времени, за которое выполняется тестируемый фрагмент кода. Данная метрика зависит от множества факторов, которые

вносят большую погрешность в измеряемую величину:

- Динамическая тактовая частота процессора: современные процессоры могут «саморазгоняться» или, наоборот, снижать тактовую частоту в случае перегрева или включать режим экономии энергии. Кроме того, может наблюдаться нестабильность тактовой частоты при использовании дешёвых комплектующих.
- Многоуровневая система кешей в процессоре: в зависимости от числа «промахов» время выполнение одного и того же кода может существенно изменяться от запуска к запуску.
- Конкуренция за процессорное время: операционная система может

выделять тестируемому процессу различное число тиков процессорного времени в зависимости от текущей загрузки и наличия других фоновых процессов.

Таким образом, во всех подобных бенчмарках возникает случайная погрешность, которую пытаются устранить проведением серии из большого числа измерений и вычислением средних значений. Всё же погрешность может оказаться существенной, и вычисленное ускорение на 5% может на самом деле оказаться замедлением на 5% и наоборот.

Сферический код в вакууме

Можно ли устранить все побочные эффекты и получить чистые измерения скорости работы кода? Для ответа на этот вопрос попробуем обратиться к архитектуре современных процессоров. Компьютеры на архитектуре фон Неймана используют память для хранения инструкций и данных и процессор для выполнения операций. В процессор загружается очередная инструкция из памяти, и происходит её выполнение. В современных системах частота работы процессора и памяти существенно различаются, к примеру, доступ в основную память может занимать до 200 циклов процессора, поэтому работа с памятью становится «бутылочным горлышком» в производительности системы.

Чтобы ускорить работу с памятью, была

создана система кешей — быстрой памяти, расположенной внутри процессора, куда предварительно загружаются фрагменты основной памяти, что ускоряет доступ. Может использоваться два и больше уровней кеша, которые различаются по объёму памяти и скорости работы. Например, кеш первого уровня L1 может составлять 64 КБ и требовать для загрузки порядка 10 тактов. Таким образом, код, использующий 100 элементов данных, которые окажутся в кеше, будет загружен на 95% быстрее, чем в случае загрузки их из памяти (так называемый «кеш-промах»). Как правило, кеш раздельно хранит инструкции и данные, поскольку промах при загрузке инструкции выливается в простой процессора, в то время как промах в данных позволяет выполнять следующие инструкции, поэтому они не должны конкурировать между собой за место в кеше.

Конечно, 64 КБ это слишком мало по сравнению с текущими объёмами оперативной памяти, поэтому далее может идти кеш второго уровня большего объёма, но меньшей скорости доступа, и так далее. Практическое значение имеет лишь объём последнего уровня кеша, поскольку разрыв в скорости доступа между ним и обычной памятью самый большой. На современных процессорах объём этого кеша может составлять 8 МБ и больше.

Таким образом, скорость выполнения кода напрямую зависит от количества загружаемых инструкций и данных, а также количества промахов в кешах. Также имеет большое значение правильное предсказание условных переходов, т.е. чтобы последующие инструкции заранее оказались в кеше.

Cachegrind

На сегодняшний день существует инструмент, который позволяет измерить описанные выше характеристики — профайлер кеша cachegrind утилиты valgrind.

В отличие от специализированных профайлеров, например, perf для Linux, cachegrind использует симуляцию процессора, системы кешей и предсказания условных и косвенных переходов. С одной стороны, полученные результаты могут отличаться от данных на реальных процессорах, но, с другой стороны, это работает на множестве различных архитектур и операционных систем, не зависит от текущей загрузки системы, даёт высокую воспроизводимость результатов, позволяя проводить точные сравнения производительности.

Рассмотрим пример, для программы на perl, которая выводит слово hi на экран:

```
$ valgrind --tool=cachegrind --branch-sim=yes \
    --cachegrind-out-file=/dev/null \
    perl -e 'print "hi\n"'

==19617== Cachegrind, a cache and branch-prediction profiler
==19617== Copyright (C) 2002-2013, and GNU GPL'd, by Nicholas Nethercote et al.
==19617== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==19617== Command: perl -e print\ "hi\\n"
==19617==
hi
==19617==
==19617== I   refs:
```

```
    1,129,633
==19617== I1  misses:
    4,618
==19617== LLi misses:
    3,168
==19617== I1  miss rate:
    0.40%
==19617== LLi miss rate:
    0.28%
==19617==
==19617== D   refs:
    407,661  (264,407 rd   +
    143,254 wr)
==19617== D1  misses:
    11,694  (  7,945 rd   +
    3,749 wr)
==19617== LLd misses:
    7,612  (  4,354 rd   +   3,258
     wr)
==19617== D1  miss rate:
    2.8% (    3.0%     +     2.6%
     )
==19617== LLd miss rate:
```

```
    1.8% (    1.6%     +      2.2%
     )
==19617==
==19617== LL refs:
    16,312  ( 12,563 rd   +
    3,749 wr)
==19617== LL misses:
    10,780  (  7,522 rd   +
    3,258 wr)
==19617== LL miss rate:
    0.7% (    0.5%     +      2.2%
     )
==19617==
==19617== Branches:
    200,099  (194,318 cond +
    5,781 ind)
==19617== Mispredicts:
    17,627  ( 16,447 cond +
    1,180 ind)
==19617== Mispred rate:
    8.8% (    8.4%     +     20.4%
      )
```

О чём говорит полученный вывод:

- I refs — количество загруженных инструкций;
- I1 misses — число промахов в кеше первого уровня при загрузке инструкций;
- LLi misses — число промахов в кеше последнего уровня при загрузке инструкций;
- I1/LLi miss rate — соответствующая доля промахов в кешах в процентах;
- D refs — количество операций с данными (суммарно для чтения и записи);
- D1 misses — число промахов в кеше первого уровня при операциях с дан-

ными (суммарно для чтения и записи);

- LLd misses — число промахов в кеше последнего уровня при операциях с данными (суммарно для чтения и записи);
- D1/LLd miss rate — соответствующая доля промахов в кешах в процентах;
- LL refs — сумма промахов в кеше первого уровня I1 + D1;
- LL misses — сумма промахов в кеше последнего уровня LLi + LLd;
- LL miss rate – доля промахов в кешах для данных и инструкций в процентах;
- Branches — общее число переходов (условных и косвенных);

- Mispredicts — число ошибок предсказания перехода;
- Mispred rate — доля ошибок предсказаний перехода в процентах.

Если несколько раз повторить запуск команды, то можно увидеть, что каждый раз будет выводиться примерно одни и те же цифры с точностью до пятой(!) значащей цифры. Это означает, что можно выявить даже небольшие изменения в производительности. Например, добавив операцию конкатенации в программе: `perl -e 'print "hi"."\n"'`, можно увидеть, что это будет стоить нам загрузки дополнительно порядка 6000 инструкций и 3000 данных.

Сравнение производительности разных версий Perl

В дереве исходников Perl есть утилита `Porting/bench.pl`, которая под капотом использует cachegrind и может проводить сравнение производительности для различных версий Perl. Она использует только базовые модули и работает на perl ≥ 5.10, поэтому её можно свободно загрузить и использовать отдельно от исходного кода Perl.

В оригинальном анонсе утилиты bench.pl показывалось сравнение пяти последних на тот момент стабильных версий perl и разрабатываемой версии perl 5.21.6:

```
$ Porting/bench.pl -j 8 \
        perl5125o perl5144o
           perl5163o perl5182o
```

```
                perl5201o perl5216o
...
    expr::assign::scalar_lex
    lexical $x = 1

                perl5125o perl5144o
                   perl5163o perl5182o
                    perl5201o
                   perl5216o
               __________ __________
                  __________ __________
                   __________
                  __________
          Ir     100.00    107.05
                 101.83    106.37
                 107.74    103.73
          Dr     100.00    103.64
                 100.00    105.56
                 105.56    100.00
          Dw     100.00    100.00
                  96.77    100.00
                 100.00     96.77
        COND     100.00    120.83
```

```
            116.00    126.09
          126.09    120.83
      IND    100.00     80.00
             80.00     80.00
             80.00     80.00

 COND_m    100.00    100.00
        100.00    100.00
     100.00    100.00
  IND_m    100.00    100.00
         100.00    100.00
      100.00    100.00

  Ir_m1    100.00    100.00
         100.00    100.00
      100.00    100.00
  Dr_m1    100.00    100.00
         100.00    100.00
      100.00    100.00
  Dw_m1    100.00    100.00
         100.00    100.00
      100.00    100.00
```

```
22          Ir_mm    100.00    100.00
                100.00    100.00
              100.00    100.00
23          Dr_mm    100.00    100.00
                100.00    100.00
              100.00    100.00
24          Dw_mm    100.00    100.00
                100.00    100.00
              100.00    100.00
```

В данном примере сравнивалось, как менялась производительность простейшей операции присвоения значения лексической переменной `$x = 1`. Все указанные цифры — относительные, от первого значения, т.е. в сравнении с perl 5.12.5. Чем больше значение, тем лучше (меньше реальных операций). Наименование строк в первой колонке соответствуют данным cachegrind: *Ir* — чтение инструкций, *Dr* — чтение данных, *Dw* — запись данных,

COND и *IND* — количество условных и косвенных переходов, суффикс *m* означает соответствующие «промахи» в кешах.

Здесь была замечена регрессия в производительности между perl 5.20.1 и perl 5.21.6 по показателю Ir: 107.74 против 103.73. С помощью `git bisect` удалось найти, что это связано с коммитом, в котором началась использоваться опция компилятора, закручивающая гайки в безопасности, `-fstack-protector-strong`, которая и привела к замедлению работы perl.

Как работает утилита bench.pl?

Утилита использует файл t/perf/benchmarks, в котором перечислены множество небольших тестов в следующем формате:

```
'expr::assign::scalar_lex' => {
    desc    => 'lexical $x = 1',
    setup   => 'my $x',
    code    => '$x = 1',
},
```

Поле *desc* описывает суть теста, *setup* — некоторый инициализирующий код, затем *code* — собственно код, который будет тестироваться.

Для каждого подобного теста утилита создаёт тестовый скрипт вида:

```
package $test;
BEGIN { srand(0) }
$setup;
for my \$__loop__ (1..\$ARGV[0])
    {
    $code;
}
```

Где `$setup` — это код инициализации, `$code` — тестируемый код. Утилита делает запуски со значением итераций цикла 10 (short) и 20 (long), с пустым тестируемым кодом и актуальным тестируемым кодом (всего четыре запуска valgrind). Таким образом, можно рассчитать и убрать оверхед, вносимый кодом инициализации, и сравнивать только тестируемый код.

Для чего можно применять bench.pl обычным пользователям?

Утилита bench.pl может быть полезна обычным пользователям, например, чтобы тестировать своё приложение для работы на новых версиях Perl. Критические к производительности участки кода могут быть вынесены в виде отдельного теста и прогоняться на blead или других версиях

Perl, чтобы заранее обнаружить регрессии производительности и начать бить тревогу.

Утилиту bench.pl также можно использовать и для привычных бенчмарков, когда сравнивается не perl, а код программы. Например, если есть матрица 101x101, то что быстрее: записывать в колонку или ряд? Для этого можно создать файл с тестами `my.bench`:

```
[
    'test1' => {
        'desc'  => 'row',
        'setup' => 'my @x; $x
          [100][100] = 1',
        'code' => '$x[100][$_]=1
          for 0..100'
    },
    'test2' => {
        'desc'  => 'col',
        'setup' => 'my @x; $x
          [100][100] = 1',
```

```
        'code' => '$x[$_][100]=1
           for 0..100 '
    }
]
```

Далее запустим с параметром `--raw`, который покажет не относительные результаты, а фактические:

```
$ bench.pl -j 8 --benchfile=my.bench \
    --fields=Ir,Dr,Dw,COND,IND,COND_m,IND_m \
    --raw perl

test1
row

          perl
      ----------
   Ir  77847.0
   Dr  27711.0
   Dw  12982.0
```

```
  COND  11314.0
   IND   1435.0
COND_m    108.0
 IND_m   1128.0

test2
col

            perl
          ________
    Ir  77847.0
    Dr  27711.0
    Dw  12982.0
  COND  11314.0
   IND   1435.0
COND_m    107.0
 IND_m   1128.0
```

Ответ: абсолютно всё равно.

Сравнение производительности perl 5.22.0, stableperl 1.001, cperl 5.22.1

После выпуска Perl 5.22.0 произошёл раскол в Perl-сообществе, проявившийся в виде серии форков: сначала stableperl от Марка Леманна, затем cperl от Рейни Урбана. Их объединяет несогласие с официальным направлением развития Perl, когда возникают нарушения обратной совместимости или принимаются спорные решения в реализации тех или иных новшеств без всестороннего обсуждения.

Положительный момент в данной истории, это возможность сторонним наблюдателям оценить для себя, насколько им по душе тот или иной путь развития.

Попробуем сравнить производительность этих трёх реализаций: perl 5.22.0, cperl 5.22.1 и stableperl-1.001.

Можно воспользоваться perlbrew для установки подопытных:

```
# perl 5.22.0
$ perlbrew install perl-5.22.0

# stableperl 1.001
$ perlbrew install --as=
    stableperl-1.001 \
    http://stableperl.schmorp.de/
        dist/stableperl
        -5.22.0-1.001.tar.gz

# cperl 5.22.1
# хак: имя каталога внутри архива
    не соответствует имени архива
$ wget https://github.com/perl11/
    cperl/archive/cperl-5.22.1.tar
    .gz \
```

```
    -O cperl-cperl-5.22.1.tar.gz
$ perlbrew install --as=cperl
    -5.22.1 $(pwd)/cperl-cperl
    -5.22.1.tar.gz
```

Загрузим скрипт bench.pl:

```
$ wget http://perl5.git.perl.org/
    perl.git/blob_plain/blead:/
    Porting/bench.pl
```

Загрузим файл бенчмарков:

```
$ wget http://perl5.git.perl.org/
    perl.git/blob_plain/blead:/t/
    perf/benchmarks
```

Запустим бенчмарк:

```
$ bench.pl -j 8 --benchfile=
    benchmarks \
    $PERLBREW_ROOT/perls/perl
        -5.22.0/bin/perl5.22.0=
```

```
        perl5220 \
3       $PERLBREW_ROOT/perls/
        stableperl-1.001/bin/perl5
        .22.0=stable5220 \
4       $PERLBREW_ROOT/perls/cperl
        -5.22.1/bin/cperl5.22.1=
        cperl5221
```

В результате получим простыню с результатами тестов по всем тестируемым микрооперациям, последним будет идти среднее значение производительности по всем тестам:

```
1 AVERAGE
2
3           perl5220 stable5220
              cperl5221
4           ---------- ----------
              ----------
5       Ir    100.00     101.37
        101.19
6       Dr    100.00     100.39
```

```
          100.60
     Dw   100.00      100.00
          98.77
   COND   100.00      100.00
          99.76
    IND   100.00      100.00
          100.00

COND_m    100.00       96.17
     62.23
 IND_m    100.00      100.00
          100.00

 Ir_m1    100.00      100.55
          100.00
 Dr_m1    100.00      100.00
          100.00
 Dw_m1    100.00      100.00
          100.00

 Ir_mm    100.00      100.00
          100.00
 Dr_mm    100.00      100.00
```

```
      100.00
Dw_mm   100.00      100.00
      100.00
```

В среднем показатели близки (кроме завала на промахах с условными переходами в cperl), но, возможно они имеет такой же смысл, как и средняя температура по больнице. Поэтому лучше посмотреть те тесты, в которых есть существенные различия.

Например, практически во всех тестах, где используются операции с хешами, заметно проигрывает Perl 5.22:

```
expr::hash::ref_lex_2var
lexical $hashref->{$k1}{$k2}

          perl stableperl  cperl
        ------- ---------- -------
    Ir 100.00      120.42 119.16
    Dr 100.00      105.10 106.45
```

```
    Dw 100.00      100.00  93.94
  COND 100.00      100.00 100.00
   IND 100.00      100.00 100.00

COND_m 100.00      100.00 111.11
 IND_m 100.00      100.00 100.00
```

Это объясняется тем, что как в cperl, так и stableperl по умолчанию используется хеш-функция **FNV-1A**, которая существенно быстрее медленной, но более безопасной дефолтовой хеш-функции Perl.

В остальных тестах сохраняется практически полный паритет.

Выводы

Безусловно, cachegrind даёт большие возможности для проведения замеров производительности. К сожалению, данные не всегда удобно сравнивать, поскольку нет абсолютных значений затраченных циклов процессора. В любом случае, радует, что при разработке Perl начали проводить тестирование на регресс производительности.

■ *Владимир Леттиев*

4 Обзор CPAN за август 2015 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц

Статистика

- Новых дистрибутивов — 155
- Новых выпусков — 804

Новые модули

HTTP::PublicKeyPins

Модуль `HTTP::PublicKeyPins` позволяет генерировать специальные HTTP-

заголовки для привязки публичного ключа сервера в соответствии с RFC 7469. Это позволяет веб-браузерам запоминать на некоторое время сертификат веб-ресурса и противостоять возможным MITM-атакам (например, в случае компрометации центра сертификации).

C::Blocks

С помощью `C::Blocks` становится возможным делать вставки C-кода внутри Perl-программ. В отличии от схожих по функционалу модулей, как например, `Inline::C`, в `C::Blocks` возможно использовать также Perl-переменные внутри C-кода:

```
# расчёт суммы целых
    положительных чисел от 1 до
    100
```

```
my $N = 100;
my $result;
cblock {
    int i;
    int result = 0;
    int N = SvIV($N); /* скаляр
       $N */
    for (i = 1; i < N; i++)
       result += i;
    sv_setiv($result, result); /*
        скаляр $result */
}
print "Сумма чисел от 1 до $N =
   $result\n";
```

Crypt::HSXKPasswd

Модуль содержит утилиту hsxkpasswd, которая позволяет создавать надёжные пароли, устойчивые к атакам грубым

перебором и по словарю, но при этом их достаточно просто запомнить. Идея модуля возникла по мотивам известного комикса XKCD №936 и online-сервиса grc по оценке сложности пароля.

```
# Какие ассоциации вызывает у вас
    следующий пароль?
$ hsxkpasswd
==89&drink&PATTERN&holland&17==
```

Menlo

`Menlo` — это кодовое имя для CPAN-клиента cpanm версии 2.0. Основная предпосылка для создания menlo — упростить дальнейшее развитие cpanm с возможностью создания плагинов, хуков, скриптового API. Всё это было раньше затрудненно из-за способа реализации

cpanm: огромный класс и зависимые модули, упакованные в один исполняемый файл. На данный момент `Menlo` — это отдельная ветка в git-репозитории cpanm и пока находится в стадии разработки.

HTTP::Headers::Fast::XS

`HTTP::Headers::Fast::XS` — это XS-реализация модуля `HTTP::Headers::Fast`. Судя по бенчмарку, модуль в несколько раз быстрее `HTTP::Headers/HTTP::Headers::Fast`.

PDLA

`PDLA` — это форк известного модуля `PDL` — языка данных Perl для научных расчётов

и работы с многомерными массивами данных. Акроним PDLA расшифровывается как *PDL Agile*, и основная цель форка ускорить и упростить разработку PDL. Текущий PDL будет поддерживаться, все исправления в PDL будут портироваться в PDLA. Отказ от использования имени PDL3 в пользу DPLA был мотивирован отрицательным примером Perl 5 с Perl 6.

Обновлённые модули

DBI 1.634

Новый релиз универсального интерфейса к базам данным `DBI` содержит несколько небольших исправлений и улучшений. Кроме того, теперь каждый модуль дистрибутива содержит прагму `use strict`, и

хотя это изменение достаточно тривиальное, есть вероятность того, что это сломает чей-то работающий код. Поэтому следует внимательно протестировать релиз перед отправкой в рабочее окружение.

Devel::NYTProf 6.02

Обновлён мощный профайлер Perl-кода `Devel::NYTProf`. В новой версии появилась возможность поиска по имени функции на генерируемых страницах «огненных графиков» (Flame Graph). Также обновлена зависимость с `JSON::Any` (модуль устарел и не поддерживается) на `JSON::MaybeXS`.

ExtUtils::MakeMaker 7.06

После 29 пробных релизов вышел новый стабильный релиз модуля `ExtUtils::MakeMaker` для генерации мейк-файлов. Выпуск содержит множество исправлений ошибок, а также новые возможности, например, `PREREQS` теперь поддерживает указание диапазона версий. Переработана поддержка XS-модулей, появилась опция `XSMULTI=>1`, которая позволяет помещать несколько xs-файлов в одном каталоге в lib.

Mouse 2.4.5

Новые версии `Mouse` содержит важные исправления для работы модуля на Perl ≥ 5.22.0 с поддержкой тредов, также исправлена сборка модуля на старых версиях Perl

≤ 5.14. Теперь новый майнтейнер — Syohei Yoshida, который подхватил сопровождение модуля у Fuji Goro.

Text::Xslate 3.3.7

Обновлённая версия шаблонизатора `Text::Xslate` также была выпущена новым майнтейнером: Syohei Yoshida. Исправлена работа на Perl ≥ 5.22.0 и возвращена возможность сборки на Perl 5.8.8. Исправлена ошибка в работе директивы INCLUDE внутри фильтров.

Git::CPAN::Patch 2.2.0

Вышла новая версия утилиты `git-cpan`, которая позволяет импортировать

любой CPAN-модуль в локальный git-репозиторий с полной историей релизов и одной командой отправлять патчи в CPAN RT-трекер. В новой версии появилась возможность интерактивного вызова send-email (например, чтобы ввести smtp-пароль), а также сохранение имени модуля в конфиге git.

Feersum 1.403

Обновлён PSGI-совместимый веб-сервер Feersum. В новой версии появилась поддержка метода `OPTIONS`.

Mail::IMAPClient 3.37

После длительного перерыва вышло обновление популярного IMAP-клиента `Mail::IMAPClient`. Исправлено множество ошибок, долгое время висевших в RT-трекере, включая исправление ошибки аутентификации для пользователей с обратным слешем в имени.

Perl::Tidy 20150815

`Perl::Tidy` — модуль для разбора и форматирование исходного кода на Perl. В новой версии исправлено множество ошибок. Появилась новая опция `–utf8`, которая интерпретирует входной и выходной поток как текст в кодировке UTF-8, что, в частности, исправляет проблему с форматированием

юникодных строк в исходном коде, позволяя правильно вычислять их ширину.

Pod::Markdown 3.002

Вышла новая мажорная версия модуля `Pod::Markdown` для конвертирования POD-документации в формат Markdown. Есть определённые изменения в API, исправлены проблема с кодированием символов & и < в ситуациях, когда они не должны трактоваться как HTML-разметка. В состав дистрибутива теперь входит модуль `Pod::Perldoc::ToMarkdown`, что позволяет отображать POD-документацию в формате Markdown с помощью команды `perldoc`:

```
$ perldoc -o Markdown Plack::
    Handler
```

App::Sqitch 0.9993

Утилита `sqitch` — это полноценная система для управления изменениями в базе данных, позволяющая проводить контролируемые обновления и откаты схемы и данных, независимая от используемой СУБД, ORM и фреймворков. В новой версии множество исправлений и улучшений, также появилась концепция «каталога переделок» (*reworked directory*), когда при большом числе переделок становится тяжело визуально (или в IDE) отслеживать другие скрипты изменений, теперь можно перенести все переделки в отдельный каталог.

Image::ExifTool 10.00

Вышел новый мажорный релиз модуля `Image::ExifTool` для чтения и записи мета-информации для множества форматов файлов, создаваемых цифровыми камерами различных производителей. Новый релиз содержит информацию о новых типах камер и объективов. Версия 10.00 отмечена как текущая стабильная версия.

HTML::Shakan 2.02

Обновлён модуль генератора и валидатора HTML-форм `HTML::Shakan`. В новой версии исправлена XSS-уязвимость в формировании виджета `textarea`.

Win32 0.52

Вышел новый релиз модуля `Win32` для доступа к Win32 API на платформе Windows. В новой версии реализована базовая поддержка Windows 10.

■ *Владимир Леттиев*

5 Интервью с Алексеем Капрановым

Алексей Капранов (kappa) — Perl-программист со стажем, менеджер крупных проектов на Perl и не только

Как и когда научился программировать?

Начал что-то программировать в 1993 году, когда учился в 9-м классе. Это были Искры-1030 (советский аналог IBM PC XT, гениальные машины с хардверной клавишей рус/lat на клавиатуре и ручкой громкости у спикера) с GW-BASIC-ом. Там ещё надо было нумеровать строки. В отличие от многих ровесников у меня в жизни почти совсем не было не-интеловских машинок, таких как Спектрумы или БК-шки. Чуть-чуть зацепил УКНЦ, где нам преподавали Школьный Алгоритмический Язык.

Очень, кстати, хороший. Я писал к нему небольшой компилятор в Parrot VM.

Увлёкся программированием очень быстро, начал ходить в олимпиадную группу, пересел на Турбо Паскаль. Тогда нас начали пускать за “двойки” с цветными экранами. Через год-два уже что-то там выигрывал.

А дальше я пошёл учиться на профессионального программиста в БГТУ. Ну и понеслось.

Какой редактор используешь?

Работаю в vim, но в последнее время в рамках личной программы расширения горизонтов живу в режиме `alias vim = 'echo use emacs'` + spacemacs. Получается, хотя и чертыхаюсь иногда. vi для меня родной.

Как и когда познакомился с Perl?

Уже в институте, опять же расширяя личные горизонты, пробовал разные операционки. Пожил пару недель на модной в те годы OS/2, плюнул, поставил FreeBSD 2.1.7 — Линукс тогда в Брянске найти было сложнее. FreeBSD меня полностью поглотила — в первую очередь своей открытостью к любопытному пользователю. Постоянного доступа в интернет не было, но и базовая система предоставляла настолько много всего, что хватило надолго. И там был Перл (правда, сначала четвёртый, который совсем без ООП). К третьему курсу я начал делать на Перле все институтские задания. В конце концов дошло до того, что и дипломная работа у меня состояла из десятка скриптов, соединённых через пайпы и inetd.

С какими другими языками интересно работать?

Следующий мой любимый язык после Перла — Лисп, причём не конкретная реализация, а всё семейство и сам принцип, когда в качестве исходного кода программист пишет сразу AST. Современный Лисп, на котором я иногда пишу, это Clojure, хотя в нём и чувствуется некоторое предательство в угоду практичности (например, ограничения в recur или отдельные контейнерные типы кроме списков).

Очень люблю чистый Си, но не C++. Считаю, что реализация ООП в C++ грязная, хотя и очень мощная. Её подчистили и довели до приличного вида только в Java.

Много кода написал на Яваскрипте.

В принципе, сейчас уже могу с удовольствием писать на чём угодно, если там есть анонимные функции.

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

Это один из вопросов, сильно привязанных к контексту сравнения. Я убеждён, что все конкретные технические преимущества за годы были повторены в других языках — начиная от очевидных регексов и заканчивая CPAN-ом. Перл же, во-первых, остался удобной комбинацией таких преимуществ (достаточно быстрый, достаточно мощный, отлично поддерживает Unicode, имеет культуру тестирования и обратной совместимости и так далее), а во-вторых нарастил себе уникальное сообщество.

Что, по-твоему, является самой важной

особенностью языков будущего?

Название начинается на „J“, а заканчивается на „T“. Ближайшее будущее, мне кажется, это время полного доминирования JavaScript везде. Интересно, что у него нет никаких выдающихся особенностей, которые привели к такому положению вещей. Он просто выиграл в лотерею, когда стал ассемблером веб-страничек в Netscape Navigator-е.

Я не уверен, что развитие языков программирования будет продолжаться бесконечно. Мне кажется, что мы близки к порогу, когда программы будут не писаться последовательностью инструкций, а выучиваться на примерах. Выживут языки, на которых будут делать платформы обучения и сбора примеров. Но это длинный горизонт, за который заглядывать тяжело

и бессмысленно.

Что думаешь о Perl 6?

Очень много всего передумал про Perl 6 за эти годы. Сейчас смотрю на него с лёгким светлым сожалением. Мне уже не предоставится случая использовать его в продакшене, это поезд ушёл. Но я точно буду писать на Perl 6 всякую мелочь, которую сейчас я делаю на Perl 5/Clojure. Однако, я понимаю, что без большого продакшена у Perl 6 не вырастет свой CPAN, без которого невозможно за полчаса накидать очередную домашнюю мелочь, какую-нибудь бэкапилку или веб-скрейпер.

Как проект Perl 6, конечно, абсолютно провалился. Интересно будет лет через 15 прочитать книжку о том, как так вышло, да ведь никто её не напишет. Надо Дэмиану

под это дело Кикстартер завести.

Расскажи про первые годы Moscow.pm, первые воркшопы и конференции в СНГ. Есть ли будущее у специализированных конференций типа YAPC?

Я начал участвовать в оффлайновой перловой движухе после поездки в 2007 году на свой первый европейский YAPC в Вене. Ну как начал — просто помогал Андрею Шитову, который загорелся этим тогда же и начал активно воплощать идеи в действительность. Moscow.pm к тому времени существовала, но без регулярных мероприятий. Что было дальше, хорошо описано здесь: http://yapcrussia.org/. Я играл тут небольшую роль, помогал в разных местах, например с блиц-докладами. Как участник и докладчик я получал регулярной приток знакомств, мотивирующих идей и поводов

сгонять в прекрасный Киев или даже в Европу. Кажется, что участие несколько раз помогало мне решать кадровые проблемы на работе.

Есть ли будущее у YAPC — безусловно. Даже по инерции эта тусовка будет продолжать собираться, а ведь там ещё и каждый год сколько всего нового появляется. Интересное слово «специализированных». Да, тематические конференции никуда не денутся, даже наоборот, будут появляться ещё более узкие тематики. См. MojoConf, DancerConf. Объём знаний постоянно растёт, а люди же не умнеют, поэтому им приходится суживаться.

Как учиться продвинутым программистам, у которых новые проекты не появляются даже каждый год?

На Курсере. Я попытался чуточку соптимизировать свой прошлый опыт с Курсерой — взял курс с очень интересной и новой для меня тематикой (Биоинформатика) и начал делать все задания на новом, хоть и знакомом мне языке (Clojure). Получилось тяжело, часов по 10-15 в неделю, но очень хорошо. Два зайца убиты точно, а может и больше.

Например, сейчас абсолютно всем программистам необходимо заниматься машинным обучением. Курсов полно, а язык можно использовать любой, так что рекомендую взять не Перл. Пойдёт тяжело, со скрипом, но разгонитесь, после Перла мозги быстрые и ловкие.

Если программирование это по большей части ремесло, почему возникают вопросы этики и свободы программ?

Профессионально ли отказываться от «неэтичных» проектов?

Я не вижу никакой зависимости между свободой отказаться от плохого, неэтичного проекта и степенью «ремесленности» профессии. Это вопрос общей этики любой деятельности. Помогать программировать цензуру в Роскомнадзоре также плохо, как придумывать про неё законы. И также необходимо, если нет другого способа сохранить населению России доступ к мировому Интернету.

Судя по planetperl.ru, на русском о Perl пишут довольно мало. С чем это связано?

Я довольно неактивно слежу за новыми источниками, planetperl последние пару лет пополняется только читательскими усилиями, вполне возможно, что многое не охва-

чено. Но вообще популярность Перла среди новичков очевидно падает, и пишут о нём действительно всё меньше.

Победит(л) ли open source?

Так себе вопрос :) Open source как модель разработки ПО занял важное место в ряду других моделей. Станет ли весь софт открытым? Нет, безусловно не станет до тех пор, пока человечество не перейдёт к фазе всеобщего обязательного благосостояния.

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

Я работаю в Яндексе техническим менеджером инфраструктуры Яндекс.Почты. В продакшене Яндекса ровно 0 строк моего кода. Гордиться тут нечем, но вот такой факт.

Стоит ли советовать молодым программистам учить сейчас Perl?

Да, стоит, но в ряду других необязательных языков. Люди разные, и мы точно по себе, по нашей тусовке знаем, что встречаются люди с головой, повёрнутой таким особенным образом, что им очень нравится Перл. Будет плохо, если молодой программист с именно такой головой не попробует этот язык, а будет мучаться с Питоном, Явой или C++.

Вопросы от читателей

Что лучше: быть программистом или менеджером?

Менеджером быть полезнее, а программистом легче. Что из этого лучше? Убеждён, что каждому следует попробовать обе

ипостаси и принять решение на основе собственного опыта. Есть примеры отличных программистов-интравертов, которые внезапно обнаружили в себе талант управления людьми и стали гораздо счастливее. Обратных примеров не знаю.

Похож ли турецкий на Perl?

О, предвкушал такой вопрос. Турецкий язык — моё хобби. Сложный, богатый язык, совершенно инопланетянский, очень далёкий от всех индоевропейских языков. Шутка ли, рядом с турецким английский, испанский, русский, персидский и хинди — братья одной семьи.

Синтаксис постфиксный, зависимые слова всегда перед главным. Поэтому в предложении сказуемое всегда в самом конце, перед точкой. Благодаря такому синтаксису язык

скорее похож на Лисп (префиксные и постфиксные языки друг к другу гораздо ближе, чем к инфиксным).

В турецком огромная, ветвистая система причастий/деепричастий и разных других глагольных форм, многим из которых нет аналогов в русском. Причастия используются там, где в русском и английском принято вводить вспомогательные предложения. Фраза типа: «человек, который стоял под деревом, когда я пробегал мимо, так как спешил на работу» на турецком будет состоять из нанизанных друг на друга отглагольных имён (что-то типа: «на работу моей спешки из-за, моего мимо пробегания во время, под деревом стоящий человек»). А это очень похоже на анонимную функцию, которая передаётся параметром в другую анонимную функцию, а результат, тоже являющийся

анонимной функцией, вызывается ещё раз.

Сближает с Перлом обилие синонимов. Наверное основной источник богатства турецкого языка — зоопарк разных слов, обозначающих примерно одно и то же. Этому есть интересные исторические причины, например, Османская империя в бытность халифатом впитала два огромных словаря мусульманского мира — персидский и арабский. Позже, налаживая контакты с Европой, турки подружились с французами, и словарь западной цивилизации восприняли на французском. И наконец уже националисты Турецкой республики, начиная с Ататюрка, в борьбе за чистоту языка многим заимствованным словам изобрели и изобретают аналоги на основе древнетюркских корней. Ну вот как в России аэроплан называют самолё-

том, так и в Турции компьютер называют «знаниесчётчиком».

В результате, для турецкого очень верно TIMTOWTDI — можно выбрать набор подходящих синонимов для любого слова и получить либо очень официальный стиль, либо совсем разговорный, либо вычурно академический, либо произвольную дикую комбинацию. Красота.

MovableType еще можно пользоваться?

Можно, но в этом есть смысл только если уже имеется старый блог, а конвертацию базы делать неохота. Для новых сайтов MT я бы не рекомендовал. Там очередная итерация закрывания лицензий сейчас происходит, а собрать небольшую блогосистему из кирпичей с CPAN-а сейчас проще простого. Было бы кстати интересно сделать с ну-

ля отображение блога напрямую из базы в формате MT.

Больше нравится вести свои открытые проекты или контрибьютить?

Второе. Руководство и управление, слежение и контроль — это тяжёлые занятия, я их могу делать, и даже хорошо, но только за зарплату или ради очень светлой цели. Помочь, подтолкнуть что-то внешнее, но открытое и полезное в роли программиста или аналитика мне проще.

Вопросы из Фейсбука

Кроме Perl, какой современный язык (или технология) наиболее достоин применения в онлайн-проектах?

Мне не нравится выбор глагола «достоин».

Сейчас программы для интернета лучше писать на JavaScript, если уж идёт речь о рекомендациях. Стратегически такой выбор обеспечивает хороший приток кадров, приемлемую производительность (и труда, и вычислений), гарантию поддержки. Однако, если у вас есть талантливый разработчик на другом языке или вам надо что-то необычное, то выбирайте с ним другие технологии. То есть про JavaScript — это слабая рекомендация, легко перебиваемая сильными критериями. Начинать сейчас большой проект на будущее на Перле в отсутствие хотя бы заготовки команды — это приключение. Но нет ничего плохого в том, чтобы хотеть приключений.

А что из мира Perl за последние пару лет больше всего удивило/порадовало?

Порадовало — недавнее объявление Ларри

Уолла о выпуске Перл 6 к грядущему Рождеству. Удивляло и удивляет — популярность Dancer; объявление о том, что бешеные японцы-организаторы YAPC::Asia перестают её делать; как некрасиво сломали smartmatch и given в 5.18.

■ *Вячеслав Тихановский*