Pragmatic Perl 12

pragmaticperl.com

Выпуск 12. Февраль 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: Андрей Шитов, Роман Чепляка, Владимир Леттиев

Корректоры: Андрей Шитов, Георгий Бажуков

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 16:17

Оглавление

1	От редактора	1
2	Мероприятия 2014-го года . .	3
3	Perl на FOSDEM 2014	7
4	Как настроить сервер для CPAN Testers	16
5	Обход дерева директорий на Perl и Haskell (часть 2)	46
6	Обзор CPAN за январь 2014 г.	68
7	Интервью с Рэнделом Шварцем (Randal Schwartz)	79

1 От редактора

Друзья, несколько коротких новостей, прежде чем вы начнете читать этот выпуск.

В Perl могут появиться сигнатуры функций. Можно уже сейчас собрать Perl с их поддержкой, например, как это делается с `perlbrew`:

```
git clone https://github.com/Perl/perl5
cd perl5
perlbrew install ./ --as perl-with-signatures
```

и дальше:

```
sub foo ($one, $two=2) {
    ...
}
```

Пару недель назад был взломан http://blogs.perl.org. Если вы зарегистрированы там и используете пароль где-то еще, самое время его сменить. Детальнее по ссылке http://perlhacks.com/2014/01/blogs-perl-org/.

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

Приятного чтения.

■ *Вячеслав Тихановский*

2 Мероприятия 2014-го года

Куда поехать пообщаться с любителями перла в этом году

Этот год принес несколько brand new-мероприятий и городов. Наряду с традиционными городами и странами, где регулярно проходят Perl-мероприятия, появилось несколько новых.

26–28 марта — 16-й немецкий Perl-воркшоп в Ганновере. Это старейшее ежегодное мероприятие и один из немногих воркшопов, который длится три дня. Большинство докладов на немецком языке.

25 апреля — голландский Perl-вокршоп в Утрехте. В этом году активное уча-

стие в организации мероприятия принял Theo van Hoesel, номинант программы Send-a-Newbie 2013 года, получивший бесплатную турпутевку на YAPC::Europe 2013 в Киеве. Несмотря на то, что в правилах написано, что предпочтительным языком выступлений является голландский, среди участников с каждым годом все больше и больше тех, кто говорит по-русски.

16–18 мая — второй польский Perl-воркшоп в Познани. Первый, состоявшийся год назад в Варшаве, был очень удачным и привлек много посетителей как из самой Польши, так и из нескольких других стран, поэтому все доклады, даже от местных спикеров, были на английском языке. Полгода спустя еще один польский энтузиаст озвучил желание провести воркшоп в Познани. Должно получиться интересно.

20–21 мая — первый чешский Perl-воркшоп в Праге. На сегодня зарегистрировано менее десяти участников, но это мероприятие привлекательно именно тем, что оно первое в Чехии. (В Словакии проходили уникальные двухдневные воркшопы TwinCity: один день был в Братиславе, второй — в Вене.)

24 мая — Mojolicious conference в Осло. Мероприятие, проводимое группой Oslo.pm, состоит из однодневной конференции, дня тренингов и дня на хакатон.

14 июня — шестая YAPC::Russia в Киеве. Конференция с 2008 года перемещается между Москвой и Киевом, и в этом году дата выбрана таким образом, чтобы участники из России могли воспользоваться праздниками и приехать в Киев на два-три дня.

23–25 июня — YAPC::NA в городе Орландо. Североамериканские YAPC — вторые по численности после японских.

22–24 августа — пятнадцатая конференция YAPC::Europe. В этом году она в Софии. В очередной раз конференция приезжает в Восточную Европу, что не может не радовать. До этого в Софии в течение пяти лет проходили болгарские Perl-воркшопы. Не исключено, что воркоп состоится и в этом году, ориентировочно **1 марта**.

На этом календарь объявленных мероприятий заканчивается, но в ближайшие месяцы наверняка появится новая информация о событиях второй половины 2014 года.

■ *Андрей Шитов*

3 Perl на FOSDEM 2014

Рассказ о потоке про Perl на конференции FOSDEM 1 февраля 2014

Примерно в то же время, когда год назад редактор журнала писал о возрождении перла, произошло еще одно возрождение. На ежегодной опенсорсной конференции FOSDEM усилиями голландской энтузиастки Wendy van Dijk был организован Perl devroom, для которого в спешном порядке удалось собрать несколько докладов, заполнивших целый день. Тогда посетителей было больше, чем позволяла вместить комната, некоторых внутрь попросту не пустили.

В этом году в расписании одиннадцать докладов, которые идут непрерывным пото-

ком с 11 до 19 часов.

Большинство докладов были обзорными и освещали современное состояние языка и технологии, связанные с ним.

FOSDEM, как обычно, перегружен посетителями, но в целом здесь стало заметно чище и аккуратнее. Я пропустил открытие и первый доклад про «современный IRC-клиент для браузера» Маркуса Рамберга.

Затем вместо отмененного доклада «Asynchronous programming: Futures» был Питер Рэббитсон (ribasushi) с докладом «**Benchmarking is hard**». Предыдущая версия этого доклада, которая называлась «Benchmarking is *REALLY* hard», доступна в видеоархиве YAPC::Europe 2013 (а сам доклад живет уже два года).

Салве Нильсон — основатель группы Oslo.pm и любитель порассказывать о том, как создать PM-группу и что из этого можно сделать. Он предложил доклад «**Perl Community Essentials**. How to get the most out of the Perl community?», рассчитанный на тех, кто хочет присоединиться и еще не знает, с чего начать. Салве рассказал о том, какие ресурсы существуют вокруг CPAN, какие есть сайты по перлу, что происходит в IRC и про основные места и теги в Твиттере, про конференции, воркшопы и хакатоны, про организации типа YAPC Europe Foundation и, разумеется, про сами PM-группы. Отдельно была затронута важная тема о том, как улучшать репутацию языка и сообщества.

Следующий доклад, «**Writing novels using Perl**», сделанный испанским программистом Х. Х. Мерело, был скорее развлека-

тельным, хотя он наверняка оказался бы полезным для тех, кто занимается SEO и рерайтингом. Первая часть выступления была посвящена тому, как с помощью перла возможно генерировать более или менее осмысленные тексты, которые даже можно оформить в виде книги и продавать на Амазоне: «The book of Pi» написана несложным скриптом на перле. Из более серьезных тем автор поговорил об авторских правах, открытых исходниках, хуках в гите для автоматической публикации и о том, надо ли тестировать все подряд перед публикацией.

В противовес этому докладу далее последовал «**A/B testing: what your mother never told you**» Куртиса Поэ (Ovid). Докладчик — автор книги Beginning Perl, изданной в августе 2012 года. Все места в зале были заняты, и многие слушали стоя вдоль

стен или сидя на подоконниках. Куртис рассказал о том, что А/Б-тестирование направлено на тестирование не кода, а посетителей и их поведения, о том, в каких условиях уровень доверия в 90,% лучше 99,%, о важности изменения *только одного* параметра в пределах эксперимента, о том, что надо иметь терпение и проводить эксперимент как минимум в пределах одного бизнес-цикла (например, не останавливать начатый в понедельник эксперимент, который в четверг кажется очень успешным, но еще ничего не известно о том, что произойдет на выходных), про А/А-тесты и конфликтующие эксперименты. Одним словом, хороший обзорный доклад про А/Б-тестирование. Слайды доступны на SlideShare.

Sawyer X представил доклад под названием «**Perl and the Web — A Love Story**».

Доклад на самом деле является призывом к использованию PSGI и отказу от старых технологий вроде Apache, mod_perl и модуля CGI. Вначале прозвучал тезис о том, что CGI и mod_perl — «сегодня это просто глупо». Автор рассказал о преимуществах использования протокола PSGI, о фреймворках и веб-серверах, работающих с этим протоколом, и показал примеры простейших приложений с использованием Dancer 2, Web::Simple, Mojolicious, Amon 2, пример middleware-кода для запроса пароля, а также варианты конфигурации сервера и настроек для запуска Starman через Ubic.

Дэйвид Лоу (David Lowe) выступил с докладом «**Perl 5 and Unicode**». Несмотря на то, что в рассылках часто возникают вопросы о том, как решить бесконечные проблемы с перекодировками при чтении и записи в файлы и в консоль, реализация юникода в

перле считается самой полной среди других языков программирования. Тем не менее, вопросов меньше не становится, и время от времени очень полезно в очередной раз разобраться с тем, что такое юникод и как он реализован в перле. Рекомендую к просмотру, когда появится видео. Хорошее дополнение в коллекцию полезных докладов Александра Орловского: «Почему вы не знаете Unicode» и «Unicode. Ликбез».

Автор отметил, что в самом ядре перла есть темные моменты при работе с юникодом, вызывающие типовые ошибки и терминологические трудности (например, рекурсивное определение строки: «строка — это последовательность строковых элементов»). Отдельное напоминание о том, что флаг UTF-8 должен использоваться только самим интерпретатором для внутренней оптимизации хранения строки в

памяти и работы встроенных функций (например, length или ucfirst), но не для того, чтобы прикладная программа пыталась определить кодировку строки.

В вечерней программе было заявлено два доклада одного выступающего, который не пришел из-за болезни. Его время частично заняли спонтанно вызвавшиеся докладчики, в том числе вновь Питер Рэббитсон.

Из непослушенного мной остались доклады «Net::LDAP. Basic concepts of LDAP, the Net::LDAP module and some real life examples» и «Perl 6: what can you do today? State of the Butterfly».

Организаторы конференции вели видеозапись, поэтому через какое-то время файлы будут доступны на сайте конференции.

■ *Андрей Шитов*

4 Как настроить сервер для CPAN Testers

В этой статье рассказано о том, как поднять окружение для автоматического тестирования новых модулей из CPAN и отправки отчетов на сайт cpantesters.org

CPAN Testers — сайт, где публикуются результаты автоматических тестов модулей, загружаемых на CPAN. Тесты выполняются под управлением разных операционных систем, на разной архитектуре и под разными версиями перла.

На январь 2014 статистика показывала более ста версий Perl, от 5.3.0 до 5.19.8, и около тридцати типов операционных систем. Разумеется, не все комбинации доступны, а матрица пересечений крайне разрежена. В топе операционных систем

находятся Linux (на него приходится максимум вариантов версий перла), OpenBSD, FreeBSD, Solaris, NetBSD, Windows, OS X и Debian. Неудивительно, что основные стабильные версии (5.8, 5.10, 5.12, 5.14 и все следующие четные версии и их подверсии) тестируются наиболее активно.

Такое разнообразие получено практически бесплатно, потому что вся работа по тестированию ведется отдельными энтузиастами на своих компьютерах. (Неплохой пример работающей уже несколько лет распределенной вычислительной системы, созданной на перле.)

Архитектура CPAN Testers и история создания сайта кратко описана в документации к одноименному модулю. Работа над сайтом велась на нескольких QA-хакатонах, проходящих с 2008 года. До 2010 года от-

четы о тестировании высылались (в том числе при автоматическом тестировании) по ~~факсу~~ электронной почте, а теперь все отправляется через HTTPS в формате JSON. Отчеты принимает сервер под названием Metabase.

Вступить в ряды тестеров (эпизодических или постоянных) относительно легко. В простейшем случае требуется настроить свою среду таким образом, чтобы при установке или тестировании вами любого модуля (вместе со всеми зависимостями) автоматически отправлялся отчет. На этой основе очень просто построить сервер, который будет круглосуточно загружать и тестировать новые модули, попутно отправляя в метабазу все отчеты.

CPAN Testers рекомендует для установки модулей использовать классический

CPAN и CPANPLUS. Именно вокруг них построена вся архитектура генерации и отправки отчетов. Несмотря на то, что сегодня популярнее cpamninus, есть смысл изучить работу CPAN Testers именно на оригинальном модуле CPAN. CPANPLUS я рассматривать не буду, но зато расскажу про то, как тестировать модули с помощью `cpanminus` — это совсем никак не отражено на вики wiki.cpantesters.org, но с апреля 2013 года Breno G. de Oliveira (garu) разрабатывает утилиту `cpanm-reporter`, которую, возможно, выберут приверженцы cpanm.

Даже если ваш выбор — `cpanm`, но вы хотите разобраться в том, как работает вся экосистема CPAN Testers, не переходите сразу к чтению последнего раздела, а изучите сначала работу с `CPAN.pm`; многое из этого будет использовать и `cpanm-reporter`.

Создание и отправка отчетов с помощью CPAN.pm

Подготовка

Основная архитектурная идея заключается в том, что вся работа по отчетам тестирования выполняется через хуки в самом CPAN.pm. Требуется лишь установить модули для CPAN Testers, настроить SSL и создать локальный файл профиля для cpantesters.org.

Прежде всего необходимо убедиться, что перл сможет ходить на сайты по HTTPS. Если этого нет, надо поставить `Crypt::SSLeay`, например, так:

```
sudo apt-get install libssl-dev
sudo cpan Crypt::SSLeay
```

(В скобках замечу, что CPAN Testers не против и HTTP, но вряд ли стоит пользоваться такой возможностью.) Если у вас нет поддержки SSL, то при при первой попытке отправить отчет вы получите такую ошибку:

```
Scheme 'https' is not supported
    by your LWP::UserAgent.
You must install Crypt::SSLeay or
     IO::Socket::SSL or use http
    instead.
```

Теперь надо установить модуль `Task::CPAN::Reporter`, вместе с которым поставится и все необходимое для отчетов, их генерации, редактирования и отправки на сервер Metabase:

```
sudo cpan Task::CPAN::Reporter
```

Среди прочего установится и модуль `Test::Reporter::Transport::Metabase`,

именно он отвечает за формат отчета и общение сервером metabase.cpantesters.org.

Завершающие шаги подготовки — инициализация CPAN::Reporter и создание своего Metabase-профиля.

Это делается из командной строки `cpan` (без `sudo`):

```
$ cpan
cpan[1]> o conf init test_report
```

Вам предложат ответить на несколько вопросов:

- `Generate test reports if CPAN::Reporter is installed (yes/no)?` (правильный ответ: yes)
- `Would you like me configure CPAN::Reporter now?` (yes)

И попросят указать имя и электронный адрес:

- `email_from?`

В ответ следует ввести сразу и имя, и PAUSE ID (если он есть), и e-mail (хотя вопрос сформулирован как `email_from?`, больше никаких вопросов про имя и псевдоним не последует). В моем случае это:

```
"Andrew Shitov (ANDY)" <
   andy@shitov.ru>
```

- `edit_report?` (no, хотя вы можете зачем-то ответить yes)
- `send_report?` (yes, в этом случае отчет будет отправляться сразу после установки или тестирования модуля, педанты privacy должны быть

начеку)

Наконец, последний вопрос — про транспорт. Система построена таким образом, что готовые отчеты могут, например, сохраняться в файл, уходить в /dev/null или отправляться по почте. По умолчанию предлагается самый разумный вариант: отправка на сервер Metabase.

- `transport? [Metabase uri https://metabase.cpantesters.org/api/v1/ id_file metabase_id.json]` (просто соглашаемся)

(Если вы заинтересовались возможными способами доставки отчетов, изучите модули Test::Reporter::Transport::*. При отправке в методе `send` модуля `Test::Reporter` создается инстанс одного из подклассов:

```
my $transport_class = "Test::
    Reporter::Transport::
    $transport_type";
```

Однако, в документации указано, что этот подход не предназначен для создания своих транспортных модулей-плагинов, а все это планируется делать в неймспейсе `CPAN::Testers`.)

Наконец, создание профиля:

- `Would you like to run 'metabase --profile' now to create '/home/ash/.cpanreporter/metabase_.json'?` (y)

Все почти настроено, надо сохранить сделанные изменения в файле `~/.cpan/CPAN/MyConfig.pm`:

```
cpan[2]> o conf commit
```

После этой процедуры будут созданы файлы `~/.cpanreporter/config.ini` и `~/.cpanreporter/metabase_id.json`.

Файл `metabase_id.json` рекомендуют положить в отдельный каталог `~/.cpantesters`. Его можно перенести из `~/.cpanreporter` и поправить путь в строке `transport` файла `~/.cpanreporter/config.ini`:

```
transport=Metabase uri https://
    metabase.cpantesters.org/api/
    v1/ id_file ~/.cpantesters/
    metabase_id.json
```

(Не пытайтесь в оболочке `cpan` в последнем вопросе сразу указывать `~/.cpantesters/metabase_id.json`, этот каталог не будет создан, и профиль никуда не запишется.)

Файл профиля `metabase_id.json` следует копировать на все компьютеры, на которых вы собираетесь выполнять тестирование и отправлять отчеты в метабазу. В документации отдельно указано, что всегда необходимо подписываться одним и тем же именем, независимо от того, откуда вы работаете. Для внесения изменений в профиль существует утилита `metabase-profile`.

Эксплуатация

Настройка закончена. Теперь любой вызов `cpan install` или `cpan test` будет отправлять на CPAN Testers отчеты о модулях и всех зависимостях, которые пришлось поставить.

Например:

```
cpan test Date::Converter
```

Среди вывода утилиты `cpan` для каждого устанавливаемого модуля окажутся такие строки:

```
CPAN::Reporter: Test result is 'pass', All tests successful.
CPAN::Reporter: preparing a CPAN Testers report for Date-Converter-1.1
CPAN::Reporter: sending test report with 'pass' via Metabase
```

Убедиться, что отчет получен, можно, заглянув в файл metabase.cpantesters.org/tail/log.t который обновляется раз в пять минут и содержит список последних 1000 отчетов. Вот искомая запись:

```
[2014-01-16T00:22:35Z] [Andrew Shitov (ANDY)] [pass] [ANDY/
```

```
    Date-Converter-1.1.tar.gz] [
    arm-linux-gnueabihf-thread-
    multi-64int] [perl-v5.14.2] [4
    c0d34b6-7e44-12e5-a0c2-
    f318f6b0b9cd] [2014-01-16T00
    :22:35Z]
```

Чтобы не отправлять дубликаты, ведется файл `~/.cpanreporter/reports-sent.db`. При повторном тестировании модуля в этом случае появится подобное сообщение:

```
1 CPAN::Reporter: this appears to
    be a duplicate report for the
    test phase:
2 PASS Test-utf8-1.00 armv6l-linux
    3.6.11+
3
4 Test report will not be sent.
```

Если же отчет все-таки отправился, то его тело будет выглядеть примерно так:

```
{
  "content": "[{\"content
     \":\"{\\\"osversion
     \\\":\\\"3.2.0-4-amd64\\\",
  \\\"textreport\\\":\\\"This
     distribution has been tested
      as part of
  the CPAN Testers .....}]",
  "metadata": {
      "core": {
          "resource": "cpan:///
             distfile/ANDY/Acme-
             Test-42-0.1.tar.gz",
          "update_time": "
             2014-01-15T22:26:51Z
             ",
          "creator": "metabase:
             user:98903baf-5fc8-3
             cf9-a47b-0
             c7b5e930d28",
          "type": "CPAN-Testers-
             Report",
          "creation_time": "
```

```
                2014-01-15T22:26:51Z
                ",
12          "guid": "8879b2e5-9c71
                -3094-beab-988
                c20b655f3",
13          "schema_version": 1,
14          "valid": 1
15      }
16  }
17 }
```

В поле `textreport` показана только первая строка, в реальности там очень много текста, любопытные могут перенаправить отчет в файл, исправив конфигурацию в `~/.cpanreporter/config.ini` или сдампив переменную `$metabase_report` из модуля `Test::Reporter::Transport::Metabase`.

Отчеты при установке модулей

В предудущем разделе была продемонстрирована команда `cpan test`, которая лишь тестировала, но не устанавливала модуль, отправляя при этом отчет. С установкой дело обстоит точно так же, но есть момент, связанный с правами пользователя.

Если вы устанавливаете модуль от имени root (например, используя `sudo`), то отчеты не создадутся и не отправятся, потому что для этого пользователя нет ни Metabase-профиля, ни правильно сконфигурированного `CPAN.pm`. Решить эту проблему очень просто: достаточно повторить все процедуры от имени суперпользователя или скопировать соответствующие файлы профиля и конфигурации в папку `/root/`. После этого отчеты начнут отправляться и при командах типа

```
sudo cpan install Test::utf8
```

Возможен и другой вариант, без привлечения суперпользователя. Если вы пользуетесь Perlbrew, то установка модулей происходит в локальные каталоги, и будет использоваться профиль основного пользователя. Установка Perlbrew простая, хотя компилирование нужной версии может занять определенное время.

Установка и инициализация:

```
sudo cpan App::perlbrew
perlbrew init
```

Установка нужной версии перла:

```
perlbrew install perl-5.18.2
```

Запуск оболочки с нужным перлом:

```
perlbrew switch perl-5.18.2
```

Если продолжать работать в этой оболочке, то отчеты о тестировании будут создаваться и отправляться как при тестировании, так и при установке модулей:

```
cpan test Moose
cpan install Moo
```

Smoke-тестирование на основе CPAN.pm

Теперь все настроено для того, чтобы начать автоматически загружать с CPAN новые модули, тестировать их и отправлять отчеты на CPAN Testers. Это и есть smoke-тестирование.

Для smoke-тестирования можно выделить или отдельный сервер или виртуальную машину. Следует быть осторожным, по-

скольку модули, попадающие на CPAN, не проверяются антивирусом. Поэтому лучше всего запускать автоматизацию не от имени суперпользователя, а как минимум с помощью perlbrew, тем более что в таком случае появляется возможность тестировать модули с разными версиями перла.

Перед началом работы необходимо установить модуль `CPAN::Reporter::Smoker`:

```
1 cpan CPAN::Reporter::Smoker
```

Запуск тестов тоже крайне прост:

```
1 perl -MCPAN::Reporter::Smoker -e
    start
```

Функция `start` загружает список последних обновлений на CPAN и начинает по очереди тестировать все модули из этого

списка. По умолчанию в очередь на тестирование попадают последние сто модулей, но для каждого из них будут протестированы и все зависимости, поэтому не исключено, что фактическое число обработанных модулей окажется на порядок больше.

Загрузка списков модулей и их индексирование могут занять значительное время, после чего программа начинает цикл, в котором выполняется командная строка для тестирования или установки модуля:

```
# Start smoking
DIST:
for my $d ( 0 .. $#{$dists} ) {
  my $dist = CPAN::Shell->
     expandany($dists->[$d]);

  ...
    # invoke CPAN.pm to test
       distribution
```

```
8       system($perl, "-MCPAN", "-e",
9         "\$CPAN::Config->{
             test_report} = 1; " .
             $trust_string
10        . $reset_string . ($args{'
             install'} ? 'install' :
             'test')
11        . "( '$dists->[$d]' )"
12      );
```

Для каждого модуля формируется и выполняется подобная командная строка:

```
1 /usr/bin/perl -MCPAN -e "$CPAN::
    Config->{test_report} = 1;
    test( 'SHLOMIF/App-Countdown-
    v0.4.3.tar.gz' )"
```

Всю остальную работу ведет модуль `CPAN.pm`, поэтому если он к этому моменту настроен для работы с Metabase-сервером, то сам по себе будет отправлять отчеты для

всех «раскуриваемых» модулей.

Дополнительно имеет смысл попросить `CPAN.pm` не перепроверять модули, если они уже были протестированы и записаны в историю. Для этого следует установить флаг `trust_test_report_history` и сохранить его в конфигурации модуля:

```
$ cpan
cpan[1]> o conf init
    trust_test_report_history
cpan[2]> o conf commit
```

Кроме исключительно тестирования возможно сразу устанавливать модули. Для этого требуется передать соответствующий аргумент при вызове функции `start`:

```
perl -MCPAN::Reporter::Smoker -e'
    start(install => 1)'
```

В любом случае дистрибутивы загруженных модулей сохраняются локально в каталоге `~/.cpan/sources/authors/id`, поэтому отсутствие установленного зависимого модуля совместно с проверкой файла `~/.cpanreporter/reports-sent.db` почти никак не сказывается на скорости тестирования.

Отключение ожидания ввода

Тесты, входящие в дистрибутивы некоторых модулей, могут запрашивать ввод от пользователя. Чтобы автоматическое тестирование не останавливалось на таких местах, модуль CPAN::Reporter::Smoker устанавливает несколько переменных окружения, поэтому тесты имеют возможность узнать, стоит ли ждать ввода или нет:

```
# Let things know we're running
    automated
local $ENV{AUTOMATED_TESTING} =
    1;

# Always accept default prompts
local $ENV{PERL_MM_USE_DEFAULT} =
     1;
local $ENV{
    PERL_EXTUTILS_AUTOINSTALL} = "
    --defaultdeps";
```

Первая переменная — флаг, сигнализирующий об автоматическом режиме, две других позволяют всегда принимать ответы по умолчанию, не спрашивая человека.

На эту возможность стоит обратить внимание тем, кто создает тесты для своих модулей. Например, модуль Term::Title, позволяющий изменить заголовок окна терминала, во время тестов останавливается и дважды

спрашивает пользователя, видит ли он изменения в заголовке:

```
# Do you see '[Hello]' in the title bar of this window? (y/n)?
y

t/01-Term-Title.t .. 5/8 # Has the title bar been cleared? (y/n)?
y
```

При автоматическом тестировании это не имеет смысла, поэтому в файле `t/01-Term-Title.t` выполняется проверка переменной окружения AUTOMATED_TESTING:

```
SKIP:
{
    skip "Automated testing not supported for tab titles",
     2
```

```
4           if $ENV{AUTOMATED_TESTING
               };
```

Создание и отправка отчетов с помощью App::cpanminus

Чтобы отправлять отчеты тестирования для модулей, установленных с помощью `cpanm`, следует воспользоваться утилитой `cpanm-reporter`, для установки которой достаточно поставить модуль `App::cpanminus::reporter`:

```
1 sudo cpanm App::cpanminus::
     reporter
```

Как и в случае с тестированием через `CPAN.pm`, для начала работы требуется настройка конфигурации и создание профиля. Команда для инициализации выглядит так:

```
cpanm-reporter --setup
```

Если профиль уже был создан для `CPAN.pm`, то эту команду запускать необязательно, поскольку для для `cpanm-reporter` будут использоваться те же файлы конфигурации и профиля. В этом случае на все вопросы во время настройки будут предложены ответы, прочитанные из `~/.cpanreporter/config.ini`. При отсутствии файла `metabase_id.json`, где хранится ваш Metabase-профиль, будет предложено запустить утилиту `metabase-profile`, описанную выше.

Чтобы отправить отчет на CPAN Testers, необходимо сначала обычным способом установить модуль, используя `cpanm`, а затем запустить утилиту `cpanm-reporter`:

```
cpanm Validator-LIVR
cpanm-reporter -v
```

При запуске без ключа `--verbose` или `-v` утилита `cpanm-reporter` ничего не напечатает, и будет трудно ощутить свою причастноть к полезному делу тестирования модулей.

(`cpanm-reporter` не ведет `reports-sent.db`, поэтому при каждом запуске будут отправляться одни и те же отчеты. Чтобы этого избежать, запускайте отправку только после установки модуля, когда `cpanm` обнуляет файл `build.log` — **прим. ред.**)

Обратите внимание, что при установке модуля с помощью `sudo` придется решить те же проблемы с поиском правильных файлов логов, настроек и профилей, что и у `CPAN::Reporter`.

Важно, чтобы `cpan-reporter` был вызван как можно раньше после установки модуля.

Если этого не сделать в течение получаса, вы получите жалобу на устаревший файл `build.log`:

```
Fatal: build.log was created
    longer than 30 minutes ago.
```

Если вы уверены, что за это время ничего не могло произойти, смело запускайте репортер с ключем `--force`:

```
cpanm-reporter -v --force
```

Точно так же, как и ранее, успешно загруженные отчеты (вместе с отчетами по модулям, которые пришлось поставить в качестве зависимостей) появляются в логе metabase.cpantesters.org/tail/log.txt.

■ *Андрей Шитов*

5 Обход дерева директорий на Perl и Haskell (часть 2)

Напомню, в первой части этой статьи (см. предыдущий номер журнала) мы написали обобщенный обход дерева директорий на языках Perl и Haskell. Он представляет собой функцию `dir_walk`, которая в качестве аргументов принимает два коллбека — один для файлов, один для директорий — и обходит дерево, вызывая соответствующий коллбек на каждом элементе дерева.

Одним из ограничений написанной нами функции `dir_walk` является тот факт, что из коллбеков мы не можем остановить обход. Предположим, нам надо узнать, есть ли в дереве файл с данным именем. Если мы нашли файл, нет смысла продолжать обход дальше.

Есть несколько способов решить эту проблему.

Например, можно модифицировать коллбеки и саму функцию `dir_walk` так, чтобы они возвращали флаг, который бы прекращал обход. Можно также использовать исключения.

Здесь я хочу показать инструмент, который решает указанную проблему в качестве бесплатного бонуса, но также является интересным сам по себе и помогает во множестве других ситуаций.

Речь идет о продолжениях (continuations) и CPS (стиль передачи продолжений — continuation-passing style).

CPS знаком каждому, кто использовал IO::Async или похожие библиотеки. Срав-

ните получение ввода в традиционном стиле:

```
my $line = <>;
print "Got $line";
```

и в CPS-стиле:

```
my $stream = IO::Async::Stream->
    new(
    read_handle  => \*STDIN,
    on_read => sub {
      my ( $self, $buf, $eof ) =
         @_;
      print "Got $$buf";
      ...
    }
);
```

В традиционном стиле функции (в нашем случае оператор <>) возвращают значение напрямую. В стиле передачи продолжений функции принимают дополнительный

аргумент (в нашем случае — `on_read`), который представляет собой коллбек. Этот коллбек принимает в качестве аргумента возвращаемое значение функции (`$buf`) и дальше делает с ним все то, что в традиционном стиле происходило бы после вызова рассматриваемой функции (отсюда его название — *продолжение*). Обратите внимание, что `print` вызывается *после* `read` в традиционном стиле и *внутри* `on_read` в CPS.

Мы только что увидели, как вызывать функцию, написанную в CPS — надо ей передать коллбек (продолжение), которое делает все то, что мы хотели бы сделать с результатом функции после ее завершения. Но как преобразовать саму функцию в CPS? Для этого надо добавить еще один аргумент-коллбек, и вместо `return` передать возвращаемое значение этому

коллбеку. Также, в случае необходимости, само тело функции аналогично можно преобразовать в CPS.

Например, рассмотрим код

```
sub square($) {
  return $_[0] * $_[0];
}

sub inc($) {
  return $_[0] + 1;
}

print square inc 5;
```

После преобразование в CPS функции `inc` получим

```
sub square($) {
  return $_[0] * $_[0];
}

```

```
sub inc($$) {
  $_[1]->($_[0] + 1);
}

inc(5,
  sub { print square $_[0], "\n"
      }
);
```

Если применить CPS-преобразование еще и к `square`, то код будет выглядеть так:

```
sub square($$) {
  $_[1]->($_[0] * $_[0]);
}

sub inc($$) {
  $_[1]->($_[0] + 1);
}

inc(5,
  sub { square($_[0],
    sub { print $_[0], "\n"; }
```

```
12     )}
13 );
```

Чем же хорошо CPS-преобразование? Тем, что у CPS-функции есть контроль над тем, что будет происходить после ее завершения. Ведь вместо того, чтобы послушно передавать возвращаемое значение продолжению, она может взбунтоваться и не вызывать продолжение вовсе. Именно так мы и реализуем раннее завершение обхода дерева директорий.

Напомню, вот определение функции `dir_walk` (в традиционном стиле):

```
1 # From Higher-Order Perl by Mark
      Dominus, published by Morgan
      Kaufmann Publishers
2 # Copyright 2005 by Elsevier Inc
3 # LICENSE: http://hop.perl.plover
      .com/LICENSE.txt
```

```
4
5  sub dir_walk {
6    my ($top, $filefunc, $dirfunc)
       = @_;
7    my $DIR;
8
9    if (-d $top) {
10     my $file;
11     unless (opendir $DIR, $top) {
12       warn "Couldn't open
           directory $code: $!;
           skipping.\n";
13       return;
14     }
15
16     my @results;
17     while ($file = readdir $DIR)
         {
18       next if $file eq '.' ||
           $file eq '..';
19       push @results, dir_walk("
           $top/$file", $filefunc,
           $dirfunc);
```

```
20      }
21      return $dirfunc->($top,
           @results);
22    } else {
23      return $filefunc->($top);
24    }
25  }
```

Итак, давайте применим CPS-преобразование к нашему обходу.

```
1  sub dir_walk($$$$) {
2    my ($top, $filefunc, $dirfunc,
        $dir_walk_cb) = @_;
3
4    if (-d $top) {
5      my (@results, $DIR);
6      if (opendir $DIR, $top) {
7        my $while_loop;
8        $while_loop = sub {
9          my $file = readdir $DIR;
10         unless ($file) {
11           closedir $DIR;
```

```
12            $dirfunc->($top, \
                 @results,
                 $dir_walk_cb);
13          }
14          if ($file eq '.' || $file
                eq '..') {
15            $while_loop->()
16          } else {
17            dir_walk("$top/$file",
                 $filefunc, $dirfunc,
18              sub {
19                push @results, @$_
                     [0];
20                $while_loop->();
21              }
22            );
23          }
24        };
25
26        $while_loop->();
27
28      } else {
29        warn "Couldn't open
```

```
            directory $top: $!;
            skipping.\n";
30        $dir_walk_cb->();
31      };
32    } else {
33      $filefunc->($top,
           $dir_walk_cb);
34    }
35  }
```

Код стал менее читаемым, и в нескольких местах его пришлось переделать — например, цикл `while` был преобразован в рекурсивную функцию.

Тем не менее, код работает (кроме случаев, когда он упирается в предел по глубине рекурсии — см. раздел 5.4.1 Tail-Call Elimination книги Higher-Order Perl).

Например, вот аналог `find -type f`:

```
dir_walk(
    $ARGV[0],
    sub { print $_[0]; $_[1]->();
        },
    sub { $_[2]->(); },
    sub { });
```

А вот пример программы, которая заканчивает обход досрочно — то, зачем мы и стали преобразовывать код в CPS:

```
dir_walk(
    '/etc',
    sub { if ($_[0] =~ /\.conf$/)
        { print $_[0]; } else {
       $_[1]->(); } },
    sub { $_[2]->(); },
    sub { });
```

Этот код находит в иерархии `/etc` первый файл с расширением `conf`, печатает его имя, и на этом завершается. Так происхо-

дит из-за того, что передаваемый коллбек не вызывает свое продолжение, когда имя файла удовлетворяет регулярному выражению.

Теперь перейдем к Haskell. Там тоже можно сделать аналогичное CPS-преобразование вручную, но можно поступить и лучше. Как писалось выше, при CPS-преобразовании к каждой функции добавляется еще один аргумент — продолжение (`$dir_walk_cb` в нашем коде на Perl). В силу эффекта, который называется каррированием (см. раздел 7.1 Currying в Higher-Order Perl), это равносильно тому, что функция возвращает другую функцию. Эта вторая функция принимает в качестве своего единственного аргумента продолжение и возвращает конечный результат. Мы можем определить новый тип

```
1 newtype Cont r a = Cont ((a -> r)
```

```
      -> r)
```

Тогда функция, до CPS-преобразования возвращавшая `a`, в CPS будет возвращать `Cont r a`, где `r` — это тип результата всей программы.

Красота этого подхода заключается в том, что после определенных объявлений мы можем интерпретировать `Cont r a` как тип *действий*, возвращающих результат типа `a`. (Читателю рекомендуется обратится к первой части статьи, чтобы вспомнить, что такое действия и как с ними работать.)

Точно так же, как `IO a` — тип действий, которые могут, помимо обычных вычислений, производить ввод-вывод и другое взаимодействие с внешним миром, `Cont r a` — тип действий, у которых есть доступ к собственному продолжению. Иными

словами, функции, которые возвращают `Cont r a`, уже преобразованы в CPS.

Вместо того, чтобы использовать свой собственный тип `Cont`, мы воспользуемся готовыми определениями из модуля `Control.Monad.Cont` библиотеки `mtl`. Однако подчеркнем, что это не специальная «магия», встроенная в компилятор Haskell, а обычная библиотека, аналог который мы могли бы написать и сами. Некоторые другие языки (например, Scheme), напротив, имеют встроенную поддержку продолжений.

Вот как выглядит CPS-версия функции `dir_walk` на Haskell:

```
1 import System.FilePath
2 import System.Directory
3 import Control.Monad.Cont
4
```

```
dir_walk
  :: FilePath
  -> (FilePath -> ContT r IO a)
  -> (FilePath -> [a] -> ContT r
     IO a)
  -> ContT r IO a
dir_walk top filefunc dirfunc =
   do
  isDirectory <- liftIO $
     doesDirectoryExist top

  if isDirectory
    then do
      files <- liftIO $
         getDirectoryContents top
      let nonDotFiles = filter (
         not . (`elem` [".", ".."
         ])) files
      results <- mapM (\file ->
         dir_walk (top </> file)
         filefunc dirfunc)
         nonDotFiles
      dirfunc top results
```

```
19      else
20        filefunc top
```

В отличие от Perl-версии, изменения здесь настолько минимальны, что я не могу удержаться и не показать diff классической и CPS-версий:

```
1  --- dirwalk.hs  2014-02-02
       17:30:35.260611983 +0200
2  +++ dirwalk-cps.hs  2014-02-02
       17:32:00.820418259 +0200
3  @@ -1,17 +1,18 @@
4   import System.FilePath
5   import System.Directory
6  +import Control.Monad.Cont
7
8   dir_walk
9     :: FilePath
10 -   -> (FilePath -> IO a)
11 -   -> (FilePath -> [a] -> IO a)
12 -   -> IO a
13 +   -> (FilePath -> ContT r IO a)
```

```
+  -> (FilePath -> [a] -> ContT r
    IO a)
+  -> ContT r IO a
 dir_walk top filefunc dirfunc =
    do
-  isDirectory <-
   doesDirectoryExist top
+  isDirectory <- liftIO $
   doesDirectoryExist top

   if isDirectory
     then do
-      files <-
   getDirectoryContents top
+      files <- liftIO $
   getDirectoryContents top
       let nonDotFiles = filter (
          not . (`elem` [".", ".."
          "])) files
       results <- mapM (\file ->
          dir_walk (top </> file)
           filefunc dirfunc)
          nonDotFiles
```

Изменения сводятся к замене типов вида `IO a` на `ContT r IO a` и использованию `liftIO` для преобразования `IO`-действий в `Cont`-действия.

Поиск в `/etc` `conf`-файла с ранним завершением будет выглядеть так:

```
import Control.Monad.Cont
import Data.List (isSuffixOf)

main = runContT walk (\result ->
    return ())
  where
    walk = dir_walk
      "/etc"
      (\file -> ContT $ \k -> if
          ".conf" `isSuffixOf`
          file
           then putStrLn file
           else k ())
```

```
11          (\dir results -> return ())
```

Если к этому моменту у вас все еще остались сомнения по поводу того, что такое продолжения и как они работают, рекомендую еще посмотреть раздел 8.8.1 книги Higher-Order Perl.

Другим интересным подходом к проблеме обхода дерева, тесно перекликающимся с предыдущим, являются итераторы. Ни в Perl, ни в Haskell встроенной поддержки итераторов нет, хотя она есть во многих других языках — например, в Python.

Тем не менее, и в Perl, и в Haskell итераторы можно реализовать. Про итераторы на Perl можно почитать в главе 4 Higher-Order Perl. В частности, в разделе 4.2.2 приводится реализация dir_walk с помощью итераторов. Реализация итераторов на Perl исполь-

зует изменяемые переменные в замыканиях — не очень функционально, но что поделаешь.

На Haskell итераторы можно реализовать без использования изменяемых переменных (хотя, конечно, можно и с ними). Заинтересованному читателю рекомендуется попробовать реализовать обход дерева с использованием библиотеки pipes (и, в частности, функции yield).

Совсем храбрые и заинтересованные читатели могут обратиться к трудам Олега Киселёва о продолжениях и итераторах (генераторах).

Заключение

Надеюсь, понятно, что эта статья — не о том, как обходить дерево директорий в практических программах (иронично, учитывая название журнала). Тем не менее, какие-то идеи, изложенные здесь, могут когда-нибудь пригодиться и в реальных задачах.

Скорее, моей целью было показать, что даже если вы знаете регулярные выражения и имеете опыт использования 30 модулей с CPAN, вам есть еще чему учиться и куда развиваться в программировании. Если статья подстегнет кого-то к изучению Haskell или к прочтению Higher-Order Perl, она писалась не зря.

■ *Роман Чепляка*

6 Обзор CPAN за январь 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

В этом месяце произошёл сбой с дисковым хранилищем сервера PAUSE. С 8 по 11 января сервер был отключён, и у авторов не было возможности выкладывать модули на CPAN. Тем не менее, проблема была решена, и сейчас PAUSE работает в штатном режиме.

Статистика

- Новых дистрибутивов — 213
- Новых выпусков — 1028

Новые модули

- Shardcache

Обвязка к C-библиотеке libshardcache для создания распределённого хранилища пар ключей-значений. Библиотека была вдохновлена идеями и логикой groupcache, которая используется в Google и написана на языке Go.

- Thread::Channel

Альтернативная реализация очередей `Thread::Queue`. Благодаря использованию «умной» сереализации (`Sereal`) удаётся добиться повышения производительности, не жертвуя гибкостью.

- Text::Markdown::PerlExtensions

Модуль расширения для синтаксиса Markdown, который добавляет два элемента `M<Module>` и `A<Author>`, которые после преобразования в HTML раскрываются в ссылки на соответственно страницу модуля и автора на metacpan.org.

- Hyle

Создание простого CRUD веб-интерфейса к базе данных ещё никогда не было столь простым:

```
$ hyle.pl --dsn'dbi::SQLite::dbname=file.db'

$ curl http://localhost:8000/collection/id/7
```

```
    ...
```

- Plack::Middleware::DetectRobots

Модуль для Plack, который устанавливает переменную окружения `robot_client`, если клиент является роботом. Модуль использует список регулярных выражений из проекта анализатора веб-логов AWStats.

- Strict::Perl

`Strict::Perl` накладывает ограничения на использование устаревших и небезопасных конструкция в коде. Для Perl старше 5.12 это соответствует:

```
use strict;
use warnings qw(FATAL all);
```

```
3     use autodie;
```

Запрещаются к использованию ключевые слова `goto`, `until`, `foreach`, `format` и другие устаревшие и небезопасные операторы. Запрещены также некоторые специальные переменные (`$[`, `$#`, ...) и оператор умного сравнения `~~`.

- Moo::Lax

В противоположность предыдущему модулю `Moo::Lax` пытается облегчить жизнь тем, кому не нравится, что модуль `Moo` делает все предупреждения фатальными. Для этого достаточно вместо загрузки `Moo` загружать `Moo::Lax`.

- Lembas

`Lembas` — это тестовый фреймворк для приложений командной строки, близкий по духу Cram из мира python. Файлы с записанной shell-сессией, состоящей из команд и их вывода, используются для тестирования консольных программ. Для создания тестов из выполняемых команд используется `Test::Builder`.

- Starlight

Ещё один HTTP/1.1 веб-сервер, использующий prefork-модель для рабочих процессов. Создан как форк тредового веб-сервера `Thrall`, который в свою очередь является форком prefork веб-сервера `Starlet`. Пройдя этот «испорченный телефон» форков, `Starlight` позиционируется как веб-сервер с зависимостью только от `Plack`.

Обновлённые модули

- Type::Tiny 0.038

Хороший модуль для описания и проверки типов данных. Релиз примечателен тем, что он был самым первым в этом году на CPAN. Автор выложил его буквально через две минуты после полуночи. А остались ли у вас воспоминания о том, что вы делали в новогоднюю ночь?

- Imager 0.98

Обновился модуль для создания изображений `Imager`. В новом релизе присутствует несовместимое изменение в возвращаемом значении `setpixel()`. В случае ошибки

возвращается пустой список и устанавливается сообщение об ошибке в `errstr()`, иначе возвращается число нарисованных пикселей или строка `0 but true` если ни один пиксель не нарисован (например, за пределами изображения).

- Net::DNS 0.74

В новом релизе исправлена ошибка с крахом днс-сервера на основе `Net::DNS` при получении специально сформированного UDP-пакета.

- Pod::Perldoc 3.21

Команда `perldoc` получила новый ключ командной стоки `-a` для поиска по функциям Perl API.

- DBIx::Class 0.8270

После длительного перерыва обновился модуль `DBIx::Class` с огромным списком изменений и исправлений. Документация модуля пополнилась мануалом DBIx::Class::Manual::QuickStart о том, как разобраться с DBIC за 10 минут.

- perl 5.18.2

В этом году вышел новый стабильный релиз Perl 5.18.2, в котором исправлено несколько регрессий и ошибок.

- Net::SSLeay 1.58

Обновился модуль для работы с openssl из Perl. В числе изменений множество исправлений, включая проблемы, потенциально затрагивающие безопасность (крах в `SSL_get_peer_cert_chain` и использование указателя после освобождения в `next_proto_select_cb_invoke`).

- B::C 1.43

Почти два года потребовалось *Reini Urban*, чтобы выпустить новый релиз компилятора/транслятора Perl-кода в C. Данный релиз поддерживает Perl 5.16 и частично поддерживает 5.18 и 5.20 (blead).

- Promises 0.08

Множество улучшений и изменений в новых релизах `Promises`, реализации спецификации *Promise/A+*. Улучшено взаимодействие с реализациями мультиплексоров событий для поддержки асинхронного разрешения событий. Добавлены новые методы: `done`, `finally`, `catch`.

■ *Владимир Леттиев*

7 Интервью с Рэнделом Шварцем (Randal Schwartz)

Рэндел Шварц — соавтор нескольких книг по Perl, автор «преобразования Шварца», ведущий «FLOSS Weekly».

Когда и как начали программировать?

Научился сам очень рано, это было около сорока пяти лет назад. И я все еще учусь :)

Какой редактор используете?

GNU Emacs! Каждое утро я загружаю последний релиз из git-репозитория и компилирую, чтобы убедиться, что он все еще работает под OSX.

Когда и как познакомились с Perl?

Я загрузил Perl 1 из Usenet-рассылки как только он вышел, так как я уже был фаном Ларри из-за его программ `patch` и `rn`. Немного поигрался с языком, но вернулся к awk и sed. Потом вышел Perl 2, и это уже было большой разницей… начал переписывать *все* на нем и убеждал других делать то же самое. И для Perl 3 вместе с Ларри мы написали Camel book (правда, он переименовал Perl 3.xx в Perl 4 как только книга была выпущена).

С какими другими языками интересно работать?

Я довольно свободно себя чувствую в Smalltalk, использую его с 1983. Также я осваиваю Dart. Скорее всего это будет следующим языком после Perl, где я буду известен.

Какое, по-вашему, наибольшее преимущество Perl?

Как говорит Ларри Уолл: «правильная смесь manipulexity и whipuptitude». Если выражаться простым языком, то язык делает простые вещи легкими, а сложные — возможными. Многие языки хороши либо в одном, либо в другом.

Какими, по-вашему, свойствами должны обладать языки будущего?

Возможность решить задачу Ханойской башни :) Шучу.

Все практические языки оптимизированы для какой-то конкретной практической отрасли. Javascript, как мне кажется, не сильно подходит для программирования консольных приложений, но хорошо ра-

ботает в браузере. (Dart, по-моему, будет первым языком, который хорошо работает везде.) Поэтому нет «самого важного свойства» кроме как метасвойства «легко решать задачи в конкретной области».

Что думаете о будущем Perl?

С будущим у Perl все нормально. Развитие немного приостановилось после дотком-бума, но вполне себе оправилось с тех пор. У Perl сейчас ежегодные релизы, больше современных фишек, с каждым годом растущее число загрузок на CPAN, это говорит о том, чтоб все больше проектов выбирают Perl. Возможно, это небольшой кусок большого пирога, но пирог становится гигантским.

Самый любимый JAPH?

```
$Old_MacDonald = q#print #;
```

```
    $had_a_farm = (q—q:Just
    another Perl hacker,:—);
2 s/^/q[Sing it, boys and girls
    ...],$Old_MacDonald.
    $had_a_farm/eieio;
```

Как были вовлечены в «FLOSS Weekly»?

Я встретил Лео Лапорте (Leo Laporte) на круизе InSightCruise (тогда они назывались «Geek Cruises»). Мы подружились, он мне помог советом о том, как начать свой подкаст «GeekCruisesNewses», в итоге получилось около 150 выпусков. Он был гостем на моих ранних программах, и как-то после одной записи мы разговорились, и он рассказал, что он и Крис ДиБона (Chis DiBona) начинают подкаст об открытом ПО. Я предложил себя в роли гостя, что и воплотилось в жизнь в 9 выпуске. После 17 выпусков у Криса появилось много дел

(у него появился ребенок... что бывает), и Лео приостановил программу. Я поинтересовался, что произошло, он все объяснил и сказал, что ищет соведущего. Я вызвался и в течение ста выпусков был соведущим. У Лео появились другие подкасты, и когда он стал уверен, что я смогу вести программу, он дал мне возможность вести ее самостоятельно. Сейчас у меня есть несколько соведущих в ротации, а когда я не могу вести программу, то двое из них выполняют роли ведущего и соведущего.

Есть ли все еще рынок Perl-консультаций?

Этот рынок все еще оплачивает мои счета :)

Стоит ли молодым программистам советовать учить Perl?

Да. Сейчас довольно мало Perl-программистов. Я могу порекомендовать несколько хороших книг, если они захотят :)

Вопросы от читателей

Пишите сейчас на Perl?

В данный момент в другом окне редактора у меня открыт Perl-код. Так что да, я пишу на Perl практически каждый день.

Что думаете о текущем состоянии Perl?

Я рад, что он оправился от дотком-бума. В середине 2000-х я думал, что же я буду делать дальше.

Что думаете о Perl 6?

С одной стороны я рад, что он есть. С дру-

гой, мне неприятно, что усилия, вначале потраченные на Perl 6, могли быть направлены на Perl 5, и мы были бы, где мы сейчас, уже где-то пять лет назад. (Возможно, я не прав, но это личное мнение.)

Я *хотел бы* язык со свойствами Perl 6, особенно грамматики, бесконечные списки и многое другое, что я вижу. Но как поставщику услуг мне нужно знать, *когда* стоит вкладывать усилия в развитие (себя и других) для оказания консультационных услуг, тренингов и написания статей о Perl 6.

Что думаете о Modern Perl?

При всем уважении к chromatic, я не сильно этим интересуюсь.

Кроме вашего знаменитого преобразова-

ния, что вам самому нравится их выших находок в Perl?

Думаю, что это мое открытие (которое было позже несколько раз доказано), что любой нетривиальный Perl-код не может быть разобран статическим парсером без наличия Perl-интерпретатора под рукой. Таким образом, Perl здесь уникален. (Больше можно почитать в моем посте на http://www.perlmonks.org/?node_id=44722.)

■ *Вячеслав Тихановский*