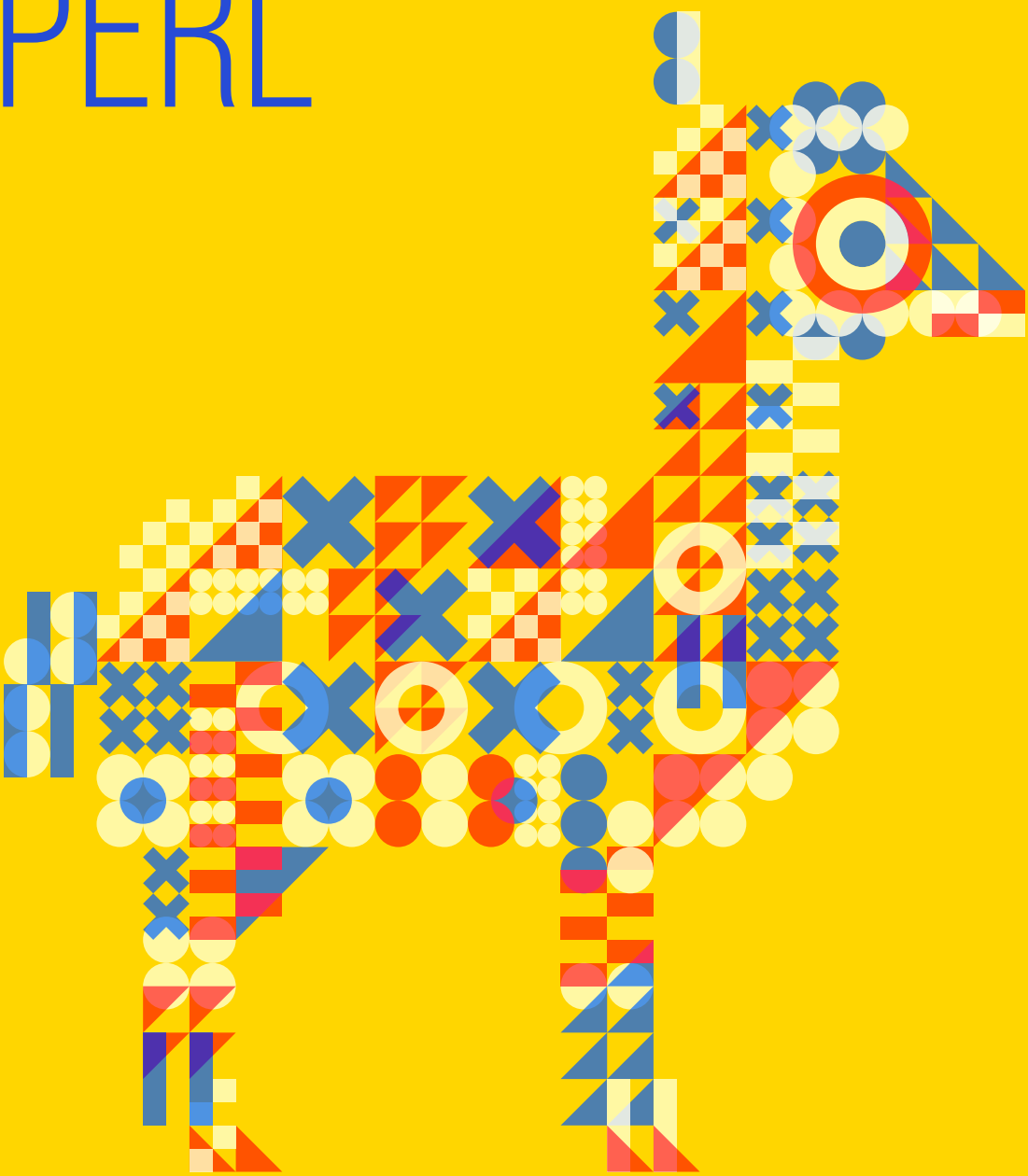
PRAGMATIC PERL

32



10/2015

pragmaticperl.com

Pragmatic Perl 32

pragmaticperl.com

Выпуск 32. Октябрь 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: Георгий Бажуков, Владимир Леттиев, Роман Миловский, Денис Федосеев

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2015-11-01 13:24

Оглавление

1 От редактора

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

Приятного чтения.

■ *Вячеслав Тихановский*

2 Анонс коллективного блога @backendsecr

@backendsecret — это коллективный твиттер для разработчиков бэкенда. Каждую неделю новый ведущий делится своими знаниями, опытом и ответами на вопросы читателей. Популярные темы: Perl, PHP, Go, Python, функциональное программирование, безопасность приложений, управление разработкой, highload и BigData. Нас уже 1100 человек. Присоединяйтесь!

Если вам есть о чём рассказать на темы, связанные с бэкендом, вы можете стать следующим ведущим. Для этого напишите куратору @backendsecret, Роману на почту: backendsecret@gmail.com — или в твиттер: @dcromster.

Подробности о проекте и правила для веду-

щих: http://backendsecret.ru/about/.

Другие коллективные твиттеры:

- @jsunderhood — Javascript и фронтенд
- @rubyunderhood — Ruby
- @cssunderhood — HTML, CSS и вёрстка
- @iamspacegray — Всё о дизайне

■ *Роман Миловский*

3 Как стать Perl-автором

Пошаговое руководство по загрузке своего первого модуля на CPAN

Думаю, многие видели обзоры CPAN-модулей от товарища Crux'а: какие интересные модули появились, в каких появились новые возможности... Кому-то, возможно, захотелось и самому сделать что-нибудь этакое. Ну что же, эта заметка для вас!

Захотелось поработать — ляг поспи

Первым пунктом в памятках "при чрезвычайных ситуациях" зачастую можно встре-

тить фразу: “сохраняйте спокойствие!”. Ровно также стоит поступать, когда захотелось написать какой-то модуль.

Для начала определитесь: а что вы хотите от этого получить? Славу, денег, женщин? Поверьте, есть более простые способы. Если же цель — решить какую-то конкретную проблему — вы на правильном пути.

Теперь нужно понять, решали ли её до вас, и устраивает ли людей это решение. Быть может не стоит писать *своё*, а присоединиться к уже существующему проекту? Зачастую это полезнее и быстрее. Но как искать, сделали ли это уже? Помимо уютного чата/форума/google есть и специализированный сайт для Perl-программистов, где можно посоветоваться. Там вам расскажут о нужности вашей идеи, а также приведут десяток модулей, которые делают то, что

вы хотели :)

С чего начинаются модули

Вы всё ещё это читаете, а значит нужно сделать что-то новое!

Если спросить у программиста: “Что самое сложное в твоей работе?”, можно услышать: “Придумать название переменной” в ответ. Действительно, задача нелёгкая — описать всю глубину происходящего в паре слов. Причём так, чтобы окружающие вас поняли. Так что `${"капец"}` и `${"AAAAA!!!"}` — не подойдут. Что уж и говорить о названии модуля — там-то происходит ещё больше всякого!

Что ж, здесь нам может помочь опыт на-

ших собратьев: посмотрите, как названы модули, которые вы используете. Также посмотрите на то, для чего вы их используете. Если эти две величины не сходятся, то либо модуль назван криво, либо вы его не тем местом применяете…

К примеру, модуль `Test::WWW::Selenium` имеет довольно говорящее название. А вот имя модуля `Mojolicious` не наводит на мысль, о чём он.

Думаю, общий принцип именования ясен. Но! Есть же ещё и общепринятые договорённости. К примеру, `Test::WWW::Selenium` мог быть и `WWW::Test::Selenium`, и даже `Selenium::Test`. Почему же он назван так? Вероятно, потому, что есть уже большое количество модулей с именем `Test::*`, что намекает на принадлежность к инструментам тестирования.

В общем, тут столько негласных правил, ещё больше исключений. Поэтому — милости просим на prepan.org, дабы во всём разобраться по ходу дела.

Страна должна знать своих героев

Если я не успел у вас отбить желание программировать и публиковать свои творения, вам стоит познакомиться с сервисом PAUSe (The [Perl programming] Authors Upload Server). Тут-то и происходит публикация модулей на CPAN (The Comprehensive Perl Archive Network).

Первым делом надо завести учётку (Request PAUSE account). В принципе, тут всё просто и незатейливо — пока скан паспорта, тьфу-тьфу, не просят. Заранее проверьте, чтобы

Desired ID, который вы запрашиваете, ещё не существует. Сделать это просто: `http://search.cpan.org/~$desired_id/` не должно быть :)

Раньше на PAUSe был ещё раздел, в котором запрашивались имена модулей (к примеру, очередной принципиально новый веб-фреймворк должен называться нескучно!), но ныне я этого пункта в меню не вижу. Впрочем, и раньше было легко получить желаемое название — достаточно было сформулировать свою идею.

Show me the code

Каждая хозяйка готовит борщ по-своему, так и каждый программист по-своему собирает пакеты. И хотя результат схож, могут

быть нюансы. Для начала нам понадобится работающий модуль, структура которого приблизительно такова:

```
.
├── Build.PL
├── CHANGES
├── example
│   └── small.pl
├── lib
│   └── MyModule.pm
├── LICENSE
├── README
└── t
    ├── 00-test_some_aspect.t
    ├── ...
    └── 99-last_test.t
```

Первый файл — `Build.PL` — как раз тот, что мы должны написать для сборки. Он нам нагенерит кучу прочих файлов, которые должны быть в пакете. Пример:

```
#!/usr/bin/env perl
use strict;
use warnings;
use Module::Build;

Module::Build->new(
  module_name          => '
     MyModule',
  dist_abstract        => 'Мой
     модуль для решения проблем.
     Возможно, и ваших',
  license              => 'perl',
  dist_author          => 'Моё имя
      <крутой_ник@cpan.org>',
  dist_version_from    => 'lib/
     MyModule.pm',
  build_requires       => { 'Test
     ::More' => 0, },
  configure_requires   => { '
     Module::Build' => '0.40', },
  requires => {
    'perl'
```

```
        => 5.008,
    'Ещё::Один::Модуль::
        Зависимости' => 0,
  },
  meta_merge => {
    resources => {
      repository => 'https://
        github.com/реп_модуля',
    },
    keywords => [ qw/Ключевые
      слова а-ля SEO/ ],
  },
  add_to_cleanup     => [],
  create_makefile_pl => '
    traditional',
)->create_build_script();
```

В файле CHANGES можно увидеть что-то подобное:

```
Revision history for MyModule

0.02  22-10-2015
```

```
  — Fix everything

0.01  21—10—2015
  — Add everything
```

Со стороны может показаться, что это атавизм. Но я с удивлением обнаружил, что его читают. Однажды хотел пропиарить свой модуль `Pony::Object` — мол вышла новая версия, объекты теперь ещё более объектные. На что услышал, что мол и без вас знаем (собеседник не знал о моей роли в проекте :)).

Следующая на очереди — папка `examples`. По-хорошему, этой папки быть не должно. Если у вас понятно написаны доки, а тесты хорошо оформлены, то необходимость в `examples` отсутствует. Иначе — приводите там скрипты с использованием вашего модуля.

`lib` — душа и сердце вашего модуля — именно там расположен код, который вы публикуете.

`LICENSE` — правила использования вашего кода (обычно используется “The Artistic License”).

`README` — руководство. Предлагаю генерировать (pod2text) из POD-документации вашего модуля.

`t` — тесты. Вы ведь пишете тесты, правда?

Стоит также упомянуть, что нужно указывать версию модуля в файле, куда указывает `dist_version_from` файла `Build.PL`:

```
1 our $VERSION = "0.01";
```

Кстати, нумерация может быть произвольной. Но я бы рекомендовал, чтобы номер

версии увеличивался от релиза к релизу (*увеличение версии обязательно* — прим. ред.).

Мистер Сулу, установите курс, скорость варп два!

Что же, вот и готов наш модуль. Осталось только выполнить пару команд Build.

```
perl Build.PL
```

- сгенерирует исполняемый файл `Build`.

  ./Build help

- покажет доступные команды.

  ./Build distmeta # чтобы были META файлы как у “пацанов” ./Build disttest

чтобы не облажаться ./Build manifest
генерирует файл MANIFEST со списком файлов дистрибутива ./Build dist

- теперь мы имеем файл `MyModule-0.01.tar.gz`. Его следует загрузить на PAUSe (Upload a file to CPAN).

Совсем скоро ваш модуль появится на CPAN (дайте серверу немного времени, чтобы создать веб-страничку и добавить модуль в индекс), и вы будете просто обязаны рассказать о нём всему миру! Ведь если вы написали модуль, а он не решает проблем человечества, то всё зря…

P.S.

Теперь, когда ваш модуль используют люди по всему миру, вам будут слать pull-реквесты, предлагать фичи. Придётся поддерживать и дорабатывать своё детище. В общем, если всё сделано правильно — работы вам только прибавилось.

Так стоит ли писать и публиковать модули на CPAN? “Если можешь не писать — не пиши”, — как сказал то ли Чехов, то ли Толстой, а иной раз эту цитату приписывают и Хемингуэю. В общем, вы поняли.

■ *Георгий Бажуков*

4 Развертывание Perl приложений при помощи Docker

Плюсы и минусы такой инфраструктуры

Вы все еще боитесь деплоить по пятницам? За предложение поставить компилятор на продакшен сервер сисадмин 2 часа бегал за вами с топором? Обновление библиотек ломает все приложения которые только может сломать и вдобавок сжигает блок притания сервера? Эти и другие проблемы нам поможет решить `Docker` (а так же создаст некоторое кол-во новых).

Краткий ликбез.

Q: что такое `Docker`?

A: Docker это инструмент для контейнерной виртуализации. Представляет собой обертку (`libcontainer`) вокруг технологий изоляции приложений предоставляемых LXC.

Q: Какая от него польза?

A: Он инкапсулирует зависимости и настройки приложения внутри контейнера. Грубо говоря, проблема поддержания окружения нужного для работы приложения инкапсулируется внутрь контейнера и сисадмину больше не приходится придумывать как заставить работать на одном сервере 2 приложения которые требуют несовместимых версий одной библиотеки. В идеальном случае ему нужен только сервер на котором стоит демон `Docker`.

Q: Чем это лучше виртуальной машины?

A: Меньше сложность, в общем случае вам не надо настраивать ничего кроме вашего приложения. Ниже оверхед производительности и использования ОЗУ и диска хоста.

Q: Это можно использовать для обеспечения безопасности хоста?

A: Краткий ответ — нет, в общем случае `Docker` ухудшает параметры безопасности хост-машины. Длинный ответ.

Приступим к внедрению

В официальном репозитории `Docker` есть готовые образы perl, но там есть только последние стабильные выпуски (5.20 и 5.22). Соответственно, если нужна другая версия или специфичные флаги компиляции —

придется собирать образ самому. Чем мы и займемся.

Установим Docker

Для начала надо поставить `Docker`. В репозитории `Ubuntu 14.04` (которым я пользуюсь дома) версия `Docker` безнадежно древняя (1.0.1) и кривая. Пользователям `CentOS 6/7` повезло больше, в `EPEL` версия 1.7, что практически близко к текущей 1.8. Так что для установки воспользуемся скриптом с официального сайта:

```
1 wget —qO— https://get.docker.com/
      | sh
```

Да, не делайте так ни на чем кроме тестовой виртуалки, истинные параноики и те кто уже делают бэкапы не одобряют. Хоть

эта команда и дана в официальной доке, но она легко может скомпрометировать вашу машину при определенных условиях.

Но для опытов сойдет, а для продакшена есть мануал по установке куда угодно — http://docs.docker.com/installation.

Тем временем у нас уже установился `Docker`, проверим:

```
root@docker:~# docker --version
Docker version 1.8.3, build f4bf5c7
```

По умолчанию `Docker` работает от `root`, для использования в продакшене вариант спорный, но пока мы на этом заострять внимание не будем (у нас же пока тестовая виртуалка).

Проверим что все установилось корректно

и даже делает вид что работает:

```
root@docker:~# docker run hello-world
```

Если все в порядке, то мы увидим много букв вида:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b901d36b6f2f: Pull complete
0a6ba66e537a: Pull complete
Digest: sha256:517f03be3f8169d84711c9ffb2b3235a4d27

Status: Downloaded newer image for hello-world:latest
WARNING: Your kernel does not support memory swappiness capabilities, memory swappiness discarded.
```

`Docker` попытался запустить контейнер `hello-world`, не нашел его локально, скачал из `registry` и запустил. Warning вылез т.к. у меня на машине используется Debian 8, а ядром 3.16 поддерживаются не все функциональные возможности. Для совсем полной поддержки требуется ядро из 4-й ветки.

Далее мы увидим текст вида:

```
1 Hello from Docker.
2 This message shows that your
     installation appears to be
     working correctly.
3 ....
```

Поздравляю, `Docker` установился и работает.

Теперь пришло время создать наш собственный образ.

Создадим директорию для этих целей:

```
mkdir docker-perl
cd docker-perl
```

Создадим `Dockerfile`:

```
FROM ubuntu:14.04
```

Можно собрать образ и посмотреть как это бывает:

```
root@docker:~/docker-perl# docker
    build -t docker-perl .
Sending build context to Docker
   daemon 2.048 kB
Step 0 : FROM debian:jessie
jessie: Pulling from library/
   debian
d0ca40da9e35: Pull complete
```

```
d1f66aef36c9: Pull complete
Digest: sha256:1179
    b696ceb85111a6928ba699d38a56a1f5
Status: Downloaded newer image
    for debian:jessie
 ---> d1f66aef36c9
Removing intermediate container
    d0ca40da9e35
Successfully built d7b316059744
```

Ура, мы собрали первую часть базового образа `Perl`, время создать образ с нужной нам версией.

Для установки `Perl` в контейнер есть 2 варианта:

Первый — собрать его локально и сунуть в контейнер. Плюсы этого варианта — нам не требуется в контейнере ничего лишнего, что достаточно актуальная проблема. Т.к.

после установки пакетов для сборки наш контейнер потолстел со 125мб, до 313мб:

```
1       root@docker:~/docker-perl#
           docker images
2       REPOSITORY                TAG
                                 IMAGE ID
                        CREATED
                         VIRTUAL SIZE
3       docker-perl               latest
                               d7b316059744
                     8 minutes ago
                    313 MB
4       debian                    jessie
                               d1f66aef36c9
                     5 days ago
                       125.1 MB
```

Минус — все пакеты придется собирать на хосте, что не имеет особого смысла если вы не собираетесь все это применять в промышленных масштабах.

Второй — собрать руками в контейнере нужную версию. Вариант для простых людей которым нужно чтобы просто работало.

Есть еще третий вариант — установить через `perlbrew` в контейнере. Но для контейнера это самый сложный вариант с наибольшим кол-вом граблей по пути и неясным профитом. Я пробовал — если расписать весь объем проблем, то статья выйдет побольше этой.

Установка Perl в базовую систему и перенос в контейнер

Итак, рассмотрим первый вариант. Я буду делать не очень канонично — просто установлю `Perl` с помощью `perlbrew` в то же место на ФС где он и будет лежать в обра-

зе. Правильнее будет настроить chroot и установить туда или установить в нужный префикс, а потом перенести в контейнер. Но глобальных отличий между методами не будет, а люди которые собирают кастомный Perl настроят chroot без проблем, ну мне так кажется.

Устанавливаем `perl` в `/perl5`:

```
root@docker:~# export PERLBREW_ROOT=/perl5
root@docker:~# perlbrew init
root@docker:~# source /perl5/etc/bashrc
root@docker:~# perlbrew install perl-5.18.4
Fetching perl 5.18.4 as /perl5/dists/perl-5.18.4.tar.bz2
Download http://www.cpan.org/src/5.0/perl-5.18.4.tar.bz2 to /perl5/dists/perl-5.18.4.tar.bz2
```

```
7 Installing /perl5/build/perl
    -5.18.4 into /perl5/perls/perl
    -5.18.4
8 ...
```

Копируем установленный Perl в каталог сборки:

```
1 root@docker:~/docker-perl# cp -r
    /perl5 .
```

И приведем Dockerfile к виду:

```
1 FROM ubuntu:14.04
2 COPY perl5 /perl5
3 ENV PATH /perl5/perls/perl
    -5.18.4/bin:$PATH
```

Зачем копировать Perl в рабочую папку? При запуске docker build, грубо говоря, создается chroot в текущей папке сборки. И все что находится вне ее становится недо-

ступно. Проблема в том, что символические ссылки при этом перестают работать.

Если сделать так:

```
root@docker:~/docker-perl# ln -s /perl5 perl5
```

То при сборке образа мы получим сообщение об ошибке:

```
root@docker:~/docker-perl# docker build -t docker-perl .
Sending build context to Docker daemon  2.56 kB
Step 0 : FROM ubuntu:14.04
 ---> 1d073211c498
Step 1 : COPY perl5 /
Forbidden path outside the build context: perl5 (/perl5)
```

Теоретически можно использовать hard-link, но в Ubuntu хардлинки на директории

запрещены в настройках, а менять я считаю не очень целесообразным.

Итак, скопировали `Perl`, запускаем сборку:

```
root@docker:~/docker-perl# docker build -t docker-perl .
Sending build context to Docker daemon 250.5 MB
Step 0 : FROM ubuntu:14.04
 ---> 1d073211c498
Step 1 : COPY perl5 /perl5
 ---> 6039c2920dcf
Removing intermediate container ef8bbc7532d2
Step 2 : ENV PATH /perl5/perls/perl-5.18.4/bin:$PATH
 ---> Running in 51a9fe8521b0
 ---> 77e9682a6241
Removing intermediate container 51a9fe8521b0
```

Проверяем:

```
root@docker:~/docker-perl# docker run -it docker-perl bash
root@98d5a2b98558:/#
```

Приглашение тонко намекает нам что мы находимся в более другом шелле чем до запуска контейнера, а именно в шелле контейнера.

Проверяем что Perl установлен корректно и нужной нам версии:

```
root@8ea3d44d36c0:/# perl -v

This is perl 5, version 18, subversion 4 (v5.18.4) built for x86_64-linux
...
```

Если вы увидели что это приглашение — то все в порядке и можно собирать контейнер дальше.

Запускать мы будем классическое Enterprise приложение “hello world”

Для этого создадим еще один контейнер дабы не пересобирать каждый раз основной, т.к. это долго и приводит к замусориванию системы контейнерами. Т.к. на каждую сборку Docker создает промежуточный контейнер и не удаляет его автоматически.

```
root@docker:~# mkdir docker-app
root@docker:~# cd docker-app
root@docker:~/docker-app# cp ../
    docker-perl/Dockerfile .
root@docker:~/docker-app# vim app
    .pl
```

Содержимое `app.pl`

```
#!/usr/bin/env perl
print "Hello world from container
    !\n";
```

Выдадим ему права на запуск, затем создадим `Dockerfile`:

```
root@docker:~/docker-app# chmod +x app.pl
root@docker:~/docker-app# vim Dockerfile
```

Содержимое `Dockerfile`:

```
FROM docker-perl
COPY app.pl /app/
WORKDIR /app
CMD /app/app.pl
```

И соберем контейнер:

```
root@docker:~/docker-app# docker build -t perl-app .
Sending build context to Docker daemon 3.072 kB
Step 0 : FROM docker-perl
 ---> 77e9682a6241
Step 1 : COPY app.pl /app/
```

```
 ---> Using cache
 ---> 045ca8292484
Step 2 : WORKDIR /app
 ---> Using cache
 ---> 50b52640e8d1
Step 3 : CMD /app/app.pl
 ---> Running in 1ef85c52970b
 ---> fe01b30beff4
Removing intermediate container 1
    ef85c52970b
Successfully built fe01b30beff4
```

И запускаем его:

```
root@docker:~/docker-app# docker
    run perl-app
Hello from container!
```

Поздравляю, вы только что собрали и запустили контейнер с perl-приложением.

Что делать если приложению нужны мо-

дули устанавливаемые в систему? Краткий ответ — страдать. Длинный ответ — модули надо установить в базовую систему и обновить образ docker-perl от которого унаследован наш образ, затем обновить образ с приложением. Т.е. каждый раз повторять все что мы проделали выше в немного сокращенном виде.

Так заморачиваться имеет смысл, если стоят повышенные требования к составу ПО в образе или из любви к прекрасному (но тогда надо собирать deb-пакеты, чтобы уж совсем прекрасно было). Если их нет — то все можно сделать проще — собирать зависимости прямо в контейнере.

Устанавливаем Perl в контейнере

Модифицируем наш первоначальный образ:

Во первых — установим мета-пакет для сборки. Во вторых — установим нужный `Perl` прямо в контейнер.

Приведем `Dockerfile` к виду:

```
1 FROM ubuntu:14.04
2
3 RUN apt-get -y update && apt-get
    install -y build-essential
4
5 ADD http://www.cpan.org/src/5.0/
    perl-5.18.4.tar.gz /tmp/
6 RUN cd /tmp && tar -zxf /tmp/perl
    -5.18.4.tar.gz
7 WORKDIR /tmp/perl-5.18.4
8
```

```
9  RUN ./Configure -des -Dprefix=/
       perl5 && make && make test &&
       make install
10 ENV PATH /perl5/bin:$PATH
11
12 WORKDIR /
13 RUN rm -rf /tmp/perl-5.18.4.tar.
       gz /tmp/perl-5.18.4
```

Что здесь происходит?

```
1  RUN apt-get -y update && apt-get
       install -y build-essential
```

Эта команда запустит apt который обновит дерево пакетов и установит метапакет с компиляторами и заголовочными файлами.

```
1  ADD http://www.cpan.org/src/5.0/
       perl-5.18.4.tar.gz /tmp/
```

Скачает с указанного адреса `Perl` и положит в `/tmp/`. Если у вас не очень быстрый интернет — то можно скачать дистрибутив вручную и добавлять его из папки где происходит сборка. Тогда будет не нужна следующая операция распаковки архива, т.к. `ADD` при работе с локальными архивами распаковывает их в целевую директорию. Но на моем интернете получается быстрее скачать, чем аплоадить увеличившийся `build-context` в docker-демон.

Дальше все стандартно — устанавливаем рабочую директорию в директрорию где распаковыны сырцы и собираем `Perl` в `/perl5`, после чего выставляем окружение и чистим за собой.

Теперь пришло время создать контейнер для нашего enterprise-приложения. Переходим в директорию где оно собирается и

модифицируем app.pl:

```
#!/usr/bin/env perl
use Dancer2;

get '/' => sub {
  return 'Hello World!';
};

start;
```

Теперь правим Dockerfile т.к. нам надо установить зависимости приложения.

```
FROM docker-perl
COPY app.pl /app/
WORKDIR /app

RUN cpan -i Dancer2
ENTRYPOINT /app/app.pl
```

Собираем:

```
root@docker:~/docker-app# docker build -t perl-app .
Sending build context to Docker daemon 3.072 kB
Step 0 : FROM docker-perl
 ---> 987aba3529c5
Step 1 : COPY app.pl /app/
 ---> 134e2e82fb74
Removing intermediate container 47436b48e7d5
Step 2 : WORKDIR /app
 ---> Running in 998eb9641a97
 ---> eb3bfc1ccc34
Removing intermediate container 998eb9641a97
Step 3 : RUN cpan -i Dancer2
 ---> Running in 6609cd09aca6
...
```

Пока ставятся зависимости к Dancer2 можно сходить попить чайку.

Запускаем:

```
root@docker:~/docker-app# docker
    run -td -p 3000:3000 perl-app
4548826053
    f5500203f7b516040b34dfab9e246086
```

Контейнер запустился и выдал свой ID. Теперь если зайти на 3000 порт хост-машины мы увидим приветствие от нашего приложения.

Можно посмотреть список запущенных контейнеров:

```
root@docker:~/docker-perl# docker
    ps
CONTAINER ID        IMAGE
                    COMMAND
                  CREATED
                  STATUS
                   PORTS
```

```
                                    NAMES
4548826053f5        perl-app
                "/app/app.pl"
          4 seconds ago       Up 3
    seconds
    0.0.0.0:3000->3000/tcp
    gloomy_turing
```

Ну и остановить контейнер:

```
root@docker:~/docker-perl# docker
    stop 4548826053f5
```

Контейнеру сначала будет послан сигнал `SIGTERM`, а через 15 секунд `SIGKILL`, так что если вашему приложению нужно время на завершение — учитывайте это.

Теперь у вас есть рабочий контейнер с Perl и вашим приложением, можно дальше углубиться в волшебный мир DevOps с его контейнерами, облачными сервера-

ми и прочими серебрянными пулями в промышленных масштабах ;)

А теперь о самом веселом — о граблях!

Что не может не радовать — тут практически нет `Perl`-специфичных граблей, со всеми ними вы столкнетесь при сборке любого контейнера.

Изначальная грабля из мануала докера, запускаем контейнер так:

```
docker run -it -p 3000:3000 perl-app
```

И все, терминал потерян. Контейнер с этого терминала не остановить, после остановки контейнера его корежит неимоверно, при-

ходится сбрасывать через reset. Бага из серии первого входа в vim, ничего критичного, но нериятно.

Дальше, т.к. контейнер это минимальное окружение, то этого самого окружения там реально по минимуму.

```
root@docker:~/docker-app# docker run -it --rm docker-perl bash
root@0fe4d45827d0:/tmp/perl-5.18.4# env
HOSTNAME=0fe4d45827d0
TERM=xterm
LS_COLORS= ...
PATH=/perl5/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/tmp/perl-5.18.4
SHLVL=1
HOME=/root
LESSOPEN=| /usr/bin/lesspipe %s
LESSCLOSE=/usr/bin/lesspipe %s %s
```

```
12 _=/usr/bin/env
```

Если ваше приложение рассчитывает на какие-то переменные окружения, то за этим надо следить и не рассчитывать на то что они выставлены по умолчанию как в настоящих операционках. Особенно весело становится когда используется большое кол-во зависимостей и приложение только что работавшее на хосте падает в контейнере.

Еще небольшая грабелька — `Docker` активно использует кэширование операций. Что это значит? Рассмотрим наш `Dockerfile` от базового образа:

```
1 FROM ubuntu:14.04
2
3 RUN apt-get -y update && apt-get
    install -y build-essential
4
```

```
ADD http://www.cpan.org/src/5.0/perl-5.18.4.tar.gz /tmp/
RUN cd /tmp && tar -zxf /tmp/perl-5.18.4.tar.gz
WORKDIR /tmp/perl-5.18.4

RUN ./Configure -des -Dprefix=/perl5 && make && make test && make install
ENV PATH /perl5/bin:$PATH
```

Допустим, нам надо установить что-то из пакетов, пусть это будут заголовочные файлы psql. Мы можем сделать так:

```
RUN apt-get -y update && apt-get install -y build-essential libpq-dev
```

К чему это приведет? Это приведет к инвалидации ВСЕХ кэшей операций после этой операции. Т.е. в данном случае у

нас будет запущена сборка образа с нуля. Это правильный вариант для подготовки финального продакшен образа, но раздражающий при активной разработке приложения. Так что в данном случае правильнее сделать так:

```
....
ENV PATH /perl5/bin:$PATH

RUN apt-get install -y libpq-dev
```

Тогда мы используем кэши предыдущих операций и быстро доустанавливаем все что нам надо.

Самая жесткая фича — по умолчанию докер демон запускается от рута и пользователь внутри контейнера тоже root, а теперь делаем так:

```
VOLUME ["/"]
CMD rm -rf /* && echo Ooops!
```

Пример немножко надуманный, но идея понятна. Docker это не средство изоляции хоста от гостевой системы, так что следует быть внимательным при работе с внешними разделами и образами из недоверенных источников.

Далее, Docker склонен к замусориванию. После написания этой статьи его состояние примерно таково:

```
root@docker:~/docker-app# docker images
REPOSITORY    TAG       IMAGE ID          CREATED          VIRTUAL SIZE
perl-app      latest    7                 ff215a87f66   31 minutes ago   610.5 MB
docker-perl   latest    987               aba3529c5   About an hour ago   541.4 MB
<none>        <none>
```

```
    e0fdf25a2c4d  10 hours ago
          524.1 MB
<none>          <none>   33
    a0c1f4d5da  10 hours ago
          524.1 MB
<none>          <none>
    aca6ab415167  10 hours ago
          389.7 MB
<none>          <none>   38
    e530df4806  10 hours ago
          558.6 MB
<none>          <none>
    f7f8d000dec5  11 hours ago
          428 MB
<none>          <none>
    fe01b30beff4  11 hours ago
          428 MB
<none>          <none>
    f24de2f3ca26  11 hours ago
          428 MB
<none>          <none>   5
    b24e9b06ca2  12 hours ago
          428 MB
```

```
<none>              <none>   66
    ff9778c629  12 hours ago
          428 MB
<none>              <none>   78
    f405da27fc  12 hours ago
          428 MB
ubuntu              14.04    1
    d073211c498  7 days ago
            187.9 MB
hello-world   latest   0
    a6ba66e537a  2 weeks ago
          960 B

root@docker:~/docker-app# du -sh
    /var/lib/docker/*
2.2G    /var/lib/docker/aufs
836K    /var/lib/docker/
    containers
6.0M    /var/lib/docker/graph
16K    /var/lib/docker/linkgraph.
    db
4.0K    /var/lib/docker/
    repositories-aufs
```

```
4.0K    /var/lib/docker/tmp
4.0K    /var/lib/docker/trust
4.0K    /var/lib/docker/volumes
```

2.2Gb после десятка пересборок образов, неплохо.

■ *Денис Федосеев*

5 Обзор CPAN за сентябрь 2015 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц

Статистика

- Новых дистрибутивов — 155
- Новых выпусков — 920

Новые модули

Net::PhotoBackup::Server

`Net::PhotoBackup::Server` — это простая реализация сервера для бэкапа фото-

графий проекта PhotoBackup. Суть проекта в создании «домашнего» облака, т.е. сервера под вашим контролем, на котором располагается веб-приложение, принимающее для загрузки фотографии с ваших мобильных устройств. Существует также клиентское приложение для android с открытым исходным кодом, которая обеспечивает фоновое сохранение фотографий на ваш сервер. К недостаткам можно отнести то, что сервер пока не работает по https.

LibJIT

LibJIT — это обвязка к одноимённой библиотеке. Модуль позволяет создавать различные низкоуровневые функции во время исполнения программы, используя api библиотеки libjit.

DBI::Log

Модуль `DBI::Log` позволяет вести лог всех SQL-запросов, которые отправляются через интерфейс DBI. SQL-запросы вместе с данными выводятся на стандартный вывод ошибок, или, если определена переменная `$DBI::Log::path`, записываются в файл.

Image::JpegMinimal

`Image::JpegMinimal` позволяет создавать коллекцию миниатюр для набора изображений, при этом раздельно сохраняет jpeg-заголовок, который общий у всех изображений, и данные сильно сжатого изображения. После чего эти данные могут быть встроены в html-страницу, в которой небольшой js-код заменяет ещё

незагруженные полноразмерные изображения встроенными увеличенными миниатюрами с эффектом размывания. Таким образом, можно увеличить первоначальную скорость загрузки сайта, которую наблюдает пользователь, что особенно актуально в случае медленных мобильных подключений, при этом полноразмерные изображения могут догружаться в фоне и постепенно заменять миниатюры.

MOP4Import

`MOP4Import` — это метаобъектный протокол для создания расширяемых Exporter-подобных классов. Если вам не требуется писать свою собственную реализацию Exporter, но интересно воспользоваться новым ООП-фреймворрком, то более полезными могут стать `MOP4Import::`

Types и MOP4Import::Base::Configure. Например,

```
1 package My::Pkg {
2     use MOP4Import::Base::
          Configure -as_base, [
          fields => qw /app data/];
3 }
4
5 my $pkg = My::Pkg->new( app => '
    value' );
6 print $pkg->{app};  # 'value'
7 print $pkg->app;    # 'value'
8 print $pkg->{app1}; # ошибка --
    поле не задано
```

Как видно в MOP4Import предпринята попытка реинкарнации использования прагмы fields, модуль даже содержит детальное предложение о современной практике использования этой прагмы.

Try::Tiny::Warnings

`Try::Tiny::Warnings` — это небольшое расширение для `Try::Tiny`, которое позволяет отлавливать не только фатальные ошибки, но и предупреждения.

```
try_warnings {
    warn "Первое предупреждение!"
    warn "Следующее";
}
catch {
    print "Этот блок не будет
       исполнен, т.к. не было
       ошибок\n";
}
catch_warnings {
    print "Предупреждение: $_"
       for @_;
};
```

Image::Libpuzzle

Модуль `Image::Libpuzzle` — это обвязка к C-библиотеке libpuzzle. Модуль позволяет быстро определять являются ли два изображениями схожими, даже если одно было изменено в размере, пережато, изменены оттенки или добавлены незначительные изменения.

Mercury

`Mercury` — это брокер сообщений для WebSockets. Mercury может быть использован в случае, если используется веб-приложение, которое использует многопроцессную модель с несколькими рабочими процессами (воркерами), позволяя организовать пересылку сообщений

между процессами. Таким образом, может быть организована и коммуникация по протоколу WebSocket между клиентами, даже если они подключены к различным серверным воркерам. Также присутствует поддержка широковещательной шины сообщений и модели pub/sub.

Обновлённые модули

IO::Socket::SSL 2.020

Новая версия модуля `IO::Socket::SSL` теперь поддерживает указание нескольких каталогов для параметра `SSL_ca_path`, который указывает где должны будет производится поиск сертификатов доверенных центров.

RPerl 1.100002

В сентябре вышел очередной значимый релиз альтернативного компилятора RPerl, транслирующий Perl-код в C++ и собирающий его в виде загружаемой xs-библиотеки. Как утверждает автор, в данной версии RPerl удалось собрать приложение-симмулятор солнечной системы N-Body, которое используется в системе бенчмарков Alioth Benchmark Game для ранжирования языков программирования по скорости. RPerl позволил запустить выполнение N-body со скоростью сравнимой с C++, уменьшив время исполнения с 19 минут до 13 секунд.

Term::ReadLine::Gnu 1.28

В новой версии `Term::ReadLine::Gnu` добавлена поддержка альфа-версии библиотеки Gnu Readline 7.0.

perl 5.20.3/5.23.3

В сентябре вышли два релиза perl: 5.20.3 — баг-фикс релиз предыдущей стабильной версии Perl и новый релиз для разработчиков 5.23.3.

В релиз 5.20.3 вошло несколько исправлений, включая поддержку сборки с помощью GCC 5, исправление краха при обработке `eval { LABEL: }`, исправление проблемы при использовании UTF-8 имён переменных в индексах массивов

и для ограничителей встроенных документов, исправление фатальной ошибки, возникающей при попытке отлаживать Perl-программу, загружающую модуль `threads/threads::shared`.

Релиз 5.23.3 теперь поддерживает использование расширенных классов символов `qr/(?[ ])/` в UTF-8 локалях, появилась поддержка платформы AmigaOS, исправлены несколько ошибок, например крах при обработке выражения `BEGIN <>`.

Mail::GPG v1.0.10

Обновлён модуль `Mail::GnuPG` для работы с зашифрованными и заверенными цифровой подписью почтовыми сообщениями. В новой версии исправлена потенциальная ошибка безопасности при

декодировании почтовых адресов: ранее для конвертирования из бинарного формата к строке использовалась функция `Encode::_utf8_on`, т.е. просто устанавливался внутренний флаг, что строка в формате UTF-8, без какой-либо проверки на валидность кодовых последовательностей.

MongoDB 1.0.1

Вышел первый мажорный релиз официального драйвера MongoDB для Perl. Релиз содержит внушительное число изменений, в том числе несовместимых. Пользователям настойчиво рекомендуется ознакомится с документом MongoDB::Upgrading, чтобы детально ознакомиться с подробностями изменений. Из заметных изменений:

- Не требуется компилятор, используются модули `IO::Socket::SSL` для `Authen::SASL` для поддержки SSL и SASL,
- Замена Moose на Moo,
- Использование исключений для индикации ошибок при вызове методов
- Удалено большинство низко-уровневых функций

Image::PNG::Libpng 0.41

`Image::PNG::Libpng` — это обвязка к C-библиотеке libpng для работы с изображениями в формате PNG. В новой версии исправлена ошибка с использованием памяти, после её освобождения, что могло приводить к краху или повреждению данных.

Compress::Raw::Zlib 2.0.69

Обновлён модуль `Compress::Raw::Zlib` — низкоуровневый интерфейс к библиотеке сжатия данных zlib. В новой версии исправлено несколько ошибок, в том числе потенциальных ошибок в безопасности: переполнение буфера при копировании. Данные ошибки были найдены сканером безопасности Coverity.

DateTime::Locale 0.92

В сентябре вышло существенное обновление модуля `DateTime::Locale` — модуля поддержки локали для DateTime. Релиз основан на базе CLDR v28, до этого использовалась база от 2009 года, поэтому модуль претерпел существенные изменения в том

числе и несовместимые. Если вы использовали этот модуль опосредовано через `DataTime`, то эти изменения скорее всего пройдут незаметно.

■ *Владимир Леттиев*

6 Интервью с Анатолием Шарифулиным

Анатолий Шарифулин (sharifulin) — Perl-программист, харизматичный докладчик, предприниматель

Как и когда научился программировать?

Это хороший вопрос, на который у меня есть смешной ответ. В 8 классе, как у всех в школе, была информатика, и нас пытались научить программировать на Бейсике. Только почему-то мы полгода рисовали то круги, то линии, то арбуз с семечками (SCREEN 12 — до сих пор помню). А потом на контрольной нам дали задачу посчитать сумму массива из 10 элементов (так сказать, внезапно). Я, не долго думая, написал “A[1] + A[2] + A[3] + A[4] + A[5]

+ A[6] + A[7] + A[8] + A[9] + A[10]”, за что получил три балла, т. к. сумму из тысячи элементов я таким образом не посчитаю, а я же мог :)

Перед первым курсом я решил освоить Паскаль, всё лето я учил его, освоил циклы, условия и основные алгоритмы, финалом моего обучения была программа, которая запускала точку внутри квадрата и она хаотично двигалась, отталкиваясь от стен квадрата, угол падения = углу отражения :)

Какой редактор используешь?

Раньше я использовал SciTe на Windows, потом, перейдя на Mac, использовал TextMate, Sublime 2, Atom и снова вернулся на TextMate. vim иногда вспоминаю, правя crontab.

Как и когда познакомился с Perl?

С Perl я познакомился на втором курсе, когда понял, что “кайф делать сайты с гостевой книгой”, сайты я уже верстал, а вот гостевую книгу все никак не мог сделать. Купил “ламма-бук”, и всё началось. Но первой настоящей работой была система тестирования студентов, это было тестовое задание для одного научного руководителя, на кафедру которого я хотел попасть. После этого я на 3-и, 4-м и 5-м курсах делал курсовые и дипломные работы на Perl. В голове была такая схема: сейчас я научусь делать сайты на Perl с БД и буду получать много денег, очень много :) На 5-м курсе я уже пошёл работать перловиком в Точка Кипения, где и проработал более 5 лет. Так хобби (я до последнего считал, что Perl — это хобби) превратилось в настоящую работу.

С какими другими языками интересно работать?

Учитывая, что в данный момент я не позиционную себя программистом, а программирование снова стало хобби, то можно сказать, что я “замерз” на уровне 2012 года, уйдя с головой в mobile, которым и по сей день занимаюсь. Но до сих пор интересно работать с Perl. Я уже не программирую в чистом виде, не создаю идеальный и, к сожалению, никому не нужный код, я делаю “фичи” и прототипы, которые уже получилось “продать”, автоматизирую ручные процессы, которые достали. Это отличный опыт, и я безумно рад, что умею программировать.

На мой взгляд, в 2015 году любой предприниматель или ко-фаундер в стартапе должен знать основы программирования

и, как минимум, должен уметь верстать. Тогда всё получается в разы быстрее, и не требуется годами разрабатывать то, что УЖЕ никому не нужно.

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

Скорость написания кода и количество кода для того, чтобы сделать ту или иную задачу. Я фанат TIMTOWTDI и лаконичного кода. Меня прёт, когда я могу написать две строчки кода, чтобы построить аналитический или маркетинговый отчёт из сырых данных :)

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

Всё-таки скорость работы, а из этого следует простой синтаксис. И это основной ми-

нус Perl. В глобальном смысле TIMTOWTDI — зло, я это понимаю… и люблю :)

Что думаешь о Perl 6?

Это тлен. Помню 2005 год, когда купил книгу про Perl 6 и Parrot с надеждой, что вот светлое будущее, которое вот-вот случится. Но этого не случилось. Любители писать “идеальный код” до сих пор нам обещают “рождество”, но оно уже не нужно. НЕ НУЖНО!

Часто посещаешь и выступаешь на Perl-конференциях. Чем они тебя привлекают?

Да-а, было дело, всё началось в 2008 году с May Perl. До сих пор помню, как я волновался, выступая на 100+ аудиторию, рассказывая про POE. Первоначальная

цель была узнать, что делают другие, и рассказать, что делаю я. Тем самым можно было расширить свой кругозор и получить отличный фидбек. Плюс внезапно я понял, что мне нравится делать презентации, штудировать материал, которым владею, тем самым затыкая моменты, в которых сомневаюсь, и, чего там скрывать, мне нравится выступать и доказывать себе, что я делаю что-то важное, а не “унылое говно”. Я старался не пропускать конференции и митапы в России и СНГ, а приезжать на конференцию и не выступать — это уныло :) Забегая вперед скажу, что опыт выступлений и созданий презентаций очень сильно мне пригодился, очень сильно :)

Но в 2013 году после воркшопа в Нидерландах я решил взять тайм-аут с Perl-конференциями, т. к. полностью переключился на mobile. Мне хотелось повторить

успех выступлений на Perl-конференций в новой для меня теме. Как вы понимаете, это получилось :)

Твои доклады обычно довольно эмоциональны и зажигательны. В чем секрет?

Когда что-то любишь и делаешь с душой, то аудитория не остается равнодушной, слушая тебя. Им может нравиться или не нравиться то, что я делаю или рассказываю, но точно не всё равно. Мне важно рассказывать всё так, как есть, всё так, как я думаю, и услышать фидбек. Думаю, в этом секрет, поэтому я часто эмоционален во время своих выступлений.

Расскажи про техническую сторону appfollow.ru

Сейчас мы находимся на этапе, когда важ-

но не то, на чём ты это делаешь, а то, что ты делаешь! Поэтому мы взяли рабочую связку технологий (Ubuntu, Perl/Mojolicious, MySQL, Redis, nginx, sphinx, git), которая давно доказала свою состоятельность и работоспособность. За всю техническую часть в AppFollow отвечает технический директор — Павел Власов. Поэтому про архитектуру, парсеры, очереди и т. д. лучше с ним поговорить :)

Кстати, мы с ним познакомились на YAPC::Russia + Perl Mova в Киеве в 2012 году, это было яркое мероприятие (если кто-то там был, то понимает о чём я). Потом мы работали вместе в Островке, выиграли хакатон AngelHack Moscow 2013, а теперь делаем общую компанию — AppFollow.

Нравится ли быть предпринимателем? Чем лучше/хуже работы в чужой компа-

нии?

В целом да, но в первое время финансовая сторона этого вопроса сильно напрягала. Нет больше фиксированного дня, когда ты точно получишь зарплату, из-за этого сложно планировать. Но это заставляет работать больше и быстрее. Сложно было перестроиться, но сейчас уже не думаешь об этом, а думаешь, как же сделать лучше или оптимальнее свою компанию.

Сколько времени проводишь за написанием Perl-кода?

Например, в августе я вернулся в кодинг и программировал 80% времени, у нас был внутренний хакатон и рефакторинг. Сейчас же 20-30%. На мне сайт, админка, прототипы нового функционала, быстрые исследования и мелкие правки шаблонов.

Всё же остальное на Паше.

Сейчас запуск одной новой функции ведёт к кратному увеличению беклога задач, поэтому очень не хватает свободных рук. Надеюсь, намёк понятен, контакты мои остались прежними :)

Чем еще увлекаешься помимо программирования?

Сейчас я больше стал уделять внимание спорту: бег, турник, 7 минут workout и велосипед — это то, что мне нравится, и я стараюсь не забывать об этом. А так — это путешествия, смена обстановки отлично влияет на работоспособность и формирование свежего взгляда на проблему или задачу.

Стоит ли советовать молодым програм-

мистам учить сейчас Perl?

Думаю, что нет. Во-первых, мы в ответственности за тех, кого приручили. Во-вторых, будущее за JavaScript и Swift. А перловики всегда найдут себе работу, по крайней мере, в течение ещё 5+ лет (букинг, привет!).

Вопросы от читателей

Где тебя можно увидеть в ближайшее время?

Очень хочу посетить Saint Perl в 2015 году, если он будет. Поэтому давайте все там увидимся, ну и не только.

До сих пор Mojolicious?

А разве в Perl есть что-то лучше?! :) Если

и есть, то с меня хватит экспериментов, по крайней мере, сейчас.

Сколько у тебя аккаунтов в соцсетях?

Хочется сказать, что все, но это не так. Актуальные есть на моем сайте — sharifulin.ru. Welcome!

■ *Вячеслав Тихановский*