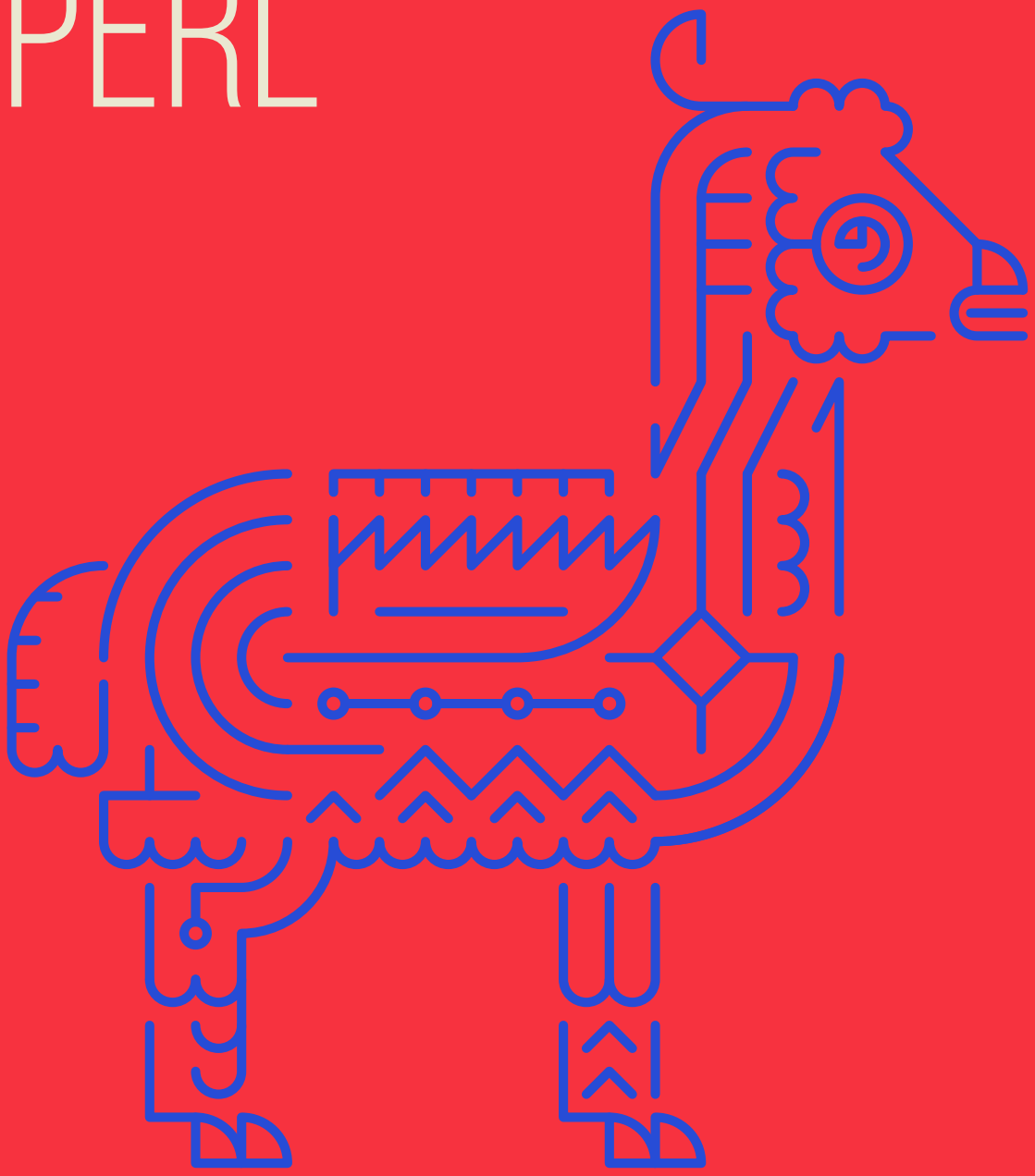
PRAGMATIC PERL

19



09/2014

pragmaticperl.com

Pragmatic Perl 19

pragmaticperl.com

Выпуск 19. Сентябрь 2014

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: Елена Большакова, Владимир Леттиев

Обложка: Марко Иваныk

Корректоры: Георгий Бажуков, Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 16:23

Оглавление

1 От редактора

22-24 августа в Софии прошла YAPC::Europe 2014. К сожалению, никто не прислал отчет в журнал, поэтому несколько ссылок на англоязычные ресурсы: видео докладов, My first YAPC, Xing: YAPC::Europe 2014, Things I learned at YAPC::Europe 2014 in Sofia, acme: Day 1, acme: Day 2, Tobez: Day 1, Tobez: Day 2.

Друзья, журнал остро нуждается в новых авторах. Не упускайте такой возможности! Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2 Постепенная автоматизация рутинных задач

Рассмотрена последовательность автоматизации повторяющихся задач

Бывает так: есть несложная последовательность операций (в командной строке, разумеется), которую приходится время от времени повторять; хорошо бы написать нормальный скрипт, да все руки не доходят, и времени жалко, да и на самом деле не такое уж плевое дело — аккуратно запрограммировать эту рутину, а задача опять должна быть выполнена “вот прям сейчас же”, и ни минуты лишней нет. И повторяются, и повторяются однотипные действия: `mkdir`, `mysql_install_db`, `chown`, `vim my.cnf`, `service start`, `create database`, `grant all privileges` ....

Знакомая картина? Как насчет что-нибудь изменить? Хочу рассказать про один метод как раз для таких случаев.

Итак, автоматизация рутины. На что обычно похожа такая задача?

- задача не очень-то сложная, но какая-то неформализованная;
- в задаче не видно хитрых алгоритмов и структур данных, зато есть внешние программы, интерактивное редактирование конфигов, обращения к другим серверам и т.п.;
- этой разработки нет в квартальных планах вашей организации: либо это ваша личная задача, либо никто не заботится о планировании подобных вещей;
- самый заинтересованный пользо-

ватель — вы; вы сам себе заказчик, product owner, project manager и QA engineer;
- потенциальная аудитория пользователей невелика[1];
- время на разработку *очень* ограничено.

Последнее обстоятельство — ограниченность времени — очень важно. Про опасности прерывистого времени для разработчиков пишут много (комикс, еще примеры см. в конце статьи), но иногда это просто реальность. В таких условиях разработку полезно организовать так, чтобы она состояла из серии маленьких последовательных доработок, каждая из которых занимает... скажем, от 15 минут до часа, не больше.

---

[1]Конечно, бывают исключения, см. описание 7 шага.

Так как главный получатель выгоды — вы сами, в ваших собственных интересах сделать первый прототип очень быстро, хотя бы и в ущерб полной функциональности: первый успех воодушевит на дальнейшие доработки, а хотя бы частичная автоматизация сэкономит вам время.

Сверхбыстрое прототипирование без сложных алгоритмов, вызов внешних программ, ssh, проверки существования файлов и т.п. — это область, в которой силен шелл (shell). Однако большие шелл-программы тяжелы в отладке и доработке, так что в какой-то момент стоит перейти от шелльного прототипа к Perl.

Вот и готов метод «постепенной автоматизации»: быстрый первый прототип на шелле, серия коротких самодостаточных доработок, своевременный переход на Perl.

Пожалуй, это сработает и с другим интерпретируемым языком вроде Ruby или Python, но переход от шелла к универсальному языку самый гладкий именно с Perl, надо пользоваться этим.

В этой статье я опишу пошаговый рецепт постепенной автоматизации и свои соображения «почему это работает именно так». В соседней статье — пример применения этого алгоритма.

Хочу предупредить: в этих шагах нет ничего сверхъестественно сложного или неожиданного, возможно, вы уже давно так и действуете. С другой стороны, еще не все прониклись идеей постепенных улучшений, так что думаю, что лишней статья тоже не будет.

Пошаговый алгоритм автоматизации рутинных операций

Каждый шаг разработки почти ничего не стоит, не требует принятия сложных решений, и при этом делает рутинную задачу ощутимо более легкой по сравнению с предыдущим шагом.

Каждая стадия порождает пригодный к использованию инструмент, при этом важные фичи и свойства реализуются в первую очередь, поэтому в любой момент можно остановиться и получить инструмент как раз той степени проработанности, которая требуется для задачи, и ровно с теми затратами времени и усилий, которых эта задача заслуживает.

Полезно, чтобы между последовательными шагами проходило время, достаточное по крайней мере для нескольких применений вашего нового инструмента. Так вы получите ценный фидбек, столкнетесь с особыми случаями, которые сможете учесть в новых версиях.

И помните: после любого шага можно остановиться — все равно у вас в руках останется пригодный к использованию инструмент.

Итак, поехали. Дано: имеем задачу, которая в принципе решается в командной строке.

Шаг 1. Выполнить все команды вручную

Первым делом убеждаемся, что действительно можем решить задачу вручную:

открываем шелл и выполняем все необходимые команды.

Приобретения первого шага:

- понятно, что задача в самом деле решается, нет крупных препятствий;
- определены все необходимые внешние программы и известно, как устроены их параметры;
- можно научить кого-нибудь другого решать ту же задачу.

Шаг 2. Шелльный однострочник

Небольшая последовательность команд легко склеивается в шелльный однострочник. Выполнение действий одно за другим обеспечивается точкой с запятой `;`, выпол-

нение при условии успеха или неудачи предыдущей команды — `&&` и `||`, также полезны перенаправление вывода, пайпы и фоновое выполнение. Интерактивное редактирование обычно несложно заменяется на `sed -i`, генерация текста по шаблону — `tpage`, повторение однообразных действий реализуется циклами `for` и `while`.

Если у получившегося микроскрипта нет параметров или они устроены очень просто – однострочник можно сделать шелльным алиасом и вызывать по имени.

Приобретения второго шага:

- однострочник легко вытащить из истории команд по `Ctrl-r` и запустить снова (возможно, подправив некоторые параметры);

- атомарность: последовательность действий запускается целиком, нет возможности отвлечься посреди выполнения;
- однострочник легко скопировать и поделиться им с коллегами.

Если последовательность действий оказывается длиннее 5-6 операций или включает в себя сложное ветвление — можно после ручного выполнения сразу перейти к 3 шагу.

Шаг 3. Шелльный скрипт

Команды, выполняемые вручную в шелле, играючи превращаются в шелльный скрипт: копируем команды из истории и сохраняем их в отдельном файле.

Как и для однострочника, пригодятся &&, `||`, `sed -i` и `tpage`. Кроме того, удобнее, чем в однострочнике, использовать циклы, переменные, временные файлы. Появляется возможность передавать в скрипт параметры (лучше ограничиться позиционными).

Приобретения третьего шага:

- скрипт можно вызывать по короткому и понятному имени;
- при запуске в скрипт можно передавать параметры.

Этот этап — прототипирование на шелле — стоит пройти, даже если сразу понятно, что в дальнейшем понадобится скрипт на Perl.

Во-первых, шелльные конструкции для ма-

нипуляций с процессами и файлами еще более емкие, чем в Perl, поэтому прототип можно сделать совсем быстро.

Во-вторых, нет соблазна преждевременно заняться несущественным: структурой классов, сложной валидацией и т.п. А лишний раз отвлекаться при автоматизации рутины очень опасно: время ограничено, и если не займетесь в первую очередь главным — рискуете остаться безо всякого рабочего инструмента. На всякий случай уточню: речь идет о быстрой автоматизации рутинных ручных действий. Для сложных программ и библиотек архитектура вообще и структура классов в частности не являются несущественными деталями.

Шаг 4. Механическая трансляция в Perl-скрипт

Шелльный скрипт очень легко превратить в простецкий Perl-скрипт: вызовы внешних программ заменить на `qx()`, `if`-ы и `while`-ы заменить на Perl-овые, аргументы командной строки `$1`, `$2` заменить на `$ARGV[0]`, `$ARGV[1]` и т.д.

Сразу же стоит добавить `use strict` и `use warnings`.

При всей своей простоте этот шаг, пожалуй, самый неочевидный из всей последовательности: велик соблазн выкинуть шелльную версию и сразу начать писать «взрослый» Perl, с модулями, плагинами, валидацией и инкапсуляцией. Однако я очень (очень) рекомендую все-таки сдержать себя и начать с «наивного» Perl. Во-

первых, у вас уже есть шелльный скрипт, отлаженный на реальных случаях. Глупо просто так терять все знания, собранные в нем. Во-вторых, переписать скрипт с нуля — дольше, чем переделать shell в Perl. А долгая разработка — это риск вообще ее не закончить.

Приобретения четвертого шага:

- контроль очевидных глупых ошибок в коде (неинициализированные переменные и т.п.);
- пригодный к любым улучшениям код на удобном языке.

Вообще-то желательно перейти от шелльного скрипта к Perl-овому достаточно рано, пока скрипт не вырос и не обзавелся сложными конструкциями. Но даже если

вам в наследство достался уже достаточно большой и сложный шелльный скрипт, в котором понадобилось сделать существенные улучшения, поступите с ним как с прототипом с шага 2. Вместо того, чтобы с нуля переписывать скрипт на Perl (или другой язык), сделайте «механический» подстрочник. Это действительно просто и не требует сложных решений, и ничего из заковыристой логики не потеряется. А затем уже беритесь рефакторить получившийся «автоперевод».

Шаг 5. Простой, но аккуратный Perl-скрипт

Начинаем улучшать бесхитростный Perl-скрипт, полученный на предыдущем шаге. Рекомендую действовать в таком порядке:

1. `use Getopt::Long;` именованные параметры командной строки.
2. По `-h` выдаем простейшую справку: самые типичные примеры использования. О подробном описании параметров пока можно не заботиться, все равно они еще поменяются.
3. Базовая валидация параметров. Страховка от самых досадных ошибок.
4. Очевидные значения по умолчанию.

Сложные случаи, когда непонятно, каково должно быть дефолтное поведение, смело оставляйте на потом.

Что получаем на этом шаге:

- именованные опции и значения по умолчанию делают использование скрипта более простым;

- «пользовательская документация» позволяет легко разобраться с использованием скрипта не только автору, но и любому другому заинтересованному потребителю;
- скрипт делает нужную работу и делает это предсказуемым, документированным образом, так что можно начинать рекомендовать свой скрипт коллегам.

Шаг 6. Постепенные улучшения Perl-скрипта

Перечислю очевидные улучшения и преимущества, которые они дают. Выбирайте то, что для вас важнее, дополняйте список своими любимыми рефакторингами.

- вызовы внешних `grep`, `sed`, `awk` заменить на внутренние `grep`, `map` – скорость и переносимость;
- использовать хеши и вообще сложные структуры данных — скорость, возможность реализовать сложное поведение;
- деление кода на функции — удобство отладки и доработки;
- использовать библиотеки вместо внешних программ — переносимость;
- полноценная валидация параметров — надежность;
- завершаться с правильным кодом — пригодность к использованию в автоматическом режиме;
- подробная справка — удобство для интерактивного использования;
- умные значения по умолчанию в сложных случаях — удобство для

интерактивного использования;

- другие улучшения юзабилити (см. например статью про юзабилити CLI) — тоже удобство при интерактивном использовании;
- обработка краевых случаев (race condition, недоступность ресурсов и т.п.) — надежность, пригодность к использованию в автоматическом режиме;
- настройка через внешние конфигурационные файлы — гибкость, пригодность для широкого класса задач;
- логирование — пригодность к использованию в автоматическом режиме, удобство отладки.

Особые случаи: администрирование Если задача совсем админская (установка ПО, активная работа с удаленными серверами,

однотипная обработка многих серверов) — рассмотрите фреймворки Rex, Fabric, Ansible.

Шаг 7. Большому кораблю — большое плавание

Если разработка небольшой автоматизации дошла до этого шага, значит, автоматизация оказалась (или стала) не такой уж небольшой ^_^

Стоит подумать над тем, чтобы выложить свою утилиту в open source и порекламировать в сообществе: если инструмент так хорошо работает для вас, значит, может пригодиться еще кому-нибудь.

Приобретениями могут стать всенародная слава и любовь, обширный фидбек, сто-

ронние пулл-реквесты, темы для блога и выступлений на конференциях.

И немного «философии»

Польза постепенности

О постепенной (инкрементальной, итеративной) разработке тоже много пишут (например, см. раздел “Succession” здесь, еще ссылки в конце статьи).

Самое важное для наших задач:

- разработка состоит из серии маленьких изменений;
- каждый шаг очень маленький, почти ничего не стоит, но добавляет полез-

ности;

- на каждом шаге получается полезный инструмент;
- полезные свойства программы появляются в порядке приоритета;
- после каждого изменения скрипт готов к использованию;
- после каждого шага можно прекратить разработку на неопределенное время;
- можно выбрать, когда вообще остановиться — в зависимости от того, сколько усилий можно затратить и насколько проработанную программу требуется получить.

Почему это хорошо работает:

- маленькое изменение сделать проще, чем большое;

- частично автоматизированная задача отнимает времени меньше, чем никак не автоматизированная;
- психологически легче доработать уже работающий полезный инструмент, чем с нуля начать разрабатывать новый с неизвестными перспективами;
- каждое изменение делает скрипт удобнее и полезнее, поэтому применение каждой следующей версии радует вас и поощряет потратить еще немного времени на улучшения.

Perl и shell — братья навек

Хочу еще написать о связке «прототипе на шелле» — «скрипт на Perl». Довольно часто встречаются мнения *«зачем мне шелл, у меня есть Perl»* или наоборот *«Perl? Да ну, за-*

чем, на баше напишу».

По-моему, надо использовать и тот, и другой — в тех ситуациях, где проявляются их сильные стороны:

- Шелл весьма выразителен для обеспечения базового поведения: запуск внешних программ, условное выполнение, повторение однотипных команд. Поэтому простой скрипт на шелле можно написать быстрее, чем скрипт на Perl. Кроме того, шелл побуждает сконцентрироваться на главной функциональности. Так что полезно начинать разработку автоматизаций именно с простого шелльного скрипта.
- Перейти от шелла к Perl надо, потому что Perl гораздо лучше подходит для умной валидации, проще для отлад-

ки и доработок, позволяет использовать сложные структуры данных; шелл же тяжел для рефакторинга и отладки, на нем сложно обеспечить бескомпромиссно-удобное поведение.

- Переходить от шелла к Perl стоит через механический «подстрочный» перевод, потому что с одной стороны его можно сделать очень быстро и просто, а с другой — он обеспечивает сохранность уже реализованной логики работы.

И если вам досталось поддерживать и дорабатывать большой и сложный шелльный скрипт — рассматривайте его как готовый прототип, «протранслируйте» на Perl и рефакторите дальше.

Заключение

Еще раз отмечу, что описанный метод касается легковесной автоматизации ручных действий. Для промышленной разработки сложного и отказоустойчивого кода можно попробовать обойтись без сверхбыстрого прототипирования, зато на ранних стадиях стоит оценить требования по производительности и надежности, продумать архитектуру и схему данных.

И напоследок пара слов о происхождении «постепенной автоматизации». Простота манипуляций с файлами и процессами, отсутствие границы между интерактивным шеллом и интерпретатором шелльных скриптов, бесшовная интеграция шелла и Perl-а – все это уже давно встроено в Unix и Unix-подобные системы, шеллы и Perl соответственно, нам остается просто

использовать эти возможности для накапливаемых пошаговых улучшений своих программ. Так что большое спасибо Кену, Брайану, Стивену и Ларри[2] ^_^

Удачных автоматизаций!

Ссылки

О вреде прерываний для программистов:

- It's all in your head — комикс;
- Никогда не отвлекай программиста — статья на Хабрахабре;

---

[2]Естественно, я имею в виду Кена Томпсона, Брайана Кернигана, Стивена Борна и Ларри Уолла. Благодаря им у нас есть вся эта классная юниксовая среда, и это здорово.

- Знайте о пользе сосредоточенности — раздел из статьи “Геймдизайнеру о программистах”;
- Maker’s Schedule, Manager’s Schedule — статья Поля Грэма;
- Интервью с Джейсоном Фридом (37signals);
- Programmer Interrupted;
- Programmers, Teach Non-Geeks The True Cost of Interruptions;
- The hidden cost of interrupting knowledge workers.

О пользе постепенности:

- Про Scrum в Википедии;
- Facebook Engineering / Software Design Glossary — см. раздел “Succession”;
- How To Survive a Ground-Up Rewrite Without Losing Your Sanity — см.

раздел “Worship at the Altar of Incrementalism”;
- Перевод предыдущей статьи;
- Potentially shippable code;
- Ship it! - Scrum’s “Potentially Shippable” Product Increment;
- Does producing potentially shippable product make you less agile? — дискуссия на stackexchange.com.

Фреймворки для легковесной автоматизации администрирования:

- Rex,
- Fabric,
- Ansible.

Статья про юзабилити CLI

■ *Елена Большакова*

3 Постепенная автоматизация в примерах

Примеры к статье Постепенная автоматизация рутинных задач.

Пример из жизни: массовый strace

Ситуация: на сервере работает веб-сервер (для определенности — apache) с Perl-приложением с богатой логикой. Иногда случаются аварии: одна из баз данных или внешний сервис начинают отвечать долго. Время обработки запросов нашим сервером возрастает по крайней мере до величины таймаута на БД/внешний сервис, и, если это достаточно много, то постепен-

но все воркеры apache оказываются заняты ожиданием этого внешнего сервиса, и новые запросы не успевают обработаться.

Внешние симптомы такой аварии: увеличивается время ответа, уменьшается количество обработанных в единицу времени запросов, apache не отвечает на server-status, а ps показывает количество воркеров, соответствующее настройке MaxClients.

Один из способов понять, на чем именно зависли воркеры — это взять наугад один из них и посмотреть `strace`-ом. Если процесс действительно проводит время в `read`-е, `flock`-е, `select`-е — скопировать соответствующий дескриптор и с помощью `lsof -a` посмотреть, какой именно файл или сокет открыт под этим дескриптором (или использовать `strace -y`).

Давайте поавтоматизируем эти действия.

Для демонстрации я буду пользоваться скриптом, который зависает, пытаясь вторично взять лок на уже залоченный файл.

`flock-twice.pl`:

```
#!/usr/bin/perl

use strict;
use warnings;

use Fcntl qw(:flock);

my $filename = "/tmp/lockfile";
open(my $fh1, ">>", $filename) or
    die "open 1";
open(my $fh2, ">>", $filename) or
    die "open 2";

print STDERR localtime()." start:
```

```
    $$\n";
flock($fh1, LOCK_EX) or die "$$:
    flock LOCK_EX 1 $!";

# на этом вызове ——— виснет
flock($fh2, LOCK_EX) or die "$$:
    flock LOCK_EX 2 $!";

flock($fh1, LOCK_UN) or die "$$:
    flock LOCK_UN 1 $!";
flock($fh2, LOCK_UN) or die "$$:
    flock LOCK_UN 1 $!";

close $fh1;
close $fh2;
unlink $filename;

print STDERR localtime()."done\n"
    ;
exit 0;
```

Ручное выполнение

```
> ps gaxuww |grep flock
lena  3154  0.0  0.0  29304   232
    pts/19   SN   May17   0:00 /
   usr/bin/perl ./flock–twice.pl
lena  3157  0.0  0.0  29304   232
    pts/19   SN   May17   0:00 /
   usr/bin/perl ./flock–twice.pl
lena  3164  0.0  0.0  29304   232
    pts/19   SN   May17   0:00 /
   usr/bin/perl ./flock–twice.pl
lena 20114  0.0  0.0  10860   636
    pts/19   S+   20:44   0:00
   grep flock

> sudo strace –p 3154
Process 3154 attached – interrupt
    to quit
flock(4, LOCK_EX^C <unfinished
   ...>
Process 3154 detached

```

```
> lsof –a –p 3154 –d 4 |tail –n 1
flock–twi 3154 lena    4wW  REG
      8,1         0 7222737 /tmp/
    lockfile
```

Однострочник

```
# ps gaxuww |grep 'floc[k]' |awk
    '{print $2}' |while read i ;
    do strace –y –p $i & ; sleep 1
     ; kill –9 $! ; done
[5] 28989
Process 28972 attached
flock(4</tmp/lockfile>, LOCK_EX
    [6] 28993
[5]  – killed     strace –y –p $i
Process 28976 attached
flock(3</tmp/lockfile>, LOCK_EX
    [5] 28997
[6]  – killed     strace –y –p $i
Process 28979 attached
flock(3</tmp/lockfile>, LOCK_EX#
```

```
[5]  + killed     strace -y -p $i
```

Шелльный скрипт

multitrace-0.sh

```
#!/bin/sh

ps gaxuww |
    grep $1 |
    awk '{print $2}' |
    while read i
    do
        echo $i
        strace -y -p $i &
        sleep 1
        kill -9 $!
        echo
    done 2>&1 |
    grep -v 'attached'

#multitrace-0.sh 'floc[k]'
```

```
29032
flock(4</tmp/lockfile>, LOCK_EX
29035
flock(3</tmp/lockfile>, LOCK_EX
29038
flock(3</tmp/lockfile>, LOCK_EX#
```

Шелльный скрипт, версия 2

Но может быть, не стоит массово дергать процессы `strace`-ом? Кстати: статья об опасностях strace. На Linux можно попробовать обойтись информацией из `/proc/<pid>/syscall`. Для расшифровки номера системного вызова используем хедер `unistd_64.h`. Кроме того, добавим расшифровку дескриптора, если он идет первым параметром системного вызова:

`multitrace-1.sh`

```
ps gaxuww |grep $1 |awk '{print $2}' |
    while read i
    do
        str=`cat /proc/$i/syscall `
        n=${str%% *}
        s=${str#* 0x}
        fd=${s%% *}
        lsof=`lsof –p $i –d $fd –a –w |tail –n 1`
        file=${lsof##* }
        syscall=`cat /usr/include/x86_64–linux–gnu/asm/unistd_64.h |grep "\<$n\>" |sed –e 's/^.*_NR_\([^ \t]*\).*$/\1/'`
        echo "$syscall $fd $file $i\t$str"
    done


```

```
#multitrace-1.sh 'floc[k]'
flock 4 /tmp/lockfile 29120
    73 0x4 0x2 0x0 0x0 0x0 0x0 0
    x7fffc5728e28 0x7f13acf7c957
flock 3 /tmp/lockfile 29125
    73 0x3 0x2 0x0 0x7fff2cda7500
    0x0 0x0 0x7fff2cda7738 0
    x7ff67492c957
flock 3 /tmp/lockfile 29130
    73 0x3 0x2 0x0 0x7fff874eb6b0
    0x0 0x0 0x7fff874eb8e8 0
    x7f50ada58957
```

Нормально, только десктипторы с номерами больше 9 обрабатывает неправильно... Исправим в перловой версии.

Подстрочный перевод на Perl

`multitrace-2.pl`

```
#!/usr/bin/perl

use strict;
use warnings;

my @pids = split /\s+/, `ps
    gaxuww |grep $ARGV[0] | awk '{
    print \$2}'`;

for my $p (@pids){
    my $proc_str = `cat /proc/$p/
        syscall`;
    (my $syscall_num = $proc_str)
        =~ s/ .*$//s;
    (my $fd = $proc_str) =~ s
        /^.*?0x([0-9]+).*/$1/s;
    $fd = hex($fd);
    my $lsof=`lsof -p $p -d $fd -
        a -w |tail -n 1`;
    (my $file = $lsof) =~ s/.*
        ([^ ]+)\n$/$1/;
    my $syscall=`cat /usr/include
        /x86_64-linux-gnu/asm/
```

```
        unistd_64.h |grep "\\<
        $syscall_num\\>"`;
16      $syscall =~ s/^.*_NR_([^ \t
        ]*).*\n/$1/;
17      print "$syscall $fd $file $p\
        t$proc_str";
18  }
```

Работает аналогично multitrace–1.sh:

```
1  #multitrace–2.pl 'floc[k]'
2  flock 4 /tmp/lockfile 29206
       73 0x4 0x2 0x0 0x0 0x0 0x0 0
       x7fff1a387348 0x7f9ff4b88957
3  flock 3 /tmp/lockfile 29209
       73 0x3 0x2 0x0 0x7fffa7e20270
       0x0 0x0 0x7fffa7e204a8 0
       x7f560524a957
4  flock 3 /tmp/lockfile 29212
       73 0x3 0x2 0x0 0x7fff406a49e0
       0x0 0x0 0x7fff406a4c18 0
       x7f0db5e62957
```

Большие номера дескрипторов обрабатывает правильно.

Аккуратный скрипт на Perl

После нескольких серий доработок у меня получилось следующее: multitrace@github.

Что могло бы быть дальше

Пока мне хватает имеющихся возможностей ^_^.

А вообще, интересно было бы сделать скрипт более кроссплатформенным, не привязанным к Linux.

Пример 2: поиск в коде

Дано: хотим удобный поиск в коде. Удобный — значит только по файлам `.pl` и `.pm` (а также `.html`, `.tt2`, `.js` и т.п.), исключая метаданные `svn`, `git`, `hg`, а также пару каталогов, где хранится автоматически сгенерированный код.

Нужные файлы можно найти `find`-ом, ненужные пути выкинуть опцией `–prune` или же `grep`-ом, найденные файлы передать `xargs`-ом в `grep`.

Это уже практически рецепт шелльного однострочника, и он решал бы многие задачи поиска в коде. А если пойти дальше и дальше по дороге улучшений — можно написать что-то вроде Ack — better than grep ^_^

Ну вот и все

Если у вас есть свои примеры итеративной разработки — поделитесь в комментариях, это интересно!

■ *Елена Большакова*

4 Обзор CPAN за август 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

16 августа 2014 года впервые отмечался день CPAN. Многие годы день рождения CPAN отмечали 26 октября, когда был сделан официальный анонс CPAN. *Neil Bowers* вместе с *Philippe Bruhat* предложили отметить дату 16 августа как день CPAN, поскольку именно 16 августа 1995 года *Andreas König* загрузил на CPAN первый модуль (`Symdump` 1.20) через интерфейс *PAUSE*. За прошедшие 19 лет с этого момента ещё около 6’500 разработчиков выпустили более 35’000 дистрибутивов с 230’000 модулями.

День CPAN предлагалось отметить выпус-

ком новых модулей, исправлением ошибок в своих или чужих модулях, написанием статьи или осуществлении пожертвования фонду Перл.

Связано ли это с днём CPAN или нет, но в августе произошло необычно большое пополнение в `Games`:

- Games::FrogJump — забавная консольная логическая игра `Games::FrogJump` с прыгающими лягушками. Обладателям Перл 5.20 перед началом игры предварительно потребуется решить головоломку с ошибкой `push on reference is experimental`
- Games::ArrangeNumber — консольная игра «пятнашки»

- Games::Hangman — консольная игра «виселица»
- Games::Cellulo — консольная программа, реализация клеточного автомата (по мнению автора)
- Game::Life::NDim — реализация классической игры Джона Конвея «Жизнь»
- Games::Maze::SVG — генерация интерактивных лабиринтов в SVG-формате

Статистика

- Новых дистрибутивов — 282
- Новых выпусков — 875

Новые модули

- Git::Reduce::Tests

Дистрибутив `Git::Reduce::Tests` содержит утилиту `reduce-tests`, которая может быть полезна при smoke-тестировании git-репозитория, позволяя ограничить список запускаемых тестов. Например, когда модуль имеет множество тестов, запуск которых занимает продолжительное время, а требуется проверить изменение, покрываемое лишь какой-то частью тестов. Утилита создаёт ветку с урезанным набором тестов, позволяя сократить время работы смокера.

- Throw::Back

Ещё один модуль для генерации исклю-

чений. Необычен он своим интерфейсом, позволяя создавать исключение через вызов метода произвольного класса или объекта:

```
1       eval {
2           URI->throw::back("http://
              example.com");
3       };
4       if ( $@ && $@->{type} eq 'URI
            ' ) {
5           # URI exception
6           ...
7       }
```

- AnyEvent::FTP

Смахнув пыль с RFC 959, *Graham Ollis* реализовал модуль ftp-сервера и клиента на основе фреймворка обработки событий `AnyEvent`.

- Mojo::Redis2

Экспериментальный релиз нового драйвера `Redis` для фреймворка `Mojolicious`. Основные отличия от первой версии: поддержка синхронных операций, более одного соединения с сервером Redis и возможность обработки ошибок в функциях обратного вызова.

- API::Instagram

Модуль `API::Instagram` позволяет использовать в ваших приложениях API сервиса Instagram. В качестве бэкенда для отправки веб-запросов используется модуль Furl.

- Perl::Lint

`Perl::Lint` — это реализация статического анализатора перл-кода на основе `Compiler::Lexer` и `Compiler::Parser`, совместимого с `Perl::Criric`. Теоретически подобный анализатор должен быть на порядок быстрее `Perl::Criric`, работающего на базе `PPI`. Это может позволить использовать его для проверки кода в реальном времени, например, в различных IDE.

Данный проект финансируется грантом фонда Перл и на данный момент находится в стадии активной разработки.

- Event::Distributor

`Event::Distributor` — это реализация шаблона проектирования издатель — подписчик (pub/sub) в рамках одного процесса.

Модуль позволяет создавать события, на которые могут устанавливаться обработчики (синхронные или асинхронные), по генерации события вызываются все подписанные на данное событие обработчики.

- App::Ringleader

`App::Ringleader` — реализация прокси для редкоиспользуемых веб-приложений. Это своего рода компромиссное решение между подходом CGI (запуск на каждое обращение) и PSGI (постоянно работающее). Прокси запускает PSGI-приложение по запросу и по истечении периода неактивности (по умолчанию 60 минут) завершает приложение, чтобы не расходовать ресурсы впустую.

- Syntax::Feature::Void

Как известно, функции в перле без явного `return` всегда возвращают результат последней инструкции. В определённых ситуациях это может сыграть с программистом злую шутку, если он этого не учёл. Модуль `Syntax::Feature::Void` дополняет синтаксис Perl ключевым словом `void`, принудительно задавая пустой контекст:

```
1     use syntax qw( void );
2
3     sub foo {
4         void bar();
5     }
```

Это аналогично следующему коду:

```
1     sub foo {
2         bar();
3         return;
4     }
```

- AnyEvent::Future

`AnyEvent::Future` — небольшой модуль, который интегрирует модуль `Future` с `AnyEvent`, позволяя оборачивать функции `AnyEvent` в `Future`-объекты, используя все его возможности.

- Test::Kantan

`Test::Kantan` — новый фреймворк тестирования в соответствии с парадигмой *Behavior Driven Development* (разработка, основанная на поведении/функционировании). Поддерживаются три типа описания тестов: стиль RSpec/Jasmine, Given-When-Then стиль и старый добрый Test::More.

Обновлённые модули

- DBD::mysql 4.028

В новой версии драйвера для MySQL исправлено несколько ошибок, в том числе крах при использовании опции `mysql_auto_reconnect` в случае выполнения запроса в момент, когда сервер недоступен.

- Plack 1.0031

В новом релизе суперклея для веб-фреймворков и веб-серверов Plack присутствует исправление, касающееся безопасности. `Plack::App::File` удалял все завершающие слэши из запрошенного

пути, что при традиционном использовании `Plack::Middleware::Static` давало возможность злоумышленнику обходить существующие правила исключений для нестатических страниц.

- PPI 1.218

После трёхлетнего перерыва выпущена новая версия модуля `PPI` для разбора и анализа перл-кода. Проект переехал на гитхаб, исправлены некоторые ошибки парсера и документации, снято ограничение на размер обрабатываемого файла в 1 МБ.

- CGI::Emulate::PSGI 0.19

В `CGI::Emulate::PSGI` исправлена ошибка при работе с Perl 5.20: в случае когда CGI-

скрипт делал какой-либо вывод на STDERR, возникали рекурсия и аварийное завершение процесса.

- JE 0.62

Обновлена реализация движка JavaScript на чистом перле. Новая версия теперь успешно работает на Перл 5.20.

- Plack::Middleware::Session 0.23

В версии 0.22 `Plack::Middleware::Session` появилось предупреждение, а в версии 0.23 — фатальная ошибка в случае, если используется `Plack::Session::Middleware::Cookie` без параметра `secret`. Так как по умолчанию в куках

используется сериализация `Storable`, отсутствие `secret` позволяет злоумышленнику проводить атаку с возможностью выполнения кода на стороне сервера.

- MongoDB v0.704.5.0

В новом релизе драйвера MongoDB теперь генерирует исключение, если в BSON (де)кодируются строки с некорректным UTF-8.

- Pinto 0.09995

Исправлена критическая уязвимость в вебсервере `Pinto::Server`. Запрос с использованием `/../` в пути позволял получать файлы за пределами корневого каталога репозитория.

- Net::SSLeay 1.66

Новая версия модуля `Net::SSLeay` выпущена под лицензией *Perl Artisitic License 2.0*, в то время как раньше модуль имел лицензию *OpenSSL*. Это важное изменение снимает ограничение на возможность включения модуля в состав базового дистрибутива перла.

- Perl::Critic 1.122

Обновлённый `Perl::Critic` теперь требует последнюю версию `PPI`. Обновлены и сделаны более строгими и некоторые другие зависимости.

- Scalar-List-Utils 1.40

В составе новой версии дистрибутива `Scalar-List-Utils` появился новый модуль `Sub::Util`, в который был перенесён метод `set_prototype`, а также появились функции `subname` и `set_subname` для, соответственно, получения и задания имени подпрограммы. Теперь можно выкидывать `Sub::Name` и `Sub::Identify`...

■ *Владимир Леттиев*

5 Интервью с Куртисом “Ovid” По

Куртис По (Ovid) — Perl-программист, рекрутер, автор книги Beginning Perl, автор популярного блога про жизнь иммигранта.

Когда и как научился программировать?

Впервые я научился программировать в 1982 году, когда учитель геометрии разрешил мне поиграться с компьютерами в лаборатории, хотя они и были для старших учеников. Я выучил BASIC и к 1984 году написал относительно полноценный (для того времени) графический редактор для TRS-80, но графика была настолько малого разрешения, что любая картинка занимала несколько экранов (и файлов) и можно было сдампить их на матричные принтеры. Студенты в компьютерной ла-

боратории проводили много времени за моим редактором. Через пару лет я начал писать текстовую игру (для развлечения, у меня до сих пор не было работы в качестве программиста). Помню, что парсинг моего тестового предложения занимал восемь секунд на BASIC. Этим предложением было “hit the skeleton with the sword, take his ring and then walk north and touch the alter”. Чтобы это как-то ускорить, я выучил assembler, а затем и C. К сожалению, так как я жил в небольшом городишке, и у меня не было диплома, с 1988-го я программировал лишь изредка. В 1998-м я впервые получил свою первую работу в качестве программиста. С тех пор этим и занимаюсь.

Какой редактор используешь?

Я использую vim, но это только из-за мышечной памяти, а не потому что я

считаю, что vim обязательно лучше или хуже других редакторов. Я был вынужден выучить vim на моей первой Perl-работе в 2000-м году. Я ненавидел этот чертов редактор и постоянно смотрел туториал. На сегодняшний день у меня очень сильно кастомизированное vim-окружение, это круто, но сильно усложняет парное программирование, потому что я передвигаюсь по коду настолько быстро, что другие разработчики не могут угнаться за тем, что я делаю. Или я правлю строчку кода, и внезапно запускаются тесты, не выходя из vim. Или я вбиваю `<leader>rb`, и внезапно весь мой проект компилируется. Или выделяю несколько строчек кода, и бам — выделяется метод! Я очень быстр в рефакторинге и вычищении кода, но это в основном благодаря утилитам, которые я встроил в vim.

Любопытен тот факт, что мощность моей настройки vim может быть минусом в дебатах «текстовый редактор или IDE». Я не считаю себя особенно талантливым в vim, но, несмотря на это, иногда больно наблюдать как им пользуются другие разработчики. За годы моего опыта я понял, что большинство программистов так и не научились им правильно пользоваться. Поэтому в стандартном IDE, несмотря на раздутость (я смотрю на тебя, Eclipse) и множество особенностей, которые никогда не используются, разработчики могут делиться фишками, указав какой-нибудь шорткат или настройку в меню. Мое окружение настроено так, как я программирую, и этим тяжело делиться или как-то объяснить. Оно оптимизирует меня, но не оптимизирует массы. Я подозреваю, что IDE лучше с этим справляется. Я даже пробовал сделать урезанную версию моего vim-конфига, потому

что многие просили им поделиться, но в итоге это превращается в какую-то кашу.

Когда и как познакомился с Perl?

В 1999-ом году я работал COBOL-программисто
в страховой компании и там познакомился с одним Unix-администратором. Он мне все уши прожужжал про эту новую штуку “Perl”. Я пытался решить одну задачу с помощью COBOL, где нужно было преобразовать CSV-файл из NT-системы в формат с фиксированной шириной, который бы принимался COBOL. У COBOL много слабостей, и работа с тестом одна из них. Код был длиной в 150 строк, но все потому, что автор не понимал как работает функция `unstring`. Мне удалось уменьшить код до 80 строк. Чисто из любопытства я попробовал Perl и получил файл в 10 строк понятного кода, и это с обработкой

ошибок. Когда тот Unix-админ уволился, чтобы начать свою компанию, я с радостью перепрыгнул с COBOL на Perl.

По иронии судьбы многие Perl-программисты пишут на Perl как на C, но я начал писать как на COBOL (это слишком длинная история). Perl-сообщество научило меня, как по-настоящему программировать. Я за это очень благодарен.

С какими другими языками нравится работать?

Моим любимым будет JavaScript. Например, написание таких вещей как вращающаяся 3D-карта звездного неба приносит мне много удовольствия, и это в той области, где Perl не сильно подходит. Мне также нравилось писать на Python и Ruby, но к сожалению, в моем резюме Perl на-

ходится настолько долго, что компании, которые предлагают мне контракты, не будут платить мне столько, сколько я могу заработать с Perl.

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

Самым большим преимуществом Perl является его сообщество, тут даже думать не надо. Я наблюдал, как сообщества других языков переживают те же проблемы, которые Perl-сообщество уже давно пережило. Оно старше других сообществ, и мне кажется, что оно более зрелое.

Кроме этого, я бы назвал феноменальную поддержку юникода и CPAN, на котором хоть и много хлама, но, если разобраться, можно найти множество отличных модулей, которые превращают тяжелые задачи

в простые.

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

Параллелизм. Мы еще с этим не столкнулись, но закон Мура приблизится к своему завершению в ближайшее десятилетие или раньше. Если мы хотим быстрые системы, надежная система распараллеливания просто необходима. Форки слишком дороги для небольших программ, а треды просто ужасны (сколько опытных программистом смогут назвать четыре условия, необходимых для блокировки?). Помню, что Джонатан Вортингтон сравнивал треды с ассемблером параллельного программирования, и как бывший ассемблер-программист, я вынужден с этим согласиться. Нужна всего лишь одна небольшая, совсем небольшая ошибка, и вы сталкиваетесь с кошмаром

сложно воспроизводимых проблем. Очень много работы сейчас проводится для улучшения моделей распараллеливания, и нам они очень пригодятся в будущем. Если процессоры не могут становиться быстрее, мы должны утилизировать пользу распараллеливания, если хотим улучшения производительности.

(Хотя некоторая часть меня считает, что если CPU станут настолько дешевыми и распространенными, что у многих разработчиков по-прежнему будет достаточно работы для написания стандартного кода, без параллельности.)

Как тебе опыт написания Beginning Perl? Что мотивировало больше всего?

Когда Wiley (издательство — *прим. перев.*) в первый раз связалось со мной по поводу

книги, я хотел отказаться. Моя жена Лейла только что родила, мы переехали в Амстердам, жизнь была хаотична. Однако Лейла настояла на том, чтобы я написал книгу, сказав, что позаботится о нашей дочери. Это было адом для нас обоих. Я провел восемь месяцев за написанием, по-моему, самой большой Perl-книги, написанной одним автором. Мой технический редактор, chromatic, выполнил героическую работу по нахождению разного рода проблем в книге, но к сожалению, опечатки все равно прокрались. Я все еще очень доволен книгой, и критика была отличная, но было бы хорошо, если бы у меня было немного больше времени для исправления всех опечаток.

Хочу рассказать небольшую инсайдерскую историю, которую, я думаю, никогда никому не рассказывал. В одном из первых

черновиков я пошутил про терроризм (в книге по программированию!), и мой редактор спросил, считаю ли я, что это уместно, если учесть современный политический климат. Я просто ответил «да». Вообще, я обычно уступаю редактору право решать, но в этот раз я настоял на своем. Вскоре после этого редактора назначили на работу с другой книгой, и, хотя мне сказали, что это была всего лишь внутренняя реорганизация, я до сих пор думаю, не обиделся ли он на мое чувство юмора настолько, что не хотел больше со мной работать. Это дает мне возможность вспомнить о Мэри Джеймс, Маурен Спирс и Сан Ди: трех замечательных людях в Wiley, которые работали со мной над книгой. Если бы не они, сомневаюсь, что у меня получилось бы сохранить здравый рассудок. Не могу говорить за других людей, но я бы с радостью поработал с Wiley еще раз.

Популярны ли до сих пор книги о Perl, если брать во внимание количество продаж?

Все зависит от того, что понимать под «популярны». Я вполне доволен продажами своей книги, кроме того я знаю, что она продается лучше, чем некоторые другие. Но несмотря на это, есть некоторое сокращение рынка книг о Perl. Так как я много занимаюсь консалтингом и тренингом в http://www.allaroundtheworld.fr/, я вижу, что Perl до сих пор популярный язык и новые проекты и компании постоянно возникают, но есть предубеждение, что Perl не настолько силен как раньше. Думаю, что это сильно вредит книжному рынку.

Какие были причины для написания еще одного модуля для тестирования в Perl? Как он отличается от других?

Я написал много из них, но, думаю, имеется в виду `Test::Class::Moose`. Изначально я его написал как замену древнего модуля `Test::Class` от Адриана Ховарда. К сожалению этот модуль начал проявлять свой возраст, тогда как `Test::Class::Moose` разработан как современный xUnit-фреймворк. Меня сильно удивило, что все больше и больше компаний начинают его использовать, настолько он мощный. Как только вы с ним освоитесь, его просто использовать, и он действительно готов к использованию в серьезном деле.

Из коробки тесты запускаются быстрее, легко распараллеливаются (включая простоту написания алгоритмов для распараллеливания), вы получаете отчеты, возможность реорганизации тестовых наборов (запуск только тех тестов, над которыми вы работаете), утилиты для миграции

с `Test::Class`, запуск тестов в разных контекстах (спасибо Дэйву Рольски!), и полная интеграция с `Moose` для мощного ООП-тестирования. До сих пор помню, как однажды показывал модуль в одной компании, и они сразу же согласились его использовать, когда увидели, как просто можно запускать подмножества тестов в зависимости от требований заказчика.

Несмотря на это, я много раз игрался с идеей написания Testament. Это был бы преемник `Test::Class::Moose`, который бы решал две самые большие проблемы: он бы изменил мнение людей, что он только для кода, написанного с помощью `Moose`, и он позволил бы компании решать, какие тестовые модули они хотят включить в поставку, вместо того, чтобы использовать `Test::Most` по умолчанию.

Почему считаешь, что у людей должен быть опыт жизни и работы в разных странах? В чем преимущества иммиграции?

Легче ненавидеть того, кого не знаешь. Я сейчас живу во Франции и должен сказать, что большинство стереотипов о французах просто неправдивы. Они не высокомерны. Они не ленивы. Они не грязны. Они радушные, духовно богатые люди с фантастичной культурой и глубокой любовью к хорошей еде и вину (и, к сожалению, бюрократии). Вообще, я жил в пяти странах, и каждая из них отличается от того, что думают о ней извне. Главное, с чем вы сталкиваетесь, если знакомитесь с местными, что люди — это просто люди. Мы не американцы, французы, японцы, иракцы или что-либо подобное. Мы люди. Когда вы пожили в стольких странах, во скольких жил я, вы узнаете, что большинство людей

по своей природе хорошие, и, к большому сожалению, новостные репортажи о плохих вещах формируют общественное мнение.

Иммиграция — это одна самых стоящих вещей, которые человек может переживать, особенно если вы долго живете в стране, где язык сильно отличается от вашего. Даже если мы фундаментально одинаковы, в зависимости от культуры это по разному проявляется. Когда вы переживаете это лично, вы начинаете понимать, почему люди такие, какие они есть. Я искренне рекомендую!

Среди многих вещей ты также занимаешься рекрутингом. Можешь рассказать чем твой подход отличается от общепринятого?

Для рекрутеров слишком низкий пород входа. Все что нужно, это сказать «Я рекрутер» и бац! — ты рекрутер. Это понижает и качество, и цены. Компании тратят очень мало на рекрутинг, а затем жалуются, когда получают плохой результат. Рекрутеры, в свою очередь, обычно ищут ключевые слова, без понимания о нужной позиции, на которую ищут кандидата, затем пропихивают резюме компаниям. И все по новой.

Нам такое не нравится, и мы отказались от нескольких рекрутинговых контрактов, потому что компании хотели лишь резюме. Мы любим работать с компаниями, которые хотят получить качественных кандидатов, особенно для узкоспециализированных позиций, которые тяжело найти, и которые позволяют нам делать всю работу правильно. Это включает в

себя понимание, какие именно профессиональные качества требуются компании (технические качества), а также качества социальные (работа в срок, убеждение коллег, умение работать с постоянно меняющимися требованиями и так далее), необходимые для нахождения правильного кандидата.

Вообще, в начале мы проводим интервью с компанией. Когда мы готовы, мы начинаем интервьюировать кандидатов. Вначале мы фильтруем резюме. Если они прошли через этот фильтр, мы обычно проводим технический тест — большинство кандидатов отсеивается на этом этапе — и далее мы проводим тестирование их нетехнических качеств, используя так называемое «труктурированное интервью». К сожалению, такие интервью неизвестны рекрутинговым компаниям, но в отличие

от обычного интервью (что не сильно отличается от случайного шанса найти хороших кандидатов), структурированное интервью является отличным предсказателем успешности или неуспешности кандидата. Оно требует времени для изучения, управления, оценки, но когда мы передаем кандидата компании, их финальное интервью обычно сводится к «захочу ли я посидеть в баре с этим человеком?» с гораздо меньшим риском.

Также хочу добавить, что это отлично работает в нашей специализации с международным рекрутингом. Если ваша компания находится в небольшом городе, как например, Партен во Франции, вам будет довольно сложно привлечь французов для работы. По факту, даже несмотря на то, что Франция — это часть Европейского Союза, граждане союза, переезжающие во

Францию, обычно хотят в такие города как Париж, Бордо или Ницца. Кто хочет переезжать в глушь? Но компания в Партене, которая открыла себя для глобальный рынок труда, теперь переполнена кандидатами, которые хотят жить во Франции, даже если это небольшой город. Это то, где мы можем пригодиться: мы рассказываем компании, как работает иммиграция, как они могут помочь своим работникам, как работают Голубые карты в ЕС, как справляться с переездами, и мы также проводим рекрутинг таким образом, что кандидаты, переезжающие во Францию, уже готовы к интеграции, и к тому же отлично подходят компании. Это очень много работы, но я сам иммигрант, мне это очень нравится, и мне нравится видеть, как другие реализуют свои мечты жизни за границей.

Что думаешь о будущем Perl?

У Perl все лучше, чем многие люди думают. Он был очень популярен, когда практически не было конкуренции, но когда конкуренция появилась, его часть поуменьшилась. К сожалению, многие люди не понимают, как работает экономика. Когда есть низкий порог входа на популярный рынок, всегда появляются конкуренты. Это означает, что часть рынка у лидеров уменьшается и они не проигрывают, а не соревнуются.

На сегодняшний день Perl — все еще один из самых быстрых динамических языков, с отличной поддержкой юникода и процветающим сообществом. Есть две вещи, которые исторически нам вредили: отсутствие сигнатур методов/функций и отсутствие приличного ООП-ядра. С Perl 5.20 у нас отличные сигнатуры. А Рикардо Сайнс сказал, что если появится приличный MOP (метаобъектный протокол), он

добавит его в ядро.

Если это будет, то два самых больших ограничения перла исчезнут, и наша ООП-система уже не будет сравнимой с тем, что есть в других динамических языках: она их попросту порвет. К тому времени, мне кажется, многие технические проблемы будут уже позади, но маркетинг, проводимый перл-сообществом, по-прежнему не слишком хорош.

Несмотря на это, наш маркетинг в течение последних лет, воодушевил многих в сообществе. И сейчас пришло время выйти за сообщество.

Стоит ли советовать молодым программистам сейчас учить Perl?

Мы должны учить их программировать.

Я, конечно, буду рад увидеть больше молодых людей, изучающих Perl, но в реальности, если вы хороший программист, вы должны быстро схватывать большинство языков. Не привязывайтесь к одной технологии.

Вопросы от читателей

Напишешь еще одну книгу о Perl? Или, возможно, о другом языке или технологии?

Сейчас не знаю, у меня планов пока нет. Я думал, что было бы весело написать «Perl для криминала», объясняя как начинающему программисту использовать Perl для улучшения эффективности криминального синдиката. Это обязательно будет замечено, но, наверное, не так, как мне хотелось бы. Я так занят своей компа-

нией, своей ролью мужа и отца, а книги приносят так мало дохода, что это вряд ли произойдет.

Как найти работу за рубежом?

Это займет слишком много текста, поэтому хочу направить всех на мою страницу “Start Here” на моем блоге про иммиграцию.

■ *Вячеслав Тихановский*