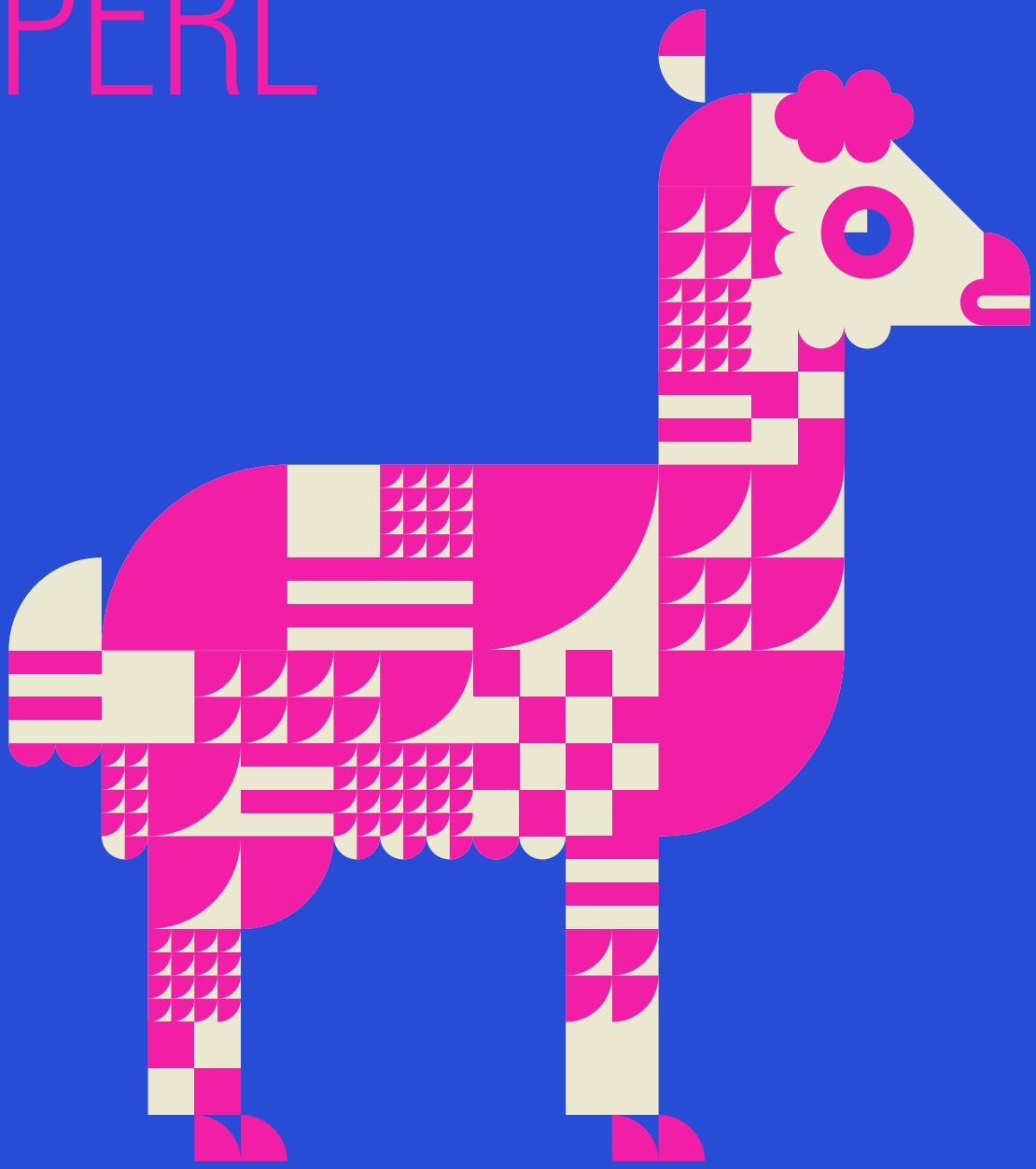
PRAGMATIC PERL

15



05/2014

pragmaticperl.com

Pragmatic Perl 15

pragmaticperl.com

Выпуск 15. Май 2014

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

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

Авторы статей: Елена Большакова, Константин Уварин, Владимир Леттиев

Обложка: Марко Иваныk

Корректоры: Андрей Шитов, Георгий Бажуков

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 16:19

Оглавление

1 От редактора

Конференция YAPC::Russia 2014 в этом году пройдет в Санкт-Петербурге (http://event.yapcrussia.org/yr2014/). Известны место проведения и даты 13-14 июня. А можно воспользоваться модулем `YAPC::Russia`. Для участия в конференции необходимо зарегистрироваться. Как обычно посещение бесплатное и организаторы не откажутся от спонсоров.

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

Приятного чтения.

■ *Вячеслав Тихановский*

2 Простые способы сделать консольную утилиту удобнее

Рассмотрены основные принципы написания удобных консольных приложений.

Качественная командная строка — отличнейшее окружение для работы. История, автокомплит, конвейеры, перенаправления, широчайший набор готовых программ, возможность скопировать-и-вставить даже самую сложную команду — все это делает CLI мощным и удобным инструментом. Ничего удивительного, что разработчики охотно пишут собственные консольные программы и скрипты: автоматизация сборки и деплоя, разворачивание тестовой БД, статистика, мониторинг, хитроумный поиск — поводов не счесть.

Однако «консольная утилита» не означает автоматически «удобная в использовании утилита». Неудачное имя, неполное или отсутствующее описание, большое количество позиционных параметров, запутанное именование переключателей — командную строку можно наполнить когнитивным сопротивлением[1] похлеще самого запутанного графического интерфейса.

В этой статье собраны несложные советы, следование которым поможет сделать свои консольные скрипты более удобными в работе.

Оговорки: советы довольно простые, и

---

[1]О когнитивном сопротивлении можно прочитать в книге Алана Купера «Психбольница в руках пациентов», глава «Поведение, не связанное с физическими силами»

если они покажутся вам само собой разумеющимися — отлично. К сожалению, даже о таких простых вещах временами забывают. Далее, я предполагаю, что и вы, и ваши пользователи живете в unix-подобном окружении: полноценный шелл и полноценный набор стандартных программ. Кроме того, я использую слова «программа», «скрипт» и «утилита» как взаимозаменяемые синонимы. И наконец, в качестве примеров в статье приведены фрагменты кода на Perl, однако все советы в равной мере относятся к скриптам и программам на любом языке программирования, а технические приемы легко транслируются на любой язык.

А стоит ли вообще писать новый скрипт?

Лучший скрипт — тот, который не надо писать. Может быть, задача решается простой комбинацией уже существующих программ? Утилиты `make`, `sort`, `find`, `grep`, `lsof`, `netstat`, `strace` и т.д. не только хороши сами по себе, но и отлично склеиваются через конвейеры («пайпы»).

А может быть, задача решается коротким perl-однострочником, который проще написать заново[2], чем вспоминать название готового скрипта? Кстати, perl-однострочники тоже прекрасно встраиваются в конвейеры.

---

[2]Подход «написать однострочник и забыть» называется ad hoc ;)

Хорошее имя

Какое оно — хорошее имя для хорошей программы? Короткое или длинное? Абстрактное или описывающее поведение? Однословное или составное?

Вот несколько простых правил:

- чем реже используется программа, тем длиннее может быть ее имя, и наоборот; сравните часто используемую `ls` и гораздо более редкую `apt-get install`;
- чем более узкую задачу решает скрипт, тем более подробным и точным должно быть имя; сравните обобщающе-уклончивое `make` и донельзя конкретное `ps2pdf`;

- имя может быть абстрактным и ничего не значащим, но ни в коем случае оно не должно вводить в заблуждение; скрипт для подготовки релиза можно назвать и `make-release`, и `rc1`, но не стоит называть его `test-mainline`, `launch` или `new-version`.

Именованные параметры

Если параметры скрипта становятся сложнее, чем простой список файлов (как у `rm`) или пары «что-куда» (как в `cp`), «что-где» (как в `grep`) — скрипту нужны именованные параметры командной строки.

Getopt::Long входит в стандартную поставку perl с 1994 года и позволяет легко разби-

рать самые разнообразные опции:

- короткие (однобуквенные),
- длинные (многобуквенные),
- флаги с автоматически доступным отрицанием (`--cache`, `--no-cache`),
- синонимы (`-q` и `--quiet`, `-h` и `--help`),
- опции с обязательными значениями (тип значения можно указать: строка, число, вещественное, шестнадцатеричное),
- умолчальные значения для опций,
- автоматическое заполнение массивов и хешей,
- склеивание коротких опции (как `perl -lane`, `ls -la`).

В общем, изучить документацию на `Getopt::Long` и попрактиковаться в его применении — стоящее дело.

Однобуквенные или многобуквенные опции?

Мне кажется, что практически для каждой опции стоит иметь и однобуквенный, и многобуквенный варианты (как `-q` и `--quiet`). Однобуквенные ключи удобны при интерактивной работе, так как их быстрее набирать, а многобуквенные — в мейкфайлах и других скриптах, так как более понятны при чтении.

Традиционные имена параметров

Хотите, чтобы пользователи быстрее запомнили параметры вашей утилиты — называйте привычные действия привычными именами:

- `-h`, `--help` — помощь,
- `-V`, `--version` — вывод версии программы,
- `-q`, `--quiet`, `--silent` — режим с менее подробным выводом,
- `-v`, `--verbose` — режим с более подробным выводом (интересный пример находим в `ssh`: `-v`, `-vv`, `-vvv` дают все более и более подробное логирование),

- `-n`, `--dry-run` — пробный запуск без выполнения пишущих действий,
- или `-n` — количество элементов, которые следует обработать,
- `-o`, `--output` — файл для записи результата,
- `-f`, `--file` — файл с данными для обработки,
- `-r`, `--reverse` — обработка в обратном порядке,
- `-j`, `--jobs`, `--parallel` — во сколько процессов распараллеливать обработку.

Антипримеры можно часто наблюдать в windows-версиях популярных unix-программ. Например, `ping`: бесконечная

отправка пакетов включается ключом `-t` вместо умолчального поведения, количество пакетов регулируется ключом `-n` вместо `-c`, размер пакета `-l` вместо `-s`, TTL `-i` вместо `-t` и т.п. Или `tracert` в сравнении с `traceroute`: максимальное число прыжков `-d` и `-m` соответственно, не резолвить адреса в имена `-d` и `-n`. Такой разнобой в именовании очень неудобен, особенно если приходится работать попеременно то с одним, то с другим вариантом программы.

Как объяснить пользователю, что он неправ

Если переданные скрипту параметры не проходят разбор и валидацию, надо корректно сообщить об этом пользователю:

```
GetOptions(...) or die "can't
    parse command line arguments,
    stop\n";
die "You must give at least one
    search pattern" unless
exists $OPT{search_pattern};
```

Важно, чтобы сообщение об ошибке было коротким и ясно говорило, что именно не так с параметрами. Недопустимо в ответ на неправильные параметры выводить полную справку — это никак не поможет вашему потребителю понять, что происходит.

Посмотрим, как ведут себя популярные программы:

```
> git up
git: 'up' is not a git command.
    See 'git --help'.

```

```
Did you mean one of these?
    pull
    push


> grep --li
grep: option '--li' is ambiguous; possibilities: '--line-buffered'
'--line-regexp' '--line-number'
Usage: grep [OPTION]... PATTERN [FILE]...
Try `grep --help' for more information.
```

Здесь все коротко и по существу:

- короткое сообщение об ошибке,
- подсказка: где смотреть полную справку,

- предположение: что на самом деле могло иметься в виду.

Когда приходится умирать

Если скрипт по каким-то внутренним причинам не может продолжать работу — пора вызывать `die` (=вывести сообщение и завершиться с ненулевым кодом).

Полезный совет: сообщение об ошибке завершать недвусмысленным `', stop.'` это делает для пользователя очевидным, что программа остановилась именно из-за обнаруженной ошибки.

```
> show-releases.pl -n 10
can't connect to tracker, stop.
```

Умирать правильно

Скрипт обязательно должен возвращать честный код возврата: в случае успешного завершения — 0, в случае неудачи — что-нибудь другое. Правильный код выхода позволяет успешно использовать скрипт в makefile'ах, для svn-bisect и т.п.

Обратите внимание: у `grep`'а есть специальный режим, когда он *вообще ничего ничего не печатает*, только сигнализирует кодом выхода: нашел или не нашел.

Кстати, Perl'овый `die` автоматически обеспечивает ненулевой код завершения.

Справка (-h)

По `-h` (желательно и по `--help` тоже) скрипт должен выводить справку о себе.

Проверьте, что в справке описано:

- назначение программы;
- все опции и параметры, с делением на обязательные/необязательные и принятыми умолчальными значениями;
- типичные и заковыристые примеры использования: пользователь сможет их скопировать и сразу же получит пример работы программы.

Иногда справку пытаются выводить в

`stderr`. Это неправильно. Справка должна попадать в `stdout`, чтобы ее легко было обрабатывать `grep`'ом, `less`'ом и т.п.

Еще можно обращать внимание на переменную окружения `$PAGER`, и если она выставлена — передавать справку через пайп этой программе. Например, так поступает `git help <command>`.

Еще бывает, что вывод справки заканчивают ненулевым кодом выхода (`exit 2;`). Это неправильно. Если пользователь запрашивал справку, то ее вывод — успешно выполненная задача, и скрипт должен сообщать, что закончился успешно (`exit 0;`).

Разумные умолчания

Хорошие умолчательные значения критически важны для эффективной работы с программой. Выбирать умолчания следует исходя из того, что является основной задачей программы и каким образом она будет использоваться чаще всего.

Чем чаще нужна какая-либо опция, тем проще она должна включаться, а самое частое значение параметра должно предполагаться по умолчанию. Идеал: программа выполняет наиболее часто требующуюся задачу вообще без параметров (например: `cal`, `debuild`, `gzip`, `ls`, `make`, `passwd`, `plackup`).

И опять хороший пример подает `grep`: поиск в `stdin` делается по умолчанию; поиск по списку файлов включается простым

их перечислением; рекурсивный поиск, размер контекста и нечувствительность к регистру включаются однобуквенными опциями; экзотика типа управления буферизацией — многобуквенными опциями.

Интересно устроено у `GNU grep` управление цветной раскраской вывода: по умолчанию при выводе на интерактивный терминал вывод раскрашен, при выводе в файл — не раскрашен, а для ручного управления раскраской есть многобуквенная опция `--color`.

Автокомплит

Если у вашего скрипта много возможных параметров (особенно многобуквенных), напишите и выдайте вашим пользова-

телям функции для автокомплита (автодополнения) в популярных шеллах. Документация: для zsh, для bash.

Кстати, обратите внимание на функцию `gnu_generic` в `zsh`: если по `--help` ваш скрипт рассказывает о своих параметрах в достаточно общепринятом формате, для включения автодополнения по параметрам будет достаточно сделать

```
compdef _gnu_generic my-script.pl
```

Маленькая хитрость: раскраска вывода

Если вашим скриптом будут пользоваться люди в интерактивном режиме — упростите восприятие вывода, раскрасив его в раз-

ные цвета. См. например Term::ANSIColor.

Маленькая хитрость-2: молчаливый запрос пароля

Если скрипту надо спросить у пользователя пароль или иную секретную информацию, отключите отображение вводимых символов. Например, с помощью Term::ReadKey:

```
use Term::ReadKey;

ReadMode("noecho");
chomp(my $password = <STDIN>);
ReadMode(0);
```

Итого

Таковы, по моему опыту, простейшие способы улучшения user experience консольных программ.

Если вам тоже есть чем поделиться на эту тему — пишите в комментариях или ответными статьями, тема того заслуживает.

■ *Елена Большакова*

3 Модульное тестирование под AnyEvent

Рассмотрены некоторые простые соображения, касающиеся тестирования модулей Perl под управлением AnyEvent. Также рассмотрен вспомогательный модуль AE::AdHoc, берущий на себя часть рутинных задач, возникающих при тестировании.

Общие соображения

Здесь и далее мы будем предполагать наличие модуля `My::Module` с методами `new` и `do_something` без аргументов, а также блока `ok_result`, который в действительности может быть как отдельной функцией, так и «портянкой» из `ok`, `like` и т.д.

Мы также предполагаем для простоты,

что конструктор можно вызвать без параметров и он никак не взаимодействует с окружением.

Лирическое отступление 1. Вообще в написании тестов очень помогает возможность порождать объекты с минимальными усилиями, а не путём тщательной фальсификации внешнего мира. Если инициализация реального объекта этого требует, можно сделать отдельный метод `startup()` или вроде того.

Вот как выглядит простейший тест (предположим, что `use_ok` было в соседнем тесте).

```
1 use strict;
2 use warnings;
3
4 use Test::More;
5
```

```
use My::Module;

my $object = My::Module->new;

my $result = $object->
    do_something;
ok_result( $result );

done_testing;
```

Теперь добавим немного асинхронности. Вместо метода `do_something` у нас будет `do_something_async`, который ничего не возвращает, но имеет одним из аргументов (обычно либо последним, либо это элемент хеша с именем `callback`, `on_done` и т.п.) коллбэк, то есть функцию, в которую будет передан результат.

Я начинал примерно следующим образом:

```
use strict;
```

```
use warnings;

use Test::More;
use AnyEvent;

use My::Module;

my $object = My::Module->new;

my $cv = AnyEvent->condvar;    # or AE::condvar if one prefers
$object->do_something_async(
    sub {
        ok_result(shift);
        $cv->send();
    }
);
$cv->recv;

done_testing;
```

Этот код рабочий, но он неправильный (так

бывает).

Во-первых, если по какой-то причине коллбэк не вызовется никогда, то скрипт попросту зависнет. За это админы вашей системы деплоймента скажут вам отдельное спасибо. Так что хорошо бы к скрипту добавить таймер.

Во-вторых, `ok_result` на самом деле может выбросить исключение. В самом тесте это не страшно — будет проваленный тест и, следовательно, сигнал, что что-то пошло не так. В коллбэке это приведет все к тому же зависанию скрипта.

Лирическое отступление 2. Разумеется, можно написать тест так, чтобы он не умирал никогда. Однако написание неумирающего кода — отдельное и в данном случае лишнее умственное упражнение. Тесто-

вый код по возможности должен быть в разы *проще тестируемого, иначе это не упрощение разработки, а просто двойная работа.*

Ну и напоследок чисто стилевое замечание. `ok_result`, напоминаю, не обязательно состоит из одной строчки. При дальнейшем редактировании теста может получиться лапша, в которой непонятно, где тестирующий код, а где тестируемый.

Да, а еще неплохо добавить `use AnyEvent::Strict;` — тоже страховка на случай неудачного редактирования `My::Module`.

При учете этих соображений получается примерно следующий код.

```
1 use strict;
2 use warnings;
3
```

```
use Test::More;
use AnyEvent::Strict;

use My::Module;

my $object = My::Module->new;

my $cv    = AnyEvent->condvar;
my $timer = AnyEvent->timer(
    after => 10,
    cb    => sub {
        $cv->croak("Timeout
           exceeded");
    }
);

$object->do_something_async(
    sub {
        $cv->send(shift);
    }
);
my $result = $cv->recv;
undef $timer;
```

```
26
27 ok_result( $result );
28
29 done_testing;
```

Тут бы и остановиться, но...

Модуль Ae::AdHoc

...написание однотипной обвязки очень быстро надоедает. Поэтому был создан заменяющий ее модуль. Стало немного короче, но, главное, теперь и ошибиться-то толком негде.

```
1 use strict;
2 use warnings;
3
4 use Test::More;
5 use AE::AdHoc;
```

```
6
7  use My::Module;
8
9  my $object = My::Module->new;
10
11 my $result = ae_recv {
12      $object->do_something_async(
           ae_send);
13 } timeout => 10;
14
15 ok_result( $result );
16
17 done_testing;
```

Оговорюсь сразу, что название не совсем удачное и, доведись мне писать его сейчас, я бы и переименовал, и внутри тоже по-другому сделал (возможно, через Future или Promise — тогда я про них попросту не знал).

Обратите внимание: `ae_send` возвращает *коллбэк*, останавливающий event loop.

Чтобы вернуться сразу, нужно написать `ae_send->()`. Аналогично действует `ae_croak` — только на этот раз `ae_recv` выбросит исключение.

Не буду переводить man-страницу собственного модуля целиком, остановлюсь на интересных, как мне кажется, моментах.

Таймаут

Обязательный аргумент — `timeout` или `soft_timeout`. Он не может быть равен нулю во избежание разночтений. Если таймаут все-таки не нужен, следует использовать отрицательное значение (“Yes, do as I say”).

`soft_timeout` означает, что в случае его достижения будет просто возвращено

`undef`, без `die`.

Изоляция вызовов

Все генераторы коллбэков работают только внутри `ae_recv`, иначе умирают. Более того, коллбэк запоминает «свой» событийный цикл и отказывается работать в последующих, генерируя только warning. Умирать под управлением AnyEvent нежелательно, поэтому так.

При желании все сообщения об ошибках такого рода можно посмотреть в переменной `@AE::AdHoc::errors`.

Например,

```
1 use strict;
2 use warnings;
3
```

```
use Test::More;
use AE::AdHoc;

use My::Module;

my $object = My::Module->new;

my $result = ae_recv {
    $object->do_something_async(
       ae_send ); # takes 5 sec
} soft_timeout => 3;
is ($result, undef, "Not finished
    yet");

$result = ae_recv {
    # just wait
} soft_timeout => 3;
is ($result, undef, "Not finished
    either");
is (scalar @AE::AdHoc::errors, 1,
    "1 runaway callback detected"
   );

```

```
22 note join "\n", @AE::AdHoc::
       errors;
23
24 done_testing;
```

Здесь результат `do_something` не вернется никогда. Хотя `do_something`, конечно, доделается.

Множественные задания

Во время обсуждения будущего модуля в рассылке moscow.pm меня попросили добавить примитивы для `begin/end`, что и было сделано. Однако на практике с `begin/end` слишком легко ошибиться на один.

Поэтому была реализована дополнительная функция `ae_goal("identifier")`. Каждый такой вызов создает «задание» и

возвращает коллбэк, который в свою очередь помещает в специальный хеш ссылку на массив аргументов (@_). Когда/если все задания выполнены, производится вызов ae_send.

Результат можно посмотреть в функции AE::AdHoc->results;

Вот это уж точно следовало бы делать через future/promise. См. также модуль Async::MergePoint.

Вот такой вот скрипт, например, опрашивает параллельно насколько адресов вида host:port на предмет открытости порта. Обратите внимание на то, что здесь полностью отсутствует упоминание Test::More и вообще каких-либо тестов.

```
1 use warnings;
2 use strict;
```

```
3
4  use AE::AdHoc;
5  use AnyEvent::Socket;
6  use Data::Dumper;
7
8  my ( $timeout, @probe ) = @ARGV;
9  $timeout or die "Usage: $0 <
       timeout> <host:port> ...\n";
10
11 ae_recv {
12     foreach (@probe) {
13         warn "Trying $_...\n";
14         /(\S+):(\d+)/ or die "
               Wrong format, must be
               'host:port': $_";
15     tcp_connect $1, $2, ae_goal(
           $_);
16     };
17 } soft_timeout => $timeout;
18
19 my $ready = AE::AdHoc->results;
20 print Dumper($ready);
21 foreach my $host (keys %$ready) {
```

```
22      # skip rejected hosts
23      ref $ready->{$host}->[0] and
            print "Port open: $host\n"
            ;
24  };
```

Заключение

В заключение хотелось бы сказать, что написание самих тестов все-таки менее важно с технической точки зрения, чем написание кода, хорошо поддающегося тестированию. Я перепробовал не так много способов. Например, вообще не пробовал Coro — хотелось бы увидеть комментарий на эту тему.

Наиболее перспективным, однако, мне показался подход, при котором бизнес-логика по максимуму выносится в чистые

функции, то есть такие, которые возвращают значение, зависящее от входных параметров, и больше никак себя не проявляют. *Вот короткая, но емкая заметка насчет рефакторинга такого рода.*

А доля ветвления в асинхронных вызовах и, как следствие, потребность их в тестировании сводится к минимуму.

Но это тема для отдельной статьи.

Благодарности

- Евгению Понизовскому — за приобщение к миру AnyEvent.
- Вадиму Власову — за бета-тест модуля и обратную связь.
- Алексею Шрубу — за то, что модуль

был-таки выложен на CPAN, а не остался на личном жестком диске.

- Ивану Петрову — за замечание о begin/end.
- Корректору Наталье Витько.

Приложение: My/Module.pm

Все-таки приятнее работать с кодом, чем с псевдокодом. Поэтому вот модуль для превращения примеров в полноценные скрипты.

```
use warnings;
use strict;

package My::Module;

sub new { return bless {}, shift
    };
```

```
sub do_something { return 42; };

use AnyEvent;

sub do_something_async {
    my $self = shift;
    my $code = shift;

    my $timer;
    $timer = AnyEvent->timer(
        after => 5,
        cb    => sub {
            undef $timer;

            $code->(42);
        }
    );

    return;
}

use Test::More;
use parent qw( Exporter );
```

```
30 our @EXPORT = qw( ok_result );
31 sub ok_result { is shift, 42, "
       Don't panic" };
32
33 1;
```

■ *Константин Уварин*

4 Тестирование интерфейса веб-приложен Применение WWW::WebKit

Если задача тестирования бэкенда веб-приложения решается достаточно просто, то тестирование фронтенда представляет собой уже значительно более ресурсоёмкую задачу, решаемую с применением специализированных программных средств. Данная статья содержит обзор существующих средств тестирования и рассказывает о применении модуля WWW::WebKit.

Selenium WebDriver

Общепринятый на сегодняшний день инструмент тестирования интерфейса веб-приложений — это Selenium WebDriver. Selenium — это открытый проект, вклю-

чающий в себя несколько программных продуктов, в основном написанных на Java, которые позволяют автоматизировать тестирование интерфейса веб-приложений. Selenium открывает тестируемое веб-приложение в экземпляре какого-либо браузера и, отправляя различные команды (открытие URL, движение мыши, нажатие кнопок и т.д.), регистрирует реакцию веб-страницы на эти команды.

Selenium WebDriver имеет удобный API, который позволяет получить доступ к его функциям из приложений, написанных на различных языках программирования. Как правило, используется либо Selenium Server, либо непосредственно WebDriver какого-либо браузера, к которому по сети может подключаться ваше приложение и отправлять команды в JSON-формате по HTTP-протоколу, и получать данные

о реакции браузера. Спецификация этого протокола была принята консорциумом W3C как интернет-стандарт WebDriver. На текущий момент действует уже 12-я версия черновика.

Основные преимущества Selenium WebDriver

- Это интернет-стандарт, с ним знакомы многие QA-команды, тестирующие веб-приложения вне зависимости от используемого языка программирования.
- Можно тестировать веб-приложение на множестве различных актуальных браузеров: Firefox, Chrome и IE.

Недостатки Selenium WebDriver

- Необходимость установленного веб-браузера для выполнения тестов (IE также потребует Windows-инсталляции).
- Временные затраты на запуск браузера для теста.
- Необходимость иметь запущенный графический сервер (или хотя бы виртуальный, например, Xvfb).
- Запущенный экземпляр службы Selenium Server/WebDriver.

Указанные недостатки в основном сводятся к высоким требованиям на ресурсы тестового сервера, особенно в условиях параллельного выполнения тестов.

Perl-модули для взаимодействия с Selenium

Существует несколько Perl-модулей для взаимодействия с Selenium:

- WWW::Selenium — самый старый и известный модуль для работы ещё со старой версией Selenium RC-сервера, протокол которого пока поддерживается в Selenium Server. Дистрибутив имеет в своём составе модуль `Test::WWW::Selenium`, который удобен для непосредственного создания тестов веб-приложения.
- Selenium::Remote::Driver — актуальный модуль для работы с современным API Selenium WebDriver. Также включает в свой состав модуль для

тестирования `Test::Selenium::Remote::Driver`.

Альтернативы Selenium

Phantomjs — это проект на основе движка рендеринга веб-страниц WebKit (QT-порт WebKit), который не требует графического сервера для работы. Phantomjs управляется с помощью скриптов на JavaScript/CoffeeScript, позволяя загружать страницы, делать скриншоты, конвертировать и сохранять страницы в pdf-формате. Скрипт может выполнять код в контексте открытой страницы браузера, позволяя взаимодействовать с DOM-элементами страницы.

Начиная с версии 1.8 Phantomjs включает

компонент GhostDriver, который реализует WebDriver API. Таким образом, появляется возможность работать с Phantomjs непосредственно с помощью существующих средств работы с Selenium WebDriver.

Perl-модуль Wight — это модуль для тестирования веб-приложений, который был создан специально для работы с Phantomjs, используя существующий там интерфейс WebDriver поверх WebSocket.

Можно также упомянуть модуль WWW::Mecha, который использует браузер Firefox с установленным расширением MozRepl, которое позволяет управлять браузером с помощью команд telnet-сессии.

WWW::WebKit — это Perl-модуль, построенный на основе `Gtk3::WebKit` — обвязке к порту WebKitGTK+, используемого в про-

екте Gnome. На устройстве и интерфейсе этого модуля и хотелось бы остановиться в рамках данной статьи.

WWW::WebKit

Модуль `WWW::WebKit` был создан Стефаном Сейфертом (*Stefan Seifert*). Вероятно некоторые знают его как одного из авторов CiderWebmail и CiderCMS, он также выступал в лайтингах третьего дня на YAPC::Europe в Киеве в 2013 г. с докладом о них. Модуль был задуман как альтернатива реализации `WWW::Selenium`, которая вместо использования сервера Selenium использовала бы встроенный движок рендеринга веб-страниц `Gtk3::WebKit`. По этой причине модуль почти один в один копирует API `WWW::Selenium`, что

позволяет обеспечить простой перевод существующих тестов от одного модуля к другому.

Применение встроенного движка рендеринга веб-страниц даёт несколько преимуществ:

- Меньший объём используемой памяти.
- Нет необходимости в дополнительных программных средствах и заботе об их управлении.
- Упрощается параллельный запуск тестов.

Недостатки:

- Требуется виртуальный графический

сервер Xvfb.
- Отображение веб-страниц проверяется только на одном из существующих движков — WebKit.

API WWW::WebKit

```
my $webkit = WWW::WebKit->new(
    xvfb => 1);
$webkit->init;
```

В данном примере создаётся объект `WWW::WebKit`. Параметр xvfb указывает, есть ли необходимость запуска виртуального фреймбуфера Xvfb, что актуально если вы пытаетесь запустить скрипт на сервере без графического интерфейса.

Перед выполнением каких-либо операций над объектом `$webkit` необходимо

выполнить его инициализацию с помощью вызова `init()`. В этом случае будет запущен Xvfb (если требуется) и инициализированы WebKit и Gtk3.

```
$webkit->open("https://www.google.com");
```

Метод `open` загружает веб-страницу по заданному URL. Выполнение программы блокируется до завершения загрузки. К сожалению, никак не обрабатывается ситуация ошибки при загрузке, и скрипт может зависнуть. Пока этот баг не исправлен, можно использовать классическую схему обхода:

```

eval {
    local $SIG{ALRM} = sub { die };
    alarm 5;
    $webkit->open("http://not_correct_url");
```

```
    alarm 0;
};

warn "failed to load URL" if $@;
```

Локатор Загрузив веб-страницу, можно начать использовать некоторые методы для поиска элементов на странице, для изменения состояния элементов форм и т.д. Многие из этих методов оперируют с локатором, который задаёт положение элемента и имеет следующий вид:

```
тип локатора = значение
```

Обрабатываются следующие типы локаторов:

- **label**=текст — поиск текстового элемента с соответствующим содержимым;

- **link**=тест — поиск ссылки с соответствующим текстом;
- **value**=значение — поиск элемента с указанным значением атрибута value;
- **index**=значение — поиск элемента option с заданным номером позиции;
- **id**=идентификатор — поиск элемента по идентификатору id;
- **css**=селектор CSS — поиск по CSS-селектору;
- **class**=класс — поиск по имени класса;
- **name**=имя — поиск по атрибуту name элемента;
- **xpath**=выражение xpath — поиск по заданному XPath-выражению.

Если тип селектора не указан, то по умолчанию считается, что задано выражение XPath.

Примеры использования локатора:

```
# Получить значение элемента с
    атрибутом name = "q"
$webkit->get_value('name=q');
```

Получение данных со страницы Список методов, реализованных в `WWW::WebKit` для получения текстовой информации:

- `get_text($locator)` — получить значение текстового элемента;
- `get_body_text()` — текстовое содержимое `<body>`;
- `get_title()` — заголовок страницы;
- `get_value($locator)` — значение элемента;
- `get_attribute($locator)` — значение атрибута;

- `get_html_source()` — полный исходный код страницы.

События мыши `WWW::WebKit` позволяет генерировать события мыши:

```
# Клик на кнопке с именем btnG
$webkit->click("name=btnG");
# Ожидание загрузки страницы (или
    пока не ёпройдт 5 секунд)
$webkit->wait_for_page_to_load
    (5000);
```

В данном примере происходит клик мыши по элементу, после чего происходит ожидание загрузки страницы с таймаутом 5 секунд. Стоит отметить, что если перезагрузка документа не происходит, например, выполнен AJAX-запрос, то необходимо использовать функцию `wait_for_pending_requests`:

```
# Клик на кнопке с id="someid"
$webkit->click("id=someid");
# Ожидание выполнения ajax-запроса (или пока не ёпройдт 5 секунд)
$webkit->wait_for_pending_request(5000);
```

Кроме клика существуют также следующие методы:

- `mouse_over($locator)` — мышь движется поверх элемента;
- `mouse_down($locator)` — нажата левая кнопка мыши;
- `mouse_up($locator)` — отпущена левая кнопка мыши;
- `native_drag_and_drop_to_position(...)` — перемещение элемента к заданной позиции;

- `native_drag_and_drop_to_object(...)` — перемещение одного элемента на другой.

События клавиатуры Для генерации нажатий клавиш используются средства модуля `X11::Xlib`, поскольку имеющихся возможностей самого Gtk3 оказалось недостаточно.

```
# установка значения поля элемента с именем "q"
$webkit->type("name=q", "hello world");

# тоже самое, но с генерацией нажатий кнопок
# клавиатуры для каждого символа строки
$webkit->type_keys("name=q", "hello world");
```

Выполнить нажатие отдельной клавиши можно с помощью метода `key_press()`.

Операции с формами Для работы с формами реализовано несколько методов.

- `submit($locator)` — отправка формы по заданному локатору:

```
1 $webkit->submit("id=gbdf");
2 $webkit->
      wait_for_page_to_load
      (2500);
```

- `select($select_locator, $option_l` `)` — выбор опции в элементе `<select >`;

- `check($locator)` — установка “галочки” элемента checkbox;

- `uncheck($locator)` — снятие “галочки” с элемента checkbox.

API Test::WWW::WebKit

Был создан модуль `Test::WWW::WebKit`, предназначенный для построения тестов с использованием `WWW::WebKit`. Назначение тестовых функции интуитивно понятно, название многих методов построено из названия метода родительского класса и суффикса `_ok` или `_is`. Пример теста с отправкой формы и ожидания ответа:

```
use Test::WWW::WebKit;
my $webkit = Test::WWW::WebKit->
    new(xvfb => 1);
$webkit->init;

# Загрузить страницу
```

```
$webkit->open_ok("http://www.
   google.com");

# Ввести текст в поле формы
$webkit->type_ok("name=q", "hello
    world");

# Успешная отправка формы
$webkit->submit_ok("id=gbqf");

# Ожидание ответа
$webkit->
   wait_for_pending_requests
   (5000);

# Заголовок совпал с ожидаемым
$webkit->title_is("foo");
```

Заключение

Надеюсь, данный обзорный материал дал вам представление о том, какие средства применяются для тестирования UI веб-приложений. Каждое средство имеет свои преимущества и недостатки, а также сферу применения. На мой взгляд, `WWW::WebKit`, несмотря на определённую недоработанность и незрелость, заслуживает внимания как удобная легковесная среда тестирования. WebKitGtk+ имеет широкий спектр возможностей, включая и создание скриншотов, и возможностей конвертации веб-страниц в pdf. `WWW::WebKit` вполне может быть доработан для включения этих возможностей, в соответствии с декларируемой совместимостью с API `WWW::Selenium`.

■ *Владимир Леттиев*

5 Обзор CPAN за апрель 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 240
- Новых выпусков — 875

Новые модули

- WWW::HKP

Модуль `WWW::HKP` — это реализация клиента протокола HTTP Keyserver Protocol

(HKP), который может использоваться для получения информации или отправки на сервер OpenPGP-ключей.

- MetaCPAN::Client

`MetaCPAN::Client` — это с чистого листа переписанный `MetaCPAN::API`, который теперь становится официальным клиентом для API MetaCPAN. В модуле используется объектная система `Moo` и `Search::Elasticsearch` как бэкенд для сложных поисковых запросов.

- PLON

Модуль PLON предназначен для сериализации объектов в Perl-код. Для десереализа-

ции достаточно использовать `eval()`. В отличие от `Data::Dumper` модуль `PLON` имеет схожий с `JSON` интерфейс и по умолчанию не экранирует многобайтовые символы (поддерживается только кодировка UTF-8).

- Throw

Ещё один легковесный модуль для создания исключений. В `$@` помещается объект `Throw`, который содержит сообщение об ошибке и дополнительные данные, которые вы передали функции в виде хеша. `Throw` содержит функцию `classify` для классификации типа ошибки, что может быть удобно при работе совместно с `Try::Tiny` и подобными модулями. Это выгодно его отличает от `Acme::Throw`, в котором только один тип ошибки:

```
1 □
2 (□□□□°° □□□
```

- Exception::Chain

`Exception::Chain` — это тоже модуль для создания исключений. Особенностью данного модуля является возможность создания вложенных исключений, когда одно исключение вызывает цепочку последующих исключений, каждое из которых содержит какую-либо дополнительную информацию об ошибке. При классификации ошибки модуль может обходить всю цепочку исключений для поиска заданного типа (типов) ошибки.

- Homer

Ещё один пример того, как в Perl можно сэмулировать любую парадигму программирования. Модуль `Homer` реализует объектную систему, основанную на прототипах, которая применяется, например, в JavaScript. Вместо классов используются только объекты, на основе которых можно создавать другие объекты, добавлять атрибуты, изменять прототипы объектов.

- Method::Cascade

Когда требуется выполнить несколько вызовов методов у объекта, бывает очень удобно объединить их в цепочку вызовов. К сожалению, не все API предоставляют такую возможность. Модуль `Method::Cascade` позволяет выполнить каскадный вызов методов для произвольного API:

```
1       use Method::Cascade;
```

```
    use IO::Socket::INET;

    cascade(IO::Socket::INET->new
       ('google.com:http'))
     ->timeout(5)
     ->setsockopt(SOL_SOCKET,
        SO_KEEPALIVE, pack("l",
        1))
     ->print("GET / HTTP/1.0\r\n\
        r\n")
     ->recv(my $response, 4096);

    print $response;
```

- Cogit

`Cogit` — это реализация интерфейса к git-репозиториям на чистом Perl. Проект является форком `Git::PurePerl`. Автора не устроило, что `Git::PurePerl` основан

на Moose (который не является чистым Perl) и имеет неудачное API. В новом проекте объектная система базируется на `Moo`.

- bare

Модуль `bare` позволяет вам создавать скаляры в виде простых слов без знака сигила $. Подобные скаляры могут использоваться также как и обычные скаляры, им даже можно присваивать значения, но никакой практической пользы, кроме косметического эффекта, это не даёт.

Обновлённые модули

- Starman 0.4009

В новой версии веб-сервера Starman исправлена ошибка, когда сервер отправлял тело ответа в chunked-кодировке при получении HEAD-запроса.

- Mouse 2.2.0

Новый релиз ООП-реализации `Mouse` исправляет несовместимое с `Moose` поведение логического типа `Bool`.

- Math::BigInt 1.9993

Версия модуля на CPAN для работы с большими числами была синхронизирована со всеми последними изменениями дистрибутива в perl blead за последние три года.

- Dancer2 0.140000

В новом релизе веб-фреймворка Dancer2 произошёл переход на семантическое версионирование, что заметно по появившимся четырём нулям в конце номера версии. Многим улучшениям новый релиз обязан Dancer2-хакатону, который прошёл в штаб-квартире Booking.com.

- Search::Tools 1.00

Вышел первый мажорный релиз старых добрых `Search::Tools` — инструмент для создания приложений для поиска. Новая версия теперь использует объектную систему `Moo` и `Class::XSAccessor`.

- Selenium::Remote::Driver 0.20

`Selenium::Remote::Driver` — это Perl-клиент для Selenium WebDriver, инструмента тестирования интерфейса веб-приложений с использованием браузера с поддержкой JavaScript. В новых апрельских релизах появилась поддержка управления профилями Firefox и завершён перевод модуля на использование `Moo`.

- Proclet 0.34

В новой версии супервизора процессов `Proclet` появилась экспериментальная поддержка периодического запуска процессов с узнаваемым API crontab.

- BDB 1.91

Новый релиз модуля асинхронный доступа к Berkeley DB включает исправление для работы на windows с Perl 5.18, декларирована поддержка 6-й версии Berkeley DB.

- HTML::FormFu 2.00

Вышел новый мажорный релиз модуля для создания, рендеринга и валидации HTML-форм `HTML::FormFu`. Появился метод `layout` для задания порядка вывода полей элементов, обновились TT-шаблоны, исправлен вывод предупреждений при работе с perl 5.19.x.

- MetaCPAN::API 0.44

Модуль `MetaCPAN::API` объявлен устаревшим в пользу нового официального клиен-

та `MetaCPAN::Client`.

■ *Владимир Леттиев*

6 Интервью с Кристианом Вальде (Christian Walde)

Кристиан Вальде (mithaldu) — немецкий Perl-программист, автор модулей на CPAN, активный участник Perl-сообщества, автор http://perl-tutorial.org.

Когда и как начал программировать?

Вообще это началось очень рано, когда в возрасте восьми лет в 1991 г. мне дали Amiga, и я начал возиться с командным интерфейсом и ARexx. Ничего особенного, но в дальнейшем я многому научился, особенно благодаря игрушкам Clonk и Starsiege: Tribes, обе были написаны на Си-подобном скриптовом языке и были чрезвычайно настраиваемыми. (У Tribes выпускались аддоны, которые позволяли

игрокам играть в Тетрис, сидя в это время на холме, ожидая целей.)

Позже в колледже я использовал PHP в своих проектах и также C++ и Bash на курсах. В феврале 2005 г. на случайном IRC-канале меня попросили помочь починить их форум (это было что-то вроде 4chan), потому что они умудрились завалить свою SQL-базу. К счастью, к тому времени я постиг искусство чтения ошибок и поиска их в Гугле. Короче говоря, я копался в своей первой перловой программе и быстро поднял сайт.

В результате этого мне выдали рутовый пароль к той машине, который у меня до сих пор есть, и у меня был под рукой сайт с двадцатью тысячами посетителей в сутки, полностью написанный на Perl и MySQL. Из-за этого сайта и помощи от

nrr (http://twitter.com/nrr) на IRC я начал учить Perl. Где-то через три года я написал свою первую Perl-программу, за которую получил деньги (небольшую утилиту для переводов), и еще через год я был нанят на свою первую серьезную работу Perl-разработчиком (группа занималась биоинженерией по анализу и улучшению картофеля).

В кратце: я начал учиться программировать с момента, когда мне подарили первый компьютер и получил большинство знаний за короткий промежуток времени, когда часто пользовался IRC.

Какой редактор используешь?

В основном Komodo IDE. Это мой основной инструмент при работе с Perl, потому как в одном продукте объединены ре-

дактор, визуальный отладчик для Perl и других языков (как Visual Studio), PerlTidy, PerlCritic, проверка синтаксиса Perl, отладчик регулярных выражений, менеджер проектов, grep и самое важное, это настройки по умолчанию. Все это возможно и в других редакторах, я видел у многих людей редакторы с похожими возможностями, но для меня Komodo остается единственным выбором, потому что все эти возможности доступны из коробки и не нуждаются в глубокой настройке, только лишь небольшие изменения для удобства.

В дополнение я использую Notepad++, так как он не требует много памяти и процессорного времени, запоминает когда что было открыто, даже если вываливается, и просто предоставляет быструю возможность подправить текст. Между прочим, у него самая лучшая подсветка синтак-

сиса Perl, и я копирую настройки во все мои Komodo-инсталляции и подстраиваю подсветку для других языков.

С какими другими языками интересно работать?

Мне нравится работать с Javascript, потому как на нем можно написать несколько полезных вещей, и нет лучшего момента, когда я, закончив править очередной JS-файл, наконец могу выкинуть этот отвратительный язык из своей головы.

Кроме этого мне нравится играться с графическим программированием, что включает в себя возню с Си и GLSL. Не думаю, что стоит заострять внимание на Си, но GLSL заслуживает особого внимания, потому что этот язык слегка напоминает Си, но требует особого мышления при написании каж-

дой программы, так как они запускаются на сотне микроядер, и не только параллельно, но и синхронно. Это означает, что ветвления if/else тратят время, вместо сохранения. Это меня радует.

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

Если говорить о языке, то это CPAN. Из всех языков это самый высококачественный архив сторонних библиотек, все это благодаря своей экосистеме качественных утилит, которые нещадно выявляют и собирают проблемы модулей для авторов, делая библиотеки как можно лучше.

Если говорить о perl-интерпретаторе, то его самое большое преимущество, насколько я могу судить, это то, что все его возможности реализованы таким способом, что

удобно писать простые вещи, когда есть только один интерпретатор, но все же они достаточно низкоуровневые, чтобы можно было с помощью CPAN-модулей делать язык удобным для использования. (Например, Moo, IO::All.)

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

Человечность и параллелизм.

Perl уже достаточно хорош сам по себе. Это язык, а не калькулятор. Это язык, который можно читать, на котором можно говорить и думать. И не только это, многое приходит интуитивно, если вы знаете английский язык. Думаю, что у языков программирования должно быть больше таких свойств. Разработчики языков должны делать их более простыми для понимания людьми.

Среднестатистическими людьми.

Параллелизм это второй момент. Технологии упираются в стену, когда идет речь о производительности процессоров. Стало довольно сложно делать процессоры на одном ядре быстрее. Это означает, что нам нужно научиться распределять задачи одной программы между несколькими ядрами, что означает, что языки должны позволять делать это просто. Я слышал, что это хорошо получается у Go, также и у Java. Однако такие скриптовые языки как Python и Perl сильно отстают и должны догонять.

Что не так с индексами популярности языков? Как это можно исправить?

На самом высоком уровне: они не отвечают на вопрос, который хоть сколько

нибудь полезен для людей. Можно задать вопрос “Какой язык самый популярный?”, но почему возникает такой вопрос? Тот, кто его задает, хочет знать, какой язык позволит больше зарабатывать? Стабильно зарабатывать? Какой язык позволит быть на передовой технологического прогресса? Какой язык позволит им быть счастливыми в каждодневной работе? Это вещи, которые волнуют людей, но на эти вопросы не отвечают индексы популярности.

Даже больше того, многие люди не в состоянии правильно прочитать и понять данные индексов популярности. Чтобы смотреть на Google Trends и понимать, что линия, которая идет вниз, все еще может значить, что число пользователей этого языка растет, требует некоторого понимания статистики. Похожим образом, если смотреть на рейтинг TIOBE, нужно

понимать, почему языки с названиями, похожими на другие вещи, выигрывают именно из-за метода, который TIOBE применяет для недопущения такого искажения (если говорить прямо: их метод создает другое искажение).

Как их можно исправить? Честно говоря, я не думаю, что это возможно. Люди, которые создают рейтинги, используют их для споров, и их вполне устраивает, что рейтинги неточные и позволяют им доказывать свою точку зрения, или более точно, заставляют людей кликать на их ссылки и увеличивать прибыль от рекламы. Чтобы исправить эти рейтинги, надо изменить сущности, которые используются для их составления, и на практике это будет похоже на борьбу с ветряными мельницами.

Лучшее, что я могу сказать, когда люди озабочены такими вещами, так это что и Getty сказал в своем выступлении (https://www.youtube.com/watch?v=VFJ672tpRG0): Тратьте свое время не на бессмысленные вещи, а на вещи, которые вы хотите использовать, и которые захотят использовать другие.

Почему до сих пор используешь Win32 и как последнее время Perl себя чувствует на этой платформе?

Как я уже говорил насчет редакторов: настройки по умолчанию. И — Amiga.

Еще ребенком я начал с компьютерной платформы, которая предоставляла практически все то же, что и современная Windows (http://www.gregdonner.org/workbench/images/wb_13.gif). В отличие

от людей, которые начинали на *nix или DOS, у меня в качестве устройств ввода были и клавиатура, и мышь. И до сих пор я предпочитаю окружения, где все мои задачи можно выполнить с помощью и клавиатуры и мыши, позволяя мне делать работу наиболее удобным для меня способом.

Если говорить про настройки по умолчанию: Windows, начиная с Windows 2000, чрезвычайно мало изменилась в плане интерфейса. Если цвета и поменялись, и машины стали мощнее, очень мало изменилось в способах использования. Это означает, что многие разработчики ПО следуют тем же негласным соглашениям. В дополнение, благодаря некоторой извращенности человеческого ума, у закрытого ПО лучше настройки по умолчанию. *nix-разработчики обычно говорят: “Тебе это

нужно? Пожалуйста, реализовывай". Таким образом спасаясь от подгонки программы под нужды других, в то время как у разработчиков закрытого софта нет такой возможности.

Если говорить о том, как Perl чувствует себя на Windows: совершенно отлично. Есть множество людей, использующих *nix-платформы, у которых очень мало опыта работы с Windows или с Perl на Windows, но они с охотой распространяют свои предрассудки как факты. На практике, однако, почти все CPAN-модули работают на Windows, и часто поломки случаются, когда кто-то забывает открывать потоки данных в бинарном режиме. На сегодняшний день все дистрибутивы Perl для Windows поставляются со своим компилятором, существенно упрощая установку XS-модулей, и обычно имеют в

своем наборе некоторые сложные в установке модули. С ActivePerl, Strawberry Perl и DWIMPerl есть возможность выбрать Perl для корпоративной среды, для базовой поставки Perl и для тех, кто хочет иметь практически все модули из коробки.

Почему ты начал сайт perl-tutorial.org? Какова его главная задача, для чего он создавался?

Летом 2011 года первой ссылкой в выдаче на запрос “perl tutorial” была ссылка на урок для Perl 4. Остальные результаты были подобного качества. Это означало, что люди, которые учили Perl, имели высокие шансы учить устаревший Perl и упускали из виду все те позитивные изменения, которые произошли с Perl с 2000 года.

Однако правильным решением не было

написать новый урок. Мы оказались в такой ситуации потому, что Perl уже давно существует, и старые уроки принимались поисковиками лучше, чем современные. http://perl-tutorial.org решает эту проблему будучи публичной wiki-страничкой, которая следит за уроками и поднимает наиболее новые наверх. Таким образом, через десять лет кто-то кликнув на самую верхнюю ссылку не попадет на Perl 5.12.

Играешь ли ты какую-то роль в организации немецких Perl-воркшопов?

Очень несущественную. Здесь в Ганновере у нас очень активная локальная Perl-группа (которая организовалась путем поиска Perl-разработчиков из Ганновера на YAPC::EU 2012), и многие были воодушевлены всем тем, что было в Берлине в 2013 году и поэтому решили организовать следую-

щий воркшоп в Ганновере. Другие люди из группы выполнили львиную долю работы как то поиск места и с помощью Frankfurt.pm организовали всю бумажную работу и составление расписания, в то время как я занимался мелочами, например, помощью уже на месте проведения, или спонсированием 102 плюшевых верблюдов для раздачи посетителям.

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

Последние два с половиной года я работаю в Profihost.com здесь в Ганновере, в основном пишу код для бэкенда их сайта и помогаю другим разработчикам в компании, пока они учат Perl. Недавно я начал заниматься фрилансом, все также работаю на Profihost, но и ищу другую работу для интереса.

Это описывает всю оплачиваемую работу. Все остальное время я трачу на открытые проекты или исправляю некоторые вещи на CPAN, который ломаются на моем тестовом сервере, или реализую новые модули, которые хочу использовать, как, например, недавние улучшения системы ввода-вывода в IO::All. Также я немного провожу времени за собственными личными проектами, такими как 3d-визуализатор для Dwarf Fortress (https://github.com/wchristian/lifevis) (https://dl.dropboxusercontent.com/u/10190786/DF/perl%202012-06-12%2003-07-39-88.jpg https://dl.dropboxusercontent.com/u/10190786/DF/perl%202012-06-12%2003-08-30-52.jpg), OpenGL-шейдер (https://github.com/wchristian/perl_shader_toy) (который может создавать что-нибудь наподобие https://dl.dropboxusercontent.com/u/10190786/your_perl_on_drugs.avi) или новый проект

https://github.com/wchristian/Microidium астероидный мультиплеер, похожий на Luftrausers; что позволяет мне поработать с SDL и написать фреймворк, который смогут использовать другие для написания игр.

Короче говоря, практически ВСЕ свое время я провожу с Perl.

Стоит ли советовать молодым программистам учить сейчас Perl?

Конечно. Даже если они не будут использовать Perl в своей дальнейшей жизни, будь-то по собственному желанию, или из-за обстоятельств, у них будет понимание принципов, которыми не обладает ни один язык программирования, особенно если они будут учить Modern Perl.

Интересно посещать Perl-воркшопы и конференции?

Очень. Даже если я ничего нового не узнаю из докладов (а я узнаю), даже если я не познакомлюсь с новым людьми (я знакомлюсь), даже если я не встречу новичков, которых я могу чему-то научить (и я учу), даже если вечера в пабах не слишком веселые (а они веселые), я всегда возвращаюсь домой с кучей заметок о том, что можно сделать, на что стоит посмотреть, и что особенно важно — с огромной мотивацией что либо делать. Нет ничего более мотивирующего для написания кода, чем проведение выходных на конференции или воркшопе.

Какое пиво наиболее подходит для сближения Perl-разработчиков?

Без всяких сомнений — бельгийское! Я из Германии, которая известна как страна пива, но, честно говоря, люди делают здесь скучную горькую воду. Я ненавидел пиво, пока не познакомился с бельгийским и понял впоследствии, что оно отлично подходит для социальных встреч, так как оно может удовлетворить любой вкус. Даже если в вашей группе есть кто-то, кто обычно не пьет, их скорее всего заинтересует практически безалкогольное фруктовое пиво; для финнов или других сильнопьющих, есть 10% Gulden Draak 9000. Между ними есть множество сортов на любой вкус, включая Gueuze, которое кислое, а не горькое.

Несмотря на все вышесказанное, если такого пива нет, британская палитра сидра тоже вполне подойдет.

■ *Вячеслав Тихановский*