Язык программирования tcl

Автор: Михаил Полушкин (MihailPolushkin@gmail.com)

Оглавление

#1. Введение

Tcl –интерпретируемый язык высокого уровня, обладающий компактным расширяемым синтаксисом. Изначально он предназначался только для встраивания в программы, в качестве скриптового языка, однако давно перерос свое первоначальное предназначение и превратился в самостоятельный универсальный язык, применяемый во многих областях компьютерных технологий.

Прошлое

Tcl появился на свет в конце 80-х. Джон Остераут, его создатель и профессор университета Беркли, в те времена работал над новой экспериментальной операционной системой Sprite. Группа его студентов занималась разработкой ядра системы, в то время как сам Остераут, используя свой новый язык, создавал текстовый редактор и эмулятор терминала для новой ОС.

Тогда Tcl еще не рассматривался как самостоятельное средство разработки программ. Никто не задумывался над тем, что бы писать на нем целые приложения. Однако, за несколько лет с момента выпуска первой версии в 1989 году сообщество программистов Tcl выросло до нескольких сотен тысяч. Практически полное отсутствие альтернативы превратило Tcl в «народного любимца». Каждый программист находит ему собственное применение.

В 1991 году выходит первая реализация Tk – библиотеки для X Window System, которая позволяет не только молниеносно разрабатывать графические интерфейсы на Tcl, но и выводит интеграцию приложений в X Window на качественно новый уровень. С этого момента Tcl массово используется программистами для прототипирования графических приложений, а администраторы имеют возможность добавлять графические интерфейсы к своим скриптам.

В 1993 году появляется первое объектно-ориентированное расширение для Tcl – incr Tcl. К этому времени язык уже воспринимается как вполне самостоятельный и пригодный для создания «Pure Tcl» программ, написанных только на Tcl.

Год 1994 был отмечен двумя примечательными событиями. Вопервых, в этом году Джон Остераут покинул университет Беркли и перешел работать в SunScript – отделение компании Sun Microsystems. Во вторых Ричард Столлмен опубликовал статью «Почему вы не должны использовать Tcl».

Врядли эта статья повлияла на мнение тех, кто к тому времени уже использовал Tcl, однако авторитетное для новичков и всего сообщества GNU мнение Столлмена повлияло на популярность этого языка. Трудно сказать, что стало причиной столь резкой критики, ибо доводы Столлмена были не убедительны. Видимо статья писалась после очень краткого знакомства с этим языком.

Период Sun отметился бурным развитием Tcl. Интерпретатор стал кросплатформенным и теперь программы на Tcl могли исполняться под множеством операционных систем. Кроме интерпретатора разрабатывались и программы на самом Tcl. Это были HTML редакторы, WEB-серверы, почтовые клиенты и множество другого программного обеспечения. Скрипты на Tcl входили в состав операционной системы Sun Solaris.

Но Sun параллельно спонсировала многие, часто конкурирующие проекты. Практически параллельно с Tcl велась разработка Java. Не смотря на то, что это были две совершенно разные

технологии, они конкурировали в плане финансирования. Почувствовав то, что Tcl более не будет фаворитом Sun, в 1998-м году Джон Остераут решает создать собственную компанию - Scriptics, которая продолжит развитие Tcl. Sun с одобрением отнеслась к такому решению и даже подарила Scriptics первый сервер.

В этот же год Джон Остераут пишет статью «Сценарии: высокоуровневое программирование для XXI века», в которой раскрывает свою точку зрения на скриптовые языки и их место в мире информационных технологий. По его мнению, новое поколение языков должно не столько ориентироваться на непосредственную работу с данными, сколько объединять инструментальные средства операционной среды.

Следующим большим шагом Tcl явилась версия 8.0, выпущенная в 1999 году. Интерпретатор был переработан коренным образом и теперь транслировал исходный текст программы в промежуточный байт код. В результате скорость интерпретатора увеличилась в 6 раз и Tcl наконец то освободился от дурной славы самого медленного языка.

Последним, на сегодняшний день, важным событием в жизни проекта Tcl стала передача его сопровождения свободному сообществу программистов. В 2000 году была образованна группа Tcl Core Team (TCT), которая занялась поддержкой языка, от которого отказался новый владелец перекупивший Scriptics.

Последние десять лет развитие Tcl было плавным. Не в том смысле, что замедлилась разработка, а в том, что более не было передач языка новым владельцам. С каждой новой версией Tcl становился все быстрее, надежнее, добавлялись новые команды, улучшался внешний вид виджетов tk, разрабатывались новые библиотеки.

Так как Tcl неразрывно связан с именем его создателя, то в конце этой истории считаю необходимым упомянуть, что сам Джон Остераут в 1987 году был удостоен премии ACM имени Грейс Мюррей Хоппер. Эта премия вручается молодым ученым за особый вклад в развитие компьютерной науки. Так же ее получали, наряду со многими выдающимися программистами, Дональд Кнут и Бьерн Страуструп.

В 1997 году ему была вручена награда ACM Software System Award за язык Tcl. Наряду с ним награду получали: Ричи и Томпсон за UNIX, Дональд Кнут за TeX, разработчики TCP/IP, WWW и нескольких других технологий, определивших историю человечества.

Настоящее

Текущая стабильная версия Tcl/Tk – 8.5.8, но для Minix3 на данный момент доступна только версия 8.4.14. Они не различаются коренным образом и обратно совместимы.

Приятно и то, что большинство «Pure Tcl» программ полностью кросплатформенные. Лишь некоторые, использующие особенности операционных систем, не могут запускаться под другими ОС. А это значит, что перенос Tcl под Minix3 добавляет в неё всё богатство мира Tcl.

Из таких программ особенно следует отметить библиотеку tcllib, которая реализует все популярные сетевые протоколы (ftp, http, smtp, ntp, pop3…), предоставляет инструменты для работы с алгоритмами шифрования, разными форматами изображений, html парсер и даже пакеты для документирования исходных текстов в стиле литературного программирвоания Дональда Кнута. Эта библиотека на самом деле настоящий клад разработчика и системного

администратора. Практически на лету можно создавать собственные сервисы, работающие в точном соответствии с вашими требованиями, с легкостью дополняя их графическим интерфейсом Tk.

Вместе с tcllib распространяется и библиотека tklib, которая включает виджеты с расширенным функционалом. В ней можно найти, например, виджет текстового редактора ctext, который умеет нумеровать строки и подсвечивать синтаксис по заданным регулярным выражениям, очень удобный виджет таблицы, а так же еще пару десятков других расширений над Tk.

Интересен факт низкой популярности этого языка среди начинающих программистов. Особенно в России. Скорее всего, это связано с практически полным отсутствием литературы по Tcl технологиям. По самому языку на русском было выпущено две книги, из которых внимания заслуживает только одна.

Tcl не привлекает новичков. Они не умеют копать документацию, изучать библиотеки экспериментальным методом. Но те, кто знаком с C/C++, sh, хотя бы поверхностно знает, что такое Lisp, имеет хоть какой-то опыт программирования, а еще лучше хорошую теоретическую основу, Tcl дается без особого труда. Сам автор этой статьи был вынужден изучить основы синтаксиса языка буквально за пару дней по нескольким примерам программ и лишь через полгода узнал, что это был Tcl.

В виду низкой популярности среди масс язык не пользуется популярностью и у мелкого бизнеса, который не может позволить себе такой экзотики как квалифицированный Tcl программист. Незаменимость такого человека – вполне объективный риск для всего проекта. Поэтому они предпочитают технологии, пользующиеся массовой популярностью, для которых уже отлажена вся технология производства программного обеспечения и обучены десятки тысяч разработчиков.

Его почитатели – это в основном крупный бизнес (IBM, ActiveState, ORACLE, AOL), Open Source проекты (Tkabber и DejaGNU) и наукоемкие проекты, вроде GRASS GIS.

В общем, перечислить всех областей применения Tcl просто не реально, однако можно выделить основные:

- короткие десятистрочные админские скрипты (в том числе с использованием expect);
- фронтенды и прототипы приложений;
- тестирование приложений (тот же DejaGNU);
- полноценные пользовательские приложения (Tkabber);
- CGI скрипты и WEB-приложения (OpenACS);
- встраивание в приложения в качестве скриптового языка;
- аплеты для WEB браузеров (так же как и Java аплеты);
- IRC боты.

Будущее

Основным нововведением текущей бета версии 8.6 является встроенная в ядро система объектно-ориентированного программирования XOTcl. Аналогично Lisp CLOS эта система не является неотъемлемой частью языка, а реализуется отдельно в виде библиотеки.

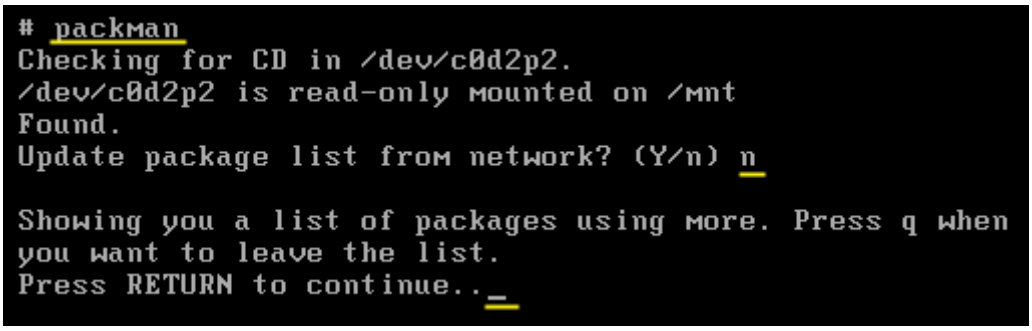
Если говорить про долгосрочную перспективу, то Tcl будет становиться все более универсальным языком 4-го поколения. Уже версия 8.6 по объектно-ориентированным возможностям ничем не будет уступать языку Python, а так же будет поддерживать парадигму аспектно-ориентированного программирования.

Взрывной рост количества программистов не является следствием повышения уровня IQ общества, поэтому скриптовые языки будут играть все большую роль при разработке информационных систем, как более простые для изучения. А Tcl как нельзя лучше подходит для «непрограммирующих программистов».

#2. Установка пакетов tcl и tk

Как уже говорилось выше, Tcl/Tk версии 8.4.14 имеется в стандартной поставке MInix3. А это значит, что установка настолько же элементарна, как и для любого другого программного обеспечения Minix3. Если вы знаете, как устанавливать программы с помощью packman, то можете не читать этот раздел, и установить пакеты Tcl и Tk самостоятельно.

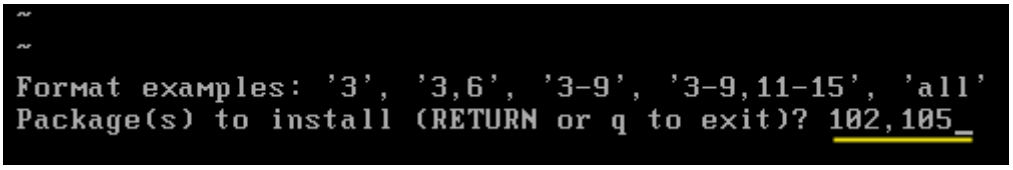
Для установки программ с CD диска он должен находиться в приводе. Далее набираем команду packman, которая сообщает нам, что найден CD диск и предлагает поискать обновленные пакеты в сети. Мы отказываемся, набрав в ответ «n» и нажав Enter.



Программа сообщает нам, что далее будет выведен список доступных пакетов и предлагает нажать Enter, что мы и делаем.

С помощью пробела пролистываем список и находим в нем пакеты tcl и tk. Запоминаем их номера.

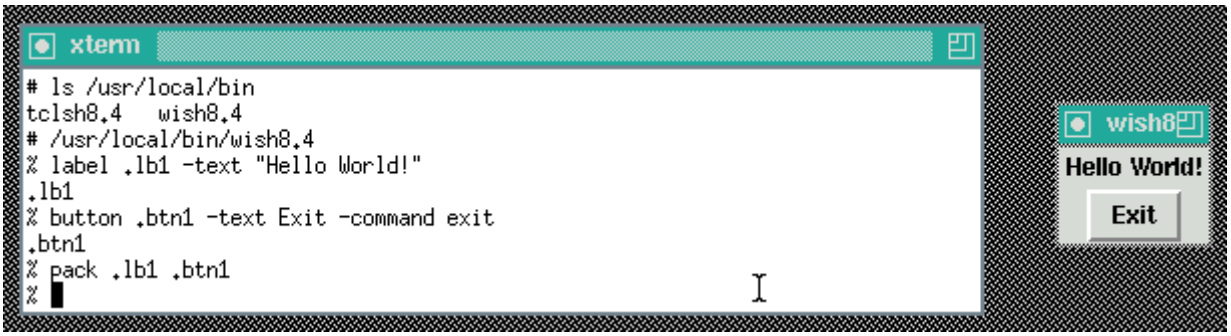
Нажав q, или пролистав до конца списка, вводим через запятую номера пакетов и опять нажимаем Enter.



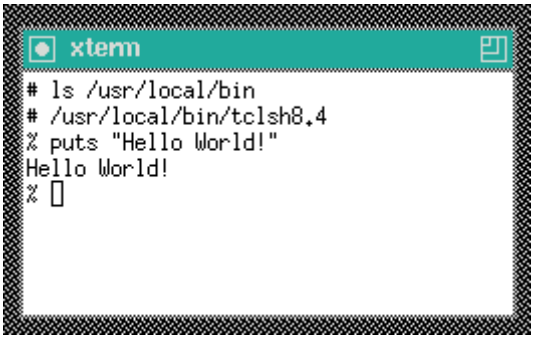
Теперь собственно про то, что же мы установили. Пакет **tcl8.4.14** содержит ядро языка и оболочку **tclsh8.4**. Это консольная оболочка, которая не поддерживает графических интерфейсов Tk, если не указать явно, что мы хотим использовать пакет Tk. Пакет **Tk8.4.14** содержит библиотеки Tk и шелл **wish8.4**, который специализирован для исполнения программ с графическим интерфейсом Tk.

Ниже традиционные примеры программ «Hello World» написанные с использованием wish и tclsh в интерактивном режиме. То есть, когда пользователь пишет Tcl команды непосредственно в окне шелла.

Wish создает новое окно, в котором размещает создаваемые виджеты.



Tclsh выводит текст прямо в той же консоли, в которой вы пишете команды.



Используйте интерактивный режим, когда вам нужно протестировать какой-то участок кода, или просто попрактиковаться в языке. Но большую практическую ценность имеет пакетный режим работы интерпретатора. В этом режиме при запуске интерпретатора ему передается текстовый файл с исходным текстом программы на Tcl.

Что бы сделать файл скрипта исполнимым, вопервых нужно разрешить операционной системе его исполнять. Для этого в командной оболочке нужно набрать строку:

```
chmod +x file.tcl
```

А в самом файле первая строка должна иметь следующий вид:

```
#!/usr/local/bin/wish8.4
```

Или так:

```
#!/bin/sh
# \
exec wish "$0" ${1+"$@"}
```

Последний вариант позволяет не указывать пути к интерпретатору, который может быть различным в разных операционных системах. Вместо этого сначала запускается **sh**, который ищет wish по всем возможным путям, находящимся в переменной *$PATH*, и запускает его с помощью команды *exec*. Вся хитрость заключается в том, что в Tcl символ обратной косой черты (или бекслеш «\») экранирует перевод строки, и третья строка, содержащая вызов exec, в Tcl считается продолжением закомментированной второй строки. Проще говоря, **sh** видит третью строку, а **wish** и **tclsh** считают ее закомментированной.

#3. Язык Tcl

Как вы помните из введения, язык Tcl был разработан не простым программистом-экспериментатором, а доктором наук. То есть, человеком, который изучил опыт предыдущих похожих разработок (Lisp, sh, шелла ОС Multics…) Досконально, с теоретической стороны знал их преимущества и недостатки. С другой стороны разработка нового языка была продиктована чисто практическими мотивами, а не академическим интересом. В результате получился язык с удобным в использовании и логически законченным синтаксисом.

При описании синтаксиса Tcl используется два подхода. Первый более простой. Он описывает синтаксические правила языка. То есть, как нужно писать программу. Второй более точный, но требующий большей теоретической подготовки. Он описывает процесс разбора программы интерпретатором.

Дело в том, что Tcl имеет контекстно-зависимый синтаксис и его нельзя полностью описать с помощью правил и можно писать такие программы, исходный текст которых будет мутировать по ходу исполнения. Однако, большая часть программистов привыкла именно к описанию грамматики языка.

На начальном этапе такой подход ничему не вредит. Просто он не раскрывает всей сути языка. А со временем, практикой применения языка в реальных проектах и экспериментах понимание сути языка придет само собой. Либо, вы просто прочете это в какой-либо другой статье. Эта же статья рассчитана на широкий круг программистов, от новичков (если так можно назвать пользователей Minix3) до профессиональных программистов, незнакомых с Tcl. Ее предназначение, прежде всего, публицистическое и уже потом информационное. Заинтересованные начнут более подробное изучение языка и будут перелопачивать сеть, а кому-то и такого описания, для общего представления языка будет достаточно.

Основы синтаксиса

Исходный текст программы на Tcl состоит из последовательного вызова команд. Каждая команда отделяется от следующей переводом строки или точкой с запятой.

```
puts "Oh!"
puts "Hello World!"; puts "I am Tcl programm"
```

Команда ***puts*** печатает переданную ей строку на консоль. Если хотите, можете запустить **tclsh** или **wish** и пробовать все встречающиеся далее примеры.

Параметры команды разделяются пробелами. Если параметр не содержит внутри себя пробелов, то он не нуждается ни в каких ограничителях вроде кавычек.

```
puts Hi!
--> Hi!
```

Однако если параметр содержит в себе пробелы, или переводы строки, то программист должен явно указать его границы, что бы интерпретатор понял, где начинается параметр и где он

заканчивается. Явно указать границы параметра можно двумя способами: с помощью кавычек и с помощью фигурных скобок.

```
puts "Двери настежь у вас, а душа в заперти
Кто хозяин здесь? Напоил бы вином ..."

--> Двери настежь у вас, а душа в заперти
Кто хозяин здесь? Напоил бы вином

puts {А в ответ мне: Видать, был ты долго в пути
И людей позабыл - мы всегда так живем}

--> А в ответ мне: Видать, был ты долго в пути
И людей позабыл - мы всегда так живем
```

Если параметр представляет собой просто текст, то большой разницы между двумя этими методами нет. Но кроме обычного текста в качестве параметра может быть указана другая команда в квадратных скобках:

```
set переменная "Значение переменной"
puts [set переменная]
--> Значение переменной
```

Эта команда выполнится, и ее результат будет подставлен вместо нее в качестве параметра.

Команда ***set*** устанавливает и возвращает значения переменных. Если ей передано только имя переменной, то она возвращает ее значение, установленное прежде. Если же передано имя переменной и новое значение, то оно присваивается переменной и команда set возвращает уже новое значение переменной.

Так как Tcl полностью поддерживает Unicode, то кириллицу (или даже иероглифы) можно использовать в любой части программы. Вопрос русификации X Window System, или EDE – вопрос отдельный, поэтому тут мы его рассматривать не будем.

Теперь о различиях в группировке кавычками и фигурными скобками.

Группировка кавычками разрешает подстановку вложенных команд. В этом случае, их результат вставляется в параметр вместо команды:

```
puts "Какая ОС вам более всего нравится?"

set osName [gets stdin]
<-- Minix3

puts "Да, [set osName] отличная ОС!"
--> "Да, Minix3 отличная ОС!"
```

Команда ***gets*** считывает с указанного входного канала текстовую строку и возвращает ее. В данном случае был указан канал *stdin*, который по умолчанию связан с клавиатурой. Значение прочитанной строки было присвоено переменной *osName*, а затем подставлено в третьей строке кода как часть параметра команды ***puts***.

В wish команда ***gets*** с каналом *stdin* работает не корректно, так как **wish** не предназначен для работы с консолью. Тестируйте этот пример в **tclsh**.

Группировка фигурными скобками запрещает подстановку вложенных команд. То есть, в фигурных скобках текст никогда не изменяется.

```
puts {Да, [set osName] отличная ОС!}
--> Да, [set osName] отличная ОС!
```

Группировка без подстановки нужна, когда мы хотим передать параметр команде без изменений. Это часто требуется, когда не желательна «ранняя подстановка». Для примера рассмотрим команду цикла ***while***:

```
set i 0
while {[set i] < 3} {
    puts "Номер [set i]"
    incr i
}
--> 0
1
2
```

Команда ***while*** принимает два параметра: условие повторения цикла и тело цикла. В нашем случае каждый из параметров заключен в фигурные скобки, поэтому передается команде ***while*** без изменений. Команда ***while*** на каждой итерации цикла сама подставляет в условии повторения цикла новое значение переменной i, приращенное командой ***incr*** (тоже что и оператор ++ в Си).

Но если бы мы написали:

```
set i 0
while "[set i] < 3" {
    puts "Номер [set i]"
    incr i
}
--> 0
1
2
3
4
…
```

То значение переменной *i* было бы подставлено один раз интерпретатором, еще до передачи аргументов команде ***while***. В этом случае ***while*** постоянно будет проверять условие "*0 < 3*" и цикл будет бесконечным. Даже не смотря на то, что команда ***incr*** будет приращивать переменную *i* (она же уже не проверяется на каждой итерации цикла).

В Tcl, как и во многих скриптовых языках можно использовать «долларовую подстановку» значений переменных. То есть, вместо *[**set** i]* можно просто писать *$i*. Например:

```
set i 0
while {$i < 10} {
    puts "Номер $i"
    incr i
}
```

Эти две формы подстановки значения переменной полностью равнозначны, хотя и различаются внешне.

Немного практики

Tcl имеет простой синтаксис. Однако, что бы разбираться в законченных программах, знания одного синтаксиса недостаточно. Нужна практика чтения и написания программ. Поэтому в данном разделе мы разберем одну небольшую, но реально работающую программу.

Программа наша будет брать список IP адресов из файла, и вызывать для каждого из них утилиту внешнюю ping. Кроме того, она будет позволять добавлять и удалять IP адреса из файла.

Создадим новый файл, который назовем «autoping.tcl», и откроем его в редакторе. Это будет файл исходного кода нашей программы. Кроме того, будет файл ~/.aping, который программа создаст сама и будет там хранить список тестируемых IP адресов.

Первым делом выделим основные функции, которая должна выполнять программа:

1. добавлять IP адрес к списку в файле ~/.aping, если командная строка содержит опцию –a со следующим за ней IP адресом;
2. удалять IP адрес из файла ~/.aping, если командная строка содержит опцию –d со следующим за ней IP адресом;
3. пинговать все адреса этого списка, если командная строка не содержит никаких опций.

Начнем по порядку и напишем сначала процедуру, которая добавляет IP адрес к списку в файле.

```
proc addIp {file ip} {
      set fd [open $file a]
      puts $fd $ip
      close $fd
}
```

В этом кусочке кода много новых для Вас команд. Ниже мы разберемся с их назначением, но более подробную информацию по всем параметрам вам придется искать в документации к этим командам. Остановимся только на самом интересном.

Команда ***proc*** – создает новую процедуру, к которой можно дальше обращаться как к обычной Tcl команде. Она принимает три параметра – имя процедуры (в данном случае ***addIp***), список формальных параметров процедуры (два параметра - *{file ip}*) и тело процедуры (все остальное).

Новая процедура ***addIp*** будет получать два параметра: имя файла и ip адрес. Внутри процедуры эти параметры имеют соответствующие имена *file* и *ip*. Конечно, это формальные имена, при вызове команды значение имеет только их порядок (как и в любом другом языке).

Первая строка тела процедуры открывает файл и получает его дескриптор. Вложенная команда ***open*** открывает файл и возвращает его описатель, а ***set*** записывает полученный от ***open*** идентификатор потока файла в переменную *fd*. В дальнейшем этот описатель используется во всех операциях с файлом для ссылки на него. Параметр *a* указывает, что файл должен открываться на дополнение.

Далее идет уже знакомый вам ***puts***. Первый параметр команды ***puts*** – идентификатор потока. Если он не указан, то он считается равным *stdout*. По этому, по умолчанию текст выводится в консоль. Далее идет сам выводимый текст. В нашем случае – это значение переменной *ip*.

Команда ***close*** закрывает файл по его идентификатору.

Следующая процедура удаляет IP адрес из файла:

```
proc delIp {file ip} {
      set fd [open $file r]
      while {![eof $fd]} {
            lappend ipList [gets $fd]
      }
      close $fd

      set fd [open $file w ]
      foreach curip $ipList {
            if {$ip ne $curip  && $curip ne ""} {puts $fd $curip}
      }
      close $fd
}
```

Сначала весь файл читается в переменную (список) *ipList*. Затем, файл открывается на запись с уничтожением всего содержимого. В цикле каждый IP адрес сравнивается с удаляемым и если он ему не равен, то записывается в файл.

Таким образом, в файл будут записаны все старые адреса, кроме удаляемого.

Команда ***gets*** читает строку из файла с идентификатором *$fd* и возвращает ее команде ***lappend***. Команда ***lappend*** предназначена для добавления строк к списку. Список можно представлять себе как обычный массив строк, хотя, если вы собираетесь изучить Tcl, то на эту тему нужно обратить особое внимание. Нельзя сказать, что она сложная, просто есть свои нюансы.

Далее файл закрывается, а переменная *fd* используется повторно, для открытия этого же файла, но уже на запись.

Цикл ***foreach***, тоже встречается во многих языках. Он проходит все элементы списка *ipList*, поочередно присваивая их значение переменной *curip*. Внутри цикла команда ***if*** проверяет, не равен ли текущий IP (*curip*) удаляемому (*$ip ne $curip*) и пустой строке (*$curip ne “”*), и только в этом случае выводит текущий IP в файл (*puts $fd $curip*).

Теперь, собственно, процедура пинга:

```
proc aPing {file} {
      set fd [open $file r]
      while {![eof $fd]} {
```

```
            lappend ipList [gets $fd]
      }
      close $fd

      foreach curip $ipList {
            if {$curip ne ""} {
                  if {[catch {set re [exec ping $curip]} errMsg]} {
                        puts "$errMsg ($curip)"
                  } else {
                        puts $re
                  }
            }
      }
}
```

Первые строки полностью идентичны предыдущей процедуре. Цикл считывает все IP адреса в список *ipList*, только теперь мы будем их поочередно передавать утилите **ping**.

В цикле ***foreach*** каждый ip адрес из списка *ipList* поочередно помещается в переменную *curip*. Затем проверяется, не является ли *curip* пустой строкой. В принципе, без этой проверки ничего страшного не произойдет, однако, обращение к **ping** происходит дольше, чем это сравнение. Мы просто экономим время на лишних вызовах внешней утилиты.

Команда ***catch*** контролирует исполнение участка кода и при возникновении ошибки возвращает 1 и помещает описание ошибки в *errMsg*. В нашем случае ошибкой будет считаться состояние, когда компьютер с указанным адресом не отвечает, или указана недоступная сеть. В случае корректной работы catch возвращает 0.

Контролируемый участок кода (***set*** *re [**exec** ping $curip]*) вызывает утилиту **ping** командой ***exec*** и передает ей очередное значение IP адреса. Команда «***set*** *re …*» записывает возвращенный **ping**`ом текст в переменную *re*.

В результате, если возникла ошибка, то выводится сообщение об ошибке и IP адрес, при обращении к которому она произошла (***puts*** *"$errMsg ($curip)"*), иначе выводится положительный ответ **ping** о том, что хост включен в сеть.

Фух! Вы запутались? Это только первое знакомство с языком. Сейчас просто важно понять идеи, общую структуру кода на Tcl. Если вы хотя бы понимаете ход мысли, то все нормально. Если же вам все понятно, то поздравляю. Несколько недель практики и у вас будут получаться отличные скрипты.

Для разрядки простенькая процедура ***usage***, которая сообщает пользователю правильный формат командной строки:

```
proc usage {} {puts "USAGE: $::argv0 \[-a|-d ip_address\]"}
```

Два двоеточия – это указание на то, что следует использовать глобальное пространство имен. Точно как в C/C++. Переменная *argv0* хранит имя скрипта, аналогично как элемент массива *argv[0]* в Си хранит имя самой программы.

Символ бекслеша «\» экранирует квадратную скобку, что бы текст в квадратных скобках не был воспринят как вложенная команда.

Теперь нам осталось рассмотреть основную часть программы, которая обрабатывает параметры и вызывает процедуры описанные выше.

```
set afile ~/.aping

if {$argc != 0} {
      if {$argc == 2} {
            switch -- [lindex $argv 0] {
                  -a {addIp $afile [lindex $argv 1]}
                  -d {delIp $afile [lindex $argv 1]}
                  default usage
            }
      } else {
            usage
      }
} else {
      aPing $afile
}
```

Переменная *argc* содержит общее количество параметров переданных скрипту через командную строку. В нашем случае, если параметров передано не было, то вызывается процедура ***aPing***, которой передается значение переменной *afile* (то есть, файл ~/.aping).

Если же количество параметров равно двум, то первый должен быть обязательно либо флагом *–a*, либо *–d*. Если это *–a*, то мы добавляем второй параметр командной строки (который должен быть IP адресом) к содержимому файла. Если *–d*, то удаляем из файла указанный вторым параметром IP адрес. Если в качестве первого параметра указана какая-то другая опция, или количество параметров не равно 2, то вызываем процедуру ***usage***.

Команда ***lindex*** возвращает элемент списка по его индексу. Первым параметром этой команды идет список, а вторым индекс возвращаемого элемента. Если опять же сравнивать Tcl с Си, то это практически обычное обращение по индексу массива, как например, *argv[1]*.

На этом пример заканчивается. Тому же, кто хочет поближе познакомиться с командами Tcl – прямой путь к документации. Ссылки на документацию смотрите в конце статьи.

4. Основы tk

Tk – это библиотека создания графического пользовательского интерфейса. Хотя она была разработана для Tcl и традиционно используется в нем, ее можно использовать и из программ на многих других языках, таких как C, Python, Ruby, Perl и Lua.

Для Tcl существует большое количество других библиотек разработки графического интерфейса, которые, как правило, имеют более широкие возможности, чем Tk. Существуют и библиотеки, расширяющие возможности самой Tk. Однако, для многих задач (особенно, небольших скриптов) мощности этой библиотеки вполне достаточно. Более того, по мнению многих специалистов, она эффективнее для простых задач, чем любая другая.

В ходе разработки графического интерфейса программист вызывает команды, которые создают виджеты, манипулируют ими, привязывают к ним команды обработки различных событий.

Например, что бы создать виджет метку (label) нужно вызвать одноименную команду:

```
label .l –text “Hello World!”
```

В качестве первого параметра передается путь и имя создаваемого виджета. Путь начинается с точки, которая обозначает главное окно приложения и включает все фреймы, в которые вложен виджет. Имя – текстовая строка. В данном случае метка с именем *l* создана в главном окне приложения. В более сложных случаях, когда метка вложена в несколько фреймов (окон-контейнеров), команда ее создания может выглядеть следующим образом:

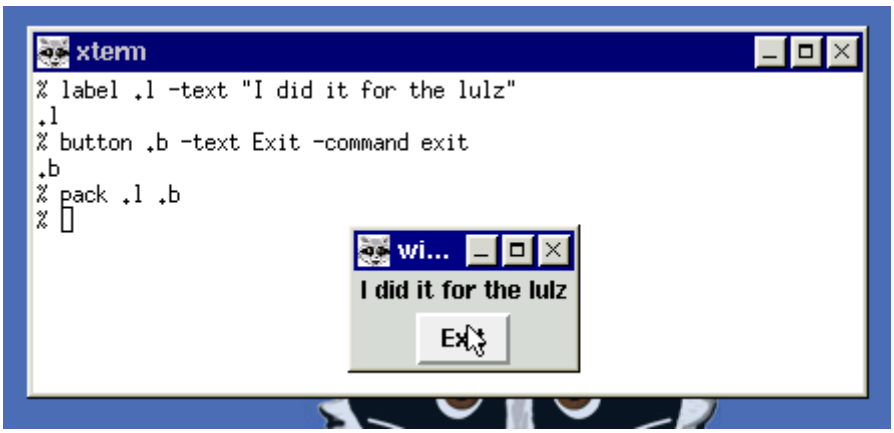
```
label .frame1.frame2.l –text “Hello World!”
```

Где *frame1* – фрейм, созданный в главном окне; *frame2* – фрейм, созданный в *frame1*; *l* – имя метки.

До создания любого нового виджета путь к нему уже должен существовать, но виджет с таким же именем по этому пути должен отсутствовать. То есть, *frame1* и *frame2*, в данном случае, должны быть уже созданы, тогда как виджета с именем *l* быть не должно.

Второй и третий параметры связаны и вместе представляют собой опцию, задающую текст метки. Опции очень широко применяются во всем Tcl. В Tk они задают разные параметры виджетов (текст, шрифт, цвет, размеры, привязанные команды…). Многие опции имеют значения по умолчанию, которые хранятся в специальной базе данных и выбираются оттуда при создании виджета. Если изменить значение какой-то опции в этой базе данных, то оно изменится для всех создаваемых виджетов. Таким образом, можно создавать целые стили оформления и поведения виджетов, для всего приложения.

Команды создания виджетов не публикуют их на форму. Это делается командами менеджерами геометрии. Самый простой менеджер геометрии называется ***pack***. Он позволяет размещать виджеты по сторонам окна (сверху, снизу, справа, слева). Если сторона не указана, то виджеты размещаются сверху вниз.



Кнопки создаются командой ***button***. Опция *–command* указывает команду, или список команд, которые должны вызываться при нажатии этой кнопки. В данном примере указана команда ***exit***, которая завершает работу приложения.

Имя виджета одновременно является и идентификатором его внутреннего объекта. По этому, через него можно обращаться к командам этого объекта. Каждый виджет обязательно имеет две команды: *cget* (возвращает значения опций) и *configure* (изменяет значения опций). Кроме этого виджеты могут иметь и другие специфичные для своего класса внутренние команды. Обращение к этим командам выглядит так:

```
.l configure –text “New text”
.b configure –command {.b flash}
```

При этом метка поменяет текст на *«New text»*, а кнопка при нажатии будет мигать. Параметр *flash* так же является внутренней командой кнопки .b.

Аналогичным образом действует *cget*:

```
.l configure –text [.b cget –text]
```

Заметьте, что при вызове *cget* опция указывается без значения.

Пример Tk программы

Попробуем разобраться в небольшом скрипте, использующем Tk. Приведенная ниже программа предназначена для пользователей Wiki систем, которым часто приходится создавать списки ссылок на новые wiki страницы из списка слов (разных терминов).

Wiki ссылки – это гиперссылки, форматированные в стиле wiki систем. Обычно (как и в данном случае) ссылка заключается в двойные квадратные скобки и разделяется на две части. Первая часть должна ее однозначно идентифицировать в пределах wiki, вторая отображается читателям wiki и, прежде всего, должна быть понятна им.

Так, например, если у нас есть страница описания Си функции printf. Первая часть ссылки равна «stdc_printf», а вторая «printf()», то вся ссылка целиком будет выглядеть как – «[[stdc_printf|printf()]]». Тут можно выделить корень обеих частей – «printf» и две приставки – «stfc_» и «()».

Наша программа будет иметь два основных текстовых поля: для исходного списка слов и для результирующего списка ссылок. Кроме того, будут поля для ввода приставок и окончаний, для обеих частей ссылки.

Давайте сначала взглянем на весь исходный текст, а затем разберем его по частям.

```
#!/bin/sh
# \
exec wish8.4 "$0" ${1+"$@"}

set lpre "tk_"
set lpost ""
set cpre ""
set cpost ""

pack [frame .ftop] -side top

grid \
        [label .ftop.llpre -text "Before link:"] \
        [entry .ftop.lpre -textvar lpre] -sticky w

grid \
        [label .ftop.llpost -text "After link:"] \
        [entry .ftop.lpost -textvar lpost] -sticky w

grid \
        [label .ftop.lcpre -text "Before caption:"] \
        [entry .ftop.cpre -textvar cpre] -sticky w

grid \
        [label .ftop.lcpost -text "After caption:"] \
        [entry .ftop.cpost -textvar cpost] -sticky w

pack [label .lintext -text "Source:"]
pack [text .intext -height 10] -fill both -expand true

pack [label .louttext -text "Result:"]
pack [text .outtext -height 10] -fill both -expand true

pack [button .wikize -text "Start!" -command \
        {
                set s [.intext get]
                foreach txt $s  {
                        .outtext insert end \
"\[\[${lpre}${txt}${lpost}|${cpre}${txt}${cpost}]]\n"
                }
        }
] -side bottom
```

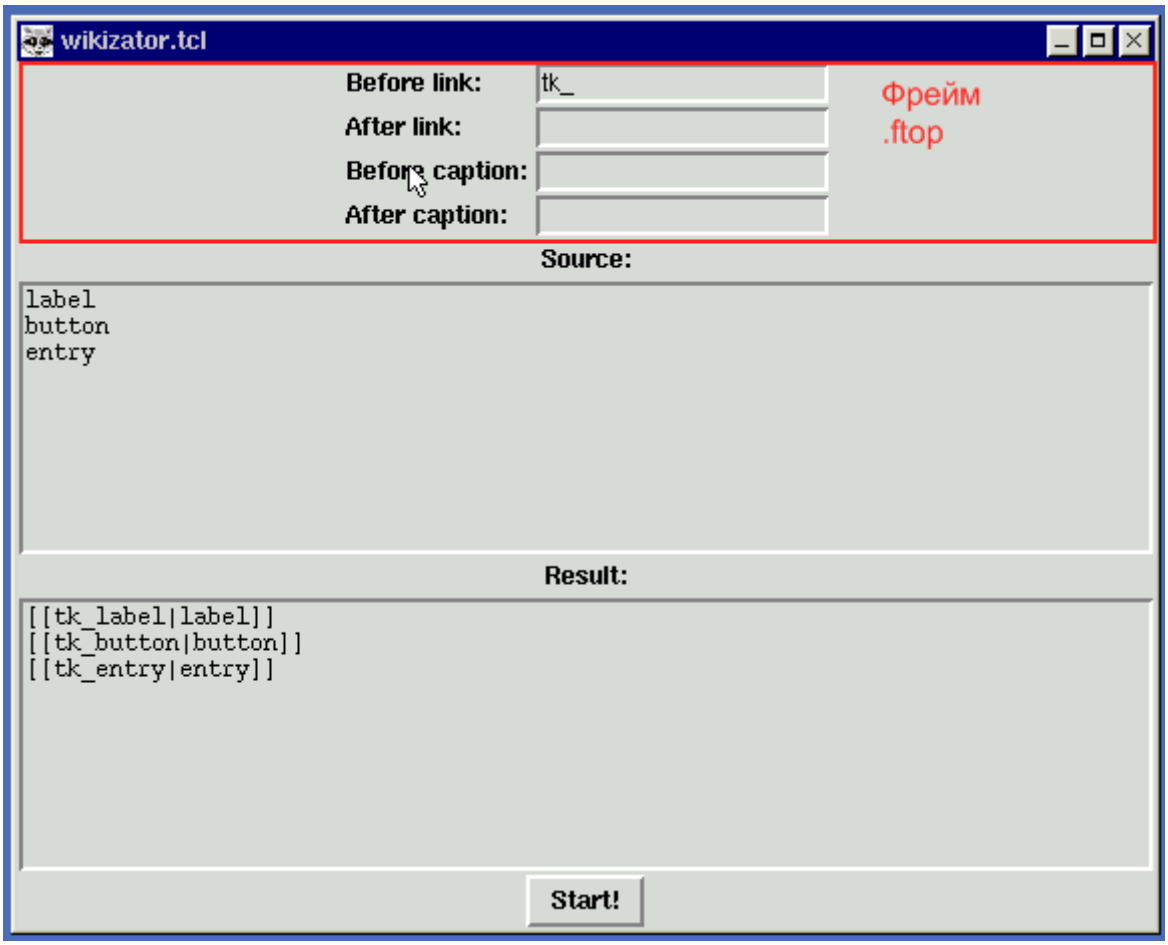
Первые три строки вам уже знакомы. Они запускают нашу программу через **sh**.

Далее идет задание начальных значений переменных. Впоследствии эти переменные будут привязаны к полям ввода, и каждое изменение поля ввода будет изменять и эти переменные.

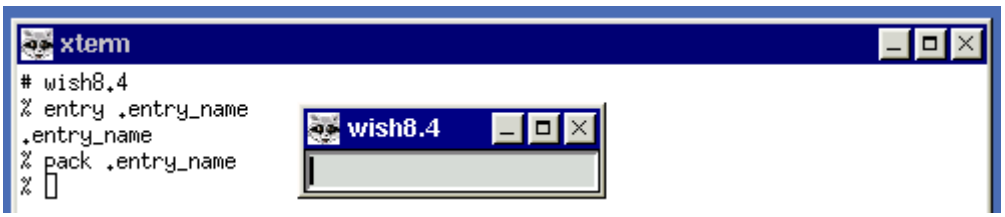
Отметьте, что перед комментарием стоит точка с запятой. Если комментарий следует за командой, то такое отделение обязательно. Дело в том, что символ # равноправен со всеми остальными и может входить в значение параметров команд. Например, команда “***puts #***” напечатает знак шарп на консоли.

Далее создается фрейм *.ftop*. Дело в том, что использовать разные менеджеры геометрии в одном фрейме нельзя, а для упаковки виджетов в два столбца (как мы и сделали вверху окна) удобнее использовать команду ***grid***. По этому, мы создали фрейм *.ftop*, внутри которого будем использовать менеджер геометрии ***grid***, а вне этого фрейма – ***pack***.

Заметьте, что вызов команды ***frame***, и всех последующих команд, создающих виджеты, оформлен в виде вложенных команд (***pack*** *[****frame*** *.ftop] -side top*). Это сделано, что бы исходный текст программы был более компактным. Эти команды возвращают идентификаторы создаваемых виджетов, а по тому их можно использовать как вложенные, напрямую подставляя возвращаемые значения командам ***pack*** и ***grid***.

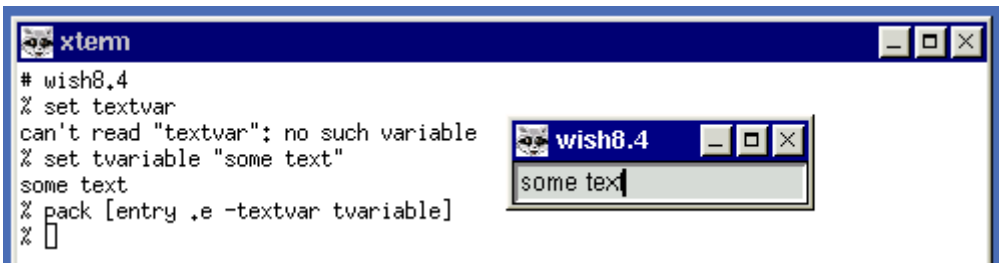


Так, как с созданием меток (***label***) вы уже знакомы, то на этом мы останавливаться не будем, а остановимся на новой команде – ***entry***. Эта команда предназначена для создания полей ввода. Простейший ее вызов выглядит так:



При этом создается пустое поле ввода.

Что бы поле ввода изначально содержало какой-либо текст, необходимо задать опцию *-text*, или связать его с переменной. Во втором случае, мы получаем более удобный метод контроля за содержимым поля ввода.



Теперь изменяя переменную, мы можем изменять содержимое поля ввода, а изменяя текст в поле ввода, мы изменяем содержимое переменной.

Вернемся к рассматриваемой программе. В каждой из строк команд ***grid*** создается и публикуется в фрейм *.ftop* пара виджетов: метка описывающая поле ввода и само поле ввода, к которому привязана переменная. Что бы вам не листать в верх, приведу эти команды снова:

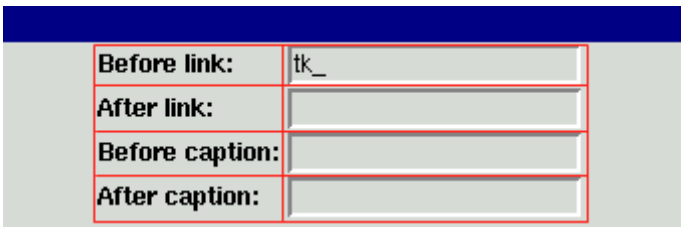
```
grid \
        [label .ftop.llpre -text "Before link:"] \
        [entry .ftop.lpre -textvar lpre] -sticky w

grid \
        [label .ftop.llpost -text "After link:"] \
        [entry .ftop.lpost -textvar lpost] -sticky w

grid \
        [label .ftop.lcpre -text "Before caption:"] \
        [entry .ftop.cpre -textvar cpre] -sticky w

grid \
        [label .ftop.lcpost -text "After caption:"] \
        [entry .ftop.cpost -textvar cpost] -sticky w
```

Бекслеш экранирует перевод строки, что бы команда продолжалась на следующей строке. Опция *–sticky w* определяет выравнивание в каждой из ячеек таблицы менеджера ***grid***. Или, по другому, к какому краю ячейки будет «приклеен» виджет. На рисунке ниже границы этой таблицы обведены красным:



Кроме *-textvar* виджет ***entry*** имеет множество полезных опций. В том числе, опции позволяющие проводить проверку корректности введенной информации (валидацию). Например, на виджете ***entry*** легко сделать поле ввода, принимающее только цифры, или IP адреса.

На следующем шаге программа создает поля ввода исходного текста и результата:

```
pack [label .lintext -text "Source:"]
pack [text .intext -height 10] -fill both -expand true

pack [label .louttext -text "Result:"]
pack [text .outtext -height 10] -fill both -expand true
```

Многострочное текстовое поле создается командой ***text***. Опция *–height* задает высоту текстового поля в строках.

Опция *–fill both* команды ***pack*** указывает, что следует заполнить все имеющееся в окне пространство как по вертикали, так и по горизонтали. Кроме *both* (что значит «оба») в качестве значений этой опции можно использовать *x*, или *y*, которые указывают заполнить окно только по горизонтали, и вертикали соответственно. Опция *–expand true* указывает, что виджеты должны расширяться при изменении размеров окна. Таким образом, опция *–heigth* задала только начальный размер.

Последний виджет в окне – кнопка «Start!», которая запускает весь процесс обработки исходного списка слов:

```
pack [button .wikize -text "Start!" -command \
        {
                set s [.intext get]
                foreach txt $s  {
                        .outtext insert end \
"\[\[${lpre}${txt}${lpost}|${cpre}${txt}${cpost}]]\n"
                }
        }
] -side bottom
```

Особое внимание тут нужно уделить значению опции *–command* команды ***button***, по тому, как это основная часть нашей программы. Поэтому мы разберем ее по строкам.

```
set s [.intext get]
```

Команда ***.intext*** *get* возвращает весь текст, содержащийся в текстовом поле. Команда ***set*** *s …* помещает возвращенный текс в переменную *s*.

Далее идет уже знакомый нам цикл ***foreach***. Он воспринимает текст из *s*, как список слов разделенных пробелами, новыми строками и знаками табуляции. Каждое слово поочередно помещается в переменную txt и передается команде:

```
.outtext insert end \
     "\[\[${lpre}${txt}${lpost}|${cpre}${txt}${cpost}]]\n"
```

Внутренняя команда *insert* виджета текстового поля ***.outtext*** вставляет в него текст. Параметр *end* указывает, что текст нужно вставлять в самый конец текста. Вместо *end* может использоваться индекс, указывающий на любую позицию в текстовом поле. Последний параметр – вставляемая строка.

Тут нужно сделать небольшое отступление и упомянуть один тонкий момент в синтаксисе Tcl. Как говорилось раньше, группировка параметра двойными кавычками разрешает подстановку переменных и вложенных команд. Обычно переменная подставляется командой ***set*** или с помощью «долларового синтаксиса». Однако, может возникнуть ситуация, когда имя переменной сливается с остальным текстом строки и интерпретатор не может понять, какую же переменную мы собрались подставлять. Например:

```
set x 10
puts "$xx zoom"
```

Программист может ждать, что этот участок кода выведет текст «10x zoom», однако интерпретатор выдаст ошибку – «неизвестная переменная xx». В этом случае используют следующий синтаксис:

```
puts "${x}x zoom"
```

То есть, обрамляют имя переменной фигурными скобками. К группировке фигурными скобками это никакого отношения не имеет.

Так вот и во вставляемой нами в ***.outtext*** строке имена переменных ограничены фигурными скобками, что бы не сливаться с другими символами:

```
     "\[\[${lpre}${txt}${lpost}|${cpre}${txt}${cpost}]]\n"
```

Бекслеш экранирует открывающие квадратные скобки, сообщая интерпретатору, что они тут не для подстановки команды, но являются частью вставляемого текста. Экранирование закрывающих квадратных скобок не нужно, так как без открывающих они никаких специальных действий не вызывают.

Если же мы не заполняли какое-либо из полей ввода, и ассоциированная с ним переменная (*lpre, lpost, cpre, cpost*) не содержит никакого значения, то и в этой строке вместо нее ничего подставлено не будет.

Теперь давайте просуммируем все выше написанное, что бы уложить в голове принцип действия этой конкретной программы, использующей Tk. Сначала, мы создаем переменные, хранящие приставки и окончания обоих частей ссылки. Затем создаем верхний фрейм, что бы в нем можно

было использовать упаковщик ***grid***, размещающий виджеты по таблице. Создаем в новом фрейме поля ввода и связываем их с переменными. Подписываем каждое поле ввода виджетом-меткой, что бы было понятно, что куда вводить.

Далее, в основном окне программы, создаем два текстовых поля. Одно для ввода списка слов, а другое для выходного списка ссылок. В самом низу создаем кнопку, по нажатию на которую запускается основной алгоритм программы.

Этот основной алгоритм считывает все содержимое входного окна в переменную и передает ее циклу ***foreach***. Этот цикл воспринимает ее не как цельную переменную, а как список слов, разделенных пробелами, табами и/или переводами строк. В цикле берется каждое слово из этой переменной и, обрамляясь квадратными скобками, приставками и окончаниями по формату wiki ссылки вставляется в выходное текстовое поле ***.outtext***.

Просто? После .NET и Си я был в восторге от такой простоты. Нет никаких файлов проекта, никакой компиляции. Один файл, который сам запускается на исполнение и притом это программа с графическим интерфейсом, работающая во всех ОС, где есть Tcl/Tk (Windows, Linux, *BSD, Solaris, Mac OS X…).

5. Заключение

Скептически настроенный читатель, дочитав до сего места, может задаться вопросом – «не скрыл ли автор от меня какие-то подводные камни?» Действительно, описал какие-то «молочные реки» с «кисельными берегами». В реальности же всегда сложнее!

Наверное, нет такого языка, синтаксис которого не вызывал бы споров и не порождал бы религиозных войн фанатиков. Каждый язык более приспособлен к решению одних задач, упуская из виду другие. Это и позволяет каждому из них находить свою собственную нишу в мире IT, в которой они становятся непревзойденными.

Часто языки содержат опасные или неудобные конструкции. Например, в Си возможно переполнение буфера; Java и C# принуждают программиста к чрезмерному оформительству; C++ имеет такой обширный синтаксис, что даже его создатель заявляет, – полностью его не может знать ни один человек; Lisp отпугивает количеством скобочек; заклинания Perl никогда не разобрать непосвященному; Ada обвиняется в чрезмерной статичности, а Smalltalk в чрезмерной динамичности; BASIC преследуется за GOTO, тогда как в Assembler инструкция jmp вполне равноправна с остальными.

Однако, все выше перечисленные негативные свойства при решении определенных задач как раз и являются преимуществами языка программирования. Позволяют ему делать то, что другим не под силу. Даже переполнение буфера в Си иногда используется во вполне мирных и законных целях.

Tcl не исключение. Он имеет свойства, которые будучи полезными в одной ситуации, в другой становятся злейшим врагом. То же метапрограммирование, которое позволяет значительно сократить количество строк программы, сильно осложняет отладку.

Просто нужно научиться всем этим правильно пользоваться, как ребенка в детстве учат пользоваться огнем. Согласитесь, нет смысла спорить о том полезен ли огонь, или опасен, т.к. он полезен и опасен одновременно.

Полезные источники

Интернет

http://tcl.tk/ - основной сайт группы Tcl Core Team (содержит много страниц,);

http://wiki.tcl.tk/ - вики, в которой есть ответы практически на все вопросы Tcl программиста;

http://ActiveState.com/ - разработчик коммерческого дистрибутива Tcl;

http://tclstudy.narod.ru/ - небольшой учебник для начинающих на русском языке;

http://www.opennet.ru/docs/RUS/tcltk/ - перевод документации и несколько отличных статей.

Книги

1. Брент Уэлш, Кен Джонс, Джеффри Хоббс. Практическое программирование на Tcl и Tk, 4-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2004, - 1136 с.
2. Азбука Tcl. Москвин. 2003г