ЧАСТЬ I

Основы Tcl

В части I приводятся основные сведения о языке Tcl. Главу 1 следует прочитать каждому читателю, независимо от того, какие задачи он собирается решать на практике. В этой главе описаны основные свойства языка. Tcl достаточно прост, поэтому освоить его не составит труда даже для начинающих программистов. Квалифицированным специалистам также необходимо просмотреть главу 1, чтобы избежать подходов, неприменимых при работе с TCL. В главе 2 приведено краткое руководство по запуску Tcl и Tk в операционных средах Unix, Windows и Macintosh. Возможно, что главу 2 вам придется прочитать в первую очередь, чтобы иметь возможность выполнить примеры, приведенные в главе 1.

В главе 3 представлена типичная TCL-программа — CGI-сценарий, реализующий гостевую книгу на Web-узле. При создании этой программы применены средства, подробно описанные в последующих главах. Основная цель главы — ознакомить читателя с реальным примером, демонстрирующим возможности Tcl.

Остальные главы части I посвящены непосредственно программированию на Tcl. Обработка строк описана в главе 4. О работе со списками рассказывается в главе 5. Управляющие структуры, например циклы и выражения `if`, описаны в главе 6. Глава 7 посвящена рассмотрению процедур Tcl, посредством которых можно реализовывать новые Tcl-команды. В главе 8 обсуждаются массивы Tcl. Массивы — одна из наиболее полезных структур в Tcl; они обеспечивают гибкость создаваемых приложений. В главе 9 рассказывается об операциях ввода-вывода и о запуске других программ. Этими средствами создаются Tcl-сценарии, которые позволяют объединять различные программы и обрабатывать данные, содержащиеся в файлах.

Прочитав часть I, вы получите знания, достаточные для того, чтобы разрабатывать простые Tcl-программы, а также читать и понимать код, написанный другими программистами.

Глава 1

Общие сведения о языке Tcl

В данной главе описывается синтаксис языка сценариев Tcl. В частности, здесь рассматриваются подстановка и группировка, выполняемые в процессе интерпретации Tcl-программ. В процессе обсуждения будут рассмотрены следующие команды Tcl: `puts`, `format`, `set`, `expr`, `string`, `while`, `incr` и `proc`.

Tcl представляет собой командный язык, ориентированный на обработку строк. В нем используется лишь несколько основных языковых конструкций. Синтаксис языка крайне прост, что позволяет быстро изучить его. Язык Tcl в основном ориентирован на решение задач, предполагающих объединение набора "строительных блоков" в готовое приложение. Tcl — интерпретируемый язык; в процессе выполнения приложения его код обрабатывается интерпретатором. Благодаря такому подходу существенно упрощается процесс разработки и модернизации приложения; необходимые для этого операции могут выполняться в интерактивном режиме. Возможность выполнения команд в интерактивном режиме существенно упрощает изучение Tcl. Если вы еще не умеете запускать Tcl-интерпретатор в вашей системе, вам следует прочитать главу 2, в которой рассматривается выполнение Tcl в операционных средах Unix, Windows и Macintosh.

В этой главе излагаются общие сведения о языке Tcl. Даже если вы считаете себя опытным программистом, постарайтесь выделить время на то, чтобы прочитать данную главу. Это поможет вам убедиться, что вы действительно знаете язык Tcl, и позволит избежать в дальнейшем многих ошибок. Основные механизмы, лежащие в основе Tcl, связаны с обработкой и подстановкой строк, что позволяет без труда проследить действия интерпретатора. Модель Tcl несколько отличается от других языков программирования, и в этом вы убедитесь, прочитав данную главу.

Команды Tcl

Tcl — это аббревиатура от Tool Command Language. Команды выполняют различные действия, например: вывод строки, вычисление значения арифметического выражения или отображение компонентов (widget) на экране. В Tcl в виде команд представляются все действия, даже присвоение значений переменным и определение процедур. Для вызова команд в языке Tcl предусмотрены достаточно простые синтаксические конструкции. Все сложные действия выполняет интерпретатор, реализующий команды. Команда Tcl записывается в следующем виде:

команда параметр1 параметр2 параметр3...

Команда может представлять собой либо встроенную команду Tcl, либо процедуру. Имя команды отделяется от параметров с помощью одного или нескольких пробелов или знаков табуляции; те же символы используются для разделения параметров. Символ конца строки или точка с запятой завершают команду. В процессе интерпретации Tcl осуществляет *группировку* (объединение нескольких слов в один параметр) и *подстановку* (замену переменных и вызовов вложенных процедур). Действия интерпретатора Tcl по обработке команд можно разделить на три этапа.

- Группировка параметров.
- Подстановка вложенных команд, переменных и символов, которым предшествует обратная косая черта.
- Вызов команды. Интерпретация параметров самой командой. (Этот вопрос будет рассмотрен далее в настоящей главе.)

Hello, World!

Листинг 1.1. Программа "Hello, World!"

```
puts stdout {Hello, World!}
=> Hello, World!
```

В данном примере команде `puts` передаются два параметра: идентификатор потока ввода-вывода и строка. Команда `puts` записывает строку в поток, добавляя символ новой строки. Данный простой пример демонстрирует две следующие особенности Tcl.

- Параметры интерпретируются командой. В данном случае поток ввода-вывода идентифицируется с помощью имени `stdout`. Данное имя потока используется `puts` и другими командами ввода-вывода. Имя `stderr`

определяет стандартный поток ошибок, а имя `stdin` — стандартный поток ввода. Обмен данными с файлами подробно рассматривается в главе 9.

- Фигурные скобки применяются для группировки нескольких слов в один параметр. В результате команда `puts` воспринимает слова `Hello, World!` как второй параметр.

Скобки не являются частью значения.
Фигурные скобки предназначены для предоставления дополнительной информации интерпретатору. Перед тем как значение будет передано команде, скобки удаляются. Посредством скобок группируются все символы, включая переводы строк и вложенные скобки. Группировка оканчивается при появлении закрывающей фигурной скобки. Для группировки в Tcl также могут использоваться двойные кавычки. Группировка будет более подробно рассмотрена далее в этой главе.

Переменные

Для присвоения значений переменной используется команда `set`. Этой команде передаются два параметра: имя переменной и значение. Имя переменной может быть любой длины; регистр символов учитывается. В составе имени допустимы любые символы.

При написании Tcl-программ нет необходимости объявлять переменные перед их использованием.
Интерпретатор создает переменную в тот момент, когда ей необходимо впервые присвоить значение. Для того чтобы обратиться к значению переменной, надо указать перед ее именем символ $, как показано в листинге 1.2.

Листинг 1.2. Использование переменных Tcl

```
set var 5
=> 5
set b $var
=> 5
```

Вторая команда `set`, содержащаяся в листинге, присваивает переменной `b` значение переменной `var`. В данном примере вы впервые встречаетесь с под-

становкой. Чтобы лучше понять, как выполняется вторая команда `set`, ее можно переписать, заменив `$var` значением переменной `var`:

```
set b 5
```

В действительности подстановка осуществляется несколько другими, более эффективными способами, что очень важно в том случае, когда значением переменной является длинная строка.

Подстановка команд

Помимо *подстановки переменных*, в Tcl используется *подстановка команд*. Вложенные команды помещаются в квадратные скобки. Интерпретатор Tcl воспринимает любую последовательность символов, находящуюся между открывающей и закрывающей квадратной скобкой, как команду. В процессе интерпретации осуществляется замена выражения в квадратных скобках (включая сами скобки) результатом выполнения вложенной команды. Квадратные скобки выполняют те же функции, что и одинарные кавычки в некоторых оболочках, кроме того, такой подход позволяет работать с командами любого уровня вложенности.

Листинг 1.3. Подстановка команд

```
set len [string length foobar]
=> 6
```

В листинге 1.3 приведенное ниже выражение представляет собой вложенную команду.

```
string length foobar
```

Данная команда возвращает длину строки "`foobar`". Команда `string` будет подробно описана в главе 4. При выполнении выражения в первую очередь выполняется вложенная команда. Затем осуществляется подстановка, в результате чего внешняя команда приобретает следующий вид:

```
set len 6
```

Если в составе внешней команды находится несколько вложенных команд, интерпретатор обрабатывает их слева направо. Закрывающая скобка является признаком конца очередной команды. Встретив ее, интерпретатор выполняет команду. Это надо учитывать в тех случаях, когда результат одной команды может влиять на выполнение другой.

Математические выражения

Сам по себе интерпретатор Tcl не вычисляет значения математических выражений. Он лишь выполняет группировку, подстановку и вызов команд. Для разбора и вычисления значений математических выражений используется команда `expr`.

Листинг 1.4. Простое арифметическое выражение

```
expr 7.2 / 4
=> 1.8
```

Синтаксис выражений, передаваемых команде `expr` в качестве параметров, такой же, как и синтаксис выражений в языке C. Команда `expr` обрабатывает целые числа, числа с плавающей точкой и логические значения. Результатом выполнения логических операций является значение 0 (`false`) либо 1 (`true`). По необходимости целые числа преобразуются в значения с плавающей точкой. Восьмеричные числа начинаются с нуля (например, значение 033 равно целому числу 27). Шестнадцатеричные значения начинаются с символов 0x. В Tcl поддерживается представление чисел с плавающей точкой. Сведения о приоритете операций приведены в конце данной главы.

В математических выражениях могут присутствовать ссылки на переменные и вложенные команды. В примере, приведенном в листинге 1.5, посредством команды `expr` выполняется сложение значения переменной `x` с числом, равным длине строки "`foobar`". В результате подстановки команде `expr` передается 6 + 7 и при выполнении команды `set` переменной `len` присваивается значение 13.

Листинг 1.5. Вложенные команды

```
set x 7
set len [expr [string length foobar] + $x]
=> 13
```

Средства обработки выражений поддерживают ряд встроенных функций, применяемых при проведении математических вычислений. (Перечень таких функций приведен в конце данной главы.) В примере, показанном в листинге 1.6, вычисляется число `pi`.

Листинг 1.6. Использование встроенных математических функций

```
set pi [expr 2*asin(1.0)]
=> 3.1415926535897931
```

При реализации функции `expr` были приняты меры для обеспечения корректности значений и предотвращения нежелательных преобразований числовых значений в строковые. Разработчики программ, со своей стороны, могут повысить эффективность выполнения операции `expr`, применяя фигурные скобки для группировки выражений. Это связано с особенностями работы компилятора, преобразующего исходный текст в байтовый код. Более подробно данный вопрос будет рассмотрен далее в этой главе. Пример использования фигурных скобок для повышения быстродействия вычислений приведен в листинге 1.7. Все выражения, содержащиеся в этом листинге, составлены корректно.

Листинг 1.7. Использование фигурных скобок для группировки выражений

```
expr {7.2 / 4}
set len [expr {[string length foobar] + $x}]
set pi [expr {2*asin(1.0)}]
```

Подстановка символов, представленных с помощью обратной косой черты

Еще один тип подстановки, выполняемый интерпретатором Tcl, связан с использованием обратной косой черты. Таким способом в состав параметров можно включать символы, которые в обычных условиях имеют специальные значения. Предположим, например, что вы собираетесь использовать в составе параметра символ `$` или какую-либо скобку. Перед таким символом надо указать обратную косую черту. Если для составления требуемого выражения вам надо использовать большое число символов обратной косой черты, следует помнить, что существует более простой способ, позволяющий получить тот же результат. В частности, команда `list`, которая будет описана в главе 5, автоматически отменяет специальные значения символов. В листинге 1.8 приведен пример использования обратной косой черты для получения литерального значения символа `$`.

Листинг 1.8. Отмена специального значения символа с помощью обратной косой черты

```
set dollar \$foo
=> $foo
set x $dollar
=> $foo
```

Интерпретация выполняется в один проход.
Вторая команда `set` в приведенном выше примере иллюстрирует важную особенность Tcl. Значение переменной `dollar` не испытывает на себе влияния операции подстановки. Другими словами, при подстановке разбор значений переменных не производится. В данном примере значением как переменной `dollar`, так и переменной `x` является строка "`$foo`". Общее правило таково: при использовании `eval` нет необходимости заботиться о значениях переменных. Подробнее этот вопрос будет обсуждаться в главе 10.

С помощью последовательностей знаков, начинающихся с обратной косой черты, можно записывать шестнадцатеричные, восьмеричные значения символов, а также представлять их в формате Unicode.

```
set escape \u001b
set escape \0x1b
set escape \033
```

В каждом из этих выражений переменной `escape` присваивается символ ASCII ESC, код которого в десятичном представлении равен 27. Правила представления символов последовательностями, начинающимися с обратной косой черты, приведены в табл. 1.1.

Часто символ обратной косой черты используется для записи длинной команды в нескольких строках. Как было сказано ранее, перевод строки завершает команду. В примере, представленном в листинге 1.9, обратная косая черта обязательно должна присутствовать, иначе последним символом в записи команды `expr` будет знак `+`.

Листинг 1.9. Запись длинной команды в двух строках с использованием обратной косой черты

```
set totalLength [expr [string length $one] + \
    [string length $two]]
```

Отменить специальное значение перевода строки можно двумя способами. Первый способ состоит в формировании параметра посредством группировки. В этом случае никаких действий для отмены специального значения перевода строки не требуется; соответствующий символ становится частью группы и не завершает команду. Второй способ — включить в конец строки обратную косую черту. В этом случае последний символ в строке преобразуется в пробел, а все пробелы в начале следующей строки удаляются. Другими словами, символ обратной косой черты не только позволяет продолжить команду в следующей строке, но и удаляет ненужные пробелы, включаемые в начало строки для форматирования.

Группировка с помощью фигурных скобок и двойных кавычек

Двойные кавычки и фигурные скобки используются для группировки нескольких слов в один параметр. Различие между ними состоит в том, что кавычки допускают подстановку в группе, а фигурные скобки запрещают ее. Это правило действует по отношению к командам, переменным и последовательностям, начинающимся с обратной косой черты.

Листинг 1.10. Группировка с помощью двойных кавычек и фигурных скобок

```
set s Hello
=> Hello
puts stdout "The length of $s is [string length $s]."
=> The length of Hello is 5.
puts stdout {The length of $s is [string length $s].}
=> The length of $s is [string length $s].
```

При выполнении второй команды, приведенной в данном примере, интерпретатор Tcl, обрабатывая второй параметр `puts`, выполняет подстановку как значения переменной, так и команды. В третьей команде подстановка запрещена, поэтому строка выводится в том виде, в котором она указана в записи параметра.

На практике группировка с помощью фигурных скобок в основном выполняется тогда, когда подстановка параметра должна быть отложена на более позднее время либо вовсе не должна выполняться. В качестве примеров можно привести циклы, условные выражения и объявления процедур. Двойные кавычки чаще всего применяются в командах, подобных `puts`.

Кавычки также часто используются в команде `format`. Эта команда выполняет те же функции, что и функция `printf` в языке C. Первый параметр команды `format` определяет формат вывода. В нем часто присутствуют специальные символы, например перевод строки, знаки табуляции и пробелы. Проще всего указать эти символы с помощью последовательностей, начинающихся с обратной косой черты (например, перевод строки можно представить как `\n`, а знак табуляции — как `\t`). Перед выполнением команды `format` последовательности, начинающиеся с обратной косой черты, должны быть преобразованы, поэтому для формирования параметра, определяющего формат, целесообразно использовать двойные кавычки.

```
puts [format "Item: %s\ t%5.3f" $name $value]
```

В данном случае команда `format` используется для выравнивания при выводе значений переменных `name` и `value` с помощью символа табуляции.

Выражения `%s` и `%5.3f` задают формат вывода остальных параметров команды. Заметьте, что символ `\n`, обычно указываемый при вызове команды `printf` языка C, здесь отсутствует. Он не нужен, так как команда `puts` сама добавляет символ перевода строки. Подробно команда `format` будет описана в главе 4.

Особенности использования квадратных скобок

Квадратные скобки используются для подстановки команд и не выполняют группировку. Вместо этого вложенная команда, определяемая с помощью квадратных скобок, рассматривается как часть текущей группы. В приведенной ниже команде двойные кавычки формируют последний параметр, выполняя группировки, а результаты выполнения вложенной команды включаются в группу.

```
puts stdout "The length of $s is [string length $s]."
```

Если в состав параметра входит только вложенная команда, вам нет необходимости группировать элементы с помощью двойных кавычек, поскольку средства разбора Tcl рассматривают вложенную команду как единое целое.

```
puts stdout [string length $s]
```

В приведенном ниже выражении кавычки излишни.

```
puts stdout "[expr $x + $y]"
```

Группировка перед подстановкой

Интерпретатор Tcl выполняет разбор команды за один проход. В течение единственного прохода он принимает решение о группировке и подстановке. Группировка осуществляется перед подстановкой. Эту особенность интерпретатора Tcl необходимо иметь в виду при написании программ. Значения, полученные в результате подстановки, не влияют на группировку, так как решение о группировке было принято ранее.

Приведенный ниже пример демонстрирует влияние вложенных команд на группировку. Вложенная команда рассматривается как неразрывная последовательность символов, причем внутренняя структура этих символов не учитывается. При формировании параметров основной команды результаты подстановки включаются в текущую группу символов.

Листинг 1.11. Вложенная команда и подстановка переменных

```
set x 7; set y 9
puts stdout $x+$y=[expr $x + $y]
=> 7+9=16
```

В листинге 1.11 второй параметр команды `puts` имеет следующий вид:

```
$x+$y=[expr $x + $y]
```

Пробелы в составе вложенной команды при группировке игнорируются. В тот момент, когда интерпретатор Tcl встречает левую квадратную скобку, подстановка некоторых переменных уже выполнена и сформирована следующая строка:

```
7+9=
```

При появлении левой квадратной скобки осуществляется рекурсивный вызов интерпретатора для выполнения вложенной команды. Перед выполнением `expr` подстановка переменных `$x` и `$y` уже выполнена. Результат выполнения команды `expr` включается вместо всей последовательности символов, находящихся между левой и правой квадратной скобкой. Поэтому команда `puts` получает в качестве второго параметра следующее значение:

```
7+9=16
```

Группировка выполняется перед подстановкой

В рассмотренном выше примере при обработке второго параметра команды `puts` решение о группировке принимается перед решением о подстановке. Даже если результат выполнения вложенной команды содержит пробелы или другие специальные символы, аргумент будет передан команде без учета специальных значений. Группировка и подстановка переменных соотносятся между собой так же, как группировка и подстановка команд. Пробелы или другие специальные символы в составе значений переменных не оказывают влияние на решение о группировке, так как это решение принимается еще тогда, когда значения переменных недоступны.

Если вы хотите, чтобы при выводе результатов выполнения команды символы `+` и `=` были отделены от чисел пробелами, вам надо явным образом сгруппировать параметр, используя двойные кавычки. При этом команда `puts` примет следующий вид:

```
puts stdout "$x + $y = [expr $x + $y]"
```

В данном случае для группировки используются кавычки. Это важно, так как в составе группы должна быть разрешена подстановка значений переменных и команды.

Группировка математических выражений с помощью фигурных скобок

Команда `expr` выполняет некоторые действия по подстановке внутри фигурных скобок. Более подробно этот вопрос будет рассмотрен далее в этой

главе. Таким образом, оказывается, что в приведенной ниже команде выполняется подстановка значений переменных в выражении, помещенном в фигурные скобки.

```
puts stdout "$x + $y = [expr {$x + $y}]"
```

Примеры подстановки

Если в составе команды присутствует несколько подстановок, не разделенных пробелами или знаками табуляции, группировка выполняется по умолчанию. Использовать двойные кавычки в этом случае не обязательно. В приведенной ниже команде осуществляется конкатенация значений переменных `a`, `b` и `c`.

```
set concat $a$b$c
```

Если же вам необходимо разделить значения переменных пробелами, необходимо использовать кавычки.

```
set concat "$a $b $c"
```

В общем случае команды, помещенные в квадратные скобки, и ссылки на переменные можно указывать в любой позиции строки. В приведенном ниже примере имя команды определяется в результате выполнения другой команды.

```
[findCommand $x] параметр параметр
```

При работе с Tk имя компонента часто используется как имя команды.

```
$text insert end "Hello, World!"
```

Процедуры

Для определения процедур в Tcl используется команда `proc`. Единожды определенная Tcl-процедура может быть использована многократно, причем вызывается она точно так же, как и встроенная команда Tcl. Определение процедуры осуществляется с помощью следующего выражения:

```
proc имя список_параметров тело_процедуры
```

Первый параметр — это имя определяемой процедуры. В качестве второго параметра команды задается список параметров процедуры. Третий параметр — тело процедуры, включающее одну или несколько Tcl-команд.

В имени процедуры могут содержаться практически любые символы; регистр символов учитывается. В данной книге принято соглашение об именовании, согласно которому имена процедур начинаются с символа верхнего

регистра, а имена переменных — с символа нижнего регистра. По мере развития Tcl все большее значение приобретает стиль программирования. Этому вопросу посвящена глава 12.

Листинг 1.12. Определение процедуры

```
proc Diag {a b} {
   set c [expr {sqrt($a * $a + $b * $b)}]
   return $c
}
puts "The diagonal of a 3, 4 right triangle is [Diag 3 4]"
=> The diagonal of a 3, 4 right triangle is 5.0
```

Процедура `Diag`, определенная в данном примере, вычисляет длину гипотенузы прямоугольного треугольника. В качестве параметров процедуре передаются длины катетов. Функция `sqrt` — одна из функций, поддерживаемых командой `expr`. Переменная `c` — это локальная переменная процедуры; она существует только при выполнении `Diag`. Области видимости переменных будут обсуждаться в главе 7. Переменная `c` в данном примере не обязательна; при необходимости можно было бы составить код процедуры, не используя переменных. В этом случае процедура выглядела бы следующим образом:

```
proc Diag {a b} {
   return [expr {sqrt($a * $a + $b * $b)}]
}
```

Команда `return` возвращает результаты выполнения процедуры. В данном случае она не обязательна, так как интерпретатор Tcl по умолчанию возвращает значение последней команды в теле процедуры. Таким образом, процедура `Dial` может выглядеть так:

```
proc Diag {a b} {
   expr {sqrt($a * $a + $b * $b)}
}
```

Обратите внимание на использование фигурных скобок. Скобка в конце первой строки является началом третьего параметра команды `proc`; этот же параметр является телом процедуры. Поскольку при разборе команды интерпретатор Tcl встречает открывающую фигурную скобку, он игнорирует специальное значение символов новой строки и читает текст до появления закрывающей скобки. Использование двойных кавычек для группировки даст тот же результат. Интерпретатор будет группировать символы, включая переводы строк, до появления второй двойной кавычки. В результате группировки

формируется третий параметр `proc`, представляющий собой последовательность команд. В дальнейшем при выполнении процедуры символы перевода строки будут завершать команды.

Помещение тела процедуры в фигурные скобки дает еще один эффект: подстановка откладывается до тех пор, пока не начнется выполнение процедуры. Это очень важно, так как в данном примере переменные `a`, `b` и `c` будут определены лишь при вызове процедуры, поэтому в определении `Diag` выполнить подстановку этих переменных невозможно.

Команда `proc` предоставляет также дополнительные возможности, например, поддерживает различное число параметров и позволяет использовать значения параметров по умолчанию. Данный вопрос будет подробно обсуждаться в главе 7.

Пример вычисления факториала

Чтобы закрепить полученные знания, рассмотрим пример, в котором для вычисления факториала применяется цикл.

Листинг 1.13. Использование цикла `while` для вычисления факториала

```
proc Factorial {x} {
   set i 1; set product 1
   while {$i <= $x} {
      set product [expr {$product * $i}]
      incr i
   }
   return $product
}
Factorial 10
=> 3628800
```

Точка с запятой в первой строке используется для разделения команд. Этот символ выполняет те же функции, что и перевод строки. Цикл `while` применяется для того, чтобы перемножить числа от единицы до значения `x`. Первый параметр `while` представляет собой логическое выражение, а второй параметр является телом цикла. Команда `while` и другие управляющие структуры описаны в главе 6.

Для вычисления значения логического выражения применяются те же средства, которые использует команда `expr`. По этой причине в первый параметр команды `while` не включается команда `expr`. Даже если вычисляется сложное выражение, в использовании `expr` нет необходимости.

Как и тело процедуры, тело цикла формируется с помощью фигурных скобок. Открывающая фигурная скобка, предназначенная для формирования тела процедуры, должна находиться в той же строке, что и команда `proc`, а если скобка формирует тело цикла, ее следует располагать в той же строке, что и команду `while`. Если же вы хотите поместить фигурную скобку в следующей строке, то текущая строка обязательно должна оканчиваться символом обратной косой черты. Как вы уже знаете, обратная косая черта отменяет специальное значение перевода строки.

```
while {$i < $x} \
{
   set product ...
}
```

Для группировки логических выражений и команд, составляющих тело цикла, всегда надо использовать фигурные скобки.

При группировке логического выражения фигурные скобки нельзя заменять двойными кавычками, поскольку подстановка переменных должна быть отложена до того момента, когда интерпретатор начнет вычислять выражение. Ниже приведен пример бесконечного цикла.

```
set i 1; while $i<=10 {incr i}
```

Этот цикл никогда не будет завершен[1]. Дело в том, что интерпретатор Tcl выполнит подстановку `$i` перед вызовом команды `while`, в результате логическое выражение примет вид `1<=10`. Значение такого выражения всегда равно `true`. Чтобы устранить подобные ошибки, надо соблюдать рекомендуемый стиль составления кода, в частности всегда применять для группировки выражений фигурные скобки.

```
set i 1; while {$i<=10} {incr i}
```

Для увеличения значения переменной `i` в теле цикла используется команда `incr`. Она очень удобна, так как заменяет более длинную команду.

```
set i [expr {$i + 1}]
```

Команде `incr` может передаваться дополнительный параметр: положительное или отрицательное целое число, указывающее, на какую величину должно изменяться значение переменной. Используя данную форму команды, можно отказаться от применения переменной цикла и ограничиться переменной `x`. В этом случае цикл примет следующий вид:

[1]Это может показаться смешным, но при использовании Tcl 8.0 данное утверждение неверно. В версии 8.0 был представлен компилятор байтового кода, при реализации которого была допущена ошибка, приводящая к тому, что цикл завершается. Данная ошибка была исправлена в реализации 8.0.5. — Прим. авт.

```
while {$x > 1} {
   set product [expr {$product * $x}]
   incr x -1
}
```

В листинге 1.14 приведена модифицированная процедура вычисления факториала, в которой используется рекурсивный вызов. Рекурсивной называется такая функция, которая в процессе выполнения вызывает саму себя. При каждом рекурсивном вызове значение x уменьшается на единицу, а когда оно станет равным 1, рекурсивные обращения прекращаются.

Листинг 1.14. Рекурсивная функция, вычисляющая факториал

```
proc Factorial {x} {
   if {$x <= 1} {
      return 1
   } else {
      return [expr {$x * [Factorial [expr {$x - 1}]]}]
   }
}
```

Дополнительные сведения о переменных

Если команде `set` передается один параметр, она возвращает значение переменной. Данная команда интерпретирует параметр как имя переменной и возвращает ее текущее значение. Символ `$`, который указывается перед именем переменной для доступа к ее значению, — это лишь сокращенный вариант команды `set`. Пример, представленный в листинге 1.15, показывает, какой интересный результат можно получить, присваивая переменной имя другой переменной.

Листинг 1.15. Использование команды `set` для доступа к значению переменной

```
set var {the value of var}
=> the value of var
set name var
=> var
set name
=> var
set $name
=> the value of var
```

Данный пример несколько сложнее предыдущих. В последней команде вместо `$name` подставляется `var`. Затем команда `set` возвращает значение переменной `var`, т.е. строку "`the value of var`". Вложенные команды `set` реализуют косвенное обращение к переменным. Последняя команда `set` в приведенном выше примере может быть переписана следующим образом:

```
set [set name]
=> the value of var
```

Вначале использование переменной для хранения имени другой переменной может показаться сложным. Однако в некоторых случаях такой подход оказывается очень полезным. Имеется даже специальная команда `upvar`, которая упрощает действия с использованием косвенной адресации. Эта команда будет рассмотрена в главе 7.

Особенности применения различных символов в именах переменных

Интерпретатор Tcl использует некоторые соглашения об именовании переменных, которые упрощают включение ссылок на переменные в состав строк. По умолчанию интерпретатор предполагает, что имена переменных содержат только буквы, цифры и знаки подчеркивания. Выражение `$foo.o` представляет собой конкатенацию значения переменной `foo` и литерала "`.o`".

Если ссылка на переменную не отделена от остальной части строки знаками пунктуации или пробелами, вы можете выделить имя переменной с помощью фигурных скобок (например, `${x}`). Использование фигурных скобок также позволяет включать в состав имен произвольные символы, однако работать с такими переменными не всегда удобно и для того, чтобы поступать так, надо иметь веские основания. Если вы хотите включить в имя переменной символы, отличные от букв, цифр и знаков подчеркивания, либо если вам необходимо формировать имя в процессе вычислений, вы можете воспользоваться командой `upvar`.

Листинг 1.16. Различные способы формирования ссылок на переменные

```
set foo filename
set object $foo.o
=> filename.o
set a AAA
set b abc${a}def
=> abcAAAdef
set .o yuk!
```

```
set x ${.o}y
=> yuk!y
```

Команда unset

Переменную можно удалить с помощью команды `unset`.

`unset ?-nocomplain? ?--? имя_переменной_1 имя_переменной_2 ...`

Команде `unset` может быть передано любое количество имен переменных. Если переменная не определена, `unset` сообщает об ошибке. Для того чтобы подавить генерацию сообщений об ошибке, надо указать опцию `-nocomplain`. Чтобы удалить переменную с именем `-nocomplain`, надо включить символы `--`.

Проверка наличия переменных

Проверить, имеется ли переменная с определенным именем, позволяет команда `info exists`. Некоторым командам, например команде `incr`, можно передавать только имена существующих переменных. Поэтому перед выполнением подобной команды желательно проверить наличие переменной с данным именем.

Листинг 1.17. Использование команды `info exists` для проверки наличия переменной

```
if {![info exists foobar]} {
   set foobar 0
} else {
   incr foobar
}
```

В листинге 7.6 показана версия `incr`, в процессе выполнения которой осуществляется проверка наличия переменной.

Дополнительные сведения о математических выражениях

В данном разделе описываются некоторые особенности работы с математическими выражениями в Tcl-сценариях. В Tcl 7.6 и более ранних версиях математические вычисления выполнялись неэффективно из-за преобразования строковых значений в числовые. Преобразование последовательности символов в число должна осуществлять команда `expr`. Эта же команда выполняет вычисления, используя числа двойной точности с плавающей точкой.

Результат преобразуется в строковое значение, содержащее по умолчанию 12 цифр. Число значащих цифр можно изменить, устанавливая значение переменной `tcl_precision`. Для того чтобы при преобразовании строки в число с двойной точностью и числа в строку не терялась информация, достаточно семнадцати значащих цифр.

Листинг 1.18. Управление точностью с помощью переменной `tcl_precision`

```
expr 1 / 3
=> 0
expr 1 / 3.0
=> 0.333333333333
set tcl_precision 17
=> 17
expr 1 / 3.0
# Завершающая цифра 1 появляется вследствие специфики округления
# по правилам IEEE
=> 0.33333333333333331
```

В Tcl 8.0 и более поздних версиях необходимость преобразования в большинстве случаев устраняется за счет использования встроенного компилятора. Но несмотря на это, Tcl нежелательно применять для создания приложений, в процессе выполнения которых должны выполняться интенсивные математические вычисления. По необходимости вы можете реализовать функцию, выполняющую математические вычисления, на компилируемом языке и зарегистрировать ее как команду Tcl. Подробно этот вопрос будет рассмотрен в главе 47.

Команда `expr` поддерживает сравнение строк, поэтому вы можете проверять строковые значения в выражениях `if`. Для того чтобы команда `expr` знала о том, что надо сравнивать строки, в выражении следует использовать кавычки.

```
if {$answer == "yes"} { ... }
```

Следует заметить, что более надежными являются команды `string compare` и `string equal`, так как команда `expr` может выполнять нежелательное преобразование строк, которые напоминают числа. Вопросы обработки строк и использования функции `expr` рассматриваются в главе 4. В Tcl 8.4 были введены операции `eq` и `ne expr`, непосредственно предназначенные для сравнения строк.

В выражениях, предполагающих подстановку переменных и команд, может выполняться группировка с помощью фигурных скобок. Это возможно потому, что в параметре команды `expr` подстановка осуществляется в два

прохода: первый проход выполняет интерпретатор Tcl, а второй — сама команда `expr`. Как правило, при этом проблемы не возникают, так как математические выражения не содержат символов, имеющих специальное значение. Второй проход подстановки необходим для поддержки `while`, `if` и других команд, в которых вычисляется значение логических выражений.

Группировка позволяет обеспечить более эффективное выполнение команд.

Чтобы команда `expr` самостоятельно выполняла подстановку команд и переменных, группировку необходимо выполнять с использованием фигурных скобок. В противном случае могут осуществляться нежелательные преобразования числовых значений в строковые и строковых — в числовые. Это замедляет выполнение программ, кроме того, в некоторых случаях преобразования могут влиять на точность вычислений. Предположим, что переменной `x` присваивается значение, являющееся результатом вычисления выражения:

```
set x [expr {sqrt(2.0)}]
```

Как и следовало ожидать, значение `x` представляет собой число с плавающей точкой двойной точности. Предположим теперь, что полученное значение `x` вы используете в следующем выражении:

```
set two [expr $x * $x]
```

Вполне возможно, что результат не будет равен 2.0! Причина состоит в том, что Tcl подставляет `$x`, `expr` объединяет все параметры в одну строку и снова выполняет разбор выражения. Если вы запишете то же выражение следующим образом:

```
set two [expr {$x * $x}]
```

то команда `expr` самостоятельно выполнит подстановку и значение `x`, являющееся числом с плавающей точкой, сохранится. Поскольку преобразование в строку символов не производится, значение выражения будет вычислено точнее и эффективнее. Более детально обработка Tcl-переменных обсуждается в главе 47.

Комментарии

Для обозначения комментариев в Tcl используется символ `#`. В отличие от многих других языков, он должен быть указан в начале команды. Тот же символ в другой позиции не будет обработан специальным образом. Часто комментарии располагают после окончания команды, предваряя символ `#` точкой с запятой, завершающей предыдущую команду.

```
# Использование комментариев
set rate 7.0 ;# Общая оценка
set months 60 ;# Срок возвращения кредита
```

Заметьте, что символ обратной косой черты позволяет продолжить комментарии в следующей строке. Точка с запятой в составе комментариев рассматривается как обычный символ. Строку комментариев завершает только символ перевода строки.

```
# Начало комментариев в программе на языке Tcl \
продолжение комментариев (обратите внимание на отсутствие символа #).
```

Пример использования обратной косой черты при составлении комментариев приведен в листинге 2.3.

Комментарии Tcl имеют следующую особенность: открывающей фигурной скобке в составе комментариев обязательно должна соответствовать закрывающая скобка. Это странное правило было принято для того, чтобы упростить реализацию программ разбора Tcl. В результате приведенный ниже фрагмент кода будет работать некорректно.

```
# if {boolean expression1} {
if {boolean expression2} {
   команды
}
```

В предыдущем примере в комментариях присутствует лишняя открывающая скобка. При разборе будет сгенерировано сообщение о том, что в конце сценария отсутствует закрывающая фигурная скобка. Для того чтобы запретить выполнение больших фрагментов кода, можно поместить требуемые команды в блок, который никогда не будет выполнен.

```
if {0} {
   неиспользуемый код
}
```

Правила подстановки и группировки

Ниже приведен перечень правил группировки и подстановки, осуществляемых интерпретатором Tcl перед выполнением команды.

- Параметры команды разделяются пробелами. Если параметр формируется посредством группировки с использованием фигурных скобок или двойных кавычек, пробелы в составе группы теряют специальное значение и рассматриваются как обычные символы.

- Группировка с помощью фигурных скобок запрещает подстановку. Интерпретатор объединяет в группу все символы, находящиеся между левой и соответствующей ей правой фигурной скобкой, в том числе символы перевода строки, точки с запятой и вложенные скобки. Группировка оканчивается при появлении закрывающей скобки. Внешние скобки не включаются в состав группы.
- Группировка с помощью двойных кавычек разрешает подстановку. Интерпретатор группирует все символы, включая переводы строк и точки с запятой, до появления второй двойной кавычки. Кавычки не включаются в состав группы. Если в группу должен войти символ двойной кавычки, перед ним надо указать обратную косую черту.
- Решение о группировке принимается перед подстановкой, поэтому значения переменных и результаты выполнения программ не влияют на группировку.
- Символ `$` вызывает подстановку переменной. Имя переменной может быть любой длины; регистр символов принимается во внимание. Если в строке содержится ссылка на переменную, не отделенная от остальной части строки символами, имеющими специальное значение, или если в составе имени переменной содержатся символы, отличные от букв, цифр и знака подчеркивания, то ссылка записывается в виде `${имя_переменой}`.
- Квадратные скобки вызывают подстановку команды. Выражение в квадратных скобках (включая сами скобки) заменяется результатом выполнения команды. Вложенные команды, в свою очередь, могут содержать другие команды.
- Обратная косая черта отменяет специальное значение символов. Использование ее можно рассматривать как подстановку, при которой обратная косая черта и следующие за ней один или несколько знаков заменяются другим символом.
- Подстановка может выполняться в любой позиции, за исключением тех случаев, когда для группировки используются фигурные скобки. Часть строки, сформированной при группировке, могут составлять строковые литералы, а остальную часть — результаты подстановок. С помощью подстановки можно сформировать даже имя команды.
- Перед вызовом команды осуществляется подстановка; она выполняется в течение одного прохода. Результат подстановки повторно не интерпретируется. Благодаря этому появляется возможность задавать значения переменных, содержащие специальные символы, такие как символ `$`, пробелы, квадратные или фигурные скобки. Поскольку подстановка

выполняется в один проход, эти символы не выполняют специальных функций в сформированной строке.

Особенности группировки и подстановки

- При написании Tcl-программ начинающие программисты, используя фигурные скобки или двойные кавычки для группировки, иногда не указывают пробел между параметрами. Это приводит к возникновению ошибки. Дело в том, что пробелы играют роль разделителей, а фигурные скобки или кавычки лишь выполняют группировку. Если пробел пропущен, интерпретатор сообщит о том, что после закрывающей скобки или кавычки ожидается другой символ. В следующем примере ошибка возникает вследствие того, что между закрывающей и открывающей фигурной скобкой пропущен пробел:

```
if {$x > 1}{puts "x = $x"}
```

- Двойная кавычка начинает группу лишь в том случае, если перед ней указан пробел или знак табуляции. Это означает, что вы можете разместить кавычку внутри последовательности символов, не предваряя ее обратной косой чертой. В этом случае группа должна быть отделена от других частей строки пробелами или фигурными скобками. Несмотря на то что приведенная ниже команда корректна, использовать подобные выражения не рекомендуется, так как текст программы становится трудным для восприятия.

```
set silly a"b
```

- Если для группировки используются двойные кавычки, то фигурные скобки теряют свои специальные свойства. В группе, ограниченной кавычками, подстановка осуществляется независимо от наличия скобок. В следующем примере видно, что фигурные скобки, находящиеся в группе, сформированной с помощью кавычек, не отменяют подстановку:

```
set x xvalue
set y "foo {$x} bar"
=> foo {xvalue} bar
```

- Если для группировки используются кавычки, а в составе группы находится вложенная команда, то в команде также может содержаться группа, созданная с помощью кавычек. Сказанное иллюстрируется следующим примером:

```
puts "results [format "%f %f" $x $y]"
```

- Если для подстановки команд используются квадратные скобки, то перед открывающей скобкой и после закрывающей скобки пробелы не обязательны. Интерпретатор рассматривает все символы, находящиеся между квадратными скобками, как часть текущей группы. Приведенная ниже команда присваивает переменной `x` значение, представляющее собой конкатенацию результатов выполнения двух команд. Так происходит потому, что между закрывающей и открывающей квадратной скобкой отсутствует пробел.

  ```
  set x [ команда_1][команда_2]
  ```

- При группировке с помощью фигурных скобок или двойных кавычек символы новой строки и точка с запятой теряют свое специальное значение. Они включаются в группу так же, как и обычные символы. В следующем примере переменной `x` присваивается значение, содержащее символы перевода строки:

  ```
  set x "This is line one.
  This is line two.
  This is line three."
  ```

- При подстановке символ перевода строки и точка с запятой завершают команду. Если вам надо поместить в квадратных скобках длинную команду и вы хотите записать ее в нескольких строках, поставьте в конце каждой строки (кроме последней) обратную косую черту. Пример записи команды в нескольких строках содержится в листинге 1.9.
- Если после знака `$` нет ни буквы, ни цифры, ни знака подчеркивания, ни открывающей скобки, он интерпретируется как обычный литеральный символ. В приведенном ниже примере переменной x присваивается символ $.

  ```
  set x $
  ```

Справочная информация

Последовательности символов, начинающиеся с обратной косой черты

Таблица 1.1. Использование обратной косой черты для представления символов

\a	Звуковой сигнал (0x7)
\b	Возврат (0x8)
\f	Перевод страницы (0xc)
\n	Перевод строки (0xa)
\r	Возврат каретки (0xd)
\t	Табуляция (0x9)
\v	Вертикальная табуляция (0xb)
\<*перевод строки*>	Замена символов перевода строки и всех пробелов в начале следующей строки одним пробелом
\\	Обратная косая черта ('\')
ooo	Символ, заданный восьмеричным числом. После обратной косой черты могут следовать одна, две или три восьмеричные цифры (0–7).
\x*hh*	Символ, заданный шестнадцатеричным числом. После \x могут следовать одна или две шестнадцатеричные цифры. Используя данное обозначение, надо соблюдать осторожность, так как преобразуются все шестнадцатеричные цифры, расположенные после \x, но учитываются только две последние
\u*hhhh*	Символ в формате Unicode (16 битов), заданный с помощью четырех шестнадцатеричных цифр
c	Если символ *c* не указан в данной таблице, то данное выражение заменяется литеральным значением *c*. Например, последовательности \$, \", \{, \}, \] и \[используются для отмены специальных значений соответствующих символов

Арифметические операции

Таблица 1.2. Арифметические операции (приводятся по убыванию приоритета)

`- ~ !`	Унарный минус, побитовое отрицание, логическое отрицание
`* / %`	Умножение, деление, остаток от деления
`+ -`	Сложение, вычитание
`<< >>`	Сдвиг влево, сдвиг вправо
`< > <= >=`	Сравнение: меньше, больше, меньше или равно, больше или равно
`== != eq ne`	Равенство, неравенство, равенство строк (Tcl 8.4), неравенство строк (Tcl 8.4)
`&`	Побитовое И
`^`	Побитовое исключающее ИЛИ (XOR)
`|`	Побитовое ИЛИ
`&&`	Логическое И
`||`	Логическое ИЛИ
`x?y:z`	Если `x`, то `y`; иначе — `z`

Встроенные математические функции

Таблица 1.3. Встроенные математические функции в языке Tcl

`acos(x)`	Арккосинус `x`
`asin(x)`	Арксинус `x`
`atan(x)`	Арктангенс `x`
`atan2(y,x)`	При преобразовании прямоугольных координат (`x`,`y`) в полярные (`r`,`th`) `atan2` дает `th`
`ceil(x)`	Наименьшее целое значение, большее или равное `x`
`cos(x)`	Косинус `x`
`cosh(x)`	Косинус гиперболический `x`
`exp(x)`	Экспонента, e^x
`floor(x)`	Наибольшее целое значение, меньшее или равное `x`
`fmod(x,y)`	Остаток от деления `x/y` с плавающей точкой
`hypot(x,y)`	Возвращает `sqrt(x*x + y*y)`, т.е. компонент `r` полярных координат
`log(x)`	Натуральный логарифм `x`

Окончание табл. 1.3

`log10(x)`	Логарифм по основанию 10 от `x`
`pow(x,y)`	`x` в степени `y`, или x^y
`sin(x)`	Синус `x`
`sinh(x)`	Синус гиперболический `x`
`sqrt(x)`	Квадратный корень из `x`
`tan(x)`	Тангенс `x`
`tanh(x)`	Тангенс гиперболический `x`
`abs(x)`	Абсолютное значение `x`
`double(x)`	Преобразование `x` в формат с плавающей точкой
`int(x)`	Усечение `x` до целочисленного значения
`round(x)`	Округление `x` до целочисленного значения
`rand()`	Возвращает случайное число с плавающей точкой в интервале от 0.0 до 1.0
`srand(x)`	Определяет `x` как значение, используемое для инициализации генератора случайных чисел
`wide(x)`	Преобразует `x` в 64-битовое значение `wide integer`(Tcl 8.4)

Основные Tcl-команды

В табл. 1.4 содержатся сведения о Tcl-командах. Во втором столбце указан номер главы, в которой впервые встречается информация о соответствующей команде.

Таблица 1.4. Встроенные команды Tcl

Команда	**Глава**	**Описание**
`after`	16	Планирует выполнение TCL-команды на более позднее время
`append`	4	Добавляет параметры к значению переменной. Пробелы не добавляются
`array`	8	Определяет состояние массива и выполняет поиск
`binary`	4	Выполняет преобразование между строковым и двоичным представлением данных
`break`	6	Прерывает цикл до выполнения условия завершения
`catch`	6	Перехватывает ошибки
`cd`	9	Изменяет рабочий каталог

Продолжение табл. 1.4

Команда	Глава	Описание
`clock`	13	Предоставляет строки, содержащие данные о времени, и форматированное представление даты
`close`	9	Закрывает и открывает поток ввода-вывода
`concat`	5	Выполняет конкатенацию параметров, разделяя их пробелами. Объединяет списки
`console`	2	Осуществляет управление консолью, используемой для интерактивного ввода команд
`continue`	6	Вызывает немедленный переход к следующей итерации цикла
`error`	6	Эмулирует условия возникновения ошибки
`eof`	9	Проверяет наличие признака конца файла
`eval`	10	Выполняет конкатенацию параметров и интерпретирует результат как команду
`exec`	9	Порождает процесс и выполняет команду Unix
`exit`	9	Завершает процесс
`expr`	1	Выполняет математическое выражение
`fblocked`	16	Проверяет наличие данных в команде ввода
`fconfigure`	16	Предоставляет параметры канала ввода-вывода и устанавливает новые значения этих параметров
`fcopy`	17	Выполняет копирование из одного канала ввода-вывода в другой
`file`	9	Предоставляет информацию о файловой системе
`fileevent`	16	Регистрирует команду обратного вызова для поддержки обмена данными, управляемого событиями
`flush`	9	Осуществляет принудительный вывод информации из внутреннего буфера
`for`	6	Формирует цикл, действующий подобно циклу `for` в языке C
`foreach`	6	Организует перебор значений списка в цикле
`format`	4	Форматирует строку. Действует подобно `sprintf` в языке C
`gets`	9	Читает строку из входного потока
`glob`	9	Расширяет шаблон, используемый при проверке соответствия имен файлов
`global`	7	Объявляет глобальные переменные

Продолжение табл. 1.4

Команда	Глава	Описание
`history`	13	Обеспечивает работу со списком предыстории
`if`	6	Проверяет выполнение условия. Допускает наличие ветвей `else` и `elseif`
`incr`	1	Увеличивает значение переменной на величину, выражаемую целым числом
`info`	13	Предоставляет информацию о состоянии Tcl-интерпретатора
`interp`	19	Создает дополнительные Tcl-интерпретаторы
`join`	5	Осуществляет конкатенацию элементов списка, разделяя их указанной строкой
`lappend`	5	Добавляет элементы к концу списка
`lindex`	5	Возвращает элемент списка
`linsert`	5	Вставляет элемент в список
`list`	5	Оформляет параметры в виде списка
`llength`	5	Возвращает число элементов в списке
`load`	47	Загружает разделяемые библиотеки, определяющие Tcl-команды
`lrange`	5	Возвращает элементы списка в заданном диапазоне индексов
`lreplace`	5	Заменяет элемент списка
`lsearch`	5	Выполняет поиск элемента списка, соответствующего шаблону
`lset`	5	Включает элемент в состав списка (Tcl 8.4)
`lsort`	5	Выполняет сортировку списка
`namespace`	14	Позволяет создавать пространства имен и выполнять с ними различные действия
`open`	9	Открывает файл или канал
`package`	12	Реализует пакет или указывает, какие пакеты нужны приложению
`pid`	9	Возвращает идентификатор процесса
`proc`	7	Определяет Tcl-процедуру
`puts`	9	Выводит строку в выходной поток
`pwd`	9	Возвращает текущий каталог
`read`	9	Читает блок символов из входного потока

Окончание табл. 1.4

Команда	Глава	Описание
`regexp`	11	Предоставляет возможность использования регулярных выражений
`regsub`	11	Выполняет подстановку в зависимости от результатов сравнения с регулярным выражением
`rename`	7	Изменяет имя Tcl-команды
`return`	6	Возвращает значение процедуры
`scan`	4	Выполняет разбор строки в соответствии со спецификацией формата
`seek`	9	Устанавливает смещение для потока ввода-вывода
`set`	1	Присваивает значение переменной
`socket`	17	Устанавливает сетевое соединение TCP/IP
`source`	2	Выполняет Tcl-команды, содержащиеся в файле
`split`	5	Разделяет строку на элементы списка
`string`	4	Позволяет выполнять различные действия со строками
`subst`	10	Осуществляет подстановку вложенных команд и переменных
`switch`	6	Проверяет выполнение нескольких условий
`tell`	9	Возвращает текущее смещение для потока ввода-вывода
`time`	13	Определяет время выполнения команды
`trace`	13	Отслеживает присвоение значений переменным
`unknown`	12	Определяет действия при появлении неизвестной команды
`unset`	1	Удаляет переменные
`uplevel`	10	Выполняет команду в другой области видимости
`upvar`	7	Ссылается на переменную в другой области видимости
`variable`	14	Определяет переменные в пространстве имен
`vwait`	16	Ожидает, пока переменная не будет модифицирована
`while`	6	Формирует цикл, выполняющийся до тех пор, пока логическое выражение не примет значение `false`