Основы Tcl скриптов

Команды ядра Tcl поддерживают переменные, структуры контроля и процедуры. В дополнении, эти команды используются для доступа к файловой системе и сетевым разъёмам, а также для запуска других программ. Вы можете создать независимый от платформы графический интерфейс с помощью набора Tk элементов управления окном (виджетами).

Команды Tcl исполняются непосредственно после их ввода в интерактивной оболочке Tcl. Также вы можете создать скрипты (включая примеры из этой главы) в файлах и запускать их с помощью исполняемых компонентов Quartus II или с помощью оболочки **tclsh**.

Пример “Hello world”

Приведём пример “Hello world” на языке Tcl:

```
puts "Hello world"
```

Используйте двойные кавычки, чтобы сгруппировать слова hello и world в один аргумент. Двойные кавычки позволяют подстановку, встречающуюся в группах. Подстановка, которая может быть простой переменной подстановки или результатом запуска вложенной команды, описана в секции "Подстановка" на стр. 3-21. Используйте фигурные скобки {} для группировки, когда вы хотите запретить подстановку.

Переменные

Используйте команду **set** для задания значений переменной. Вы не можете использовать незадекларированную переменную. Имена Tcl переменных учитывают регистр клавиатуры. В примере 3-21 значение 1 задаётся переменной **a**.

Example 3–21. Assigning Variables

```
set a 1
```

Для доступа к содержимому переменной, используйте символ доллара **$** перед именем переменной. В примере 3-22 выводится "Hello world" другим способом.

Example 3–22. Accessing Variables

```
set a Hello
set b world
puts "$a $b"
```

Подстановка

Tcl выполняет три типа подстановки:

- Подстановка значения переменной,
- Подстановка вложенной команды,
- Подстановка обратным слешем

Подстановка значения переменной

Подстановка значения переменной, как показано в примере 3-22, ссылается на заданное значение, хранимое в переменной, используя знак доллара **$** перед именем переменной.

Подстановка вложенной команды

Подстановка вложенной команды ссылается на то, как Tcl интерпретатор вычисляет значение Tcl кода в квадратных кавычках [].Tcl интерпретатор вычисляет вложенные команды, начиная с самой дальней вложенной команды, а вложенные команды одного уровня – слева направо. Каждый результат вложенной команды подставляется во внешнюю команду.

В примере 3-23 переменной **a** задаётся длина строки **foo**.

Example 3–23. Command Substitution

```
set a [string length foo]
```

Подстановка обратным слешем

Подстановка обратным слешем позволяет вам раскрывать зарезервированные символы в Tcl, такие как доллар $ или квадратные скобки []. Также вы можете задать другие специальные ASCII символы табуляции и новой строки с помощью подстановки обратным слешем. Символ обратный слеш является Tcl символом продолжения строки, он используется, когда Tcl команда занимает более одной строки.

В примере 3-24 показано, как использовать обратный слеш для продолжения строки.

Example 3–24. Backslash Substitution

```
set this_is_a_long_variable_name [string length "Hello \
    world."]
```

Арифметика

Используйте команду **expr** для выполнения арифметических вычислений. Используйте фигурные скобки {} для группировки аргументов той команды, для которой арифметические вычисления нужно сделать более точно, чтобы сохранить заданную точность. В примере 3-25 переменной **b** задаётся сумма значений переменной **a** и квадратного корня из двух.

Example 3–25. Arithmetic with the expr Command

```
set a 5
set b [expr { $a + sqrt(2) }]
```

Tcl также поддерживает булевы операторы, такие как: && (AND), || (OR), ! (NOT), - и операторы сравнения, такие как: < (меньше), > (больше) и == (равно).

Списки

Списки Tcl – это последовательность значений. Поддерживаются операции над списками, включая создание списков, дополнение списков, выделение элементов списка, подсчёт длины списка, сортировка списка и т.д. В примере 3-26 задаётся переменная **a**, содержащая список из трёх чисел.

Example 3–26. Creating Simple Lists

```
set a { 1 2 3 }
```

Вы можете использовать команду **lindex** для выделения информации об определённом номере в списке. Индекс начинается с нуля. Вы можете использовать индекс **end** для задания последнего элемента в списке, или индекс **end-<*n*>** для отсчёта от конца списка. В примере 3-27 выводится второй элемент (с индексом 1) из списка, сохранённого в **a**.

Example 3–27. Accessing List Elements

```
puts [lindex $a 1]
```

Команда **llength** возвращает длину списка. В примере 3-28 выводится длина списка, сохранённого в **a**.

Example 3–28. List Length

```
puts [llength $a]
```

Команда **lappend** добавляет элементы в список. Если список ещё не существует, то он будет создан. Имя переменной списка не задаётся с символом доллара $. В примере 3-29 добавляются некоторые элементы к списку, сохранённому в **a**.

Example 3–29. Appending to a List

```
lappend a 4 5 6
```

Массивы

Массивы похожи на списки за исключением того, что они используют строковые индексы. Tcl массивы реализуются в виде хеш-таблицы. Вы можете создать массивы, задавая каждый элемент индивидуально или используя команду **array set**. Для задания элементу с индексом **Mon** значения **Monday** в массиве **days** используйте следующую команду:

set days(Mon) Monday

Команде **array set** необходим список пары индекс/значение. В этом примере задаётся массив с именем **days**:

*array set days { Sun Sunday Mon Monday Tue Tuesday *
Wed Wednesday Thu Thursday Fri Friday Sat Saturday }

В примере 3-30 показано, как доступны значения для соответствующего индекса.

Example 3–30. Accessing Array Elements

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Используйте команду **array names** для получения списка всех индексов в соответствующем массиве. Значения индексов не могут выводиться в любом порядке. В примере 3-31 показан один из способов итерации всех значений массива.

Example 3–31. Iterating Over Arrays

```
foreach day [array names days] {
   puts "The abbreviation $day corresponds to the day \
name $days($day)"
}
```

Массивы – очень гибкий способ хранения информации в Tcl скрипте и хороший способ сборки сложных структур данных.

Контролирующие структуры

Tcl поддерживает общие контролирующие структуры, используя условия **if-then-else** и циклы **for**, **foreach** и **while**. Расстановка фигурных скобок, показанная в следующем примере, отслеживает эффективность исполнения и корректность команд контролирующих структур. В примере 3-32 выводится значение переменной, когда оно положительное, отрицательное или нуль.

Example 3–32. If-Then-Else Structure

```
if { $a > 0 } {
   puts "The value is positive"
} elseif { $a < 0 } {
   puts "The value is negative"
} else {
   puts "The value is zero"
}
```

В примере 3-33 используется цикл **for** для вывода каждого элемента списка.

Example 3–33. For Loop

```
set a { 1 2 3 }
for { set i 0 } { $i < [llength $a] } { incr i } {
   puts "The list element at index $i is [lindex $a $i]"
}
```

В примере 3-34 используется цикл **foreach** для вывода каждого элемента списка.

Example 3–34. foreach Loop

```
set a { 1 2 3 }
foreach element $a {
   puts "The list element is $element"
}
```

В примере 3-35 используется цикл **while** для вывода каждого элемента списка.

Example 3–35. while Loop

```
set a { 1 2 3 }
set i 0
while { $i < [llength $a] } {
    puts "The list element at index $i is [lindex $a $i]"
    incr i
}
```

Не используйте команду **expr** в булевых соотношениях в командах контролирующих структур, поскольку они автоматически вызывают команду **expr**.

Процедуры

Используйте команду **proc** для определения Tcl процедуры (известную как подпрограмма или функция в других языках скриптирования или программирования). Область действия переменных в процедуре ограничена процедурой. Если процедура возвращает значение, используйте команду **return** для возвращения значения из процедуры. В примере 3-36 определена процедура, которая умножает два числа и возвращает результат.

Example 3–36. Simple Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
```

В примере 3-37 показано, как использовать процедуру умножения в вашем коде. Вы должны определить процедуру до её вызова в скрипте.

Example 3–37. Using a Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
set a 1
set b 2
puts [multiply $a $b]
```

Вам нужно определять процедуры сразу после начала скрипта. Если вы хотите иметь доступ к глобальным переменным в процедуре, используйте команду **global** в каждой процедуре, которая использует глобальные переменные. В примере 3-38 определяется процедура, которая выводит элемент из глобального списка чисел, которые вызываются процедурой.

Example 3–38. Accessing Global Variables

```
proc print_global_list_element { i } {
    global my_data
    puts "The list element at index $i is [lindex $my_data $i]"
}
set my_data { 1 2 3}
print_global_list_element 0
```

Входные/выходные файлы

Tcl содержит команды для чтения и записи файлов. Вы должны открыть файл прежде чем читать или писать в него, и закрыть его, если вы завершили операции чтения или записи. Для открытия файла используйте команду **open**; для закрытия файла используйте команду **close**. Когда вы открываете файл, задайте его имя и режим, в котором вы его открываете. Если вы не задаёте режим, Tcl по умолчанию устанавливает режим чтения. Для записи в файл, задайте **w** в качестве режима записи, как показано в примере 3-39.

Example 3–39. Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl поддерживает и другие режимы, включая добавление к существующим файлам и чтения и запись из одного файла. Команда **open** возвращает дескриптор файла, который используется для доступа к чтению или записи. Вы можете использовать команду **puts** для записи в файл, определённый дескриптором файла, как это показано в примере 3-40.

Example 3–40. Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

Вы можете читать за один раз одну строку из файла, используя команду **gets**. В примере 3-41 используется команда **gets** для чтения каждой строки в файле и вывода её с её номером.

Example 3–41. Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

Синтаксис и комментарии

Аргументы команд Tcl разделяются пробелом, а сами Tcl команды завершаются символом новой строки или точкой с запятой. Как показано в секции "Подстановка" на стр. 3-21, вы должны использовать обратный слеш, когда Tcl команда занимает более одной строки.

Tcl использует символ решётки # для начала комментариев. Символ # должен начинать комментарий. Если вы предпочитаете включать комментарии на той же строке, что и команда, следите за тем, чтобы команда заканчивалась точкой с запятой перед символом решётки. В примере 3-42 показана правильная строка кода, содержащего команду **set** и комментарий.

Example 3–42. Comments

```
set a 1;# Initializes a
```

Без точки с запятой, может получиться неправильная команда, поскольку команда **set** не сможет завершиться, пока символ новой строки находится после комментария.

Интерпретатор Tcl считает фигурные скобки внутри комментариев, которые могут привести к ошибкам, так как создают затруднения для анализа. В примере 3-43 выводится ошибка из-за несбалансированных фигурных скобок.

Example 3–43. Unbalanced Braces in Comments

```
# if { $x > 0 } {
if { $y > 0 } {
    # code here
}
```