**Programs**  
**1. Find the length of a string without using string length command**  
  
set s "Arun"  
set len 0  
foreach item [split $s ""] {incr len}  
puts "The lenthe of the string is : $len"  
In the above program, the a string is converted to a list [split $s ""] and then for each letter in the word (element in the list) the variable len is incremented until it reaches the end. Thus len contains the string length.  
  
  
 **List Command:**  
List command returns a list comprising of all the arguments specified.  
  
Syntax:  
**list** arg1 arg2....  
  
Example:  
list a b "c d e  " "   f { g h }"  
returns : a b {c d e } {  f {gh} }  
======================================================================  
**Lappend Command:**  
lappend command treats var1 as a list and appends all the values to the list with a space between elements. If var1 does not exist then a new list called var1 is created with the elements value1, value2, value3 etc.  
  
Syntax:  
**lappend** var1 value1 value2 value3...  
set var1 1  
1  
lappend var1 2  
1 2  
lappend var1 3 4 5   
1 2 3 4 5

======================================================================

**Easiest Example To Explain Upvar Command**

**Usage:**

Used when we have to change the value of a global variable from inside a procedure’s scope

**Example**

proc example {one two} {

upvar $one local1

upvar $two local2

set local1 Kavitha

set local2 Anbarasu

}

set glob1 David

set glob2 Beckam

puts $glob1

puts $glob2\n

example glob1 glob2

puts $glob1

puts $glob2

**Output**

David  
Beckam  
  
Kavitha  
Anbarasu  
  
In the above example we are able to change the value of two global variables glob1 and glob2 from within a procedure  
  
======================================================================  
**regsub Command Usage**

**Regsub Usage**

Replaces a value with a given value

**Syntax:**

regsub “”  $string “”

**Example:**

set string "My name is Kavitha"

regsub "Kavitha" $string "Mythri" name2

puts "$name2"  
  
**Output**  
  
My name is Mythri

======================================================================  
  
**:? Command Usage**

**Usage:**

?: is used in sub patterns in a regexp

When ever you don’t want a particular subpattern to be included as a sub-pattern use “?:” in front of the sub-pattern

**Example:**

set string "Projects: Brocade Cisco Fujitsu"

regexp "Projects: (Brocade|Cisco) (?:Fujitsu|Juniper|Cisco) (Fujitsu|Juniper)" $string sub1 sub2 sub3

puts "$sub1\n$sub2\n$sub3\n"

In the above example, the output will be

**Projects: Brocade Cisco Fujitsu**

**Brocade**

**Fujitsu**

The pattern “Cisco” does not come under sub pattern as “?:” is given

The output without ?: would be

**Projects: Brocade Cisco Fujitsu**

**Brocade**

**Cisco**

======================================================================

**Different String Commands In TCL**

set one “np”  
set two “gqaxbnp”

puts [string compare $one $two]

puts [string length $one]

puts [string index $two 2]

puts [string first $one $two]

puts [string last $one $two]

o/p:

1

2

a

5

5

**To Know More About What Happens in a TCL Network Testing Project and the Issues Faced And Their Resolutions Refer The Below url**  
  
<http://tclautomationnetworktesting.blogspot.in/>   
  
  
  
==================================================  
Though it is not relevant, just an addition information on a different subject :)  
To know about taking care of babies click here  
  
[http://kavitha-allaboutbabies.blogspot.in](http://kavitha-allaboutbabies.blogspot.in/)   
  
==================================================

Posted by Kavitha at [2:00 AM](http://tclinterviewquestions.blogspot.in/2012/01/answers-to-tcl-interview-questions.html) [2 comments](http://tclinterviewquestions.blogspot.in/2012/01/answers-to-tcl-interview-questions.html#comment-form)

[Email This](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=5012294749161699450&target=email)[BlogThis!](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=5012294749161699450&target=blog)[Share to Twitter](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=5012294749161699450&target=twitter)[Share to Facebook](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=5012294749161699450&target=facebook)[Share to Pinterest](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=5012294749161699450&target=pinterest)

**Thursday, December 1, 2011**

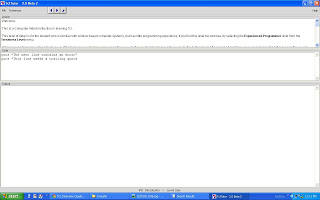
**[General Interview Tips](http://tclinterviewquestions.blogspot.in/2011/12/general-interview-tips.html)**

For any interview, the first thing to prepare is the resume. It creates an impression about you even before meeting you. Make it look neat, short and crisp. Do not leave the achievements column blank. Write a brief overview about your previous projects. Do not boast, but make diplomatic statements that will show that you are capable.  
  
Next would be your appearance. Neatly pressed formal clothes would be mandatory for an interview. Be confident when you approach the interviewer. Never show your nervousness out. Be very honest when you answer the question. Always be ready to say “I am not aware of this” when you don’t know something (But not for all questions ofcourse! :)). Never give wrong answers. Listen carefully and understand the questions. Prepare well to talk all about your project and your role that you have stated in your resume. Do not say anything negative about your current employer. Be very clear with your concepts and never mess up with that.

Posted by Kavitha at [10:51 PM](http://tclinterviewquestions.blogspot.in/2011/12/general-interview-tips.html) [2 comments](http://tclinterviewquestions.blogspot.in/2011/12/general-interview-tips.html#comment-form)

[Email This](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=4900764540666975577&target=email)[BlogThis!](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=4900764540666975577&target=blog)[Share to Twitter](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=4900764540666975577&target=twitter)[Share to Facebook](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=4900764540666975577&target=facebook)[Share to Pinterest](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=4900764540666975577&target=pinterest)

**[TCL Tutorial](http://tclinterviewquestions.blogspot.in/2011/12/tcl-tutor.html)**

[](http://3.bp.blogspot.com/-boCLM7DAmpQ/Tthz3NJoF3I/AAAAAAAAAZg/1E79h_CNNHk/s1600/tcltutor.JPG)  
If you are beginner in TCL and would like to teach yourself this simple language, you can do it using a tcl tutor. You need not buy expensive books or search the net for online materials to learn the language. All you have to do is, download tcl tutor from the below link and start working on it from the first chapter

[http://www.msen.com/~clif/TclTutor.html#download](http://www.msen.com/%7Eclif/TclTutor.html#download)

TCL tutor is designed in such a way that you can learn concepts and keywords from the chapters listed in the tutor and at the same time execute them and look for the output. Sample code will be available in each chapter which can be executed and you can also write your own code and try to execute it and look for the output.

To start with, beginners can go through the tutor chapter by chapter and execute them and understand the concepts. Experienced programmers working on TCL projects can also use this tutor. You can go to the particular concept or keyword that you have problem with, and refer the tutorial. You can also copy paste your piece of code that you have an issue with and execute them in the tutor to find out the flow and locate your mistake. TCL is mostly used for automation testing and hence if you are stuck up with a particular line in the script due to syntax errors or any other issues, you can verify them using the tutor and then make changes accordingly and then go for the next execution, instead of running the whole script again to find another syntax error in some other line! This way it is a big gift for TCL programmers.

Posted by Kavitha at [10:27 PM](http://tclinterviewquestions.blogspot.in/2011/12/tcl-tutor.html) [0 comments](http://tclinterviewquestions.blogspot.in/2011/12/tcl-tutor.html#comment-form)

[Email This](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=905091440140765091&target=email)[BlogThis!](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=905091440140765091&target=blog)[Share to Twitter](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=905091440140765091&target=twitter)[Share to Facebook](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=905091440140765091&target=facebook)[Share to Pinterest](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=905091440140765091&target=pinterest)

**[Some Useful Links To Refer Before Attending Software Testing Interviews](http://tclinterviewquestions.blogspot.in/2011/12/some-useful-links-to-refer-before.html)**

I found these links really useful for an interview (both as an interviewer and as an interviewee)

These links are with respect to software testing.

<http://testinginterviewquestionsandanswers.com/what-is-a-test-case.html>

<http://www.coolinterview.com/type.asp?iType=197>

<http://www.softwaretestinghelp.com/category/software-testing-interview-questions-with-answers/>

<http://www.allinterview.com/Interview-Questions/Manual-Testing.html>

<http://www.softwaretestingnow.com/software-testing-interview-questions-and-answers>

<http://futurethoughtsllc.com/TestingandQA.aspx>

Posted by Kavitha at [8:55 PM](http://tclinterviewquestions.blogspot.in/2011/12/some-useful-links-to-refer-before.html) [0 comments](http://tclinterviewquestions.blogspot.in/2011/12/some-useful-links-to-refer-before.html#comment-form)

[Email This](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=8539906242252348711&target=email)[BlogThis!](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=8539906242252348711&target=blog)[Share to Twitter](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=8539906242252348711&target=twitter)[Share to Facebook](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=8539906242252348711&target=facebook)[Share to Pinterest](http://www.blogger.com/share-post.g?blogID=9159321241918232441&postID=8539906242252348711&target=pinterest)

**[Testing Interview Questions And Answers](http://tclinterviewquestions.blogspot.in/2011/12/answers-to-questions.html)**

These are the answers to the "software testing" questions:-  
  
1. Difference between black box and white box testing.

Black box Testing:

Black box testing is the testing where the tester is unaware of the code and structures inside the application that is tested. The tester knows only the valid inputs supplied and the expected outputs corresponding to those inputs. If the expected result is not obtained, the tester files a bug.

White Box Testing:

White Box testing is one where the tester is aware of the internal code flow and structures and he tries to test if all the modules are hit properly when different inputs are supplied. In most cases developers perform the white box testing as they are aware of the modules they have created.

2. Who does white box testing(Developers or testers)?

Black box testing is performed by testers

White box testing is mostly performed by developers

3. Why the name 'black box' and 'white box' testing?

If we look at the application as a single box, in the case of black box testing, the tester is unaware of the code structures inside and hence the application looks like a black box to him with just the input going into it and the output that is expected to come out of it.

In the case of a white box testing, the tester (in fact the developer) knows the application in and out including the code structures. So the applications looks like a white transparent box to him, where he can see the modules in it and test if all the modules are hit properly and the code is optimized.

4. Is unit testing white box or black box?

Testing an individual module of an application is called unite testing.

Developers perform unit tests and therfore it comes under white box testing.

In white box testing the basic path coverage, loops and functions written in the code are tested

On the other hand, black box testing test the functionality of an application

5. Difference between regression and re-testing?

Regression Testing

When we perform testing after making changes to the current build it is called regression testing. In other words regression testing is performed time and again on different builds to see if the code changes in the subsequent builds affect the output.

Retesting

On the other hand, when we perform testing time and again on the same build it is called retesting. Sometimes an application might behave in an inconsistent way giving different outputs when we retest. To capture such bugs we do retesting.

6. Why is testing essential for a project? What is the significance of testing?

Testing is an integral part of a project as it improves the quality of an application that is created. Any basic functionality that is left out during the development phase can be captured during the testing phase. A developer over looks certain issues as he does not want his code to break. But a tester's main aim is to break the application when a valid input is given, which means the application has a bug and needs to be fixed.

7. Name a bug that you had raised in your project that was appreciated?

Think of something that brought you several appreciation mails from clients. Something that the developer had really missed and if delivered as is, could have caused a big blow.  
  
  
Difference between Verification and Validation testing is a very common question in any testing interview. I had faced this question time and again and was really very much interested to shoot

**#reverse a string in tcl**  
set a 12345  
set b [string length $a]  
puts $b  
for {set i $b} {$i >= 0} {incr i -1} {  
puts -nonewline [string index $a [expr $i -1]]  
}  
  
Another method:  
set string "nawrajlekhak"  
set rev [string reverse $string]  
puts $rev  
  
**#reverse a list in tcl**  
set list "5 7 66 2 1"  
set len [llength $list]  
#puts $len  
for {set i $len} {$i > 0} {incr i -1} {  
set b [lindex $list [expr $i - 1]]  
puts -nonewline "$b "  
}  
  
**#Remove duplicate entries from the list**  
set list "1 2 3 4 5 6 7 8 11 17 33 3 4 8"  
set b [lsort -unique $list]  
puts $b  
  
**#Find the number of elements in the list without using llength**  
set list "1 2 3 4 5 6 7 8 11 17 33 3 4 8"  
set i 0  
foreach ele $list {  
set b "$ele = $i"  
incr i  
}  
puts $i  
  
**#Print all elements in the list using flow controls**  
set list "1 2 3 4 5 6 7 8 11 17 33 3 4 8"  
set len [llength $list]  
puts $len  
for {set i 0} {$i < $len} {incr i} {  
puts -nonewline "[lindex $list $i] "  
}  
  
**#Create array and print all elements in array**  
array set arrayvar {1 one 2 two 3 three}  
puts [parray arrayvar]  
puts [array names arrayvar]  
puts [array size arrayvar]  
puts [array get arrayvar]  
puts [array exists arrayvar]  
  
**#Reverse a string**  
set string "nawrajlekhak"  
set rev [string reverse $string]  
puts $rev  
  
**#Write a program to find given string is palindrome or not**  
set a madam  
set len [string length $a]  
set n [expr ($len-1)/2]  
for {set i 0} {$i < $n} {incr i} {  
set b [string index $a $i]  
set c [expr $len - 1 - $i]  
set d [string index $a $c]  
if {$b != $d} {  
puts "not palindrome"  
exit  
}  
}  
puts "palindrome"  
  
  
**#Write a regsub command to replace all “.” in ip address to “\_”**  
set ip "192.168.10.1"  
set b [string map {. \_} $ip]  
puts $b  
  
**#Create a package**  
package require TclUtils  
  
**#fabonacci series.**  
set fib1 0  
set fib2 1  
set s ""  
for {set i 0} {$i < 8} {incr i} {  
    set fib3 [expr {$fib1 + $fib2}]  
    set fib1 $fib2  
    set fib2 $fib3  
    append s "$fib1, "  
}  
puts "$s"  
  
  
**#Factorial:**  
set fact 1  
for {set i 0} {$i <= 16} {incr i} {  
    puts "$i! = $fact"  
    set fact [expr {$fact \* ($i + 1)}]  
}  
 **#switch**  
set x 1  
switch $x {  
  "1" {puts "hello"}  
    "2"  {puts "hi"}  
    "default" {puts "wrong value"}  
}  
 **#matching any ip address**  
set str 192.168.3.110  
regexp {(^[0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})$} $str all first second third fourth  
puts "$all \n $first \n $second \n $third \n $fourth \n"  
if {$first <= 255 && $second <= 255 && $third <= 255 && $fourth <= 255} {  
puts "valid ip address"  
} else {  
puts "invlaid ip address"  
}  
  
**#How to swap two numbers without using swap command**  
set a 100  
set b 12  
set a [expr $a + $b]  
set b [expr $a - $b]  
puts "b = $b"  
set a [expr $a - $b]  
puts "a = $a"  
  
**#How to multiply two numbers without using multiplication operator.**  
set a 0  
set b 5  
set s 0  
for {set i 1} {$i <= $b} {incr i} {  
set s [expr $a + $s]  
}  
puts $s  
  
**#convert ascii to char**  
proc tochar ascii {  
set b [format %c $ascii]  
puts $b  
}  
tochar 65  
  
**#convert char to ascii**  
proc toascii char {  
scan $char %c ascii  
puts $ascii  
}  
toascii A  
  
**#Read**  
 set fp [open "somefile" r]  
 set file\_data [read $fp]  
 close $fp  
  
set fp [open "somefile" r+]  
 set file\_data [read $fp]  
 set data "hello how r u"  
 puts -no newline $fp $data  
 close $fp  
  
**#Write**  
set data "This is some test data.\n"  
set filename "test.txt"  
set fileId [open $filename "w"]  
puts -nonewline $fileId $data  
close $fileId  
  
**#append**  
#Open the file called "jokes.txt" for writing  
set out [open "jokes.txt" w]  
puts $out "Computers make very fast, very accurate mistakes."  
close $out  
#Now append more jokes at the end of the file  
set out [open "jokes.txt" a]  
puts $out "Computers are not intelligent. They only think they are."  
puts $out "My software never has bugs. It just develops random features."  
puts $out {All computers wait at the same speed.  
Best file compression around:  "DEL \*.\*" = 100% compression  
DEFINITION: Computer - A device designed to speed and automate errors.  
DEFINITION: Upgrade - Take old bugs out, put new ones in.}  
close $out  
  
**#what does these variable means.**  
set a [info vars]  
puts $a  
puts $argv0  
puts $tcl\_version  
puts $tcl\_interactive  
puts $errorCode  
puts $auto\_path  
puts \*\*\*$errorInfo\*\*  
puts $argc  
puts $tcl\_libPath  
puts $tcl\_library  
**#Output**  
argv argv0 tcl\_version tcl\_interactive errorCode auto\_path errorInfo env tcl\_patchLevel argc tcl\_libPath tcl\_platform tcl\_library  
C:/Users/Nawraj/Desktop/tcltutor30b6.exe/main.tcl  
8.6  
0  
TCL LOOKUP VARNAME ::tcl\_pkgPath  
C:/Users/Nawraj/Desktop/tcltutor30b6.exe/lib/tcl8.6 C:/Users/Nawraj/Desktop/tcltutor30b6.exe/lib  
\*\*\*can't read "::tcl\_pkgPath": no such variable  
    while executing  
"foreach Dir $::tcl\_pkgPath {  
    if {$Dir ni $::auto\_path} {  
 lappend ::auto\_path $Dir  
    }  
 }"\*\*  
0  
C:/Users/Nawraj/Desktop/tcltutor30b6.exe/lib/tcl8.6 C:/Users/Nawraj/Desktop/tcltutor30b6.exe/lib  
C:/Users/Nawraj/Desktop/tcltutor30b6.exe/lib/tcl8.6  
  
**#How to verify whether system installed with 32 bit or 64 bit TCL library**  
% parray tcl\_platform  
tcl\_platform(byteOrder) = littleEndian  
tcl\_platform(machine)   = i686  
tcl\_platform(os)        = Linux  
tcl\_platform(osVersion) = 2.6.18-164.el5  
tcl\_platform(platform) = unix  
tcl\_platform(threaded) = 1  
tcl\_platform(user)      = vijaym  
tcl\_platform(wordSize) = 4  
  
tcl\_platform(wordSize) = 4  - Express that this is 32 bit TCL, if this is 8 then 64 bit TCL.  
 **#Check where the tcl package has installed the shared library:**  
rpm -ql tcl | grep lib  
  
**#What do you mean by wrapper.**  
It bundles a base application such as protclsh or prowish, Tcl scripts, system library files, and any other files needed by your application into a single "wrapped" file that is completely self-contained. You can distribute this file to your customers and they can execute it directly without needing any installation or any preexisting facilities.  
All-in-one output. TclPro Wrapper creates a single wrapped file that contains everything needed to run a single application.  
Flexible sharing. Normally, TclPro Wrapper incorporates everything needed by the application into the wrapped file. However, you can choose to leave out shared library facilities if you know that the application will only be used in environments where the libraries are available. This reduces the size of the wrapped applications while still providing the convenience of bundling several source files into a single wrapped file.  
  
**#Simplest call from C to Tcl - just a wrapper**  
/\* Well House Consultant - building TCL into a C application \*/  
/\* This example is just a wrapper around a TCL interpreter \*/  
#include   
#include   
main (int argc, char \*argv[]) {  
        Tcl\_Interp \*myinterp;  
        int status;  
        printf ("Your Tcl Program will run ... \n");  
        myinterp = Tcl\_CreateInterp();  
        status = Tcl\_EvalFile(myinterp,argv[1]);  
        printf ("Your Tcl Program has completed\n");  
}  
  
**#What steps should be followed if your script fails during execution.**  
debug [[-now] 0|1] controls a Tcl debugger allowing you to step through statements, set breakpoints, etc.  
With no arguments, a 1 is returned if the debugger is not running, otherwise a 0 is returned. With a 1 argument, the debugger is started. With a 0 argument, the debugger is stopped. If a 1 argument is preceded by the -now flag, the debugger is started immediately (i.e., in the middle of the debug command itself). Otherwise, the debugger is started with the next Tcl statement.The debug command does not change any traps. Compare this to starting Expect with the -D flag.  
  
**#to match any email address**  
set a "Raj\_btech23@rediffmail.com"  
if {[regexp -- {^[A-Za-z0-9.\_-]+@[[A-Za-z0-9.-]+$} $a b]} {  
puts $b  
} else {  
puts fail  
}  
#puts $a  
  
**#To match any url**  
set url https://groups.google.com  
regexp -all {^[a-z]+\:\/\/[0-9.a-z\_/]+} $url new  
puts $new  
  
**#Generate random no in TCL**  
set random\_number [expr int(rand()\*10)]  
  
**#How do I convert a list into string in Tcl?**  
set list {a b c d e f}  
for {set i 0} {$i<[llength $list]} {incr i} {  
    append string [lindex $list $i]  
}  
puts $string  
  
**#Write a Procedure to sum two numbers with one variable has a default value.**  
proc sum {a {b 5}} {  
set c [expr $a+$b]  
puts $c  
}  
sum 1  
  
**#call by value**  
proc print\_hello {count} {  
for {} {$count>0} {incr count -1} {  
puts "hello world"  
}  
}  
set var 2  
print\_hello $var  
#Output  
hello world  
hello world  
  
**#call by reference**  
proc print\_hello {count} {  
upvar $count v  
set v 2  
for {} {$v > 0} {incr v -1} {  
puts "hello world"  
}  
}  
print\_hello var  
#Output  
hello world  
hello world  
  
**#factorial**  
proc fact {n} {  
if {$n==0||$n==1} {  
return 1  
} else {  
return [expr {$n\*[fact [expr {$n-1}]]}]  
}  
}  
set a [fact 4]  
puts $a  
  
**#To display powers of integers**  
set message "";  
set p 2  
foreach k {1 2 3 4} {  
append message "\$k to the power \$p is [expr pow(\$k,\$p)]\n"  
}  
set message  
=> 1 to the power 2 is 1.0  
2 to the power 2 is 4.0  
3 to the power 2 is 9.0  
4 to the power 2 is 16.0  
  
**#Loop Control : break and continue**  
The break command causes the innermost enclosing looping command to terminate immediately.  
The continue command causes only the current iteration of the innermost loop to be terminated. The loop continues with next iteration.  
#continue  
for {set x 0} {$x < 10} {incr x} {  
if {$x == 5} {  
   continue  
   }  
   puts "x is $x"  
 }  
#output  
x is 0  
x is 1  
x is 2  
x is 3  
x is 4  
x is 6  
x is 7  
x is 8  
x is 9  
  
#break  
for {set x 0} {$x < 10} {incr x} {  
if {$x == 5} {  
   break  
   }  
   puts "x is $x"  
 }  
#output  
x is 0  
x is 1  
x is 2  
x is 3  
x is 4  
  
**#What is difference between lappend and concat**  
set list1 {1 2 3}  
puts $list1  
set list2 {a b c}  
puts $list2  
set new [lappend list1 $list2]  
puts $new  
set lengthlist [llength $new]  
puts $lengthlist  
  
#output  
1 2 3  
a b c  
1 2 3 {a b c}  
4  
  
set list1 {1 2 3}  
puts $list1  
set list2 {a b c}  
puts $list2  
set b [concat $list1 $list2]  
puts $b  
set concatlength [llength $b]  
puts $concatlength  
  
#output  
1 2 3  
a b c  
1 2 3 a b c  
6  
  
**Note: length of output of lappend is 4 whereas for output of concat is 6**  
 **#Package**  
Tcl supports grouping multiple procedures or compiled "C" code modules into a single entity referred to as a package. A package may consist of a single file, multiple files or a mix of compiled and Tcl script files.  
 Some packages (like http and Tk) are provided in the base Tcl distribution, and many more are included in the tcllib collection. (See tcllib.sourceforge.net)  
  
 To access a package from a script:   
1. Make certain that the auto\_path global variable includes the path to the package you wish to include. The standard Tcl distributions include the proper paths by default.  
2. Include a line like this in the application that uses a package:  
**package require nameOfPackage**  
 You script will need a separate package require command for each package you wish to include.   
  
 To create a package, you'll need   
1. to include a package provide command like  
**package provide nameOfPackage revisionNumber**  
 in each file that's included in the package  
2. create a pkgIndex.tcl file to describe the files in the package.  
 This can be done with the pkg\_mkIndex command.   
  
package require name ?revision?  
Tells the interpreter to find a package with the given name and optional revision number. If the revision number is absent, the largest revision available is used.  
**package provide  name revision**  
Declares that this file provides an implementation of a specific revision of a package. A file may include only one package provide command.  
pkg\_mkIndex libdir file1 ... filen  
Creates an index file (pkgIndex.tcl) in libdir from the source code modules listed as file1 - filen. Each proc in the files will be listed in the index file, with a reference to the source code module that contains it.  
The file descriptors may be any number of strings using the same format as arguments to the glob command.  Two other commands are related to finding packages and processing them. Your code will not need to use these, but you may find them useful in understanding how the system behaves.  
  
**info library**  
Returns the name of the library directory in which the standard Tcl scripts are stored.  
  
Tcl procs that are in files in the Tcl library directory are loaded automatically when they are executed.  
Tcl expects to find init.tcl in the default library location, defined when the Tcl interpreter is built. This path can be changed by setting the environment variable TCL\_LIBRARY.  
unknown args  
This proc is called by the interpreter when it encounters a command that can be parsed, but the command name is not in the tables. Unknown attempts the following steps to execute the command   
\*Checks the index in the directories listed in auto\_path to see if the proc exists in a file. If the proc is found, the script is loaded with the auto\_load proc, and the proc is executed.  
\*If the interpreter is running interactively, Tcl attempts to exec the command with the auto\_exec proc.  
\*If the command is a unique prefix to a Tcl command, unknown completes the command name, and executes that command   
  
**#tclvars - Variables used by Tcl**  
http://tmml.sourceforge.net/doc/tcl/tclvars.html  
  
**#tclsh - Simple shell containing Tcl interpreter**  
http://tmml.sourceforge.net/doc/tcl/tclsh.html

**#Difference between TCL and Perl**

Tcl is a pure interpreter; Perl uses a bytecode engine

Tcl is a scripting language used for extending and controlling applications.

TCL is embeddable

Its interpretor is library of C procedures that can be incorporated in the applications.

TCL gives the benefit of rapid development(functions are already present we just need to use them).

Perl(Practical extraction and report language) is an interpreted language.

Usually used for text and data manipulation.

Majorly used by system administrators and web programmers.

Perl is composition of sed, awk, C, Shell and English programmers.

**#Generally how many LOC a coder can code in a day.**

"Applying our TAO-Extreme-XP-Pattern-Pair-Agile-Y2K-Guru development methodologies allows our programmers to produce 1000 LOC per a day, while the industry average is only 250 LOC per day"

"TAO-Extreme-XP-Pattern-Pair-Agile-Y2K-Guru development methodologies allow our programmers to produce code with only 3.2 defects per 100 LOC, while the industry standard is 3.7 defects per 100 LOC."

**#Namespce**

namespace bascially will be written inside packages

say u want to have telent session for two different routers say r1 and r2 then u will write one function say

telnet

but while calling telnet session for r1 ..u will use namespace r1::telent

for telent session of r2 ..use namespace r2::telnet

**#Perl: Compiler or Interpreter?**

A compiler is a program that reads a source program and converts it into object code. This object code is then used

by the Linker to produce an executable file. Both the object code and executable files are in binary form. This

means that if you try to read them all you can see are garbage and some of the literals used in the program. The

compile-link process ensures that the program is free from any syntax errors. It cannot determine logic errors in

the program.

An interpreter is a program that reads a source program and executes the source program in real time. This cuts

down the time to prepare the program as you do not need to compile and link the program. The interpreter executes the instructions as it reads the file. This however means that syntax errors can cause the program to abort in the middle of execution. This can be very frustrating and dangerous specially if your program has been running for quite a while.

Perl is a hybrid of the two. When a program is executed, perl reads the program and builds a binary file at the

same time, checking for syntax errors. If there are no errors, it then uses the binary file to execute the

instructions. Syntax errors will cause Perl to terminate BEFORE it goes on to execute the program. In a sense, Perl gives you the best of both worlds.

**#Difference in tcsh and tclsh**

tcsh (the Enhanced/Tenex C Shell) or tclsh (TCL shell)

**#Diff in braces**

{ } used to defer the expansion of variables,

commands, and backslash characters until the code

executes

set i 0

while {$i < 10} {code to execute}

is not the same as

set i 0

while ”$i < 10” ”code to execute”

which will always evaluate as true.

**# Convert a single hex digit to decimal**

set hex a

set decimal [string first [string toupper $hex] "0123456789ABCDEF"]

puts $decimal

if {$decimal == -1} {

puts "Hey, $hex isn't a hex digit"

}

**# Convert a decimal number in the range of 0 to 15 to a hex character**

set decimal 13

set hex [string index "0123456789ABCDEF" $decimal]

puts $hex

if {$hex == {}} {

puts "Hey, $decimal isn't in the range 0 to 15"

}

**# Check for runt (<64 and="" bytes="" giant=""> mtu) packets**

set mtu 1500

set packet "This is a simulated packetAAAA08..."

set packetlen [string length $packet]

if {$packetlen < 64} {

puts "Hey, this packet is a runt"

}

if {$packetlen > $mtu} {

puts "Hey, this packet is a giant"

}

**#:? Command Usage**

Usage:

?: is used in sub patterns in a regexp

When ever you don’t want a particular subpattern to be included as a sub-pattern use “?:” in front of the sub-pattern

Example:

set string "Names: Nawraj Raj Lekhak"

regexp "Names: (Nawraj|Raj) (?:Lekhak|Dinesh|Raj) (Lekhak|Dinesh)" $string match sub1 sub2 sub3

puts "$match\n$sub1\n$sub2\n$sub3\n"

In the above example, the output will be

Names: Nawraj Raj Lekhak

Nawraj

Lekhak

**#TCL Program – Check given number is odd or even in tcl**

proc oddeven {n} {

if {$n%2==0} {

puts “the given $n is even” } else {

puts “the given number $n is odd” }

}

oddeven 24

oddeven 67

o/p:

the given 24 is even

the given number 67 is odd

**# How to increment each elements in a list**

set list "1 2 3"

foreach ele $list {

puts [incr ele]

}

**#How to increment a char**

set no a

set new [expr [scan $no %c] + 1]

puts $new

set value [format %c $new]

puts $value

Output:

98

b

**#Loop**

Next small script will run for a specified times specified command and wait specified time. Variable repeat is used to specify number of loops. Variable command is a IOS command you want to run and variable wait says, how long to wait before executing the same command in a loop.

# Loop

# Run specified command in a loop for X times and wait Y ms after every execution.

set repeat 10

set command "show ip interface brief"

set wait 1000

for { set i 1 } { $i <= $repeat } { incr i } {

puts "\n===========================\nShowing: $i/$repeat\n==========================="

puts $command

after $wait

}

**Aliaser**

Just a small TCL script that demonstrates, how is switch command used. You will run this command in an infinite loop, until “q” is typed. You can use number 1, 2 and 3. You can add any character you want. After typing character, press enter.

# Aliaser

#

while {1}  {

switch [read stdin 1] {

"1" {show ip route}

"2" {show ip protocols}

"3" {show ip int brie \| i up}

"q" {break}

"\n" {puts ""}

default {puts "Unknown command, type \"q\" for quit"}

}

}

1.how to increment eacl element in a list ? eg: incrlist {1 2 3} =>2 3 4  
set list1 {1 2 3}  
set list2 {}  
foreach i $list1 {  
lappend list2 [expr {$i+1}] }  
puts $list2  
o/p:  
2 3 4  
2.How do you find the length of a string without using string length command in TCL  
set var “welcome”  
set list1 [split $var ""]  
foreach i $list1 {  
incr len }  
puts $len  
o/p:  
7  
3.How to extract “information” from “ccccccccaaabbbbaaaabbinformationabcaaaaaabbbbbbbccbb”  
in tcl using a single command  
puts [string trim "ccccccccaaabbbbaaaabbinformationabcaaaaaabbbbbbbccbb" "abc"]  
o/p:  
information  
4.How to Swap 30 & 40 in IP address 192.30.40.1 using TCL script  
set var “192.30.40.1″  
set list1 [split $var "."]  
set list2 [lreplace $list1 1 2 40 30]  
set result [join $list2 "."]  
puts $result  
o/p:  
192.40.30.1  
5.Set ip address as 10.30.20.1 write a script to replace the 30 with 40  
set var “10.30.20.1″  
regsub 30 $var 40 result  
puts $result  
o/p:  
10.40.20.1  
6.How do you check whether a string is palindrome or not using TCL script  
proc palindrome {str} {  
set l [string length $str]  
set i 0  
incr l -1  
set flag 0  
while {$l>=0} {  
set s [string index $str $i]  
set e [string index $str $l]  
if {$s==$e} {  } else {  
set flag 1  
break }  
incr l -1  
incr i 1  
}  
if {$flag ==0} { puts “The given string $str is palindrome” } else {  
puts “The given string $str is not palindrome” }  
}  
palindrome “malayalam”  
palindrome “welcome”  
o/p:  
The given string malayalam is palindrome  
The given string welcome is not palindrome

1.TCL Program – count letters in the given string using tcl

set str “LIHAKHDBLICIHJAADFDCSDBBBDFDB”  
set l [string length $str]  
puts $l  
set cnt\_A 0  
set cnt\_B 0  
set cnt\_C 0  
set i 0  
while {$i<=$l} {  
if {“A”==[string index $str $i]} {  
incr cnt\_A   } elseif {“B”==[string index $str $i]} {  
incr cnt\_B   } elseif {“C”==[string index $str $i]} {  
incr cnt\_C   }  
incr i  
}

puts “The count of A is $cnt\_A \n  
The count of B is $cnt\_B \n  
The count of C is $cnt\_C”

o/p:  
29  
The count of A is 3

The count of B is 5

The count of C is 2

2.TCL Program- Factorial value with and without using recursion in tcl

proc fact { n } {  
set f 1  
while {$n>=2} {  
set f [expr {$f\*$n}]  
incr n -1  
}  
return $f  
}

proc recfact n {  
if {$n<=1} {  
return 1  }  
expr $n \* [recfact [expr {$n-1}]]  
}

puts “The factorial value without recursion [fact 4]”  
puts “The factorial value with recursion [recfact 4]”

3.TCL Program – Find maximum number in the given 3 numbers using tcl

set a 10  
set b 20  
set c 15  
if {$a>$b && $a>$c} {  
puts “a is bigger and value is $a” } elseif {$b>$a && $b>$c} {  
puts “b is bigger and value is $b” } else {  
puts “c is bigger and value is $c” }

o/p:  
b is bigger and value is 20

4.TCL Program – reverse string using tcl

proc strrev {str} {  
set l [string length $str]  
set l [expr $l-1]  
set rev {}  
for {set i $l} {$i>=0} {incr i -1} {  
append rev [string index $str $i]   }  
puts “$rev”  
}

strrev “welcome”

o/p:  
emoclew

5.TCL Program – Check given number is odd or even in tcl

proc oddeven {n} {  
if {$n%2==0} {  
puts “the given $n is even” } else {  
puts “the given number $n is odd” }

}

oddeven 24

oddeven 67

o/p:  
the given 24 is even  
the given number 67 is odd