

**NAME**

tclsh – Simple shell containing Tcl interpreter

**SYNOPSIS**

**tclsh** *?-encoding name? ?fileName arg arg ...?*

**DESCRIPTION**

**Tclsh** is a shell-like application that reads Tcl commands from its standard input or from a file and evaluates them. If invoked with no arguments then it runs interactively, reading Tcl commands from standard input and printing command results and error messages to standard output. It runs until the **exit** command is invoked or until it reaches end-of-file on its standard input. If there exists a file **.tclshrc** (or **tclshrc.tcl** on the Windows platforms) in the home directory of the user, interactive **tclsh** evaluates the file as a Tcl script just before reading the first command from standard input.

**SCRIPT FILES**

If **tclsh** is invoked with arguments then the first few arguments specify the name of a script file, and, optionally, the encoding of the text data stored in that script file. Any additional arguments are made available to the script as variables (see below). Instead of reading commands from standard input **tclsh** will read Tcl commands from the named file; **tclsh** will exit when it reaches the end of the file. The end of the file may be marked either by the physical end of the medium, or by the character, “\032” (“\u001a”, control-Z). If this character is present in the file, the **tclsh** application will read text up to but not including the character. An application that requires this character in the file may safely encode it as “\032”, “\x1A”, or “\u001a”; or may generate it by use of commands such as **format** or **binary**. There is no automatic evaluation of **.tclshrc** when the name of a script file is presented on the **tclsh** command line, but the script file can always **source** it if desired.

If you create a Tcl script in a file whose first line is

```
#!/usr/local/bin/tclsh
```

then you can invoke the script file directly from your shell if you mark the file as executable. This assumes that **tclsh** has been installed in the default location in **/usr/local/bin**; if it is installed somewhere else then you will have to modify the above line to match. Many UNIX systems do not allow the **#!** line to exceed about 30 characters in length, so be sure that the **tclsh** executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:

```
#!/bin/sh
# the next line restarts using tclsh \
exec tclsh "$0" ${1+"$@"}
```

This approach has three advantages over the approach in the previous paragraph. First, the location of the **tclsh** binary does not have to be hard-wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30-character file name limit in the previous approach. Third, this approach will work even if **tclsh** is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the **tclsh** script selects one of several binaries to run). The three lines cause both **sh** and **tclsh** to process the script, but the **exec** is only executed by **sh**. **sh** processes the script first; it treats the second line as a comment and executes the third line. The **exec** statement causes the shell to stop processing and instead to start up **tclsh** to reprocess the entire script. When **tclsh** starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

You should note that it is also common practice to install **tclsh** with its version number as part of the name. This has the advantage of allowing multiple versions of Tcl to exist on the same system at once, but also the disadvantage of making it harder to write scripts that start up uniformly across different versions of Tcl.

## VARIABLES

**Tclsh** sets the following global Tcl variables in addition to those created by the Tcl library itself (such as **env**, which maps environment variables such as **PATH** into Tcl):

<b>argc</b>	Contains a count of the number of <i>arg</i> arguments (0 if none), not including the name of the script file.
<b>argv</b>	Contains a Tcl list whose elements are the <i>arg</i> arguments, in order, or an empty string if there are no <i>arg</i> arguments.
<b>argv0</b>	Contains <i>fileName</i> if it was specified. Otherwise, contains the name by which <b>tclsh</b> was invoked.
<b>tcl_interactive</b>	Contains 1 if <b>tclsh</b> is running interactively (no <i>fileName</i> was specified and standard input is a terminal-like device), 0 otherwise.

## PROMPTS

When **tclsh** is invoked interactively it normally prompts for each command with “% ”. You can change the prompt by setting the global variables **tcl\_prompt1** and **tcl\_prompt2**. If variable **tcl\_prompt1** exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt **tclsh** will evaluate the script in **tcl\_prompt1**. The variable **tcl\_prompt2** is used in a similar way when a newline is typed but the current command is not yet complete; if **tcl\_prompt2** is not set then no prompt is output for incomplete commands.

## STANDARD CHANNELS

See **Tcl\_StandardChannels** for more explanations.

## SEE ALSO

`auto_path(3tcl)`, `encoding(3tcl)`, `env(3tcl)`, `fconfigure(3tcl)`

## KEYWORDS

application, argument, interpreter, prompt, script file, shell