

# Re: man page syntax for +script

---

*Source:* <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2006-05/msg00276.html>

---

- *From:* Dan Smart <[news900@xxxxxxxxxxxxx](mailto:news900@xxxxxxxxxxxxx)>
  - *Date:* Mon, 8 May 2006 01:01:07 -0400
- 

On 2006-05-07 20:28:46 -0400, "Gerald W. Lester" <[Gerald.Lester@xxxxxxx](mailto:Gerald.Lester@xxxxxxx)> said:

Dan Smart wrote:

On 2006-05-07 13:56:53 -0400, "Gerald W. Lester" <[Gerald.Lester@xxxxxxx](mailto:Gerald.Lester@xxxxxxx)> said:

John Seal wrote:

My spring fever continues...  
(That's my story, and I'm sticking to it!)

The man pages for [bind] and [comm hook] both describe how you can either replace the current script or append to it, but they give slightly different syntaxes. [bind] says

```
bind tag ?sequence? ?+??script?
```

and [comm] says

```
::comm::comm hook event ?+? ?script?
```

with a space between the + and the script. In fact, both of them seem to work with or without the space. But the syntax for [comm hook] makes it appear that the + and the script are separate arguments, and that caused me some confusion.

...

I do think the [comm] man page syntax is wrong, or at least seriously misleading. The [bind] man page, at first glance, doesn't seem to permit a space, but since the script can contain whitespace I think it's implied.

## Re: man page syntax for +script

As I said, both [comm] and [bind] are perfectly happy with or without a space between the + and the rest of the script.

The man page may be wrong — but it is not an issue of syntax, but rather semantics. Tcl, before 8.5, only has eleven syntax rules and twelve rules starting with 8.5 — and "+" ain't one of them.

The plus is a semantic definition that these two unrelated commands put on their last argument, expr has a very different semantic definition of "+".

It is so a question of syntax, and you would be wise to have read the post before lecturing him.

Dan I strongly suggest that you read his post again with an eye toward what he is saying. Based on the next sentence in your reply. you have a distorted view of what is syntax and what is semantics in Tcl. For at least your benefit, I'm posting a detail explanation here.

I did read his post, I also understand the difference between "syntax" (how do I have to say something) and "semantics" (what does it mean). You on the other hand appear more than a little confused.

His issues isn't with the semantics of '+' as applied to those commands: he knows "what putting a + before the script does" (the semantics), and both commands define the effect consistently. His issue is with the syntax, "where should he put the + to get that effect", and the two commands define that inconsistently.

Where he needs to put the "+" is **\*\*NOT\*\*** a matter of Tcl syntax — it is **\*\*STRICTLY\*\*** a matter of the semantics that the command applies to its arguments. Thus the two commands are allowed to define the semantics of the "+" inconsistently, just as set and list define the semantics of their first argument "inconsistently".

I thought I had pointed out that we weren't talking about the syntax of TCL the language, but the syntax of two commands available (under certain circumstances) within a tcl interpreter. How commands parse the individual words that the TCL interpreter passes them as arguments has nothing to do with TCL syntax, but it is syntax. I'll reiterate for your benefit, and for any that you've confused, the \*semantics\* of '+' for both commands are "append the script that follows", the syntax question is "where do I place the '+' to get that effect".

Syntax: What are the rules of construction.

Re: man page syntax for +script

## Re: man page syntax for +script

Semantics: What does the phrase thus constructed mean.

Note: the 11/12 rules of TCL syntax don't cover [expr] expressions either, or regular expressions for that matter.

Duh — that is because of how those commands, or any other command, apply a meaning (or no meaning) to "+" is not a matter of syntax but rather purely the semantics that each individual command applies to its argument.

Both [expr] and [regexp] define the semantics (meanings) of individual 'symbols', and they also define the semantics (meanings) of collections of 'symbols'. They also both specify a syntax which describes exactly what constitutes a symbol, and how those symbols may be combined. Neither syntax is identical to TCL's, although [expr] is very similar.

In short, Tcl's syntax, all 11/12 rules of it, apply language wide — and that is *\*ALL\** that applies language wide. Anything else that appears to is either coincident or imitation. In fact from one installation/interpreter (e.g. inside a safe interpreter) to the next you can not really count on all of the "core" commands being there (e.g. exec may be missing) or have a different semantics (e.g. proc may have been redefined to provide some type of tracing or debugging).

This is of course correct, but largely irrelevant. Having two commands that expose different syntax to provide the same semantics is ugly, which is in part what the OP was complaining about.

Way too often people apply baggage of thinking way more is syntax in Tcl that is really only semantics to Tcl. I think this is because of the large amount of syntax other languages have — this is a personal observation from years of teaching Tcl and answering questions in the news group.

I repeat: You *\*keep\** using that word, I don't think it means what you think it means.

Syntax: How do I say something.

Semantics: What does it mean.

Dan "You killed my father, prepare to die" Smart

.