Read? Forward? Skip?

The reader wants to know right away whether to read, route, or skip a report.
— James W. Souther

Primary Audience

Tcl scripters who use Emacs

Secondary Audience

Slow typists and those who take pity on them

PACKAGE	SLIDES	EuroTcl
3.4	2.7	2019
		June 2019

Tcl Digraphs

Emacs `T;c;l` Minor Mode

Use-Case

I am a slow typist

I am generally slower with modifier keys:
escape, alt, control, etc.

I want help from my editor

Editing MACroS

Tailored for Tcl syntax

- " *string* "
- [*command*]
- { *block* }
- etc.

… making it easier to type code

Keymap tied to ;

The semicolon itself

- is on the U.S. keyboard home-row
- is rarely used in Tcl scripts
  - a (deprecated) command separator
  - or natural punctuation in strings and comments

Reference card ; ?

;CR	→	;NL	semicolon newline	;ESC	→	;	semicolon
;SP	→	;SP	semicolon space	;,	→	} {	split a block
;/	→	…\NL	add splice	;;	→	\;	escaped semicolon
;[	→	[]	subcommand	;]	→	… []	append subcommand
;b	→	… {}	append block	;c	→	[]	subcommand
;n	→	!()	not-parens	;o	→	{ § }	open-block
;p	→	()	paren	;q	→	""	quote
;t	→	… { § }	add then-block				
;u	→	{![]}	until-condition	;v	→	{}	curly-quote
;w	→	{[]}	while-condition	;x	→	{*}	expander
;{	→	{}	curly-quote	;}	→	… {}	append block

Emacs code


```
;;; tcl-digraphs.el --- Emacs Minor Mode for Tcl script files

;; Copyright (C) 2018 Michael Kaelbling, SIEMENS AG.

;; Author: Michael Kaelbling <michael.kaelbling@siemens.com>
;; Keywords: Tcl, Tcl/Tk, digraph, keyboard shortcut
;; Version: 3.4

;; This file is not part of GNU Emacs.

;; This program is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation, either version 3 of the License, or
;; (at your option) any later version.

;; This program is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;; GNU General Public License for more details.

;; You should have received a copy of the GNU General Public License
```

```
;; along with GNU Emacs.  If not, see <http://www.gnu.org/licenses/>.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Code:

(defun tcl--insert (prefix suffix n)
  "Insert text before and after the active-region or point.

PREFIX is the text to precede the active-region or point.
SUFFIX is the text to follow the active-region or point.
N is the number of characters to back-up after inserting SUFFIX."
  (let ((beg (if (region-active-p) (region-beginning) (point)))
        (end (+ (if (region-active-p) (region-end) (point))
                (length prefix))))
    (when prefix (goto-char beg) (insert prefix))
    (when suffix (goto-char end) (insert suffix))
    (when n (backward-char n))))

(defun lambda-key (keymap key def)
  "Workaround `define-key' to provide documentation."
  (set 'sym (make-symbol (documentation def)))
  (fset sym def)
  (define-key keymap key sym))

(defconst tcl-digraph-prefixed-map
  (let ((map (make-sparse-keymap)))
    (lambda-key map "\r" '(lambda () "insert ; newline"
                            (interactive "*") (insert ";")
                            (newline)))
    (lambda-key map "\e" '(lambda () "insert ;"
                            (interactive "*") (insert ";")))
    (lambda-key map " " '(lambda () "insert ; "
                           (interactive "*") (insert "; ")))
    (lambda-key map "," '(lambda () "insert } {"
                           (interactive "*") (insert "} {")))
    (lambda-key map "/" '(lambda () "append splice"
                           (interactive "*")
                           (move-end-of-line 1)
                           (insert "\\") (newline)))
    (lambda-key map ";" '(lambda () "insert \\;"
                           (interactive "*") (insert "\\;")))
    (lambda-key map "[" '(lambda () "insert command-quote []"
                           (interactive "*")
                           (tcl--insert "[" "]" 1)))
    (lambda-key map "]" '(lambda () "append command-quote []"
                           (interactive "*")
                           (move-end-of-line 1) (just-one-space)
                           (insert "[]") (backward-char 1)))
    (lambda-key map "?" '(lambda () "describe minor mode"
                           (interactive)
                           (describe-minor-mode 'tcl-digraph-minor-mode)))
    (lambda-key map "a" '(lambda (P) "insert $args"
                           (interactive "*P")
                           (insert (if P "{*}$args" "$args"))))
```

```
(lambda-key map "b" '(lambda () "append a block-quote {}"
                       (interactive "*")
                       (move-end-of-line 1) (just-one-space)
                       (insert "{}") (backward-char 1)))
(lambda-key map "c" '(lambda () "insert subcommand-quote []"
                       (interactive "*")
                       (tcl--insert "[" "]" 1)))
(lambda-key map "d" '(lambda () "insert $"
                       (interactive "*")
                       (insert "$")))
(lambda-key map "e" '(lambda () "append 'else'-block { § }"
                       (interactive "*")
                       (move-end-of-line 1) (just-one-space)
                       (insert "else {}") (backward-char 1)
                       (newline) (tcl-indent-command)
                       (move-beginning-of-line 1)
                       (open-line 1) (tcl-indent-command)))
(lambda-key map "f" '(lambda () "append 'elseif'-block {}"
                       (interactive "*")
                       (move-end-of-line 1) (just-one-space)
                       (insert "elseif {}") (backward-char 1)))
(lambda-key map "h" '(lambda (P c) "insert HTML entity"
                       (interactive "*P\nc")
                       (cond
                        ((and (>= c 0) (< c 32))
                         (insert (format "[&x24%02X]" c)))
                        ((= c 32) (insert (if P "[&x2420]"
                                            "[ ]"))) ;␠SP
                        ((= c 34) (insert "[&quot]"))
                        ((= c 38) (insert "[&amp]"))
                        ((= c 39) (insert "[&apos]"))
                        ((= c ?-) (insert "[&shy]"))   ;soft-
                        ((= c ?<) (insert "[&lt]"))
                        ((= c ?>) (insert "[&gt]"))
                        ((= c ?B) (insert "[&x2422]")) ;␢b/b
                        ((= c ?b) (insert "[&x2423]")) ;␣
                        ((= c ?h) (insert "[&hellip]"))
                        ((= c ?n) (insert "[&x2424]")) ;␤NL
                        ((= c ?p) (insert 182))        ;¶
                        ((= c ?s) (insert 167))        ;§
                        ((= c ?|) (insert 166))        ;¦
                        ((= c 127) (insert "[&x2421]"));␡DEL
                        (t (insert (format "[&%d]" c))))))
(lambda-key map "n" '(lambda () "insert 'not-parens' !()"
                       (interactive "*")
                       (tcl--insert "!(" ")" 1)))
(lambda-key map "o" '(lambda () "open { § }"
                       (interactive "*")
                       (insert "{}") (backward-char 1)
                       (newline) (tcl-indent-command)
                       (move-beginning-of-line 1)
                       (open-line 1) (tcl-indent-command)))
(lambda-key map "p" '(lambda () "insert parentheses ()"
```

```
                          (interactive "*")
                          (tcl--insert "(" ")" 1)))
    (lambda-key map "q" '(lambda () "insert quotes \"\""
                          (interactive "*")
                          (tcl--insert "\"" "\"" 1)))
    (lambda-key map "s" '(lambda (P) "insert [set ]"
                          (interactive "*P")
                          (if P (insert "$")
                            (tcl--insert "[set " "]" 1))))
    (lambda-key map "t" '(lambda () "append 'then'-block { § }"
                          (interactive "*")
                          (move-end-of-line 1) (just-one-space)
                          (insert "{}") (backward-char 1)
                          (newline) (tcl-indent-command)
                          (move-beginning-of-line 1)
                          (open-line 1) (tcl-indent-command)))
    (lambda-key map "u" '(lambda (P) "insert 'until'-block ![] | {![]}"
                          (interactive "*P")
                          (if (and (looking-at-p "}") (not P))
                              (tcl--insert "![" "]" 1)
                            (tcl--insert "{![" "]}" 2))))
    (lambda-key map "v" '(lambda () "insert curly-quote {}"
                          (interactive "*")
                          (tcl--insert "{" "}" 1)))
    (lambda-key map "w" '(lambda (P) "insert 'while'-block [] | {[]}"
                          (interactive "*P")
                          (if (and (looking-at-p "}") (not P))
                              (tcl--insert "[" "]" 1)
                            (tcl--insert "{[" "]}" 2))))
    (lambda-key map "x" '(lambda (P) "insert {*} | {*}$args"
                          (interactive "*P")
                          (insert (if P "{*}$args" "{*}"))))
    (lambda-key map "{" '(lambda () "insert block-quote []"
                          (interactive "*")
                          (tcl--insert "{" "}" 1)))
    (lambda-key map "}" '(lambda () "append block-quote []"
                          (interactive "*")
                          (move-end-of-line 1) (just-one-space)
                          (insert "{}") (backward-char 1)))
    (lambda-key map "\d" '(lambda () "cancel digraph"
                           (interactive "*")
                           (message "digraph canceled")))
    map)
  "Keymap behind the `tcl-digraph-minor-mode' prefix key.")

(defconst tcl-digraph-minor-mode-map
  (let ((map (make-sparse-keymap)))
    (define-key map [59] tcl-digraph-prefixed-map)
    map)
  "Keymap for `tcl-digraph-minor-mode'.")

(put 'tcl-digraph-minor-mode :included t)

(define-minor-mode tcl-digraph-minor-mode
```

```
  "The `tcl-digraph-minor-mode'.
digraphs:
    ;RET    ; newline       - semicolon newline
    ;ESC    ;               - semicolon
    ;SPC    ;␣             - semicolon and space
    ;,      } {             - as in for {...} {...} {...}
    ;/      \\ newline       - splice [at end of line]
    ;;      \\ ;             - backslash semicolon
    ;?                      - describe Tcl digraph minor mode
    ;[      []              - command-quote region
    ;]      ... []          - command-quote at end of line
    ;a      $args           - args reference
    ;b      ... {}          - brace-quote at end of line
    ;c      []              - command region
    ;d      $               - insert dollar-sign | insert [set ]
    ;e      ... else { § }  - else-block at end of line
    ;f      ... elseif {}   - elseif-block at end of line
    ;h      &...            - HTML character entity
    ;n      !()             - not-parenthesized region
    ;o      { § }           - open block-quote
    ;p      ()              - parenthesized region
    ;q      \"\"              - quoted region
    ;s      [set ]          - insert [set ...]
    ;t      ... { § }       - then-block at end of line
    ;u      {![]}           - until-condition region
    ;v      {}              - verbatim-quote region
    ;w      {[]}            - while-condition region
```

```
    ;x      {*}             - expand | expand $args
    ;{      {}              - verbatim-quote region
    ;}      ... {}          - verbatim-quote at end of line
    ;DEL                    - cancel digraph
"
  :lighter " T;c;l" :global nil :init-value nil
  :keymap tcl-digraph-minor-mode-map :version "3.4")

(provide 'tcl-digraphs)

;;; tcl-digraphs.el ends here
```

Take-Aways

Success!

- digraph translations are convenient
- Emacs `tcl-digraphs.el` minor mode with bonus: `lambda-key`

Adapt!

- add your own editing macros
- customize for your keyboard