ViperCard: Viper Reference Pal

(Version 3.0 for Emacs 21)

Loading Viper

Just type `M-x viper-mode` followed by `RET`

OR put

(setq viper-mode t) (require 'viper)

in .emacs

Viper States

Viper has four states: *emacs state*, *vi state*, *insert state*, *replace state*. Mode line tells you which state you are in. In emacs state you can do all the normal GNU Emacs editing. This card explains only vi state and insert state (replace state is similar to insert state). **GNU Emacs Reference Card** explains emacs state. You can switch states as follows.

from emacs state to vi state	`C-z`
from vi state to emacs state	`C-z`
from vi state to emacs state for 1 command	`\`
from vi state to insert state	`i, I, a, A, o, O`
from vi state to replace state	`c, C, R`
from insert or replace state to vi state	`ESC`
from insert state to vi state for 1 command	`C-z`

Insert Mode

You can do editing in insert state.

go back to vi state	`ESC`
delete previous character	`C-h, DEL`
delete previous word	`C-w`
delete line word	`C-u`
indent shiftwidth forward	`C-t`
indent shiftwidth backward	`C-d`
delete line word	`C-u`
quote following character	`C-v`
emulate Meta key in emacs state	`C-\`
escape to Vi state for one command	`C-z`

The rest of this card explains commands in vi state.

Getting Information on Viper

Execute info command by typing `M-x info` and select menu item `viper`. Also:

describe function attached to the key *x*	`\ C-h k` *x*

Leaving Emacs

suspend Emacs	`:st` or `:su`
exit Emacs permanently	`C-xC-c`
exit current file	`:wq` or `:q`

Error Recovery

abort command	`C-c (user level = 1)`
abort command	`C-g (user level > 1)`
redraw messed up screen	`C-l`
recover after system crash	`:rec file`
restore a buffer	`:e!` or `M-x revert-buffer`

Counts

Most commands in vi state accept a *count* which can be supplied as a prefix to the commands. In most cases, if a count is given, the command is executed that many times. E.g., `5 d d` deletes 5 lines.

Registers

There are 26 registers (`a` to `z`) that can store texts and marks. You can append a text at the end of a register (say `x`) by specifying the register name in capital letter (say `X`). There are also 9 read only registers (`1` to `9`) that store up to 9 previous changes. We will use *x* to denote a register.

Entering Insert Mode

insert at point	`i`
append after cursor	`a`
insert before first non-white	`I`
append at end of line	`A`
open line below	`o`
open line above	`O`

Buffers and Windows

move cursor to **next** window	`C-x o`
delete current window	`C-x 0`
delete other windows	`C-x 1`
split current window into two windows	`C-x 2`
switch to a buffer in the current window	`C-x` *buffer*
switch to a buffer in another window	`:n, :b,` or `C-x 4` *buf*
kill a buffer	`:q!` or `C-x k`
list existing **buffers**	`:args` or `C-x b`

Files

visit file in the current window	`v` *file* or `:e` *file*
visit file in another window	`V` *file*
visit file in another frame	`C-v` *file*
save buffer to the associated file	`:w` or `C-xC-s`
write buffer to a specified file	`:w` *file* or `C-xC-w`
insert a specified file at point	`:r` *file* or `C-xi`
get information on the current **file**	`C-c g` or `:f`
run the **directory** editor	`:e RET` or `C-xd`

Viewing the Buffer

scroll to next screen	`C-f`
scroll to previous screen	`C-b`
scroll **down** half screen	`C-d`
scroll **up** half screen	`C-u`
scroll down one line	`C-e`
scroll up one line	`C-y`
put current line on the **home** line	`z H` or `z RET`
put current line on the **middle** line	`z M` or `z .`
put current line on the **last** line	`z L` or `z -`

Marking and Returning

mark point in register *x*	`m` *x*
set mark at buffer beginning	`m <`
set mark at buffer end	`m >`
set mark at point	`m .`
jump to mark	`m ,`
exchange point and mark	`` ` ` ``
... and skip to first non-white on line	`' '`
go to mark *x*	`` ` `` *x*
... and skip to first non-white on line	`'` *x*
view contents of marker *x*	`[` *x*
view contents of register *x*	`]` *x*

Macros

Emacs style macros:

start remembering keyboard macro	`C-x (`
finish remembering keyboard macro	`C-x )`
call last keyboard macro	`*`
start remembering keyboard macro	`@ #`
finish macro and put into register *x*	`@` *x*
execute macro stored in register *x*	`@` *x*
repeat last @*x* command	`@ @`
Pull last macro into register *x*	`@ !` *x*

Vi-style macros (keys to be hit in quick succession):

define Vi-style macro for Vi state	`:map`
define Vi-style macro for Insert state	`:map!`
toggle case-sensitive search	`//`
toggle regular expression search	`///`
toggle '%' to ignore parentheses inside comments	`%%%`

Motion Commands

go backward one character	`h` or `C-h`
go forward one character	`l`
next line keeping the column	`j` or `LF` or `C-n`
previous line keeping the column	`k`
next line at first non-white	`+` or `RET` or `C-p`
previous line at first non-white	`-`
beginning of line	`0`
first non-white on line	`^`
end of line	`$`
go to *n*-th column on line	*n* `|`
go to *n*-th line	*n* `G`
go to last line	`G`
find matching parenthesis for `()`, `{}` and `[]`	`%`
go to **home** window line	`H`
go to **middle** window line	`M`
go to **last** window line	`L`

Words, Sentences, Paragraphs, Headings

forward **word**	`w` or `W`
backward word	`b` or `B`
end of word	`e` or `E`

In the case of capital letter commands, a word is delimited by a non-white character.

forward sentence	`)`
backward sentence	`(`
forward paragraph	`}`
backward paragraph	`{`
forward heading	`]]`
backward heading	`[[`
end of heading	`[]`

Find Characters on the Line

find *c* forward on line	`f` *c*
find *c* backward on line	`F` *c*
up **to** *c* forward on line	`t` *c*
up **to** *c* backward on line	`T` *c*
repeat previous `f`, `F`, `t` or `T`	`;`
... in the opposite direction	`,`

Searching and Replacing

search forward for *pat*	`/` *pat*
search backward with previous *pat*	`? RET`
search forward with previous *pat*	`/ RET`
search backward for *pat*	`?` *pat*
repeat previous search	`n`
... in the opposite direction	`N`
query replace	`Q`
replace a character by another character *c*	`r` *c*
overwrite *n* lines	*n* `R`
buffer search (if enabled)	`g` *move command*

Modifying Commands

Most commands that operate on text regions accept the motion commands, to describe regions. They also accept the Emacs region specifications **r** and **R**. **r** describes the region between *point* and *mark*, and **R** describes whole lines in that region. Motion commands are classified into *point commands* and *line commands*. In the case of line commands, whole lines will be affected by the command.

The point commands are as follows:

`h l 0 ^ $ w W b B e E ( ) / ? ‘ f F t T % ; ,`

The line commands are as follows:

`j k + - H M L { } G ’`

These region specifiers will be referred to as *m* below.

Delete/Yank/Change Commands

	delete	**yank**	**change**
region determined by *m*	`d` *m*	`y` *m*	`c` *m*
... into register *x*	`"` *x* `d` *m*	`"` *x* `y` *m*	`"` *x* `c` *m*
a line	`d d`	`Y` or `y y`	`c c`
current **region**	`d r`	`y r`	`c r`
expanded **region**	`d R`	`y R`	`c R`
to end of line	`D`	`y $`	`c $`
a character after point	`x`	`y l`	`c l`
a character before point	`DEL`	`y h`	`c h`

Overwrite *n* lines	*n* `R`

Put Back Commands

Deleted/yanked/changed text can be put back by the following commands.

Put back at point/above line	`P`
... from register *x*	`"` *x* `P`
put back after point/below line	`p`
... from register *x*	`"` *x* `p`

Repeating and Undoing Modifications

undo last change	`u` or `:und`
repeat last change	`.` (dot)

Undo is undoable by `u` and repeatable by `..` For example, `u...` will undo 4 previous changes. A `.` after `5dd` is equivalent to `5dd`, while `3.` after `5dd` is equivalent to `3dd`.

Miscellaneous Commands

	shift left	**shift right**	**filter shell command**	**indent**
region	`<` *m*	`>` *m*	`!` *m shell-com*	`=` *m*
line	`< <`	`> >`	`! !` *shell-com*	`= =`

join lines	`J`
toggle case (takes count)	`~`
view register *x*	`]` *x*
view marker *x*	`]` *x*
lowercase region	`# c` *m*
uppercase region	`# C` *m*
execute last keyboard macro on each line in the region	`# g` *m*
insert specified string for each line in the region	`# q` *m*
check spelling of the words in the region	`# s` *m*
repeat previous ex substitution	`&`
change to previous file	`C-^`
Viper Meta key	`_`

Customization

By default, search is case sensitive. You can change this by including the following line in your `~/.vip` file.

```
(setq viper-case-fold-search t)
```

The following is a subset of the variety of options available for customizing Viper. See the Viper manual for details on these and other options.

variable	**default**	**value**
`viper-search-wrap-around`	`t`	
`viper-case-fold-search`	`nil`	
`viper-re-search`	`t`	
`viper-re-replace`	`t`	
`viper-re-query-replace`	`t`	
`viper-auto-indent`	`nil`	
`viper-shift-width`	`8`	
`viper-tags-file-name`	`"TAGS"`	
`viper-no-multiple-ESC`	`t`	
`viper-ex-style-motion`	`t`	
`viper-always`	`t`	
`viper-custom-file-name`	`"~/.vip"`	
`ex-find-file-shell`	`"csh"`	
`ex-cycle-other-window`	`t`	
`ex-cycle-through-non-buffers`	`t`	
`blink-matching-paren`	`t`	
`buffer-read-only`	*buffer dependent*	

To bind keys in Vi command state, put lines like these in your `~/.vip` file:

```
(define-key viper-vi-global-user-map "\C-v" ’scroll-down)
(define-key viper-vi-global-user-map "\C-cm" ’smail)
```

Ex Commands in Viper

In vi state, an Ex command is entered by typing:

`:` *ex-command* `RET`

Ex Addresses

current line	`.`	next line with *pat*	`/` *pat* `/`
line *n*	*n*	previous line with *pat*	`?` *pat* `?`
last line	`$`	*n* line before *a*	*a* `-` *n*
next line	`+`	*a* through *b*	*a* `,` *b*
previous line	`-`	line marked with *x*	`'` *x*
entire buffer	`%`	previous context	`' '`

Addresses can be specified in front of a command. For example,

`:.,.+10m$`

moves 11 lines below current line to the end of buffer.

Ex Commands

Avoid Ex text manipulation commands except substitute. There are better VI equivalents for all of them. Also note that all Ex commands expand % to current file name. To include a % in the command, escape it with a \. Similarly, # is replaced by previous file. For Viper, this is the first file in the *:args* listing for that buffer. This defaults to the previous file in the VI sense if you have one window. Ex commands can be made to have history. See the manual for details.

Ex Text Commands

mark lines matching *pat* and execute *cmds* on these lines	`:g /`*pat*`/` *cmds*
mark lines *not* matching *pat* and execute *cmds* on these lines	`:v /`*pat*`/` *cmds*
move specified lines after *addr*	`:m` *addr*
copy specified lines after *addr*	`:co` (or `:t`) *addr*
delete specified lines [into register *x*]	`:d` [*x*]
yank specified lines [into register *x*]	`:y` [*x*]
put back text [from register *x*]	`:pu` [*x*]
substitute *repl* for first string on line matching *pat*	`:s /`*pat*`/`*repl*`/`
repeat last substitution	`:&`
repeat previous substitute with previous search pattern as *pat*	`:~`

Ex File and Shell Commands

edit file	`:e` *file*
reedit messed up current file	`:e!`
edit previous file	`:e#`
read in a file	`:r` *file*
read in the output of a shell command	`:r` *!command*
write out specified lines into *file*	`:w` *file*
save all modified buffers, ask confirmation	`:W` *file*
save all modified buffers, no confirmation	`:WW` *file*
write out specified lines at the end of *file*	`:w>>` *file*
write to the input of a shell command	`:w` *!command*
write out and then quit	`:wq` *file*
run a sub**shell** in a window	`:sh`
execute shell command *command*	`:!` *command*
execute previous shell command with *args* appended	`:!!` *args*

Ex Miscellaneous Commands

define a macro *x* that expands to *cmd*	`:map` *x cmd*
remove macro expansion associated with *x*	`:unma` *x*
define a macro *x* that expands to *cmd* in insert state	`:map!` *x cmd*
remove macro expansion associated with *x* in insert state	`:unma!` *x*
print line number	`:.=`
print last line number	`:=`
print **version** number of Viper	`:ve`
shift specified lines to the right	`:>`
shift specified lines to the left	`:<`
join lines	`:j`
mark specified line to register *x*	`:k` *x*
set a variable's value	`:se`
find first definition of **tag** *tag*	`:ta` *tag*
Current directory	`:pwd`

For more Emacs documentation, and the TEX source for this card, see the Emacs distribution, or `https://www.gnu.org/software/emacs`