The Back Side — Vim Extensions

Visual Mode

Probably the most useful extension in Vim is visual mode. It allows you to highlight a region of text for the application of an editing or “colon” command.

highlight by character	`v`
highlight by line	`V`
highlight by column	`C-V`
re–select previous visual area	`gv`

Windows

Even on text consoles, Vim allows the editing environment to be divided into separate windows which may contain the same or different files. Commands that open a new window take an optional *file* argument.

open new horizontal window	`:new`
split window horizontally	`:split`
split vertically	`:vsplit`
close current window	`:close`
close all but current window	`:only`
cycle to next window	`C-W w`
cycle to previous window	`C-W p`
make windows equal height, width	`C-W = , C-W |`
decrease, increase height one	`C-W - , C-W +`
decrease, increase width one	`C-W < , C-W >`

Objects

In addition to the usual motions that are supported by Vi, Vim supports several objects: word, sentence, paragraph, and block. These are used with editing commands (delete, change, and yank). The advantage over motions is that you do not have to be at one end of an object to use it. For example `das` will delete a sentence from anywhere within it.

a word, a sentence, a paragraph	`aw, as, ap`
same without white space (inner)	`iw, is, ip`
blocks delimited by [], (), <>, and {}	`a[, a(, a<, a{`
same without surrounding brackets	`i[, i(, i<, i{`

Extra motions

jump to beginning, end of { } block	`[{ , ]}`
same for parentheses	`[( , ])`
same for / (C–style comments)	`[/ , ]/`
previous end of word, space–delimited	`ge, gE`

Extra markers

Marks set with capital letters are “global”; jumping to a global mark may take you to a different file. Several marks are remembered between editing sessions:

position when file was last open	`’"`
position where file was last edited	`’.`
file and position of last Vim session	`’0`

Other searching

incremental search	`:set (no)incsearch`
highlight search	`:set (no)hlsearch`
ignore case in search	`:set (no)ignorecase`
find word under cursor fwd, back	`* , #`

Interface and colors

syntax coloring	`:syntax on`
switch to graphical	`:gui`

show line numbers	`:set number`
show commands in progress	`:set showcmd`
show matching bracket	`:set showmatch`

autoindent	`:set ai`
smartindent	`:set si`
file format (dos, unix, mac)	`:set ff=...`
good for editing tables	`:set virtualedit=all`
edit option set	`:options`

Insert mode editing

There are some commands available while in insert mode:

copy from line above, below	`C-Y , C-E`
complete from previous match	`C-P`
insert from (paste) register	`C-R`
insert digraph	`C-K` *di*
execute single command	`C-O` *command*

Folding

To make working with a file more convenient, multiple lines may be “folded” together into a single highlighted line. Use this to get long sections out of the way while editing.

create, delete a fold	`zf, zd`
open, (re–)close a fold	`zo, zc`
remove outermost folds, more folds	`zr, zm`
remove all, re–fold all folds	`zR, zM`
disable, (re–)enable, toggle folds	`zn, zN, zi`
save, load view (including folds)	`:mkview, :lo`

Compiling and source code

There are several commands for facilitating software development:

correct indentation of line	`==`
increase the indent level	`>>`
decrease the indent level	`<<`

run make and move cursor to first error	`:make`
move to next, previous error	`:cnext, :cprev`
move to first, last error	`:cfirst, :clast`
view error or list of errors	`:cc, :clist`
list lines with identifier under cursor	`[I`

Using ctags

You must first run ctags or etags on your file to use these features.

jump to function with tagname	`:tag` *tagname*
jump to tag under cursor	`C-]`
previous tag	`C-t`

Other

reformat margins	`gq`
make case upper, lower, toggle	`gU, gu, g~`
add subtract from number under cursor	`C-A , C-X`
edit file under cursor	`gf`
view man page under cursor	`K`
print highlighted copy	`:hardcopy`
list digraphs	`:digraphs`
editable command history	`q:`