**motion**

{motion} A command that moves the cursor. These are explained in

|motion.txt|. Examples:

w to start of next word

b to begin of current word

4j four lines down

/The<CR> to next occurrence of "The"

This is used after an |operator| command to move over the text

that is to be operated upon.

- If the motion includes a count and the operator also has a

count, the two counts are multiplied. For example: "2d3w"

deletes six words.

- The motion can be backwards, e.g. "db" to delete to the

start of the word.

- The motion can also be a mouse click. The mouse is not

supported in every terminal though.

- The ":omap" command can be used to map characters while an

operator is pending.

- Ex commands can be used to move the cursor. This can be

used to call a function that does some complicated motion.

The motion is always characterwise exclusive, no matter

what ":" command is used. This means it's impossible to

include the last character of a line without the line break

(unless 'virtualedit' is set).

If the Ex command changes the text before where the operator

starts or jumps to another buffer the result is

unpredictable. It is possible to change the text further

down. Jumping to another buffer is possible if the current

buffer is not unloaded.

motion.txt\* For Vim version 7.4. Last change: 2015 Jun 06

VIM REFERENCE MANUAL by Bram Moolenaar

Cursor motions \*cursor-motions\* \*navigation\*

These commands move the cursor position. If the new position is off of the

screen, the screen is scrolled to show the cursor (see also 'scrolljump' and

'scrolloff' options).

1. Motions and operators |operator|

2. Left-right motions |left-right-motions|

3. Up-down motions |up-down-motions|

4. Word motions |word-motions|

5. Text object motions |object-motions|

6. Text object selection |object-select|

7. Marks |mark-motions|

8. Jumps |jump-motions|

9. Various motions |various-motions|

General remarks:

If you want to know where you are in the file use the "CTRL-G" command

|CTRL-G| or the "g CTRL-G" command |g\_CTRL-G|. If you set the 'ruler' option,

the cursor position is continuously shown in the status line (which slows down

Vim a little).

Experienced users prefer the hjkl keys because they are always right under

their fingers. Beginners often prefer the arrow keys, because they do not

know what the hjkl keys do. The mnemonic value of hjkl is clear from looking

at the keyboard. Think of j as an arrow pointing downwards.

The 'virtualedit' option can be set to make it possible to move the cursor to

positions where there is no character or halfway a character.

==============================================================================

1. Motions and operators \*operator\*

The motion commands can be used after an operator command, to have the command

operate on the text that was moved over. That is the text between the cursor

position before and after the motion. Operators are generally used to delete

or change text. The following operators are available:

c change

d delete

y yank into register (does not change the text)

~ swap case (only if 'tildeop' is set)

g~ swap case

gu make lowercase

gU make uppercase

! filter through an external program

= filter through 'equalprg' or C-indenting if empty

gq text formatting

g? ROT13 encoding

> shift right

< shift left

zf define a fold

g@ call function set with the 'operatorfunc' option

If the motion includes a count and the operator also had a count before it,

the two counts are multiplied. For example: "2d3w" deletes six words.

After applying the operator the cursor is mostly left at the start of the text

that was operated upon. For example, "yfe" doesn't move the cursor, but "yFe"

moves the cursor leftwards to the "e" where the yank started.

\*linewise\* \*characterwise\*

The operator either affects whole lines, or the characters between the start

and end position. Generally, motions that move between lines affect lines

(are linewise), and motions that move within a line affect characters (are

characterwise). However, there are some exceptions.

\*exclusive\* \*inclusive\*

A character motion is either inclusive or exclusive. When inclusive, the

start and end position of the motion are included in the operation. When

exclusive, the last character towards the end of the buffer is not included.

Linewise motions always include the start and end position.

Which motions are linewise, inclusive or exclusive is mentioned with the

command. There are however, two general exceptions:

1. If the motion is exclusive and the end of the motion is in column 1, the

end of the motion is moved to the end of the previous line and the motion

becomes inclusive. Example: "}" moves to the first line after a paragraph,

but "d}" will not include that line.

\*exclusive-linewise\*

2. If the motion is exclusive, the end of the motion is in column 1 and the

start of the motion was at or before the first non-blank in the line, the

motion becomes linewise. Example: If a paragraph begins with some blanks

and you do "d}" while standing on the first non-blank, all the lines of

the paragraph are deleted, including the blanks. If you do a put now, the

deleted lines will be inserted below the cursor position.

Note that when the operator is pending (the operator command is typed, but the

motion isn't yet), a special set of mappings can be used. See |:omap|.

Instead of first giving the operator and then a motion you can use Visual

mode: mark the start of the text with "v", move the cursor to the end of the

text that is to be affected and then hit the operator. The text between the

start and the cursor position is highlighted, so you can see what text will

be operated upon. This allows much more freedom, but requires more key

strokes and has limited redo functionality. See the chapter on Visual mode

|Visual-mode|.

You can use a ":" command for a motion. For example "d:call FindEnd()".

But this can't be repeated with "." if the command is more than one line.

This can be repeated: >

d:call search("f")<CR>

This cannot be repeated: >

d:if 1<CR>

call search("f")<CR>

endif<CR>

Note that when using ":" any motion becomes characterwise exclusive.

FORCING A MOTION TO BE LINEWISE, CHARACTERWISE OR BLOCKWISE

When a motion is not of the type you would like to use, you can force another

type by using "v", "V" or CTRL-V just after the operator.

Example: >

dj deletes two lines >

dvj deletes from the cursor position until the character below the cursor >

d<C-V>j deletes the character under the cursor and the character below the cursor. >

Be careful with forcing a linewise movement to be used characterwise or

blockwise, the column may not always be defined.

\*o\_v\*

v When used after an operator, before the motion command: Force

the operator to work characterwise, also when the motion is

linewise. If the motion was linewise, it will become

|exclusive|.

If the motion already was characterwise, toggle

inclusive/exclusive. This can be used to make an exclusive

motion inclusive and an inclusive motion exclusive.

\*o\_V\*

V When used after an operator, before the motion command: Force

the operator to work linewise, also when the motion is

characterwise.

\*o\_CTRL-V\*

CTRL-V When used after an operator, before the motion command: Force

the operator to work blockwise. This works like Visual block

mode selection, with the corners defined by the cursor

position before and after the motion.

==============================================================================

2. Left-right motions \*left-right-motions\*

These commands move the cursor to the specified column in the current line.

They stop at the first column and at the end of the line, except "$", which

may move to one of the next lines. See 'whichwrap' option to make some of the

commands move across line boundaries.

h or \*h\*

<Left> or \*<Left>\*

CTRL-H or \*CTRL-H\* \*<BS>\*

<BS> [count] characters to the left. |exclusive| motion.

Note: If you prefer <BS> to delete a character, use

the mapping:

:map CTRL-V<BS> X

(to enter "CTRL-V<BS>" type the CTRL-V key, followed

by the <BS> key)

See |:fixdel| if the <BS> key does not do what you

want.

l or \*l\*

<Right> or \*<Right>\* \*<Space>\*

<Space> [count] characters to the right. |exclusive| motion.

See the 'whichwrap' option for adjusting the behavior

at end of line

\*0\*

0 To the first character of the line. |exclusive|

motion.

\*<Home>\* \*<kHome>\*

<Home> To the first character of the line. |exclusive|

motion. When moving up or down next, stay in same

TEXT column (if possible). Most other commands stay

in the same SCREEN column. <Home> works like "1|",

which differs from "0" when the line starts with a

<Tab>. {not in Vi}

\*^\*

^ To the first non-blank character of the line.

|exclusive| motion.

\*$\* \*<End>\* \*<kEnd>\*

$ or <End> To the end of the line. When a count is given also go

[count - 1] lines downward |inclusive|.

In Visual mode the cursor goes to just after the last

character in the line.

When 'virtualedit' is active, "$" may move the cursor

back from past the end of the line to the last

character in the line.

\*g\_\*

g\_ To the last non-blank character of the line and

[count - 1] lines downward |inclusive|. {not in Vi}

\*g0\* \*g<Home>\*

g0 or g<Home> When lines wrap ('wrap' on): To the first character of

the screen line. |exclusive| motion. Differs from

"0" when a line is wider than the screen.

When lines don't wrap ('wrap' off): To the leftmost

character of the current line that is on the screen.

Differs from "0" when the first character of the line

is not on the screen. {not in Vi}

\*g^\*

g^ When lines wrap ('wrap' on): To the first non-blank

character of the screen line. |exclusive| motion.

Differs from "^" when a line is wider than the screen.

When lines don't wrap ('wrap' off): To the leftmost

non-blank character of the current line that is on the

screen. Differs from "^" when the first non-blank

character of the line is not on the screen. {not in

Vi}

\*gm\*

gm Like "g0", but half a screenwidth to the right (or as

much as possible). {not in Vi}

\*g$\* \*g<End>\*

g$ or g<End> When lines wrap ('wrap' on): To the last character of

the screen line and [count - 1] screen lines downward

|inclusive|. Differs from "$" when a line is wider

than the screen.

When lines don't wrap ('wrap' off): To the rightmost

character of the current line that is visible on the

screen. Differs from "$" when the last character of

the line is not on the screen or when a count is used.

Additionally, vertical movements keep the column,

instead of going to the end of the line.

When 'virtualedit' is enabled moves to the end of the

screen line.

{not in Vi}

\*bar\*

| To screen column [count] in the current line.

|exclusive| motion. Ceci n'est pas une pipe.

\*f\*

f{char} To [count]'th occurrence of {char} to the right. The

cursor is placed on {char} |inclusive|.

{char} can be entered as a digraph |digraph-arg|.

When 'encoding' is set to Unicode, composing

characters may be used, see |utf-8-char-arg|.

|:lmap| mappings apply to {char}. The CTRL-^ command

in Insert mode can be used to switch this on/off

|i\_CTRL-^|.

\*F\*

F{char} To the [count]'th occurrence of {char} to the left.

The cursor is placed on {char} |exclusive|.

{char} can be entered like with the |f| command.

\*t\*

t{char} Till before [count]'th occurrence of {char} to the

right. The cursor is placed on the character left of

{char} |inclusive|.

{char} can be entered like with the |f| command.

\*T\*

T{char} Till after [count]'th occurrence of {char} to the

left. The cursor is placed on the character right of

{char} |exclusive|.

{char} can be entered like with the |f| command.

\*;\*

; Repeat latest f, t, F or T [count] times. See |cpo-;|

\*,\*

, Repeat latest f, t, F or T in opposite direction

[count] times. See also |cpo-;|

==============================================================================

3. Up-down motions \*up-down-motions\*

k or \*k\*

<Up> or \*<Up>\* \*CTRL-P\*

CTRL-P [count] lines upward |linewise|.

j or \*j\*

<Down> or \*<Down>\*

CTRL-J or \*CTRL-J\*

<NL> or \*<NL>\* \*CTRL-N\*

CTRL-N [count] lines downward |linewise|.

gk or \*gk\* \*g<Up>\*

g<Up> [count] display lines upward. |exclusive| motion.

Differs from 'k' when lines wrap, and when used with

an operator, because it's not linewise. {not in Vi}

gj or \*gj\* \*g<Down>\*

g<Down> [count] display lines downward. |exclusive| motion.

Differs from 'j' when lines wrap, and when used with

an operator, because it's not linewise. {not in Vi}

\*-\*

- <minus> [count] lines upward, on the first non-blank

character |linewise|.

+ or \*+\*

CTRL-M or \*CTRL-M\* \*<CR>\*

<CR> [count] lines downward, on the first non-blank

character |linewise|.

\*\_\*

\_ <underscore> [count] - 1 lines downward, on the first non-blank

character |linewise|.

\*G\*

G Goto line [count], default last line, on the first

non-blank character |linewise|. If 'startofline' not

set, keep the same column.

G is a one of |jump-motions|.

\*<C-End>\*

<C-End> Goto line [count], default last line, on the last

character |inclusive|. {not in Vi}

<C-Home> or \*gg\* \*<C-Home>\*

gg Goto line [count], default first line, on the first

non-blank character |linewise|. If 'startofline' not

set, keep the same column.

\*:[range]\*

:[range] Set the cursor on the last line number in [range].

[range] can also be just one line number, e.g., ":1"

or ":'m".

In contrast with |G| this command does not modify the

|jumplist|.

\*N%\*

{count}% Go to {count} percentage in the file, on the first

non-blank in the line |linewise|. To compute the new

line number this formula is used:

({count} \* number-of-lines + 99) / 100

See also 'startofline' option. {not in Vi}

:[range]go[to] [count] \*:go\* \*:goto\* \*go\*

[count]go Go to [count] byte in the buffer. Default [count] is

one, start of the file. When giving [range], the

last number in it used as the byte count. End-of-line

characters are counted depending on the current

'fileformat' setting.

Also see the |line2byte()| function, and the 'o'

option in 'statusline'.

{not in Vi}

{not available when compiled without the

|+byte\_offset| feature}

These commands move to the specified line. They stop when reaching the first

or the last line. The first two commands put the cursor in the same column

(if possible) as it was after the last command that changed the column,

except after the "$" command, then the cursor will be put on the last

character of the line.

If "k", "-" or CTRL-P is used with a [count] and there are less than [count]

lines above the cursor and the 'cpo' option includes the "-" flag it is an

error. |cpo--|.

==============================================================================

4. Word motions \*word-motions\*

<S-Right> or \*<S-Right>\* \*w\*

w [count] words forward. |exclusive| motion.

<C-Right> or \*<C-Right>\* \*W\*

W [count] WORDS forward. |exclusive| motion.

\*e\*

e Forward to the end of word [count] |inclusive|.

Does not stop in an empty line.

\*E\*

E Forward to the end of WORD [count] |inclusive|.

Does not stop in an empty line.

<S-Left> or \*<S-Left>\* \*b\*

b [count] words backward. |exclusive| motion.

<C-Left> or \*<C-Left>\* \*B\*

B [count] WORDS backward. |exclusive| motion.

\*ge\*

ge Backward to the end of word [count] |inclusive|.

\*gE\*

gE Backward to the end of WORD [count] |inclusive|.

These commands move over words or WORDS.

\*word\*

A word consists of a sequence of letters, digits and underscores, or a

sequence of other non-blank characters, separated with white space (spaces,

tabs, <EOL>). This can be changed with the 'iskeyword' option. An empty line

is also considered to be a word.

\*WORD\*

A WORD consists of a sequence of non-blank characters, separated with white

space. An empty line is also considered to be a WORD.

A sequence of folded lines is counted for one word of a single character.

"w" and "W", "e" and "E" move to the start/end of the first word or WORD after

a range of folded lines. "b" and "B" move to the start of the first word or

WORD before the fold.

Special case: "cw" and "cW" are treated like "ce" and "cE" if the cursor is

on a non-blank. This is because "cw" is interpreted as change-word, and a

word does not include the following white space. {Vi: "cw" when on a blank

followed by other blanks changes only the first blank; this is probably a

bug, because "dw" deletes all the blanks}

Another special case: When using the "w" motion in combination with an

operator and the last word moved over is at the end of a line, the end of

that word becomes the end of the operated text, not the first word in the

next line.

The original Vi implementation of "e" is buggy. For example, the "e" command

will stop on the first character of a line if the previous line was empty.

But when you use "2e" this does not happen. In Vim "ee" and "2e" are the

same, which is more logical. However, this causes a small incompatibility

between Vi and Vim.

==============================================================================

5. Text object motions \*object-motions\*

\*(\*

( [count] sentences backward. |exclusive| motion.

\*)\*

) [count] sentences forward. |exclusive| motion.

\*{\*

{ [count] paragraphs backward. |exclusive| motion.

\*}\*

} [count] paragraphs forward. |exclusive| motion.

\*]]\*

]] [count] sections forward or to the next '{' in the

first column. When used after an operator, then also

stops below a '}' in the first column. |exclusive|

Note that |exclusive-linewise| often applies.

\*][\*

][ [count] sections forward or to the next '}' in the

first column. |exclusive|

Note that |exclusive-linewise| often applies.

\*[[\*

[[ [count] sections backward or to the previous '{' in

the first column. |exclusive|

Note that |exclusive-linewise| often applies.

\*[]\*

[] [count] sections backward or to the previous '}' in

the first column. |exclusive|

Note that |exclusive-linewise| often applies.

These commands move over three kinds of text objects.

\*sentence\*

A sentence is defined as ending at a '.', '!' or '?' followed by either the

end of a line, or by a space or tab. Any number of closing ')', ']', '"'

and ''' characters may appear after the '.', '!' or '?' before the spaces,

tabs or end of line. A paragraph and section boundary is also a sentence

boundary.

If the 'J' flag is present in 'cpoptions', at least two spaces have to

follow the punctuation mark; <Tab>s are not recognized as white space.

The definition of a sentence cannot be changed.

\*paragraph\*

A paragraph begins after each empty line, and also at each of a set of

paragraph macros, specified by the pairs of characters in the 'paragraphs'

option. The default is "IPLPPPQPP TPHPLIPpLpItpplpipbp", which corresponds to

the macros ".IP", ".LP", etc. (These are nroff macros, so the dot must be in

the first column). A section boundary is also a paragraph boundary.

Note that a blank line (only containing white space) is NOT a paragraph

boundary.

Also note that this does not include a '{' or '}' in the first column. When

the '{' flag is in 'cpoptions' then '{' in the first column is used as a

paragraph boundary |posix|.

\*section\*

A section begins after a form-feed (<C-L>) in the first column and at each of

a set of section macros, specified by the pairs of characters in the

'sections' option. The default is "SHNHH HUnhsh", which defines a section to

start at the nroff macros ".SH", ".NH", ".H", ".HU", ".nh" and ".sh".

The "]" and "[" commands stop at the '{' or '}' in the first column. This is

useful to find the start or end of a function in a C program. Note that the

first character of the command determines the search direction and the

second character the type of brace found.

If your '{' or '}' are not in the first column, and you would like to use "[["

and "]]" anyway, try these mappings: >

:map [[ ?{<CR>w99[{

:map ][ /}<CR>b99]}

:map ]] j0[[%/{<CR>

:map [] k$][%?}<CR>

[type these literally, see |<>|]

==============================================================================

6. Text object selection \*object-select\* \*text-objects\*

\*v\_a\* \*v\_i\*

This is a series of commands that can only be used while in Visual mode or

after an operator. The commands that start with "a" select "a"n object

including white space, the commands starting with "i" select an "inner" object

without white space, or just the white space. Thus the "inner" commands

always select less text than the "a" commands.

These commands are {not in Vi}.

These commands are not available when the |+textobjects| feature has been

disabled at compile time.

Also see `gn` and `gN`, operating on the last search pattern.

\*v\_aw\* \*aw\*

aw "a word", select [count] words (see |word|).

Leading or trailing white space is included, but not

counted.

When used in Visual linewise mode "aw" switches to

Visual characterwise mode.

\*v\_iw\* \*iw\*

iw "inner word", select [count] words (see |word|).

White space between words is counted too.

When used in Visual linewise mode "iw" switches to

Visual characterwise mode.

\*v\_aW\* \*aW\*

aW "a WORD", select [count] WORDs (see |WORD|).

Leading or trailing white space is included, but not

counted.

When used in Visual linewise mode "aW" switches to

Visual characterwise mode.

\*v\_iW\* \*iW\*

iW "inner WORD", select [count] WORDs (see |WORD|).

White space between words is counted too.

When used in Visual linewise mode "iW" switches to

Visual characterwise mode.

\*v\_as\* \*as\*

as "a sentence", select [count] sentences (see

|sentence|).

When used in Visual mode it is made characterwise.

\*v\_is\* \*is\*

is "inner sentence", select [count] sentences (see

|sentence|).

When used in Visual mode it is made characterwise.

\*v\_ap\* \*ap\*

ap "a paragraph", select [count] paragraphs (see

|paragraph|).

Exception: a blank line (only containing white space)

is also a paragraph boundary.

When used in Visual mode it is made linewise.

\*v\_ip\* \*ip\*

ip "inner paragraph", select [count] paragraphs (see

|paragraph|).

Exception: a blank line (only containing white space)

is also a paragraph boundary.

When used in Visual mode it is made linewise.

a] \*v\_a]\* \*v\_a[\* \*a]\* \*a[\*

a[ "a [] block", select [count] '[' ']' blocks. This

goes backwards to the [count] unclosed '[', and finds

the matching ']'. The enclosed text is selected,

including the '[' and ']'.

When used in Visual mode it is made characterwise.

i] \*v\_i]\* \*v\_i[\* \*i]\* \*i[\*

i[ "inner [] block", select [count] '[' ']' blocks. This

goes backwards to the [count] unclosed '[', and finds

the matching ']'. The enclosed text is selected,

excluding the '[' and ']'.

When used in Visual mode it is made characterwise.

a) \*v\_a)\* \*a)\* \*a(\*

a( \*v\_ab\* \*v\_a(\* \*ab\*

ab "a block", select [count] blocks, from "[count] [(" to

the matching ')', including the '(' and ')' (see

|[(|). Does not include white space outside of the

parenthesis.

When used in Visual mode it is made characterwise.

i) \*v\_i)\* \*i)\* \*i(\*

i( \*v\_ib\* \*v\_i(\* \*ib\*

ib "inner block", select [count] blocks, from "[count] [("

to the matching ')', excluding the '(' and ')' (see

|[(|).

When used in Visual mode it is made characterwise.

a> \*v\_a>\* \*v\_a<\* \*a>\* \*a<\*

a< "a <> block", select [count] <> blocks, from the

[count]'th unmatched '<' backwards to the matching

'>', including the '<' and '>'.

When used in Visual mode it is made characterwise.

i> \*v\_i>\* \*v\_i<\* \*i>\* \*i<\*

i< "inner <> block", select [count] <> blocks, from

the [count]'th unmatched '<' backwards to the matching

'>', excluding the '<' and '>'.

When used in Visual mode it is made characterwise.

\*v\_at\* \*at\*

at "a tag block", select [count] tag blocks, from the

[count]'th unmatched "<aaa>" backwards to the matching

"</aaa>", including the "<aaa>" and "</aaa>".

See |tag-blocks| about the details.

When used in Visual mode it is made characterwise.

\*v\_it\* \*it\*

it "inner tag block", select [count] tag blocks, from the

[count]'th unmatched "<aaa>" backwards to the matching

"</aaa>", excluding the "<aaa>" and "</aaa>".

See |tag-blocks| about the details.

When used in Visual mode it is made characterwise.

a} \*v\_a}\* \*a}\* \*a{\*

a{ \*v\_aB\* \*v\_a{\* \*aB\*

aB "a Block", select [count] Blocks, from "[count] [{" to

the matching '}', including the '{' and '}' (see

|[{|).

When used in Visual mode it is made characterwise.

i} \*v\_i}\* \*i}\* \*i{\*

i{ \*v\_iB\* \*v\_i{\* \*iB\*

iB "inner Block", select [count] Blocks, from "[count] [{"

to the matching '}', excluding the '{' and '}' (see

|[{|).

When used in Visual mode it is made characterwise.

a" \*v\_aquote\* \*aquote\*

a' \*v\_a'\* \*a'\*

a` \*v\_a`\* \*a`\*

"a quoted string". Selects the text from the previous

quote until the next quote. The 'quoteescape' option

is used to skip escaped quotes.

Only works within one line.

When the cursor starts on a quote, Vim will figure out

which quote pairs form a string by searching from the

start of the line.

Any trailing white space is included, unless there is

none, then leading white space is included.

When used in Visual mode it is made characterwise.

Repeating this object in Visual mode another string is

included. A count is currently not used.

i" \*v\_iquote\* \*iquote\*

i' \*v\_i'\* \*i'\*

i` \*v\_i`\* \*i`\*

Like a", a' and a`, but exclude the quotes and

repeating won't extend the Visual selection.

Special case: With a count of 2 the quotes are

included, but no extra white space as with a"/a'/a`.

When used after an operator:

For non-block objects:

For the "a" commands: The operator applies to the object and the white

space after the object. If there is no white space after the object

or when the cursor was in the white space before the object, the white

space before the object is included.

For the "inner" commands: If the cursor was on the object, the

operator applies to the object. If the cursor was on white space, the

operator applies to the white space.

For a block object:

The operator applies to the block where the cursor is in, or the block

on which the cursor is on one of the braces. For the "inner" commands

the surrounding braces are excluded. For the "a" commands, the braces

are included.

When used in Visual mode:

When start and end of the Visual area are the same (just after typing "v"):

One object is selected, the same as for using an operator.

When start and end of the Visual area are not the same:

For non-block objects the area is extended by one object or the white

space up to the next object, or both for the "a" objects. The

direction in which this happens depends on which side of the Visual

area the cursor is. For the block objects the block is extended one

level outwards.

For illustration, here is a list of delete commands, grouped from small to big

objects. Note that for a single character and a whole line the existing vi

movement commands are used.

"dl" delete character (alias: "x") |dl|

"diw" delete inner word \*diw\*

"daw" delete a word \*daw\*

"diW" delete inner WORD (see |WORD|) \*diW\*

"daW" delete a WORD (see |WORD|) \*daW\*

"dgn" delete the next search pattern match \*dgn\*

"dd" delete one line |dd|

"dis" delete inner sentence \*dis\*

"das" delete a sentence \*das\*

"dib" delete inner '(' ')' block \*dib\*

"dab" delete a '(' ')' block \*dab\*

"dip" delete inner paragraph \*dip\*

"dap" delete a paragraph \*dap\*

"diB" delete inner '{' '}' block \*diB\*

"daB" delete a '{' '}' block \*daB\*

Note the difference between using a movement command and an object. The

movement command operates from here (cursor position) to where the movement

takes us. When using an object the whole object is operated upon, no matter

where on the object the cursor is. For example, compare "dw" and "daw": "dw"

deletes from the cursor position to the start of the next word, "daw" deletes

the word under the cursor and the space after or before it.

Tag blocks \*tag-blocks\*

For the "it" and "at" text objects an attempt is done to select blocks between

matching tags for HTML and XML. But since these are not completely compatible

there are a few restrictions.

The normal method is to select a <tag> until the matching </tag>. For "at"

the tags are included, for "it" they are excluded. But when "it" is repeated

the tags will be included (otherwise nothing would change). Also, "it" used

on a tag block with no contents will select the leading tag.

"<aaa/>" items are skipped. Case is ignored, also for XML where case does

matter.

In HTML it is possible to have a tag like <br> or <meta ...> without a

matching end tag. These are ignored.

The text objects are tolerant about mistakes. Stray end tags are ignored.

==============================================================================

7. Marks \*mark-motions\* \*E20\* \*E78\*

Jumping to a mark can be done in two ways:

1. With ` (backtick): The cursor is positioned at the specified location

and the motion is |exclusive|.

2. With ' (single quote): The cursor is positioned on the first non-blank

character in the line of the specified location and

the motion is linewise.

\*m\* \*mark\* \*Mark\*

m{a-zA-Z} Set mark {a-zA-Z} at cursor position (does not move

the cursor, this is not a motion command).

\*m'\* \*m`\*

m' or m` Set the previous context mark. This can be jumped to

with the "''" or "``" command (does not move the

cursor, this is not a motion command).

\*m[\* \*m]\*

m[ or m] Set the |'[| or |']| mark. Useful when an operator is

to be simulated by multiple commands. (does not move

the cursor, this is not a motion command).

\*m<\* \*m>\*

m< or m> Set the |'<| or |'>| mark. Useful to change what the

`gv` command selects. (does not move the cursor, this

is not a motion command).

Note that the Visual mode cannot be set, only the

start and end position.

\*:ma\* \*:mark\* \*E191\*

:[range]ma[rk] {a-zA-Z'}

Set mark {a-zA-Z'} at last line number in [range],

column 0. Default is cursor line.

\*:k\*

:[range]k{a-zA-Z'} Same as :mark, but the space before the mark name can

be omitted.

\*'\* \*'a\* \*`\* \*`a\*

'{a-z} `{a-z} Jump to the mark {a-z} in the current buffer.

\*'A\* \*'0\* \*`A\* \*`0\*

'{A-Z0-9} `{A-Z0-9} To the mark {A-Z0-9} in the file where it was set (not

a motion command when in another file). {not in Vi}

\*g'\* \*g'a\* \*g`\* \*g`a\*

g'{mark} g`{mark}

Jump to the {mark}, but don't change the jumplist when

jumping within the current buffer. Example: >

g`"

< jumps to the last known position in a file. See

$VIMRUNTIME/vimrc\_example.vim.

Also see |:keepjumps|.

{not in Vi}

\*:marks\*

:marks List all the current marks (not a motion command).

The |'(|, |')|, |'{| and |'}| marks are not listed.

The first column has number zero.

{not in Vi}

\*E283\*

:marks {arg} List the marks that are mentioned in {arg} (not a

motion command). For example: >

:marks aB

< to list marks 'a' and 'B'. {not in Vi}

\*:delm\* \*:delmarks\*

:delm[arks] {marks} Delete the specified marks. Marks that can be deleted

include A-Z and 0-9. You cannot delete the ' mark.

They can be specified by giving the list of mark

names, or with a range, separated with a dash. Spaces

are ignored. Examples: >

:delmarks a deletes mark a

:delmarks a b 1 deletes marks a, b and 1

:delmarks Aa deletes marks A and a

:delmarks p-z deletes marks in the range p to z

:delmarks ^.[] deletes marks ^ . [ ]

:delmarks \" deletes mark "

< {not in Vi}

:delm[arks]! Delete all marks for the current buffer, but not marks

A-Z or 0-9.

{not in Vi}

A mark is not visible in any way. It is just a position in the file that is

remembered. Do not confuse marks with named registers, they are totally

unrelated.

'a - 'z lowercase marks, valid within one file

'A - 'Z uppercase marks, also called file marks, valid between files

'0 - '9 numbered marks, set from .viminfo file

Lowercase marks 'a to 'z are remembered as long as the file remains in the

buffer list. If you remove the file from the buffer list, all its marks are

lost. If you delete a line that contains a mark, that mark is erased.

Lowercase marks can be used in combination with operators. For example: "d't"

deletes the lines from the cursor position to mark 't'. Hint: Use mark 't' for

Top, 'b' for Bottom, etc.. Lowercase marks are restored when using undo and

redo.

Uppercase marks 'A to 'Z include the file name. {Vi: no uppercase marks} You

can use them to jump from file to file. You can only use an uppercase mark

with an operator if the mark is in the current file. The line number of the

mark remains correct, even if you insert/delete lines or edit another file for

a moment. When the 'viminfo' option is not empty, uppercase marks are kept in

the .viminfo file. See |viminfo-file-marks|.

Numbered marks '0 to '9 are quite different. They can not be set directly.

They are only present when using a viminfo file |viminfo-file|. Basically '0

is the location of the cursor when you last exited Vim, '1 the last but one

time, etc. Use the "r" flag in 'viminfo' to specify files for which no

Numbered mark should be stored. See |viminfo-file-marks|.

\*'[\* \*`[\*

'[ `[ To the first character of the previously changed

or yanked text. {not in Vi}

\*']\* \*`]\*

'] `] To the last character of the previously changed or

yanked text. {not in Vi}

After executing an operator the Cursor is put at the beginning of the text

that was operated upon. After a put command ("p" or "P") the cursor is

sometimes placed at the first inserted line and sometimes on the last inserted

character. The four commands above put the cursor at either end. Example:

After yanking 10 lines you want to go to the last one of them: "10Y']". After

inserting several lines with the "p" command you want to jump to the lowest

inserted line: "p']". This also works for text that has been inserted.

Note: After deleting text, the start and end positions are the same, except

when using blockwise Visual mode. These commands do not work when no change

was made yet in the current file.

\*'<\* \*`<\*

'< `< To the first line or character of the last selected

Visual area in the current buffer. For block mode it

may also be the last character in the first line (to

be able to define the block). {not in Vi}.

\*'>\* \*`>\*

'> `> To the last line or character of the last selected

Visual area in the current buffer. For block mode it

may also be the first character of the last line (to

be able to define the block). Note that 'selection'

applies, the position may be just after the Visual

area. {not in Vi}.

\*''\* \*``\*

'' `` To the position before the latest jump, or where the

last "m'" or "m`" command was given. Not set when the

|:keepjumps| command modifier was used.

Also see |restore-position|.

\*'quote\* \*`quote\*

'" `" To the cursor position when last exiting the current

buffer. Defaults to the first character of the first

line. See |last-position-jump| for how to use this

for each opened file.

Only one position is remembered per buffer, not one

for each window. As long as the buffer is visible in

a window the position won't be changed.

{not in Vi}.

\*'^\* \*`^\*

'^ `^ To the position where the cursor was the last time

when Insert mode was stopped. This is used by the

|gi| command. Not set when the |:keepjumps| command

modifier was used. {not in Vi}

\*'.\* \*`.\*

'. `. To the position where the last change was made. The

position is at or near where the change started.

Sometimes a command is executed as several changes,

then the position can be near the end of what the

command changed. For example when inserting a word,

the position will be on the last character.

{not in Vi}

\*'(\* \*`(\*

'( `( To the start of the current sentence, like the |(|

command. {not in Vi}

\*')\* \*`)\*

') `) To the end of the current sentence, like the |)|

command. {not in Vi}

\*'{\* \*`{\*

'{ `{ To the start of the current paragraph, like the |{|

command. {not in Vi}

\*'}\* \*`}\*

'} `} To the end of the current paragraph, like the |}|

command. {not in Vi}

These commands are not marks themselves, but jump to a mark:

\*]'\*

]' [count] times to next line with a lowercase mark below

the cursor, on the first non-blank character in the

line. {not in Vi}

\*]`\*

]` [count] times to lowercase mark after the cursor. {not

in Vi}

\*['\*

[' [count] times to previous line with a lowercase mark

before the cursor, on the first non-blank character in

the line. {not in Vi}

\*[`\*

[` [count] times to lowercase mark before the cursor.

{not in Vi}

:loc[kmarks] {command} \*:loc\* \*:lockmarks\*

Execute {command} without adjusting marks. This is

useful when changing text in a way that the line count

will be the same when the change has completed.

WARNING: When the line count does change, marks below

the change will keep their line number, thus move to

another text line.

These items will not be adjusted for deleted/inserted

lines:

- lower case letter marks 'a - 'z

- upper case letter marks 'A - 'Z

- numbered marks '0 - '9

- last insert position '^

- last change position '.

- the Visual area '< and '>

- line numbers in placed signs

- line numbers in quickfix positions

- positions in the |jumplist|

- positions in the |tagstack|

These items will still be adjusted:

- previous context mark ''

- the cursor position

- the view of a window on a buffer

- folds

- diffs

:kee[pmarks] {command} \*:kee\* \*:keepmarks\*

Currently only has effect for the filter command

|:range!|:

- When the number of lines after filtering is equal to

or larger than before, all marks are kept at the

same line number.

- When the number of lines decreases, the marks in the

lines that disappeared are deleted.

In any case the marks below the filtered text have

their line numbers adjusted, thus stick to the text,

as usual.

When the 'R' flag is missing from 'cpoptions' this has

the same effect as using ":keepmarks".

\*:keepj\* \*:keepjumps\*

:keepj[umps] {command}

Moving around in {command} does not change the |''|,

|'.| and |'^| marks, the |jumplist| or the

|changelist|.

Useful when making a change or inserting text

automatically and the user doesn't want to go to this

position. E.g., when updating a "Last change"

timestamp in the first line: >

:let lnum = line(".")

:keepjumps normal gg

:call SetLastChange()

:keepjumps exe "normal " . lnum . "G"

<

Note that ":keepjumps" must be used for every command.

When invoking a function the commands in that function

can still change the jumplist. Also, for

":keepjumps exe 'command '" the "command" won't keep

jumps. Instead use: ":exe 'keepjumps command'"

==============================================================================

8. Jumps \*jump-motions\*

A "jump" is one of the following commands: "'", "`", "G", "/", "?", "n",

"N", "%", "(", ")", "[[", "]]", "{", "}", ":s", ":tag", "L", "M", "H" and

the commands that start editing a new file. If you make the cursor "jump"

with one of these commands, the position of the cursor before the jump is

remembered. You can return to that position with the "''" and "``" command,

unless the line containing that position was changed or deleted.

\*CTRL-O\*

CTRL-O Go to [count] Older cursor position in jump list

(not a motion command). {not in Vi}

{not available without the |+jumplist| feature}

<Tab> or \*CTRL-I\* \*<Tab>\*

CTRL-I Go to [count] newer cursor position in jump list

(not a motion command).

{not in Vi}

{not available without the |+jumplist| feature}

\*:ju\* \*:jumps\*

:ju[mps] Print the jump list (not a motion command). {not in

Vi} {not available without the |+jumplist| feature}

\*jumplist\*

Jumps are remembered in a jump list. With the CTRL-O and CTRL-I command you

can go to cursor positions before older jumps, and back again. Thus you can

move up and down the list. There is a separate jump list for each window.

The maximum number of entries is fixed at 100.

{not available without the |+jumplist| feature}

For example, after three jump commands you have this jump list:

jump line col file/text ~

3 1 0 some text ~

2 70 0 another line ~

1 1154 23 end. ~

> ~

The "file/text" column shows the file name, or the text at the jump if it is

in the current file (an indent is removed and a long line is truncated to fit

in the window).

You are currently in line 1167. If you then use the CTRL-O command, the

cursor is put in line 1154. This results in:

jump line col file/text ~

2 1 0 some text ~

1 70 0 another line ~

> 0 1154 23 end. ~

1 1167 0 foo bar ~

The pointer will be set at the last used jump position. The next CTRL-O

command will use the entry above it, the next CTRL-I command will use the

entry below it. If the pointer is below the last entry, this indicates that

you did not use a CTRL-I or CTRL-O before. In this case the CTRL-O command

will cause the cursor position to be added to the jump list, so you can get

back to the position before the CTRL-O. In this case this is line 1167.

With more CTRL-O commands you will go to lines 70 and 1. If you use CTRL-I

you can go back to 1154 and 1167 again. Note that the number in the "jump"

column indicates the count for the CTRL-O or CTRL-I command that takes you to

this position.

If you use a jump command, the current line number is inserted at the end of

the jump list. If the same line was already in the jump list, it is removed.

The result is that when repeating CTRL-O you will get back to old positions

only once.

When the |:keepjumps| command modifier is used, jumps are not stored in the

jumplist. Jumps are also not stored in other cases, e.g., in a |:global|

command. You can explicitly add a jump by setting the ' mark with "m'". Note

that calling setpos() does not do this.

After the CTRL-O command that got you into line 1154 you could give another

jump command (e.g., "G"). The jump list would then become:

jump line col file/text ~

4 1 0 some text ~

3 70 0 another line ~

2 1167 0 foo bar ~

1 1154 23 end. ~

> ~

The line numbers will be adjusted for deleted and inserted lines. This fails

if you stop editing a file without writing, like with ":n!".

When you split a window, the jumplist will be copied to the new window.

If you have included the ' item in the 'viminfo' option the jumplist will be

stored in the viminfo file and restored when starting Vim.

CHANGE LIST JUMPS \*changelist\* \*change-list-jumps\* \*E664\*

When making a change the cursor position is remembered. One position is

remembered for every change that can be undone, unless it is close to a

previous change. Two commands can be used to jump to positions of changes,

also those that have been undone:

\*g;\* \*E662\*

g; Go to [count] older position in change list.

If [count] is larger than the number of older change

positions go to the oldest change.

If there is no older change an error message is given.

(not a motion command)

{not in Vi}

{not available without the |+jumplist| feature}

\*g,\* \*E663\*

g, Go to [count] newer cursor position in change list.

Just like |g;| but in the opposite direction.

(not a motion command)

{not in Vi}

{not available without the |+jumplist| feature}

When using a count you jump as far back or forward as possible. Thus you can

use "999g;" to go to the first change for which the position is still

remembered. The number of entries in the change list is fixed and is the same

as for the |jumplist|.

When two undo-able changes are in the same line and at a column position less

than 'textwidth' apart only the last one is remembered. This avoids that a

sequence of small changes in a line, for example "xxxxx", adds many positions

to the change list. When 'textwidth' is zero 'wrapmargin' is used. When that

also isn't set a fixed number of 79 is used. Detail: For the computations

bytes are used, not characters, to avoid a speed penalty (this only matters

for multi-byte encodings).

Note that when text has been inserted or deleted the cursor position might be

a bit different from the position of the change. Especially when lines have

been deleted.

When the |:keepjumps| command modifier is used the position of a change is not

remembered.

\*:changes\*

:changes Print the change list. A ">" character indicates the

current position. Just after a change it is below the

newest entry, indicating that "g;" takes you to the

newest entry position. The first column indicates the

count needed to take you to this position. Example:

change line col text ~

3 9 8 bla bla bla

2 11 57 foo is a bar

1 14 54 the latest changed line

>

The "3g;" command takes you to line 9. Then the

output of ":changes is:

change line col text ~

> 0 9 8 bla bla bla

1 11 57 foo is a bar

2 14 54 the latest changed line

Now you can use "g," to go to line 11 and "2g," to go

to line 14.

==============================================================================

9. Various motions \*various-motions\*

\*%\*

% Find the next item in this line after or under the

cursor and jump to its match. |inclusive| motion.

Items can be:

([{}]) parenthesis or (curly/square) brackets

(this can be changed with the

'matchpairs' option)

/\* \*/ start or end of C-style comment

#if, #ifdef, #else, #elif, #endif

C preprocessor conditionals (when the

cursor is on the # or no ([{

following)

For other items the matchit plugin can be used, see

|matchit-install|. This plugin also helps to skip

matches in comments.

When 'cpoptions' contains "M" |cpo-M| backslashes

before parens and braces are ignored. Without "M" the

number of backslashes matters: an even number doesn't

match with an odd number. Thus in "( \) )" and "\( (

\)" the first and last parenthesis match.

When the '%' character is not present in 'cpoptions'

|cpo-%|, parens and braces inside double quotes are

ignored, unless the number of parens/braces in a line

is uneven and this line and the previous one does not

end in a backslash. '(', '{', '[', ']', '}' and ')'

are also ignored (parens and braces inside single

quotes). Note that this works fine for C, but not for

Perl, where single quotes are used for strings.

Nothing special is done for matches in comments. You

can either use the matchit plugin |matchit-install| or

put quotes around matches.

No count is allowed, {count}% jumps to a line {count}

percentage down the file |N%|. Using '%' on

#if/#else/#endif makes the movement linewise.

\*[(\*

[( go to [count] previous unmatched '('.

|exclusive| motion. {not in Vi}

\*[{\*

[{ go to [count] previous unmatched '{'.

|exclusive| motion. {not in Vi}

\*])\*

]) go to [count] next unmatched ')'.

|exclusive| motion. {not in Vi}

\*]}\*

]} go to [count] next unmatched '}'.

|exclusive| motion. {not in Vi}

The above four commands can be used to go to the start or end of the current

code block. It is like doing "%" on the '(', ')', '{' or '}' at the other

end of the code block, but you can do this from anywhere in the code block.

Very useful for C programs. Example: When standing on "case x:", "[{" will

bring you back to the switch statement.

\*]m\*

]m Go to [count] next start of a method (for Java or

similar structured language). When not before the

start of a method, jump to the start or end of the

class. When no '{' is found after the cursor, this is

an error. |exclusive| motion. {not in Vi}

\*]M\*

]M Go to [count] next end of a method (for Java or

similar structured language). When not before the end

of a method, jump to the start or end of the class.

When no '}' is found after the cursor, this is an

error. |exclusive| motion. {not in Vi}

\*[m\*

[m Go to [count] previous start of a method (for Java or

similar structured language). When not after the

start of a method, jump to the start or end of the

class. When no '{' is found before the cursor this is

an error. |exclusive| motion. {not in Vi}

\*[M\*

[M Go to [count] previous end of a method (for Java or

similar structured language). When not after the

end of a method, jump to the start or end of the

class. When no '}' is found before the cursor this is

an error. |exclusive| motion. {not in Vi}

The above two commands assume that the file contains a class with methods.

The class definition is surrounded in '{' and '}'. Each method in the class

is also surrounded with '{' and '}'. This applies to the Java language. The

file looks like this: >

// comment

class foo {

int method\_one() {

body\_one();

}

int method\_two() {

body\_two();

}

}

Starting with the cursor on "body\_two()", using "[m" will jump to the '{' at

the start of "method\_two()" (obviously this is much more useful when the

method is long!). Using "2[m" will jump to the start of "method\_one()".

Using "3[m" will jump to the start of the class.

\*[#\*

[# go to [count] previous unmatched "#if" or "#else".

|exclusive| motion. {not in Vi}

\*]#\*

]# go to [count] next unmatched "#else" or "#endif".

|exclusive| motion. {not in Vi}

These two commands work in C programs that contain #if/#else/#endif

constructs. It brings you to the start or end of the #if/#else/#endif where

the current line is included. You can then use "%" to go to the matching line.

\*[star\* \*[/\*

[\* or [/ go to [count] previous start of a C comment "/\*".

|exclusive| motion. {not in Vi}

\*]star\* \*]/\*

]\* or ]/ go to [count] next end of a C comment "\*/".

|exclusive| motion. {not in Vi}

\*H\*

H To line [count] from top (Home) of window (default:

first line on the window) on the first non-blank

character |linewise|. See also 'startofline' option.

Cursor is adjusted for 'scrolloff' option.

\*M\*

M To Middle line of window, on the first non-blank

character |linewise|. See also 'startofline' option.

\*L\*

L To line [count] from bottom of window (default: Last

line on the window) on the first non-blank character

|linewise|. See also 'startofline' option.

Cursor is adjusted for 'scrolloff' option.

<LeftMouse> Moves to the position on the screen where the mouse

click is |exclusive|. See also |<LeftMouse>|. If the

position is in a status line, that window is made the

active window and the cursor is not moved. {not in Vi}