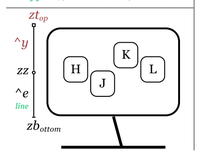
Greyes

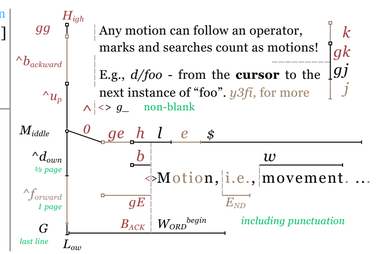
nothing left ⇕ nothing went right

Cursor and Screen

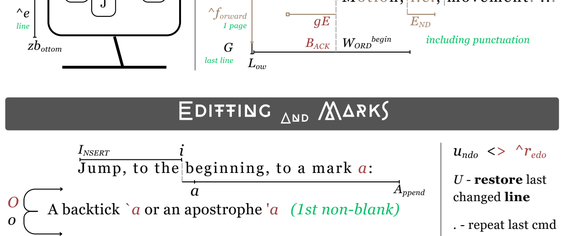
:h_{elp} operator :*h* navigation
[operator][count][motion]

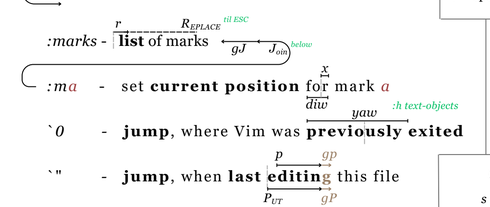
:[range][operator][count]
: $, . or % search or *d* or *y*
end of file $; *current line* .; *all lines* %





Editting and Marks





u_ndo <> ^*r_edo*
U - **restore** last changed **line**
. - repeat last cmd

cc_hange
c$ == *C*
dd_elete
d$ == *D*
yy_ank
y$ == *Y*

y`a
s == *cl*
S_UBSTITUTE == *cc*

Ctags

^]	Goto definition	Jump back up the tag-list	^t
:tp,n	Jump to p_rev, n_ext matching tag	Tag-list matching <tag-name>	:ts <...>
:ju_mps	List of jumps	List of changes	:changes
^i	Goto newer position in jump-list	Goto newer pos in change-list	g,
^o	Goto older position in jump-list	Goto older pos in change-list	g;

Visual Mode

a block *w* word
\+ (, { (), {}
i inner *t* <>

Marking text
o - goto other end of marked area
O - goto other corner of block

<, > shift text left, right

~ switch case
U, u upper, lowercase

Insert Mode

^u,w	Delete line, word before the cursor
^d,t	De-Indent *(left, right)* line one shiftwidth
^p,n	Insert (auto-complete) prev, next match before the cursor

Indent

<<, >>
x== - re-indent *x* lines

<%, >at
gg=*G* - re-indent entire buffer

Searching

p_{rev}	n_{ext}	$b_{ackward}$	f_{orward}	matches
cp	*cn*		:*vim*$_{grep}$ /*x*/ {files} :*cope*$_n$	*x*
			:*vim* /greyes/ **/*.c :*ccl*$_{ose}$	greyes
N	*n*	?*x*	/[*v*$_{ery\_magic}$] *x*	*x*
		#	*	word under cursor
,	;	*Tx*	*tx*	till *(up to)* *x*
		Fx	*fx*	find *x*

Searching

:[range]s/*x*$_{old}$/*y*$_{new}$/[flags] [count] - replace *x*$_{old}$ with *y*$_{new}$
- *x*$_{old}$ or *y*$_{new}$ contains / → use diff char as delimiter
- \n → _.
- *g* - every match on a line; *c* - confirmations :*h s_flags*

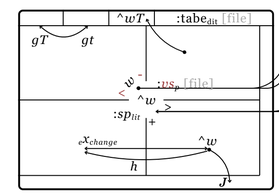
:g$_{lobal}$[*!*]/*x*/*d* - delete all lines [not] containing pattern *x*

:noh$_{lsearch}$ - **remove highlighting** of search matches
% - move to matching char in some *(:h)* **matchpairs**
{, } - move to prev, next para./func./block
gd - move to **local** declaration
gD - move to **global** declaration

r_egister
qr - record macro *r*
q - stop recording
@r - run macro *r*
@@ - rerun last run

:*w !sudo tee %*
w but sudo
:q$_{uit}$a$_{ll}$
:ter$_{minal}$
xgg == *xG* == :*x*

Buffers and Tabs



xgt	Goto tab number *x*
:*b x* or *file*	Goto a b$_{uffer}$ by index *x* or *file*
:*tabc*	c$_{lose}$ current tab, its windows
:*bd*	c$_{lose}$ a file (d$_{elete}$ a buffer)
:*tabdo* cmd	Run command on all tabs
:*bufdo* cmd	Run cmd in each open file

:*tabm$_{ove}$ x* - indexed from 0
:[*tab*]*on$_{ly}$* - close but current

:*ls* == :*buffers* - list all open buffers
^*w*= - make all windows equal size

Registers are Clipboards

All commands that *d$_{elete}$*, *y$_{ank}$*, *p$_{ut}$* text use *r$_{egisters}$*
Type *r* before a command to change which *r* is used
Think of 1st " as a short way of saying "*r$_{egister}$*"

""	Default, unnamed. ""*dd* or *dd*	Expression	"=
"/	Last pattern you searched for	Last command-line	":
"_	Delete w/o clobbering any *r*	Last text you *d* within a single line	"-
"0	Last text you *y*	Sys. clipboard RW *(OS integration)*	"+
"1	Last line(s) you *(big)* *d*	Big *d* stack, "*x* is pushed to "*x+1*	"2 - "9
"a - "z	26 *r*$_s$ for you to play with	Append rather than overwrite	"A - "Z
:reg$_{isters}$	View all current *r*$_s$	Access *r* as a variable	:echo @*r*

v, V, ^v
→ *I*
gv
gf
:r
"

Running

`# gdb <program> [core dump]`
Start GDB (with optional core dump).

`# gdb --args <program> <args…>`
Start GDB and pass arguments

`# gdb --pid <pid>`
Start GDB and attach to process.

`set args <args...>`
Set arguments to pass to program to be debugged.

`run`
Run the program to be debugged.

`kill`
Kill the running program.

Breakpoints

`break <where>`
Set a new breakpoint.

`delete <breakpoint#>`
Remove a breakpoint.

`clear`
Delete all breakpoints.

`enable <breakpoint#>`
Enable a disabled breakpoint.

`disable <breakpoint#>`
Disable a breakpoint.

Watchpoints

`watch <where>`
Set a new watchpoint.

`delete/enable/disable <watchpoint#>`
Like breakpoints.

<where>

function_name
Break/watch the named function.

line_number
Break/watch the line number in the current source file.

file:line_number
Break/watch the line number in the named source file.

Conditions

`break/watch <where> if <condition>`
Break/watch at the given location if the condition is met.
Conditions may be almost any C expression that evaluate to true or false.

`condition <breakpoint#> <condition>`
Set/change the condition of an existing break- or watchpoint.

Examining the stack

`backtrace`
`where`
Show call stack.

`backtrace full`
`where full`
Show call stack, also print the local variables in each frame.

`frame <frame#>`
Select the stack frame to operate on.

Stepping

`step`
Go to next instruction (source line), diving into function.

`next`
Go to next instruction (source line) but don't dive into functions.

`finish`
Continue until the current function returns.

`continue`
Continue normal execution.

Variables and memory

`print/format <what>`
Print content of variable/memory location/register.

`display/format <what>`
Like „print“, but print the information after each stepping instruction.

`undisplay <display#>`
Remove the „display“ with the given number.

`enable display <display#>`
`disable display <display#>`
En- or disable the „display“ with the given number.

`x/nfu <address>`
Print memory.
n: How many units to print (default 1).
f: Format character (like „print“).
u: Unit.

Unit is one of:

b: Byte,
h: Half-word (two bytes)
w: Word (four bytes)
g: Giant word (eight bytes)).

Format

`a` Pointer.
`c` Read as integer, print as character.
`d` Integer, signed decimal.
`f` Floating point number.
`o` Integer, print as octal.
`s` Try to treat as C string.
`t` Integer, print as binary (*t* = „two“).
`u` Integer, unsigned decimal.
`x` Integer, print as hexadecimal.

<what>

expression
Almost any C expression, including function calls (must be prefixed with a cast to tell GDB the return value type).

file_name`::`*variable_name*
Content of the variable defined in the named file (static variables).

function`::`*variable_name*
Content of the variable defined in the named function (if on the stack).

{type}address
Content at *address*, interpreted as being of the C type *type*.

$*register*
Content of named register. Interesting registers are $esp (stack pointer), $ebp (frame pointer) and $eip (instruction pointer).

Threads

`thread <thread#>`
Chose thread to operate on.

Manipulating the program

`set var <variable_name>=<value>`
Change the content of a variable to the given value.

`return <expression>`
Force the current function to return immediately, passing the given value.

Sources

`directory <directory>`
Add *directory* to the list of directories that is searched for sources.

`list`
`list <filename>:<function>`
`list <filename>:<line_number>`
`list <first>,<last>`
Shows the current or given source context. The *filename* may be omitted. If *last* is omitted the context starting at *start* is printed instead of centered around it.

`set listsize <count>`
Set how many lines to show in „list“.

Signals

`handle <signal> <options>`
Set how to handle signles. Options are:

(no)print: (Don't) print a message when signals occurs.

(no)stop: (Don't) stop the program when signals occurs.

(no)pass: (Don't) pass the signal to the program.

Informations

`disassemble`
`disassemble <where>`
Disassemble the current function or given location.

`info args`
Print the arguments to the function of the current stack frame.

`info breakpoints`
Print informations about the break- and watchpoints.

`info display`
Print informations about the „displays“.

`info locals`
Print the local variables in the currently selected stack frame.

`info sharedlibrary`
List loaded shared libraries.

`info signals`
List all signals and how they are currently handled.

`info threads`
List all threads.

`show directories`
Print all directories in which GDB searches for source files.

`show listsize`
Print how many are shown in the „list“ command.

`whatis variable_name`
Print type of named variable.