



Vim scripting cheatsheet

- Proudly sponsored by -

Segment Send data to any tool without having to implement a new API every time.

ethical ad by CodeFund

Start hacking

Learn by example

```
let name = "John"
echo "Hello, " . name
```

You can either put this in a script (script.vim) and run it (:source script.vim), or you can as :let and :echo.

```
function! SuperTab()
  let l:part = strpart(getline('.'), col('.')
  if (l:part =~ '^\\W\\?$')
    return "\\<Tab>"
  else
    return "\\<C-n>"
  endif
endfunction
```

```
imap <Tab> <C-R>=SuperTab(<CR>)
```

Here's another example with [functions](#), [variables](#) and

Variables

Defining

```
let var = "hello"
```

Variable prefixes

```
let g:ack_options = '-s -H'    " g: global  
let s:ack_program = 'ack'      " s: local  
let l:foo = 'bar'             " l: local
```

Other prefixes

```
let w:foo = 'bar'    " w: window  
let b:state = 'on'   " b: buffer  
let t:state = 'off'  " t: tab  
echo v:var           " v: vim special
```

```
let @/ = ''          " @ register (this clears last search pattern)  
echo $PATH           " $ env
```

Vim options

```
echo 'tabstop is ' . &tabstop  
if &insertmode  
echo &g:option  
echo &l:option
```

Operators

```
a + b                " numbers only!  
'hello ' . name     " concat
```

```
let var -= 2  
let var += 5  
let var .= 'string'  " concat
```

Prefix Vim options with &

Strings

Strings

String functions

```

let str = "String"
let str = "String with \n newline"

let literal = 'literal, no \ escaping'
let literal = 'that''s enough' " double '' => '

echo "result = " . re " concatenation

```

Also see :help literal-string and :help expr-quote. See: Strings

```

strlen(str) " length
len(str) " same
strchars(str) " character length

split("one two three") "=> ['one',
split("one.two.three", '.') "=> ['one',

join(['a', 'b'], ',') "=> 'a,b'

tolower('Hello')
toupper('Hello')

```

Also see :help functions See: String functions

Functions

Functions

```

" prefix with s: for local script-only functions
function! s:Initialize(cmd, args)
  " a: prefix for arguments
  echo "Command: " . a:cmd

  return true
endfunction

```

See: Functions

Namespacing

```
function! myplugin#hello()
```

Calling functions

```

call s:Initialize()
call s:Initialize("hello")

```

Consuming return values

Abortable

```
function! myfunction() abort
endfunction
```

Aborts when an error occurs.

```
echo "Result: " . s:Initialize()
```

Var arguments

```
function! infect(...)
  echo a:0      "=> 2
  echo a:1      "=> jake
  echo a:2      "=> bella

  for s in a:000 " a list
    echon ' ' . s
  endfor
endfunction

infect('jake', 'bella')
```

See :help function-argument. See: Var argumen

Loops

```
for s in list
  echo s
  continue " jump to start of loop
  break    " breaks out of a loop
endfor
```

```
while x < 5
endwhile
```

Custom commands

Custom commands

```
command! Save :set fo=want tw=80 nowrap
```

Custom commands start with uppercase letters. The ! redefines a command if it already exists

Commands calling functions

```
command! Save call <SID>foo()
```

```
function! s:foo()  
...  
endfunction
```

Commands with arguments

```
command! -nargs=? Save call script#foo(<args>)
```

-nargs=0	0 arguments, default
-nargs=1	1 argument, includes spaces
-nargs=?	0 or 1 argument
-nargs=*	0+ arguments, space separated
-nargs=+	1+ arguments, space separated

Flow

Conditionals

```
let char = getchar()
if char == "\<LeftMouse>"
    " ..."
elseif char == "\<RightMouse>"
    " ..."
else
    " ..."
endif
```

Truthiness

```
if 1 | echo "true" | endif
if 0 | echo "false" | endif
```

```
if 1      "=> 1 (true)
if 0      "=> 0 (false)
if "1"    "=> 1 (true)
if "456"  "=> 1 (true)
if "xfz"  "=> 0 (false)
```

Operators

```
if 3 > 2
if a && b
if (a && b) || (c && d)
if !c
```

See :help expression-syntax. See: [Operators](#)

No booleans. 0 is false, 1 is true. See: [Truthiness](#)

Strings

```
if name ==# 'John'      " case-sensitive
if name ==? 'John'      " case-insensitive
if name == 'John'       " depends on :set :
```

" also: is#, is?, >=#, >=?, and so on

Identity operators

```
a is b
a isnot b
```

Checks if it's the same instance object.

Regex matches

```
"hello" =~ '/x/'
"hello" !~ '/x/'
```

Boolean logic

Single line

```
if g:use_dispatch && s:has_dispatch
...
endif
```

```
if g:use_dispatch && s:has_dispatch
...
endif
```

Lists

Lists

Functions

```
let mylist = [1, two, 3, "four"]

let first = mylist[0]
let last  = mylist[-1]

" Suppresses errors
let second = get(mylist, 1)
let second = get(mylist, 1, "NONE")
```

```
len(mylist)
empty(mylist)

sort(list)
let sortedlist = sort(copy(list))

split('hello there world', ' ')
```

Sublists

Concatenation

```
let shortlist = mylist[2:-1]
let shortlist = mylist[2:]    " same

let shortlist = mylist[2:2]    " one item
```

Push

Map

```
call map(files, "bufname(v:val)") " use v:val for value
call filter(files, 'v:val != ""')
```

```
let alist = [1, 2, 3]
let alist = add(alist, 4)
```

Dictionaries

Dictionaries

```
let colors = {
  \ "apple": "red",
  \ "banana": "yellow"
}

echo colors["a"]
echo get(colors, "apple") " suppress error
```

See :help dict

Using dictionaries

```
remove(colors, "apple")
```

```
" :help E715
if has_key(dict, 'foo')
if empty(dict)
keys(dict)
len(dict)
```

```
max(dict)
min(dict)
```

```
count(dict, 'x')
```

Iteration

```
for key in keys(mydict)
  echo key . ': ' . mydict[key]
endfor
```

Extending

Prefixes


```
" Extending with more
let extend(s:fruits, { ... })
```

Prefixes (s:, g:, l:, etc) are actually dictionaries.

Casting

```
str2float("2.3")
str2nr("3")
float2nr("3.14")
```

Numbers

Numbers

```
let int = 1000
let int = 0xff
let int = 0755 " octal
```

See :help Number. See: [Numbers](#)

Floats

```
let f1 = 100.1
let f1 = 5.4e4
```

See :help Float

Math functions

Arithmetic

```
3 / 2      "=> 1, integer division
3 / 2.0    "=> 1.5
```

```
sqrt(100)
floor(3.5)
ceil(3.3)
abs(-3.4)

sin() cos() tan()
sinh() cosh() tanh()
asin() acos() atan()
```

Vim-isms

Execute a command

```
execute "vsplit"
execute "e " . fnameescape(filename)
```

Runs an ex command you typically run with `:`. Also see `:help execute`. See: [Execute a command](#)

Running keystrokes

```
normal G
normal! G " skips key mappings

execute "normal! gg/foo\<cr>dd"
```

Use `:normal` to execute keystrokes as if you're typing them. See `:help normal`.
keystrokes

Getting filenames

```
echo expand("%")      " path/file.txt
echo expand("%:t")    " file.txt
echo expand("%:p:h")  " /home/you/path/file.txt
echo expand("%:r")    " path/file
echo expand("%:e")    " txt
```

See `:help expand`

Silencing

```
silent g/Aap/p
```

Suppresses output. See `:help silent`

Echo

Settings

```
set number
set nonumber
set number! " toggle
set numberwidth=5
set guioptions+=e
```

```
echoerr 'oh it failed'
echomsg 'hello there'
```

Prompts

Built-ins

```
has("feature") " :h feature-list
executable("python")
globpath(&rtp, "syntax/c.vim")

exists("$ENV")
exists(":command")
exists("variable")
exists("+option")
exists("g:...")
```

```
let result = confirm("Sure?")
```

Mapping

Mapping commands

Explanation

Arguments

```
nmap
vmap
imap
xmap
```

[nvixso](nore)map

<buffer>

<silent>

```
nnoremap
vnoremap
inoremap
xnoremap
...
```

```
|      L don't recurse
|
|      L normal, visual, insert,
|      eX mode, select, operator-pending
```

```
<nowait>
```

Syntax

Highlights

```
hi Comment
  term=bold,underline
  gui=bold
  ctermfg=4
  guifg=#80a0ff
```

Filetype detection

```
augroup filetypedetect
  au! BufNewFile,BufRead *.json setf java
augroup END

au Filetype markdown setlocal spell
```

Conceal

```
set conceallevel=2
syn match newLine "<br>" conceal cchar=}
hi newLine guifg=green
```

Region conceal

```
syn region inBold concealends matchgroup=hl
hi inBold gui=bold
hi bTag guifg=blue
```

Syntax

```
syn match :name ":regex" :flags

syn region Comment start="/\*" end="\*/"
```

Include guards

```
if exists('g:loaded_myplugin')
  finish
endif
```

```
syn region String    start=+"+    end=+"+    skip=+\\\"+
```

```
syn cluster :name contains=:n1,:n2,:n3...
```

```
flags:
```

```
    keepend
```

```
    oneline
```

```
    nextgroup=
```

```
    contains=
```


```
    contained
```

```
hi def link markdownH1 htmlH1
```

```
" ...
```

```
let g:loaded_myplugin = 1
```



►  **2 Comments** for this cheatsheet. [Write yours!](#)

Search 381+ cheatsheets



Over 381 curated cheatsheets, by developers for developers.

Devhints home

Other Vim cheatsheets

Vimdiff

cheatsheet 

Vim

cheatsheet 

Tabular

cheatsheet 

Projectionist

cheatsheet

Vim digraphs

cheatsheet

Vim Easyalign

cheatsheet

Top cheatsheets

Elixir

cheatsheet 

ES2015+

cheatsheet 

React.js

cheatsheet 

Vimdiff

cheatsheet 

Vim

cheatsheet 

Vue.js

cheatsheet 