

Comprehensive presentation of Vim features

Aleksander Gajewski

adiog@brainfuck.pl

November 18, 2017

Overview

Vim is a highly customizable, modal editor. It is extensible with built-in vimscript language, python-support and huge collection of plugins.

This presentation is intended to give a general overview of Vim and be a first reference for a novice user to start using its feature efficiently.

For further information please

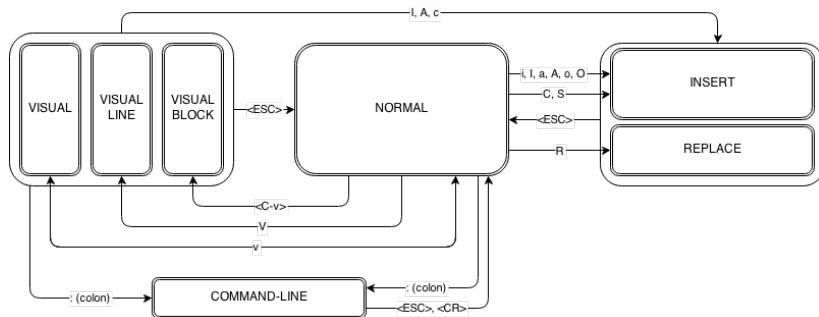
- 1 use vim internal help (:h *topic*)
- 2 run vimtutor
- 3 visit vimcasts.org
- 4 visit learnvimscriptinahardway.com
- 5 visit stackoverflow.com/tag/vim

Vi Improved

The image displays three overlapping screenshots of the Vim text editor, illustrating different operational modes:

- Top Screenshot (Visual Mode):** The title bar reads "[No Name] + - VIM". The command line shows "selecting in visual mode". The status bar at the bottom indicates "VISUAL" and "[No Name][+]".
- Middle Screenshot (Insert Mode):** The title bar reads "[No Name] + - VIM". The command line shows "typing in insert mode". The status bar at the bottom indicates "INSERT" and "[No Name][+]".
- Bottom Screenshot (Normal Mode):** The title bar reads "[No Name] - VIM". The command line shows "1". The status bar at the bottom indicates "NORMAL" and "[No Name]". The main window displays the Vim startup screen, including the version (7.4.160) and a list of help commands.

Modes overview :h vim-modes



Notation

To avoid confusion the following colors will be used to indicate the initial modes of key sequence:

- normal → insert: i, I, a, A, o, O, C, S
- visual → insert: I, A, c
- visual → normal: <ESC>
- insert → normal: <ESC>
- normal → visual: v, V, <C-v>
- normal → command: :
- visual → command: : (':<,'> indicates visual range)
- command → normal: <ESC>

By default command instructions will be marked regardless initial mode (i.e. with leading colon - e.g. :h vim-modes). Also trailing Enter (<CR>) will be mostly omitted.

Notation in examples

- `[range]` - stands for comma-separated range of lines, e.g. `15,25`, or visual range `:'<,'>`, or whole file `%`
- `[optional]` - may stand for an optional argument
- `{Visual}` - may highlight that command can be used on a given selection in visual mode
- `"a`, `"A`, `"O` - use register a, A or O for next delete, yank or put operation
remark #1: register is identified by a single character: a lowercase letter, an uppercase letter or a digit
remark #2: use uppercase character to append with delete and yank
remark #3: use record macro to clear register (i.e. `qaq`)
- `<CR>`, `<Up>`, `<BS>`, `<C-c>`, `<C-v>`, `<S-Left>` - stands for key sequence Enter, Up, Backspace, Ctrl+C, Ctrl+V and Shift+Left respectively

Load/Save/Quit

Basic commands:

- 1 `:e filename` - edit/reload file
- 2 `:w (:w!)` - save file (force overwrite)
- 3 `:q (:q!)` - quit (without saving)
- 4 `:wq` - save and quit
- 5 `:sav filename` - 'save as' file (and open in current window)
- 6 `:%!sudo tee %` - save as root

Useful ones:

- 1 `:[range]w filename` - write range to file
- 2 `:r filename` - insert file below the cursor

Save and quit directly from **normal**: `ZZ`

Basics about moving around

There is a huge boost while not using arrow keys to moving.

- 1 **h** (left), **j** (down), **k** (up), **l** (right)
- 2 **gh** (left), **gj** (down), **gk** (up), **gl** (right) (when wrapped)
- 3 **gg** (begin of file), **G** (end of file)
- 4 **: [linenumber]** - go to line

All motion will be precisely described on further slides.

Boosted typing (completion)

In **insert** mode:

- 1 **<C-p>**, **<C-n>** - previous, next autocomplete suggestion
- 2 **<C-x><C-f>** - autocomplete with filename
- 3 **<C-e>**, **<C-y>** - insert char below/above

In **normal** mode:

- 1 **<C-a>**, **<C-x>** - increment, decrement number

Buffers / Windows / Tabs

Summary:

- 1 A **buffer** is the in-memory text of a file.
- 2 A **window** is a viewport on a buffer.
- 3 A **tab page** is a collection of windows.

Buffers / Windows / Tabs: reference topics

- `:h opening-window` (`:h new`, `:h split`, `:h vsplit`)
- `:h window-move-cursor`
- ■ `<C-w>j` (move left), `<C-w>h` (move down), `<C-w>l` (move up), `C-w k` (move right)
 - `<C-w><C-w>` (move next)
- `:h window-moving`
- `:h window-resize`
- `:h buffer-list`
- `:h list-repeat` (`:h bufdo`, `:h windo`, `:h tabdo`)

Copying and Moving Text

- 1 **"{a-zA-Z0-9}** - use register {a-zA-Z0-9} for next delete, yank or put (use uppercase character to append with delete and yank)
- 2 **:reg[isters]** - display the contents of all registers
- 3 **["x]y{motion}** - yank {motion} text [into register x]
- 4 **["x]yy** - yank [count] lines [into register x]
- 5 **["x]Y** - yank [count] lines [into register x] (synonym for yy)
- 6 **{Visual}["x]y** - yank the highlighted text [into register x]
- 7 **{Visual}["x]Y** - yank the highlighted lines [into register x]
- 8 **:[range]y[ank] [x]** - yank [range] lines [into register x].
- 9 **["x]p, ["x]P** - put the text [from register x] after (for P before)
- 10 **["x]gp, ["x]gP** - just like "p" ("P"), but leave the cursor just after the new text.
- 11 **:[line]pu[t] [x], :[line]pu[t]! [x]** - put the text [from register x] after (before) [line] (default current line).

Undo/Redo/Repeat features

- 1 `[count]u` - undo `[count]` changes
 - 2 `:u[ndo]` - undo one change
 - 3 `[count]C-R` - redo `[count]` changes which were undone
 - 4 `:red[o]` - redo one change which was undone
 - 5 `U` - undo all latest changes on one line.
- . - repeat last change, with count replaced with `[count]`.

Operators and motions

In normal mode (`{operator}[count]{motion}`):

- 1 - enter an operation (e.g. delete, change, yank, cut)
- 2 - provide optional `[count]` argument
- 3 - use a motion (to indicate the text region)

Or in visual mode (`[count]{motion}{operator}`):

- 1 - provide optional `[count]` argument
- 2 - enter a motion
- 3 - use an operation

1. Operators

- 1 c - change
- 2 d - delete
- 3 y - yank into register (does not change the text)
- 4 x - cut into register
- 5 gu - make lowercase
- 6 gU - make uppercase
- 7 ! - filter through an external program
- 8 = - filter through equalprg or C-indenting if empty
- 9 gq - text formatting
- 10 g? - ROT13 encoding aka boss mode

2. Left-right motions :h left-right-motions

- 1 h, <Left>, <C-h>, <BS> - [count] characters to the left
- 2 l, <Right>, <Space> - [count] characters to the right
- 3 O, <Home> - to the first character of the line
- 4 f{char} - to [count]th occurrence of {char} to the right. The cursor is placed on {char} inclusive
- 5 F{char} - to the [count]'th occurrence of {char} to the left. The cursor is placed on {char} exclusive.
- 6 t{char} - till before [count]'th occurrence of {char} to the right. The cursor is placed on the character left of {char} —inclusive—
- 7 T{char} - till after [count]'th occurrence of {char} to the left. The cursor is placed on the character right of {char} exclusive.
- 8 ; - repeat latest f, t, F or T [count] times
- 9 , - repeat latest f, t, F or T in opposite direction [count] times

3. Up-down motions :h up-down-motions

- k, <Up>, C-p - lines upward —linewise—
- j, <Down>, C-j, C-n - lines downward —linewise—
- gk, g<Up> - display lines upward. not linewise
- gj, g<Down> - display lines downward. —exclusive— motion.
- G - goto line, default first line.
- <C-End> - goto line [count], default last line.
- <C-Home>, gg - goto line [count], default first line, on the first non-blank character —linewise—
- :[range] - set the cursor on the last line number in [range].
- N% - go to {count} percentage in the file

4. Word motions :h word-motions

- w - [count] words forward. —exclusive— motion.
- W, <C-Right> - [count] WORDS forward. —exclusive— motion.
- e - Forward to the end of word [count] —inclusive—.
- E - Forward to the end of WORD [count] —inclusive—.
- <S-Left>, b - [count] words backward. —exclusive— motion.
- <C-Left>, B - [count] WORDS backward. —exclusive— motion.
- ge - Backward to the end of word [count] —inclusive—.
- gE - Backward to the end of WORD [count] —inclusive—.

5. Text object motions :h 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.

6. Text object selection :h object-select :h text-objects

- aw - "a word", select [count] words (see ((*word*)))
- aW - "a WORD", select [count] WORDs (see ((*WORD*)))
- as - "a sentence", select [count] sentences (see ((*sentence*)))
- ap - "a paragraph", select [count] paragraphs (see ((*paragraph*)))
- a], a[- "a [] block", select [count] '[' ']' blocks
- a), a(, ab - "a block", select [count] blocks
- a>, a< - "a <> block", select [count] <> blocks
- at - "a tag block", select [count] tag blocks
- a", a', a' - "a quoted string"

6. Text object selectionv :h object-select :h text-objects

(inner variants)

- `iw` - "inner word", select [count] words (`*word*`)
- `iW` - "inner WORD", select [count] WORDs (see `((*WORD*))`)
- `is` - "inner sentence", select [count] sentences (see `((*sentence*))`)
- `ip` - "inner paragraph", select [count] paragraphs (see `((*paragraph*))`)
- `i]`, `i[` - "inner [] block", select [count] '[' ']' blocks
- `i)`, `i(`, `ib` - "inner block", select [count] blocks
- `i<`, `i>` - "inner <>block", select [count] <>blocks
- `it` - "inner tag block", select [count] tag blocks
- `i"`, `i'`, `i'` - Like `a"`, `a'` and `a'`, but exclude the quotes

7. Marks :h mark-motions

- `m{a-zA-Z}` - set mark {a-zA-Z} at cursor position (does not move the cursor, this is not a motion command).
- ``{a-z}`, ``{a-z}` - jump to the mark {a-z} in the current buffer.
- `#{A-Z0-9}`, `#{A-Z0-9}` - jump to the mark {A-Z0-9} in the file where it was set

Jump to last marks:

- ```` - jump to the specified location (motion is exclusive)
- `''` - jump to the first non-blank character in the line of the specified location (motion is linewise)

Macros

- 1 `qa` - start recording to register `a`
 - 2 series of commands to be recorded
 - 3 `q` stop recording (being in a normal mode)
- `@a` - execute macro (from register `a`)
 - `@@` - execute macro again
 - `:g/pattern/normal @a` - execute macro for given range

vimdiff - Use vim to compare files

- 1 Command line interface (maybe used together with git/svn)

```
$ vimdiff file1 file2
```

- 2 Start comparing within vim:

- `:diffthis` → `:sp file2` → `:diffthis`
- `:diffeoff` - turn off comparison on a file
- `:[range]diffget` - insert from other file
- `:[range]diffput` - insert into other file

`/pattern<CR>` (start search) → `/<CR>` (continue search)
`(:[range]/pattern` - to use within range)

- regex rules:

- `\(- \)` - groups
- `.` - any character
- `*` - any number of repetition (0+)
- `?` - optional (0 or 1)
- `+` - positive number of repetition (1+)
- `\s, \d, \w, \a` - whitespace, digit, word, alpha characters
- `^, $` - begin of line, end of outline
- `\<, \>` - begin, end of word
- `[^XYZ]` - exclude characters XYZ

- `:set hlsearch` - enable search result highlighting
- `:set incsearch` - search while typing (incremental)
- If needed: `:nohl` to disable highlighting afterwards

Power of g (global command)

`: [range] g/pattern/cmd`

- Delete all lines matching a pattern: `:g/pattern/d`
- Delete all lines that do not match a pattern: `:g!/pattern/d`
or `:v/pattern/d`
- Delete all blank lines: `:g/^\s*$/d`
- Copy all lines matching a pattern to end of file:
`:g/pattern/t$`
- Move all lines matching a pattern to end of file:
`:g/pattern/m$`
- Copy all lines matching a pattern to register 'a'.
`qaq:g/pattern/y A` (clearing register first)
- Run a macro on matching lines (example assuming a macro recorded as 'q'): `:g/pattern/normal @q`
- Call custom command on match `:g/pattern/\=call foo()`

Search and replace substitutions

Common syntax (allows custom separators e.g. `/`, `#`, `$`, `,`):

- `:%s/pattern/replace/commands`
- `:%s,pattern,replace,commands`

Where

- pattern supports regex (only grouping needs escaping):
`\(, \), ., *, ?, +, [^X]`
- in replace use groups with `\1`, `\2`, ... or whole match `&`
- commands: `g` (global), `c` (confirm), `i` (ignore case)
- call custom command on match `\=call foo()`
- use nested replace `\=substitute()` with `submatch()`

Command mode

- know how to work with `ctrl+r`
- use the marked region as default substitute pattern

- easy way to manage
- clean and readable vimrc
- rule of thumb: extract your config and store it as a plugin on github

nerdtree, vim-nerdtree-tab, nerdtree-toggle, vim-rooter

- Navigate through filesystem,
- Allows open, open split h/v, open tab, quick preview,
- Support for bookmarks.
- :NERDTreeFind
- plugin nerdtree-tabs keeps nerd tree toggle accross all tabs

```
" Press ? for help
>-----Bookmarks-----
>AST <g/Build/llvm/tools/clang/lib/AST/
>ASTMatchers <ls/clang/lib/ASTMatchers/
>clang-check </clang/tools/clang-check/
.. (up a dir)
/home/adiog/Build/llvm/tools/clang/
├─ .git/
├─ bindings/
├─ cmake/
│   ├── caches/
│   └─ modules/
│       ├── AddClang.cmake
│       ├── ClangConfig.cmake.in
│       └─ CMakeLists.txt
├─ docs/
├─ examples/
├─ include/
├─ INPUTS/
├─ lib/
├─ runtime/
├─ test/
├─ tools/
│   ├── arcmt-test/
│   ├── c-arcmt-test/
│   └─ c-index-test/
└─
```

```
1 # This file allows users to call find_package(Clang) and pick up our targets.
2
3 find_package(LLVM REQUIRED CONFIG)
4
5 @CLANG_CONFIG_CODE@
6
7 set(CLANG_EXPORTED_TARGETS "@CLANG_EXPORTS@")
8 set(CLANG_CMAKE_DIR "@CLANG_CONFIG_CMAKE_DIR")
9
10 # Provide all our library targets to users.
11 include("@CLANG_CONFIG_EXPORTS_FILE@")
```

```
NERD > <ld/llvm/tools/clang/cmake/modules/ClangConfig.cmake.in cma.. 100% : 11/11 : 13
```

works ok with `-system-clang` (still recommend `clion` as default)
have not checked carefully the `rope` variant, but it seemed also ok
(still recommend `pycharm`)

```

49 unsigned ASTContext::NumImplicitCopyConstructorsDeclared;
50 unsigned ASTContext::NumImplicitMoveConstructors;
51 unsigned ASTContext::NumImplicitMoveConstructorsDeclared;
52 unsigned ASTContext::NumImplicitCopyAssignmentOperators;
53 unsigned ASTContext::NumImplicitCopyAssignmentOperatorsDeclared;
54 unsigned ASTContext::NumImplicitMoveAssignmentOperators;
55 unsigned ASTContext::NumImplicitMoveAssignmentOperatorsDeclared;
56 unsigned ASTContext::NumImplicitDestructors;
57 unsigned ASTContext::NumImplicitDestructorsDeclared;
58
59 enum FloatingRank {
60     HalfRank, FloatRank, DoubleRank, LongDoubleRank, Float128Rank
61 };
62
63 RawComment *ASTContext::getRowCommentForDeclNoCache(const Decl *D) const {
64     if (!CommentsLoaded && ExternalSource) {
65         ExternalSource->ReadComments();
66     }
67 #ifndef NDEBUG
68     ArrayRef<RawComment *> RawComments = Comments.getComments();
69     assert(std::is_sorted(RawComments.begin(), RawComments.end(),
70         BeforeThanCompare<RawComment*>(SourceMgr)));
71 #endif
72     CommentsLoaded = true;
73 }
74
75 assert(D);
76
NORMAL > master <ext.cpp FloatingRank < cpp < 0% : 59/9377 : 6 < | trailing...

```

* Press <F1>, ? for help

► macros

▼ FloatingRank : enum
[enumerators]
- DoubleRank
- Float128Rank
- FloatRank
- HalfRank
- LongDoubleRank

▼ _anon1* : namespace
[functions]
adjustDeclToTemplate(const Decl *D)

▼ _anon2* : namespace
▼ ParentMapASTVisitor : class
[typedefs]
- VisitorBase
[functions]
- ParentMapASTVisitor(ASTContext::Pa
- TraverseDecl(Decl *DeclNode)
- TraverseNestedNameSpecifierLoc(Nes
- TraverseNode(T Node, MapNodeTy Map
- TraverseStmt(Stmt *StmtNode)
- TraverseTypeLoc(TypeLoc TypeLocNod
+ buildMap(TranslationUnitDecl &TU)

Tagbar Name ASTContext.cpp

visualize (with diff) and navigate through the history of file

```

" Gundo for README.txt (7)
" j/k - move between undo states
" p - preview diff of selected and current
" <cr> - revert to selected state

@ [7] 2 seconds ago
|
| o [6] 24 seconds ago
|
| o [5] 37 seconds ago
|
| o [4] 54 seconds ago
|
| o [3] 93 seconds ago
|/
|
| o [2] 100 seconds ago
|
| o [1] 2 minutes ago
|
| o [0] Original

Gundo
--- 3 2016-11-11 07:06:29 PM
+++ 7 2016-11-11 07:08:00 PM
@@ -1,5 +1,5 @@
//=====
-// c language family front-end
+// C LANGUAGE FAMILY FRONT-END
//=====

Welcome to Clang. This is a compiler front-

__Gundo_Preview__[-] 11% : 1/9 : 1 | <ools/clang/README.txt tex... 157 words < 26% : 7/26 : 33

```

changel

git mark modified lines (useful)

git facilities

tabular

```
| column | column|  
row | value | value|  
other row | other value | yet another value |
```

```
:%Tabularize /|
```

```
      | column      | column      |  
row    | value    | value    |  
other row | other value | yet another value |
```


useful to trigger external commands

abolish

Abbreviations and hiding text

- `:iab tte text_to_expand` - define 'tte' abbreviation
- Conceal known chars outside currently edited line:

```
∫ ½λ ∂λ                                % conceal  
\int \frac{1}{2}\lambda \partial\lambda % current line  
∫ ½λ ∂λ                                % conceal
```

Below sample config:

```
if has('conceal')  
  let g:tex_conceal="adgms"  
  set conceallevel=2  
  highlight! link Conceal texMathSymbol  
endi
```

```
syn match texMathSymbol '\\\alpha\>' contained conceal cchar=A  
syn match texMathSymbol '\\\beta\>' contained conceal cchar=B  
syn match texMathSymbol '\\\gamma\>' contained conceal cchar=G
```


Expressions register

In `insert` mode:

1 `<C-r>=`

2 enter VimScript command to be evaluated

This can be used as a in place calculator, e.g. on line:

6 * 7 =

the following sequence may be used to append a result:

`Oyt=A<C-r>=<C-r>>a<CR>`

tricks:i

- rotate over all color schemes
- pick the name of current text segment
- set the color for custom element
- display the html color format #156123 with real color

Spellchecker

- `:setlocal spell spelllang=en_us` to set language (e.g.)
- `:set spell` to enable spellchecker
- `]s` Move to next misspelled word after the cursor.
- `[s` Like `]s` but search backwards.
- `z=` For the word under/after the cursor suggest correctly spelled words. This also works to find alternatives for a word that is not highlighted as a bad word, e.g., when the word after it is bad.

additional features: add words to dictionary etc.

Folds

- `zf{motion}, {visual}zf` Operator to create a fold. This only works when 'foldmethod' is "manual" or "marker". The new fold will be closed for the "manual" method. 'foldenable' will be set.
- `: [range]fo[ld]` Create a fold for the lines in range. Works like "zf".
- `zd` - delete one fold at the cursor.
- `zo` - open one fold under the cursor.
- `zO` - open all folds under the cursor recursively.
- `zc` - close one fold under the cursor.
- `zC` - close all folds under the cursor recursively.

Other features

- list number
- built in autocomplete / no prob with connecting external tools
- topics to cover:
- context autocomplete
- inplace static source code analyser

Remaining issues

- vimrc / recommended setup / mappings vimscript basic (function declaration / scopes / if / types /)
- (sort)
- os interaction (in place edit)
- external indent

Useful links

- <http://vimcasts.org>
- <http://vimgolf.org>
- <http://vimregex.com>
- <https://github.com/tpope>
- <http://learnvimscriptthehardway.stevelosh.com/>
- <https://github.com/vim/vim>
- <https://github.com/vim-scripts>
- <http://vim.org>
- <https://neovim.io>
- <https://github.com/neovim/neovim>