Comprehensive presentation of Vim features

Aleksander Gajewski

adiog@brainfuck.pl

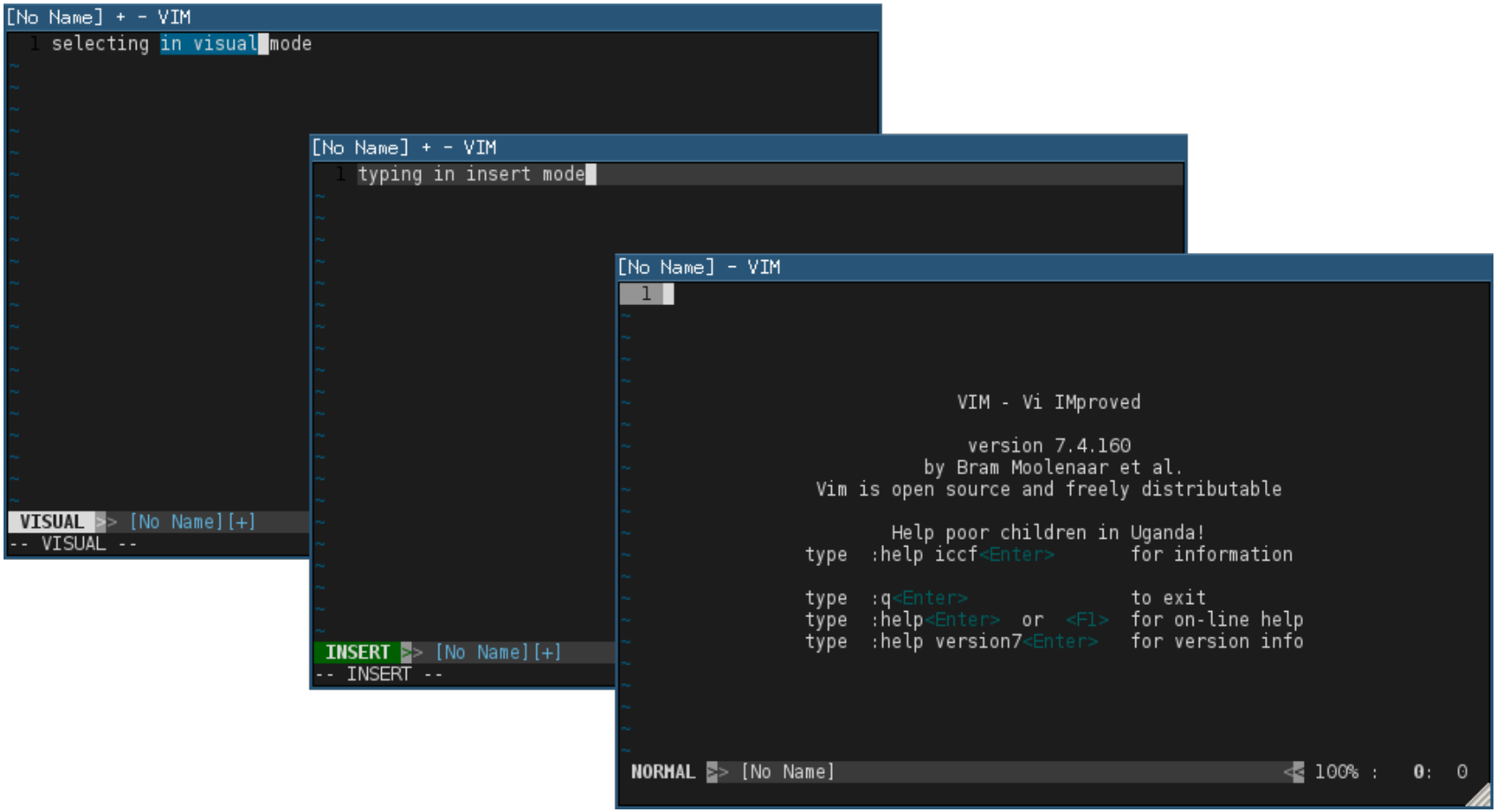
December 9, 2017

Overview

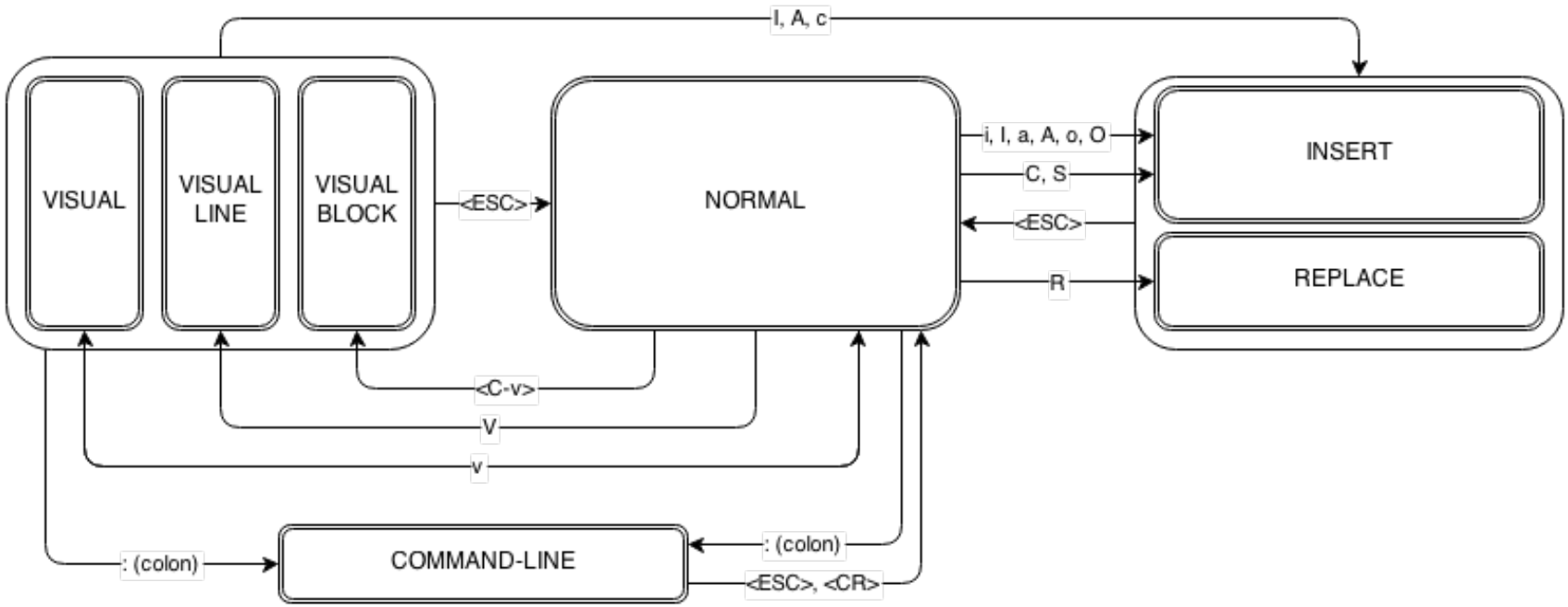
Vim is a highly customizable, modal editor. It is extensible with built-in vimscript language, python-support and huge collection of plugins.
This presentation is intended to give a general overview of Vim and be a first reference for a novice user to start using its feature efficiently.
For futher information 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


[No Name] + - VIM
1 selecting in visual mode
VISUAL >> [No Name][+]
-- VISUAL --
[No Name] + - VIM
1 typing in insert mode
INSERT >> [No Name][+]
-- INSERT --
[No Name] - VIM
1
VIM - Vi IMproved
version 7.4.160
by Bram Moolenaar et al.
Vim is open source and freely distributable
Help poor children in Uganda!
type :help iccf<Enter> for information
type :q<Enter> to exit
type :help<Enter> or <F1> for on-line help
type :help version7<Enter> for version info
NORMAL >> [No Name] << 100% : 0: 0

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`, `"0` - use register `a`, `A` or `0` 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`

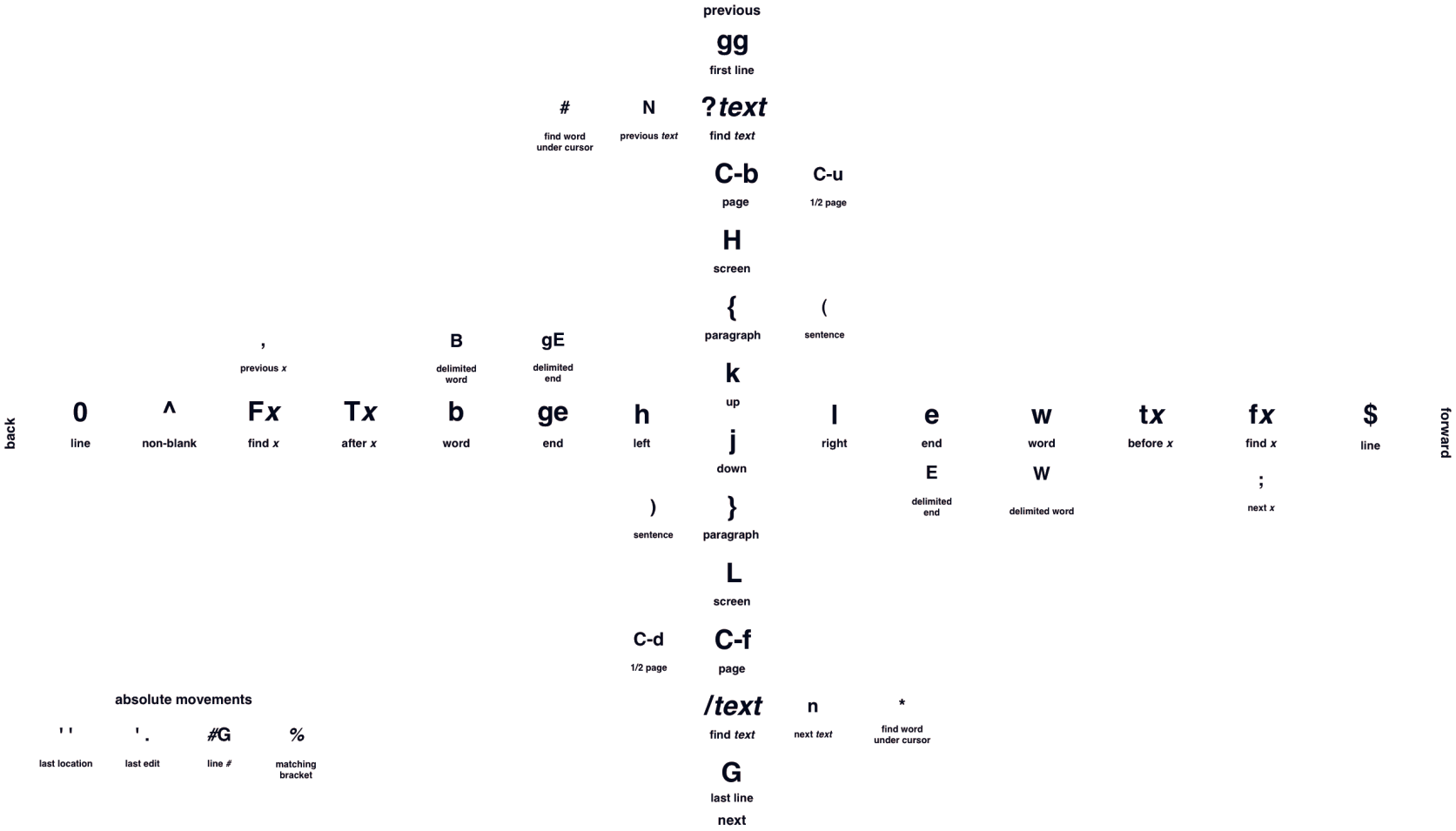
Basics about moving around

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
5. `[percent]%` - go to percent of file

All motions will be precisely described on further slides.

Sample movements



Boosted typing (completition)

In `insert` mode:

1. `<C-p>`, `<C-n>` - previous, next autocomplete suggestion
2. `<C-x><C-f>` - autocomplete filename (based on cwd)
3. `<C-e>`, `<C-y>` - insert (copy) char from below/above line

In `normal` mode:

1. `<C-a>`, `<C-x>` - increment, decrement number (also hex)
   (for operating on dates see tpope/vim-speeddating)

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.

`.` (a dot character) - repeat last change, with count replaced with [count]. (see sjl/gundo.vim plugin for boosted history navigation)

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

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

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

- k, <Up>, C-p - [count] lines upward (linewise)
- j, <Down>, C-j, C-n - [count] lines downward (linewise)
- gk, g<Up> - [count] display-wise lines upward (useful when lines are wrapped)
- gj, g<Down> - [count] display-wise lines downward (useful when lines are wrapped)
- 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
- :[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)
- W, <C-Right> - [count] WORDS forward. (exclusive)
- 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)
- <C-Left>, B - [count] WORDS backward. (exclusive)
- ge - backward to the end of word [count] (inclusive)
- gE - backward to the end of WORD [count] (inclusive)

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

5. Text object motions :h object-motions

:h text-objects

- (- [count] sentences backward (exclusive)
-) - [count] sentences forward (exclusive)
- { - [count] paragraphs backward (exclusive)
- } - [count] paragraphs forward (exclusive)
-]] - [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)
-][- [count] sections forward or to the next '}' in the first column (exclusive)
- [[- [count] sections backward or to the previous '{' in the first column (exclusive)
- [] - [count] sections backward or to the previous '}' in the first column (exclusive)

6. Text object selection :h object-select

- 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

(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`
   - `:diffoff` - turn off comparison on a file
   - `:[range]diffget` - insert from other file
   - `:[range]diffput` - insert into other file

Search

`/pattern<CR>` (start search) → `n` or `/<CR>` - next, `N` - previous
`:[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. `/`, `#`, `$`, `,`):

- `:[range]s/pattern/replace/flags`
- `:s/pattern/replace/flags` - within line
- `:%s/pattern/replace/flags`
- `:%s,pattern,replace,flags`

Where

- pattern supports regex (only grouping needs escaping): `\(`, `\)`, `.`, `*`, `?`, `+`, `[^X]`
- in replace use groups with `\1`, `\2`, ... or whole match `&`
- flags: `g` (global), `c` (confirm), `i` (ignore case)
- call custom command on match `\=call eval_replacement()`
- use nested replace `\=substitute()` with `submatch()`, e.g.: `:s/outer/\=substitute(submatch(0), 'inner', 'replace', 'g')`
- use nested replace with `:g`, e.g.: `:g/good/s/bad/ugly/g`

Shell invocation

- `:read !ls -la` or `:read !date`
- `:%!sort`
- `:let curdate=system('date')`
- backticks `:new ‘date‘`

Command mode

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

vundle

- 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

YouCompleteMe

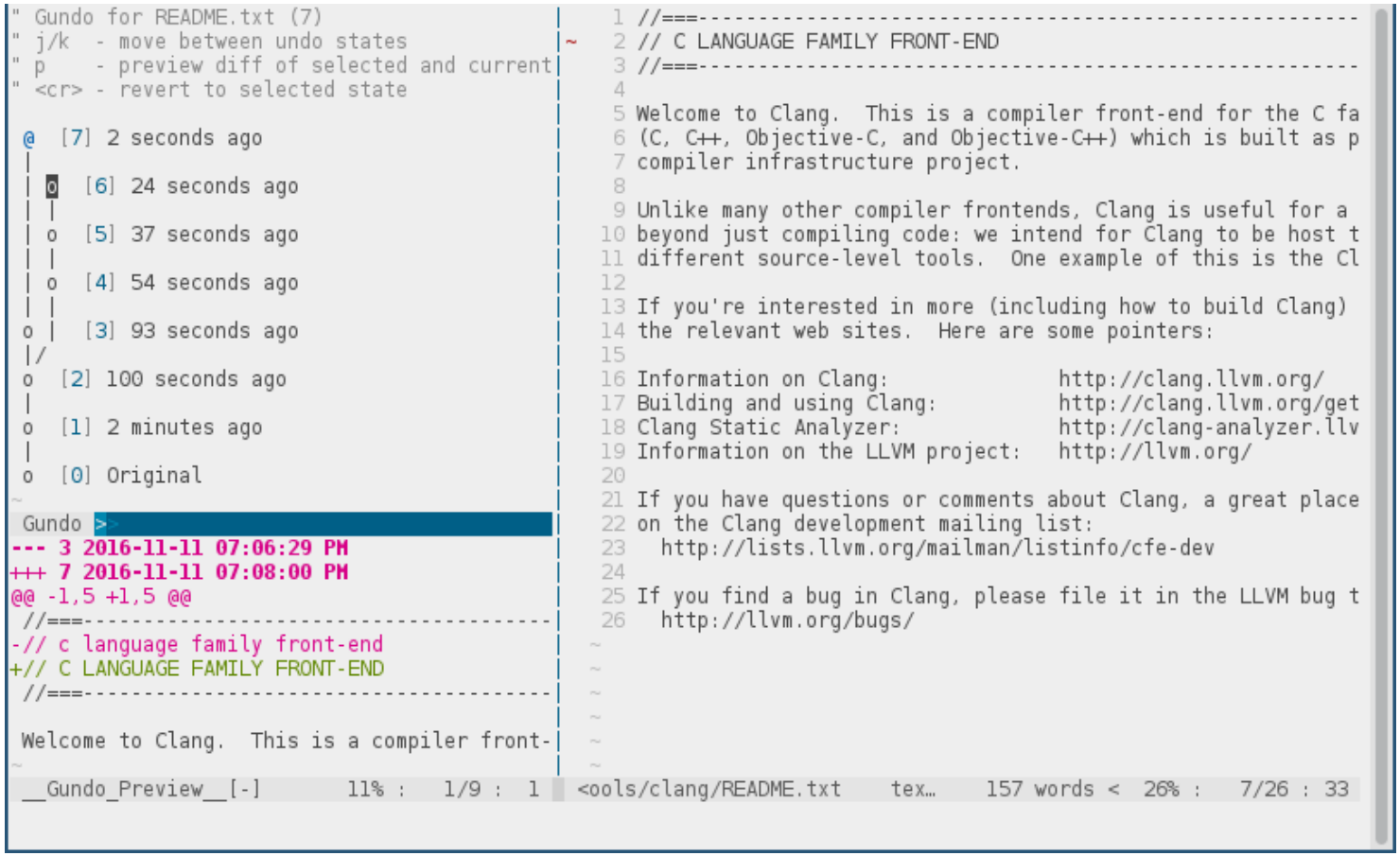
works ok with –system-clang (still recommend clion as default)
have not checked carefully the rope variant, but it seemed also ok (still recommend pycharm)

tagbar


unsigned ASTContext::NumImplicitCopyConstructorsDeclared;
unsigned ASTContext::NumImplicitMoveConstructors;
unsigned ASTContext::NumImplicitMoveConstructorsDeclared;
unsigned ASTContext::NumImplicitCopyAssignmentOperators;
unsigned ASTContext::NumImplicitCopyAssignmentOperatorsDeclared;
unsigned ASTContext::NumImplicitMoveAssignmentOperators;
unsigned ASTContext::NumImplicitMoveAssignmentOperatorsDeclared;
unsigned ASTContext::NumImplicitDestructors;
unsigned ASTContext::NumImplicitDestructorsDeclared;
enum FloatingRank {
HalfRank, FloatRank, DoubleRank, LongDoubleRank, Float128Rank
};
RawComment *ASTContext::getRawCommentForDeclNoCache(const Decl *D) const {
if (!CommentsLoaded && ExternalSource) {
ExternalSource->ReadComments();
#ifndef NDEBUG
ArrayRef<RawComment *> RawComments = Comments.getComments();
assert(std::is_sorted(RawComments.begin(), RawComments.end(),
BeforeThanCompare<RawComment>(SourceMgr)));
#endif
CommentsLoaded = true;
}
assert(D);
" 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)
NORMAL > master > <ext.cpp FloatingRank < cpp < 0% : 59/9377 : 6 < ! trailin… Tagbar Name ASTContext.cpp

undotree

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



changel

git-gutter

git mark modified lines (useful)

fugitive

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 |
```

airline

vim-slime

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:
`0vt="ayA<C-r>=<C-r>a<CR>`

colorschemes / cursorcolumn line / hexhighlight

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