Neat Features of Vim

Davis Claiborne

NCSU LUG

February 28, 2018

Linux Users Group
at NC State University

Indenting an Entire File

Formats using specified file indenting method

Format the entire file by `gg=G` in normal mode

Works for most file types (**not** Python)

See ‘`:help =`’ for more

Insert Mode Completion

Allows for automatic completion

- Entire lines: `<C-x><C-l>` [1]
- Keywords in current file: `<C-x><C-n>`
- Thesaurus: `<C-x><C-t>`
- Spelling `<C-x>s` [2] [3]
- Keywords in current and included files: `<C-x><C-i>`
- File names: `<C-x><C-f>`

See `:help ins-completion` for more

[1] `<C-x>` represents pressing “Ctrl” and “x” at the same time

[2] `spell` must be enabled

[3] **Not** `<C-s>`; in terminal Vim that suspends

Digraphs

Insert digraph characters (Ö, î, °, ...) easily

While in insert mode, press `<C-k>`, then the character and modifier

E.g. `<C-k>O:` creates Ö; `<C-k>i>` creates î

You can even define your own digraphs, allowing for things like emoji completion (if you want that for whatever reason)

E.g. Running `:digraph fi 128293` allows me to type `<C-k>fi` and insert the fire emoji

See `:help digraphs` for more

Text Objects

Allows you to select regions based on syntax

- `ip`: **i**nner **p**aragraph [4]
- `ap`: **a** **p**aragraph
- `i'`: **i**nner single quotes (text contained within single quotes)
- `it`: text within HTML **t**ags

See `:help text-objects` for more

[4] ‘Paragraphs’ are defined by blank lines

Visual Block Mode

Allows you to select blocks of text

Useful for working with blocks of text that span multiple lines, but don't include parts of the entire line.

See `:blockwise-visual` for more

Ranges

Allow you to specify commands for only specific parts of file

`5,10w temp.txt` writes lines 5-10 to a new file called `temp.txt`

`'<,'>` represent the start and end of a visual selection and are automatically put in the status line when working with visual selections

See `:help range` for more

Offsetting Ranges

Ranges can be offset by adding an amount to the end

This can be useful when you want to do some operation before or after a pattern

E.g. `/pattern/+1` will bring you to one line after the occurrence of `pattern`, or `/pattern/-1` will bring you one line before

See `:help range` for more

Global Command

Performs an action for a given command

E.g. `g/text/d` deletes every line with the word ‘text’

General pattern is `g/pattern/command`, where command is a visual-mode command, unless specified with normal

E.g. `g/text/normal d` deletes the first word on every line

See `:help global` for more

Formatting with External Programs

You can use `!` to ‘filter,’ or read, external programs

To insert the current date, run `:read !date`

To format columns, run `!column -t` on a visual selection

For instance, you can format a bunch of text to a fixed width of 80 by doing `vipJ:.!fmt -s -w 80` [5]

See `:help filter` for more

[5] Recall that `vip` selects the inner paragraph; `J` collapses all the lines to one; `!` passes `fmt` the text selected, then replaces the selected text with the result

Marks

Marks are useful for quickly navigating between sections of text

Create a mark with `m<letter>`, where `<letter>` is any letter

- Lower-case letters are valid only for one file
- Upper-case letters are valid for multiple files

Jump to the start of the line where the mark was made with `'<letter>` (single quote)

Jump to the exact location of the mark with `` `<letter> `` (backtick)

List all current marks with `:marks`

See `:help mark` for more

Recording Motions and Registers

Recordings are used for motions that will be repeated many times

Registers are used for storing text

Create a recording with `q<char>`, where `<char>` is any character that represents a register

- Lower-case registers are for “regular” motions
- Upper-case registers are appended to the corresponding lower-case registers
- Numbered registers 0-9 are used internally by Vim

See `:help recording` and `:help registers` for more

Undo Tree

Vim contains powerful undo capabilities

Vim helps prevent losing work with "undo trees:"

- Undo structure is stored as a tree
- Undoing then performing a new action creates a new branch

View tree with `:undolist`

Cycle through undos with `g-` and `g+`

Undos can be persistent across sessions with an `undofile`

Plugins exist to allow easier visualization of undo tree [2]

See `:help undo-tree` for more

References I

[1] Vim Documentation: `:help toc` or http://vimdoc.sourceforge.net/htmldoc/usr_toc.html

[2] Undotree https://github.com/mbbill/undotree

The End