Neat Features of Vim

Davis Claiborne

NCSU LUG

October 24, 2018

Linux Users Group
at NC State University

Why Vim?

- Large number of users means virtually any plug-in or theme you want has already been developed

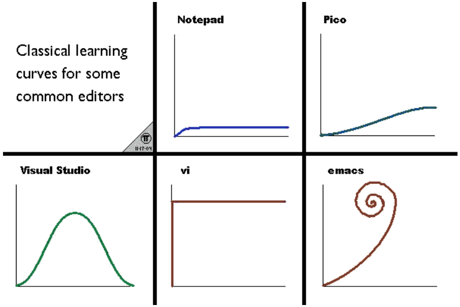
- Vs. IDEs:
  - Many plugins exist for Vim to mimic IDE features [1]
  - Keyboard-centric design means you can be faster with it

- Vs. Other terminal editors:
  - Lightweight and configurable
  - Modal editing allows for easier, more logical keyboard controls
  - Vim is ubiquitous
  - No Emacs Pinky :) [?]

[1] E.g. UltiSnips, fugitive, etc.

Why Not Vim?

- Vs. IDEs:
  - Requires significant tinkering to get just right
  - Much higher learning curve



- Vs. Other terminal editors:
  - Vimscript stinks
  - Not a one-stop-shop

Indenting an Entire File

Formats using specified file indenting method

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

More generally, `:`*`<start>`*`,`*`<end>`*`=` formats the lines from *`<start>`* to *`<end>`*

Format visual selection by typing = on the range

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

See '`:help =`' for more

Text Objects

Allows you to select regions based on syntax

- `ip`: **i**nner **p**aragraph [2]
- `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

[2] 'Paragraphs' are defined by blank lines

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

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 `:help blockwise-visual` 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 -s $‘\t’` on a range

To sort, text, run `sort -k` *<column>*

Format text to a fixed with: `!fmt -s -w 80`

See `:help filter` 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 dw` deletes the first word on every line

`:v/pattern/command` (or `:g!`) performs `command` on all lines that **don't** match `pattern`

See `:help global` for more

Insert Mode Completion

Allows for automatic completion

- Entire lines: `<C-x><C-l>` [3]
- Keywords in current file: `<C-x><C-n>`
- Thesaurus: `<C-x><C-t>`
- Spelling `<C-x>s` [4] [5]
- Keywords in current and included files: `<C-x><C-i>`
- File names: `<C-x><C-f>`

See `:help ins-completion` for more

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

[4] `spell` must be enabled

[5] **Not** `<C-s>`; in terminal Vim that suspends; use `<C-q> to resume`

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

E.g. running `:digraph ps 968` allows me to type `<C-k>ps 968` and insert the Greek character psi

See `:help digraphs` for more

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)

Jump between you last jumped from with `''` (double single quote)

Plugins exist for visualizing marks more easily, or you can list all current marks with `:marks`

See `:help mark` for more

Registers

Registers are used for storing text

The clipboard register is `"+`, so you can copy text to your clipboard with `"+y`*`<motion>`*

In insert mode, you can paste from your clipboard with `<C-r>+`

Other basic registers rules:

- Lower-case registers are “basic” registers
- Upper-case registers are appended to lower-case
- Numbered registers 0-9 are used internally by Vim
- . register contains the last inserted text
- % register contains the name of the current file

Run `:registers` to see the current registers

For more, see `:help registers`

Recording Motions

Recordings are used for motions that will be repeated many times

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

Execute a recording with `@<char>` (can use a count to perform it multiple time)

Because recordings are stored in registers, you can append to recordings

See `:help recording` for more

Folds

Folds can be used to hide regions of text; fold method changes how folds are interpreted

Use `set foldmethod=marker` to specify folding regions with {{{ and }}} along with an optional name and indent-level

Other fold method options:

- manual
- indent
- expr
- marker
- syntax
- diff

Use `zf` to create a fold [6]

See `:help folds` for more

[6] Fold method must be `manual` or `marker`

Undo Tree

Vim contains powerful undo capabilities

Vim helps prevent losing work with “undo trees:”

- Actions are stored as points on a tree
- Undoing then performing a new action creates a new, independent branch

View tree with `:undolist`

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

Undos can be persistent across sessions with an `undofile` (see `:help undo-persistence` for more)

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

See `:help undo-tree` for more

Buffers, Windows, and Tabs

According to the Vim manual: [?]

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

Buffers don't necessarily have to be visible

You can have multiple windows viewing a single buffer

Navigating Buffers, Windows, and Tabs

Buffers:

- Use command `:buffers` or `:ls` to view list of buffers
- Use command `:buffer` *<name>* to switch window's buffer
- Use `<C-^>` to rapidly switch window between last two buffers

Windows:

- Switch active window: `<C-w>` and `h`, `j`, `k`, or `l`
- Alternate active window: `<C-w><C-w>`
- Move windows: `<C-w>` and `<S-h>`, `<S-j>`, `<S-k>`, or `<S-l>` [7]

Tabs:

- Use command `:tabs` to view list of tabs
- Switch tabs with `:tabnext`

Save your current session with `:mksession` *<name>*, then load it with `:source` *<name>* or `vim -S` *<name>*

See `:help windows` for more

[7] `<S-h>` represents pressing "shift" and "h" at the same time

Miscsellaneous

Use `gf` to edit the filename under the cursor, or `gF` to edit the file at a specific line number if it's included

Use `]s` to jump to the next spelling mistake, `[s` for the previous, or `z=` on top of a word to bring up suggested words

Run vim with the `-d` flag to diff files in Vim

Tired of reaching for the escape key? `inoremap jk <ESC>`

Want your command line to be vim-like? `set editing-mode vi`

Vim has a built-in file browser: `vim <dir>`

Vim can edit and create encrypted files: `vim -x`

References I

[1] Emacs Pinky https://en.wikipedia.org/wiki/Emacs#Emacs_pinky

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

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