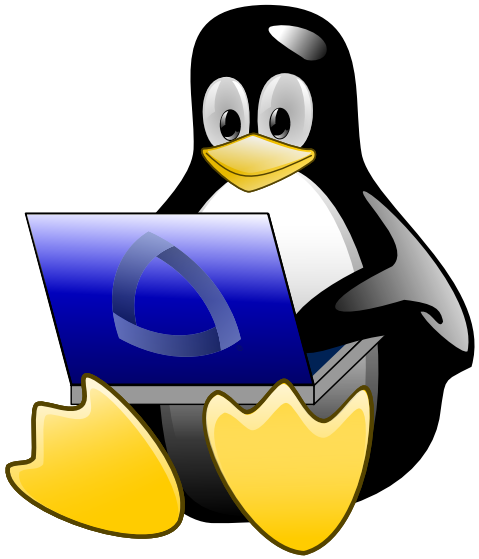
VIM: The Awesome Part

Jack Rosenthal

September 8, 2016



Colorado School of Mines
Linux Users Group

Generalizing our Keystrokes I

Most beginners at VIM see the keystrokes we type **statically**, i.e. they see the only keystrokes we make are the ones we remember the keystrokes for.

Keys remembered	**What it does**
`dd`	Delete line
`cw`	Change word
`dw`	Delete word
`yy`	Yank line
⋮	⋮

This is tedious, and incredibly hard to remember. Chances are, if you remember VIM this way, then you will hate it.

Generalizing our Keystrokes II

Let's divide our `NORMAL` mode keystrokes as two types[1]: **actions** and **movements**.

- Movements alone will move your cursor.
- Actions do things, and often take movements to act on.
- Repeating an action keystroke for its movement will act on the movement of the line
- When using one of the `VISUAL` modes, the *m* parameter to actions are dropped, and actions apply to the selection

Action		**Movement**	
d*m*	delete *m*	`h j k l`	left, down, up, right
y*m*	yank *m*	`b w`	word left, right
c*m*	change *m*	`^ $`	beginning, end of line
g~*m*	swap case on *m*	`gg G`	beginning, end of file
⋮	⋮	⋮	⋮

[1] not all keystrokes fall under these two categories

Repeating NORMAL Mode Commands

Adding a number *n* in front of a movement will do it *n* times.

Examples:

- `d5w` – delete the next 5 words
- `5j` – move 5 lines down
- `c2j` – change this line, and the next two

You can also use the `.` key to repeat a command again. For example, change a word, then move 5 words over, press `.` to make the same substitution again.

`find` and `till`

- `f`*c* will be the motion on to the next occurrence of the *c* character on the current line
- `t`*c* does the same, but goes right before *c* rather than on
- `F`*c* and `T`*c* do the same, but go backwards on the line

Examples:

- `fs` – move to the next occurrence of `s` on the line
- `ct.` – change till `.`
- `dTA` – delete backwards until just before `A`

inside and around

VIM has special movements that only act with actions called **inside** and **around**. They take a parameter of what to go inside or around. It works a bit like this:

Inside

- `i" i'` – inside a string
- `i) i} i]` – inside parens, squirly braces, brackets
- `iw is ip` – inside a word, sentence, paragraph

Around

- `a" a'` – around a string
- `a) a} a]` – around parens, squirly braces, brackets
- `aw as ap` – around a word, sentence, paragraph

For example:

- `ci"` – change inside the string you are on
- `vip` – select the current paragraph
- `dis` – delete the current sentence

The promised ROT13 encoding action

g?*m*

Taking Advantage of Deletion

`d`, `x` will put what was deleted into the paste register. This means you can swap things really easily:

- `xp` – swap characters
- `ddp` – swap lines
- and so fourth...

Some `Ex` mode stuff: `:g/`

- `:g/`*regex*`/`*cmd* will run the `Ex` mode command *cmd* on every line that matches *regex*
- For example, to delete all blank lines, do `:g/^$/d`
- Because the order on which `:g/` acts, you can use it to flip all the lines in a file using `:g/^/m0`

Getting some command output in your buffer

- `:!` followed by some shell command will run that command using your shell from vim.
- `:r`*cmd* will redirect some `Ex` mode command to the buffer
- We can combine them to redirect shell commands to our buffer. For example `:r!fortune | cowsay`

VIM Plugins

- VIM plugins are awesome, they let you alter VIM to nicely fit your work flow
- There are crap tons of VIM plugins
- You can turn VIM into a full fledged IDE with plugins
- I invite you to do a web search to find the ones you like best!