# Vim

Sumner Evans

September 7, 2017

Mines Linux Users Group

# Text Editors

# Why do we need text editors when we have IDEs?

- Often need to edit configuration files and you don't want to fire up Visual Studio to edit a simple text file.
- Most tasks do not need any fancy IDE features to complete. For example, writing a simple Markdown file, writing a LaTeX document, writing this presentation with Beamer, programming, etc.
- Most of the time, users don't use nearly all of the features that IDEs provide. This is a waste of computer resources.
- Good text editors have the ability to be extended to bring IDE-like features to your text editor. This gives greater customisation ability to the user.

# Why do we need text editors when we have IDEs?

- Often need to edit configuration files and you don't want to fire up Visual Studio to edit a simple text file.
- Most tasks do not need any fancy IDE features to complete. For example, writing a simple Markdown file, writing a LaTeX document, writing this presentation with Beamer, programming, etc.
- Most of the time, users don't use nearly all of the features that IDEs provide. This is a waste of computer resources.
- Good text editors have the ability to be extended to bring IDE-like features to your text editor. This gives greater customisation ability to the user.

# Why do we need text editors when we have IDEs?

- Often need to edit configuration files and you don't want to fire up Visual Studio to edit a simple text file.
- Most tasks do not need any fancy IDE features to complete. For example, writing a simple Markdown file, writing a LaTeX document, writing this presentation with Beamer, programming, etc.
- Most of the time, users don't use nearly all of the features that IDEs provide. This is a waste of computer resources.
- Good text editors have the ability to be extended to bring IDE-like features to your text editor. This gives greater customisation ability to the user.

# Why do we need text editors when we have IDEs?

- Often need to edit configuration files and you don't want to fire up Visual Studio to edit a simple text file.
- Most tasks do not need any fancy IDE features to complete. For example, writing a simple Markdown file, writing a LaTeX document, writing this presentation with Beamer, programming, etc.
- Most of the time, users don't use nearly all of the features that IDEs provide. This is a waste of computer resources.
- Good text editors have the ability to be extended to bring IDE-like features to your text editor. This gives greater customisation ability to the user.

# Why use a terminal-based text editor?

- Often need to edit files and you don't want to have to open a GUI file selector to navigate to the correct file and edit it. (In fact, there are some bad file selectors which don't even let you open dotfiles!)
- If you are SSH-d into a remote machine (for example, while administering a server), you may not have the luxury of a GUI (there is SSH tunnelling, but that is not always an option).
- These text editors are made by programmers for programmers.

# Why use a terminal-based text editor?

- Often need to edit files and you don't want to have to open a GUI file selector to navigate to the correct file and edit it. (In fact, there are some bad file selectors which don't even let you open dotfiles!)
- If you are SSH-d into a remote machine (for example, while administering a server), you may not have the luxury of a GUI (there is SSH tunnelling, but that is not always an option).
- These text editors are made by programmers for programmers.

# Why use a terminal-based text editor?

- Often need to edit files and you don't want to have to open a GUI file selector to navigate to the correct file and edit it. (In fact, there are some bad file selectors which don't even let you open dotfiles!)
- If you are SSH-d into a remote machine (for example, while administering a server), you may not have the luxury of a GUI (there is SSH tunnelling, but that is not always an option).
- These text editors are made by programmers for programmers.

# Vim

# Why use Vim over other terminal-based text editors?

- You don't need a mouse at all. In fact, by default, you *can't* use your mouse.
- You can do everything with the keyboard in just a few keystrokes.
- The `CTRL` key is hard to reach on standard keyboards.
- But all of this is only possible because of **modal editing**.

# Why use Vim over other terminal-based text editors?

- You don't need a mouse at all. In fact, by default, you *can't* use your mouse.
- You can do everything with the keyboard in just a few keystrokes.
- The `CTRL` key is hard to reach on standard keyboards.
- But all of this is only possible because of **modal editing**.

# Why use Vim over other terminal-based text editors?

- You don't need a mouse at all. In fact, by default, you *can't* use your mouse.
- You can do everything with the keyboard in just a few keystrokes.
- The `CTRL` key is hard to reach on standard keyboards.
- But all of this is only possible because of **modal editing**.

# Why use Vim over other terminal-based text editors?

- You don't need a mouse at all. In fact, by default, you *can't* use your mouse.
- You can do everything with the keyboard in just a few keystrokes.
- The `CTRL` key is hard to reach on standard keyboards.
- But all of this is only possible because of **modal editing**.

# Non Modal Editing

- In most editors, when you type with your keyboard, what you type is inserted right at the cursor location.
- To access functionality such as selection and scrolling you need to use your mouse.
- To access functionality such as find [and replace], you need to use a menu item or some complicated keyboard shortcut.

# Non Modal Editing

- In most editors, when you type with your keyboard, what you type is inserted right at the cursor location.
- To access functionality such as selection and scrolling you need to use your mouse.
- To access functionality such as find [and replace], you need to use a menu item or some complicated keyboard shortcut.

# Non Modal Editing

- In most editors, when you type with your keyboard, what you type is inserted right at the cursor location.
- To access functionality such as selection and scrolling you need to use your mouse.
- To access functionality such as find [and replace], you need to use a menu item or some complicated keyboard shortcut.

# Modal Editing: The Ultimate Separation of Powers

- Pressing the same keys do different actions depending on which mode you are in.
- This is extremely space-efficient.

# Modes: Normal

- This is the default (normal) mode.
- It is sometimes referred to as command mode.
- This is the mode that you are in when you enter Vim.
- You can tell you are in normal mode because there will be no text in the bottom left corner of your console window.
- Used to get to other modes, for cursor movement, copy/pasting, saving, etc. . .
- **To return here from other modes, press the `ESC` key.**

# Modes: Insert

- Allows you to actually type text.
- To get to insert mode from normal mode, press `i`.

# Modes: Visual

- Allows you to select text at a character-resolution and perform actions upon that selected text.
- To get to visual mode from normal mode, press `v`.

# Modes: Visual-Line

- Allows you to select text at a line-resolution and perform actions upon that selected text.
- To get to visual-line mode from normal mode, press `V` (capital V).

# Modes: Visual Block Mode

- Allows you to select text vertical blocks of text and perform actions on that selected text.
- To get to visual mode from normal mode, press `CTRL + v`.

# Common Commands in Normal Mode

- `h`, `j`, `k`, and `l` move the cursor left, down, up, and right, respectively.
- `i` puts you into insert mode, right where the cursor is.
- `I` puts you into insert mode at the beginning of the current line.
- `a` puts you into insert mode, one character to the right of the cursor.
- `A` puts you into insert mode at the end of the current line.
- `o` inserts a line below the current line, and puts you into insert mode on that line.
- `O` (capital O) is the same as lower-case `o`, but a line above.

# Common Commands in Normal Mode

- `dd` will delete the current line.
- `yy` will copy the current line.
- `cc` will delete the current line and put you into insert mode at the beginning of the line.
- `x` deletes the character under the cursor. `X` deletes the character before the cursor.
- `p` will paste whatever is currently in the paste buffer.
  - How do you put something into the paste buffer? With `x`, `dd`, or `yy`! These also function as what you would think of as cut and copy.
  - But I can't paste from other programs! Vim sucks. Use `"+y` and `"+p` to copy and paste, respectively, from the system clipboard. Alternatively, add `set clipboard=unnamedplus` to your `.vimrc`.

# Common Commands in Normal Mode

- `u` can be used to undo, and `Ctrl+r` to redo.
- `w` moves the cursor forward by one word at a time, and `b` moves it back.
- `gg` moves the cursor to the top of the file.
- `G` moves the cursor to the bottom of the file.

# I'm Stuck and I can't exit Vim!

- Type `:w` from normal mode to save the file (you can do this at any point in the edit process)
- `:q` will exit vim. If you have unsaved edits, it will warn you of this and not exit.
- `:q!` exits silently and without saving. Only use this if you really don't want your file changes!
- Lastly, these can be strung together to save and quit, i.e. `:wq`. There is also `:x`, which does the same thing

# A Sample of Cool Other Commands

- `ci(` "change inside parentheses" — deletes everything inside the current parenthetical statement and then puts you into insert mode. Super useful for changing function parameters.
- `cap` "change around paragraph" — deletes the paragraph and puts you into insert mode.
- `dw` "delete word" — delete the next word.
- `d3w` "delete word" — delete the next three words.
- `dt)` "delet til )" — deletes everything on the line until the next `)`.
- `df)` deletes up to and including the next `)`.
- Lots more examples in Jack's talk from last year. (`https://github.com/jackrosenthal/lug-vim-awesome`)

# Plugins

There are a lot of plugins for Vim. I use 37 different plugins! If you want Vim to do something for you, Google it and someone has probably implemented that feature. (Or come to LUG for recommendations!)

My configurations:
`https://github.com/sumnerevans/dotfiles/tree/master/.vim`

# Installing Vim

- Linux: install the `vim` package using your package manager
- macOS: install the `vim` using Homebrew
- Windows: Google it

# References

I used Caleb Jhones' and Jack Rosenthal's presentations on Vim from last year as as inspiration/source code for this talk.

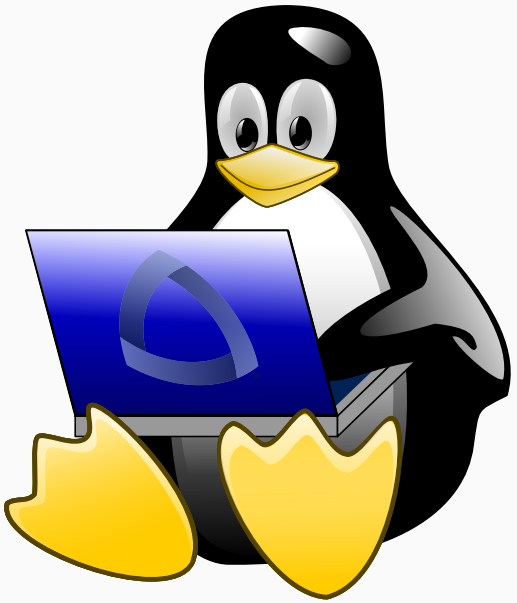
- Caleb's presentation:
  `https://github.com/ThirdOf5/VIM-Intro`
- Jack's presentation:
  `https://github.com/jackrosenthal/lug-vim-awesome`

# Questions?

# Copyright Notice

This presentation was from the **Mines Linux Users Group**. A mostly-complete archive of our presentations can be found online at `https://lug.mines.edu`.

Individual authors may have certain copyright or licensing restrictions on their presentations. Please be certain to contact the original author to obtain permission to reuse or distribute these slides.



**Colorado School of Mines**
Linux Users Group