VIM: The Basic Part

Caleb Jhones

8 September 2016
A bit of history

• Begun by Bram Moolenaar in the late 80s as a port of the Stevie editor (on the 80s Amiga computers)
• Publicly released in 1991, and has been continually updated ever since
• 'Vim' originally stood for 'Vi IMitation'. This was later changed to 'Vi IMproved' when vim's functionality surpassed that of its predecessor
A bit of history

- Begun by Bram Moolenaar in the late 80s as a port of the Stevie editor (on the 80s Amiga computers)
- Publicly released in 1991, and has been continually updated ever since
- 'Vim' originally stood for 'Vi IMitation'. This was later changed to 'Vi IMproved' when vim’s functionality surpassed that of its predecessor
Why use vim?

• Because it's the best plaintext editor available!

(please don't hurt me) (emacs users, I'm looking at you)

• In reality, either will work very well. I prefer vim because:

• All commonly-used commands are single (or sometimes two) keystrokes

• If you're using a standard keyboard layout, the control key can be inconvenient to reach

• Modal editing (more on this later)

• Mainly, it's what I started learning, and have grown to like

• Brace yourselves.... if you've never used vim before, what follows will be like a fire-hose. I'm partly intending this to be a reference for you all, so don't worry about absorbing everything right now
Why use vim?

- Because it’s the best plaintext editor available!
Why use vim?

- Because it’s the best plaintext editor available! (please don’t hurt me)
Why use vim?

• Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
Why use vim?

- Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
- In reality, either will work very well. I prefer vim because:
Why use vim?

- Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
- In reality, either will work very well. I prefer vim because:
  - All commonly-used commands are single (or sometimes two) keystrokes
Why use vim?

- Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
- In reality, either will work very well. I prefer vim because:
  - All commonly-used commands are single (or sometimes two) keystrokes
  - If you’re using a standard keyboard layout, the control key can be inconvenient to reach
Why use vim?

• Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
• In reality, either will work very well. I prefer vim because:
  • All commonly-used commands are single (or sometimes two) keystrokes
  • If you’re using a standard keyboard layout, the control key can be inconvenient to reach
  • Modal editing (more on this later)
Why use vim?

• Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
• In reality, either will work very well. I prefer vim because:
  • All commonly-used commands are single (or sometimes two) keystrokes
  • If you’re using a standard keyboard layout, the control key can be inconvenient to reach
  • Modal editing (more on this later)
  • Mainly, it’s what I started learning, and have grown to like
Why use vim?

- Because it’s the best plaintext editor available! (please don’t hurt me) (emacs users, I’m looking at you)
- In reality, either will work very well. I prefer vim because:
  - All commonly-used commands are single (or sometimes two) keystrokes
  - If you’re using a standard keyboard layout, the control key can be inconvenient to reach
  - Modal editing (more on this later)
  - Mainly, it’s what I started learning, and have grown to like
- Brace yourselves…. if you’ve never used vim before, what follows will be like a fire-hose. I’m partly intending this to be a reference for you all, so don’t worry about absorbing everything right now
Modal editing

In more GUI-dependent text editors, you need to use the mouse very often (for moving the cursor, selecting file menus, &c). It also depends on things like arrow keys and modifier keys (Shift, Control, Alt, &c). This means you need to move your hands away from the home row, which is slow and annoying. Vim removes these problems by being driven completely using a keyboard (rather than a keyboard and mouse), and making only sparing use of modifier keys. This is possible using modes. Each key on the keyboard does something different in each mode, meaning you have many possible functions of each key, beyond simply typing that letter into a file.
Modal editing

- In more GUI-dependent text editors, you need to use the mouse very often (for moving the cursor, selecting file menus, &c). It also depends on things like arrow keys and modifier keys (Shift, Control, Alt, &c).
Modal editing

- In more GUI-dependent text editors, you need to use the mouse very often (for moving the cursor, selecting file menus, &c). It also depends on things like arrow keys and modifier keys (Shift, Control, Alt, &c).
- This means you need to move your hands away from the home row, which is slow and annoying.
Modal editing

• In more GUI-dependent text editors, you need to use the mouse very often (for moving the cursor, selecting file menus, &c). It also depends on things like arrow keys and modifier keys (Shift, Control, Alt, &c)

• This means you need to move your hands away from the home row, which is slow and annoying

• Vim removes these problems by being driven completely using a keyboard (rather than a keyboard and mouse), and making only sparing use of modifier keys
Modal editing

- In more GUI-dependent text editors, you need to use the mouse very often (for moving the cursor, selecting file menus, &c). It also depends on things like arrow keys and modifier keys (Shift, Control, Alt, &c).
- This means you need to move your hands away from the home row, which is slow and annoying.
- Vim removes these problems by being driven completely using a keyboard (rather than a keyboard and mouse), and making only sparing use of modifier keys.
- This is possible using modes. Each key on the keyboard does something different in each mode, meaning you have many possible functions of each key, beyond simply typing that letter into a file.
Modes

- Normal (sometimes called command) mode:
  - No text in the bottom left corner of your console window
  - Used to get to other modes, for cursor movement, copy/pasting, saving, etc...
  - This is where you begin when you open a vim window
  - To return here from other modes, press the Esc key
• Normal (sometimes called command) mode:
  • No text in the bottom left corner of your console window
  • Used to get to other modes, for cursor movement, copy/pasting, saving, etc...
  • This is where you begin when you open a vim window
  • To return here from other modes, press the Esc key
• Insert mode:
  • Normal mode is great. But this is a text editor, and I want to type!
• **Insert mode:**
  • Normal mode is great. But this is a text editor, and I want to type!
  • From normal mode, press `i` to get to insert mode
  • You are now able to edit files! Type away to your heart’s content!
• Visual mode:
  • From normal mode, press \texttt{v}
  • You can now select text one character or (partial) line at a time using your cursor
• Visual mode:
  • From normal mode, press $v$
  • You can now select text one character or (partial) line at a time using your cursor

• Visual line mode:
  • From normal mode, press $V$ (capital V)
  • Now you can select text one (whole) line at a time
• **Visual mode:**
  - From normal mode, press `v`
  - You can now select text one character or (partial) line at a time using your cursor

• **Visual line mode:**
  - From normal mode, press `V` (capital V)
  - Now you can select text one (whole) line at a time

• **Visual block mode:**
  - From normal mode, press `Ctrl+v`
  - Now select using `h`, `j`, `k`, and `l` in blocks (hence the name)
  - Using `Shift+i`, you can insert text at the beginning of your selection (see example)
Basic commands, from normal mode

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

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

- `dd` will delete an entire line, and `yy` will copy an entire line (whichever line the cursor is on)
- `x` deletes the character under the cursor. `X` deletes the character before the cursor
Basic commands, from normal mode

- `dd` will delete an entire line, and `yy` will copy an entire line (whichever line the cursor is on)
- `x` deletes the character under the cursor. `X` deletes the character before the cursor
- `p` will paste whatever is in the buffer currently
  - 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
Basic commands, from normal mode

- u can be used to undo, and Ctrl+r to redo
Basic commands, from 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
Basic commands, from 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
Saving, loading, quitting

• So we're finally done with editing our file, and we want to save. Or maybe we decided we didn't need the edits we made after all.

• Type :w from normal mode to save the file (you can do this at any point in the edit process).

• :q will exit vim, without saving. 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.
So we’re finally done with editing our file, and we want to save. Or maybe we decided we didn’t need the edits we made afterall
Saving, loading, quitting

• So we’re finally done with editing our file, and we want to save. Or maybe we decided we didn’t need the edits we made afterall

• Type \texttt{:w} from normal mode to save the file (you can do this at any point in the edit process)

• \texttt{:q} will exit vim, without saving. If you have unsaved edits, it will warn you of this and not exit

• \texttt{: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. \texttt{:wq}. There is also \texttt{:x}, which does the same thing
Other cool tricks

• Use :G to retab an entire file (as vim sees appropriate! Not always correct, sadly)
• You can also just use == to do a single line
• You can auto-complete any word that vim has already seen in the file by using Ctrl+p
• You can also run shell commands straight from vim (particularly useful for things like make). Type :!<your command> and it will be run in your shell
Other cool tricks

- Use `gg=G` to retab an entire file (as vim sees appropriate! Not always correct, sadly)
- You can also just use `==` to do a single line
Other cool tricks

- Use `gg=G` to retab an entire file (as vim sees appropriate! Not always correct, sadly)
- You can also just use `==` to do a single line
- You can auto-complete any word that vim has already seen in the file by using `Ctrl+p`
Other cool tricks

- Use gg=G to retab an entire file (as vim sees appropriate! Not always correct, sadly)
- You can also just use == to do a single line
- You can auto-complete any word that vim has already seen in the file by using Ctrl+p
- You can also run shell commands straight from vim (particularly useful for things like `make`). Type `:!<your command>` and it will be run in your shell
Now for Jack