Modal Editing

One mode just isn't enough

Timothy

2020-06-25

'Normal' editing

Standard keybindings

You'll likely be used to some common shortcuts

Shortcut	Action
`Ctrl + a`	Select all
`Ctrl + f`	Find
`Ctrl + c`	Copy
`Ctrl + v`	Paste
`Ctrl + o`	Open
`Ctrl + s`	Save
...	

Application-specific keybindings

Word

Shortcut	Action
`Ctrl + Shift + <right>`	Select word to the right
`Ctrl + Alt + z`	Cycle through previous (4) changes

VS Code

Shortcut	Action
`Ctrl + r`	Open recent
`Alt + Shift + a`	Block comment
`Ctrl + Alt + Shift + c`	Copy relative path of file
`Ctrl + K, Ctrl + Alt + s`	Git, stage selected region

The problem with normal editing

The problem with normal editing

Shortcut overloading.

Mathematical shortcut limit

- 26 letters + 10 numbers + 12 (not-on-number) symbol keys + 12 function keys
- `Ctrl`, `Alt`, `Shift` modifiers, 4 possible combinations
- How many possible shortcuts?

Mathematical shortcut limit

- 26 letters + 10 numbers + 12 (not-on-number) symbol keys + 12 function keys
- `Ctrl`, `Alt`, `Shift` modifiers, 4 possible combinations
- How many possible shortcuts?

$$4 \times (26 + 10 + 12) + 3 \times 12 = 228$$

That's . . . a lot?

Mathematical shortcut limit

- 26 letters + 10 numbers + 12 (not-on-number) symbol keys + 12 function keys
- `Ctrl`, `Alt`, `Shift` modifiers, 4 possible combinations
- How many possible shortcuts?

$$4 \times (26 + 10 + 12) + 3 \times 12 = 228$$

That's ... a lot? Actually, once you consider all the different categories of things, it's not that many — hence clunky shortcuts like `Ctrl + Alt + Shift + c`.

Solution? Add a prefix key (VS Code style)

- VS Code reserves `Ctrl + K` to be used as a prefix for a second shortcut
- This increases the number of shortcuts to `452`
- More flexibility is good!

Solution? Add a prefix key (VS Code style)

- VS Code reserves `Ctrl + K` to be used as a prefix for a second shortcut
- This increases the number of shortcuts to `452`
- More flexibility is good! . . . but it's also even more clunky :(
- `Ctrl + K, Ctrl + Alt + s`

Solution! Modal editing

Modal Editing

Observation: There are *way* more keys than modifiers

- 3 modifier keys (`Ctrl`, `Alt`, `Shift`)
- 48 ASCII character keys
- If only there was some way we could use them...

Observation: There are *way* more keys than modifiers

- 3 modifier keys (`Ctrl`, `Alt`, `Shift`)
- 48 ASCII character keys
- If only there was some way we could use them...

Think of how much time you spend editing

Use them keys

- Turn 'normal' typing text into a special case
- The opens up the possibility to use every single key for an action

Use them keys — how?

- Define different modes
- Start off in a default mode
- From the default mode, move to other modes using any key
- Leave a mode with `Esc`
- This opens up *every single key* for use in actions

Use them keys — an example

- Let's call the default mode normal
- Let's call the you-see-what-you-type behaviour (that you're used to) insert mode
- Type `i` to enter insert mode, type like you're in any other app
- Press `Esc` and you go back to normal mode

The mathematical advantage

- Use up to three keys in a row
- Each key can be any letter, number, symbol + usual modifiers
- $(26 + 10 + 12) \times 4 = 192$ options
- How many possible keybindings can we use now?

The mathematical advantage

- Use up to three keys in a row
- Each key can be any letter, number, symbol + usual modifiers
- $(26 + 10 + 12) \times 4 = 192$ options
- How many possible keybindings can we use now?

$$192^3 = 7\,077\,888$$

This is a *huge* increase from the 192 possibilities 'normally', in fact this is a factor of 36-thousand more!

The mnemonic advantage

You are no longer forced to use weird keybindings.

With so many options, one can design mnemonic categories of keybindings. E.g. put actions to do with deletion under d, etc.

The ergonomic advantage

I no longer feel like I have to be part-human, part-spider to use uncommon shortcuts.

The capability curve

Phases:

1. WTF?
2. Clunky
3. Eh, it's alright
4. I like this
5. OMG!

Vim

What is it?

- An old text editor
- An old modal text editor
- Despite being ~30 years old, people still use it

What is it?

- An old text editor
- An old modal text editor
- Despite being ~30 years old, people still use it
- Clearly it did something right (modal editing)

A quick history

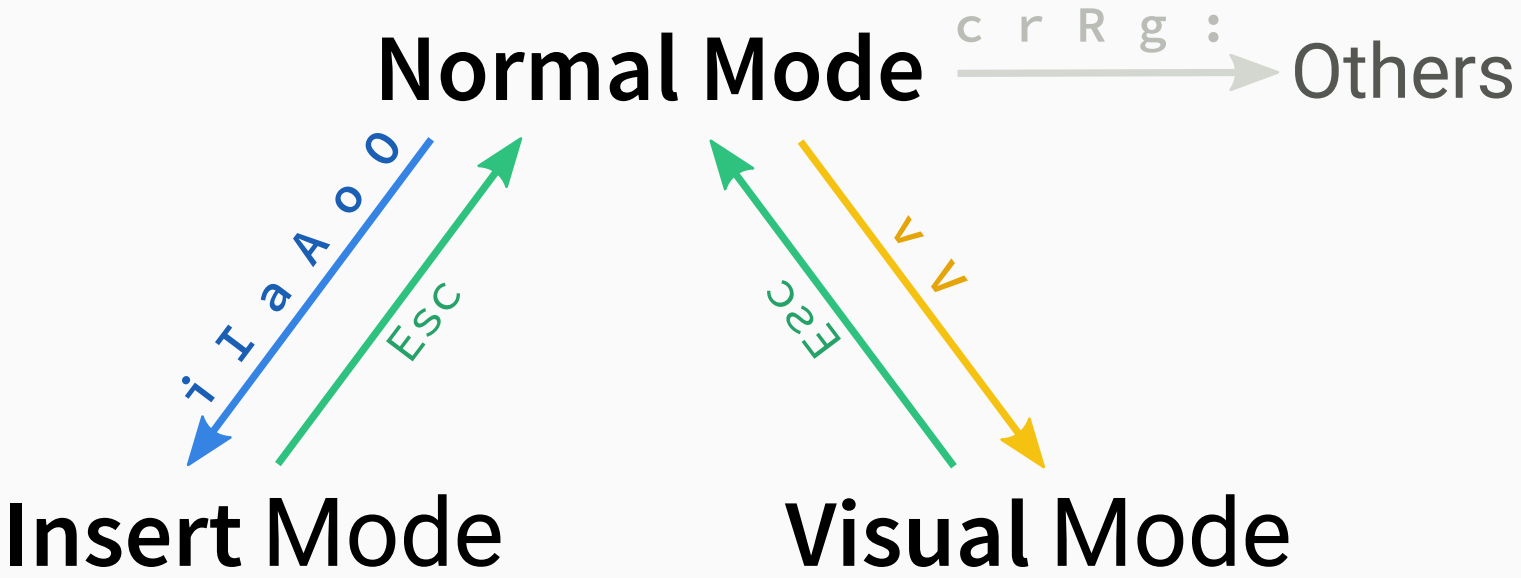
- Rewrite of the `vi` editor (which Bill Joy created in 1976)
- Motivation for creation:
  - George Coulouris worked at AT&T
  - The 'default' editor at the time was nasty and clunky
  - George had an idea for a better way (`em`)
  - Bill Joy is impressed, re-implements favourite ideas (`vi`)
- The hardware back then looked a bit different

There is no mouse and you cannot click!

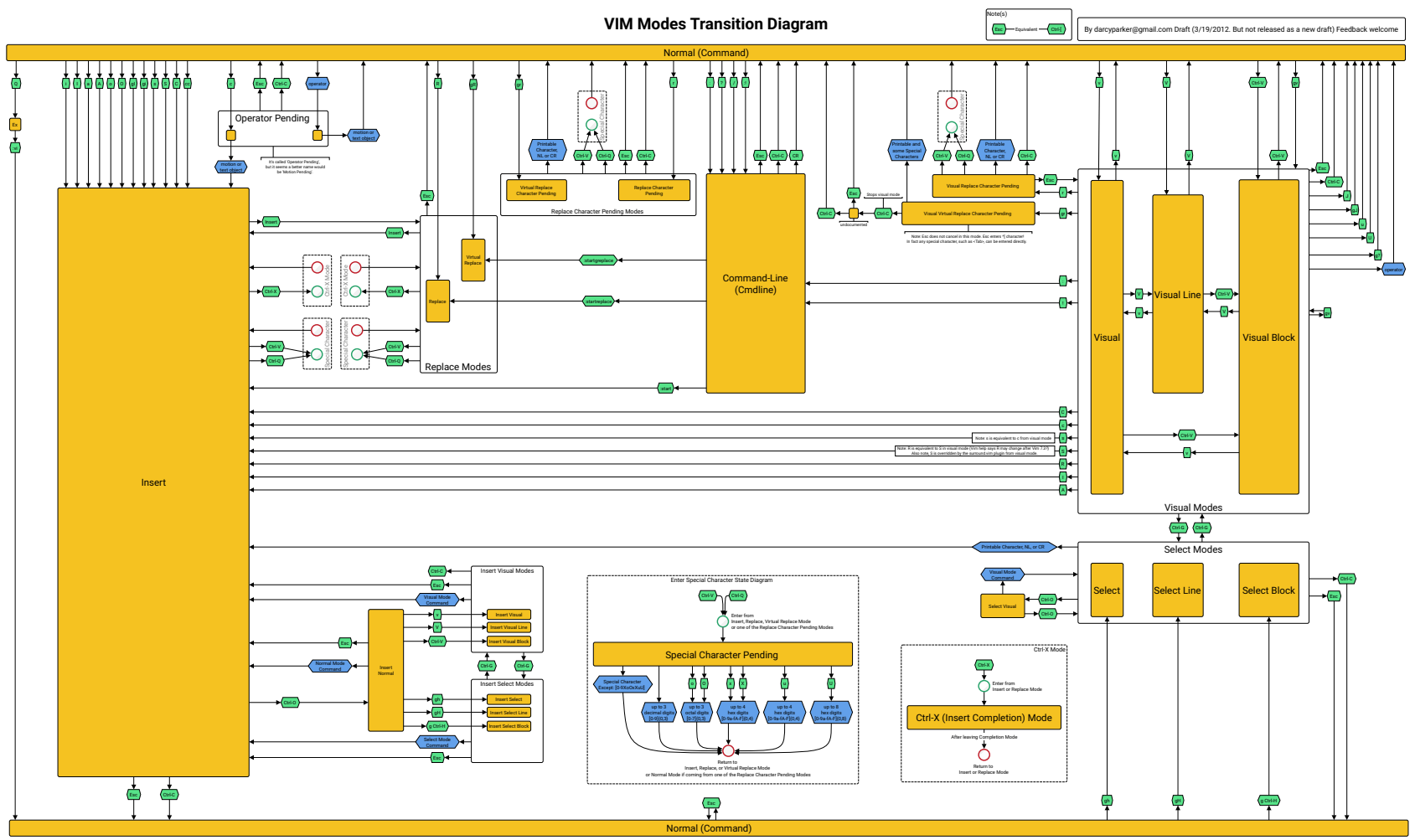
The (main) vim modes

- Normal mode — the . . . normal mode (default)
- Insert mode — type text normally
- Visual Mode — select text

A simple diagram



Modal madness



Let's learn Vim

The Basics

Vim is the name of the terminal command, `vim`. Please don't be scared of terminals, they're really quite simple. To open `file.txt` in vim, just run `vim file.txt`.

The Basics

Vim is the name of the terminal command, `vim`. Please don't be scared of terminals, they're really quite simple. To open `file.txt` in vim, just run `vim file.txt`.

From there you can edit text just knowing this

- Press `i` to enter insert mode, and type some text
- Press `Esc` to stop writing text (go to normal mode)
- Type `:wq` to save your file and exit.

The Basics — exiting Vim

- Type `:wq` to save your file and exit.

A little bit more

For going through your edits

- `u` undoes the last change
- `Ctrl + r` re-does the last change

For making and using text selections

- `v` to start a selection
- `y` to yank (copy) the text to the clipboard
- `p` to paste
- `d` to delete

Demo

More Vim

Aside — shortened shortcut notation

- We can do better than typing `Ctrl + a`, or `Shift + i` each time.
- So, we perform the following contractions
  - `Shift   + a` becomes `A`
  - `Control + a` becomes `C-a`
  - `Alt     + a` becomes `M-a`[1]
  - `Control + Alt + Shift + a` becomes `C-M-A`

    isn't that easier?

[1] `M` not `A` because `Alt` used to be called `Meta`

Insert Mode

- Enter using one of `i I a A o O gi`
  - `i` insert at point
  - `I` insert at start of line
  - `a` insert after point
  - `A` insert at end of line
  - `o` insert on new line above
  - `O` insert on new line below
  - `gi` insert at last edit

Insert Mode

- Enter using one of `i I a A o O gi`
  - `i` insert at point
  - `I` insert at start of line
  - `a` insert after point
  - `A` insert at end of line
  - `o` insert on new line above
  - `O` insert on new line below
  - `gi` insert at last edit
- `C-w` to delete the last word
- `C-u` to delete the last line (this is the same as `dd` in normal mode)

Think about the mnemonics in your head.

Demo

Visual mode

- Make a selection using on of `v` `V` `C-v` `gv`
  - `v` visual character mode
  - `V` visual line mode
  - `C-v` visual block mode (rectangle)
  - `gv` last visual selection

Visual mode

- Make a selection using on of `v` `V` `C-v` `gv`
  - `v` visual character mode
  - `V` visual line mode
  - `C-v` visual block mode (rectangle)
  - `gv` last visual selection
- `o` switch between start/end of selection
- grow/shrink selection with normal cursor movement

Demo

'Quick' actions

- Some action don't need a selection to act on
- If you press the key twice, they'll simply act on the current line

'Quick' actions

- Some action don't need a selection to act on
- If you press the key twice, they'll simply act on the current line
- This works for
  - `dd` delete line
  - `yy` yank (copy) line
  - `cc` change line
  - `==` re-indent line
  - `>>` shift line right
  - `<<` shift line left

'Quick' actions

- Some action don't need a selection to act on
- If you press the key twice, they'll simply act on the current line
- This works for
  - `dd` delete line
  - `yy` yank (copy) line
  - `cc` change line
  - `==` re-indent line
  - `>>` shift line right
  - `<<` shift line left
- You can also use `x` to delete the character under the cursor, or the active selection, and `r` to replace the character.

Actions on a selection

- d delete
- c change content (delete and enter insert mode)
- y (yank) copy selection to clipboard

Actions on a selection

- d delete
- c change content (delete and enter insert mode)
- y (yank) copy selection to clipboard
- ~ swap case
- u make lowercase
- U make uppercase

Demo

More actions on a selection

- ! run selection though an external program
- = re-indent
- > shift right
- < shift left

Demo

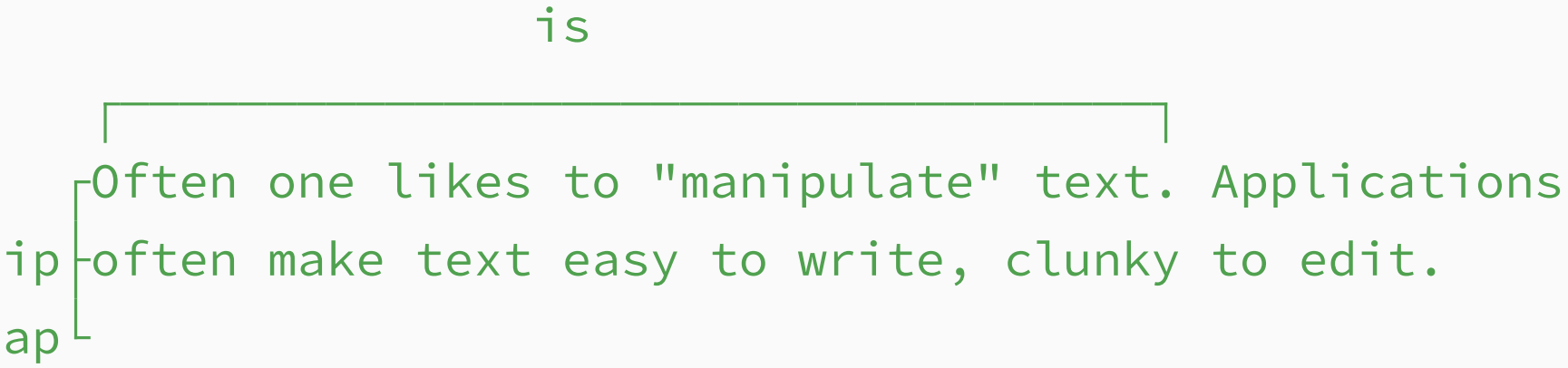
Text objects

- Used while in visual mode, or after an action (e.g. `d` for delete)
- Performed from the current cursor
- Can be prefixed with `i` or `a`
  - `i` selects the 'inner' object
  - `a` selects the 'outer' object

Text objects — a list

- `w` word
- `p` paragraph
- `b` brackets, or `()` , `[]`, `{}`, `<>` if you want to be specific
- `'` or `"` for quoted text

Text objects — an example



That's why modal editing is nice.

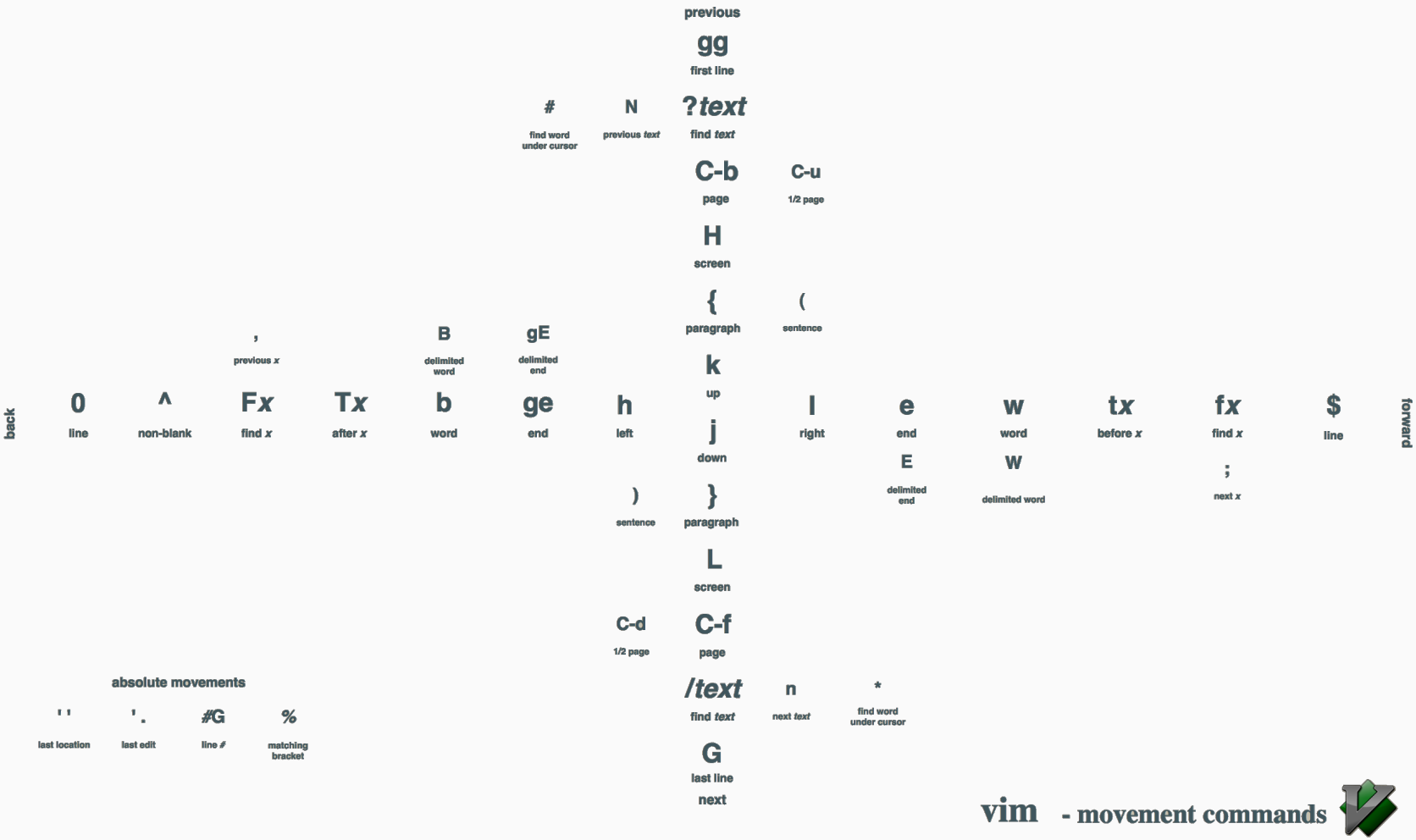
Demo

Movements

Has the same effect as arrow keys, `Home`, and `End`.

Used for moving the current cursor, and shrinking/growing a selection.

Movements — overview



Movements — put another way

Micro

target	forwards	forward upto	backwards	back upto
word	`w`	`e`	`b`	`ge`
x	`f` *x*	`t` *x*	`F` *x*	`T` *x*

Macro

target	start	end
line	`0`	
line content	`^`	`$`
sentence	`(`	`)`
paragraph	`{`	`}`
document	`gg`	`G`
page	`C-b`	`C-f`

Movements — an example

```
gg
{     w              fc                          tl   $
↓     ↓              ↓                           ↓    ↓
Pretend there is a cursor at the start of this line,
then look around at the various movements.
                                                 ↑
0 ^             Tc                  Fd           )    }
↓ ↓             ↓                   ↓                 ↓
  Pretend the cursor is at the end of this now.
                                                      ↑
                                                      G
```

Demo (Prose) Demo (Code)

Count prefix

Want to do something multiple times, for example not just delete the next word (`dw`), but the next three words?

It's easy! Just type the number of times you want to do it *before* or *during* the action.

In this example, 3-delete-word (`3dw`) or delete-3-words (`d3w`) does the trick.

This works for everything though, not just words :)

Structure of a vim operation


```
d5w   =iap   ci'   7@a   ^   "ayab   Uib
```

EXAMPLES

Command mode

- A way to tell vim to do non-text-editing stuff
- Enter command mode with `:`
- This is how you:
  - `:w` write a file
  - `:q` quit vim
  - `:wq` write a file, and quit
  - and more

Even more Vim

Oh yes, there's more

You've just seen how to perform 'basic' operations in vim.

Don't worry, while it may look intimidating, if you try this out you'll find yourself picking it up quickly — the mnemonic-ness helps a lot.

Vim does a bit more though, so we'll go over that for fun (and profit).

Marks — saving

- A way to save a position
- Save the current location with `m?` where `?` is a letter (a-z) which serves as a unique identifier
    - Lower case letters are local to the file
    - Upper case letters work globally

Marks — accessing

- `'a` goes to the line of mark a
- `` `a `` goes to the position of a
- `''` and `` `` `` go to your last line/position respectively
- `'.` / `` `. `` goes to the line/position of the last edit

Demo

Search

- `/` search forwards
- `?` search backwards
- `n` next result
- `N` previous result

Search

- / search forwards
- ? search backwards
- n next result
- N previous result
- * search forwards, for word under cursor
- # search backwards, for word under cursor

Demo

Replace

- `:s/pattern/replacement`

  acts on the current line
- `:%s/pattern/replacement`

  acts on the current file
- `:'<,'>s/pattern/replacement`

  acts on the current selection

Replace

- `:s/pattern/replacement`
  acts on the current line
- `:%s/pattern/replacement`
  acts on the current file
- `:'<,'>s/pattern/replacement`
  acts on the current selection
- You can use regex: `\(...\)` to match, `\1` to reference

Replace

- `:s/pattern/replacement`
  acts on the current line
- `:%s/pattern/replacement`
  acts on the current file
- `:'<,'>s/pattern/replacement`
  acts on the current selection
- You can use regex: `\(...\)` to match, `\1` to reference
- The general syntax is
  `:[range]s/{pattern}/{replacement}/[flags] [count]`

  - 3 of the flags are: `c` to confirm each substitution, `g` to replace all occurrences on a line, and `i` to ignore case

Demo

Registers

- Like marks, just for your text.
- You can basically save text to a named spot.
- "a to "z save the selected text set a register.
- "A to "Z add the selection to the register

Registers

- Like marks, just for your text.
- You can basically save text to a named spot.
- `"a` to `"z` save the selected text set a register.
- `"A` to `"Z` add the selection to the register
- You need to add an action, e.g.
  - `"ay` yanks (copies) an *active selection* to register `a`
  - `"ayvap` yanks (copies) the *current paragraph* to register `a`
  - `"ap` pastes the content of register `a`
  - `"Ayiw` adds the *current word* to register `a`

Demo

Macros (mini)

. repeats the last action

Demo

Recording a named macro

- qa starts recording a macro saved to a
- *do the things*^{TM}
- q to stop recording

Using a named macro

- `@a` to execute macro `a`
- `@@` to execute the last macro
- `22@a` to execute `a` 22 times...

Editing an existing macro

- You can just use the register functionality
- `"ap` pastes the content of macro `a`
- Edit, select new macro content
- `"ay` writes to the macro `a`

A named macro — example

```
qs $ S ) Ret "ayib dab x ^ "aP a : Spc Esc ^ <down> q
```

A named macro — example

```
qs $ S ) Ret "ayib dab x ^ "aP a : Spc Esc ^ <down> q
```

What does that do!?

- `$` go to end of line
- `S )` search backwards for `)`
- `Ret` go to the first match
- `"ayib` copy the content of the parenthesis to register `a`
- `dab` delete the parenthesised content
- `x` delete the current character
- `^` go to the start of the content
- `"aP` paste content of register `a`
- `a : Spc Esc` enter insert mode after the content, and add ": " before returning to normal mode
- `^` go to the start of the content

A named macro — example explained

Ok, what does that *really* do?

- Find the last parenthesised content in a line, take it out and bring it to the front

If one called it on the following line for example

```
I need to make some slides (Due Thursday) on vim
```

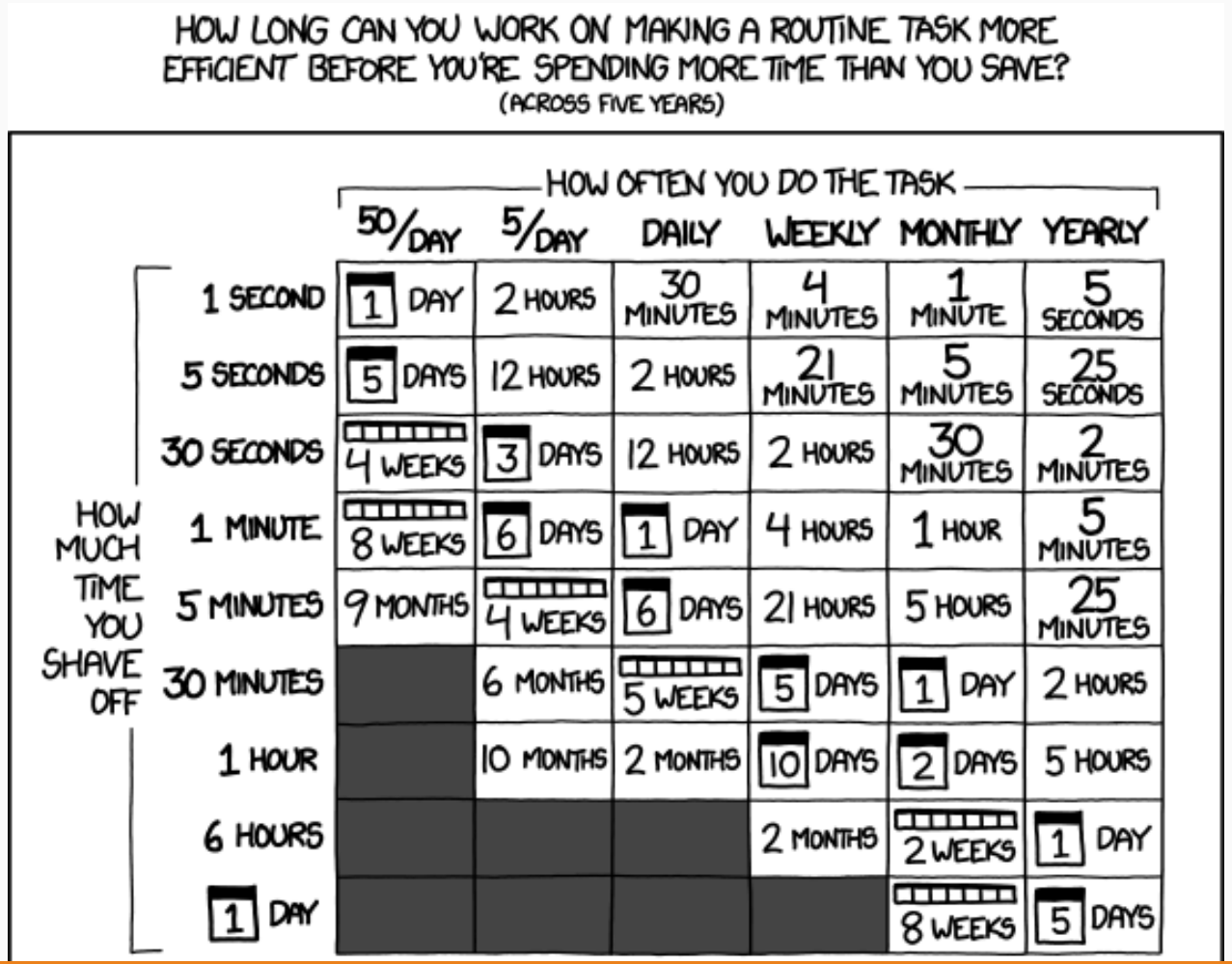
becomes

```
Due Thursday: I need to make some slides on vim
```

Demo

This seems like a lot of work

It is, but it's easy to gradually build your skill. We spend so much time editing text that I think it's worth using a method which improves the experience, even if it takes a bit of time to learn.



Resources

- A nice big Vim Cheat Sheet
- Vim functions overlayed on a keyboard
- Interactive Vim tutorial: https://www.openvim.com/
- Vim navigation in Firefox: tridactyl (add-on page)
- Vim navigation in Chrome: https://vimium.github.io/
- Blog post: Everyone Who Tried to Convince Me to use Vim was Wrong
- `vimtutor` is a little tool that comes bundled with vim

Q&A