An introduction to the *grammar* of

vim!

Hands-on with vim

- Let's use vim to work on a Markdown file
  - Markdown is a plaintext file format for writing
  - Commonly used in programming projects' README files and documentation
  - Easily converts to other formats such as HTML and PDF

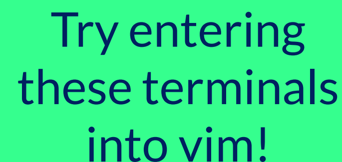
- To download the Markdown file (capital **L** important):
  - `learncli$ curl -L http://bit.ly/markdown-file > example.md`

- Then try running:
  - `vim example.md`

In Normal Mode, **vim** is driven by a grammar!

```
command     -> cursor_to

cursor_to   -> location

location    -> line-below | line-above | char-before | char-after

line-below -> 'j'

line-above -> 'k'

char-left   -> 'h'

char-right -> 'l'
```

Let's begin with a *small* subset of the grammar and grow it...



There are *lots* of location terminals in **vim**!

```
command      ->   cursor_to

cursor_to    ->   LOCATION
```

To keep the information on the slides manageable, we're going to cheat with this all caps convention that assumes there are additional rules here not shown (in table).

These are some of the most commonly useful location keys (terminals) in vim's little language.

Location	Terminal
line below	j
line above	k
char left	h
char right	l
first char of line	^
last char of line	$
next word	w
previous word	b
next end of word	e
find next / prev [c]har in line	f[c] / F[c]
before next / prev [c]har in line	t[c] / T[c]
toggle surrounding (, {, [, ...	%
specific line # of file	[line #]G
last line of file	G
search for "foo" (regexp)	/foo[enter]
next match of last search	n
previous match of last search	N

Travel wisely.

Crawling

up / down / left / right	j / k / h / l

Walking

next word	w
previous word	b
next end of word	e
toggle surrounding (, {, [, ...	%

Driving

find next / prev [c]har in line	f[c] / F[c]
to before next / prev [c]har in line	t[c] / T[c]
repeat last f or t location command	;
first char of line	^
last char of line	$
next occurrence of word under cursor	*

Teleporting

search forward for "foo" (regexp)	/foo[enter]
search backward for "foo" (regexp)	?foo[enter]
next match of last search	n
previous match of last search	N
specific line # of file	[line #]G
last line of file	G

Operations carry out actions on your text.

```
command     ->   cursor_to | operation

cursor_to   ->   LOCATION

operation   ->   verb cursor_to

verb        ->   change | delete | yank
change      ->   'c'
delete      ->   'd'
yank        ->   'y'
```

A command is *either* a cursor_to motion *OR* an operation.

An operation is a verb followed by a cursor_to.

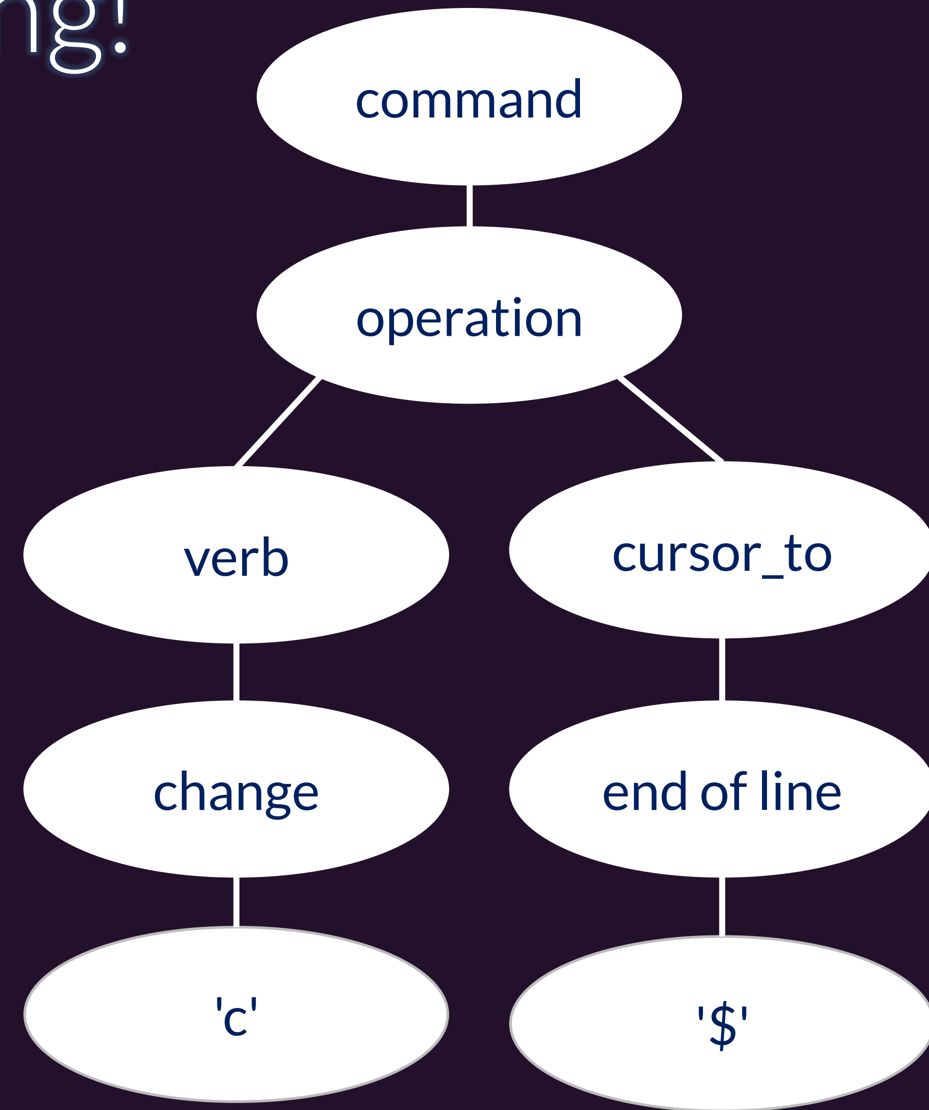
Change - cuts text, transitions to insert mode

Delete - cuts text

Yank - copies text

Let's generate a command string!

```
command    ->   cursor_to | operation

cursor_to ->    LOCATION

operation ->    verb cursor_to

verb       ->   change | delete | yank
change     ->   'c'
delete     ->   'd'
yank       ->   'y'
```



"Change from cursor to end of line."

Our grammar now has two high-level commands!

```
command      ->   cursor_to | operation

cursor_to    ->   LOCATION

operation    ->   VERB cursor_to
```

Verb	Terminal
change	c
delete	d
yank	y

Location	Terminal
line below	j
line above	k
char left	h
char right	l
first char of line	^
last char of line	$
next word	w
previous word	b
next end of word	e
find next / prev [c]har in line	f[c] / F[c]
before next / prev [c]har in line	t[c] / T[c]
toggle surrounding (, {, [, ...	%
specific line # of file	[line #]G
last line of file	G
search for "foo" (regexp)	/foo[enter]
next match of last search	n
previous match of last search	N

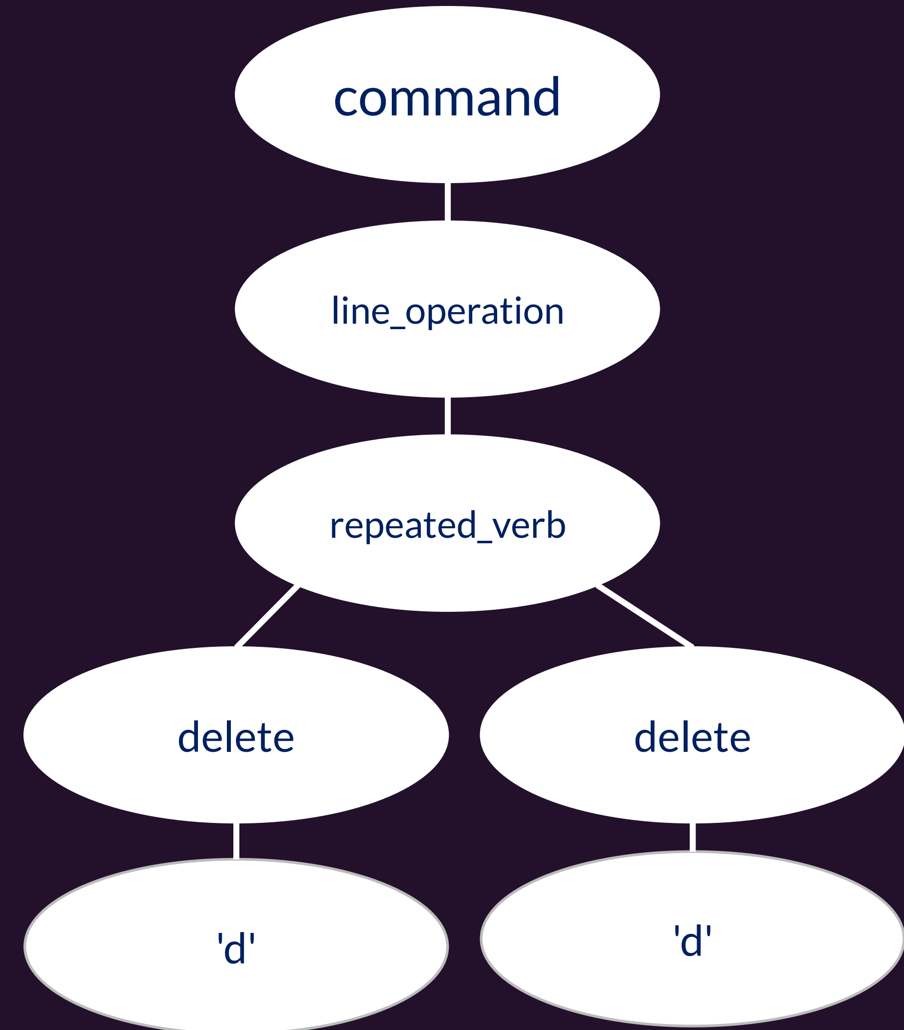
Line operations apply a verb to the whole line.

```
command        ->   cursor_to | operation | line_operation

cursor_to      ->   LOCATION

operation      ->   VERB cursor_to

line_operation ->   repeated_verb

repeated_verb  ->   delete delete |
                    change change |
                    yank yank
```

A repeated_verb is either
a delete followed by a delete OR
a change followed by a change OR
a yank followed by a yank.

Let's generated a line_operation string!

```
command        ->   cursor_to | operation | line_operation

cursor_to      ->   LOCATION

operation      ->   VERB cursor_to

line_operation ->   repeated_verb

repeated_verb  ->   delete delete |
                    change change |
                    yank yank
```



Notice the grammar composes concepts!

```
command          ->    cursor_to | operation | line_operation

cursor_to        ->    LOCATION

operation        ->    VERB cursor_to

line_operation   ->    REPEATED_VERB
```

An operation composes the concept of moving your cursor with an action verb.

It's so common you want to delete or change a whole line there's a convention of repeating a verb twice to do so.

Composition gives you combinatoric superpowers.

The # of commands you know is multiplier of your **VERBS** x **LOCATIONS**.

You can repeat / "scale" these commands, too!

```
command          -> n_repeats? (cursor_to | operation | line_operation)

cursor_to        -> LOCATION

operation        -> VERB cursor_to

line_operation   -> REPEATED_VERB

n_repeats        -> digit | digit n_repeats

digit            -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'  | '9'
```

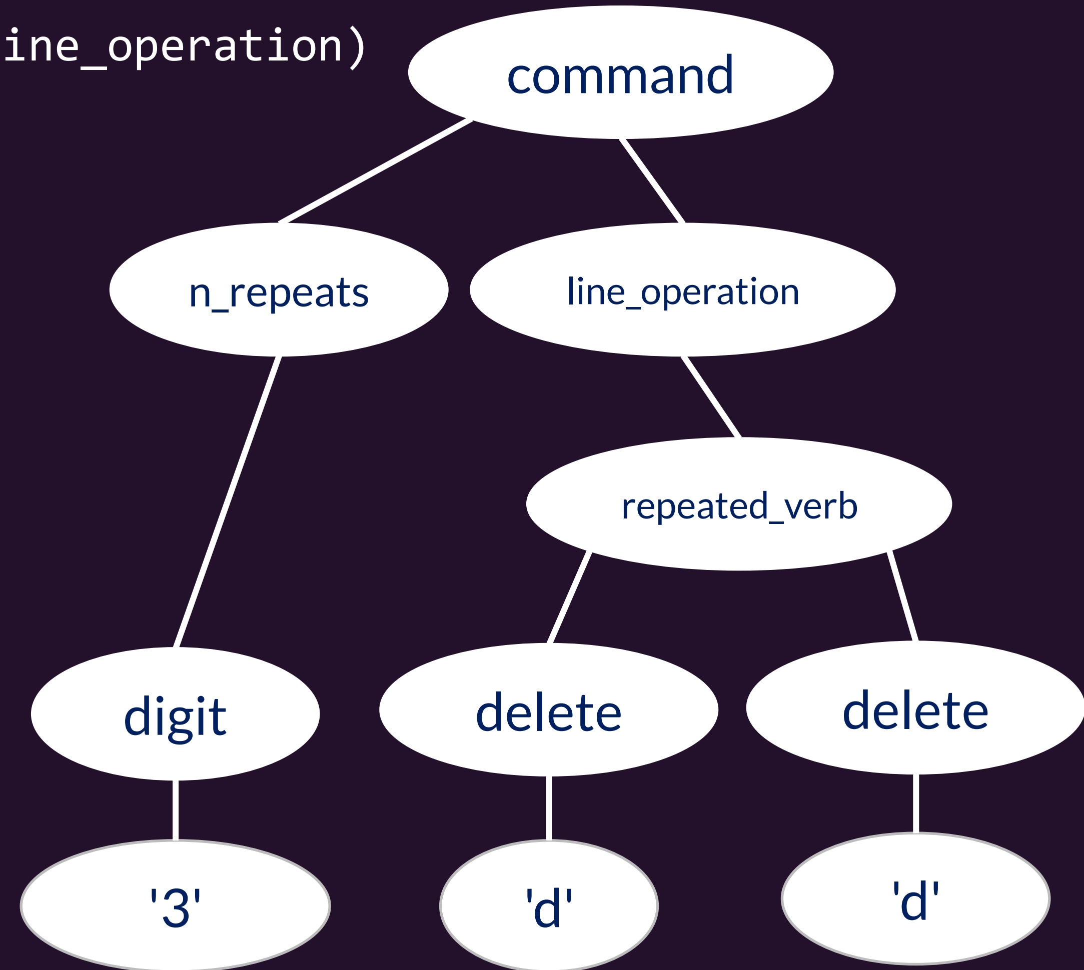
The question mark is syntactical sugar for "*optional*".

You can read this as "a command is optionally an n_repeats followed by either a..."

Let's generate a repeated line_operation string!

```
command        -> n_repeats? (cursor_to | operation | line_operation)

cursor_to      -> LOCATION

operation      -> VERB cursor_to

line_operation -> REPEATED_VERB
```

The command deletes three lines.



Changing to Insert Mode

```
command -> n_repeats? (cursor_to | operation | line_operation | to_insert_mode)

cursor_to      -> LOCATION

operation      -> VERB cursor_to

line_operation -> REPEATED_VERB

to_insert_mode -> insert | insert_below | append | ...
insert         -> 'i'
insert_below   -> 'o'
append         -> 'a'
...
```

To Insert Mode	Key
insert	i
insert at start of line	I
insert new line below	o
insert new line above	O
append after cursor	a
append at end of line	A
change to end of line	C

vim Grammar - Text Objects

```
command       -> CURSOR_TO | operation | LINE_OPERATION | TO_INSERT_MODE

operation     -> N_TIMES? VERB CURSOR_TO | VERB (inside | around) text_object

inside         -> 'i'

around         -> 'a'

object         -> surrounding | word

surrounding  -> ')' | ']' | '}' | '"' | '\'" | '`' | '>'

word             -> 'w'
```

Text Object Operation Examples

"Change Inside Parentheses"

Before: `foo(1, 2)`

Command: `ci)`

After: `foo( )` (in insert mode)

"Change Around Parentheses"

Before: `foo(1, 2)`

Command: `ca)`

After: `foo` (in insert mode)

Replaying the Last *Operation!*

```
command    -> n_repeats? (cursor_to | operation | line_operation | TO_INSERT_MODE | misc)

cursor_to         -> LOCATION

operation         -> VERB cursor_to

line_operation    -> REPEATED_VERB

misc              -> replay_last_op

replay_last_op    -> '.'
```

Wow! Think about this!

The last operation string you formed can be replayed!
(Including line operations.)

Normal Mode **vim** Grammar Cheat Sheet

```
command -> N_REPEATS? (cursor_to | operation | line_operation |TO_INSERT_MODE|MISC)

cursor_to -> LOCATION

                                         inside    around
operation -> VERB cursor_to | VERB ('i'|'a') OBJECT

line_operation -> VERB VERB
```

Location	Terminal
line below	j
line above	k
char left	h
char right	l
first char of line	^
last char of line	$
next word	w
previous word	b
next end of word	e
find next / prev [c]har in line	f[c] / F[c]
before next / prev [c]har in line	t[c] / T[c]
toggle surrounding (, {, [, ...	%
specific line # of file	[line #]G
last line of file	G
search for "foo" (regexp)	/foo[enter]
next match of last search	n
previous match of last search	N

Verb	Terminal
change	c
delete	d
yank	y

Object	Terminals
surrounding pair of	" ' }])

To Insert Mode	Terminal
insert	i
insert at start of line	I
insert new line below	o
insert new line above	O
append after cursor	a
append at end of line	A
change to end of line	C

Misc	Key
undo / redo	u / ctrl+r
replay last operation	.
paste after / before	p / P
delete a character	x
backspace a character	X
replace a character	r<char>

Save (*write*)	:w
Save + Quit	ZZ
Quit no Save	ZQ

Hands-on: Updating .bash_profile

- Open your shell configuration file in vim: **`learncli$ vim /mnt/learncli/.bash_profile`**
- Navigate down to **`# Global git config`**.
- Delete the leading #'s to uncomment the four export lines.
- Inside the "'s for AUTHOR_NAME and COMMITTER_NAME insert your first and last name.
- Inside the "'s for AUTHOR_EMAIL and COMMITTER_EMAIL insert the email address associated with your GitHub account. If you don't have a GitHub account yet, use your UNC email address and register for a GitHub account using it.
- Save and exit your file. Then run: **`learncli$ source /mnt/learncli/.bash_profile`**
- You shouldn't see any errors!