



Introduction to Vim



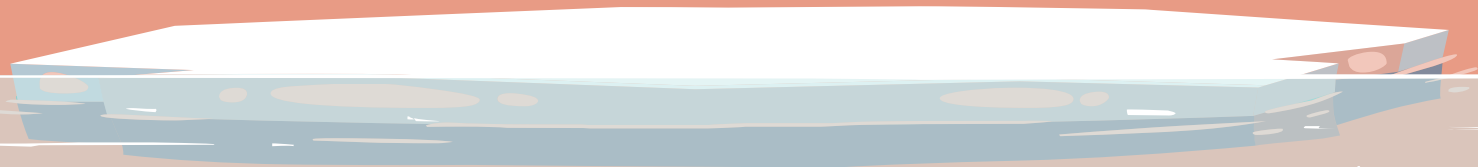


Agenda

- **Vim vs IDEs**
- **Vim Modes**
 - Normal
 - Insert
- **Horizontal Movement**
- **Vertical Movement**
- **Editing Commands**
- **Vim Modes (Cont.)**
 - Visual
 - Command line
- **Repetition (Dot Command)**



Text Editors and IDEs





Text Editor

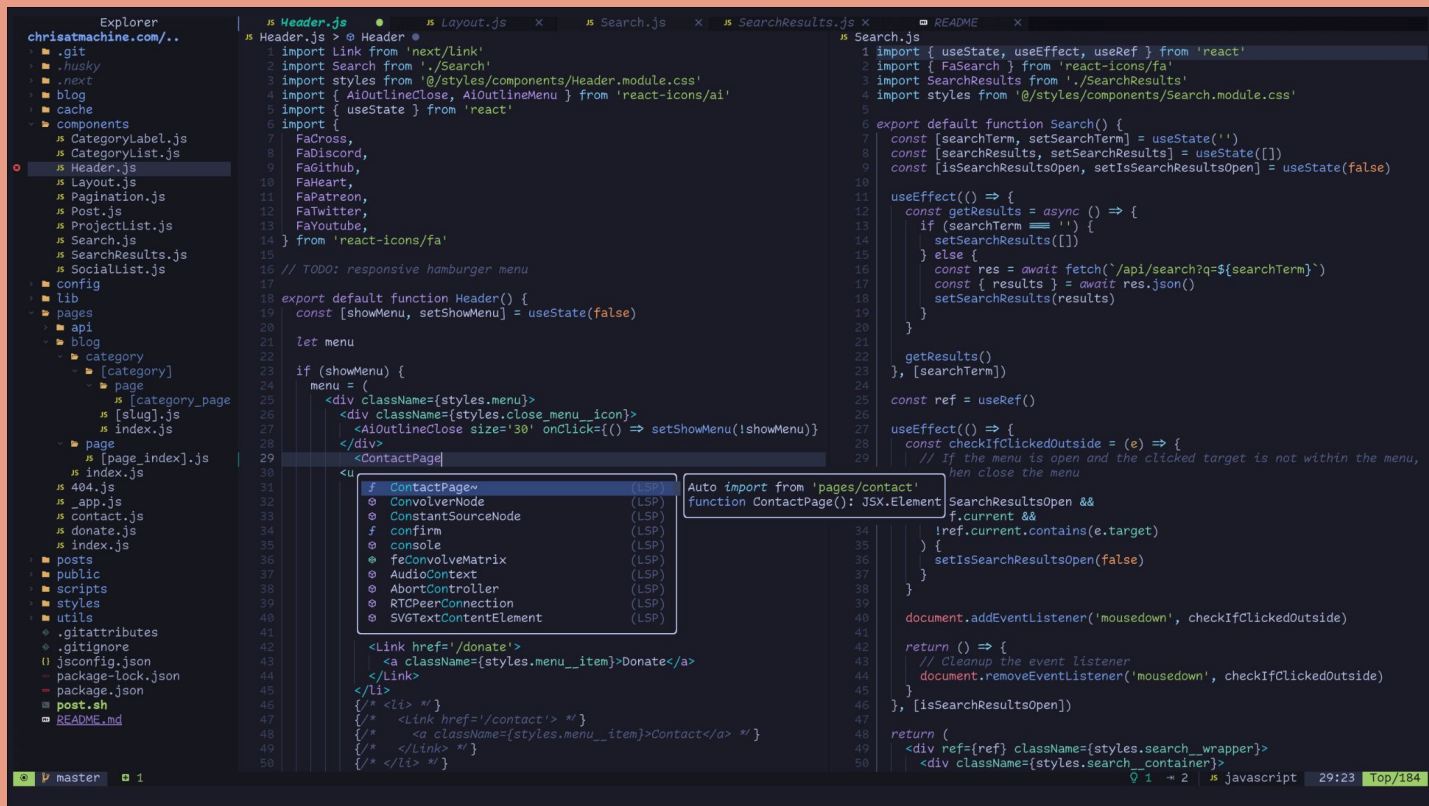
- Lightweight
- Extensions
- Command Line Integration
- Very efficient
- Can be configured to work with any language



```
[No Name] [unix] (01:59 01/01/1970)
```

0,0-1 All

Text Editor (customized)



The screenshot displays a customized text editor interface. On the left is a file explorer showing a project structure with folders like .git, .husky, .next, blog, cache, components, config, lib, pages, api, and blog, and files like 404.js, app.js, contact.js, donate.js, index.js, posts, public, scripts, styles, utils, gitattributes, gitignore, jsconfig.json, package-lock.json, package.json, post.sh, and README.md. The main editor area shows three files: Header.js, Layout.js, and Search.js. Header.js contains code for a responsive hamburger menu. Layout.js contains code for a search function. Search.js contains code for a search component. A tooltip is visible over the Search.js file, showing the auto-import path for the ContactPage component. The status bar at the bottom indicates the current file is javascript, line 29, column 23, and the top of the page is 184 lines long.

```
Header.js
1 import Link from 'next/link'
2 import Search from './Search'
3 import styles from '@styles/components/Header.module.css'
4 import { AiOutlineClose, AiOutlineMenu } from 'react-icons/ai'
5 import { useState } from 'react'
6 import {
7   FaCross,
8   FaDiscord,
9   FaGithub,
10  FaHeart,
11  FaPatreon,
12  FaTwitter,
13  FaYoutube,
14 } from 'react-icons/fa'
15
16 // TODO: responsive hamburger menu
17
18 export default function Header() {
19   const [showMenu, setShowMenu] = useState(false)
20
21   let menu
22
23   if (showMenu) {
24     menu = (
25       <div className={styles.menu}>
26         <div className={styles.close_menu_icon}>
27           <AiOutlineClose size='30' onClick={() => setShowMenu(!showMenu)} />
28         </div>
29         <ContactPage />
30       </div>
31     )
32
33     <Link href='/donate'>
34       <a className={styles.menu__item}>Donate</a>
35     </Link>
36
37     <Link href='/contact'>
38       <a className={styles.menu__item}>Contact</a>
39     </Link>
40   }
41 }
42
43 </div>
44 </div>
45 </div>
46 </div>
47 </div>
48 </div>
49 </div>
50 </div>
```

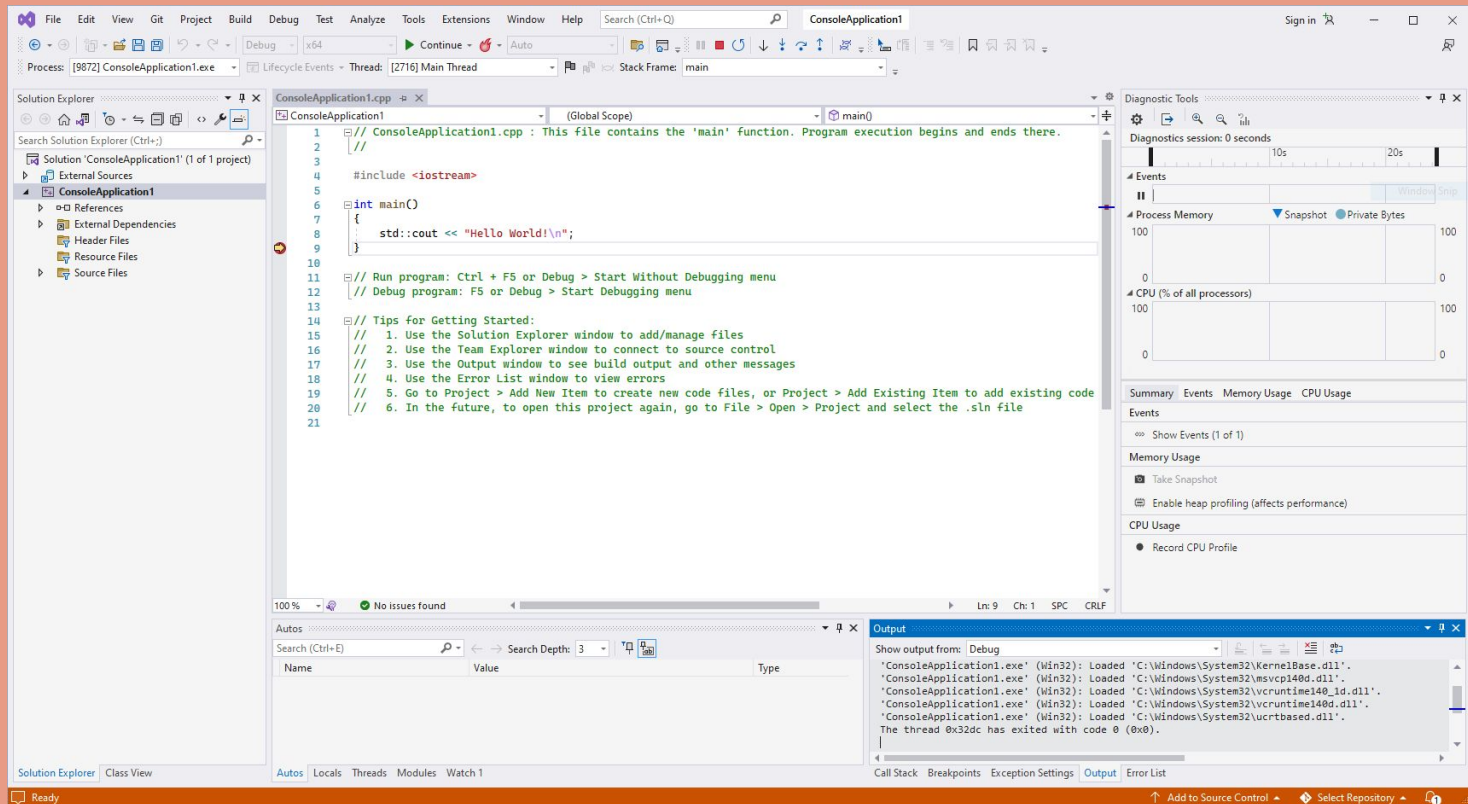
```
Search.js
1 import { useState, useEffect } from 'react'
2 import { FaSearch } from 'react-icons/fa'
3 import SearchResults from './SearchResults'
4 import styles from '@styles/components/Search.module.css'
5
6 export default function Search() {
7   const [searchTerm, setSearchTerm] = useState('')
8   const [searchResults, setSearchResults] = useState([])
9   const [isSearchResultsOpen, setIsSearchResultsOpen] = useState(false)
10
11   useEffect(() => {
12     const getResults = async () => {
13       if (searchTerm === '') {
14         setSearchResults([])
15       } else {
16         const res = await fetch(`/api/search?q=${searchTerm}`)
17         const { results } = await res.json()
18         setSearchResults(results)
19       }
20     }
21
22     getResults()
23   }, [searchTerm])
24
25   const ref = useRef()
26
27   useEffect(() => {
28     const checkIfClickedOutside = (e) => {
29       // If the menu is open and the clicked target is not within the menu,
30       // then close the menu
31       if (isSearchResultsOpen &&
32         !ref.current?.contains(e.target)) {
33         setIsSearchResultsOpen(false)
34       }
35     }
36
37     document.addEventListener('mousedown', checkIfClickedOutside)
38
39     return () => {
40       // Cleanup the event listener
41       document.removeEventListener('mousedown', checkIfClickedOutside)
42     }
43   }, [isSearchResultsOpen])
44
45   return (
46     <div ref={ref} className={styles.search_wrapper}>
47       <div className={styles.search_container}>
48         <input type='text' value={searchTerm} onChange={e => setSearchTerm(e.target.value)} />
49         <button type='submit'>Search</button>
50       </div>
51     </div>
52   )
53 }
```



Integrated Development Environments (IDEs)

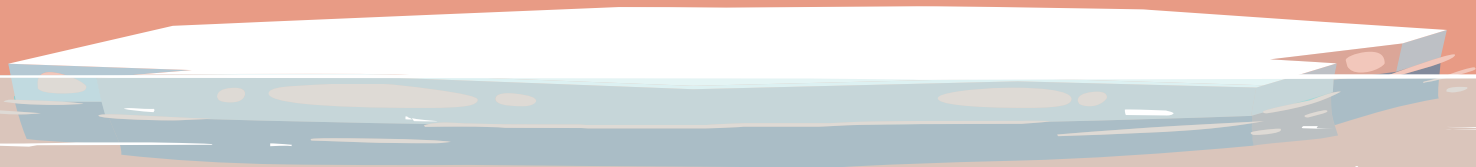
- Resource Heavy
- Specialised
- Sometimes you have to pay to get

Integrated Development Environments (IDEs)





Vim Modes





Vim Modes

- A mode is just a way to tell **Vim** how it should interpret your keystrokes.
- Most Common Modes:
 - Normal
 - Insert
 - Visual
 - Cmdline



Moving in Vim





Moving in Vim

- Vim is optimized for the touch typist.
- Keep your fingers on the Home Row

~	! 1	@ 2	# 3	\$ 4	% 5	^ 6	& 7	* 8	(9) 0	-	=	Backspace
Tab	Q	W	E	R	T	Y	U	I	O	P	{	}	
Caps Lock	A	S	D	F	G	H	J	K	L	:	"	Enter	
Shift	Z	X	C	V	B	N	M	<	>	?	Shift		
Ctrl	Win	Alt							Alt	Win	Menu	Ctrl	



Moving in Vim

``h``

Arrow-Left

``j``

Arrow-Down

``k``

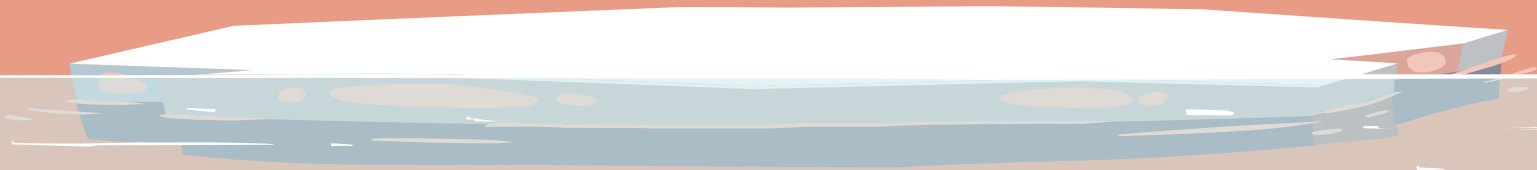
Arrow-Up

``l``

Arrow-Right



Horizontal Movement





Horizontal Movement

<code>`w`</code>	Move forward to the beginning of the next word
<code>`W`</code>	Move forward to the beginning of the next WORD
<code>`b`</code>	Move backward to beginning of the previous word
<code>`B`</code>	Move backward to beginning of the previous WORD
<code>`e`</code>	Move forward one word to the end of the next word
<code>`E`</code>	Move forward one word to the end of the next WORD
<code>`ge`</code>	Move backward to end of the previous word
<code>`gE`</code>	Move backward to end of the previous WORD

- ★ **word** - A **word** consists of a sequence of letters, digits and underscores, or a sequence of other non-blank characters, separated with white space
- ★ **Word** - A **WORD** consists of a sequence of non-blank characters, separated with white space. An empty line is also considered to be a **WORD**.



Horizontal Movement

``f {char}`` Move forward to the given {char}

``F {char}`` Move backward to the given {char}

``t {char}`` Move forward to before the given {char}

``T {char}`` Move backward to before the given {char}

``;`` Repeat the last search in the same line using the same direction

``,`` Repeat the last search in the same line using the opposite direction



Vertical Movement





Vertical Movement

<code>`gg`</code>	Go to the first line
<code>`G`</code>	Go to the last line
<code>`{`</code>	Jump to the previous paragraph
<code>`}`</code>	Jump to the next paragraph
<code>`%`</code>	Jump to the matching parenthesis
<code>`Ctrl+d`</code>	Scroll down half a page
<code>`Ctrl+u`</code>	Scroll up half a page



Editing





Deal With Files and Buffers

<code>`e {filename}`</code>	Opens {filename} to edit
<code>`r {filename}`</code>	Inserts the content of {filename} into the current file.
<code>`ls`</code>	List all opened buffers
<code>`q`</code>	Quit (fails if anything has changed)
<code>`q!`</code>	Quit, Discard any changes
<code>`qa`</code>	Quit all open buffers
<code>`w`</code>	Write buffer to disk
<code>`wa`</code>	Write all open buffers
<code>`wq`</code>	Write buffer to disk and quit
<code>`wqa`</code>	Write buffer to disk and quit all open buffers

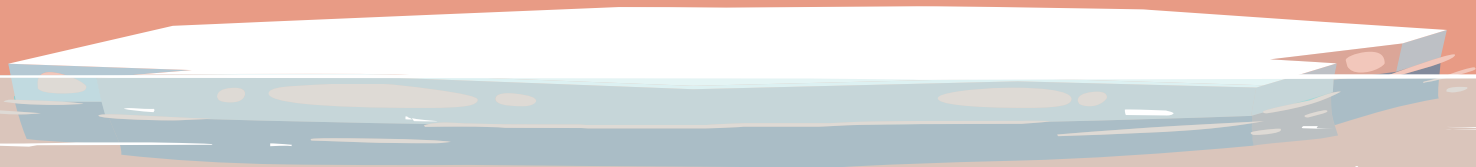


Basic Editing Commands

<code>`r`</code>	Replace a single character
<code>`R`</code>	Replace more than one character
<code>`x`</code>	Delete character
<code>`X`</code>	Delete character before cursor
<code>`p`</code>	Put the clipboard after cursor
<code>`P`</code>	Put the clipboard before cursor
<code>`.`</code>	Repeat last command



Undo and Redo





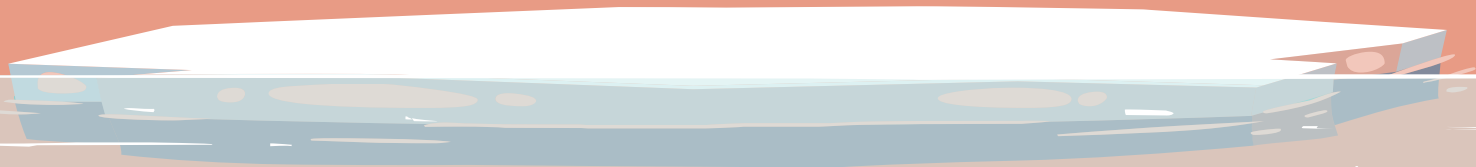
Undo and Redo

- Vim undo chunks
- A chunk is just the text you write when you go to insert mode until you return back to normal mode
- It may be single character or multiple lines or a whole file

u	Undo	change
<Ctrl-r>	Redo	change



Vim Grammer





Vim Grammer

verb + noun



VERBS

- Verbs are operators
- An operator just perform an action on some text
- These are the most used ones

y	Yank text (copy)
d	Delete text and save to register
c	Delete text, save to register, and start insert mode
gU	Convert text to UPPER CASE
gu	Convert text to lower case



NOUNS (Motions)

- Nouns can be Motions
- Motions are used to move around in Vim
- This is a list of popular vim motions

<code>`w`</code>	Move forward to the beginning of the next word
<code>`}`</code>	Jump to the next paragraph
<code>`\$`</code>	Go to the end of the line
<code>`0`</code>	Go to the beginning of the line



NOUNS (Motions) cont.

- Suppose you have this expression:

```
const learn = "vim";
```

- To yank everything from your current location to the end of the line -> y\$
- To delete from your current location to the beginning of the next word -> dw
- To change from your current location to the end of the current paragraph, say
-> c}



NOUNS (Motions) cont.

- Motions also accept count number as argument
- To yank two characters to the left -> y2h
- To delete the next two words -> d2w
- To change the next two lines -> c2j



NOUNS (Text Objects)

- Texts often come structured ("`"`", `()`, `<h1>tag<h1>`, `Word`, `P`,...)
- Vim has a way to capture this structure with text objects.
- Text objects are used with operators, same as motions
- There are two types of text objects: **inner** and **outer** text objects.



NOUNS (Text Objects) cont.

i + object -> Inner text object

a + object -> Outer text object



NOUNS (Text Objects) cont.

- **Inner text object** selects the object inside *without* the white space or the surrounding objects.
- **Outer text object** selects the object inside *including* the white space or the surrounding objects.
- To delete the text inside the parentheses *without* deleting the parentheses -> di(
- To delete the parentheses and the text inside -> da(



NOUNS (Text Objects) cont.

- **Example Usage:**

- diw -> **delete word under cursor**
- daw -> **delete word under cursor and the space after or before**
- cit -> **change text inside tag**
- cat -> **change text inside tag and the tag itself**

<code>`w`</code>	A word
<code>`p`</code>	A paragraph
<code>`s`</code>	A sentence
<code>`(or)`</code>	A pair of ()
<code>`{ or }`</code>	A pair of { }
<code>`[or]`</code>	A pair of []
<code>`< or >`</code>	A pair of < >
<code>`t`</code>	XML tags
<code>`" `</code>	A pair of " "
<code>`' `</code>	A Pair of ' '
<code>` `</code>	A pair of ` `



Basic Editing Commands (Cont.)

<code>`d`</code>	Delete up to motion/text obj.
<code>`y`</code>	yank up to motion/text obj.
<code>`c`</code>	Change up to motion/text obj.
<code>`s`</code>	Delete char, then start insert mode



Basic Editing Commands (Cont.)

- In general, by typing an operator command twice, Vim performs a linewise operation for that action.

<code>`yy`</code>	Yank (copy) a line
<code>`dd`</code>	Delete line
<code>`cc`</code>	Change line
<code>`gugu`</code>	uppercase line
<code>`gugu`</code>	Lowercase line



Basic Editing Commands (Cont.)

<code>`gu`</code>	Change to lowercase up to motion/text obj.
<code>`gu`gu or <code>guu`</code></code>	Change the current line to lowercase
<code>`gU`</code>	Change to uppercase up to motion/text obj.
<code>`gU`gU or <code>gUU`</code></code>	Change the current line to upper case
<code>`>`</code>	Indent one shiftwidth up to motion/text obj.
<code>`>>`</code>	indent line one shiftwidth
<code>`<`</code>	De-indent one shiftwidth up to motion/text obj.
<code>`<<`</code>	De-indent line one shiftwidth
<code>`=`</code>	Auto indent up to motion/text obj.
<code>`==`</code>	Auto indent line



BREAK

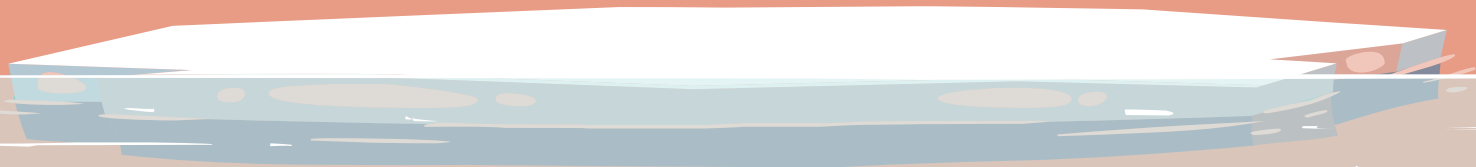


RECAP...

- Why VIM?
- VIM modes
- Horizontal Movement
- Vertical Movement
- Files and Buffers
- Undo and Redo
- VIM Grammar (verb + noun)
- Operators
- Motions
- Text objects
- NORMAL Mode



Insert Mode





INSERT Mode

- Default mode in most editors
- What you type is what you get



INSERT Mode (HOW to enter?)

<code>`i`</code>	Insert text before the cursor
<code>`I`</code>	Insert text before the first non-blank character of the line
<code>`a`</code>	Append text after the cursor
<code>`A`</code>	Append text at the end of line
<code>`o`</code>	Starts a new line below the cursor and insert text
<code>`O`</code>	Starts a new line above the cursor and insert text
<code>`s`</code>	Delete the character under the cursor and insert text
<code>`S`</code>	Delete the current line and insert text
<code>`gi`</code>	Insert text in same position where the last insert mode was stopped
<code>`gI`</code>	Insert text at the start of line (column 1)



INSERT Mode, cont...

- You can exit insert mode using `<Esc>` key
- When you make a typing mistake, you don't need to type `<Backspace>` repeatedly.
- You can also delete several characters at a time while in insert mode

```
`Ctrl-H` Delete one character  
`Ctrl-W` Delete one word  
`Ctrl-U` Delete the entire line
```



INSERT Mode, cont...

- Vim can execute a normal mode command while in insert mode
- When you press **ctrl-o** while you are in insert mode, you will be able to perform one normal mode command
- For example

```
`Ctrl-O zz`      Center window  
`Ctrl-O D `      Delete from current location to the end of the line
```



Visual Mode





Visual Mode

Vim has three different visual modes

<code>`v`</code>	Character-wise visual mode (Visual mode)
<code>`V`</code>	Line-wise visual mode (Visual Line mode)
<code>`Ctrl-v`</code>	Block-wise visual mode (Visual Block mode)



Visual Mode, cont...

- Character-wise visual mode works with individual characters.
- Line-wise visual mode works with lines.
- Block-wise visual mode works with rows and columns.



Visual Mode, cont...

- While you are inside a visual mode, you can switch to another visual mode by pressing either **v**, **V**, or **Ctrl-V**.
- For example, if you are in **line-wise** visual mode and you want to switch to **block-wise** visual mode, run **Ctrl-V**.
- While in a visual mode, you can expand the highlighted text block with Vim motions.



Visual Mode, cont...

- The visual mode shares many operations with normal mode.
- the grammar rule from normal mode, **verb + noun**, does not apply.
- The grammar rule in visual mode is

noun + verb



Dot Command





Dot Command

- Our work is repetitive by nature
- Vim is optimized for repetition
- We can always replay the last change with a single keystroke

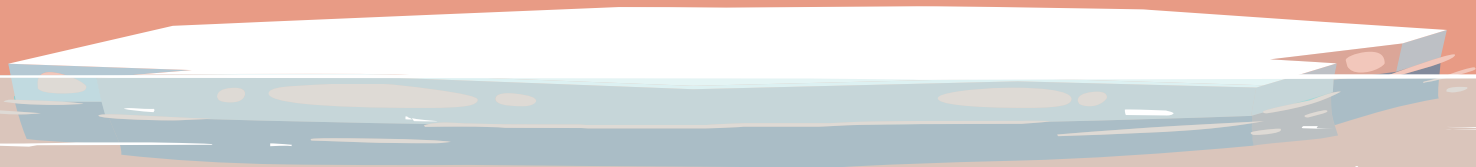


Dot Command, cont....

- The dot command lets us repeat the last change.
- So, What is a Change?
- Any time you update (add, modify, or delete) the content of the current buffer, you are making a change.



Cmdline Mode





Cmdline Mode

- In Cmdline mode, you can execute a wide range of commands to control various aspects of your editing session and manipulate the text in your files.
- Some common tasks you can perform in Ex mode include:
 - 1) File operations.
 - 2) Search and Replace.
 - 3) Copy, Cut and Paste.



Cmdline Mode, cont...

- It provides a way to interact with the editor through a command-line interface rather than using the traditional visual interface.
- Cmdline mode is invoked by typing `:` in normal mode, followed by entering the desired commands.



Thanks!

