Vim

Gentle Introduction

Ahmed Yasser
Tony Medhat

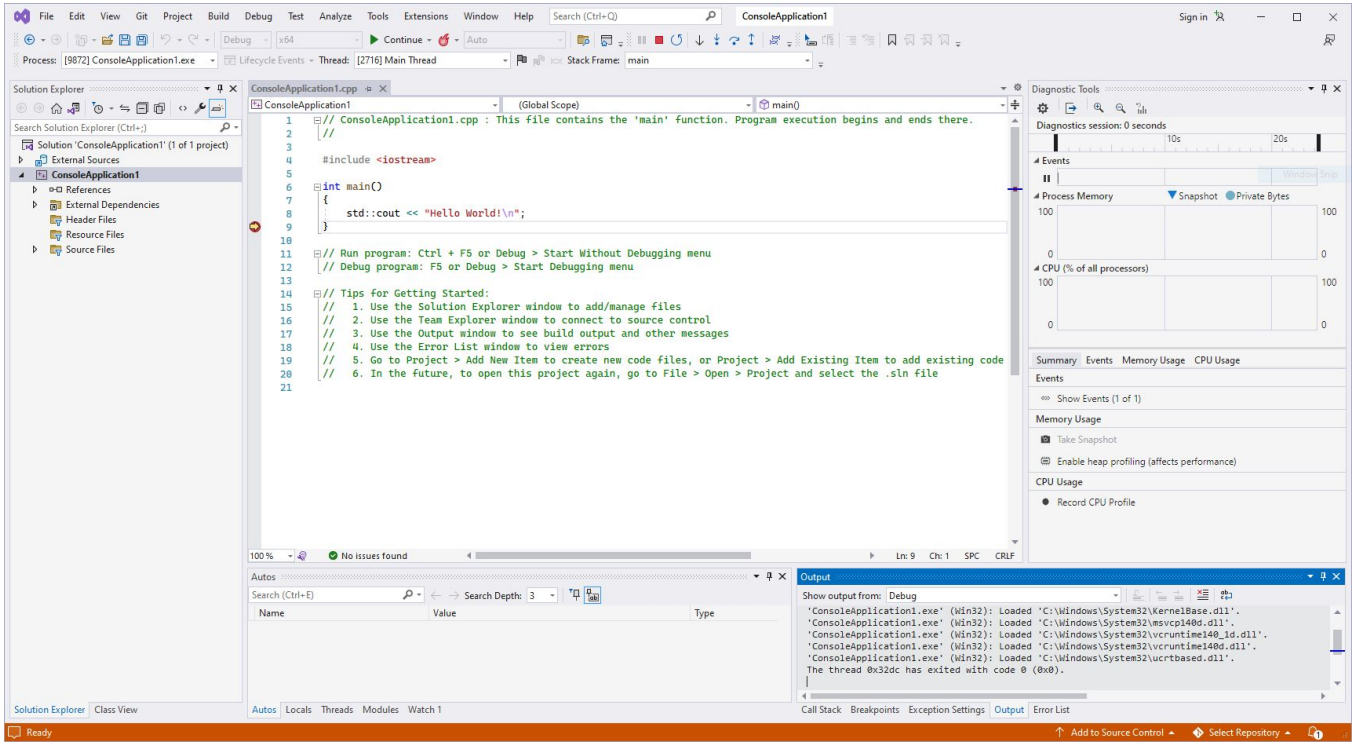
OSC
Linux
2024

Agenda

- Text Editors and IDEs
- Vim and NeoVim
- Vim Modes
- Vim Interface
- Vim Motions
- Vim Grammer
- Break
- Vim Modes (cont.)
- :Help
- LSP and DAP
- NeoVim Distributions

Text Editors and IDEs

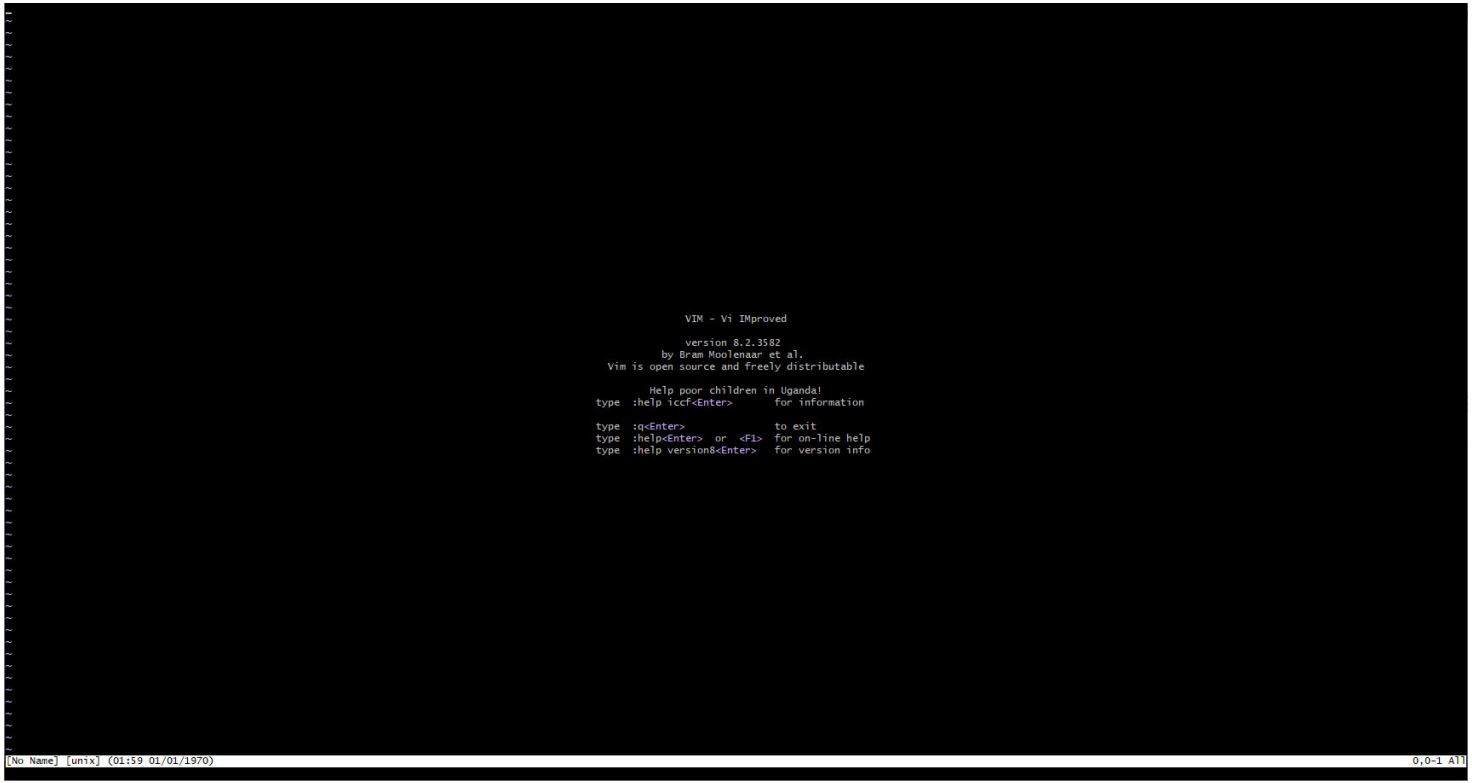
Integrated Development Environments



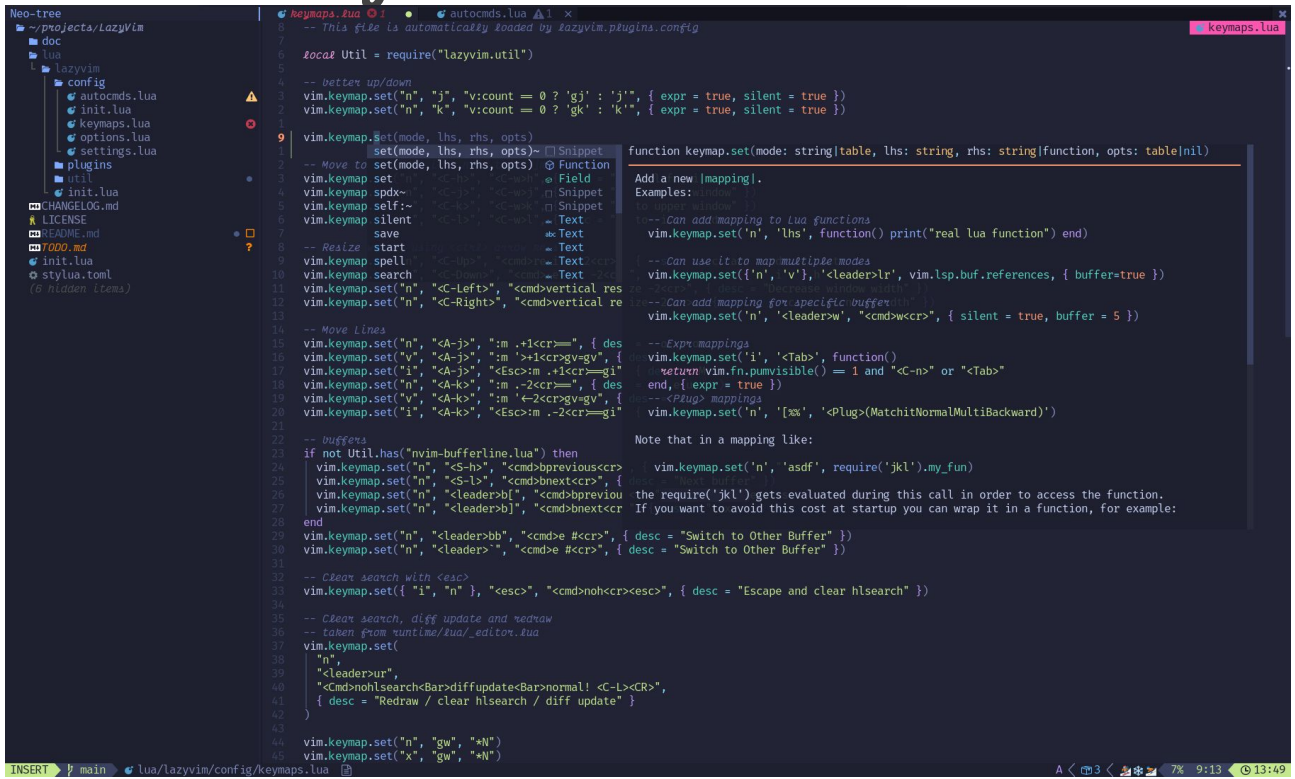
Integrated Development Environments (IDEs)

- Specialised
- Resource Heavy
- Plug and Play

Text Editor (Default)



Text Editor (LazyVim)



https://www.lazyvim.org/

Text Editors

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

Editing Experience

- Intellisense (Autocomplete)
- Code actions
- Diagnostics (i.e. errors, warnings, ...)
- Highlights
- Command line integration
- Refactoring


Vim

Vim

- *Vim* is a highly configurable text editor built to make creating and changing any kind of text very efficient.
- It is included as *"vi"* with most UNIX systems and LINUX.
- Initial release 2 November 1991; 32 years ago

NeoVim

NeoVim

- *NeoVim* is a hyperextensible *Vim*-based text editor.
- Initial release 1 November 2015; 8 years ago
- Builtin *LSP* client for semantic code inspection and refactoring (go-to definition, "find references", format, ...)
- Modern terminal features such as cursor styling, focus events, bracketed paste.
- Uses *Lua for configuration.*

Vim Modes and Interface

Vim Modes

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

Vim Interface

1) Buffer Name (File Name).
2) Buffer Contents (File Contents).
3) Active Mode.
4) Active Command.

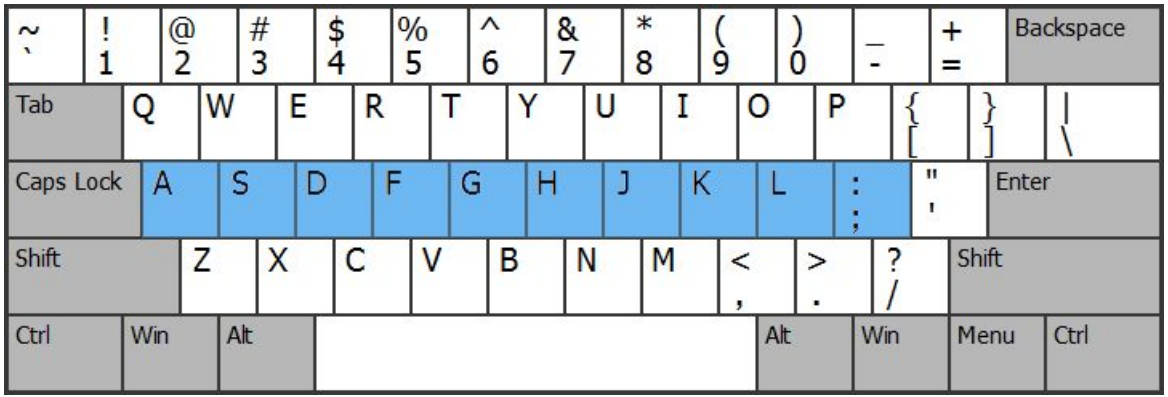
Moving in Vim

Quick Start

`i`	Enter **insert** mode
`<ESC><ESC>`	Return to **normal** mode
`:q`	Quit

Moving in Vim

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



https://kompirumpi.blogspot.com/2017/01/mengenal-home-row-keys-top-row.html

Moving in Vim

`h`	Arrow-Left
`j`	Arrow-Down
`k`	Arrow-Up
`l`	Arrow-Right

Horizontal Movement

Horizontal Movement

`w`	Move forward to the beginning of the next **word**
`W`	Move forward to the beginning of the next WORD
`b`	Move backward to the beginning of the previous **word**
`B`	Move backward to the beginning of the previous WORD
`e`	Move forward to the end of the next **word**
`E`	Move forward to the end of the previous WORD
`ge`	Move backward to the end of the previous **word**
`gE`	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

`gg`	Go to the first line
`G`	Go to the last line
`{`	Jump to the previous **paragraph**
`}`	Jump to the next **paragraph**
`Ctl-d`	Scroll down half a page
`Ctl-u`	Scroll up half a page

Editing

Basic Editing Commands

r	Replace character under cursor
x	Delete character under cursor
X	Delete character before cursor
p	Put the clipboard after cursor
P	Put the clipboard before cursor

Vim Grammer

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 UPPERCASE
`gu`	Convert text to lowercase

nouns (Motions)

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

`w`	Move forward to the beginning of the next **word**
`}`	Jump to the next **paragraph**
`$`	Jump to the **end of the line**
`0`	Jump to the **beginning of the line**
`^`	Jump to **first** char in the **beginning of the line**

Vim Grammer

```
Verb + Noun
```

- Suppose you have this expression:

```
const learn = “vim”;
```

y$	To yank everything from your current location to the **end of the line**
dw	To delete from your current location to the beginning of the next **word**

nouns (Motions)

- Motions also accept count number as argument.

`y2h`	Yank the next **two** characters to the **left**
`d2w`	Delete the next **two words**
`c2j`	Change the next **two** lines **down**

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)

i + object	Innertext object (Exclusive)
a + object	Outer text object (Inclusive)

nouns (Text Objects)

- 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.

`di(`	Delete innertext without deleting the **parentheses**
`da(`	Delete innertext and **parentheses**

nouns (Text Objects)

`diw`	Delete word under cursor
`daw`	Delete word under cursor and the space after/before
`cit`	Change text inside tag
`cat`	Change text inside tag and the tag itself

`w`	A word
`p`	A paragraph
`( or )`	A pair of ( )
`{ or }`	A pair of { }
`[ or ]`	A pair of [ ]
`< or >`	A pair of < >
`t`	XML Tag
`"`	A pair of " "
`'`	A pair of ' '
`` ` ``	A pair of ` `

Basic Editing Commands

d	Delete up to *{Motion|Text-Object}*
y	Yank up to *{Motion|Text-Object}*
c	Change up to *{Motion|Text-Object}*
s	Delete character under cursor, then enter insert mode

Basic Editing Commands

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

`yy`	Yank (copy) line under cursor
`dd`	Delete line under cursor
`cc`	Change line under cursor
`gUgU`	UPPERCASE line under cursor
`gugu`	lowercase line under cursor

Basic Editing Commands

`Y (y$)`	Yank (Copy) from cursor to the end of the line
`D (d$)`	Delete from cursor to the end of the line
`C (c$)`	Change from cursor to the end of the line

Basic Editing Commands

`>`	Indent one shiftwidth up to **{Motion|Text-Object}**
`>>`	Indent line under cursor by one shiftwidth
`<`	De-indent up to **{Motion|Text-Object}** by one shiftwidth
`<<`	De-indent line under cursor by one shiftwidth
`=`	Auto indent up to **{Motion|Text-Object}**
`==`	Auto indent line

Accidently losing a file because of pressing the wrong key in Vim

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 my be single character or multiple lines or a whole file

`U`	Undo change
`Ctl-r`	Redo change

Insert Mode

INSERT Mode

- Insert mode is like most editors.
- What you type is what you get.

INSERT Mode

`i`	Insert text before cursor
`I`	Insert text before the first non-blank character
`a`	Append text after cursor
`A`	Append text at the end of line
`o`	Begin a new line below the cursor, then enter insert mode
`O`	Begin a new line above the cursor, then enter insert mode
`s`	(Substitute) Delete character under cursor, then enter insert mode
`S`	(Substitute) Delete line under cursor, then enter insert mode
`gi`	Insert text in same position where the last insert mode was stopped
`gI`	Insert text at the start of line (column 1)

INSERT Mode

- 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.

`Ctl-h`	Delete one character to the **left**
`Ctl-w`	Delete one **word**
`Ctl-u`	Delete from the current cursor position to the beginning of the line

INSERT Mode

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

`Ctl-o zz`	Center cursor
`Ctl-o D`	Delete from current location to the end of the line

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

- The **Dot** command lets us repeat the last change.
- Any time you update (add, modify, or delete) the content of the current buffer, you are making a change.

Break

Cmdline Mode

Cmdline Mode

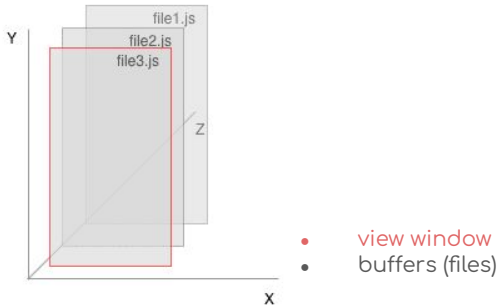
- In **Cmdline** mode, you can execute a wide range of commands to control various aspects of your editing session and manipulate the text in your files.
- Common tasks you can perform in **Cmdline** mode:
  - File operations.
  - Search and Replace.
  - Copy, Cut and Paste.

Cmdline Mode

- 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.

Windows and Buffers

`:e[dit] {filename}`	Opens *{filename}* to edit
`:r[ead] {filename}`	Inserts the content of *{filename}* into the current buffer.
`:ls`	List all opened buffers
`:b [N]`	Edit buffer *[N]* from the buffer list.
`:bn[ext]`	Go to **next** buffer in buffer list.
`:bp[revious]`	Go to **previous** buffer in buffer list.



- view window
- buffers (files)

https://dev.to/iggredible/using-buffers-windows-and-tabs-efficiently-in-vim-56jc

Windows and Buffers

`:q[uit]`	Quit current window (fails if buffer has unsaved changes)
`:q[uit]!`	Quit current window, Discard any changes
`:qa[ll]`	Exit Vim (fails if buffer has unsaved changes changed)
`:w[rite]`	Write buffer to disk
`:wa[ll]`	Write all open buffers to disk
`:wq`	Write buffer to dist and quit
`:wqa[all]`	Write buffer to disk and quit all open buffers

Visual Mode

Visual Mode

- *Vim* has three different visual modes:

`v`	Character-wise visual mode (Visual mode)
`V`	Line-wise visual mode (Visual Line mode)
`Ctl-v`	Blockwise visual mode (Visual Block mode)

Visual Mode

- 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

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

Visual Mode

- 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

:help

:help

- Open a window and display the help file in read-only mode.

```
:h[elp] {keyword}     Open help for {keyword}
```

User Manual

- Read the user manual.

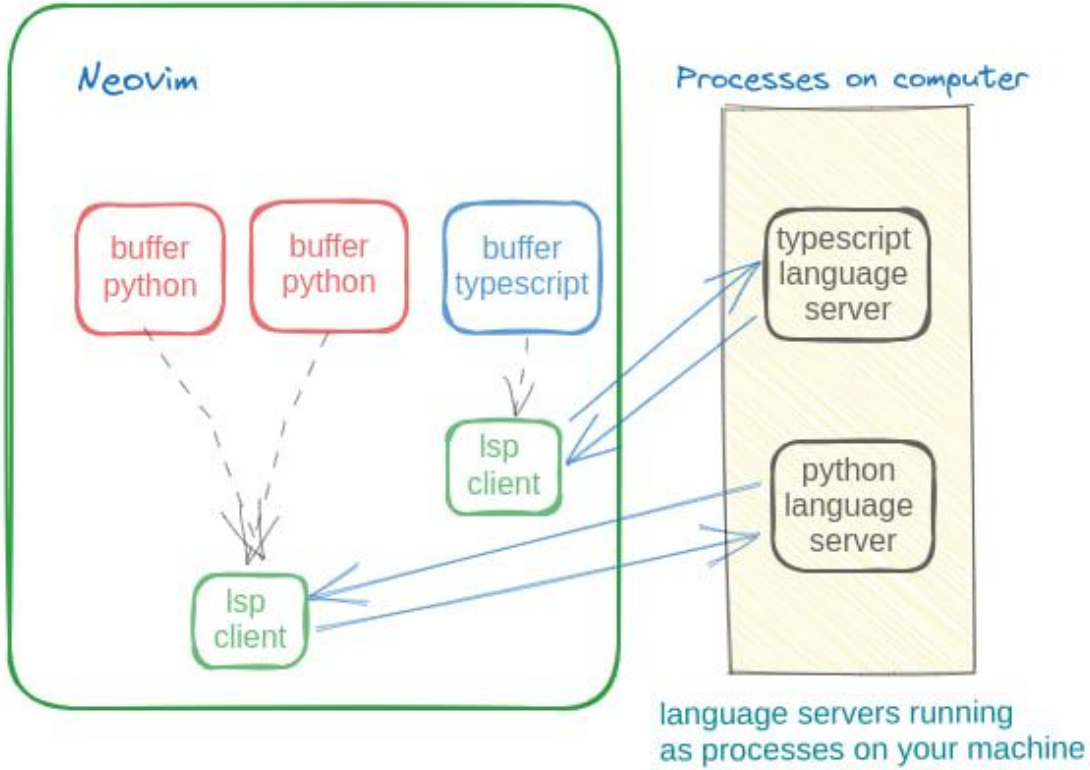
```
:help user-manual
```

Language Server Protocol (LSP)

Language Server Protocol (LSP)

- Adding features like auto complete, go to definition, or documentation on hover for a programming language takes significant effort.
- Traditionally this work had to be repeated for each development tool.
- A *Language Server* provides language-specific smarts and communicates with development tools.

Language Server Protocol (LSP)



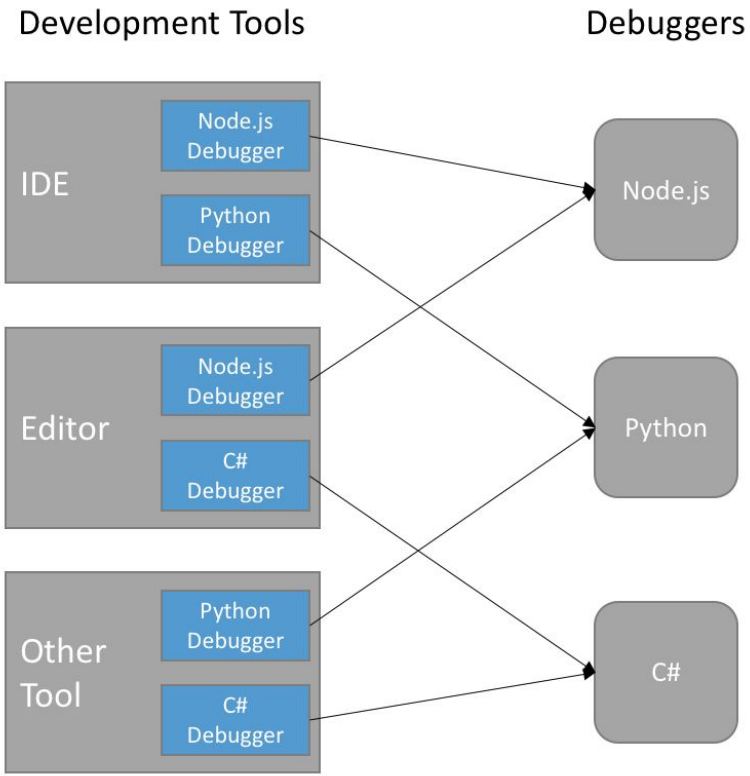
https://dev.to/rishavmngo/how-to-setup-lsp-in-neovim-nh1

Debug Adapter Protocol

Debug Adapter Protocol (DAP)

- The *Debug Adapter Protocol* makes it possible to implement a generic debugger for a development tool.
- Communicates with different debuggers via Debug Adapters.
- Debug Adapters can be re-used across multiple development tools.

Debug Adapter Protocol (DAP)



https://microsoft.github.io/debug-adapter-protocol/overview

NeoVim Distributions

NeoVim Distributions

- Provides excellent starting points for crafting your own custom configuration.
- Provides an out-of-the-box setup that can be used effectively without modification.
- Examples:
  - LazyVim
  - NvChad
  - LunarVim
  - AstroNvim
  - Kiskstart

The Vimmer Guide

The Vimmer Guide

- User-manual
- Vimtutor
- Vim Cheat Sheet
- Getting Started With Lua
- ThePrimagean Youtube Channel

- JetBrains' IdeaVim
- VSCode Neovim Plugin

Thanks

Vim has search, scroll, plugins like easymotion, vim-sneak to navigate buffer. You should use them

Vim go kkkkkjjjjkkkkk