Lecture Note 5.
Editor, Analyzer, Debugger

April 09, 2025

Kwanghee Lee
Dept. of Software
Dankook University

kh-lee@dankook.ac.kr

(본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작 되었습니다.)

Contents

- **Editor**
  - What is vim?
  - What is NeoVim?
  - How to use NeoVim?
- **Analyzer**
  - NeoVim Plugin
- **Debugger**
  - What is GDB?
  - How to use GDB?

What is vim? (1/4)

- **Editor**
  - ed: line editor for the Unix OS from Aug. 1969

```
a
ed is the standard Unix text editor.
This is line number two.
.
2i

.
,l
ed is the standard Unix text editor.$
$
This is line number two.$
3s/two/three/
,l
ed is the standard Unix text editor.$
$
This is line number three.$
w text
65
q
```

```
ed is the standard Unix text editor.

This is line number three.
```

  - Vi(Visual) editor
    - BSD, C shell, Vi by Bill Joy
    - Developed **“vi”** by adding the **“ed”** plugin
    - Not opensource => Opensource project

What is vim? (2/4)

- **Editor**
  - Vim: **Vi** + **IM**proved
    - Initial name: Vi + imitation
    - By Bram Moolenaar from 1991
    - Since vim is aliased (shortcut, link, connection) to vi, even if you type vi, it connects to vim.

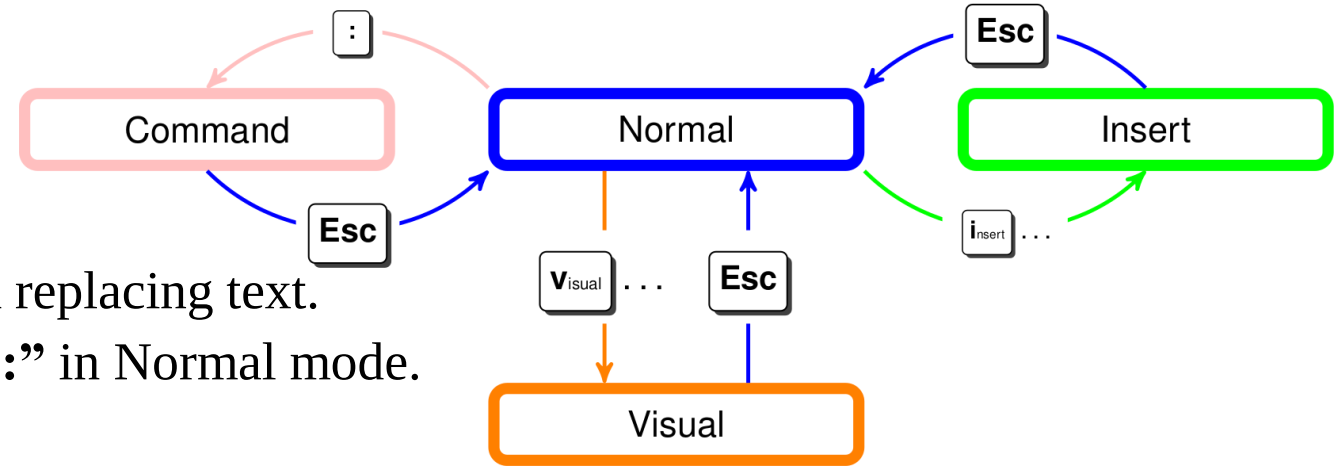
    - Linux and Unix command: vimtutor
    - New project: Neovim

```
                    VIM - Vi IMproved

                     version 8.2.2637
                 by Bram Moolenaar et al.
           Modified by <bugzilla@redhat.com>
     Vim is open source and freely distributable

            Become a registered Vim user!
type  :help register<Enter>   for information

type  :q<Enter>               to exit
type  :help<Enter>  or  <F1>  for on-line help
type  :help version8<Enter>   for version info
                                                  0,0-1         All
```

```
===============================================================================
=   W e l c o m e   t o   t h e   V I M   T u t o r   -   Version 1.7      =
===============================================================================

     Vim is a very powerful editor that has many commands, too many to
     explain in a tutor such as this.  This tutor is designed to describe
     enough of the commands that you will be able to easily use Vim as
     an all-purpose editor.

     The approximate time required to complete the tutor is 30 minutes,
     depending upon how much time is spent with experimentation.

     ATTENTION:
     The commands in the lessons will modify the text.  Make a copy of this
     file to practice on (if you started "vimtutor" this is already a copy).

     It is important to remember that this tutor is set up to teach by
     use.  That means that you need to execute the commands to learn them
     properly.  If you only read the text, you will forget the commands!

     Now, make sure that your Caps-Lock key is NOT depressed and press
     the   j   key enough times to move the cursor so that lesson 1.1
     completely fills the screen.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                        Lesson 1.1:  MOVING THE CURSOR
```

What is vim? (3/4)

- **Vim: mode**
  - Normal mode
    - Default mode, The first mode entered when running vim
    - Does not edit text, instead performs a command
      - h, j, k, l(cursor), dd(current line delete), yy(current line copy), p(paste)
    - Enter normal mode through **ESC**

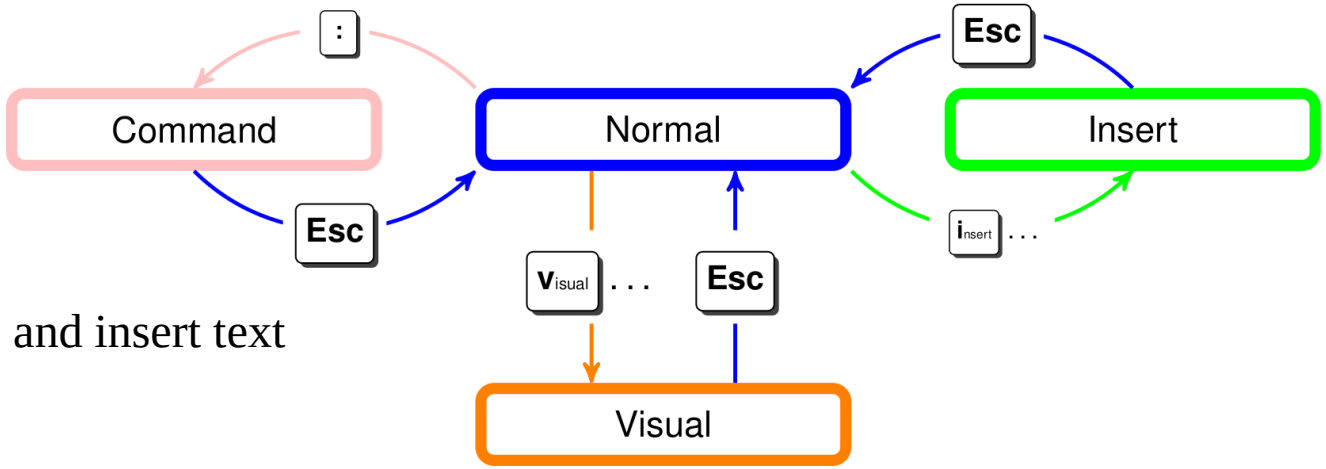


  - Command mode
    - Mode for saving, quitting, searching, and replacing text.
    - Command-line mode entered by typing **“:”** in Normal mode.
    - Command
      - :w(save the file), :q(quit vim), :wq(save and quit), :q!(quit without saving)

Ref : https://github.com/nilp0inter/Vim-for-Muggles

What is vim? (4/4)

- **Vim: mode**
  - Insert mode
    - Mode in which text can be edited
    - **Enter input mode**
      - i: Insert text before the cursor
      - a: Insert text after the cursor
      - o: Open a new line below the current line and insert text



  - Visual mode
    - Mode for selecting text blocks and executing commands.
    - Entered by typing **“v”** in Normal mode.
    - Type of Visual mode
      - v(select by character), V(select by line), Ctrl+v(select by block)

Ref : https://github.com/nilp0inter/Vim-for-Muggles

How to use vim in Linux? (1/2)

- **Vim**
  - install

```
~/Lecture/git/repo_dir
root# sudo apt install vim
```

  - open
    - vim [file name]

```
~/Lecture/git/repo_dir
root# vim
```

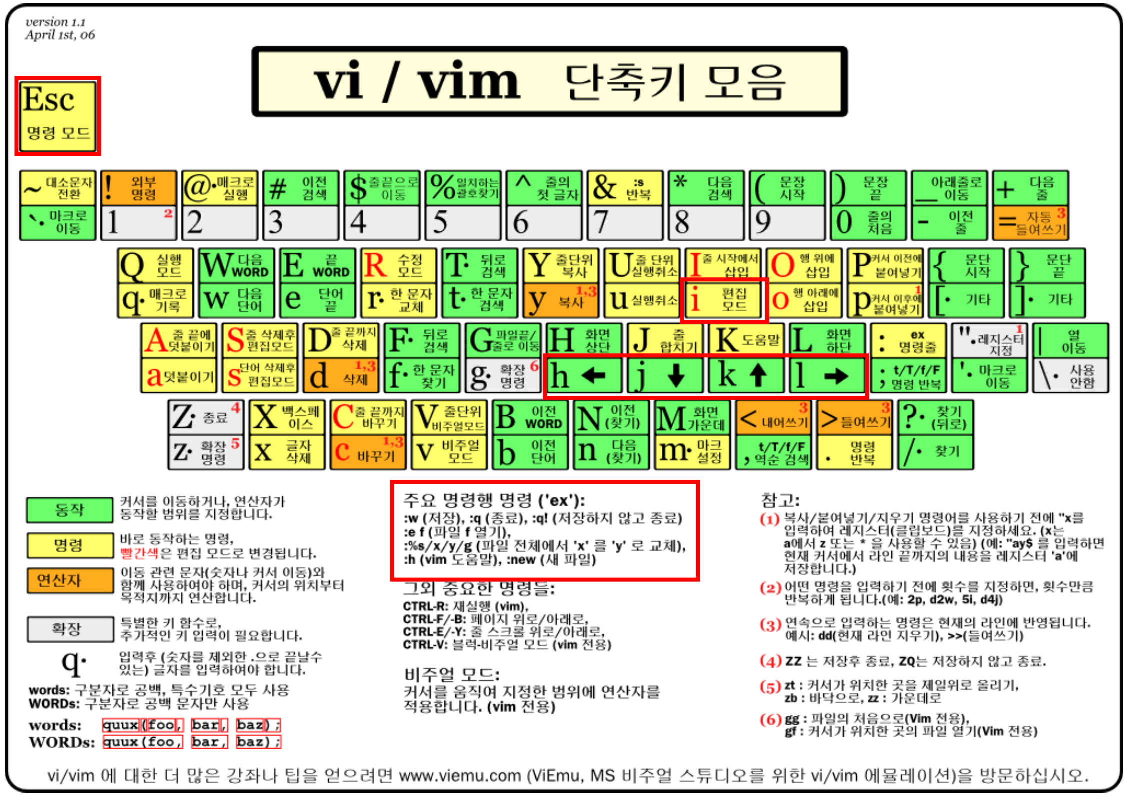
```
                    VIM - Vi IMproved

                    version 8.2.2121
                by Bram Moolenaar et al.
      Modified by team+vim@tracker.debian.org
    Vim is open source and freely distributable

           Help poor children in Uganda!
type  :help iccf<Enter>       for information

type  :q<Enter>               to exit
type  :help<Enter>  or  <F1>  for on-line help
type  :help version8<Enter>   for version info
```

How to use vim in Linux? (2/2)

- **Vim**

```
~/Lecture/git/repo_dir
root# ls
new_file  README  test_file
~/Lecture/git/repo_dir
root# vim new_file
```

```
pull request
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"new_file" 1L, 13B                                              1,1          All
```



동작: 커서를 이동하거나, 연산자가 동작할 범위를 지정합니다.

명령: 바로 동작하는 명령, 빨간색은 편집 모드로 변경됩니다.

연산자: 이동 관련 문자(숫자나 커서 이동)와 함께 사용하여야 하며, 커서의 위치부터 목적지까지 연산합니다.

확장: 특별한 키 함수로, 추가적인 키 입력이 필요합니다.

q·: 입력후 (숫자를 제외한 .으로 끝날수 있는) 글자를 입력하여야 합니다.

words: 구분자로 공백, 특수기호 모두 사용
WORDs: 구분자로 공백 문자만 사용

words: `quux(foo, bar, baz);`
WORDs: `quux(foo, bar, baz);`

주요 명령행 명령 ('ex'):
:w (저장), **:q** (종료), **:q!** (저장하지 않고 종료)
:e f (파일 f 열기),
:%s/x/y/g (파일 전체에서 'x' 를 'y' 로 교체),
:h (vim 도움말), **:new** (새 파일)

그외 중요한 명령들:
CTRL-R: 재실행 (vim),
CTRL-F/-B: 페이지 위로/아래로,
CTRL-E/-Y: 줄 스크롤 위로/아래로,
CTRL-V: 블럭-비주얼 모드 (vim 전용)

비주얼 모드:
커서를 움직여 지정한 범위에 연산자를 적용합니다. (vim 전용)

참고:
(1) 복사/붙여넣기/지우기 명령어를 사용하기 전에 **"x**를 입력하여 레지스터(클립보드)를 지정하세요. (**x**는 **a**에서 **z** 또는 * 을 사용할 수 있음) (예: **"ay$** 를 입력하면 현재 커서에서 라인 끝까지의 내용을 레지스터 **'a'**에 저장합니다.)
(2) 어떤 명령을 입력하기 전에 횟수를 지정하면, 횟수만큼 반복하게 됩니다.(예: **2p, d2w, 5i, d4j**)
(3) 연속으로 입력하는 명령은 현재의 라인에 반영됩니다. 예시: **dd**(현재 라인 지우기), **>>**(들여쓰기)
(4) **ZZ** 는 저장후 종료, **ZQ**는 저장하지 않고 종료.
(5) **zt** : 커서가 위치한 곳을 제일위로 올리기, **zb** : 바닥으로, **zz** : 가운데로
(6) **gg** : 파일의 처음으로(**Vim** 전용), **gf** : 커서가 위치한 곳의 파일 열기(**Vim** 전용)

vi/vim 에 대한 더 많은 강좌나 팁을 얻으려면 www.viemu.com (ViEmu, MS 비주얼 스튜디오를 위한 vi/vim 에뮬레이션)을 방문하십시오.

What is NeoVim? (1/2)

- Vim-based text editor
  - Offers **improved extensibility and modern features** while preserving the familiar key bindings and philosophy of **Vim** users.
- Extensible and usable
  - Significantly **improved plugin system** enables flexible extension of various features.
- Light and fast
  - Lightweight and fast performance as core advantages of Neovim.
- Configurable by Lua script
  - Configuration and extension possible with **Lua** in Neovim.

What is NeoVim? (2/2)

- **Lua**
  - A lightweight scripting language
  - PUC-Rio University, Brazil
  - 'Moon' in Portuguese

  - **Feature**
    - Lightweight and Embeddable
    - Dynamically Typed & Interpreted
    - Simple and Minimal Syntax

```lua
local a = 10 -- local variable
b = "hello" -- global variable

local function greet(name)
  print("Hello, " .. name)
end

greet("Neovim") -- call function

local config = { -- declare table
  theme = "tokyonight",
  line_numbers = true,
}

local lspconfig = require("lspconfig") -- load module
lspconfig.pyright.setup{}

if vim.fn.has("win32") == 1 then -- condition
  print("Running on Windows")
end
```

How to use NeoVim? (1/6)

- **Install**
  - https://github.com/neovim/neovim
- **Dependencies**
  - lua (for configuration), ripgrep (for search)
- **Version**
  - 0.10.4 (plugin dependency issue)

How to use NeoVim? (2/6)

- **Configuration**
  - Using init.vim

```
~/.config/nvim
|-- init.vim
```

  - **Using init.lua**

```
~/.config/nvim
|-- init.lua            Start point → require("config")
|-- lua/                Module directory
| |-- config/
|   |-- init.lua          Initialization file; auto loaded → require("config.options")
|   |-- options.lua       options file; auto loaded
| |-- plugins/          Plugins; auto loaded
| |-- utils/
```

How to use NeoVim? (3/6)

- **options.lua Basic configuration**

vim.o.[option] = value

- *vim.o.number = true*
  - Always shows line numbers.
- *vim.o.relativenumber = true*
  - Displays relative line numbers based on the current line.
- *vim.o.relativenumber = true*
  - Highlights the line where the cursor is currently located.
- *vim.o.signcolumn = "no/yes/auto/number"*
  - Adds a sign column.

How to use NeoVim? (4/6)

- **options.lua Tab/Indent configuration**
  - *vim.o.tabstop = 4*
    - Sets tab width.
  - *vim.o.shiftwidth = 4*
    - Sets indentation width.
  - *vim.o.expandtab = true*
    - Inserts **Spaces** instead of tab characters when pressing **Tab**.
  - *vim.o.smarttap = true / vim.o.smartindent*
    - Automatically sets tabs and indentation based on code context.
  - *vim.o.wrap = false*
    - Enables automatic line wrapping when lines exceed the screen width.

Item	Tab (\t)	Space ( )
Basic Concept	Uses a single tab character for indentation	Uses multiple space characters for indentation
Pros	- Smaller file size - User can customize width per editor	- Consistent appearance across all environments - Aligns with many style guides
Cons	- Width varies by editor settings - May cause misalignment	- Larger file size (uses more characters)
Consistency	Lower (depends on editor configuration)	Higher (same everywhere)

How to use NeoVim? (5/6)

- **options.lua Search configuration**
  - *vim.o.ignorecase = true*
    - Ignores case when searching.
  - *vim.o.smartcase = true*
    - Case sensitivity enabled.
  - *vim.o.hlsearch = true*
    - Highlights searched words on screen.
  - *vim.o.incsearch = true*
    - Displays search results in real time while typing the query.

How to use NeoVim? (6/6)

- **options.lua Appendix configuration**

Option	Example Usage	Description
scrolloff	vim.o.scrolloff = 8	Make sure that the cursor is not too close to the top or bottom of the screen, and that you have some white space
updatetime	vim.o.updatetime = 300	Time in ms before triggering CursorHold/autocommands (lower = faster feedback)
timeoutlen	vim.o.timeoutlen = 500	Time to wait (ms) for mapped key sequence to complete
hidden	vim.o.hidden = true	Allows switching buffers without saving (useful for editing multiple files)
splitbelow	vim.o.splitbelow = true	New horizontal splits appear below the current window
splitright	vim.o.splitright = true	New vertical splits appear to the right of the current window
showmode	vim.o.showmode = false	Hides mode display ("-- INSERT --"), often disabled if using statusline
cmdheight	vim.o.cmdheight = 1	Height of the command line (increase if you need more space for messages)
laststatus	vim.o.laststatus = 3	Statusline behavior: 3 = global statusline, 2 = each window has one
wildmenu	vim.o.wildmenu = true	Enables enhanced command-line completion interface

NeoVim Plugin (1/13)

LSP: Language Server Protocol

- **Plugin manager**
  - Neovim is lightweight and fast, but basic
  - Autocomplete, file browsing, LSP*, Git integration, themes, etc. are not built-in
    - need to extend with plugins
  - Manual installation of plugins is difficult to manage

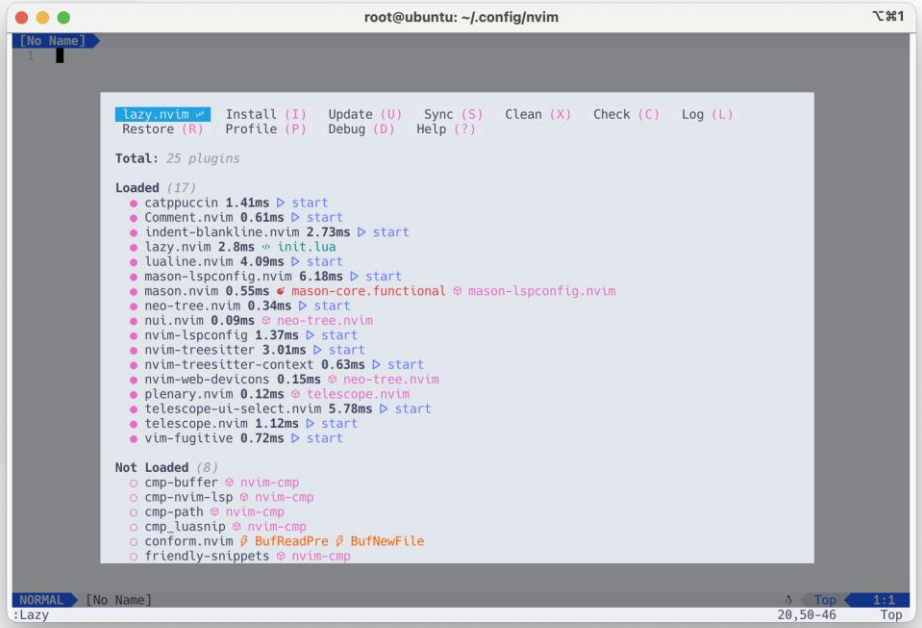
Plugin Manager	Features
vim-plug	Old favorite, highly compatible with Vim
packer.nvim	Lua-based, Neovim only
lazy.nvim ✅	Trendy, fast, powerful, easy to set up (most recommended these days)

NeoVim Plugin (2/13)

- **lazy.nvim**
  - Manage all Neovim plugins

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
|   |-- init.lua
|   |-- options.lua
| |-- plugins/
| |-- utils/
```

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim”

if not (vim.uv or vim.loop).fs_stat(lazypath) then
  local lazyrepo = "https://github.com/folke/lazy.nvim.git"
  local out = vim.fn.system({
    "git", "clone",
    "--filter=blob:none", "--branch=stable",
    lazyrepo, lazypath
  })
end
vim.opt.rtp:prepend(lazypath)

require("config.options")

local plugins = "plugins" -- .config/nvim/lua/plugins/*
local opts = {}

require("lazy").setup(plugins, opts)
```

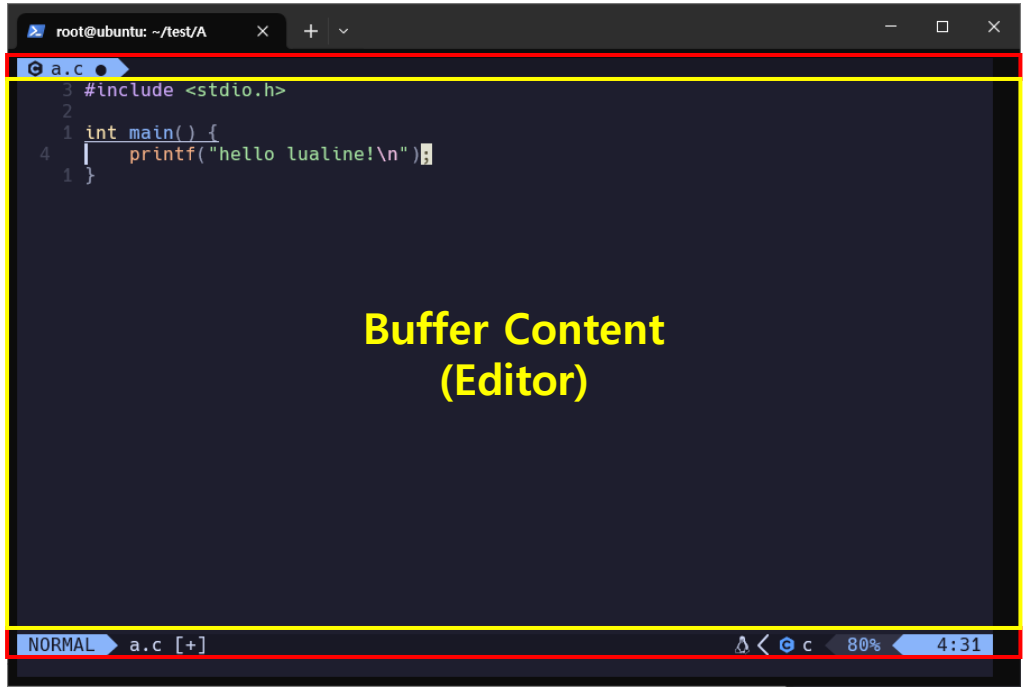


NeoVim Plugin (3/13)

- **Lualine.nvim**

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
| |-- plugins/
|   |-- lualine.lua
| |-- utils/
```

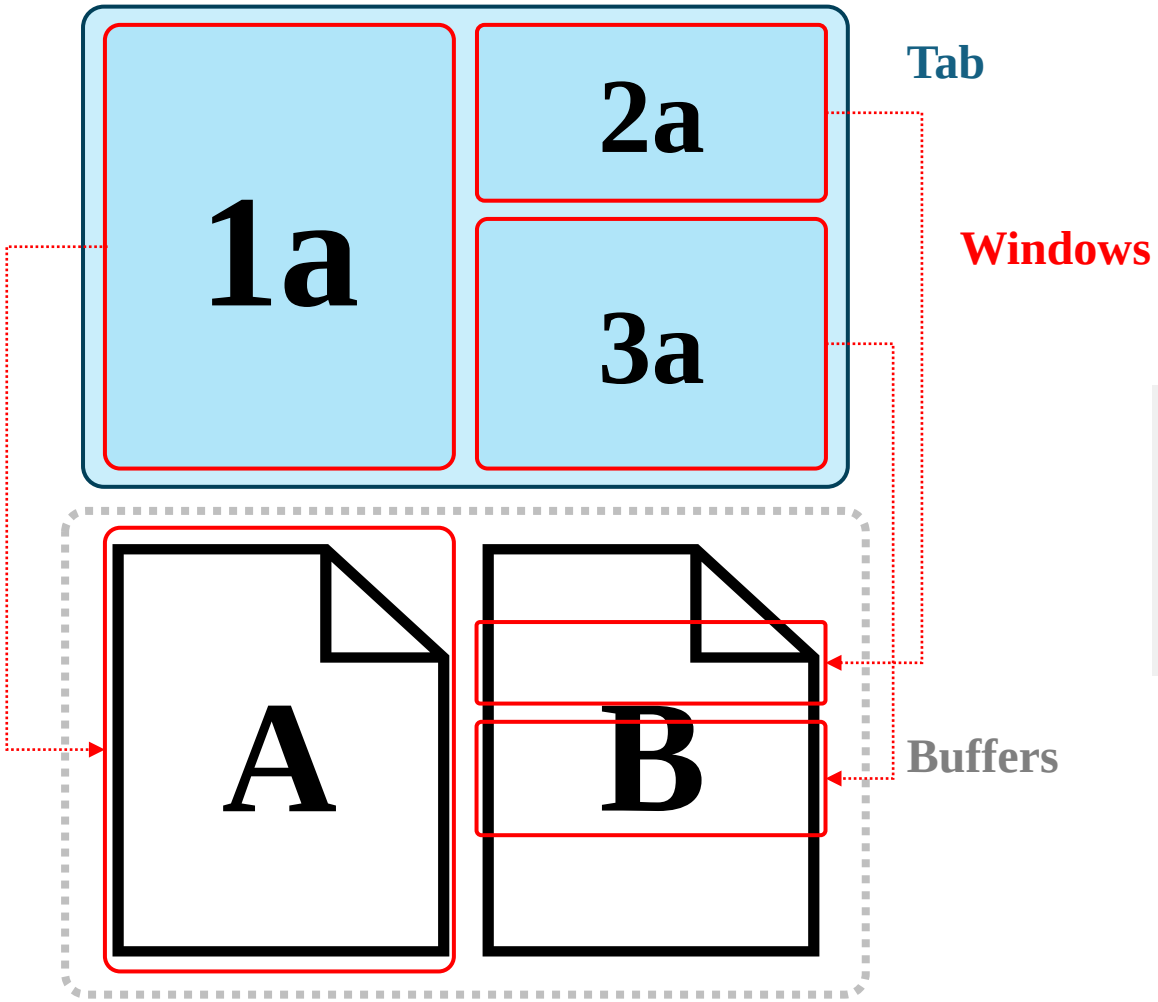
```lua
return {
  "nvim-lualine/lualine.nvim",
  dependencies = { "nvim-tree/nvim-web-devicons" },
  config = function()
    require("lualine").setup({
      tabline = {
        lualine_a = {'buffers'},
        lualine_b = {},
        lualine_c = {},
        lualine_x = {},
        lualine_y = {},
        lualine_z = {},
      },
      sections = {
        lualine_a = {'mode'},
        lualine_b = {},
        lualine_c = {},
        lualine_x = {},
        lualine_y = {},
        lualine_z = {'location'},
      },
      inactive_sections = {
        lualine_a = {},
        lualine_b = {},
        lualine_c = {},
        lualine_x = {},
        lualine_y = {},
        lualine_z = {},
      },
    })
  end,
}
```




```
+-------------------------------------------------------------+
| A |       B       |       C       ||       X       | Y | Z |
+-------------------------------------------------------------+
```

Section	Position	Information
A	좌측 제일 앞	모드 (NORMAL, INSERT 등)
B	좌측 중간	브랜치 이름 (Git), diff 상태 등
C	좌측 끝	파일 경로, 파일명, 파일 상태 등
X	우측 시작	파일 형식(filetype), encoding 등
Y	우측 중간	진행률 (%), 현재 줄/컬럼 등
Z	우측 제일 끝	커서 위치 (line:column)

Appendix – buffer, window, tab



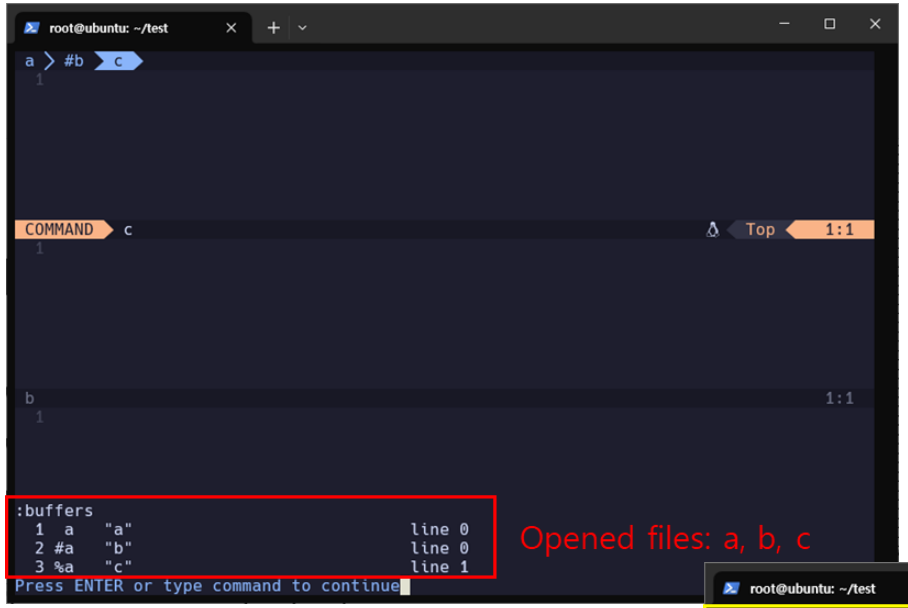
Term	Description
Buffer	In-memory text of a file
Window	Viewport on a buffer
Tab	Collection of windows

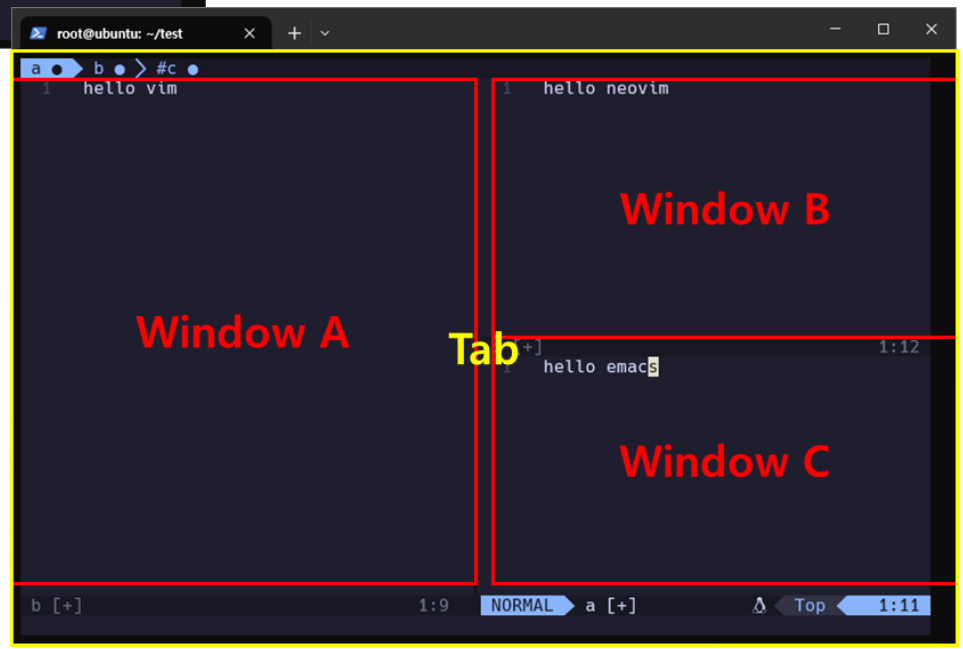
```
[ Tab Page 1 ]
 └─ [ Window 1a ] ─▶ [ Buffer A ]
 └─ [ Window 2a ] ─▶ [ Buffer B ]
 └─ [ Window 3a ] ─▶ [ Buffer B ] # same buffer, other window

[ Tab Page 2 ]
 └─ [ Window 1b ] ─▶ [ Buffer C ]
```

- Tab : Window = 1 : N
- Buffer : Window = 1 : N
- Window : Buffer = 1 : 1

Appendix – buffer, window, tab

- **Check buffer**
  - :ls or :buffers
- **Using Window**
  - :split
  - :vsplit
  - ctrl-w w
- **Using Tab**
  - :tabnew
  - :tabprev/next
  - :tabfirst/last





NeoVim Plugin (4/13)

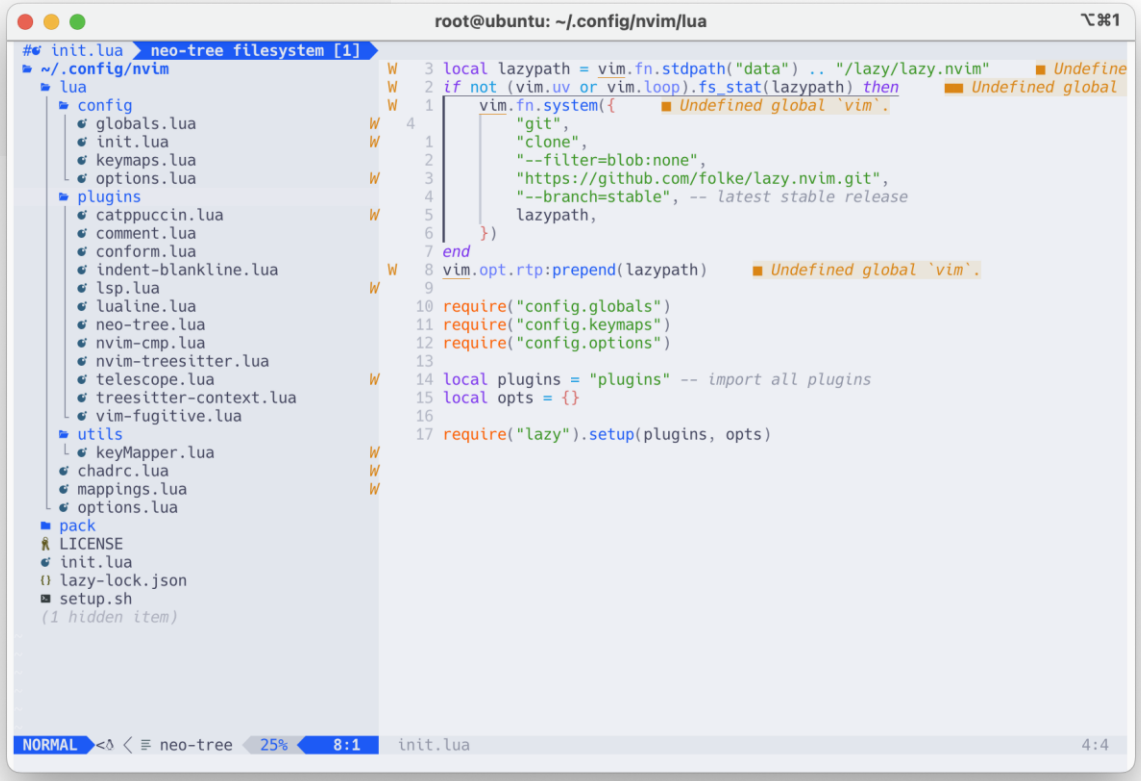
- **Neo-Tree.nvim**

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
| |-- plugins/
|   |-- lualine.lua
|   |-- neo-tree.lua
| |-- utils/
```

```lua
return {
  "nvim-neo-tree/neo-tree.nvim",
  branch = "v3.x",
  dependencies = {
    "nvim-lua/plenary.nvim",
    "nvim-tree/nvim-web-devicons", -- not strictly required, but recommended
    "MunifTanjim/nui.nvim",
  },
  lazy = false, -- neo-tree will lazily load itself
  opts = {
    -- fill any relevant options here
  },
}
```

- **Using**
  - :Neotree

```lua
mapKey('<C-o>', ':Neotree position=left dir=%:p:h:h toggle<CR>')
```

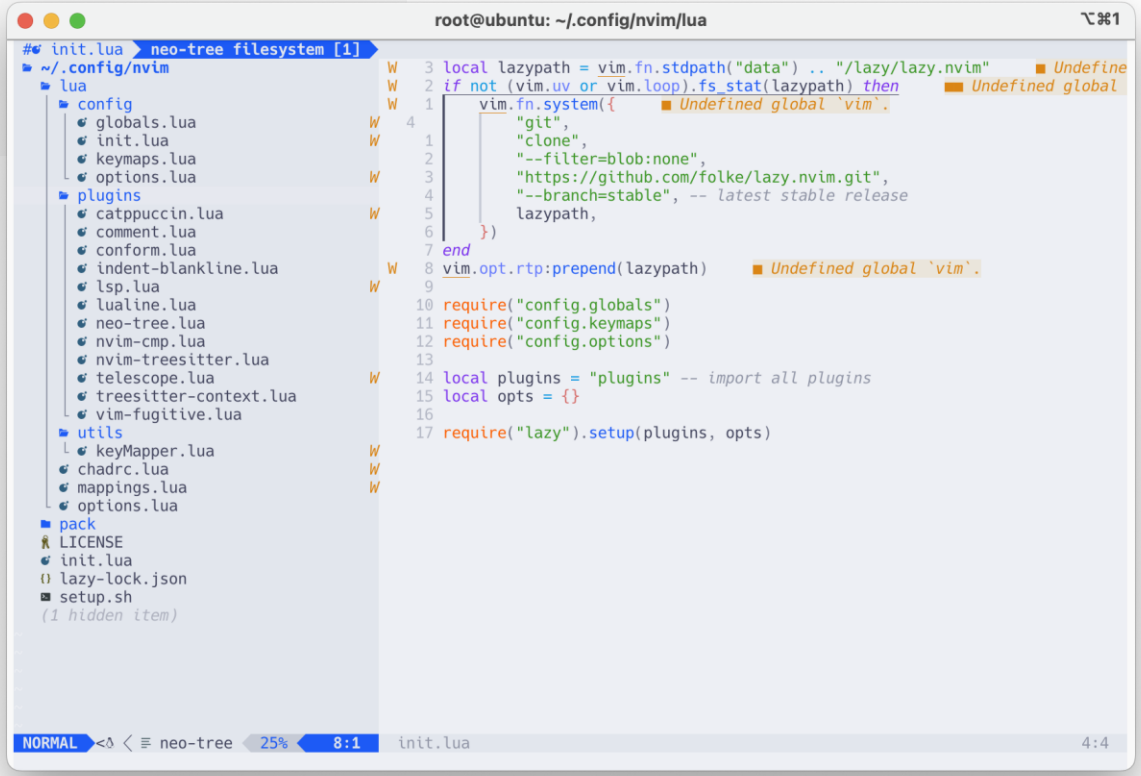


NeoVim Plugin (5/13)

- **Neo-Tree.nvim**

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
| |-- plugins/
|   |-- lualine.lua
|   |-- neo-tree.lua
| |-- utils/
```

```lua
return {
  "nvim-neo-tree/neo-tree.nvim",
  branch = "v3.x",
  dependencies = {
    "nvim-lua/plenary.nvim",
    "nvim-tree/nvim-web-devicons", -- not strictly required, but recommended
    "MunifTanjim/nui.nvim",
  },
  lazy = false, -- neo-tree will lazily load itself
  opts = {
    -- fill any relevant options here
  },
}
```



NeoVim Plugin – Code analyzer (6/13)

- **Telescope.nvim**

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
| |-- plugins/
|   |-- lualine.lua
|   |-- neo-tree.lua
|   |-- telescope.lua
| |-- utils/
```

```lua
return {
  "nvim-telescope/telescope.nvim",
  tag = "0.1.8",
  dependencies = { "nvim-lua/plenary.nvim" },
  config = function()
    local builtin = require("telescope.builtin")
    vim.keymap.set("n", "<leader>ff", builtin.find_files, { desc = "Telescope find files" })
    vim.keymap.set("n", "<leader>fg", builtin.live_grep, { desc = "Telescope live grep" })
    vim.keymap.set("n", "<leader>fb", builtin.buffers, { desc = "Telescope buffers" })
    vim.keymap.set("n", "<leader>fh", builtin.help_tags, { desc = "Telescope help tags" })
  end,
}
```

```lua
vim.keymap.set({mode}, {key}, {command}, {options})
```

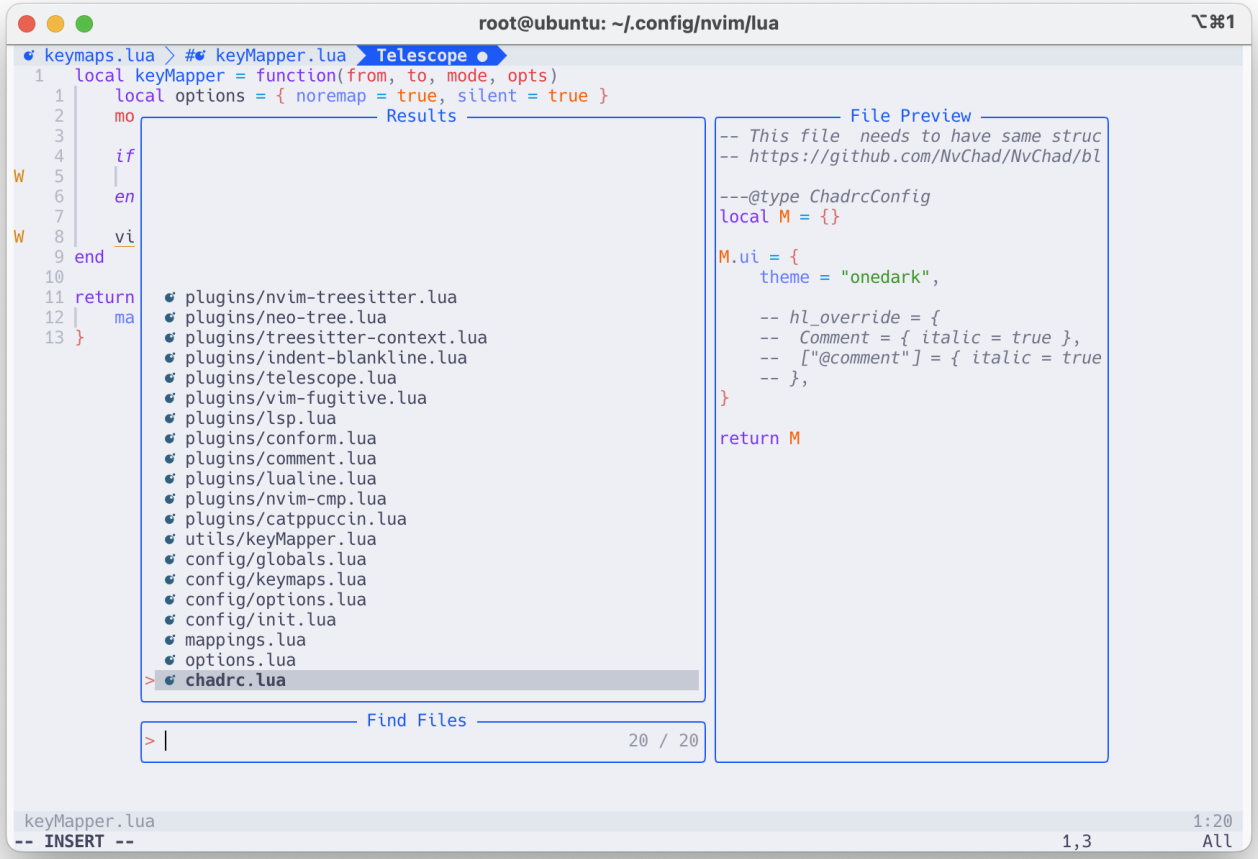
```lua
vim.g.mapleader = " " -- spacebar
```

Mode	Description
"n"	Normal mode
"i"	Insert mode
"v"	Visual mode
"x"	Visual select mode
"t"	Terminal mode

Option	Description
noremap	No reference to existing key mappings (safe)
silent	Don't print to the command line when running commands

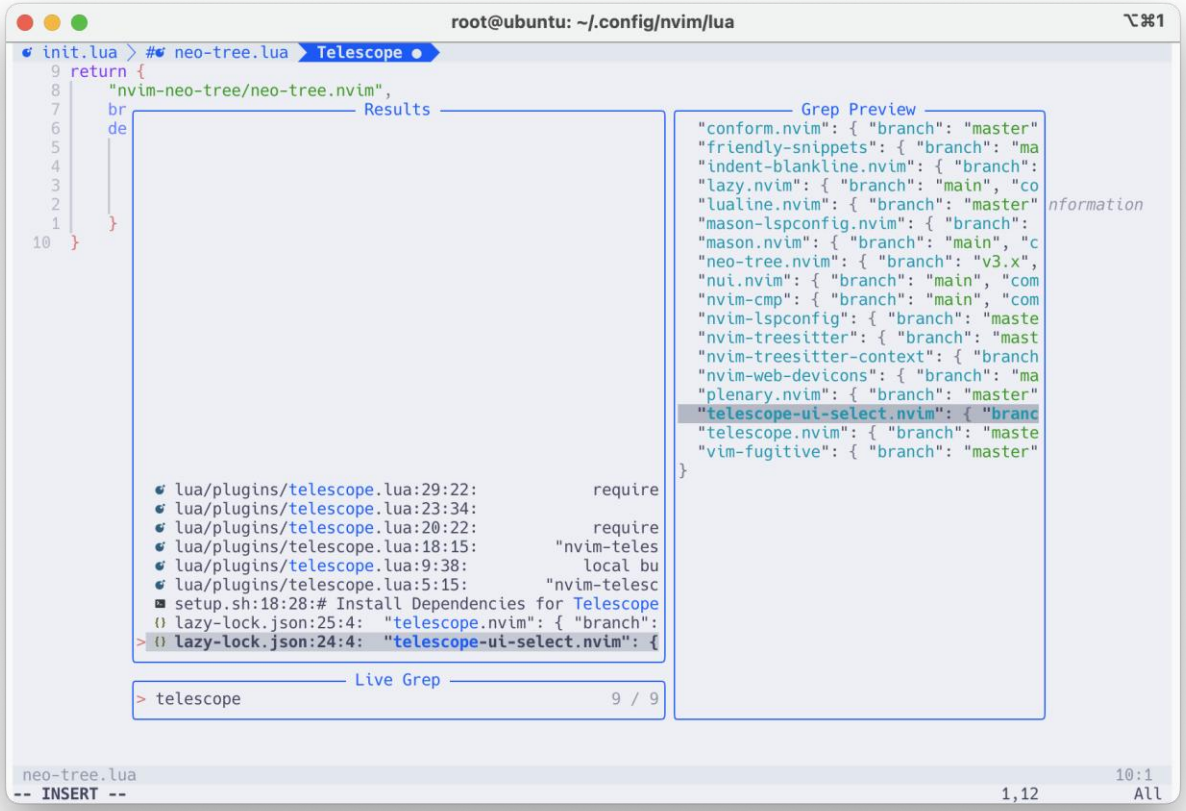
NeoVim Plugin – Code analyzer (7/13)

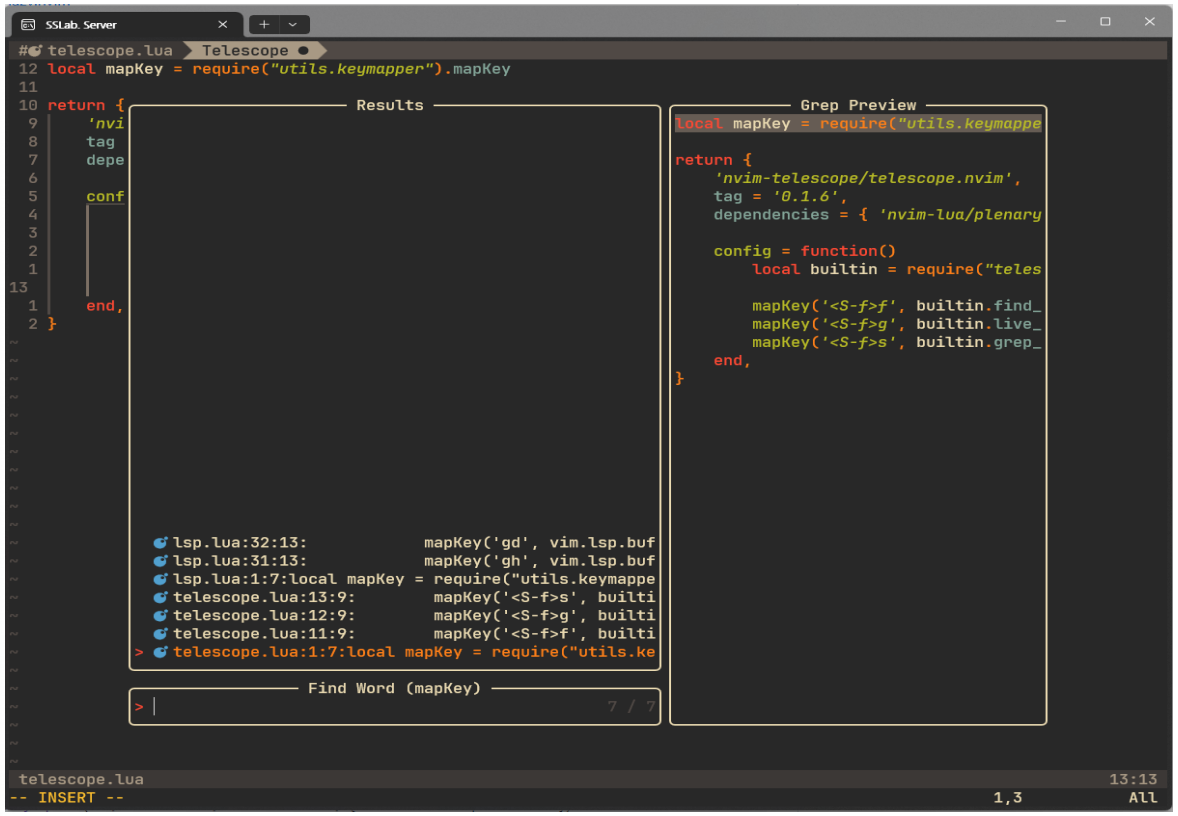
- **Telescope.nvim**
  - `vim.keymap.set("n", "<leader>ff", builtin.find_files, { desc = "Telescope find files" })`



NeoVim Plugin – Code analyzer (8/13)

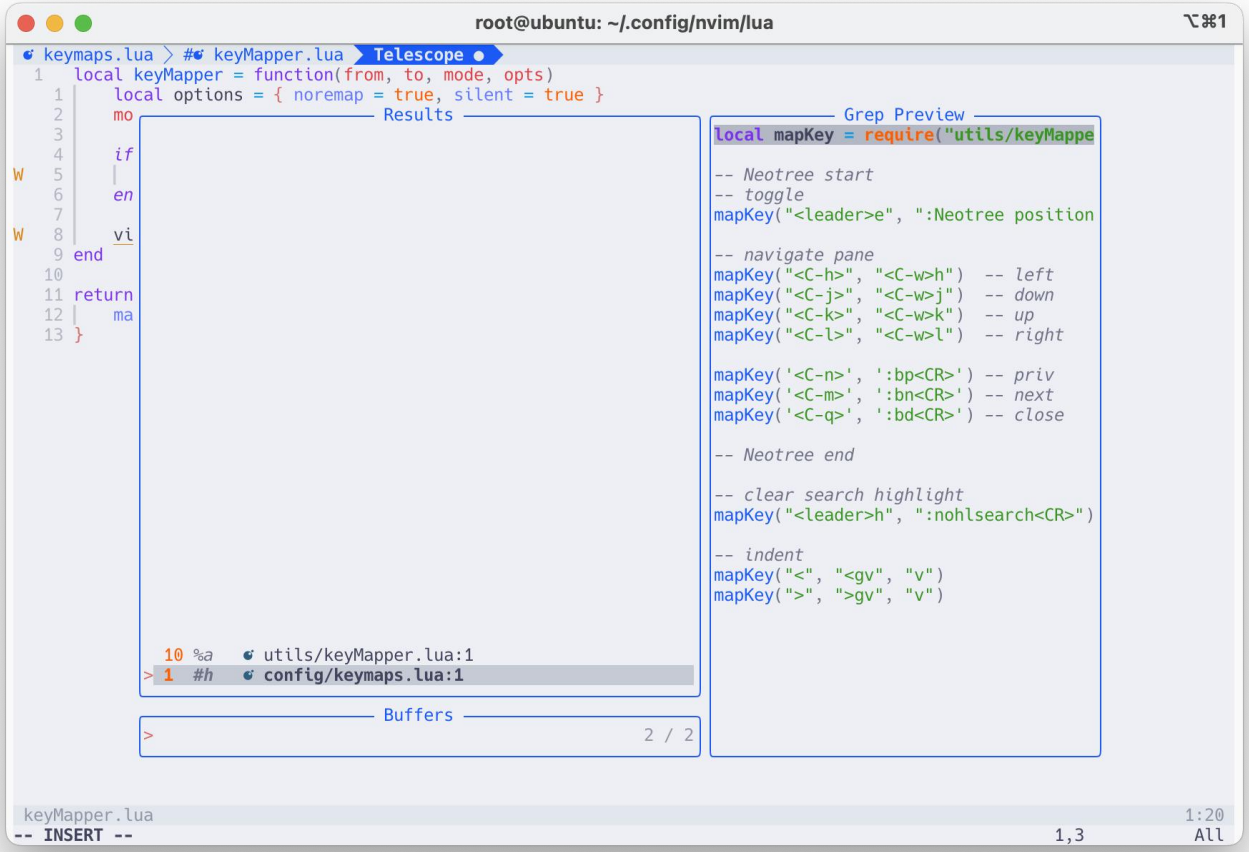
- **Telescope.nvim**
  - `vim.keymap.set("n", "<leader>fg", builtin.live_grep, { desc = "Telescope live grep" })`
  - `vim.keymap.set("n", "<leader>fs", builtin.grep_string, { desc = "Telescope grep_string" })`

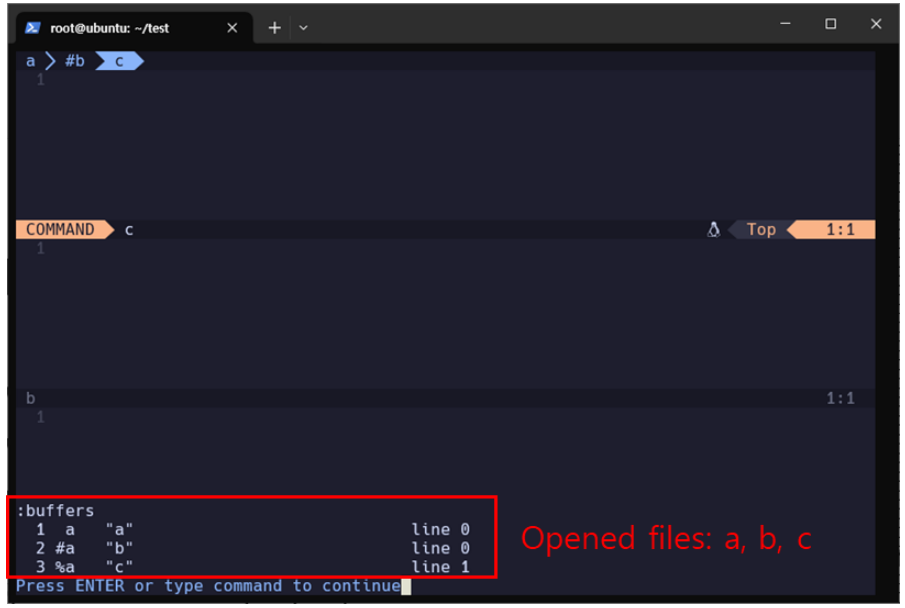




NeoVim Plugin – Code analyzer (9/13)

- **Telescope.nvim**
  - `vim.keymap.set("n", "<leader>fb", builtin.buffers, { desc = "Telescope buffers" })`





NeoVim Plugin – Code analyzer (10/13)

- **nvim-lspconfig** *LSP: Language Server Protocol*
  - mason.nvim
  - mason-lspconfig.nvim
- **nvim-treesitter**

```c
int tslua_add_language(lua_State *L)
{
  if (lua_gettop(L) < 2 || !lua_isstring(L, 1) || !lua_isstring(L, 2)) {
    return luaL_error(L, "string expected");
  }

  const char *path = lua_tostring(L, 1);
  const char *lang_name = lua_tostring(L, 2);

  if (pmap_has(cstr_t)(langs, lang_name)) {
    return 0;
  }

#define BUFSIZE 128
  char symbol_buf[BUFSIZE];
  snprintf(symbol_buf, BUFSIZE, "tree_sitter_%s", lang_name);
#undef BUFSIZE

  uv_lib_t lib;
  if (uv_dlopen(path, &lib)) {
    snprintf((char *)IObuff, IOSIZE, "Failed to load parser: uv_dlopen: %s",
             uv_dlerror(&lib));
    uv_dlclose(&lib);
    lua_pushstring(L, (char *)IObuff);
    return lua_error(L);
  }

  TSLanguage *(*lang_parser)(void);
  if (uv_dlsym(&lib, symbol_buf, (void **)&lang_parser)) {
    snprintf((char *)IObuff, IOSIZE, "Failed to load parser: uv_dlsym: %s",
             uv_dlerror(&lib));
    uv_dlclose(&lib);
    lua_pushstring(L, (char *)IObuff);
    return lua_error(L);
  }

  TSLanguage *lang = lang_parser();
  if (lang == NULL) {
    return luaL_error(L, "Failed to load parser: internal error");
  }

  pmap_put(cstr_t)(langs, xstrdup(lang_name), lang);

  lua_pushboolean(L, true);
  return 1;
}
```

```c
int tslua_add_language(lua_State *L)
{
  if (lua_gettop(L) < 2 || !lua_isstring(L, 1) || !lua_isstring(L, 2)) {
    return luaL_error(L, "string expected");
  }

  const char *path = lua_tostring(L, 1);
  const char *lang_name = lua_tostring(L, 2);

  if (pmap_has(cstr_t)(langs, lang_name)) {
    return 0;
  }

#define BUFSIZE 128
  char symbol_buf[BUFSIZE];
  snprintf(symbol_buf, BUFSIZE, "tree_sitter_%s", lang_name);
#undef BUFSIZE

  uv_lib_t lib;
  if (uv_dlopen(path, &lib)) {
    snprintf((char *)IObuff, IOSIZE, "Failed to load parser: uv_dlopen: %s",
             uv_dlerror(&lib));
    uv_dlclose(&lib);
    lua_pushstring(L, (char *)IObuff);
    return lua_error(L);
  }

  TSLanguage *(*lang_parser)(void);
  if (uv_dlsym(&lib, symbol_buf, (void **)&lang_parser)) {
    snprintf((char *)IObuff, IOSIZE, "Failed to load parser: uv_dlsym: %s",
             uv_dlerror(&lib));
    uv_dlclose(&lib);
    lua_pushstring(L, (char *)IObuff);
    return lua_error(L);
  }

  TSLanguage *lang = lang_parser();
  if (lang == NULL) {
    return luaL_error(L, "Failed to load parser: internal error");
  }

  pmap_put(cstr_t)(langs, xstrdup(lang_name), lang);

  lua_pushboolean(L, true);
  return 1;
}
```

NeoVim Plugin (11/13)

- **colorscheme**

```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
| |-- plugins/
|   |-- lualine.lua
|   |-- neo-tree.lua
|   |-- telescope.lua
|   |-- colorscheme.lua
| |-- utils/
```

colorscheme.lua

```lua
return {
    "ellisonleao/gruvbox.nvim",
    priority = 1000,
    lazy = false,
    config = function()
        vim.o.background = "dark"
        vim.cmd([[colorscheme gruvbox]])
    end,
}
```

NeoVim Plugin (12/13)

- **keymaps**

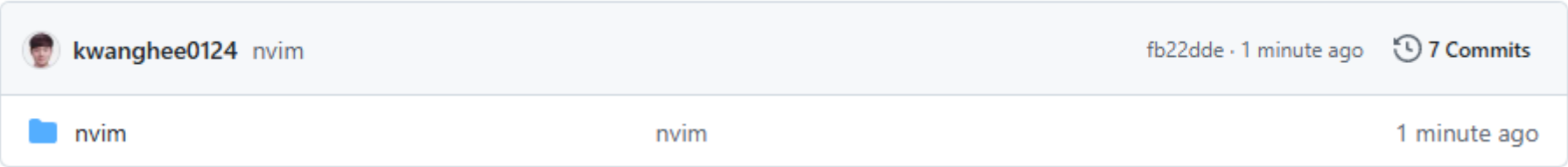
```
~/.config/nvim
|-- init.lua
|-- lua/
| |-- config/
|   |-- keymaps.lua
| |-- plugins/
| |-- utils/
```

keymaps.lua

```lua
vim.g.mapleader = " "      -- global leader
vim.g.maplocalleader = " " -- local leader

local mapKey = require("utils.keymapper").mapKey

-- Neotree toggle
mapKey('<C-o>', ':Neotree position=left dir=%:p:h:h toggle<CR>')

-- pane navigation
mapKey('<C-h>', '<C-w>h') -- Left
mapKey('<C-j>', '<C-w>j') -- Down
mapKey('<C-k>', '<C-w>k') -- Up
mapKey('<C-l>', '<C-w>l') -- Rigth

-- buffer navigation
mapKey('<C-n>', ':bp<CR>') -- priv
mapKey('<C-m>', ':bn<CR>') -- next
mapKey('<C-q>', ':bd<CR>') -- close

-- clear search hi
mapKey('<leader>h', ':nohlsearch<CR>')

-- indent
mapKey('<', '<gv', 'v')
mapKey('>', '>gv', 'v')
```

NeoVim Plugin (13/13)

- **neovim config**
  - https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OpenSourceBasic
  - Branch: nvim

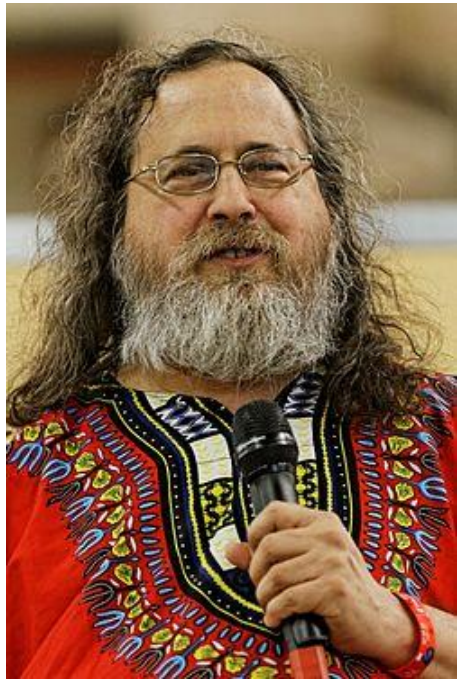



```
.
├── init.lua
├── lazy-lock.json
└── lua
    ├── config
    │   ├── init.lua
    │   ├── keymaps.lua
    │   └── options.lua
    ├── plugins
    │   ├── alpha-nvim.lua
    │   ├── autopairs.lua
    │   ├── cmp.lua
    │   ├── colorscheme.lua
    │   ├── comment.lua
    │   ├── conform.lua
    │   ├── indent-blankline.lua
    │   ├── lsp.lua
    │   ├── lualine.lua
    │   ├── neo-tree.lua
    │   ├── telescope.lua
    │   ├── treesitter.lua
    │   └── vim-fugitive.lua
    └── utils
        └── keymapper.lua
```

What is GDB? (1/2)

- **GNU Debugger (GDB)**
  - A powerful **command-line debugger** for programs written in C, C++, Fortran, and other languages. Commonly **used in Linux environments** to **trace source code** execution, inspect or modify variable values, and analyze causes of **abnormal program termination**.

  - Richard M. Stallman



GDB

The GNU Project Debugger

What is GDB? (2/2)

- **Features of gdb**
  - Breakpoint Setting
    - Allows the program to stop at a specific location to inspect its state.
  - Step Execution
    - Executes the code line by line using commands like next or step.
  - Variable Inspection and Modification
    - Displays or modifies the values of local or global variables.
  - Call Stack Inspection
    - Shows the current function call stack (e.g., using the backtrace command).
  - Memory/Register Inspection
    - Enables checking memory addresses, pointer values, and register states.
  - Conditional Breakpoints
    - Breakpoints can be set to trigger only when specific conditions are met.

How to use GDB? (1/9)

- **GCC**
  - GNU Compiler Collection
  - Richard M. Stallman

  - gcc -g
    - Generate a binary that includes debugging information (metadata).
    - gcc -g main.c => a.out

    - Include information
      - Source File Names and Line Numbers
        - Used by GDB commands like list, break, and next.
      - Function and Variable Names
        - Symbol information is required for GDB to understand commands like print my_var.
      - Local Variables, Global Variables, and Parameter Information
        - Enables tracking of call stacks and variable values.
      - Type Information
        - Allows type-based analysis of structures, pointers, etc.
      - Line-to-Address Mapping Information
        - Maps machine code addresses to corresponding lines in the source code

How to use GDB? (2/9)

- **Install**
  - sudo apt install gdb

- **Using GDB**
  - gcc -g main.c → a.out
  - gdb ./a.out

```
~/Lecture/gdb
root# ls
hello.c
~/Lecture/gdb
root# cat hello.c
#include <stdio.h>

int main()
{
    printf("hello gdb\n");
    return 0;
}
~/Lecture/gdb
root# gcc -g hello.c
~/Lecture/gdb
root# ls
a.out  hello.c
~/Lecture/gdb
root# ./a.out
hello gdb
```

```
~/Lecture/gdb
root# gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb)
```

How to use GDB? (3/9)

- **GDB basic commands**
  - gdb <program name>
    - Start GDB and load the executable
  - quit(or q)
    - Exit GDB
  - run(or r) [args]
    - Run the program (arguments can be passed if needed)
  - file <file name>
    - Change or load an executable file.
  - help
    - Display help for GDB commands

How to use GDB? (4/9)

- **GDB Breakpoint commands**
  - break(or b)
    - <function>
      - Stop when entering the specified function
    - <file>:<line>
      - Stop at a specific line in the source file
  - delete(or d) <number>
    - Delete the specified breakpoint
  - disable <number> / enable <number>
    - Temporarily disable / re-enable the specified breakpoint
  - condition <number> <condition>
    - Set a condition for the specified breakpoint

How to use GDB? (5/9)

- **GDB flow control commands**
  - next(or n)
    - Move to the next line (**skips over function calls**)
  - step(or s)
    - Move to the next line (**steps into function calls**)
  - continue(or c)
    - Resume execution until the next breakpoint
  - finish
    - Execute until the current function finishes and return to the caller
  - until
    - Run until a specific line within the current function is reached

How to use GDB? (6/9)

- **GDB check commands**
  - list(or l)
    - View source code
  - print(or p)
    - Print the value of a variable
  - display
    - Continuously display the value during execution
  - undisplay
    - Cancel a previously set display
  - set var
    - Change the value of a variable
  - x
    - Examine memory contents (e.g., x/4xb &var)

How to use GDB? (7/9)

- **GDB call stack commands**
  - backtrace(or bt)
    - Display the function call stack
  - frame(or f)
    - Move to a specific stack frame
  - up
    - Move to the previous (caller) stack frame
  - down
    - Move to the next (callee) stack frame

How to use GDB? (8/9)

- **GDB watch commands**
  - watch
    - Watch for changes to a variable
  - rwatch
    - Watch for read access to a variable
  - awatch
    - Watch for both read and write access

How to use GDB? (9/9)

- **GDB information commands**
  - info breakpoints(or i b)
    - Display all set breakpoints
  - info line(or i li)
    - Show debugging information for a specific line
  - info functions(or i func)
    - Display a list of functions
  - info args(or i ar)
    - Show arguments of the current function
  - info locals(or i lo)
    - Show a list of local variables
  - info variables(or i var)
    - Display a list of global variables
  - info watchpoints(or i wat)
    - Display the list of active watchpoints

GDB Practice 1

- Clone code
  - https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OpenSourceBasic.git
  - Branch: debug
- Compile
  - debug_demo.c, Executable file name(-o): debug_demo

1. Set a breakpoint and get started
2. Follow the code flow
3. Check variable value
   - main, compute, add, multiply
4. Check the call stack

GDB Practice 2

- Compile
  - buggy_demo.c, Executable file name: buggy_demo

- Error point tracking
  1. Option 1 - cause_segfault()
  2. Option 2 - out_of_bounds()
  3. Option 3 - null_string_copy()

Summary

- **Editor**
  - What is vim?
  - What is NeoVim?
  - How to use NeoVim?
- **Analyzer**
  - NeoVim Plugin
- **Debugger**
  - What is GDB?
  - How to use GDB?

Assignment 4

1. Practice 1, 2

- **제출 요건**
  - Include student ID and date (using whoami, date)
  - 기한: 일주일
  - 양식: 포맷 없음, 장수 제한 없음, pdf (파일명: 오픈소스SW기초_{분반}_{이름}_{학번}.pdf)
  - 제출: e-Campus => 과제

Aknowledgement

- 본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 ‘SW중심대학’의 결과물이라는 출처를 밝혀야 합니다.

과학기술정보통신부 정보통신기획평가원

SW중심대학