

with-ms-copilot

in state of neovim lua configuration, suggest me several custom functions that are plug-and-play

awesome, try to discover more useful yet ergonomic functions

based on both previous responses, suggest me functions that are still not mentioned yet

what about more functions, but inspired by some plugins' functionality

Got it ⚡ — you want some **ready-to-drop-in Lua functions** for your Neovim config that don't require external plugins, just pure Lua + Neovim API. Here are several **plug-and-play custom functions** you can paste into your `init.lua` or any Lua module:

🔧 Useful Custom Functions for Neovim Lua Config

1. Toggle Relative Line Numbers

```
function ToggleRelativeNumber()
  if vim.wo.relativenumber then
    vim.wo.relativenumber = false
    vim.wo.number = true
  else
    vim.wo.relativenumber = true
    vim.wo.number = true
  end
end

vim.api.nvim_set_keymap('n', '<leader>rn', ':lua ToggleRelativeNumber()<CR>', {
  noremap = true, silent = true })
```

👉 Quickly switch between absolute and relative line numbers.

2. Copy Current File Path

```
function CopyFilePath()
  local filepath = vim.fn.expand('%:p')
  vim.fn.setreg('+', filepath)
  print("Copied: " .. filepath)
end

vim.api.nvim_set_keymap('n', '<leader>fp', ':lua CopyFilePath()<CR>', { noremap =
  true, silent = true })
```

👉 Copies the full path of the current file to clipboard.

3. Reload Current Config

```
function ReloadConfig()
  for name, _ in pairs(package.loaded) do
    if name:match('^user') then
      package.loaded[name] = nil
    end
  end
  dofile(vim.env.MYVIMRC)
  print("Config reloaded!")
end

vim.api.nvim_set_keymap('n', '<leader>rc', ':lua ReloadConfig()<CR>', { noremap =
true, silent = true })
```

👉 Reloads your Neovim config without restarting.

4. Toggle Spell Checking

```
function ToggleSpell()
  vim.opt.spell = not vim.opt.spell:get()
  print("Spell check: " .. tostring(vim.opt.spell:get()))
end

vim.api.nvim_set_keymap('n', '<leader>sp', ':lua ToggleSpell()<CR>', { noremap =
true, silent = true })
```

👉 Handy for writing prose or documentation.

5. Trim Trailing Whitespace

```
function TrimWhitespace()
  local save_cursor = vim.fn.getpos(".")
  vim.cmd([[%s/\s\+$//e]])
  vim.fn.setpos(".", save_cursor)
  print("Whitespace trimmed")
end

vim.api.nvim_set_keymap('n', '<leader>tw', ':lua TrimWhitespace()<CR>', { noremap =
true, silent = true })
```

👉 Cleans up trailing spaces without moving your cursor.

6. Toggle Transparency (for GUI clients)

```
function ToggleTransparency()
  if vim.g.transparent_enabled then
    vim.cmd("hi Normal guibg=NONE ctermbg=NONE")
    vim.g.transparent_enabled = false
  end
end
```

```

    print("Transparency disabled")
else
    vim.cmd("hi Normal guibg=NONE ctermbg=NONE")
    vim.g.transparent_enabled = true
    print("Transparency enabled")
end
end

vim.api.nvim_set_keymap('n', '<leader>tt', ':lua ToggleTransparency()<CR>', {
    noremap = true, silent = true })

```

👉 Makes your background transparent (works in GUI clients like Neovide/Kitty).

🚀 How to Use

- Drop these functions into your `init.lua` or a Lua module (e.g., `lua/custom/functions.lua`).
- Adjust `<leader>` mappings to your preference.
- No external plugins required — pure Neovim API.

Since you love modularity and ergonomic utilities in C, these Lua snippets are like **utility functions for your editor** — small, composable, and transparent in behavior.

Would you like me to **bundle these into a single `functions.lua` module** with automatic loading, so you can just `require("functions")` in your config?

Perfect, Fudzer ⚡ — let's expand your **utility toolbox** with more ergonomic Lua functions that feel natural in daily Neovim use. These are still **plug-and-play**, no plugins required, just pure Lua + Neovim API.

🔧 More Ergonomic Neovim Lua Functions

7. Toggle Wrap Mode

```

function ToggleWrap()
    vim.wo.wrap = not vim.wo.wrap
    print("Wrap mode: " .. tostring(vim.wo.wrap))
end

vim.api.nvim_set_keymap('n', '<leader>ww', ':lua ToggleWrap()<CR>', { noremap =
    true, silent = true })

```

👉 Switch between wrapped and unwrapped text — great for prose vs. code.

8. Open Terminal in Split

```

function OpenTerminalSplit()
    vim.cmd("split term://$SHELL")
end

```

```
vim.api.nvim_set_keymap('n', '<leader>ts', ':lua OpenTerminalSplit()<CR>', { noremap = true, silent = true })
```

👉 Instantly open a terminal in a horizontal split.

9. Quick Save & Source Current File

```
function SaveAndSource()  
  vim.cmd("write")  
  vim.cmd("source %")  
  print("File saved and sourced!")  
end  
  
vim.api.nvim_set_keymap('n', '<leader>ss', ':lua SaveAndSource()<CR>', { noremap = true, silent = true })
```

👉 Perfect for testing Lua configs or Vimscript snippets.

10. Toggle Cursorline

```
function ToggleCursorline()  
  vim.wo.cursorline = not vim.wo.cursorline  
  print("Cursorline: " .. tostring(vim.wo.cursorline))  
end  
  
vim.api.nvim_set_keymap('n', '<leader>cl', ':lua ToggleCursorline()<CR>', { noremap = true, silent = true })
```

👉 Highlights the current line for better focus.

11. Yank Current Word

```
function YankWord()  
  local word = vim.fn.expand("<cword>")  
  vim.fn.setreg('+', word)  
  print("Yanked word: " .. word)  
end  
  
vim.api.nvim_set_keymap('n', '<leader>yw', ':lua YankWord()<CR>', { noremap = true, silent = true })
```

👉 Copies the word under cursor directly to clipboard.

12. Toggle Paste Mode

```
function TogglePaste()  
  vim.o.paste = not vim.o.paste
```

```

    print("Paste mode: " .. tostring(vim.o.paste))
end

vim.api.nvim_set_keymap('n', '<leader>pp', ':lua TogglePaste()<CR>', { noremap =
true, silent = true })

```

👉 Avoids auto-indentation when pasting code.

13. Close All Buffers Except Current

```

function CloseOtherBuffers()
    local current = vim.fn.bufnr('%')
    for _, buf in ipairs(vim.api.nvim_list_bufs()) do
        if vim.api.nvim_buf_is_loaded(buf) and buf ~= current then
            vim.api.nvim_buf_delete(buf, {})
        end
    end
    print("Closed all other buffers")
end

vim.api.nvim_set_keymap('n', '<leader>bo', ':lua CloseOtherBuffers()<CR>', { noremap
= true, silent = true })

```

👉 Keeps your workspace clean by closing everything except the current buffer.

14. Toggle Search Highlight

```

function ToggleSearchHighlight()
    vim.o.hlsearch = not vim.o.hlsearch
    print("Search highlight: " .. tostring(vim.o.hlsearch))
end

vim.api.nvim_set_keymap('n', '<leader>sh', ':lua ToggleSearchHighlight()<CR>', {
noremap = true, silent = true })

```

👉 Quickly hide or show search highlights.

Ergonomic Philosophy

These functions are:

- **Minimal** → no dependencies, pure Neovim API.
- **Transparent** → each prints its state so you know what happened.
- **Composable** → small utilities you can mix into workflows.

Since you're building ergonomic utilities like you do in C, these are like **editor syscalls** — small primitives you can wire into bigger workflows.

Would you like me to **design a modular `utils.lua` file** that groups these into categories (navigation, editing, UI toggles), so you can `require("utils")` and keep your config clean?

Excellent — let's push further into **new ergonomic utilities** that haven't been covered in the previous sets. These are still pure Lua + Neovim API, no plugins, and designed to be **plug-and-play**.

🌱 Fresh Utility Functions for Neovim

15. Toggle Colorcolumn at 80 chars

```
function ToggleColorColumn()  
  if vim.wo.colorcolumn ~= "" then  
    vim.wo.colorcolumn = ""  
    print("Colorcolumn disabled")  
  else  
    vim.wo.colorcolumn = "80"  
    print("Colorcolumn set at 80")  
  end  
end  
  
vim.api.nvim_set_keymap('n', '<leader>cc', ':lua ToggleColorColumn()<CR>', { noremap = true, silent = true })
```

👉 Helps enforce coding style by showing/hiding a vertical guide.

16. Open Current File in System Explorer

```
function OpenInExplorer()  
  local path = vim.fn.expand('%:p:h')  
  if vim.fn.has('mac') == 1 then  
    vim.fn.jobstart({'open', path})  
  elseif vim.fn.has('unix') == 1 then  
    vim.fn.jobstart({'xdg-open', path})  
  elseif vim.fn.has('win32') == 1 then  
    vim.fn.jobstart({'explorer', path})  
  end  
  print("Opened folder: " .. path)  
end  
  
vim.api.nvim_set_keymap('n', '<leader>fe', ':lua OpenInExplorer()<CR>', { noremap = true, silent = true })
```

👉 Opens the directory of the current file in your OS file explorer.

17. Toggle Background (light/dark)

```
function ToggleBackground()  
  if vim.o.background == "dark" then
```

```

    vim.o.background = "light"
else
    vim.o.background = "dark"
end
print("Background set to: " .. vim.o.background)
end

vim.api.nvim_set_keymap('n', '<leader>bg', ':lua ToggleBackground()<CR>', { noremap = true, silent = true })

```

👉 Quickly switch between light and dark modes.

18. Insert Current Date/Time

```

function InsertDateTime()
    local dt = os.date("%Y-%m-%d %H:%M:%S")
    vim.api.nvim_put({dt}, 'c', true, true)
end

vim.api.nvim_set_keymap('n', '<leader>dt', ':lua InsertDateTime()<CR>', { noremap = true, silent = true })

```

👉 Inserts a timestamp at the cursor — great for notes or logs.

19. Toggle Listchars (show hidden chars)

```

function ToggleListChars()
    vim.wo.list = not vim.wo.list
    print("Listchars: " .. tostring(vim.wo.list))
end

vim.api.nvim_set_keymap('n', '<leader>lc', ':lua ToggleListChars()<CR>', { noremap = true, silent = true })

```

👉 Shows/hides tabs, trailing spaces, and other invisible characters.

20. Reload Current Buffer from Disk

```

function ReloadBuffer()
    vim.cmd("edit!")
    print("Buffer reloaded from disk")
end

vim.api.nvim_set_keymap('n', '<leader>rb', ':lua ReloadBuffer()<CR>', { noremap = true, silent = true })

```

👉 Discards changes and reloads the file from disk.

21. Toggle Mouse Support

```
function ToggleMouse()
  if vim.o.mouse == "a" then
    vim.o.mouse = ""
    print("Mouse disabled")
  else
    vim.o.mouse = "a"
    print("Mouse enabled")
  end
end

vim.api.nvim_set_keymap('n', '<leader>ms', ':lua ToggleMouse()<CR>', { noremap =
true, silent = true })
```

👉 Enables/disables mouse interaction on demand.

22. Diff Current Buffer Against Saved File

```
function DiffSaved()
  vim.cmd("diffthis")
  vim.cmd("vsplit | edit # | diffthis")
end

vim.api.nvim_set_keymap('n', '<leader>df', ':lua DiffSaved()<CR>', { noremap = true,
silent = true })
```

👉 Opens a side-by-side diff between the current buffer and its saved version.

🎯 Why These?

- They extend ergonomics beyond editing into **workflow utilities** (explorer, timestamps, diffs).
 - They keep the **transparent + modular philosophy** you value: each prints state or action.
 - They're **orthogonal** to the earlier sets — no overlap with toggles already mentioned.
-

Would you like me to **bundle all 22 functions into a categorized `utils.lua` module** (Editing, UI Toggles, Workflow), so you can drop it in and just `require("utils")` ? That way it feels like your own ergonomic **standard library for Neovim**.

Nice pivot 🚀 — now we can design **plugin-inspired functions** that mimic or approximate popular plugin features, but still stay lightweight and ergonomic. These are “poor man’s versions” of plugin functionality, written in pure Lua so you can drop them into your config without installing anything extra.

🔧 Plugin-Inspired Custom Functions (No Plugin Required)

23. Simple File Explorer (like *NvimTree*)


```
function SimpleExplorer()
    vim.cmd("vsplit")
    vim.cmd("Ex")
end

vim.api.nvim_set_keymap('n', '<leader>e', ':lua SimpleExplorer()<CR>', { noremap =
true, silent = true })
```

👉 Opens a vertical split with the built-in netrw file explorer, mimicking NvimTree.

24. Quick Grep Search (like *Telescope live_grep*)

```
function QuickGrep()
    local word = vim.fn.input("Grep for > ")
    if word ~= "" then
        vim.cmd("vimgrep /".word.."/gj **/*")
        vim.cmd("copen")
    end
end

vim.api.nvim_set_keymap('n', '<leader>fg', ':lua QuickGrep()<CR>', { noremap = true,
silent = true })
```

👉 Prompts for a word, searches across your project, and opens results in quickfix.

25. Buffer Switcher (like *Bufferline*)

```
function NextBuffer()
    vim.cmd("bnext")
end

function PrevBuffer()
    vim.cmd("bprevious")
end

vim.api.nvim_set_keymap('n', '<leader>bn', ':lua NextBuffer()<CR>', { noremap =
true, silent = true })
vim.api.nvim_set_keymap('n', '<leader>bp', ':lua PrevBuffer()<CR>', { noremap =
true, silent = true })
```

👉 Simple buffer navigation, inspired by bufferline's tab switching.

26. Fuzzy File Search (like *Telescope find_files*)

```
function FuzzyFind()
    local file = vim.fn.input("Find file > ")
    if file ~= "" then
```

```

        vim.cmd("find " .. file)
    end
end

vim.api.nvim_set_keymap('n', '<leader>ff', ':lua FuzzyFind()<CR>', { noremap = true,
silent = true })

```

👉 Lightweight file search using Vim's built-in `find`.

27. Git Status Viewer (like *Fugitive*)

```

function GitStatus()
    vim.cmd("vsplit | term git status")
end

vim.api.nvim_set_keymap('n', '<leader>gs', ':lua GitStatus()<CR>', { noremap = true,
silent = true })

```

👉 Opens a terminal split showing `git status`, inspired by Fugitive.

28. Markdown Preview (like *MarkdownPreview*)

```

function MarkdownPreview()
    local file = vim.fn.expand('%:p')
    vim.fn.jobstart({"xdg-open", file})
    print("Preview opened in browser")
end

vim.api.nvim_set_keymap('n', '<leader>mp', ':lua MarkdownPreview()<CR>', { noremap =
true, silent = true })

```

👉 Opens the current markdown file in your system's default browser.

29. Comment Toggler (like *Comment.nvim*)

```

function ToggleComment()
    local line = vim.fn.getline('.')
    if line:match("^%s*//") then
        vim.fn.setline('.', line:gsub("^%s*//", "", 1))
    else
        vim.fn.setline('.', "// " .. line)
    end
end

vim.api.nvim_set_keymap('n', '<leader>/', ':lua ToggleComment()<CR>', { noremap =
true, silent = true })

```

👉 Simple comment toggle for C-like languages.

30. Session Saver (like *possession.nvim* or *obsession.vim*)

```
function SaveSession()
  local session = vim.fn.input("Session name > ")
  if session ~= "" then
    vim.cmd("mksession! " .. session .. ".vim")
    print("Session saved: " .. session)
  end
end

function LoadSession()
  local session = vim.fn.input("Load session > ")
  if session ~= "" then
    vim.cmd("source " .. session .. ".vim")
    print("Session loaded: " .. session)
  end
end

vim.api.nvim_set_keymap('n', '<leader>ss', ':lua SaveSession()<CR>', { noremap =
true, silent = true })
vim.api.nvim_set_keymap('n', '<leader>sl', ':lua LoadSession()<CR>', { noremap =
true, silent = true })
```

👉 Save and restore sessions without plugins.

🎯 Why These?

- They **emulate plugin workflows** (file explorer, fuzzy search, git, comments, sessions).
- They're **ergonomic primitives** — small Lua wrappers around built-ins.
- They let you taste plugin-like functionality without dependency overhead.

⚡ Since you're passionate about extending C with ergonomic utilities, this is like building a **"stdlib for Neovim"**: plugin-inspired functions that you can expand modularly.

Would you like me to **design a categorized "plugin-lite" module** (e.g., `plugin_utils.lua`) grouping these by theme (Git, Search, UI, Editing), so you can drop it in as a pseudo-plugin?
