Contents

Neovim plugins

The day-to-day plugins on the **`nvim-light`** profile, with the keymaps as they are actually wired here. This is the LazyVim distro (`<leader>` is **Space**); this page only covers the plugins worth knowing on purpose — the rest stay out of the way.

!!! note “Two profiles: `nvim-light` vs `nvim-full`” `~/.config/nvim` points at **`nvim-light/`**: notes (obsidian, dotmd), prose and music (pencil, strudel), plus **Rust** and **LaTeX** coding and Claude. The heavier **`nvim-full`** profile adds the IDE layer — code outline (aerial/navic/outline), **refactoring**, inc-rename, testing, eslint, and the web/devops languages (TypeScript, Docker, SQL, YAML, TOML). So `nvim-light` isn’t coding-free; it just skips the IDE machinery and the web stack. Neither profile currently enables a `lang.elixir` extra.

Quick reference

Keys	Does	Plugin
`<leader>sf` / `<leader>sg`	find files / live grep	fff.nvim
`<leader>sz` / `<leader>sw`	fuzzy grep / grep word under cursor	fff.nvim
`s` / `S` / `gs` + 2 chars	leap forward / back / across windows	leap.nvim
`<leader>sr`	project-wide search & replace	grug-far.nvim
`<leader>p`	yank history picker	yanky.nvim
`<leader>cf` / `<leader>cm`	format buffer / open Mason	conform / mason
`<leader>a…`	Claude menu (toggle, send, accept/deny)	claudecode.nvim
`<leader>gg` / `<leader>.`	lazygit / scratch buffer	snacks.nvim
`:Obsidian …`	notes, daily journal, search, backlinks	obsidian.nvim

Finding & searching

fff.nvim

A Rust-powered fuzzy file picker with frecency ranking (frequent/recent and git-dirty files float to the top). It builds a native binary on install. The keymaps here are custom — note they live under `<leader>s` (search), not the LazyVim default `<leader>f`:

Keys	Action
`<leader>sf`	Find files
`<leader>sg`	Live grep
`<leader>sz`	Live grep, fuzzy + plain modes
`<leader>sw`	Grep the word under the cursor

Recent additions: **resume the last picker**, jump to the next matching segment, and more control over query parsing.

grug-far.nvim

Project-wide find **and replace** in an editable buffer (think a scratch buffer that rewrites your files). Open with `<leader>sr` — the word under the cursor or the current visual selection is prefilled.

The top of the buffer holds the **Search / Replace / Files Filter / Flags** fields; results stream below. Edit the fields, then run the replace action shown in the buffer header (`<localleader>` actions — `<localleader>` is `\` by default). Great for rename-across-repo when LSP rename can't reach (strings, comments, config files).

Moving around

leap.nvim
Jump anywhere on screen with two keystrokes. Type the motion key, then the two characters at your target; press a label if more than one match remains.

Keys	Action
`s` + 2 chars	Leap forward
`S` + 2 chars	Leap backward
`gs` + 2 chars	Leap across all visible windows

Works in operator-pending mode too — `d` then `s` + 2 chars deletes up to the target.

!!! note "`gs` is leap's; surround lives on `gz`" mini.surround is mapped to `gz*` (`gza` add, `gzd` delete, `gzr` replace, `gzf`/`gzF` find, `gzh` highlight) so leap owns `gs` (leap-across-windows) with no `timeoutlen` delay. This matches the LazyVim leap extra's own default.

Editing

yanky.nvim
A yank ring with history — every yank is remembered, not just the last one.

Keys	Action
`<leader>p`	Open the yank-history picker
`y` / `p` / `P`	Yank / put after / put before (history-aware)
`gp` / `gP`	Put after / before the selection
`[y` / `]y`	After a paste, cycle to the previous / next ring entry
`]p` / `[p`	Put with the current indent

The usual flow: paste with `p`, then `[y`/`]y` to swap in an older clip — or `<leader>p` to pick visually.

Formatting & tooling

conform.nvim
Runs the right formatter per filetype, and **formats on save** by default.

- `<leader>cf` — format the buffer (also works on a visual selection)
- `<leader>cF` — format injected languages (e.g. code blocks in markdown)
- `:ConformInfo` — see which formatters apply to this buffer and their status

Recent: added `openapi-format`, oxfmt/oxlint/prettierd config fixes, and Neovim 0.12 compatibility.

mason.nvim
Installs and manages external tooling — LSP servers, formatters, linters, DAP adapters — outside your system package manager.

- `<leader>cm` (or `:Mason`) — open the UI
- In the UI: `i` install, `u` update package, `U` update all, `X` uninstall, `g?` help
- `mason-lspconfig` auto-installs the servers your LSP config asks for

AI

claudecode.nvim
Claude Code, embedded in Neovim, talking to the same buffers you're editing. All under the `<leader>a` menu:

Keys	Action
`<leader>ac`	Toggle the Claude terminal
`<leader>af`	Focus Claude
`<leader>ar` / `<leader>aC`	Resume / continue a session
`<leader>ab`	Add the current buffer to context
`<leader>as`	Send selection (visual) / add file (in a tree)
`<leader>aa` / `<leader>ad`	Accept / deny the proposed diff

Recent: **VS Code-style inline (unified) diffs** via `layout = "inline"`, `:checkhealth claudecode`, and a terminal `auto_insert` option that preserves scroll position.

The utility layer

snacks.nvim
A bundle of small QoL modules that LazyVim leans on — the dashboard, the notifier, lazygit integration, scratch buffers, a terminal, big-file handling, and more. The ones you'll reach for:

Keys	Action
`<leader>gg`	Lazygit (root dir) — `<leader>gG` for cwd
`<leader>gb` / `<leader>gB`	Git blame line / open in browser
`<leader>.` / `<leader>S`	Toggle / select a scratch buffer
`<leader>n` / `<leader>un`	Notification history / dismiss all
`<c-/>`	Toggle the terminal
`<leader>cR`	Rename the current file (updates imports)

Recent fix: closing one or more buffers no longer flickers the screen.

Notes

obsidian.nvim
Edit an Obsidian vault as plain markdown, with links, tags, templates and daily notes. Two workspaces are configured — **Suika Box** (`~/Documents/Suika Box`) and **Killua Chest** — with new notes under `00.notes`, journals under `00.journals`, and templates in `03.resources/templates`. Completion is wired through **blink.cmp** (triggers at 2 chars).

Commands use the `:Obsidian` prefix (legacy bare commands are off):

Command	Action
`:Obsidian today` / `tomorrow` / `yesterday`	Open the daily note
`:Obsidian new`	New note
`:Obsidian search` / `quick_switch`	Grep / switch notes
`:Obsidian backlinks` / `tags`	Find references / tag usage
`:Obsidian template`	Insert a template
`:Obsidian bookmarks`	List Obsidian bookmarks

Recent gains: native **LSP completion**, **Bases** (`.base`) support, end-to-end-encrypted sync, proper unicode tags, and configurable default keymaps.

!!! tip "Lighter quick-capture: dotmd" For throwaway notes, todos and journals outside the vault, `dotmd.nvim` lives under `<leader>z` (create note `<leader>zc`, today's todo `<leader>zt`, journal `<leader>zj`, inbox `<leader>zi`, search `<leader>zs`…).

Tips & combos

Common surrounds (mini.surround, gz)

Add is `gza` + a textobject + the surrounding character; in Visual mode just select then `gza` + the character.

Goal	Keys
Quote a word	`gzaiw"`
Parens around a word	`gzaiw)`
Surround a visual selection	*(select)* `gza)`
Surround to end of line	`gza$"`
Delete surrounding quotes	`gzd"`
Change `"` → `'`	`gzr"'`
Change `()` → `[]`	`gzr)]`
Wrap in an HTML tag	`gzaiwt` then type the tag
Wrap in a function call	`gzaiwf` then type the name
Jump to the next `)` surrounding	`gzf)`

Discover keys as you go

- Press `<leader>` (Space) and **wait** — which-key shows every follow-up.
- mini.ai text objects pair with any operator: `ciw` is built-in, but also `cif` (change inside function), `daa` (delete an argument), `vai` (select around indent).

Sessions

Sessions auto-save per directory (persistence.nvim):

- `<leader>qs` — restore this directory's session
- `<leader>ql` — restore the **last** session (any directory)
- `<leader>qS` — pick a session; `<leader>qd` — stop saving the current one

Numbers, dates & toggles

dial.nvim supercharges `<C-a>` / `<C-x>`: increment/decrement not just numbers but dates, booleans (`true`↔`false`), and more. In Visual mode, `g<C-a>` makes a column count up sequentially (1, 2, 3…).

Notes & writing

- `:PencilToggle` — soft-wrap prose mode (auto for `markdown`/`text`/`tex`)
- `<leader>um` — toggle in-editor markdown rendering (render-markdown.nvim)
- `<leader>cp` — open the browser markdown preview (markdown-preview.nvim)

LaTeX (vimtex)

Local-leader (\) drives vimtex in `.tex` files:

- `\ll` — start/stop continuous compilation
- `\lv` — forward-search to the PDF viewer
- `\lt` — table of contents; `\le` — error/quickfix list; `\lc` — clean aux files
- `<leader>K` — package docs (plain `K` is left to LSP hover)

Refactoring — and why Elixir is different

The plugin people mean by "the refactoring plugin" is **`refactoring.nvim`** (treesitter-based code surgery): extract function / variable, inline them, extract a block to a new file, plus debug-print helpers. In LazyVim it's the `editor.refactoring` extra, mapped under `<leader>r`.

!!! warning "Not installed here, and it can't help Elixir" `refactoring.nvim` is **not** on `nvim-light` (it's part of `nvim-full`). More importantly, it supports a **fixed** language list — TypeScript, JavaScript, Lua, Go, Python, C, C++, Java, PHP, Ruby. **Elixir is not supported**, so extract/inline never fire in `.ex`/`.exs` buffers regardless of which profile you're in.

For Elixir, refactoring comes from the language server and CLI tools instead:

Need	Tool	How
Rename a symbol	LSP (ElixirLS / Lexical / Expert)	`<leader>cr`
Code actions (add `@spec`, pipe conversions…)	LSP	`<leader>ca`
Manual extract by structure	mini.ai text objects	`vaf` / `vif` (a/inside function), then move it
Project-wide automated refactors	Recode (CLI)	`mix recode`
Refactoring suggestions + lint	Credo	`mix credo --strict`

!!! note "No Elixir LSP on this profile yet" `nvim-light` has no `lang.elixir` extra, so `<leader>ca` / `<leader>cr` have nothing to act on in Elixir files here. If you want Elixir

support on this machine, add `lazyvim.plugins.extras.lang.elixir` (ElixirLS + treesitter + formatting) — ask and I'll wire it in.