

rdlu's dotfiles handbook

2026-10-17

Contents

Setup guide	4
Fresh machine	4
The stow model	4
Services	5
Tool versions: pacman + mise	5
Day to day	5
Docs	5
File transfer	5
Install	5
LocalSend — phone ↔ computer	6
rsync over SSH — computer ↔ computer	6
Authorize a sender's key (one-time)	6
Receive a transfer	7
How the temporary opening works	7
Security model	7
Google Drive	7
Create the Google OAuth client (once per host)	7
Subcommands	8
How rotate-password avoids a lockout	9
Where it lives	9
Calendar	9
Install	10
How it fits together	10
Create the Google OAuth client (once per host)	10
Subcommands	11
Failure and Ctrl-C	12
What lives where (per host)	13
chroncal and oo7: the plaintext fallback	13
Troubleshooting	14
Files	14
Neovim plugins	14
Quick reference	15
Finding & searching	15
fff.nvim	15
grug-far.nvim	16
Moving around	16
leap.nvim	16
Editing	16
yanky.nvim	16

Formatting & tooling	17
conform.nvim	17
mason.nvim	17
AI	17
claudecode.nvim	17
The utility layer	17
snacks.nvim	17
Notes	18
obsidian.nvim	18
Tips & combos	18
Common surrounds (mini.surround, gz)	18
Discover keys as you go	19
Sessions	19
Numbers, dates & toggles	19
Notes & writing	19
LaTeX (vimtex)	19
Refactoring — and why Elixir is different	20
Justfile reference	20
Top-level	21
Install: essentials	21
Install: graphical stack	21
Install: optional extras	22
Stow (symlink management)	22
Services	22
niri: reload helpers	23
Maintenance	23
Docs (this site)	24
calendar	24
file-transfer	25
tmux shortcuts	25
Custom bindings	25
After the prefix	25
No prefix (root table)	26
Copy mode (vi keys)	27
OFF mode	27
tmux defaults worth remembering	27
Plugin bindings (TPM)	27
copycat	28
yank	28
extrakto (fuzzy search)	28
resurrect + continuum	28
pain-control (panes)	28
sessionist (sessions)	29
sessionx (session picker)	29
open	29
thumbs	29

logging	30
fzf	30
herdr shortcuts	30
Keybindings	30
Tabs	30
Panels	31
Workspaces & sessions	31
Agents	31
Scrollbar & misc	32
Custom commands	32
Direct keys (no prefix)	32
zellij shortcuts	32
Tmux mode — Ctrl a	33
Always available (normal & locked)	34
Pane mode — p	34
Tab mode — t	35
Resize mode — r	35
Move mode — m	36
Scroll mode — s	36
Scrolling keys (scroll & search)	36
Search options	37
Session mode — o	37
Locked mode (default)	37
Mode switches & misc	37
niri shortcuts	38
Apps & launchers	38
Mouse warp (wl-kbptr)	39
Media, volume & brightness	40
Focus	40
Moving windows & columns	41
Workspaces	42
Monitors	44
Layout & sizing	45
Screenshots & screencasting	45
Session & misc	45
Shell shortcuts (fish)	46
Custom bindings	46
thefuck — fix the last command	46
History — atuin	46
fish built-ins worth remembering	47
Handy abbreviations & functions	47
wl-kbptr	48
Install	48
Binds	48
Mid-stream click keys	48
Modes	49

Cell-pick keys (home_row_keys)	49
Universal keys	49
Useful CLI flags	50
Multi-monitor notes	50
Gotchas	50
Files	50

Setup guide

How a fresh CachyOS/Arch machine becomes one of mine. Everything is driven by just — run just from the repo root to list every recipe (full list in the justfile reference).

Fresh machine

```
git clone git@github.com:rdlu/dotfiles.git ~/.dotfiles
cd ~/.dotfiles
```

```
just full-auto      # base packages, languages, CLI tools, fish, yazi,
fastfetch
just full-auto-gui  # the above + kitty + the niri/Noctalia graphical stack
```

full-auto chains: packages → dev-setup → cli-tools → fish-shell → helix-editor → yazi-file-manager → fastfetch. The packages step is interactive — it pauses so you can review /etc/xdg/reflector/reflector.conf and add the Chaotic AUR include to /etc/pacman.conf.

On a host that loads the audio-production category, also add this under [options] in /etc/pacman.conf. zam-plugins pulls in zam-plugins-standalone, which ships one launcher entry per plugin (17 of them) that clutter the app launcher; the plugins and standalone binaries are unaffected. Reinstall the package afterwards to drop already-installed entries:

```
NoExtract = usr/share/applications/com.zamaudio.*
```

If you are not running full-auto, run just packages first: it sets up Flatpak/Flathub, reflector mirrors, the Chaotic AUR repository, and the paru AUR helper that every other recipe uses.

The stow model

Configuration is deployed as symlinks with **GNU Stow** (--dotfiles mode: dot-foo in the repo becomes ~/.foo).

- **Base packages** (stowed on every machine): home git idea scripts terminal lazyvim vim systemd niri
- **Host overlays** in hosts/<hostname>/ (stowed only on the matching machine): display layout, thermal sensors, machine-specific tools. The two hosts are daisy (AMD) and xps (Intel).

```
just stow          # symlink base + this host's overlay (auto-detected)
just stow-check    # dry-run: show what would change, touch nothing
just unstow        # remove all symlinks (host overlay first, then base)
```

--no-folding is used so base and host overlays can both populate shared directories (~/.config/mise, ~/.local/share/applications) without tree-folding conflicts.

Services

```
just services-enable
```

Enables the expected systemd **user** services (mpd, mpd-mpris, syncthing if installed, solaar) and Noctalia, the niri session shell (bar, notifications, lock, idle), which starts with the graphical session.

Tool versions: pacman + mise

Two deliberate layers:

- **pacman/paru** installs tools available in the repos (`just cli-tools`).
- **mise** pins versions for languages and tools *not* in the repos: shared config in `terminal/dot-config/mise/config.toml`, per-host additions in `hosts/<hostname>/dot-config/mise/conf.d/<hostname>.toml`.

Day to day

```
just update    # paru -Syu, mise upgrade, fisher, yazi plugins, nvim Lazy sync
just doctor    # health check: tools, login shell, host overlay, services,
               # symlinks
```

Docs

This site itself is part of the repo:

```
just docs-setup  # one-time: install hugo, pandoc, typst and the fonts
just docs        # regenerate markdown from configs, build HTML site + PDFs
just docs-serve  # live-preview the site while editing
```

`tools/gen-docs.py` (Python stdlib, no deps) re-reads `tmux.conf`, `binds.kdl`, and the `justfile` and rewrites the generated blocks in the markdown. Hugo with the vendored Hextra theme builds the HTML site from `site/`, which mounts `docs/` as its content, into `site/public/`; GitHub Pages serves it at `dotfiles.dlu.one`. Each page starts with Hugo front matter (title, linkTitle for the sidebar, weight for its order) and keeps its own `# H1`, which only the PDFs show. `pandoc + Typst` render the PDFs into `docs/pdf/`, which the site links directly; `tools/pandoc-docs.lua` drops the front matter there. The `just` recipes are thin shims over the equivalent `mise` tasks (`mise run docs`, etc.).

File transfer

How files move on and off these machines without the cloud: **LocalSend** for phone ↔ computer, and **rsync over SSH** for computer ↔ computer. Everything is local-network only, opened *temporarily* on untrusted networks (and auto-closed), and installs from a single recipe.

The one cloud exception is a Google Drive mount at `~/GoogleDrive`, maintained with `rcclone-gdrive`.

Install

```
just file-transfer      # tools + firewall + ~/Downloads/Transfers + docs
just file-transfer-harden # receiving machines only: inbound SSH key-only +
                        # LLNMR off
```

file-transfer is idempotent and machine-agnostic — it opens 53317 on whatever firewall zone holds the active interfaces, and renders a per-host ~/Downloads/Transfers/README.md. The helper scripts and the niri menu entries arrive with just stow.

Warning: Why file-transfer-harden is a separate step

It disables SSH **password** login (key-only). Run it only on machines that will *receive* rsync, and make sure you have a key installed — or are sitting at the machine — first. sshd is left disabled; sshd-here starts it on demand.

LocalSend — phone ↔ computer

LocalSend discovers devices over UDP 53317 and transfers over TCP 53317.

- **Home network:** permanently allowed on the home zone — your phone just finds the machine.
- **Untrusted network:** open it for the session from the niri power menu (**Mod+Escape** twice → **Sharing**):

Action	Menu entry	Command
Open + launch	 LocalSend share (temp)	localsend-here on
Close now	 LocalSend close	localsend-here off

The opening is runtime-only with a **30-minute timeout**, so it reverts on its own (and a reboot wipes it).

rsync over SSH — computer ↔ computer

Direction decides what you need:

- **This machine → remote** (you push out): nothing to set up — `rsync -av ~/stuff/ user@remote:/dest/`.
- **Remote → this machine** (something connects in): inbound SSH is key-only and every transfer key is locked to ~/Downloads/Transfers.

Authorize a sender's key (one-time)

`add-rsync-key 'ssh-ed25519 AAAA... you@laptop'`

Option	Effect
<i>(default)</i>	upload-only (-wo -no-del) into ~/Downloads/Transfers
--ro	download-only
--dir NAME	lock to ~/NAME instead

The key is installed with a **forced command**:

```
restrict,command="/usr/bin/rrsync -wo -no-del Downloads/Transfers" ssh-ed25519
...
```

restrict strips PTY and forwarding; rrsync (see `man rrsync`) refuses anything but rsync and jails it to that folder. So the key can do nothing but drop files there — no shell, no other paths — even if it leaks.

Receive a transfer

1. Open: menu  **SSH/rsync in (temp)** or `ssh-here on` — starts sshd, opens 22 for 30 minutes.
2. Sender: `rsync -av ./files/ <host>`: (the path is relative to the inbox folder).
3. Close: menu  **SSH/rsync close** or `ssh-here off`.

How the temporary opening works

localsend-here and ssh-here share one pattern:

1. Find the zone of the default-route interface — the network you're *actually* on.
2. Add the port to that zone **at runtime only**, with `--timeout=30m`.
3. Never --permanent, so it's gone on reboot/reload and can't silently persist.

That's why a conference-Wi-Fi opening is safe: scoped to the live network, and self-closing.

Security model

- Inbound SSH is **public-key only**, no root — `/etc/ssh/sshd_config.d/10-hardening.conf`.
- sshd is **disabled**; started on demand by `ssh-here`, stopped on `off`.
- Transfer keys are rsync-only, upload-only, no-delete, single-folder.
- LLMNR is disabled (`file-transfer-harden`) — SMB discovery (mDNS + NetBIOS) is unaffected.
- Every temporary firewall hole is runtime-only with a 30-minute expiry.

Google Drive

`rclone-gdrive.service` (systemd user unit) runs `rclone mount gdrive: ~/GoogleDrive` with a full VFS cache. Everything that makes it work is **per host** and never in this repo:

- the rclone config `~/.config/rclone/rclone.conf` (Drive token + OAuth client), **encrypted**;
- its password, in the Secret Service (oo7): `secret-tool lookup service rclone key config_password`;
- a Google OAuth client of its own (Desktop app), **one per host**.

`rclone-gdrive` (also just `gdrive <subcommand>`) sets that up and maintains it.

Create the Google OAuth client (once per host)

In the Google Cloud Console, in any project of yours:

1. **APIs & Services** → **Library** → enable the **Google Drive API**.
2. **OAuth consent screen** (Google Auth Platform): user type **External**, then under **Audience** click **Publish app** so the status is **In production**.
3. **Credentials** (Clients) → **Create credentials** → **OAuth client ID** → application type **Desktop app**, named after the host (xps, daisy). Keep the client ID and secret for `rclone-gdrive` setup.

Warning: Publish the app — don't leave it in Testing

While the consent screen is in **Testing**, Google expires the refresh token after **7 days** and the mount stops working every week. Published but *unverified* is fine for personal use: at consent Google shows “Google hasn't verified this app” — click **Advanced** → **Go to ... (unsafe)**.

One client per host means a host can be revoked or rotated without touching the other.

Subcommands

Command	When
<code>rclone-gdrive setup</code>	First time on a host. Checks the tools, generates the keyring password if there is none (never shown), encrypts an existing plaintext config, asks for the OAuth client and opens the browser for consent, then enables and starts the unit and checks the mount. Idempotent: on a finished host it just reports everything is fine.
<code>rclone-gdrive status</code>	Anytime. Read-only: tools, keyring password (yes/no), config encrypted and decryptable, gdrive: present, token works (one Drive API call, on a throwaway copy of the config), unit enabled + active, mount up. Exit 1 if anything is wrong; never prints a secret.
<code>rclone-gdrive reauth</code>	The token expired or was revoked (status says so, the mount errors). Re-consent in the browser.
<code>rclone-gdrive rotate-client</code>	New OAuth client (the old one leaked, or you're cleaning up). Asks for the new ID + secret and re-consents (the token is bound to the client), then reminds you to delete the old client in the console.
<code>rclone-gdrive rotate-password</code>	New keyring password for the config encryption. Also resolves a rotation that was interrupted.

At consent rclone asks two questions: answer **y** to “Use web browser” and **n** to “Shared Drive”. Every command that rewrites the config stops the unit first (the running mount may rewrite the token) and starts it again afterwards. If reauth or rotate-client fails, or you press Ctrl-C at the browser step, the previous config (old client + token) is put back.

`--config PATH` (or `$RCLONE_CONFIG`) points it at another config, e.g. a throwaway one for testing. The unit always mounts the default config.

How rotate-password avoids a lockout

At every step, one of the passwords in the keyring decrypts one config on disk:

1. Back up the config (still on the old password) to `rclone.conf.rotate-backup`.
2. Generate the new password into a **staging** keyring entry (`key config_password_next`), next to the current one.
3. Re-encrypt straight from the old password to the new one (`rclone config encryption set` with a password command that returns each in turn), so no plaintext config is ever written. Check that the new one decrypts it and `gdrive:` is still there.
4. Promote: copy the staged password over `config_password`, check again, then delete the staging entry and the backup.

If a step fails, or the machine dies halfway, the next `rotate-password` (or `setup`) sees the leftovers and works out what to do from which password decrypts what: finish the promotion, restore the backup, or just clean up. If nothing decrypts, it stops and deletes nothing. `rclone`'s exit code can't be trusted here (`encryption set` exits 0 even when it fails), so every step is checked with `rclone config encryption check`.

Where it lives

Thing	Path
Helper scripts	<code>~/.local/bin/{localsend-here, ssh-here, add-rsync-key}</code>
System setup	<code>~/.local/bin/{setup-file-transfer, harden-file-transfer-ssh}</code>
Menu entries	niri power menu (rodii-power-menu, Tools → Sharing)
Doc templates	<code>~/.local/share/file-transfer/*.md</code> → rendered into <code>~/Downloads/Transfers/</code>
Packages	file-transfer category in <code>setup/packages.yaml</code>
Recipes	<code>just file-transfer</code> , <code>just file-transfer-harden</code> , <code>just gdrive <subcommand></code>
Google Drive	<code>~/.local/bin/rclone-gdrive</code> , <code>~/.config/systemd/user/rclone-gdrive.service</code>

Calendar

Two terminal calendars run side by side on both hosts (daisy and xps), each synced with Google Calendar:

- **khal + vdirsyncer**: `vdirsyncer` keeps a local `vdir` in sync with Google, and `khal` reads it.
- **chroncal**: a Go TUI/CLI with its own SQLite store and built-in Google sync.

`calendar-google` (also `just gcal <subcommand> [app]`) sets both up and keeps them working. Everything secret is **per host** and never in this repo.

Install

```
just pkg-install calendar # khal, vdirsyncer, python-aiohttp-oauthlib,
libsecret
mise install # chroncal (github:DouglasdeMoura/chroncal,
terminal mise config)
just stow # khal + vdirsyncer configs, calendar-google in
~/.local/bin
calendar-google setup # both apps; or: calendar-google setup vdirsyncer|
chroncal
```

How it fits together

	khal + vdirsyncer	chroncal
	Talks to Google CalDAV (apidata.googleusercontent.com), the same scope auth/calendar	
OAuth client	the Secret Service (007), fetched by vdirsyncer/config	inside chroncal's own credential
Token	~/.local/share/vdirsyncer/google_token	inside chroncal's own credential
Local data	~/.local/share/calendars/google/<calendar>/*.ics	~/.local/share/chroncal/chroncal.db
Background sync	vdirsyncer.timer (packaged): vdirsyncer sync every 15 min	chroncal-alarm.timer: chroncal service run every minute (alarms), sync every 15 min

By default both apps use **one** Google OAuth client per host. setup asks for it once, and setup and rotate-client let you give chroncal a separate one.

Create the Google OAuth client (once per host)

In the Google Cloud Console, in any project of yours (the Google Drive one works too):

1. **APIs & Services** → **Library**: enable the **Google Calendar API** and the **CalDAV API**. Both apps use CalDAV, and until that API is enabled Google answers 403 accessNotConfigured.
2. **OAuth consent screen** (Google Auth Platform): user type **External**. Then, under **Audience**, click **Publish app** so the status is **In production**.
3. **Credentials** (Clients) → **Create credentials** → **OAuth client ID**, with application type **Desktop app**, named after the host (xps calendar, daisy calendar). Keep the client ID and secret for calendar-google setup.

Warning: Publish the app. Don't leave it in Testing

While the consent screen is in **Testing**, Google expires refresh tokens after **7 days**, and both calendars stop syncing every week. Published but *unverified* is fine for personal use.

At consent, Google shows “Google hasn’t verified this app”: click **Advanced** → **Go to ... (unsafe)**, and tick the calendar access box if Google shows one.

With one client per host, you can revoke or rotate one host without touching the other.

Subcommands

app is vdirsyncer or chroncal. Without it, a command covers both apps.

Command	When
<code>calendar-google setup [app]</code>	First time on a host. Checks the tools and asks every question up front: the OAuth client (only if the keyring has none), whether chroncal should share it, and the Google account email for chroncal. Stores the client in oo7, then opens the browser once per app. vdirsyncer then runs discover, the first sync and metasync, and enables vdirsyncer.timer. chroncal runs account add (which imports every calendar and syncs) and service install. Idempotent: on a finished host it only checks, and it re-installs chroncal’s service if the unit points at a binary that no longer exists.
<code>calendar-google status [app]</code>	Anytime. Read-only: tools, keyring entries (yes/no), token file, a live token check for vdirsyncer, timers enabled + active + last run, calendars and events on disk, khal list, the chroncal account, where its credential lives, whether it shares vdirsyncer’s client, what the last background sync recorded, and the service binary. Exit 1 if anything is wrong. Never prints a secret or token.
<code>calendar-google reauth [app]</code>	A token expired or was revoked (status says <code>invalid_grant</code>). Re-consent in the browser.
<code>calendar-google rotate-client [app]</code>	New OAuth client (the old one leaked, or you’re cleaning up). Asks for the new ID + secret (optionally a separate one for chroncal), re-consents (tokens are bound to their client), then prints the old client ID(s) to delete in the console. It warns instead if the other app still uses the old client.

What each check touches:

- **vdirsyncer live check** (status, and after every consent): a CalDAV discovery with a throwaway config, a throwaway status dir and a *copy* of the token. A token refresh can't rewrite the real token, and nothing under `~/ .local/share` changes. It only runs when the token file holds a refresh token, with no display and a no-op `$BROWSER`, so it can never start a sign-in.
- **chroncal in status** makes no network call. It reads `chroncal sync status`, which is what the background sync recorded (an error, or a last sync more than an hour ago, counts as unhealthy). `setup`, `reauth` and `rotate-client` check `chroncal live` with `chroncal account calendars list`. Like any `chroncal` run, that may refresh the access token and update calendar names and colours in its database.
- `status` never creates `chroncal`'s database. On a host where `chroncal` never ran, it just reports that.

Failure and Ctrl-C

`reauth` and `rotate-client` back up what they replace, and put it back if the consent or the check afterwards fails, or if you press Ctrl-C:

App	What is backed up	Where
<code>vdirsyncer</code>	token file (moved aside, which also forces a fresh sign-in)	<code>google_token.pre-change</code>
<code>vdirsyncer</code>	OAuth client (rotate-client)	keyring service <code>google-caldav</code> key <code>client_id_prev</code> / <code>client_secret_prev</code>
<code>chroncal</code>	its credential (client + tokens)	<code><credential file>.pre-change</code> , or keyring service <code>calendar-google key chroncal_credential_backup</code>

Each app's timer is stopped while its credentials change, so a background sync without a token can't start its own sign-in, and it is started again at the end. With two apps, a failure in the second one keeps the first on its new token or client and says so. Don't delete the old client in that case.

If the script itself is killed (crash, power loss), backups can be left behind. `status` flags them, and the next `setup`, `reauth` or `rotate-client` resolves them first: if the current state works, it discards the backup; if not, it restores the backup. It also restarts a timer the interrupted run left stopped.

What lives where (per host)

Thing	Where
vdirsyncer's OAuth client	oo7: secret-tool lookup service google-caldav key client_id (and key client_secret)
vdirsyncer token	~/.local/share/vdirsyncer/ google_token (refresh token, 0600)
vdirsyncer state	~/.local/share/vdirsyncer/status/, and the calendars in ~/.local/share/ calendars/google/
chroncal database	~/.local/share/chroncal/chroncal.db (\$CHRONCAL_DB or a db key in its config override it)
chroncal credential, if its keyring works	Secret Service item service chroncal, username db_<namespace>_account_<id>, holding JSON with the client ID + secret and the tokens
chroncal credential, plaintext fallback	~/.config/chroncal/credentials/ db_<namespace>/account_<id>.json (0600)
chroncal config (local only, not stowed)	~/.config/chroncal/config.toml. On daisy it sets [security] allow_plaintext = true
chroncal service	~/.config/systemd/user/chroncal- alarm.{service,timer}, written by chroncal service install (not stowed)

<namespace> is chroncal's per-database credential scope: the database UUID plus a hash of the database file's inode. A copied database gets a new one, and therefore no credential. Copying chroncal.db to another host doesn't carry the account over. Set it up there with calendar-google setup chroncal.

chroncal and oo7: the plaintext fallback

chroncal stores credentials through go-keyring. go-keyring looks for the Secret Service collection /org/freedesktop/secrets/collection/login. oo7 names it Login, so go-keyring falls back to unlocking the alias /org/freedesktop/secrets/aliases/default. oo7 (0.6) answers that Unlock with an empty list, which go-keyring treats as a failure. chroncal then has no keyring, and it refuses to store a new secret with *"no secure credential store is available"*. It refuses before it prompts or opens the browser.

The fallback is opt-in: chroncal writes the credential (client secret + refresh token, in cleartext) to a 0600 file under ~/.config/chroncal/credentials/, outside the repo. There are two ways to opt in:

- **daisy**: the local-only `~/.config/chroncal/config.toml` sets `security.allow_plaintext`, so `chroncal` uses the file without asking.
- **xps / anywhere else**: `calendar-google` setup sees the refusal, explains it, and asks. On `y`, it re-runs `account add` with `--allow-plaintext`. After that, `chroncal` keeps rewriting that file (token refresh, reauth) without the opt-in.

`status` shows which store holds the credential. Keep `~/.config/chroncal` out of anything that syncs or backs up to places you don't trust. `vdirsyncer` is not affected: `libsecret` resolves the alias itself.

Troubleshooting

- **Everything stops after a week**: the consent screen is still in Testing. Publish it, then run `calendar-google reauth`.
- **403 accessNotConfigured**: enable the **CalDAV API** in the client's project.
- **invalid_client**: the client was deleted or its secret changed. Run `calendar-google rotate-client`.
- **invalid_grant**: the token was revoked or expired. Run `calendar-google reauth`.
- **chroncal service points at a missing binary** (after `mise` upgrade prunes the version it was installed from): run `calendar-google setup chroncal`.
- **A new Google calendar**: `vdirsyncer discover` picks it up (the pair has `implicit = "create"`, so it doesn't ask `y/N`). For `chroncal`, run `chroncal account calendars add Google --all`.

Files

Thing	Path
Script	<code>~/.local/bin/calendar-google</code> (<code>scripts/dot-local/bin/</code>)
<code>vdirsyncer</code> config	<code>terminal/dot-config/vdirsynchroner/config</code>
<code>khal</code> config	<code>terminal/dot-config/khal/config</code>
<code>chroncal</code> (<code>mise</code>)	<code>terminal/dot-config/mise/config.toml</code>
Launchers	<code>terminal/dot-local/share/applications/{khal,dev.rdlu.Khal,dev.rdlu.Chroncal}.desktop</code>
Packages	<code>calendar</code> category in <code>setup/packages.yaml</code>
Recipe	<code>just gcal <subcommand> [app]</code>

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 frequency 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
<code>s + 2 chars</code>	Leap forward
<code>S + 2 chars</code>	Leap backward
<code>gs + 2 chars</code>	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, `g zr` 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
<code><leader>p</code>	Open the yank-history picker
<code>y / p / P</code>	Yank / put after / put before (history-aware)
<code>gp / gP</code>	Put after / before the selection
<code>[y /]y</code>	After a paste, cycle to the previous / next ring entry
<code>]p / [p</code>	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
<code><leader>gg</code>	Lazygit (root dir) — <code><leader>gG</code> for cwd
<code><leader>gb</code> / <code><leader>gB</code>	Git blame line / open in browser
<code><leader>.</code> / <code><leader>S</code>	Toggle / select a scratch buffer
<code><leader>n</code> / <code><leader>un</code>	Notification history / dismiss all
<code><c- /></code>	Toggle the terminal
<code><leader>cR</code>	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
<code>:obsidian today / tomorrow / yesterday</code>	Open the daily note
<code>:obsidian new</code>	New note
<code>:obsidian search / quick_switch</code>	Grep / switch notes
<code>:obsidian backlinks / tags</code>	Find references / tag usage
<code>:obsidian template</code>	Insert a template
<code>:obsidian bookmarks</code>	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 " → '	g zr "'
Change () → []	g zr []
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.vim 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.vim):

- <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.vim 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.vim)
- <leader>cp — open the browser markdown preview (markdown-preview.vim)

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 / Ex-pert)	<code><leader>cr</code>
Code actions (add <code>@spec</code> , pipe conversions...)	LSP	<code><leader>ca</code>
Manual extract by structure	mini.ai text objects	<code>vaf / vif</code> (a/inside function), then move it
Project-wide automated refactors	Recode (CLI)	<code>mix recode</code>
Refactoring suggestions + lint	Credo	<code>mix credo --strict</code>

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.

Justfile reference

All automation in this repo is a `mise` task (defined in the repo-root `mise.toml`), with a matching just recipe kept as a thin shim — `just X` and `mise run X` are interchangeable. This page is generated from the live justfile + `mise` tasks by `tools/gen-docs.py` — run `just docs-update` to refresh it.

Run from the repo root. `just` with no arguments lists everything. Tasks are defined in the repo-root `mise.toml` — every recipe is equally runnable as `mise run <task>` (the justfile is a thin shim that delegates to `mise`).

Top-level

Recipe	Description	Runs
<code>just full-auto</code>	Full auto installation	<code>packages,</code> <code>dev-setup,</code> <code>cli-tools,</code> <code>fish-shell,</code> <code>helix-editor,</code> <code>yazi-file-</code> <code>manager,</code> <code>fastfetch</code>
<code>just full-auto-gui</code>	Fuller auto (graphical) in- stallation	<code>full-auto,</code> <code>kitty-</code> <code>terminal,</code> <code>niri-window-</code> <code>manager</code>
<code>just packages</code>	(Run first unless running full-auto) Setup Flatpak, pacman mirrors, Chaotic AUR, and paru AUR helper	
<code>just systemd-niri-config</code>	Enables the systemd ser- vices for some essential niri helpers	<code>systemd-niri-config-</code> <code>install,</code> <code>noctalia-reload</code>
<code>just systemd-niri-</code> <code>config-install</code>		

Install: essentials

Recipe	Description	Runs
<code>just cli-tools</code>	Essential CLI tools from the pacman repos	
<code>just dev-setup</code>	Programming languages, runtimes, toolchains, and git	
<code>just doom-bootstrap</code>	Install Emacs + Doom: clone the framework, stow config, sync packages	
<code>just fish-shell</code>	fish shell and plugins	
<code>just helix-editor</code>		
<code>just yazi-file-manager</code>	Yazi terminal file manager and plugins	

Install: graphical stack

Recipe	Description	Runs
<code>just kitty-terminal</code>		
<code>just niri-window-manager</code>	Install niri and tools used with it	

Install: optional extras

Recipe	Description	Runs
<code>just fastfetch</code>		
<code>just herdr-plugins</code>	ghzinga's viewer binary gzg comes from mise (cargo:ghzinga).	
<code>just keybase-ssh</code>	Import the Keybase PGP key into GPG and expose it for SSH auth	
<code>just pkg-categories</code>	List the manifest categories with their package counts	
<code>just pkg-install</code>	Install this host's missing manifest packages (or one category, or all)	
<code>just syncthing-file-sync</code>		

Stow (symlink management)

Recipe	Description	Runs
<code>just stow</code>	Stow base packages + this host's overlay (hosts/\$host- name/), auto-detected	
<code>just stow-check</code>	Dry-run the stow above (shows what would change, touches nothing)	
<code>just unstow</code>	Remove all symlinks created by stow (host overlay first, then base)	

Services

Recipe	Description	Runs
<code>just services-enable</code>	Enable + start the user ser- vices this setup expects	

niri: reload helpers

Recipe	Description	Runs
just caffeine	Keep the screen awake (Noctalia's caffeine: no idle lock or blanking). Also on the power menu.	
just ext-brightness	External monitor brightness over DDC/CI: up down 0-100 (bare shows current)	
just ext-display	Interactive TUI for the external monitor over DDC/CI (brightness, contrast, input, presets)	
just noctalia-reload		

Maintenance

Recipe	Description	Runs
just doctor	Health check: tools present, shell, host overlay, services, mise, key symlinks	
just pkg-drift	Explicit installs (post day-0) not yet in the manifest or any recipe	
just pkg-missing	Manifest packages this host expects but lacks (setup/hosts/.categories)	
just pkg-tui	Interactive manifest manager: browse categories, install/remove/audit (fzf TUI)	
just update	Bring an existing machine fully up to date (each step best-effort)	

Docs (this site)

Recipe	Description	Runs
<code>just docs</code>	Full docs build: regenerate markdown, then PDFs + cheatsheets, then the HTML site	<code>docs-pdf</code> , <code>docs-cheatsheets</code> , <code>docs-html</code>
<code>just docs-cheatsheets</code>	Landscape cheatsheet PDFs, Catppuccin Latte (print) + Mocha (dark) variants	<code>docs-update</code>
<code>just docs-html</code>	Build the HTML site into <code>site/public/</code> (Hugo + Hextra)	<code>docs-update</code>
<code>just docs-open</code>	Open a PDF (default: the combined handbook; e.g. <code>just docs-open tmux-shortcuts</code>)	<code>docs-pdf</code>
<code>just docs-pdf</code>	Build the PDFs (per-page references + combined handbook) into <code>docs/pdf/</code>	<code>docs-update</code>
<code>just docs-serve</code>	Live-preview the docs site while editing (opens the browser once it's up)	<code>docs-update</code>
<code>just docs-setup</code>	One-time install of the docs toolchain (hugo, pandoc, typst, fonts)	
<code>just docs-update</code>	Regenerate the generated markdown blocks from <code>tmux.conf</code> , <code>binds.kdl</code> , and the justfile	

calendar

Recipe	Description	Runs
<code>just gcal</code>	Google Calendar (khal+vdirsyncer, chroncal): <code>setup</code> <code>status</code> <code>reauth</code> <code>rotate-client</code> [app]	

file-transfer

Recipe	Description	Runs
<code>just file-transfer</code>	Set up LocalSend + rsync-over-ssh receiving (packages, firewall, folder, docs)	
<code>just file-transfer-harden</code>	Harden a receiving machine: inbound SSH key-only + LLNMR off (disables passwords!)	
<code>just gdrive</code>	Google Drive mount: setup status reauth rotate-client rotate-password	

tmux shortcuts

Aliases `mux` / `mux1` open (or re-attach to) the first and second sessions.

Custom bindings

Generated from `terminal/dot-tmux.conf` by `tools/gen-docs.py` — run `just docs-update` to refresh.

The prefix is `C-a` (`Ctrl+a`).

After the prefix

Press the prefix first, then the key.

Key	Description	Command
C-a	Jump to the last active window	last-window
Escape	Enter copy mode	copy-mode
p	Paste buffer	paste-buffer
r	Reload ~/.tmux.conf	source-file \${HOME}/.tmux.conf \; display-message "source- file reloaded"
C-l	Send a literal Ctrl+L (clear screen) to the pane	send-keys 'C-l'
=	Split pane horizontally (keeps current path)	split-window -h -c "#{pane_current_path}"
-	Split pane vertically (keeps current path)	split-window -v -c "#{pane_current_path}"
M	Mouse mode ON	set -g mouse on \; display 'Mouse: ON'
m	Mouse mode OFF	set -g mouse off \; display 'Mouse: OFF'
u	Capture pane and pick a URL to open (urlview popup window)	capture-pane \; save- buffer /tmp/tmux-buffer \; new-window -n "urlview"...
C-t	Popup terminal in the current pane's path	display-popup -E - h 70% -w 70% -d "#{pane_current_path}"

No prefix (root table)

Active anywhere, no prefix needed.

Key	Description	Command
F12	Toggle OFF mode for nested/remote tmux — all keys (incl. the prefix) pass through to the pane; status bar turns red	set prefix None \; set key-table off \; set status-style "fg=colour245,..."
C-t	Unified file finder — pass through in fish/editors, else a popup that injects paths into the pane's TUI (pick-files)	if shell "case '#{pane_current_command}' in fish\ nvim\ vim\ hx) exit 0 ;;...

Copy mode (vi keys)

Active while in copy mode.

Key	Description	Command
v	Begin selection (vi-style)	send-keys -X begin-selection
C-v	Toggle rectangle (block) selection	send-keys -X rectangle-toggle
y	Copy selection and leave copy mode	send-keys -X copy-selection-and-cancel

OFF mode

Active only while OFF mode is toggled on.

Key	Description	Command
F12	Leave OFF mode (restore the local prefix)	set -u prefix \; set -u key-table \; set -u status-style \; set -u wind...

tmux defaults worth remembering

Key (after prefix)	Action
c	New tab (window)
1 2 3...9	Change to tab #1 to #9
!	Promote pane to tab
,	Rename tab
w	List windows
f	Find window
&	Kill window
.	Move window (prompt #)
:movew + Enter	Move window (unused #)
\$	Rename session

Plugin bindings (TPM)

Plugin keymaps are defined by the plugins themselves, not in `tmux.conf`, so these tables are maintained by hand. The plugin list lives in the `@tpm_plugins` block of `tmux.conf`. Manage plugins with `Prefix + I` (install) and `Prefix + U` (update all).

copycat

Key (after prefix)	Action
/	regex search + copy mode
n	jump to the next match in copy mode
N	jump to the previous match
y	copy entire line
Enter	copy highlighted match

yank

- y — copy selection to the system clipboard

Tip with mouse support: press y before releasing the mouse.

extrakto (fuzzy search)

Key (after prefix)	Action
Tab	fuzzy search + fzf mode
Enter	copy selection inside fzf mode

Requires fzf (paru -S fzf).

resurrect + continuum

Key (after prefix)	Action
Ctrl+s	save all tabs
Ctrl+r	restore all tabs

Continuum is configured with @continuum-restore on, so sessions also restore automatically when the tmux server starts.

pain-control (panes)

Splitting:

Key (after prefix)	Action
\	split current pane horizontally
-	split current pane vertically

Navigation:

Key (after prefix)	Action
h C-h	select pane on the left
j C-j	select pane below the current
k C-k	select pane above the current
l C-l	select pane on the right

Resizing:

Key (after prefix)	Action
H	resize left
J	resize down
K	resize up
L	resize right

Swapping windows:

Key (after prefix)	Action
<	move one position to the left
>	move one position to the right

sessionist (sessions)

Key (after prefix)	Action
g	prompt for session name and switch to it
C	prompt for creating a new session by name
X	kill current session without detaching tmux
S	switch to the last session
@	promote current pane into a new session

sessionx (session picker)

Key (after prefix)	Action
o	open the sessionx picker (zoxide mode, ~/Projects)

open

Key (in copy mode)	Action
o	open a highlighted selection with the system default program
Ctrl+o	open a highlighted selection with \$EDITOR
S	web-search the selection (DuckDuckGo)

thumbs

Hint-based copy: trigger with Prefix + Space, then press the hint key to copy that text (configured to use wl-copy via OSC52).

logging

Key (after prefix)	Action
P	start/stop logging
Alt+p	log current screen
Alt+P	save complete history

fzf

Prefix + F opens the tmux-fzf menu (sessions, windows, panes, commands).

herdr shortcuts

herdr is an *agent multiplexer* — a terminal multiplexer built around AI coding agents, with workspaces, tabs, panes, git worktrees, and persistent sessions. Launch it with herdr; it re-attaches to the running session.

For how this maps onto my old tmux setup, see tmux vs herdr.

Keybindings

Generated from terminal/dot-config/herdr/config.toml (prefix + custom & plugin commands) plus herdr's built-in defaults, by tools/gen-docs.py — run just docs-update to refresh.

The prefix is Ctrl+b — press it, release, then the key; every Ctrl+b X below means *prefix then X*.

Tabs

Key	Action
Ctrl+b c	New tab
Ctrl+b ,	Rename tab
Ctrl+b n	Next tab
Ctrl+b p	Previous tab
Ctrl+b 1-9	Jump to tab 1-9
Ctrl+b Shift+x	Close tab

Panes

Key	Action
Ctrl+b h j k l	Focus pane ← ↓ ↑ →
Ctrl+b =	Split side-by-side
Ctrl+b -	Split stacked
Ctrl+b v	Split side-by-side (alias)
Ctrl+b z	Zoom / fullscreen pane
Ctrl+b r	Resize mode (then arrows / hjkl)
Ctrl+b x	Close pane
Ctrl+b Tab	Cycle to next pane
Ctrl+b Shift+Tab	Cycle to previous pane
Ctrl+b Shift+p	Rename pane

Workspaces & sessions

Key	Action
Ctrl+b w	Workspace picker
Ctrl+b Shift+n	New workspace
Ctrl+b .	Rename workspace
Ctrl+b Shift+d	Close workspace (confirm)
Ctrl+b Shift+g	New git-worktree workspace
Ctrl+b g	Goto / jump prompt
Ctrl+b Shift+1-9	Jump to workspace 1-9
Ctrl+b [	Previous workspace
Ctrl+b]	Next workspace
Ctrl+b q	Detach session

Agents

Key	Action
Ctrl+b Alt+1-9	Focus agent 1-9
Ctrl+b {	Previous agent
Ctrl+b }	Next agent

Scrollback & misc

Key	Action
Ctrl+b e	Edit / copy pane scrollback
Ctrl+b b	Toggle sidebar
Ctrl+b Shift+o	Open latest notification's target
Ctrl+b s	Settings
Ctrl+b ?	Help / keybinding overlay
Ctrl+b Shift+r	Reload config

Custom commands

Key	Action
Ctrl+b t	Scratch shell (disposable pane)
Ctrl+b f	Pick files (fd+fzf) → inject paths
Ctrl+b y	Pick files (yazi UI) → inject paths
Ctrl+b u	Grab URLs / paths / IDs from scrollback
Ctrl+b o	Switch / open a project workspace
Ctrl+b Up	herdr-plus: projects
Ctrl+b Down	herdr-plus: quick actions
Ctrl+b Shift+v	reviewr: toggle review sidebar

Direct keys (no prefix)

Key	Action
Alt+h	vim-nav: focus split / pane left
Alt+j	vim-nav: focus split / pane down
Alt+k	vim-nav: focus split / pane up
Alt+l	vim-nav: focus split / pane right

zellij shortcuts

Key bindings for zellij — configured with `clear-defaults=true`, so everything below is the complete map.

Generated from `terminal/dot-config/zellij/config.kdl` by `tools/gen-docs.py` — run `just docs-update` to refresh.

The default mode is **locked** (keys pass through to the shell). `Ctrl a` enters **tmux mode** — a tmux-style prefix — and `Ctrl g` toggles the regular zellij modes. Most actions return to locked automatically.

Tmux mode — Ctrl a

Keys	Action
[	→ scroll mode
Ctrl a	Toggle last tab
z f	Fullscreen pane
c t	New tab
s \$:	Open session-manager
\	New pane right
-	New pane down
p	Previous tab
n	Next tab
1...6	Go to tab 1...6
,	Rename tab
Left h	Focus pane left
Right l	Focus pane right
Down j	Focus pane down
Up k	Focus pane up
.	Rename pane
H	Grow left
L	Grow right
K	Grow up
J	Grow down
e	Embed ⇌ float pane
w	Show/hide floating panes
d	Detach session
Space	Next layout
x	Close pane
/	Search

Always available (normal & locked)

Keys	Action
Alt left Alt h	Focus pane/tab left
Alt down Alt j	Focus pane down
Alt up Alt k	Focus pane up
Alt right Alt l	Focus pane/tab right
Alt + Alt =	Grow
Alt -	Shrink
Alt f	Show/hide floating panes
Alt i	Move tab left

Pane mode — p

Keys	Action
left h	Focus pane left
down j	Focus pane down
up k	Focus pane up
right l	Focus pane right
c	Rename pane
d	New pane down
e	Embed ⇌ float pane
f	Fullscreen pane
n	New pane
p	→ normal mode
r	New pane right
w	Show/hide floating panes
x	Close pane
z	Toggle pane frames
tab	Next pane

Tab mode — t

Keys	Action
left up h k	Previous tab
down right j l	Next tab
1...9	Go to tab 1...9
[	Break pane to tab left
]	Break pane to tab right
b	Break pane into new tab
n	New tab
r	Rename tab
s	Sync input to all panes in tab
t	→ normal mode
x	Close tab
tab	Toggle last tab

Resize mode — r

Keys	Action
left h	Grow left
down j	Grow down
up k	Grow up
right l	Grow right
+ =	Grow
-	Shrink
H	Shrink left
J	Shrink down
K	Shrink up
L	Shrink right
r	→ normal mode

Move mode — m

Keys	Action
left h	Move pane left
down j	Move pane down
up k	Move pane up
right l	Move pane right
m	→ normal mode
n tab	Move pane (next)
p	Move pane backwards

Scroll mode — s

Keys	Action
Alt left Alt h	Focus pane/tab left
Alt down Alt j	Focus pane down
Alt up Alt k	Focus pane up
Alt right Alt l	Focus pane/tab right
e	Edit scrollbar in \$EDITOR
f	Search
s	→ normal mode

Scrolling keys (scroll & search)

Keys	Action
PageDown right Ctrl f l	Page down
PageUp left Ctrl b h	Page up
down j	Line down
up k	Line up
Ctrl c	Jump to bottom
d	Half page down
u	Half page up

Search options

Keys	Action
c	Toggle case-sensitivity
n	Next match
o	Toggle whole-word
p	Previous match
w	Toggle wrap

Session mode — o

Keys	Action
c	Open configuration
d	Detach session
o	→ normal mode
p	Open plugin-manager
w	Open session-manager

Locked mode (default)

Keys	Action
Ctrl g	→ normal mode
Ctrl a	→ tmux mode

Mode switches & misc

Keys	Action
Ctrl g Ctrl a enter esc Ctrl c	→ locked mode
Ctrl q	Quit zellij
m	→ move mode
o	→ session mode
t	→ tab mode
s Ctrl c esc	→ scroll mode
p	→ pane mode
r	→ resize mode
enter	→ search mode
esc	Cancel rename · → tab mode
esc	Cancel rename · → pane mode

niri shortcuts

Key bindings for the niri scrollable-tiling Wayland compositor.

Generated from `niri/dot-config/niri/binds.kdl` by `tools/gen-docs.py` — run `just docs-update` to refresh. For the mouse-warp binds see also the `wl-kbptr` guide.

Mod is the **Super** key. Press `Mod+Shift+/,` to open these docs as a local web app.

Apps & launchers

Keys	Action
<code>Mod+Shift+Slash</code>	Dotfiles Docs (local)
<code>Mod+F1</code>	Keybinding Cheatsheet: Noctalia
<code>Mod+T</code>	Open a Terminal: <code>ghostty</code>
<code>Mod+D</code>	Run an Application: Noctalia
<code>Mod+Slash</code>	Run an Application: <code>fuzzel</code>
<code>Super+Alt+L</code>	Lock the Screen
<code>Mod+Escape</code>	Menu Power (press again: Tools)
<code>Mod+Shift+Escape</code>	Menu Just: dotfiles commands
<code>Mod+E</code>	File Manager: <code>superfile</code>
<code>Mod+Y</code>	File Manager: <code>yazi</code>
<code>Mod+Shift+C</code>	Color Picker: <code>hyprpicker</code>
<code>Mod+Ctrl+N</code>	Notifications: toggle Do Not Disturb
<code>Mod+XF86MonBrightnessUp</code>	External monitor brightness up (<i>works on lock screen</i>)
<code>Mod+XF86MonBrightnessDown</code>	External monitor brightness down (<i>works on lock screen</i>)
<code>Mod+Shift+Ctrl+S</code>	Swap Workspaces Between Monitors
<code>Alt+Tab</code>	Run <code>noctalia msg window-switcher hold</code>
<code>Mod+Period</code>	Emoji Finder
<code>Mod+V</code>	Manage clipboard
<code>Mod+B</code>	Toggle Bar
<code>Print</code>	Screenshot region
<code>Ctrl+Print</code>	Screenshot monitor
<code>Shift+Print</code>	Screenshot and annotate

Mouse warp (wl-kbptr)

Keys	Action
Mod+G	Mouse Warp: wl-kbptr
Mod+Shift+G	Mouse Warp + Click
Mod+Alt+G	Mouse Warp: coarse (tile only)
Mod+Ctrl+G	Mouse Warp: tile + split
Mod+A	Mouse Warp: other monitor
Mod+Shift+A	Mouse Warp + Click: other monitor
Mod+Alt+A	Mouse Warp: coarse, other monitor
Mod+Ctrl+A	Mouse Warp: tile + split, other monitor
Mod+N	Mouse Warp: bisect then tile
Mod+Shift+N	Mouse Warp + Click: bisect then tile
Mod+M	Mouse Warp: bisect then tile, other monitor
Mod+Shift+M	Mouse Warp + Click: bisect then tile, other monitor
Mod+X	Mouse Warp: detect + click (X-ray)
Mod+Shift+X	Mouse Warp: detect only

Media, volume & brightness

Keys	Action
XF86AudioRaiseVolume	Run noctalia msg volume-up (<i>works on lock screen</i>)
XF86AudioLowerVolume	Run noctalia msg volume-down (<i>works on lock screen</i>)
XF86AudioMute	Run noctalia msg volume-mute (<i>works on lock screen</i>)
XF86AudioMicMute	Run noctalia msg mic-mute (<i>works on lock screen</i>)
XF86MonBrightnessUp	Run noctalia msg brightness-up (<i>works on lock screen</i>)
XF86MonBrightnessDown	Run noctalia msg brightness-down (<i>works on lock screen</i>)
XF86AudioNext	Run playerctl next (<i>works on lock screen</i>)
XF86AudioPause	Run playerctl play-pause (<i>works on lock screen</i>)
XF86AudioPlay	Run playerctl play-pause (<i>works on lock screen</i>)
XF86AudioPrev	Run playerctl previous (<i>works on lock screen</i>)

Focus

Keys	Action
Mod+Left	Focus column left
Mod+Down	Focus window down
Mod+Up	Focus window up
Mod+Right	Focus column right
Mod+H	Focus column left
Mod+L	Focus column right
Mod+Home	Focus column first
Mod+End	Focus column last
Mod+WheelScrollRight	Focus column right
Mod+WheelScrollLeft	Focus column left
Mod+Shift+WheelScrollDown	Focus column right
Mod+Shift+WheelScrollUp	Focus column left

Moving windows & columns

Keys	Action
Mod+Ctrl+Left	Move column left
Mod+Ctrl+Down	Move window down
Mod+Ctrl+Up	Move window up
Mod+Ctrl+Right	Move column right
Mod+Ctrl+H	Move column left
Mod+Ctrl+L	Move column right
Mod+Ctrl+Home	Move column to first
Mod+Ctrl+End	Move column to last
Mod+Ctrl+WheelScrollRight	Move column right
Mod+Ctrl+WheelScrollLeft	Move column left
Mod+Ctrl+Shift+WheelScrollDown	Move column right
Mod+Ctrl+Shift+WheelScrollUp	Move column left
Mod+BracketLeft	Consume or expel window left
Mod+BracketRight	Consume or expel window right
Mod+Comma	Consume window into column
Mod+Shift+Comma	Expel window from column

Workspaces

Keys	Action
Mod+J	Focus window or workspace down
Mod+K	Focus window or workspace up
Mod+Ctrl+J	Move window down or to workspace down
Mod+Ctrl+K	Move window up or to workspace up
Mod+Shift+Ctrl+U	Move Workspace to Previous Monitor
Mod+Shift+Ctrl+I	Move Workspace to Next Monitor
Mod+Page_Down	Focus workspace down
Mod+Page_Up	Focus workspace up
Mod+U	Focus workspace down
Mod+I	Focus workspace up
Mod+Ctrl+Page_Down	Move column to workspace down
Mod+Ctrl+Page_Up	Move column to workspace up
Mod+Ctrl+U	Move column to workspace down
Mod+Ctrl+I	Move column to workspace up
Mod+Shift+Page_Down	Move workspace down
Mod+Shift+Page_Up	Move workspace up
Mod+Shift+U	Move workspace down
Mod+Shift+I	Move workspace up
Mod+WheelScrollDown	Focus workspace down (<i>cooldown 150ms</i>)
Mod+WheelScrollUp	Focus workspace up (<i>cooldown 150ms</i>)
Mod+Ctrl+WheelScrollDown	Move column to workspace down (<i>cooldown 150ms</i>)
Mod+Ctrl+WheelScrollUp	Move column to workspace up (<i>cooldown 150ms</i>)
Mod+1	Focus workspace 1
Mod+2	Focus workspace 2
Mod+3	Focus workspace 3
Mod+4	Focus workspace 4
Mod+5	Focus workspace 5
Mod+6	Focus workspace 6
Mod+7	Focus workspace 7
Mod+8	Focus workspace 8
Mod+9	Focus workspace 9
Mod+Ctrl+1	Move column to workspace 1
Mod+Ctrl+0	Move column to workspace 0

Monitors

Keys	Action
Mod+Shift+Left	Focus monitor left
Mod+Shift+Down	Focus monitor down
Mod+Shift+Up	Focus monitor up
Mod+Shift+Right	Focus monitor right
Mod+Shift+H	Focus monitor left
Mod+Shift+J	Focus monitor down
Mod+Shift+K	Focus monitor up
Mod+Shift+L	Focus monitor right
Mod+Shift+Ctrl+Left	Move column to monitor left
Mod+Shift+Ctrl+Down	Move column to monitor down
Mod+Shift+Ctrl+Up	Move column to monitor up
Mod+Shift+Ctrl+Right	Move column to monitor right
Mod+Shift+Ctrl+H	Move column to monitor left
Mod+Shift+Ctrl+J	Move column to monitor down
Mod+Shift+Ctrl+K	Move column to monitor up
Mod+Shift+Ctrl+L	Move column to monitor right

Layout & sizing

Keys	Action
Mod+R	Switch preset column width
Mod+Shift+R	Switch preset window height
Mod+Ctrl+R	Reset window height
Mod+Z	Maximize column
Mod+Shift+Z	Fullscreen window
Mod+Ctrl+Z	Expand column to available width
Mod+C	Center column
Mod+Ctrl+C	Center visible columns
Mod+Minus	Set column width -10%
Mod+Equal	Set column width +10%
Mod+Shift+Minus	Set window height -10%
Mod+Shift+Equal	Set window height +10%
Mod+F	Toggle window floating
Mod+Shift+F	Switch focus between floating and tiling
Mod+W	Toggle column tabbed display

Screenshots & screencasting

Keys	Action
Alt+Print	Screenshot window
Mod+P	Set dynamic cast window
Mod+Ctrl+P	Set dynamic cast monitor
Mod+Alt+P	Clear dynamic cast target

Session & misc

Keys	Action
Mod+O	Toggle overview (<i>no key-repeat</i>)
Mod+Q	Close window
Mod+Alt+Escape	Toggle keyboard shortcuts inhibit (<i>never inhibited</i>)
Mod+Shift+E	Quit
Ctrl+Alt+Delete	Quit
Mod+Shift+P	Power off monitors

Shell shortcuts (fish)

Interactive-shell bindings: custom bind lines from this repo, plus the bindings injected by atuin and fish itself.

Custom bindings

Generated from the bind lines in fish/ by tools/gen-docs.py — run just docs-update to refresh.

Keys	Action	Defined in
Alt+y	Open the yazi file manager	fish/ aliases.d/files.fish
Ctrl+t	Insert file path(s) (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+Alt+d	Insert a directory path (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+Alt+g	Insert pacman package(s) (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+Alt+l	Insert a commit SHA (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+Alt+s	Insert changed file path(s) (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+Alt+p	Insert a PID (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+v	Insert a shell variable name (fzf)	fish/aliases.d/fzf- pickers.fish
Ctrl+h	Delete the word left of the cursor	fish/ aliases.d/keybinds.fish
Ctrl+e	Neovim config picker (de- fault / nvim-lazy / nvim-full)	fish/conf.d/nvim- switcher.fish

thefuck — fix the last command

Bound inside fish_user_key_bindings (fish/conf.d/thefuck.fish), so they only appear in interactive shells.

Keys	Action
Esc Esc or Ctrl+f	Correct the previous command (thefuck)
fuck (typed)	Same, as a function

History — atuin

atuin init fish --disable-up-arrow (in config.fish), so the up arrow keeps plain fish history.

Keys	Action
Ctrl+r	Fuzzy-search shell history (atuin)
?	Ask atuin AI (on an empty prompt)

fish built-ins worth remembering

Keys	Action
Alt+e Alt+v	Edit the command line in \$EDITOR
Alt+s	Prepend sudo (or doas) to the current (or last) command
Alt+b	Previous directory (prevd) on an empty line, else back one word
Alt+↑ Alt+↓	Search history for the token under the cursor
Alt+l	List the directory of the token under the cursor
Alt+p	Append & less (paginate output)
Alt+h F1	Man page for the command under the cursor
Alt+w	One-line description of the command under the cursor
Ctrl+w	Delete one path component to the left
Ctrl+z	Undo last edit

Inside herdr: Alt+h/j/k/l are intercepted by vim-herdr-navigation for pane/split movement, so fish's Alt+h (man page) and Alt+l (list dir) only fire in a bare terminal — use F1 for the man page inside herdr. (Alt+y is rebound to the yazi launcher — see Custom bindings above — which shadows fish's yank-pop.)

Handy abbreviations & functions

The abbreviation-tips plugin reminds you of these when you type the long form.

Type	Expands to / does
mux / muxl	open or re-attach tmux session mux0 / mux1
nvims	fzf picker for neovim configs (default / nvim-lazy / nvim-full)
nv / lvi	neovim with the light / lazy config
z / zi	zoxide: jump to dir / interactive picker
b	back to the previous directory and list it
mcd	mkdir -p + cd into it
cpv	copy with progress bar (rsync)

wl-kbptr

Keyboard-driven mouse pointer for Wayland. Replaces mouseless, which is X11-shaped and breaks on niri.

Upstream: <https://github.com/moverest/wl-kbptr>

Install

Bundled with the niri install target:

```
just niri-window-manager
```

Or standalone:

```
yay -S wl-kbptr jq
```

jq is required by the wl-kbptr-other-output helper (see Multi-monitor notes). Config lives at ~/.config/wl-kbptr/config, stowed from niri/dot-config/wl-kbptr/config in this repo.

Binds

	Current monitor	Other monitor
default (tile,bisect)	Mod+G	Mod+A
+ click	Mod+Shift+G	Mod+Shift+A
coarse (tile only)	Mod+Alt+G	Mod+Alt+A
tile + split	Mod+Ctrl+G	Mod+Ctrl+A
bisect → tile	Mod+N	Mod+M
bisect → tile + click	Mod+Shift+N	Mod+Shift+M
detect + click (X-ray)	Mod+X	—
detect only	Mod+Shift+X	—

The Mod+N/Mod+M families coarse-aim first, then refine on a grid: press the chord, bisect into the rough quadrant with a/s/d/f/z/x/c/v (positional left hand), recurse as deep as you want, then press **Enter** to hand the current area off to tile. Tile then shows labeled cells over that area — type the label, cursor warps. Different mental model from Mod+G (“scan grid, type label”): “point first, label second”.

Other-monitor binds run ~/.local/bin/wl-kbptr-other-output (stowed from niri/dot-local/bin/wl-kbptr-other-output). The helper picks the non-focused output, calls niri msg action focus-monitor to flip keyboard focus there, then runs wl-kbptr -o <name>. Extra args are forwarded so each variant passes its own -o modes=...

Mid-stream click keys

In any mode, pressing j/k/l commits the current cursor position and clicks. No need to chain click mode for occasional clicks.

Key	Click
j	left
k	right
l	middle

Modes

Mode	What it does
tile	Grid of labeled cells. Type the label → cursor jumps to cell center.
bisect	Splits area into 2x4 (or 2x2 if square). One keypress per cell to refine.
split	Crosshair-style refinement; arrow keys (or wasd/h) shrink the area toward that side.
click	Commits position with a click. Default left; override with <code>-o mode_click.button=right</code> .
floating	Reads WxH+X+Y rectangles from stdin. Skip unless you script your targets.

Chains are left-to-right: each mode hands the chosen area to the next.

Cell-pick keys (home_row_keys)

home_row_keys is exactly 11 characters: first 8 = bisect cells, last 3 = click buttons (left/right/middle).

Our config is asdfzxcvjl:

keyboard	screen cells
a s d f	0 1 2 3 (top)
z x c v	4 5 6 7 (bottom)

j = left click
k = right click
l = middle click

Left hand drives bisect, right hand drives clicks. Symmetric, all stays on the home row.

Universal keys

Key	Action
Enter	Commit current position; advance to next mode in chain
Backspace	Undo last keypress (works across mode boundaries)
Escape	Cancel — no warp, no click

Useful CLI flags

Flag	Use case
-O <NAME>	Target a specific output (DP-2, eDP-1, ...).
-r WxH+X+Y	Restrict overlay to a region (e.g. inside a single window).
-o KEY=VAL	Override config inline (-o modes=tile,bisect,click).
-p	Print result, don't warp or click. Test mode.
-c FILE	Load alternate config file.

Multi-monitor notes

- Default behavior (no -O): wl-kbptr passes NULL to layer-shell; niri assigns the overlay to the focused output. So Mod+G always uses the monitor your keyboard focus is on.
- Mod+M family targets the *other* monitor via the helper script. niri only routes keyboard input to a layer-surface on the focused output, so the helper calls `niri msg action focus-monitor <name>` before launching wl-kbptr. Without that, the overlay appears but typing keeps going to the previously-focused window.
- 3+ monitors: the helper picks an arbitrary non-focused one. For deterministic ordering, edit `wl-kbptr-other-output`.

Gotchas

- **~/.local/bin is not in niri's PATH.** Bind helper scripts with the full `~/.local/bin/<name>` path. Bare command names silently no-op.
- **Split mode + our home_row_keys.** j/k/l are click buttons, so they don't navigate in split. Use arrow keys, or w/a/s/h for up/left/down/left fallback. See issue #42.
- **Maintainer comment in #42 misstates click order.** The thread says i, j, k = right/left/middle, but the v0.4.1 source has indices 8/9/10 as LEFT/RIGHT/MIDDLE. Trust the source.

Files

<code>docs/wl-kbptr.md</code>	<code># this file</code>
<code>niri/dot-config/wl-kbptr/config</code>	<code># home_row_keys + default mode chain</code>
<code>niri/dot-config/niri/binds.kdl</code>	<code># Mod+G and Mod+M bind families</code>
<code>niri/dot-local/bin/wl-kbptr-other-output</code>	<code># helper for the Mod+M family</code>