# rbterm User Manual

**Version 0.3.28**

A cross-platform terminal emulator written in C, rendered with raylib. Designed for macOS, Linux, and Windows. Single binary. No runtime dependencies on the user side once the .app/.dmg or release archive is installed.

---

## Table of Contents


1. Introduction
2. Installation
3. Quick start
4. Tabs and panes
5. Keyboard reference
6. Mouse
7. Settings
8. SSH
9. SFTP transfers
10. Recording
11. HUD overlay
12. Search in scrollback
13. Broadcast input
14. Graphics protocols
15. Quake hotkey (macOS)
16. Recursive split layouts
17. Building from source

18. Auto-update
19. Logging
20. Limitations
21. License

---

## Introduction

rbterm is a terminal emulator focused on a thin, predictable code path between input and output. raylib supplies the windowing, OpenGL context, and font rasterisation; a small `pty.h` abstraction sits between the rest of the code and three PTY backends (forkpty on Unix, ConPTY on Windows, libssh for remote shells). Everything else — the ANSI/VT parser, the grid buffer, scrollback, reflow, the recursive split tree, the recording engine, the settings modal — lives in plain C with no framework lock-in.

### What it gives you out of the box

- 256-colour + truecolour, SGR, OSC 4/8/10/11/52/104/133, bracketed paste, focus reporting (DECSET 1004), mouse modes (1000/1002/1006).
- Recursive split tree per tab — split any pane into V or H children, recursively, drag splitters, cycle focus.
- Native sixel + kitty graphics — `img2sixel`, `chafa -f sixel`, `gnuplot`, `ranger`, `kitten icat` all work without configuration.
- Native colour emoji on macOS via CoreText; stub fallback elsewhere.
- Bundled fonts (FiraCode, plus several fallbacks) — no missing-font failures on a fresh install.
- Per-pane recording → asciinema cast, GIF, WebP, MP4, WebM, APNG. No external ffmpeg dependency for GIF + WebP; MP4/WebM/APNG still require ffmpeg on PATH (Windows release archive bundles it).
- SSH tab ( `Cmd+Shift+T` ) with libssh — key auth, trust-on-first-use host keys, per-host saved profiles, init cwd / init cmd, optional per-host recursive split layout that replays on connect.
- SFTP upload and download buttons on every SSH tab — files and folders, riding the same session (no second auth).
- Per-pane HUD overlay with hostname, IP, load, memory, disk, CPU sparkline. Works for both local and SSH shells.
- Search in scrollback ( `Cmd+F` ) with substring match + selection highlight.
- Broadcast input ( `Cmd+Shift+I` ) — every keystroke fans out to every leaf of the active tab; useful for running the same command on multiple servers at once.
- Quake-style summon/dismiss hotkey on macOS ( `Cmd+CapsLock` by default when launched in borderless mode).
- macOS auto-update channel — the in-app updater checks `rbterm.ximg.app/updates/manifest.json` and pulls signed, notarised, stapled .app bundles in place.

### What it deliberately doesn't do

- No plugins, no Lua/Python config, no remote-control socket. Behaviour lives in the source tree; preferences live in the settings modal and `~/.config/rbterm/config.ini`.
- No ligatures (HarfBuzz shaping is on the wishlist; the current glyph cache is fixed-width).
- No DirectWrite emoji on Windows (stub renderer; ASCII fallback for symbols).

---

## Installation

### macOS

The canonical install path is the signed, notarised `.app` bundle published to https://rbterm.ximg.app/downloads/.

**Drag-install (.dmg):**

1. Download `rbterm-macos-arm64.dmg`.
2. Open it, drag rbterm.app into the Applications folder shortcut on the mounted volume.
3. Eject and launch from Applications.

**Auto-update** — once installed, rbterm checks the manifest at startup and prompts to install newer signed builds in place. No re-download or re-drag required.

**Homebrew tap:**

```
brew install --cask binrick/tap/rbterm
brew upgrade --cask rbterm
```

The cask follows the same manifest, so `brew upgrade` and the in-app updater stay in lockstep.

### Linux

Download `rbterm-linux-x86_64.zip` from the GitHub release, unzip, and run.

Runtime deps (Ubuntu/Debian package names):

```
sudo apt-get install -y unzip libssh-4 libharfbuzz0b libwebp7 \
    libwebpmux3 libfontconfig1 libxinerama1 libxcursor1 \
    libxi6 libgl1 libxrandr2 libxkbcommon0
```

For GUI use you need a desktop session (Wayland or X11). Headless boxes can install but won't open a window.

### Windows

Download `rbterm-windows-x86_64.zip` from the GitHub release. Unzip and double-click `rbterm.exe`. Windows 10 1809 or newer is required for ConPTY support.

Defender may quarantine on first launch — right-click → Properties → Unblock, or add a Defender exclusion if you trust the binary.

---

## Quick start

Launch rbterm. The first tab opens with your login shell in your home directory (or the directory you launched from, if you set `startup_window=default` and ran from a terminal).

Common gestures:

| Action | macOS | Linux/Windows |
|---|---|---|
| New tab | `Cmd+T` | `Ctrl+Shift+T` |
| Close tab | `Cmd+W` | `Ctrl+Shift+W` |
| Next tab | `Cmd+Shift+]` | `Ctrl+Tab` |
| Prev tab | `Cmd+Shift+[` | `Ctrl+Shift+Tab` |
| Tab 1..9 | `Cmd+1` … `Cmd+9` | `Ctrl+1` … `Ctrl+9` |
| Split vertical | `Cmd+D` | `Ctrl+Shift+D` |
| Split horizontal | `Cmd+Shift+D` | `Ctrl+Shift+H` |
| Close pane | `Cmd+Shift+W` | `Ctrl+Shift+P` |
| Cycle pane | `Cmd+]` | `Ctrl+]` |
| Settings | `Cmd+,` | `Ctrl+,` |
| Search scrollback | `Cmd+F` | `Ctrl+F` |
| Copy | `Cmd+C` | `Ctrl+Shift+C` |
| Paste | `Cmd+V` | `Ctrl+Shift+V` |
| SSH connect | `Cmd+Shift+T` | `Ctrl+Shift+S` |
| Broadcast input | `Cmd+Shift+I` | `Ctrl+Shift+I` |
| Font size + / – | `Cmd+=` / `Cmd+-` | `Ctrl+=` / `Ctrl+-` |
| Reset font size | `Cmd+0` | `Ctrl+0` |

Throughout this manual the macOS modifier is `Cmd`; on Linux/Windows substitute `Ctrl`. Ctrl is reserved for C0 chords like `Ctrl+C` (SIGINT) on all platforms — rbterm distinguishes Cmd vs Ctrl on macOS so `Cmd+C` copies and `Ctrl+C` interrupts.

---

## Tabs and panes

rbterm's split model is a **recursive tree**. Each tab owns a `PaneNode *root`. A leaf node carries a single `Pane` (PTY + Screen + selection + title + cwd). An internal node carries a `SplitMode` (vertical or horizontal), a `ratio`, and two children — each of which can itself be a leaf or another internal node.

This means there's no "pane 0 / pane 1" model. Any pane can be split into two children, which can each be split again, to arbitrary depth.

### Splitting

- `Cmd+D` — replace the active leaf with a **vertical** split (two side-by-side panes).
- `Cmd+Shift+D` — **horizontal** split (top and bottom).

The new pane inherits the focused leaf's working directory.

### Resizing

Drag the splitter bar between any two panes. Each internal node tracks its own drag state, so deeply nested layouts resize correctly. The minimum ratio is 0.15 and the maximum is 0.85 — panes don't collapse to zero width.

### Focus

- `Cmd+]` cycles forward through the leaves in DFS order.
- `Cmd+[` cycles backward.
- Click any pane to focus it directly.

### Closing

`Cmd+Shift+W` closes the active leaf. Its parent internal node collapses: the sibling takes over the parent's slot. If the leaf is the only one in the tab, the whole tab closes.

---

# Keyboard reference

## Navigation

| Chord | What |
|---|---|
| `Cmd+T` | New tab |
| `Cmd+W` | Close tab |
| `Cmd+Shift+T` | SSH connect modal |
| `Cmd+N` | New window (separate process) |
| `Cmd+1` … `Cmd+9` | Jump to tab N |
| `Cmd+Shift+[` / `]` | Prev / next tab |
| `Cmd+D` | Split vertical |
| `Cmd+Shift+D` | Split horizontal |
| `Cmd+]` / `[` | Cycle pane forward / back |
| `Cmd+Shift+W` | Close pane |

## Editing & clipboard

| Chord | What |
|---|---|
| `Cmd+C` | Copy selection |
| `Cmd+V` | Paste (bracketed when the app requests it) |
| `Cmd+A` | Select all visible cells |
| `Cmd+F` | Search scrollback |
| `Esc` (in search) | Close search, restore scroll position |
| `Enter` / `F3` / `Down` | Next match |
| `Shift+Enter` / `Shift+F3` / `Up` | Previous match |

### Font and view

| Chord | What |
|---|---|
| `Cmd+=` | Bigger font |
| `Cmd+-` | Smaller font |
| `Cmd+0` | Reset font size |
| Wheel up/down | Scroll scrollback |
| `Shift+Wheel` | Horizontal scroll (where supported) |
| `Cmd+,` | Open settings |

### Modes

| Chord | What |
|---|---|
| `Cmd+Shift+I` | Toggle broadcast input |
| `Cmd+CapsLock` (macOS borderless mode) | Quake summon/dismiss |
| `Cmd+Q` | Quit |

### C0 chords (sent to the shell)

`Ctrl+letter` sends `letter - 0x60` (so `Ctrl+A` = 0x01, `Ctrl+C` = 0x03, `Ctrl+Z` = 0x1A, etc.). `Ctrl+[`, `Ctrl+\`, `Ctrl+]`, and `Ctrl+Space` send 0x1B, 0x1C, 0x1D, 0x00 respectively. Arrow keys, function keys, Home/End/PgUp/PgDn, and Insert/Delete all generate the usual xterm-compatible CSI sequences.

---

## Mouse

- **Single click**: position the cursor (in apps that read mouse reports) or start a selection.
- **Double click**: select word. rbterm uses an intelligent word-boundary algorithm — runs of non-whitespace are picked up, then trailing punctuation ( `,;:.!?` ) is stripped unless you clicked directly on it. Unmatched brackets/quotes are also trimmed. Hyphens and dots interior to a token are kept (so `--flag`, `foo.bar`, `user@host`, and `/usr/local/bin` select cleanly).
- **Triple click**: select line.
- **Click + drag**: extend selection.
- **Shift+click**: extend the current selection to the click point.
- **Middle click** (Linux/Windows): paste primary selection.
- **Right click**: contextual paste (configurable).

- **Wheel up/down**: scroll scrollback (the alt screen does not have scrollback — programs like vim/tmux see arrow keys instead, but only when the foreground app has opted into mouse mode).

The selection is copied to the system clipboard immediately on mouse-up (configurable in settings). Selection over wide characters (CJK, emoji) handles continuation cells correctly.

---

## Settings

`Cmd+,` opens the settings modal. The layout is computed once per frame so click handling and drawing share the same rectangles.

Preferences live in `~/.config/rbterm/config.ini`. Edits made in the modal are process-local until you click **Save as Default**, after which they survive a restart.

### General

- **Theme** — built-in palettes (Solarized, Dracula, Gruvbox, Tango, Tomorrow, Nord, Monokai, Catppuccin, ...) or current SGR colours.
- **Cursor style** — block / bar / underline / blinking variants.
- **Cursor colour** — fixed hex code or "use cell foreground" (the default).
- **Font size** — base point size (per-tab adjustments via `Cmd+=` / `Cmd+-` overlay on top).
- **Scrollback lines** — default is large enough that you almost never hit the cap during normal use.

### Cursor

Dedicated cursor tab with a colour picker (eight presets + "default") and a live preview. Per-host overrides live in the SSH form.

### Logging

- **Enabled** toggle and **Log directory** field.
- Writes are **raw PTY bytes** — ANSI escapes and SGR included. Strip with `sed 's/\x1b\[[0-9;]*[a-zA-Z]//g'` for plain text.
- `fflush` after every write.
- Toggling the switch or changing the directory immediately opens or closes log files on every open tab.

### HUD

- **Master toggle** ( `show_hud` ) and **position** (TL / TR / BL / BR).

- Per-field grid (Host / IP / Load / Memory / Disk): visibility toggle, colour swatch, size –/+ (10..18 pt).
- **CPU graph** toggle ( `hud_show_cpu` ).

### Launch

A list of up to 16 tabs to spawn at startup. Each entry is either `local` (current shell, no extra config) or `ssh:<alias>` (uses the matching profile from `~/.ssh/config` ). Rows can be reordered with the ▲ / ▼ buttons and deleted with the × button. SSH connect failures log to stderr and skip — a single typo can't lock you out.

### Window startup modes

Choose how the window appears when rbterm launches:

| Mode | Behaviour |
|---|---|
| **Default** | raylib's choice (whatever the OS hands you) |
| **Small / Medium / Large** | Fixed size, centred on the primary monitor |
| **Fill** | `MaximizeWindow()` — fills the work area, keeps the OS chrome |
| **Borderless** | Maximised + `FLAG_WINDOW_UNDECORATED` — covers the screen but is **not** native fullscreen, so `Cmd+Tab` and the Quake hotkey work |
| **Maximized (Own Space)** | macOS native fullscreen via `toggleFullScreen:` — its own Space, no menu bar |

Borderless is the setting you want for the Quake hotkey to work reliably; native fullscreen suspends the app deeply enough that the hotkey re-summon stutters.

### Keys (SSH key manager)

Built-in SSH key manager — never opens a child `ssh-keygen` or `ssh-copy-id` process.

- **List** — every `*.pub` in `~/.ssh` , sorted by modification time descending. Each row shows the algorithm, fingerprint (SHA256), and absolute paths.
- **Generate** — ed25519 (no parameters) or RSA-4096. Sub-modal collects the basename; produces `~/.ssh/id_<name>` and `~/.ssh/id_<name>.pub` with proper modes (0600 / 0644).
- **Install** — pick a saved SSH profile, click Install. Opens a one-shot libssh session, appends the key to the remote `~/.ssh/authorized_keys` idempotently, reports success or the libssh error.
- **Delete** — × on the row → confirm modal → unlink both files.

---

# SSH

## The connect modal — `Cmd+Shift+T`

The modal collects:

- **Host** — alias from `~/.ssh/config`, or `user@host[:port]` form for one-off connections.
- **User** / **Port** — auto-populated from `ssh_config`, overridable.
- **Key file** — manual path or pick from the `▾` dropdown (sourced from `ssh_keys_rescan` — same list as the key manager).
- **Password** — used for encrypted private-key passphrases. Empty = load the key without prompting; non-empty is passed to libssh via a passphrase callback (the OpenSSL tty fallback is suppressed so no invisible prompt sits on the launching terminal).
- **Cursor colour** — per-host override.
- **Init cwd** / **Init cmd** — first line sent on connect.
- **Save layout from active tab** — only when the active tab has a recursive split layout; serialises it into `~/.ssh/config`.

Auth order:

1. SSH agent
2. `~/.ssh/id_*` matching the profile
3. Per-host `IdentityFile`
4. The Key file you picked (with passphrase if provided)

Host-key handling is trust-on-first-use into `~/.ssh/known_hosts`. A changed key aborts with the libssh error rendered in red in the modal.

## Saved profiles

Saved profiles live in `~/.ssh/config` as standard `Host` blocks, plus rbterm-specific comment fields:

```
Host mia
    HostName 172.238.205.61
    User rich
    Port 22
    IdentityFile ~/.ssh/id_ed25519
    # rbterm-init-cwd: /var/log
    # rbterm-init-cmd: htop
    # rbterm-cursor-color: #88c0d0
    # rbterm-layout: V0.50(H0.50(0,1),2)
    # rbterm-pane-0-cwd: /var/log
    # rbterm-pane-2-cmd: htop
```

The `# rbterm-*` lines are written by rbterm and ignored by openssh, so the file stays a valid `ssh_config` and is portable.

### Per-host layouts

The layout grammar is:

```
expr := DIGIT
      | ('V' | 'H') ratio '(' expr ',' expr ')'
ratio := '0.' DD       (clamped to 0.15..0.85)
```

Integers index into the per-host `pane_cwds[]` and `pane_cmds[]` arrays. The serializer walks the tab's tree in DFS pre-order so the parser can rebuild it without any extra numbering. Up to 8 leaves per layout.

On connect, after the initial auth + first pane open, rbterm replays the layout: at each integer leaf in DFS order it sends `cd "<cwd>"; <cmd>\r` if either is set. The result is a multi-pane session that looks the same every time you connect.

---

## SFTP transfers

Every SSH tab has ↑ and ↓ buttons in the tab bar.

### Upload

Click ↑. macOS opens a native file picker. Select one or more files (or folders). For each, rbterm:

1. Takes the existing SSH session's lock cooperatively.
2. `sftp_new` + `sftp_init` on the same channel — no second auth.
3. `sftp_open(O_WRONLY | O_CREAT | O_TRUNC, 0644)`.
4. Writes 32 KB chunks.
5. Closes the SFTP file, releases the lock.

If the remote path is a directory, the local basename is appended automatically. `~/foo` strips the `~/` because SFTP's cwd is your remote home and tilde expansion happens on the shell, not the protocol.

A toast appears bottom-left with percentage and bytes transferred. Multiple toasts stack.

### Download

Click ↓. A modal opens showing the remote home directory listing (directories first, alphabetical). Navigate with double-click. To download:

- **A file** — single-click + Download → native save dialog.
- **A folder** — single-click + Download → native "pick parent directory" dialog. Rbterm walks the tree recursively, calling `sftp_stat` before each file, creating local directories at 0755.

Both directions ride the same libssh session; nothing in `~/.ssh/config` needs to mention SFTP.

Set `RBTERM_DEBUG=1` (the `run.sh` script does this) to log every transfer step to `~/rbterm-upload.log` with libssh and SFTP error codes.

---

## Recording

Every pane has a record button in the bar (red dot when armed).

### Capture

Pressing the button arms recording globally — there is one recording per process. The first frame is a synthetic snapshot at t=0 (clear + per-row cursor + cell codepoints) so playback opens with what you already see, not a blank screen. After that, every byte going to `screen_feed` is mirrored to the cast buffer as asciinema v2 events.

### Save formats

Stop recording → save modal with seven format pills:

| Format | How it's produced |
|---|---|
| **cast** | Plain rename — asciinema v2 JSON |
| **txt** | `strip_feed` walks the cast, follows CR/BS/LF cursor moves, drops every escape, writes the visible text |
| **gif** | Native `gif_encoder.c` (LZW, 6×6×6 colour cube + 40-step grey ramp) |
| **webp** | Native `webp_encoder.c` (libwebp + libwebpmux animations, q=75) |
| **mp4** | `ffmpeg -pix_fmt yuv420p -movflags +faststart` from rawvideo on stdin |
| **webm** | `ffmpeg -c:v libvpx -b:v 1M` |
| **apng** | Two-pass: temp gif → `ffmpeg -i tmp.gif -plays 1` |

Render proceeds in 6-frame chunks with a fresh modal frame between chunks — without this the OS shows a beachball on long saves.

GIF and WebP are produced by native encoders, so they work even if ffmpeg is not installed. MP4, WebM, and APNG still need ffmpeg on PATH (the Windows release archive bundles a static build; macOS and Linux currently rely on the system one).

---

## HUD overlay

Each pane carries a `hud_*` snapshot — hostname, primary IPv4, load average, free memory, root disk percentage, and CPU ticks. The HUD draws on top of the pane content with a translucent black background and a thin border. The slab auto-sizes to the longest visible line.

### Data source

- **Local panes**: 1 Hz `hud_local_poll` in the main process — `gethostname` (stripped of the trailing `.local`), `getifaddrs`, `getloadavg`, `host_statistics64` (mac) or `/proc/meminfo` (linux), `statfs` / `statvfs`.
- **SSH panes**: a per-session probe thread inside `shelld` opens a fresh exec channel every 2 s and runs a portable POSIX snippet:

```
sh printf 'HOST=%s\n' "$(hostname -s)" printf 'IP=%s\n' "$(hostname -I 2>/dev/null | awk '{print $1}')" … printf 'END=1\n'
```

The probe takes the SSH session lock with `pthread_mutex_trylock` — if a heavy shell holds it, the probe skips the cycle rather than stalling. The snippet has macOS and Linux fallbacks for every field.

### CPU sparkline

A 60-slot ring stored per pane. Each tick computes `busy_delta / total_delta` from cumulative tick counters (`host_statistics(HOST_CPU_LOAD_INFO)` on macOS, `/proc/stat` on Linux). Bars draw oldest → newest, height proportional to percentage, colour green → yellow → red. The most-recent value is shown numerically as a text tag.

### Customisation

All HUD options live in Settings → HUD. Per-field colour swatches use an 8-entry palette (`HUD_PALETTE`) with stable indices across releases.

---

## Search in scrollback

`Cmd+F` opens a per-pane search bar above the cells. It searches the rendered scrollback (current screen plus history) with substring + ASCII case-folding semantics.

- **Enter / F3 / Down** — next match.
- **Shift+Enter / Shift+F3 / Up** — previous match.
- **Esc** — close, restore the scroll position you had before opening the bar.
- **Cmd/Ctrl+V** — paste into the query field.

Matches highlight in translucent yellow; the current match is brighter. `search_scroll_to_match` centres each match in the viewport.

Known limits in v1:

- ASCII fold only — case-insensitive over ASCII, exact match otherwise.
- One row at a time — auto-wrapped lines aren't joined yet, so hits split across two visual rows are missed.
- Inside tmux / vim / less the alt screen is in play and there is no scrollback. Use `<prefix>[` in tmux or `:!` in vim.

---

## Broadcast input

`Cmd+Shift+I` (or the radio-tower button in the tab bar) toggles broadcast mode for the **active tab**. Every keystroke and every paste fans out to every leaf of that tab's split tree.

Visual cues are deliberately redundant — once is too easy to miss:

- The toggle button glows red.
- A `BROADCAST` pill appears in the right tab-bar cluster.
- Every leaf gets a 3-pixel red border.

The button is hidden when the tab has fewer than 2 panes. Broadcast auto-disarms when you switch tabs, so you can't accidentally type into panes you forgot were broadcast targets.

Note: **mouse clicks and selection drags are not broadcast** — only keystrokes and pastes. Output stays per-pane.

---

## Graphics protocols

rbterm advertises sixel support in the Primary-DA response (`CSI ? 65;1;4;9 c`). Common tools auto-select the right output:

```
img2sixel hello.png        # works
chafa -f sixel hello.png   # works
gnuplot                    # set terminal sixel
ranger                     # image previews
```

Kitty's APC graphics protocol (`f=100` PNG over a single APC message) also works — `kitten icat` and any client that speaks v1.

Implementation:

- Parse: DCS / APC payloads buffered in `screen.c`. Terminator dispatches by leading byte — DCS digits + `q` = sixel, APC `G` = kitty.
- Decode: heap RGBA8. Sixel uses a two-pass extents-then-raster decoder; kitty v1 uses raylib's `LoadImageFromMemory`.
- Store: each image attaches to an anchor row + column on the current screen. Scrolls decrement the anchor; rows < 0 evict. Cap is 32 images per screen, oldest evicted first.
- Blit: per-frame texture cache keyed by `(ScreenImage*, generation)`. The cursor advances past the image so text doesn't overlap.

v1 gaps: sixel doesn't yet honour `P1=0`, `P3`, or custom colour registers; kitty doesn't yet support `f=24` / `f=32`, `m=1` chunked transfer, animation, delete commands, or the reply protocol; no selection over images; no reflow on resize.

---

## Quake hotkey (macOS)

When the startup window mode is **Borderless**, rbterm registers a system-wide hotkey (`Cmd+CapsLock` by default) that summons and dismisses the window. This is not a poll loop — it's wired through multiple OS-level layers to stay responsive:

1. **Carbon** `RegisterEventHotKey` is the reliable backstop. NSEvent monitors need Input Monitoring permission and silently fail without it; Carbon doesn't.
2. **App Nap throttling is suppressed** with `[NSProcessInfo beginActivityWithOptions:UserInitiated|LatencyCritical]` and an `IOPMAssertion` of type `kIOPMAssertionTypePreventUserIdleSystemSleep`. Both are required to keep events from buffering for 1–2 seconds after focus loss.
3. **No** `[NSApp hide:]` — that deep-suspends the run loop. Instead each window is `orderOut:`'d individually and the previously active app is `activateWithOptions:`'d back (tracked via `NSWorkspaceDidActivateApplicationNotification`).
4. **The toggle source is** `[NSApp isActive]`, not a flag rbterm maintains itself — `Cmd+Tab` away can change the state without rbterm seeing it.
5. **Caps Lock state is reset** via `IOHIDSetModifierLockState` so the keyboard doesn't drift to ALL CAPS after a press.

The toggle runs inline from the NSEvent or Carbon block — not via a polled flag — so the response is immediate even when rbterm's main loop is paused waiting for events.

---

## Recursive split layouts

The split tree is the foundation for both interactive splits (`Cmd+D`) and per-host SSH replay. The iteration pattern across the codebase is:

```
for (PaneNode *leaf = pane_tree_first_leaf(t->root); leaf;
     leaf = pane_tree_next_leaf(leaf)) {
    Pane *p = leaf->pane;
    if (!p) continue;
    /* ... */
}
```

Avoid hardcoding "pane 0 / 1" or `t->active_pane` — those don't exist. The active leaf is `t->active`, a pointer into the tree.

When you split, `tab_split` replaces the active leaf with a new internal node whose two children are (a) the old leaf and (b) a fresh leaf. When you close, `tab_close_leaf` collapses the parent node into the sibling.

Splitter drag math uses the press point's *outer rect* (not the current internal node's rect), stashed in a function-local static, so the resize is consistent across frames even as the tree mutates.

---

## Building from source

### macOS

```
brew install raylib libssh libwebp glib harfbuzz pcre2 freetype
make                                # → ./rbterm   (fast dev path)
make app                            # → ./rbterm.app (signed locally)
make app-distributable              # → signed + notarised + stapled .app
make app-dmg                        # → ./rbterm.dmg (drag-to-/Applications)
```

The fast `make` path links against `brew raylib` (embedded GLFW; ~3–9% idle CPU). The `make app` / `make app-distributable` paths use CMake with `USE_EXTERNAL_GLFW=ON`, which links against `brew libglfw` and exposes `glfwWaitEventsTimeout` — this drops idle CPU to ~0.1% focused and unfocused.

### Linux

```
sudo apt-get install -y cmake ninja-build pkg-config \
    libglfw3-dev libssh-dev libwebp-dev libharfbuzz-dev \
    libfreetype-dev libfontconfig-dev libxinerama-dev \
    libxcursor-dev libxi-dev libxrandr-dev libxkbcommon-dev libgl1-mesa-dev
cmake -S . -B build -GNinja
cmake --build build
```

### Windows

Open the repo in Visual Studio with CMake support, or use Ninja from a Developer Command Prompt:

```
cmake -S . -B build -GNinja
cmake --build build --config Release
```

Requires the Windows SDK + a recent MSVC. ConPTY ( `pty_win.c` ) requires Windows 10 1809 or newer.

### Cross-platform shortcut

```
cmake -S . -B build && cmake --build build
```

Works on all three platforms once the appropriate dev packages are installed.

---

## Auto-update

On macOS, rbterm checks `https://rbterm.ximg.app/updates/manifest.json` at startup and again periodically. The manifest is signed with an Ed25519 key whose public half is embedded in the binary:

```
d0b2d28ee8eaf2be152122a239035dd1a6e6c65ffb2427a0f3adbfdda549ec15
```

When a newer version is advertised, the in-app updater downloads the zip, verifies the signature on `manifest.json` , checks the SHA-256 of the zip matches the manifest, runs `spctl --assess --type execute` on the unpacked bundle, and atomically replaces the running `.app` in place. The current rbterm process re-execs into the new bundle.

If `spctl` rejects the bundle, the update is aborted — a stale signature or a broken notarisation can't replace a working binary.

The publish path on the server side refuses to upload a zip that doesn't pass `spctl` locally first (see `tools/publish-update.sh` ), so a misfire during notarisation can't push out an unsigned build.

---

## Logging

Per-pane PTY logging is enabled in Settings → Logging.

- **Bytes written are raw PTY bytes** — every escape, every SGR, every cursor move. This makes the logs replayable with `cat file.log > /dev/tty` (terminal must support the escapes the app produced) but means plain-text grepping needs a filter:

`bash sed 's/\x1b\[[0-9;]*[a-zA-Z]//g' file.log`

- **Filename** — the filename slot is the tab's index at the moment the log was opened. Closing an earlier tab doesn't rename the remaining files; they keep their original slot number.
- **fflush** runs after every write, so a crash doesn't lose buffered bytes.
- Toggling the master switch or changing the log directory immediately opens or closes handles on every open tab.

---

## Limitations

Shopping list, not a roadmap. Open an issue if you want one moved up.

**Scrollback**: regex search, vi mode, jump to previous prompt (OSC 133 needed), URL detection + click, hints kitten, OSC 8 hyperlinks, smart selection, triple/quad-click, rectangular Alt+drag.

**Shell integration**: OSC 133 prompt marks shipped (parser + gutter badge — green = exit 0, red otherwise). Future: jump-prev/next-prompt, select-last-output, long-cmd notification — all use the already-stored marks.

**Graphics**: iTerm2 `OSC 1337 ; File=…` inline images, ligatures (HarfBuzz shaping), per-local-tab font size, background image / blur / transparency, cursor trail, min-contrast, dim inactive panes, gradients.

**Config**: TOML / Lua config file with live reload, per-local-tab profiles, per-cwd auto-switching.

**Input**: user keybindings, leader chords, Option-as-Meta left/right distinct, compose key, password autofill.

**Extensibility**: remote-control protocol ( `kitty @` equivalent), regex triggers, watchers / event hooks, status bar segments, SSH kitten-equivalent that ships terminfo + dotfiles.

**Bell / notification**: visual bell, audible toggle, OSC 9 / OSC 777, per-tab unread / activity / silence.

**Performance**: damage-tracked redraw, vsync / fps cap when idle (largely done — see the "Idle CPU floor" section in `CLAUDE.md` ), headless daemon mode.

**Windows-specific**: DirectWrite emoji rasterisation.

**Multi-window**: same-process multi-window architecture is prototyped but not shipped — today's `Cmd+N` forks a new process and you get a separate Dock icon.

---

## License

rbterm is open source — see `LICENSE` in the repository root.

Third-party components (raylib, libssh, libwebp, FiraCode font, …) retain their own licenses; copies are bundled in the source tree where appropriate.

*Generated for rbterm 0.3.28. For the latest version of this manual, see `docs/manual.pdf` in the repository, or rebuild from `docs/manual.md`.*