Vibe42

DOCUMENTATION SNAPSHOT

Vibe42 Documentation

Install Vibe42 on your computer, pair your phone or browser, and keep your coding agents running from anywhere.

10 sections

58 articles

Contents

01 **Getting started** 4

What is Vibe42 4
Install on your computer 6
Pair a device and start your first session 9

02 **How it works** 11

Computers, devices and teams 11
How connections work 13
Security model 15
Sessions, windows and Recents 17

03 **Terminal** 21

Terminal engine 21
Typing on a phone or tablet 23
Windows, panes and the terminal list 25
Keyboard shortcuts 27
Themes and fonts 29

04 **Coding agents** 31

Supported coding agents 31
Nothing injected, nothing locked in 33
Launching an agent 34
Agent status 37
Notifications 38
OSC 7501 status protocol 42
Let your agent use Vibe42 45
Claude Code in Vibe42 46
OpenAI Codex CLI in Vibe42 47
OMP in Vibe42 49
Pi Coding Agent in Vibe42 50
OpenCode in Vibe42 51
Grok Build in Vibe42 53
Devin CLI in Vibe42 54
Antigravity CLI in Vibe42 56
Cursor Agent in Vibe42 57

05 **Workspaces & files** 59

Workspaces 59
Files 61
Previewing results and forwarding ports 64
Terminal isolation 66
Compression and deduplication 69
Memos, rest reminders and housekeeping 71

06 **Remote desktop** 73

Remote desktop: getting started 73
Controls, quality and clipboard 75
Unlocking the screen and opening apps 77

07 **Account, teams & plans** 80

Account and sign-in 80
Teams, roles and sharing computers 82
Plans and slots 83
Credits and usage 85
Trial, promo codes and refunds 87

08 **Platforms** 89

macOS app 89
Linux desktop app 91
Linux server and headless machines 93
iPhone and iPad 95
Android 97
Web app and PWA 99
Windows 101
Updating and uninstalling 102

09 **v42 CLI** 106

v42 CLI overview 106
Command reference 108
Automation recipes 113

10 **Troubleshooting** 116

Connection problems 116

Not getting notifications 119

Terminal and input problems 121

Collecting logs 123

FAQ 126

01 Getting started

What is Vibe42

Access your Mac or Linux terminals, coding agents, files, and screen from any device without network setup.

vibe42.ai/docs/getting-started/what-is-vibe42

Read this page if you are evaluating Vibe42 and want to know in three minutes whether it fits your workflow and where to start.

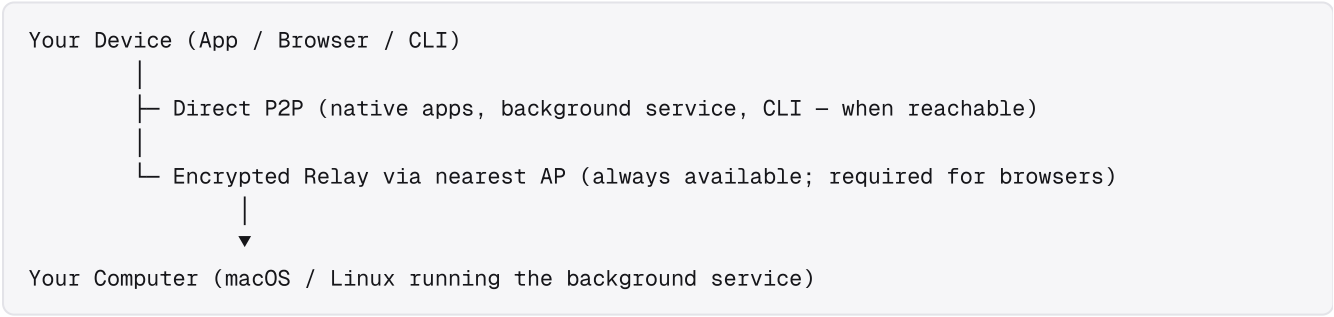
Vibe42 is a remote terminal and AI workspace that requires no network configuration. Install Vibe42 on your Mac or Linux computer, pair your phone, tablet, browser, or another computer, and keep working with the terminals, coding agents, files, and screen on that machine from anywhere.

Typical ways developers use Vibe42:

- **Follow a coding agent out the door**: Start Claude Code on your workstation, leave your desk, receive a push notification on your phone when it needs permission or finishes, and reply right away.
- **Review work from anywhere**: Open `https://app.vibe42.ai` in a browser at a cafe to inspect git diffs, run tests, and preview or download files from your home machine.
- **Unlock and drive GUI apps**: Enter your macOS password remotely to unlock a locked Mac, then run Xcode or inspect a simulator over Remote Desktop.

How it works

Vibe42 connects your devices to your computers without public IPs, inbound firewall rules, port forwarding, or manual key distribution.



Both your device and the Vibe42 background service on your computer make outbound connections to the nearest Vibe42 AP (access point). Every session uses end-to-end encryption negotiated directly between your device and your computer. An AP only sees connection metadata and encrypted traffic.

When you connect from a native app or the CLI, Vibe42 tries a direct peer-to-peer connection in parallel and switches off the relay automatically if a direct path succeeds. Browser sessions always use an AP relay. For details, see How connections work and Security model.

Three core concepts

Everything in Vibe42 revolves around three objects:

- **Computer**: A Mac, Linux machine, VM, or container running the background service that hosts your persistent terminals, coding agents, and files.
- **Device**: Any phone, tablet, desktop app, or web browser you use to connect to a computer.
- **Team**: The billing and quota unit that holds your plan, computers, devices, and monthly Credits. Every account automatically comes with a personal Team.

There is no free tier. A Team Owner can start a 3-day free trial (card required) or subscribe to a plan; members invited to someone else's Team do not need their own plan. Learn how slots and roles work in Computers, devices and teams.

What Vibe42 is not

- **Not a cloud IDE or cloud VM**: Your source code, toolchains, and local databases stay and run on your own hardware.
- **Not a traditional SSH client**: You do not manage static IPs, Dynamic DNS, VPN subnets, or `authorized_keys`. Devices authorize with short-lived pairing codes.
- **Not a code or key host**: We never store your terminal output, keystrokes, source files, or model API keys.
- **Not a hosted agent runner**: Coding agents run directly on your computer under your user account. Vibe42 does not inject hooks, skills, or plugins into your agents—see Zero injection.
- **Not a Windows Remote Desktop host**: On Windows, the Server edition runs inside WSL for terminals, coding agents, and files, but cannot view or control the Windows desktop screen (native Windows support is coming soon).

Platform support

Platform	As a Computer (controlled)	As a Device (controller)
macOS	macOS 14+ desktop app (Apple Silicon & Intel, built-in background service); macOS 13 via one-line Server installer	macOS 14+ desktop app, or web app
Linux desktop	x86_64 / arm64 AppImage (glibc 2.39+, e.g. Ubuntu 24.04 / Debian 13 / Fedora 40+, built-in background service)	x86_64 / arm64 AppImage, or web app
Linux server / container	One-line Server installer (`x86_64` and `arm64` , static binary)	CLI / web app
iPhone & iPad	Not supported	iOS / iPadOS 17+ native app (App Store), or web app
Android	Not supported	Android 11+ native app (Google Play), or web app
Windows	Server edition in WSL: terminals, agents, files; no Remote Desktop; native version coming soon	Modern browser via `https://app.vibe42.ai` (installable as a PWA, a web app you can add to your desktop or taskbar)

Next steps

- Install on your computer: Set up the macOS app, Linux AppImage, or one-line Server edition.
- Pair a device and start your first session: Connect your phone or browser and verify push notifications.
- Point your coding agent at `https://vibe42.ai/install.md` (or read Let your agent install Vibe42) to have it install Vibe42 for you.

Install on your computer

Install the Vibe42 desktop app or one-line Server edition on macOS and Linux, and approve pairing for your computer.

`vibe42.ai/docs/getting-started/install`

Use this guide to pick the right installer for your Mac or Linux machine, set up the Vibe42 background service, and link the computer to your Team.

Choose your installation path

Machine type	What to install
Your Mac (macOS 14+ with GUI)	macOS desktop app (`.dmg` from `https://vibe42.ai/#downloads` , built-in background service)
Linux desktop (x86_64 or arm64, glibc 2.39+)	Linux AppImage (`.AppImage` from `https://vibe42.ai/#downloads` , built-in background service)
Linux server, container, older distro, Windows (WSL), macOS 13, or headless Mac	One-line Server installer (`get.vibe42.app/install.sh` , installs only the background service without a desktop app)
Controller-only machine	No background service needed—use `https://app.vibe42.ai` in a browser, or install the desktop app and choose **Not now** when asked to enable the computer

> **IMPORTANT**
>
> Never run the one-line installer on a machine where the macOS desktop app or Linux AppImage is installed—the installer refuses to run alongside a desktop app. Always run the installer as the regular user who owns your terminals (do not use `root` or `sudo` unless you intentionally work as `root`). You can install and pair before activating a plan; the computer connects automatically once your plan is active.

macOS desktop app

Requires **macOS 14 or later**. Separate `.dmg` packages are provided for Apple Silicon and Intel Macs. If you already have `Vibe42.app` (or the earlier `Vibe X.app`) installed, do not reinstall from a `.dmg` —just open the existing app to update automatically.

1. Download the `.dmg` for your chip from `https://vibe42.ai/#downloads` and drag **Vibe42** into `Applications`.
2. Open Vibe42 and click **Sign in**. Your default browser opens to sign in; verify the device details and click **Approve**.
3. Depending on your account state, the app walks you through up to five setup screens:
   - **Choose a team**: Shown if you belong to multiple Teams or have pending invites.
   - **Subscribe to Vibe42 to keep working** (or **This team isn't active yet** if you are not the Team owner): Shown if the Team has no active subscription yet.
   - **Use this computer as a remotely controllable computer?**: Click **Enable this computer** to run the background service and make this Mac accessible remotely, or **Not now** to use the app only as a controller.
   - **Allow remote control of this Mac**: Grants **Screen Recording** and **Accessibility**. You can skip this step and enable it later in **Settings → Remote Desktop**.
   - **Background permission**: Keep Vibe42 enabled under **System Settings → General → Login Items & Extensions** so the background service stays online.

For macOS-specific details, see macOS.

Linux desktop AppImage

Requires **x86_64** or **arm64** Linux with **glibc 2.39+** (Ubuntu 24.04, Debian 13, Fedora 40, or newer), FUSE (the Linux component that mounts AppImage packages), and a `systemd` user session.

Optional packages improve desktop integration:

- **System tray**: StatusNotifierItem support (GNOME requires the AppIndicator extension).
- **Sandboxed file preview**: `bubblewrap` and `xdg-dbus-proxy` for code, Markdown, HTML, and PDF previews.
- **X11 input methods**: GTK4 modules for `fcitx5` or `ibus`.
- **Video decoding**: VA-API or NVIDIA drivers for hardware Remote Desktop decoding (otherwise Cisco OpenH264 is downloaded on demand).

1. Download the `.AppImage` and its matching `.sha256` file from `https://vibe42.ai/#downloads`, verify the checksum, make it executable, and launch it:

```
sha256sum -c Vibe42-*.AppImage.sha256
chmod +x Vibe42-*.AppImage
./Vibe42-*.AppImage
```

If FUSE is unavailable, install FUSE or launch with `--appimage-extract-and-run`. On first run, the app registers a `.desktop` entry, icon, and the `vibex://` URL handler. If you move the AppImage file later, launching it once updates the path automatically.

2. Click **Sign in** and click **Approve** in your browser. If the browser does not jump back to the app, copy the `vibex://…` callback link from the browser page and paste it into the app's sign-in screen.

The background service installs into `~/.v42/` and runs as the systemd user unit `v42-companion.service`. If your Linux setup lacks a `systemd` user session, use the one-line Server installer with `VIBEX_SERVICE_MODE=supervisor` instead. See Linux desktop for more.

One-line installer (Server edition)

Server edition includes only the background service without a desktop app. It works on `x86_64` and `arm64` Linux servers, headless hosts, containers, and macOS 13 (statically linked on Linux). On Windows, run the exact same command inside your WSL distribution's terminal. Requires `curl`, `tar`, and `tmux` 3.0+.

1. Run the installer as your normal user:

```
curl -fsSL https://get.vibe42.app/install.sh | sh
```

2. The script installs the background service (`v42-companion`) and prints an 8-character pairing code and approval URL:

```
Pairing code: ABCD-EFGH
https://vibe42.ai/console/pair?code=ABCD-EFGH
```

3. Approve within **10 minutes** either by opening `https://vibe42.ai/console/pair?code=ABCD-EFGH` in a browser, or by opening an already signed-in Vibe42 app and going to **Computers & Devices → Connect a new computer or server**.

The installer places files in `~/.v42` (legacy installs used `~/.vibex`) and adds `~/.v42/bin` to your `PATH`. On systemd hosts, it registers the user unit `v42-companion.service` plus a watchdog timer. Where systemd is unavailable (such as most containers), it automatically falls back to supervisor mode; you can also force this with `VIBEX_SERVICE_MODE=supervisor`. Re-running the exact same install command upgrades in place while preserving pairing.

Common background service commands

Command	Purpose
`~/.v42/bin/v42-companion status [--json]`	Check running state, pairing status, and connection health
`~/.v42/bin/v42-companion logs -f`	Follow live background service logs
`~/.v42/bin/v42-companion start`	Start the background service
`~/.v42/bin/v42-companion stop`	Stop the background service
`~/.v42/bin/v42-companion update --check`	Check for updates (or omit `--check` to update)
`~/.v42/bin/v42-companion uninstall [--purge]`	Uninstall service (`--purge` also removes local state and keys)

See Linux server for container and service tips.

After installation

Once approved, your machine appears in your Team's computer list. Next, open `https://app.vibe42.ai` or pair your phone in Pair a device and start your first session.

Pair a device and start your first session

Activate your Team, pair your phone without a password, launch a terminal or coding agent, and verify push notifications.

`vibe42.ai/docs/getting-started/first-session`

Once you have installed Vibe42 on at least one computer (see Install on your computer), follow this page to connect your phone or browser, start a session, and confirm that push notifications bring you straight back to your terminal.

1. Check account and plan prerequisites

Before your devices can open a terminal on a computer:

1. **Verify your account**: Register at `https://vibe42.ai/register` with an email and password (at least 10 characters), or choose **Continue with GitHub** / **Continue with Google**. If you registered with email, click the verification link within 24 hours—unverified accounts cannot sign in anywhere.
2. **Activate a Team plan**:
   - **Team Owner**: Open `https://vibe42.ai/console/checkout?resource=access` in a browser to start a 3-day free trial (card required) or subscribe to a plan. Pricing and subscription management are never shown inside the iPhone or Android apps.
   - **Invited member**: Accept your Team invite at `https://vibe42.ai/console/invites`. You share the Team's plan and do not need your own subscription.

2. Choose how to connect

You can control your computer from any combination of devices:

- **Web app**: Open `https://app.vibe42.ai` in any modern browser (you can also install it as a PWA, a web app you can add to your desktop or home screen). No native install is required. If you only have your phone with you, sign in to `https://app.vibe42.ai` in your mobile browser first and use it to pair the native mobile app.
- **Mobile & tablet apps**: Download Vibe42 for iPhone/iPad (iOS 17+, App Store) or Android (11+, Google Play) from `https://vibe42.ai/#downloads`.
- **Desktop app**: Use the Vibe42 app on macOS or Linux.

3. Pair the mobile app

The iPhone, iPad, and Android apps **do not use password sign-in**. Instead, you pair them from a device where you are already signed in:

1. **Generate a pairing code**: On an already signed-in desktop app or web app (`https://app.vibe42.ai`), open **Computers & Devices → Downloads and setup → Pair a phone or tablet**. Vibe42 displays a QR code and an 8-character code (`XXXX-XXXX`).
2. **Scan or type on your phone**: Open the Vibe42 mobile app and tap **Scan to pair** to scan the QR code, or enter the 8-character code manually.

3. **Verify and approve**: Return to the original device that generated the code, confirm the requesting device details match your phone, and approve the pairing.

> NOTE
>
> Pairing codes expire after **10 minutes**, work only once, and pair one device at a time. Only the original screen that generated the code can approve the request.

4. Start your first session

1. Tap or click **+** in the sidebar to create a new terminal.
2. Select your target computer (or a Workspace on that computer).
3. On the third step, choose what to launch:
   - **Terminal** (Just a shell): Opens your default interactive shell.
   - **A coding agent**: Pick from your configured agent list—by default **Claude Code**, **OpenAI Codex CLI**, **OMP**, **Pi Coding Agent**, **OpenCode**, **Grok Build**, **Devin CLI**, **Antigravity CLI**, and **Cursor Agent**.

Your agent list comes from **Settings → Agents**, where you can add or reorder entries from a catalog of around 30 supported agents or define custom commands. Vibe42 launches the agent binary already installed on your computer without modifying its config—see Launching agents.

5. Verify the end-to-end loop

Run a quick check to confirm persistence and notifications work together:

1. Launch **Claude Code** (or another agent) from the **+** flow and give it a prompt that takes a moment or asks for tool permission.
2. Lock your phone or switch to another app. The mobile app does not stay running in the background and receives push notifications through APNs (iOS) or FCM (Android).
3. When the agent's status changes to **Needs input**, **Done**, or **Error**, you receive a push notification. Tap the notification to jump directly into that exact terminal window.
4. Open `https://app.vibe42.ai` on another device and select the same window—your session is live in both places because it keeps running on your computer even when you close the app.

Desktop apps show system notifications only while running, the web app shows in-app notifications, and you can optionally link Telegram. Delivery channels and event filters are configured per computer—see Agent notifications.

Common roadblocks

What you see	Where to fix it
Device limit reached or **Computer limit reached**	Your Team has used all available device or computer slots. Sign out unused devices, delete old computers, or upgrade in Console—see Plans and slots.
No push notifications on your phone	Check OS notification permissions, per-computer notification toggles, and whether the agent changed state—see Troubleshooting notifications.
Computer appears offline	Make sure the computer is awake, connected to the internet, and running the Vibe42 background service (`v42-companion`)—see Troubleshooting connections.

02 How it works

Computers, devices and teams

Understand how computers, devices, Teams, and plan slots fit together, how slots are counted, and how to free them.

`vibe42.ai/docs/how-it-works/computers-devices-teams`

Read this page whenever you want to understand how Vibe42 counts slots on your plan, why a machine uses both a computer and a device slot, or how to free up capacity when replacing hardware.

The four core objects

Object	What it is
Computer	A Mac, Linux machine, VM, or container running the Vibe42 background service so you can access its terminals, coding agents, files, and screen.
Device	Any signed-in native app, signed-in `app.vibe42.ai` browser session, or one-line Server installation (listed as **Server edition**).
Team	The billing and access unit that owns a plan, computers, devices, members, and Credits. Every account automatically starts with a personal Team.
Plan	The subscription attached to a Team. Your plan sets your computer slots, device slots, member seats, and monthly Credits (see Console and `https://vibe42.ai/#pricing` for current numbers).

What uses a slot

Vibe42 tracks three separate quotas on a Team:

- **Computer slots**: Every **enabled** computer uses 1 computer slot.
- **Device slots**: Every signed-in native app (macOS, Linux, iOS, Android), every signed-in web app browser (`https://app.vibe42.ai`), and every one-line Server installation (shown in your device list as **Server edition**) uses 1 device slot.
- **Member seats**: Checked and consumed only when an invitee **accepts** an invitation—sending an invite does not reserve a seat in advance.

How typical machines count toward slots:

- **Mac with the desktop app**: Signing into the app uses **1 device slot**. Choosing **Enable this computer** uses **1 computer slot** as well (1 computer + 1 device total; if you choose **Not now**, it uses only 1 device slot).
- **Linux desktop AppImage (enabled) or Linux server (one-line installer)**: Each uses **1 computer slot + 1 device slot**.

What **does not** consume extra slots—or free them automatically:

- Signing into Console (`vibe42.ai/console`) in a browser to manage billing or settings does **not** use a device slot.

- Signing in again on the exact same device does **not** duplicate its device slot.
- Turning off a machine, going offline, or toggling features off does **not** release its computer or device slot.

What happens when a limit is reached

In Console, the **Computers**, **Authorized devices**, and **Members** tabs display `x of y used` so you can see current usage at a glance.

- **Computer slots full**: You can still install and pair a new computer. It comes online normally, but stays disabled as **Over plan limit** until you free a computer slot or upgrade your plan—at which point it activates automatically.
- **Device slots full**: Signing in or pairing an additional device stops with **Device limit reached** until a device is signed out or the plan is upgraded.

Freeing slots and replacing machines

- **Free a computer slot**: Go to Console → **Computers** and click **Delete**. Deleting a computer permanently removes its cloud-stored Workspaces and notes metadata, while leaving all local files and directories on the machine untouched.
  - Deleting a one-line Server computer also frees its **Server edition** device slot automatically (provided that installation does not host another computer).
  - Deleting a Mac or Linux desktop app computer frees **only the computer slot**; the desktop app stays signed in as a device until you also sign it out under **Authorized devices**.
- **Free a device slot**: Go to Console → **Authorized devices** and click **Unpair** next to the device (it must sign in or pair again to reconnect). You can also click **Unpair offline devices** to sign out every device that has been offline for more than 3 days in one action.
- **Replacing vs. upgrading a computer**: When moving to new hardware, delete the old computer in Console first, then install on the new machine. Re-running the one-line installer on the *same* machine upgrades in place and does not use a new slot.

Who can manage what

- **Owner**: Selects plans, manages billing and payment, and has full control over computers, devices, and members.
- **Admin**: Can delete any computer, sign out any device, and view Team usage, but cannot change plans or billing.
- **Member**: Can only sign out their own devices. Owners and Admins can create (add) computers by default; Members cannot unless granted permission individually.

Computer visibility within a Team

Each computer has an access setting of **Only me** (default) or **Whole team** to control whether Members on the Team can see and connect to it. For full role permissions and sharing rules, see Teams, roles and sharing computers and Plans and slots.

How connections work

Learn how Vibe42 selects the nearest AP, upgrades native apps to direct P2P, separates terminal and file traffic, and handles firewalls.

`vibe42.ai/docs/how-it-works/connections`

Read this page to understand how Vibe42 routes traffic between your devices and computers, when sessions use direct peer-to-peer versus an AP relay, and what outbound ports corporate networks require.

Connection paths at a glance

Path type	Who can use it	Counts toward Credits?
Direct	Native apps (macOS, Linux, iOS, Android), the Vibe42 background service, and `v42` CLI when peer-to-peer dialing or hole punching succeeds	**No**—direct P2P traffic is never billed
Relayed	Always used by browser sessions (`app.vibe42.ai`); used as an automatic fallback when native apps cannot reach each other directly, or when **Force relay** is on	**Yes**—application data relayed through an AP counts toward Credits
Mixed	Native apps and CLI when some of a computer's three channels connect directly while another remains relayed	Only the relayed channel's data counts

AP (access point) selection

Neither your device nor your computer needs a public IP or open inbound port. Both make outbound connections to a Vibe42 AP.

On startup, clients measure RTT (round-trip latency) across available regions at the same time—including Hong Kong, Los Angeles, Tokyo, Frankfurt, and others—and connect to the lowest-latency AP. You can manually pin a preferred AP in **Settings → Network diagnostics**; if a pinned AP becomes unavailable, the client falls back automatically.

Upgrading from relay to direct P2P

Sessions start immediately over the AP relay so you never wait on network setup before typing. In the background, native apps, the background service, and the CLI exchange candidate addresses—LAN subnets, private interfaces, VPN virtual adapters, and public endpoints—and try direct dialing and NAT hole punching (opening a direct path through home or office routers) in parallel.

If a direct path succeeds, traffic switches over automatically without dropping your terminal, and data stops flowing through the AP. If router or firewall rules block a direct connection, the session stays on the encrypted relay without interruption.

Three independent channels per computer

For each connected computer, Vibe42 maintains three separate connections:

- **control**: Metadata, state synchronization, and management commands.

- **terminal**: Interactive terminal input and output streams.
- **file**: File browsing, previews, uploads, and downloads.

Because `file` and `terminal` travel over separate connections, transferring large archives does not slow down typing in your active shell.

Disconnections and automatic recovery

Vibe42 recovers automatically from network switches (Wi-Fi to cellular), laptop sleep and wake cycles, and brief dropouts:

- **Fast resume**: If a drop recovers on the same AP within **30 seconds**, the session resumes quickly in place.
- **Full reconnect**: Longer outages or AP changes start a fresh handshake with retry intervals that back off from **1 to 60 seconds**.
- **Permanent errors**: Non-recoverable failures—such as a revoked device, exceeded slot limits, or rejected credentials—stop retrying immediately and display the exact reason.
- **Mobile backgrounding**: On iOS and Android, moving the app to the background closes active connections so the app uses zero background battery; returning to the foreground reconnects within a few seconds.

Force relay

Both devices and computers have a **Force relay** toggle. If either side enables **Force relay**, Vibe42 skips direct dialing and hole punching and keeps all traffic on the AP relay. Web browser sessions always run on the relay.

Network diagnostics

Open **Settings → Network diagnostics** to inspect live connection details (refreshed every 2 seconds):

- Current AP name, region, transport protocol, and RTT.
- Per-computer path status across all three channels: **Direct QUIC**, **Direct TCP**, **WebSocket**, **Relayed**, or **Not connected**.

From a terminal, run `v42 network [computer]` to print the same diagnostic data as JSON (see CLI commands).

Corporate networks and firewalls

Vibe42 requires outbound connectivity only:

- `ap-<region>-direct.vibe42.app`: UDP QUIC on port `6042`, TCP on port `6042`, and WSS on port `443`.
- `ap-<region>.vibe42.app`: WSS on port `443` only (fronted by Cloudflare).

Clients try protocols in order: **QUIC → TCP → WSS**. On restricted corporate or hotel networks that allow only outbound TCP `443`, Vibe42 still connects over WSS, though strict firewalls usually block P2P hole punching—meaning sessions stay on the relay and use Credits. See Credits and usage and Troubleshooting connections.

Security model

Review Vibe42's end-to-end Noise encryption, local Ed25519 keys, zero payload retention, audit boundaries, and revocation rules.

`vibe42.ai/docs/how-it-works/security`

Read this page to verify how Vibe42 protects your terminals and files in transit, what keys exist and where they live, and what our servers can and cannot see.

Vibe42 is built around three principles: **end-to-end encryption**, **zero content retention**, and **short-lived pairing instead of shared machine passwords**.

Where keys live and how sessions encrypt

Every installation—the Vibe42 background service on each computer and each client app—generates its own **Ed25519 root keypair** locally on the machine. Private keys stay inside the operating system's secure storage, never leave the device, and are never held by Vibe42 servers.

Every connection between a device and a computer performs an end-to-end **Noise** encryption handshake:

- **Direct P2P connections** flow straight between your device and your computer over the Noise-encrypted tunnel.
- **Relayed connections** negotiate session keys directly between your device and your computer as well. The AP forwards encrypted data and cannot decrypt your traffic.

What Vibe42 sees, never sees, and never trains on

What an AP or server can see

- Authenticated device and computer identities
- Connection timestamps, AP region, and byte counts
- Encrypted Noise traffic
- Notification titles and bodies—the only content that passes through and is stored on Vibe42 servers in readable form, because the server must hand it off to Apple, Google, or Telegram; can be turned off per computer, and when turned off is never generated (see Notifications)

What we never store or read

- Terminal sessions, commands, keystrokes, or screen output
- Source code, directory listings, or file contents
- Port-forwarded data or session encryption keys

What we never do

- Train AI or machine learning models on your sessions, code, files, or metadata
- Handle your model API keys: keys configured on your machine stay local and never pass through Vibe42

What audit logs record

To enforce quotas, access control, and security reviews, Vibe42 records only administrative and routing metadata:

- Account sign-ins, step-up verifications, and pairing or revocation events.
- Access control decisions and Team management actions.
- Connection path types (direct vs. relayed), P2P connection outcomes, source and target IDs, AP region, byte totals, and timestamps.

Audit records never contain terminal text, file contents, port-forwarding data, or notification bodies.

Account security

- **Passwords**: Stored only as **Argon2id** hashes.
- **Passkeys & 2FA**: You can register Passkeys (private keys never leave your device) and TOTP two-factor authentication (time-based one-time codes from an authenticator app) with one-time recovery codes in Console.
- **Step-up verification**: Sensitive actions—such as changing security settings, linking GitHub/Google, revoking account-level installations, or deleting your account—require an extra verification check, prioritized in order: **Passkey > TOTP > password**.
- **OAuth identity**: **Continue with GitHub** and **Continue with Google** accept only verified primary email addresses reported by the identity provider.

See Account and sign-in for setup instructions.

Device authorization and revocation

- **Short-lived pairing**: Pairing codes expire after **10 minutes**, can be used only once, and can be approved only on the exact screen that generated the code.
- **Fast revocation**: You can sign out any device or delete any computer from Console or another authorized device at any time. Revocation takes effect within **60 seconds**, closing active connections—including established direct P2P links.
- **Strict Team isolation**: Computers and devices across different Teams cannot reach one another, and there is no administrative backdoor into your background service.

Your side of the security boundary

- **OS user permissions**: The background service runs under the local OS user account that installed it. Anyone allowed to connect to that computer in Vibe42 has the same file and process permissions as that local user.
- **Team visibility**: Owners and Admins can access all computers in a Team; control whether Members can connect by setting each computer to **Only me** or **Whole team** (see Teams, roles and sharing computers).
- **Approve only your own requests**: Never share a pairing code with someone you do not trust, and never click **Approve** on a browser sign-in prompt you did not start.

For legal data handling terms, read our Privacy Policy at `https://vibe42.ai/privacy`.

Sessions, windows and Recents

See how Vibe42 persists terminals with tmux, recreates them after a reboot, tracks windows in Recents, and opens multi-window views.

`vibe42.ai/docs/how-it-works/sessions`

Read this page to understand how Vibe42 keeps your shells and coding agents running across disconnects, how window and pane actions map to tmux, and how **Recents** tracks sessions across computers.

Persistent tmux-backed sessions

On every computer, Vibe42 runs your terminals inside tmux (a terminal multiplexer that keeps programs running in the background on the host machine) using windows and panes (splits inside a window). Closing the Vibe42 app, losing cellular signal, or switching from your desktop to your phone leaves your running commands and coding agents untouched—reopen the window from any device to pick up right where you left off.

> **IMPORTANT**
>
> Sessions are unaffected by network drops, closing the app, or restarting or updating the background service. When your computer restarts, running programs stop, and the background service recreates your windows the next time it starts as described in the next section.

After your computer restarts

The background service continuously records the state of every terminal window on your computer. When your computer restarts, the background service uses that record to recreate your windows the next time it starts.

Recreation happens only when all Vibe42-managed terminals on that computer have stopped—most commonly after a reboot, or if all terminals stop for any other reason. Because the background service runs under your system user account, it starts after you log back in to your system account on a Mac; on an SSH-only Linux server, enable user lingering (see Linux server) so the background service starts automatically at boot. Restarting or updating only the background service (without restarting the computer) leaves your terminals running, so nothing needs to be recreated and programs inside your windows are not interrupted.

What comes back

- **Windows and panes**: The same panes and split layout.
- **Window names**: Windows with a fixed name keep it; the others keep renaming themselves after the command that is running.
- **Custom titles**: Any custom title you set with **Rename window**.
- **Workspace**: The Workspace each window belongs to.
- **Terminal isolation**: Windows with terminal isolation enabled reconnect to their isolated folder copy as long as that folder still exists.
- **Working directory**: Each pane opens in the folder it was in before, or falls back to the default folder if that folder no longer exists.

- **Active selection**: The window and pane that were active (selected) before the restart.

When each pane is recreated, the top of the pane replays the colored output saved before the restart, followed by a dimmed line noting that the session was restored and when the output above was saved, and then the new shell or program starts below it. Full-screen programs (such as editors or `htop` that take over the whole screen) save only the single screen that was visible at the time.

Coding agents and launch commands

- **Started with a launch command**: Windows created by picking an agent in the new-terminal flow or running `v42 term new --agent` or `--command` run that same launch command again when recreated. Agents that support reopening by session—including all 9 default agents (see Supported agents)—reopen your last conversation directly from the agent's own storage. If the agent had not started a conversation before the restart (for example, before you sent a first message), Vibe42 simply starts the agent again. Any turn that was in progress at the moment of the restart was interrupted and does not continue on its own—send another message to have the agent keep going. Agents that do not support reopening by session, as well as custom commands, run the same command again from the beginning.
- **Agents you typed into a plain shell**: If you started an agent by typing its command into a regular shell yourself, Vibe42 replays the saved output and gives you a plain shell in the same folder without re-running the agent automatically.
- **Other programs**: Dev servers, builds, tests, and other programs are not re-run automatically. Their saved output is replayed at the top of the pane, and you can start them again manually.

Limits

- **Output size**: Each pane saves up to 1 MiB of output and drops the oldest part when exceeded. Across the whole computer, saved output is capped at 64 MiB total, dropping non-agent panes first when exceeded.
- **When output is saved**: Output is saved 5 seconds after output stops, at most once every 30 seconds while output continues, and once more when the background service exits normally (including a normal shutdown). On a sudden power loss, roughly the last 30 seconds of output may not be saved.
- **Local storage only**: Saved output stays on that computer and is never uploaded to Vibe42.

Window and pane actions

You can organize terminals using the window and pane menu (see Windows and panes for UI gestures and shortcuts):

- **New window here**: Create a new tmux window in the current directory.
- **Split right** / **Split down**: Split the active terminal horizontally or vertically into panes.
- **Rename window**: Set a custom title that overrides automatic terminal title updates (persists while the background service is running and is restored after a computer restart, though restarting or updating only the background service may reset it to the automatic title).
- **Move into this window**: Move an existing pane into the current window.
- **Break into new window**: Detach the active pane into its own standalone window.
- **Close window** / **Close pane**: Close the window or pane after a confirmation prompt warning that running processes inside it will be terminated.

Recents

Recents gives you quick access to up to **100** recently visited windows across your computers, with support for pinning (**Pin**) important entries to the top.

Each entry shows one of five states:

State	Meaning
Available	The computer is online and the window is ready to open.
Computer offline	The host computer is currently offline or unreachable.
Syncing…	Connected to the computer and waiting for live window details.
Window closed	The computer confirmed that this tmux window was closed.
Window no longer available · Remove from Recents	An outdated entry from an incompatible earlier session format.

Vibe42 marks a window closed **only** when the computer confirms its deletion—a network drop never marks your entry closed. Closed entries stay visible so you know what happened rather than seeing a window disappear silently; choose **Hide from Recents** whenever you want to clear an entry. A window recreated after a computer restart is treated as the same window, so its **Recents** entry and pin (**Pin**) continue pointing to it instead of showing as closed, and open terminal links in the web app automatically jump to the recreated window.

Multi-device viewing

Multiple devices—or multiple Team members on a shared computer—can watch and type in the same terminal at the same time. The **Who's viewing** indicator shows other active connections currently attached to that window.

Open in New Window

To view terminals side by side, open a sidebar item's menu and select **Open in New Window**. Closing that secondary app window closes only your local view and **never terminates** the remote process.

Platform	Behavior of Open in New Window
macOS app	Opens a staggered native macOS window
iPad app	Opens an iPadOS system multi-window scene (Split View / Stage Manager)
iPhone app	Not offered
Android app	Offered only in Android desktop mode
Web app	Opens a separate browser popup window

How Vibe42 interacts with your own tmux usage

Vibe42 manages its own tmux sessions and leaves your personal default tmux sessions untouched. Keep one difference in mind: panes created through Vibe42's UI or `v42` CLI automatically answer OSC 7501 probes (control sequences that coding agents use to report their status to Vibe42) through the background service,

whereas panes you create manually with raw `tmux` commands inside a shell do not have background service probe handling attached—see OSC 7501 protocol.

03 Terminal

Terminal engine

How Vibe42 renders terminals with Ghostty, encodes TUI keystrokes, displays inline images, and handles clipboard and file attachments across devices.

`vibe42.ai/docs/terminal/engine`

Vibe42 uses the Ghostty terminal emulation core across every client so command output, full-screen TUIs (text-based interfaces like Claude Code or Neovim), and coding agents behave the same way on a browser, phone, tablet, or desktop app. Read this page when you want to check rendering backends, keyboard protocol support, inline image limits, cross-device copy, or paste and attachment quotas.

Ghostty core and rendering backends

Every Vibe42 client runs the same VT parser (the component that interprets terminal text and control sequences) from Ghostty, paired with a native GPU renderer on your device. Frame updates follow your display's refresh rate automatically, staying smooth on high-refresh-rate screens.

Client platform	Emulation core	Rendering path
macOS & iOS / iPadOS	`libghostty-vt`	Metal
Linux desktop app	`libghostty-vt`	wgpu
Android	`libghostty-vt`	TextureView (follows high-refresh-rate displays)
Web app (`app.vibe42.ai`)	`ghostty-vt` (WASM)	WebGL, with automatic fallback to Canvas 2D

The program running inside your shell sets the foreground and background colors, and you can switch the overall color scheme at any time in Themes and fonts.

Smooth output and scrollback

Fast build logs and frequent TUI redraws stay responsive because Vibe42 draws text on the GPU instead of creating browser DOM nodes.

- **Local scrollback**: Each pane (one split of a terminal window) keeps a 10,000-line scrollback buffer on your device so you can scroll up immediately without waiting on the network.
- **Synchronized output (DEC mode 2026)**: When a TUI or coding agent wraps a screen update in begin and end markers, Vibe42 waits for the complete update and draws it all at once to prevent flicker and tearing.
- **Display-paced frames**: Rendering matches your screen's native refresh rate without manual tuning.

Keyboard protocols for TUIs and agents

Terminal apps such as Claude Code and terminal editors rely on extended keyboard protocols to tell combinations like `Shift+Enter` , `Ctrl+;` , or `Alt+Arrow` apart from plain keys.

The Vibe42 background service on your computer tracks which input modes the active program has turned on:

- **Kitty keyboard protocol**
- **xterm** `modifyOtherKeys`
- **Application cursor keys**
- **Bracketed paste mode**

When you press a key on any device, the background service encodes the keystroke on the computer based on the active terminal state. Modifier combinations inside TUI apps work the same way they do in a local terminal window, with no manual escape-sequence setup. For app-level shortcuts that Vibe42 handles before the terminal sees them, see Keyboard shortcuts.

Inline images (Kitty graphics protocol)

Vibe42 supports the Kitty graphics protocol so command-line tools can display charts, screenshots, and image previews directly inside the terminal grid. The background service decodes the image data on your computer and sends it to connected devices.

Limit	Value
Maximum size per image	16 MiB
Storage budget per pane	64 MiB
Concurrent images per pane (web app)	Up to 5 images

Remote copy with OSC 52

When a program running on your computer sends an OSC 52 sequence (a terminal control code used to copy text, such as yanking in Neovim or copying in tmux), Vibe42 puts that text into the system clipboard of the **device currently viewing** the terminal. This is the only cross-device path for a terminal program to write to your clipboard.

> **NOTE**
>
> On desktop apps, the Vibe42 window must be in focus when the OSC 52 sequence arrives for the clipboard write to succeed. Vibe42 shows a confirmation message whenever an OSC 52 copy finishes.

Paste and file attachment limits

You can paste text, paste images, or drag files directly onto a terminal. Vibe42 uploads images and files to your computer and inserts their file paths at your cursor.

Input type	Limit and behavior
Plain text paste	Up to 1 MiB per paste
Image paste or upload	Up to 4 MiB as-is; larger images and HEIC photos are converted to JPEG automatically (max 4096 px on the longest edge)

Input type	Limit and behavior
Drag-and-drop or pasted files	Uploaded to the computer; the file path on the computer is inserted into the terminal
Phone and tablet attachments	Take a photo, pick from your photo library, or choose a file (see Typing on a phone or tablet)
Insert Remote Path	No upload—inserts the single-quote-escaped path of an existing file or folder on the computer at the cursor without pressing Enter (see Files)

Typing on a phone or tablet

Use the command composer, draggable D-pad, sticky modifiers, touch gestures, keyboard avoidance modes, and photo attachments on iOS and Android.

`vibe42.ai/docs/terminal/mobile-input`

Phone and tablet keyboards lack physical modifier keys, an `Esc` key, and stable screen space for full-screen TUIs (text-based terminal interfaces). Vibe42 adds touch controls on iPhone, iPad, and Android—plus hardware-keyboard support on tablets—so you can run shells and coding agents comfortably away from your desk. Read this page when you want to set up mobile input controls, touch gestures, or keyboard behavior.

Command composer

Typing directly into a live shell on a glass keyboard makes autocorrect mistakes risky. Open the command composer bar when you want to write a multi-line agent prompt, check command flags before running them, or paste and edit a long command safely.

- Tap **Send to terminal** to send your draft to the terminal.
- Choose **Send with Enter** to run the command right away, or **Send Line by Line** to feed multi-line input into an interactive prompt or shell one line at a time.

Draggable D-pad and modifier keys

When an agent or TUI asks you to pick an item from an interactive menu, opening the system keyboard covers half the screen. Use the floating **D-pad** instead:

- Gives you the four arrow keys and `Space` without opening the on-screen keyboard.
- Drag the control anywhere over the terminal so it does not cover active prompts.

On the accessory toolbar, modifier keys (`Ctrl` , `Alt` , and others) work like sticky keys:

- **Single tap**: Applies the modifier to the next key you press, then turns off automatically.
- **Double tap**: Locks the modifier on for repeated key combinations until you tap it again.

Custom shortcut buttons

You can pin your most-used keys and key combinations to the mobile shortcut bar.

- Custom shortcut buttons are saved to your account and sync across all signed-in mobile devices.
- When a button uses a recognized key name, Vibe42 displays its key icon automatically to save space on the bar.

Touch gestures

Use these touch gestures on the terminal canvas to scroll, select text, or interact with mouse-aware TUI programs:

Gesture	Action
Single-finger drag	Scroll through local terminal history
Long-press (500 ms)	Select words and drag the selection handles to copy text
Two-finger drag	Send scroll or drag events when the active program has mouse reporting turned on
Two-finger pinch	Increase or decrease the terminal font size on this device

Software keyboard avoidance

Resizing the PTY (the underlying channel between the terminal and the running program) every time the on-screen keyboard opens or closes can make full-screen TUIs redraw and jump around. Vibe42 gives you three keyboard avoidance modes:

Mode	Behavior when the keyboard appears
Shift page	Slides the terminal view up without changing the PTY row or column count
Resize terminal	Shrinks the terminal row count to fit the visible area above the keyboard
Follow cursor	Keeps the PTY size unchanged and pans the view only when needed to keep the cursor visible

Camera, photos, and file attachments

When you want to show a UI screenshot or pass a file to an agent from your phone, attach it directly to the terminal:

1. Open the attachment menu and choose **Take Photo**, **Choose from Library**, or **Choose File**.
2. Vibe42 uploads the file to your computer and inserts its file path into the active terminal at your cursor.

The attachment menu also includes **Insert Remote Path** for inserting the path of a file or folder that is already on that computer into the terminal: pick the file or folder in the file browser window that opens, and Vibe42 inserts its path at your cursor without uploading anything or pressing Enter (see Files).

Images over 4 MiB and HEIC photos are converted to JPEG (up to 4096 px on the longest edge) automatically before upload. See Terminal engine for all paste and attachment limits.

iPad hardware keyboards and IME input

When you attach a physical keyboard to an iPad, you can use desktop-style keyboard shortcuts. Hold `⌘` at any time to bring up the on-screen shortcut cheat sheet; see Keyboard shortcuts for the full list.

For Chinese, Japanese, Korean, and other multi-step input methods (IME):

- **iOS and iPadOS** use native system text input so candidate bars and composition work normally.
- **Web app** displays in-progress IME composition text before sending the final characters to the terminal.
- **Linux desktop** works out of the box on Wayland; on X11, install the GTK4 input method module for `fcitx5` or `ibus`.

Windows, panes and the terminal list

Organize persistent terminal windows and split panes, browse and sort the terminal list, search across computers, and see who else is viewing a session.

`vibe42.ai/docs/terminal/windows-and-panes`

Every terminal window in Vibe42 keeps running on your computer, and you can split a window into multiple panes (individual shell areas inside the same window). Closing the Vibe42 app or switching devices leaves your programs running; see Sessions for how session persistence works. Read this page when you want to arrange windows and splits, browse the terminal list, or jump to a session with global search.

Terminal windows and split panes

Because the Vibe42 background service manages windows and panes on your computer, layout changes you make in the desktop app show up right away on your phone or browser.

Use the window and pane menu to arrange your layout:

Menu action	What it does
New window here	Opens a new terminal window on the same computer (and in the same Workspace, if applicable)
Split right	Splits the active pane vertically and opens a new shell on the right
Split down	Splits the active pane horizontally and opens a new shell below
Rename window	Sets a custom title for the terminal window across all your devices
Move into this window	Moves another terminal into the current window as a split pane
Break into new window	Turns the active split pane into its own standalone terminal window
Close pane	Closes the active split pane after confirmation
Close window	Closes the entire terminal window and all panes inside it after confirmation

Renaming and rearranging panes

Use **Rename window** to label long-running tasks (such as `api-server` or `claude-refactor`) so they are easy to find on your phone. When a side-by-side split feels cramped on a small screen, choose **Break into new window** to turn that pane into a full window. To compare two separate terminals side by side later, use **Move into this window** to combine them into one window.

Closing vs. disconnecting

Leaving a terminal view or quitting Vibe42 never stops your programs. By contrast, choosing **Close window** or **Close pane** permanently closes that window or pane on your computer. Vibe42 always asks you to confirm first and warns that running programs inside it will stop.

Opening terminals in a separate app window

On the desktop and web apps, you can open extra app windows (**New Window**: `⌘N` on macOS, `⌥⌘N` on Apple browsers, `Ctrl+Shift+N` on Linux desktop, or `Alt+Shift+N` on other browsers) to watch several terminals across multiple monitors.

Closing a local app window on macOS (`⇧⌘W`) only closes that window on your screen without stopping the terminal running on your computer. Multi-window options depend on your operating system and browser, as shown in the app. For all window and pane shortcuts, see Keyboard shortcuts.

The terminal list

The home terminal list shows active terminals across every online computer in your Team:

- **Group by computer**: Groups terminals under the computer running them.
- **Sort by added time** / **Sort by name**: Sorts terminals by creation time or alphabetically by name.
- **List view** / **Grid view**: Switches between a compact list and a grid view.

If you organize projects by folder, you can also browse terminals from the **Agents**, **Workspaces**, and **Recents** tabs in the sidebar; see Workspaces.

Global search and Who’s viewing

Press `⇧⌘P` (or `Ctrl+Shift+P` on Windows, Linux, and Android; unavailable in Firefox) to open global **Search** and jump straight to:

- **Workspaces**
- **Terminals**
- **Remote Screens**
- **Remote Windows**

When another connection—such as your phone, a second browser tab, or a teammate sharing the computer—has the same terminal open, the **Who’s viewing** indicator shows everyone viewing that session in real time.

Keyboard shortcuts

Complete keyboard shortcut reference for Vibe42 across macOS, iPadOS, Linux, Windows, Android, and web browsers, plus focus and browser override rules.

`vibe42.ai/docs/terminal/shortcuts`

Vibe42 includes keyboard shortcuts for switching terminals, splitting panes (individual shell areas inside a window), jumping between Workspaces, and changing font size without reaching for the mouse. Keep this page handy when you work on macOS, Linux, an iPad with an external keyboard, or the web app.

How shortcuts are dispatched

Four rules keep Vibe42 shortcuts from conflicting with your terminal programs or browser:

- **Terminal focus comes first**: When your cursor is inside a terminal, keys go straight to the running program by default—so combinations like `Ctrl+B` or Vim shortcuts reach your shell untouched. Vibe42 only intercepts the shortcuts listed in the table below.
- **Disabled while overlays are open**: Global shortcuts are paused while a dialog, sheet, or pop-up menu is open.
- **Web browser alternatives**: Browsers reserve shortcuts like `⌘N`, `⌘T`, and `⌘W` (or `Ctrl+N`, `Ctrl+T`, `Ctrl+W`) for their own windows and tabs. On `app.vibe42.ai`, those actions use the `⌥` or `Alt` combinations shown in parentheses below.
- **Firefox limitation**: Firefox reserves `Ctrl+Shift+P` for Private Browsing and does not let websites override it, so you cannot open **Search** with a keyboard shortcut in Firefox on Windows or Linux.

> **TIP**
>
> On an iPad with a physical keyboard attached, hold `⌘` to view an on-screen shortcut cheat sheet. See Typing on a phone or tablet for more tablet input details.

Shortcut reference

In the table below:

- **mac** covers the macOS app, iPadOS with a physical keyboard, and the web app on Apple devices (web-only overrides appear in parentheses).
- **other** covers the Linux desktop app, Android with a physical keyboard, and the web app on Windows or Linux (web-only overrides appear in parentheses).

Action	mac	other
Settings	`⌘,`	`Ctrl+,`
New Window	`⌘N` (Web `⌥⌘N`)	`Ctrl+Shift+N` (Web `Alt+Shift+N`)
New Workspace…	`⇧⌘N` (Web `⌥⇧⌘N`)	`Ctrl+Alt+N`
New Terminal	`⌘T` (Web `⌥⌘T`)	`Ctrl+Shift+T` (Web `Alt+Shift+T`)
Search	`⇧⌘P`	`Ctrl+Shift+P` (unavailable in Firefox)

Action	mac	other
Toggle Sidebar	`⌘B`	`Ctrl+B` (outside terminal) / `Ctrl+Shift+B`
Toggle Full Screen	`⌘↩` or `^⌘F`	`Ctrl+Enter`
Close Window	`⇧⌘W` (not on Web)	—
Close All Windows	`⌥⇧⌘W` (macOS app only)	—
Close Terminal	`⌥⌘W` (Web `⌥⇧⌘W`)	`Ctrl+Shift+W` (Web `Alt+Shift+W`)
Close Pane	`⌘W` (Web `⌥⌘W`)	—
Previous Terminal / Next Terminal	`^⇧Tab` / `^Tab`; `⇧⌘[` / `⇧⌘]` (Web `⌥⇧⌘[` / `]`)	`^⇧Tab` / `^Tab`; `Ctrl+Shift+←` / `→`; `Ctrl+PgUp` / `PgDn`
Terminal 1–8 / Last Terminal	`⌘1-8` / `⌘9`	`Alt+1-8` / `Alt+9`
Workspace Group 1–8 / Last Workspace Group	`^⌘1-8` / `^⌘9`	`Alt+Shift+1-8` / `Alt+Shift+9`
Split Right / Split Down	`⌘D` / `⇧⌘D`	`Ctrl+Shift+O` / `Ctrl+Shift+E`
Previous Pane / Next Pane	`⌘[` / `⌘]`	`Ctrl+Meta+[` / `]`
Select Pane ↑↓←→	`⌥⌘↑↓←→` (Web `⌥⇧⌘↑↓←→`)	`Ctrl+Alt+↑↓←→`
Move Divider ↑↓←→	`^⌘↑↓←→`	`Ctrl+Meta+Shift+↑↓←→`
Increase Font Size / Decrease Font Size / Reset Font Size	`⌘=` or `⌘+` / `⌘-` / `⌘0`	`Ctrl+=` / `Ctrl+-` / `Ctrl+0`
Scroll to Top / Scroll to Bottom	`⌘Home` / `⌘End`	`Shift+Home` / `Shift+End`
Page Up / Page Down	`⌘PgUp` / `⌘PgDn`	`Shift+PgUp` / `Shift+PgDn`
Copy / Paste / Select all	`⌘C` / `⌘V` / `⌘A`	`Ctrl+Shift+C` (`Ctrl+Insert`) / `Ctrl+Shift+V` (`Shift+Insert`) / `Ctrl+Shift+A`
Line start / Line end / Delete to line start; Word left / right	`⌘←` / `⌘→` / `⌘⌫`; `⌥←` / `⌥→`	—

For how Vibe42 encodes extended key combinations inside TUI (text-based terminal interface) apps, see Terminal engine.

Menu commands without default shortcuts

You can run these actions from the app menu bar or toolbar; they do not have default keyboard shortcuts:

- **Toggle Right Panel**
- **Computers**
- **Connect a new computer or server…**
- **Pair a phone or tablet…**
- **Keyboard Shortcuts**

Unsupported actions

Vibe42 does not include built-in commands or shortcuts for:

- Clearing the terminal scrollback with `⌘K` (run `clear` or press `Ctrl+L` inside your shell instead)
- Temporarily zooming a single split pane
- Equalizing split pane sizes automatically

Themes and fonts

Customize Day and Night color themes from 114 palettes, choose built-in monospace fonts, tune line height and ligatures, and keep CJK characters aligned.

`vibe42.ai/docs/terminal/themes-and-fonts`

A terminal font size and contrast that look comfortable on a 27-inch desk monitor rarely work well on a phone screen outdoors. Vibe42 includes 114 color themes and five built-in monospace fonts, and saves display settings locally on each device. Read this page when you want to configure color themes, font settings, CJK character alignment, or quick zoom shortcuts.

Color themes and Day / Night switching

Vibe42 comes with **114 color themes**—**81 dark** and **33 light**—from `terminalcolors.com`.

- **Separate Day and Night picks**: Choose one palette for **Day** and another for **Night**.
- **Follow system**: Switch automatically whenever your operating system changes between light and dark mode, or lock the app to either mode.
- **Matching app interface and terminal**: Changing a theme updates both the terminal colors and the surrounding app interface so sidebars, tabs, and panes (splits inside a terminal window) share the same contrast.

Standard ANSI colors output by programs in your shell map to the active theme's palette; see Terminal engine for rendering details.

Bundled fonts and typography settings

Every client includes five developer monospace fonts so terminals look the same without installing extra fonts on each device:

- **Geist Mono**
- **JetBrains Mono**
- **Fira Code**
- **Blex Mono**
- **Noto Sans Mono**

In **Settings**, you can adjust five text settings:

Setting	What it controls
Font	Picks your primary monospace font from the built-in list
Font Size	Sets the base character size in the terminal grid

Setting	What it controls
Line Height	Adjusts vertical spacing between lines for denser logs or easier reading
Ligatures	Turns programming ligatures (combined symbols like `->`, `===`, and `!=`) on or off in supported fonts
Bolder text	Draws terminal text with slightly thicker strokes for clearer reading on low-DPI or bright screens

Resizing text on the fly

You can change the font size at any time without opening **Settings**:

- **Keyboard**: Press `⌘=` (or `⌘+`) to make text larger, `⌘-` to make it smaller, and `⌘0` to reset on macOS and iPadOS (`Ctrl+=`, `Ctrl+-`, and `Ctrl+0` on Linux, Windows, and Android). See Keyboard shortcuts for the full list.
- **Touch**: Pinch with two fingers anywhere on the terminal canvas on iOS or Android to zoom text in or out; see Typing on a phone or tablet.

CJK double-width alignment and font fallback

When English text, icons, and Chinese, Japanese, or Korean characters appear on the same line, mismatched fallback fonts can push TUI (text-based terminal interface) box borders out of alignment. Vibe42 looks up characters through a fixed four-step font order:

```
Primary font → Symbol fallback → CJK fallback → monospace
```

Each Chinese, Japanese, or Korean character is measured and locked to exactly two terminal columns, keeping split borders, tables, and coding agent diffs lined up.

Per-device storage

All terminal display settings—theme mode, Day and Night palettes, font family, font size, line height, ligatures, and **Bolder text**—are saved **on the current device only**. Setting a larger font or a high-contrast light theme on your phone never changes the layout on your desktop computer.

04 Coding agents

Supported coding agents

Run any terminal-based coding agent in Vibe42, and see how native OSC 7501 reporting and the wrapper engine track agent status.

`vibe42.ai/docs/agents/supported-agents`

Any program that runs in a terminal works in Vibe42. Because Vibe42 connects you to persistent terminals rendered with the Ghostty terminal engine on your Mac or Linux computer, you do not need a Vibe42 plugin or integration just to run a tool. Read this page to see how Vibe42 tracks agent status and sends push notifications across different coding agents.

Three tiers of status support

While any CLI (command-line tool) runs normally in a terminal window, Vibe42 uses two mechanisms—the OSC 7501 terminal escape sequence (a short control code a program prints to report status) and the built-in wrapper engine in the Vibe42 background service—to power the **Agents** tab, sidebar status dots, and notifications.

1. **Native full reporting**: The agent emits OSC 7501 sequences directly, reporting its own state, progress, summary, and specific input prompts without involving the wrapper engine. Supported by **Claude Code** (2.1.295 and newer) and **Pi Coding Agent** (1.1.0 and newer).
2. **Partial self-reporting + wrapper engine**: The agent emits partial status signals, and the background service's wrapper engine fills in the remaining states using screen rules and session logs. Supported by **OpenAI Codex CLI**, **OMP**, **Grok Build**, **Antigravity CLI**, **Crush**, and **Amp**.
3. **Wrapper engine reporting**: The agent does not emit OSC 7501 yet. When you start it through Vibe42, the background service's wrapper engine observes screen patterns and local session records to report status on its behalf. This covers the rest of the built-in catalog, including **OpenCode**, **Devin CLI**, **Cursor Agent**, **Gemini CLI**, **GitHub Copilot CLI**, **Cline**, **Kimi**, **Kiro**, **Droid**, **Aider**, and **Goose**.

> **IMPORTANT**
>
> The wrapper engine only attaches to agents started from the new terminal flow or `v42 term new --agent` . If you type an agent's command manually inside an existing shell, only agents with native full OSC 7501 support will report status.

Default list and built-in catalog

When you open a new terminal in Vibe42, the third step presents **Terminal** (Just a shell) followed by your agent list. By default, the list includes nine agents, and the built-in catalog offers around 30 entries that you can add, remove, or reorder in **Agents** settings.

Agent	How to launch	Status source	Notes
Claude Code	New terminal flow, `v42 term new --agent`, or manual command in a Vibe42 pane (one split of a terminal window)	Native OSC 7501 (≥ 2.1.295); wrapper engine on older versions	Default list; full state, summary, and permission prompts
OpenAI Codex CLI	New terminal flow or `v42 term new --agent`	Partial self-reporting + wrapper engine	Default list
OMP	New terminal flow or `v42 term new --agent`	Partial self-reporting + wrapper engine	Default list
Pi Coding Agent	New terminal flow, `v42 term new --agent`, or manual command in a Vibe42 pane	Native OSC 7501 (≥ 1.1.0)	Default list
OpenCode	New terminal flow or `v42 term new --agent`	Wrapper engine	Default list
Grok Build	New terminal flow or `v42 term new --agent`	Partial self-reporting + wrapper engine	Default list
Devin CLI	New terminal flow or `v42 term new --agent`	Wrapper engine	Default list
Antigravity CLI	New terminal flow or `v42 term new --agent`	Partial self-reporting + wrapper engine	Default list
Cursor Agent	New terminal flow or `v42 term new --agent`	Wrapper engine	Default list
Crush, Amp	New terminal flow or `v42 term new --agent`	Partial self-reporting + wrapper engine	Available in built-in catalog
Gemini CLI, GitHub Copilot CLI, Cline, Kimi, Kiro, Droid, Aider, Goose, and others	New terminal flow or `v42 term new --agent`	Wrapper engine	Available in built-in catalog (~30 agents total)

Using an unlisted or custom agent

If you use a new CLI, a wrapper script, or an internal company agent that is not in the catalog, open **Agents** in settings and add a custom entry:

- Give it a display name (up to 64 characters) and a single-line launch command (`<program> <args>`).
- Leave the agent type set to **Detect from the command**, or pick a specific agent type if your script wraps a known tool from the catalog.
- If the tool is unrecognized and does not emit OSC 7501 itself, status tracking may be limited to **Working** and **Idle** (refer to what is shown in the app).

If you control the tool's source code or wrapper script, you can emit OSC 7501 escape sequences directly to get full **Working**, **Done**, **Needs input**, and **Error** tracking along with push notifications.

Nothing injected, nothing locked in

Vibe42 runs your coding agents in a clean login shell without injecting hooks, installing plugins, modifying configs, or proxying API keys.

vibe42.ai/docs/agents/zero-injection

Vibe42 runs your coding agents as ordinary programs on your computer, not as plugins hosted inside our service. Read this page to verify what Vibe42 touches on disk, where your API keys and source code go, and how status tracking works without modifying your agent setup.

What Vibe42 never does

When you install Vibe42 or launch an agent from the app, Vibe42 follows strict boundaries:

- **No hooks**: We do not write lifecycle hooks (scripts triggered on agent events) into your repositories or agent hook directories.
- **No skills or plugins**: We do not install custom skills, extensions, or helper packages into your agents.
- **No config edits**: We never modify `~/.claude` or any other agent's configuration files.
- **No API proxying**: Model requests go straight from your computer to your model provider. Your API keys, OAuth tokens, and source code stay on your machine, and you use your own agent subscriptions directly.

Your agent list in Vibe42 is simply a synced list of bookmarks for launch commands.

What this means in practice

- **Zero switching cost**: Try a new CLI (command-line tool) tomorrow or run three different agents side by side in the same Workspace. Nothing is left behind to uninstall or clean up if you stop using a tool.
- **Immunity to upstream breaking changes**: When an agent ships an update, changes its config schema, or adds new interactive flags, it works immediately in Vibe42 because you interact with the real terminal UI.
- **Clear security perimeter**: Vibe42 transports encrypted terminal streams and files between your paired devices and your computer (see Security). We never sit between your agent and the model API.

How status works without injection

Instead of patching your agent's configuration, Vibe42 gets agent status through two non-invasive paths:

1. **OSC 7501**: Agents with native support emit a standard terminal escape sequence (a short control code printed to the terminal)—just like setting a window title or printing ANSI colors. The Vibe42 background service parses and strips the sequence before forwarding terminal output.
2. **Wrapper engine**: For supported agents that do not emit full OSC 7501 yet, the background service wraps the process when launched from Vibe42 and infers status from screen patterns and the agent's existing session records.

Tradeoffs compared to other approaches

Remote agent tools generally choose one of three architectures:

Approach	How it integrates	Tradeoff
Hook or plugin injection	Writes scripts into the agent's config directory to post webhooks on events	Breaks when config formats change, leaves files behind, and only works with agents that expose a hook API
Provider or chat wrapper	Wraps the agent SDK or API behind a custom chat UI	Loses native terminal features, slash commands, and keyboard workflows; often requires pasting API keys into a third-party service
Terminal-native (Vibe42)	Runs the unmodified CLI in tmux; reads OSC 7501 or observes the pane (one split of a terminal window) externally	Keeps 100% of the native CLI and leaves disk configs untouched, while still delivering unified status and push notifications

The single point of contact—and how to bypass it

The only contact Vibe42 has with an agent process is the wrapper engine: when you start an agent from the new terminal flow or `v42 term new --agent`, the background service observes that pane's screen output so it can report **Working**, **Done**, **Needs input**, or **Error** on the agent's behalf (agents with native full OSC 7501 support skip the wrapper engine entirely).

If you prefer not to use the wrapper engine at all, choose **Terminal** (Just a shell) when opening a window and type the agent command yourself. Vibe42 never wraps a command you run manually inside a shell.

Launching an agent

Start coding agents from the new terminal flow, context menus, or the v42 CLI, and customize your synced agent list.

`vibe42.ai/docs/agents/launching-agents`

Launching an agent through Vibe42 starts it inside a persistent window that keeps running on your computer and connects status tracking and push notifications automatically. Read this page to learn every entry point for starting an agent, how to customize your agent list and launch commands, and how login shells and isolated Workspaces behave.

The three-step flow and quick entry points

The full new terminal flow picks a computer, chooses a Workspace or directory, and lands on step three: choosing what to start.

- The first item is always **Terminal** (Just a shell).
- Below it is your agent list, with your last selection pre-selected by default—press Enter to launch immediately.
- From this picker, you can type to search, go back to adjust the location, or click **Manage agents…** to edit your list.

When the target location is already known (or when your account has only one computer so computer selection is skipped), Vibe42 jumps straight to the launch picker from contextual shortcuts:

- Workspace menu: **New terminal in this workspace**
- File tree context menu: **New terminal here**
- Window menu: **New window here**
- Global search and the **+** button in the terminal list
- Workspace creation dialog: **Open a terminal in this workspace after creation**

> NOTE
> The Vibe42 background service on your computer executes the launch itself. If your phone loses signal or you close your browser right after tapping launch, the agent still starts cleanly.

Managing your agent list

Open settings and go to **Agents** to customize what appears in the launch picker. Your default list starts with Claude Code, OpenAI Codex CLI, OMP, Pi Coding Agent, OpenCode, Grok Build, Devin CLI, Antigravity CLI, and Cursor Agent, while the built-in catalog includes around 30 agents.

In **Agents** settings, you can:

- Add agents from the built-in catalog or remove ones you do not use
- Drag and drop entries to reorder them
- Add custom entries with your own display name and command
- Restore the default list at any time

Both your customized agent list and your last-selected entry sync across all devices signed in to your account.

Custom commands and rules

When adding or editing a custom agent entry in **Agents** settings, configure three fields:

- **Name**: Up to 64 characters. New windows launched with this entry take the agent's name as their window title.
- **Agent type**: Defaults to **Detect from the command**, or choose a specific agent type so the wrapper engine applies matching status rules.
- **Command**: A single-line `<program> <args>` command where arguments are passed verbatim to the program:

```
<program> <args>
```

Launch commands must follow these constraints:

- Single line, 1024 bytes or fewer
- Must not start with `-` and must not contain control characters
- Direct program plus arguments only—shell operators such as `&&`, `;`, redirection, or pipes (`|`) are not supported

If you need multi-step setup or environment exports before starting an agent, put those steps in an executable shell script on your computer and set the script path as your command.

Login shell, window lifecycle, and isolation

Every agent command runs inside your interactive login shell (`<shell> -l -i -c`, the same shell environment you get when opening a local terminal). Tools and environment variables configured in your shell startup files—such as `PATH` additions, `nvm`, `pyenv`, or credential helpers—work the same way they do locally.

When the agent process exits, the window stays open and drops back to a regular interactive shell so you can inspect `git diff` or run tests.

If your computer restarts, the background service recreates the window and runs the same launch command again the next time it starts. Supported agents reopen their last conversation (any turn interrupted by the restart needs you to send another message to continue), whereas commands you typed manually into a shell are not re-run—see Sessions.

If you launch an agent inside a Workspace with terminal isolation enabled, Vibe42 creates a fresh isolated copy first and starts the agent inside that copy. See Terminal isolation for details.

Launching from the CLI

You can also create windows and launch agents from scripts with the `v42` CLI (see CLI commands):

```
v42 term new <computer> [--workspace <ws>] [--cwd <dir>] [--name <n>] [--agent <id>] [--command "
<cmd>"] [--attach]
```

Pass `--agent <id>` when you want the background service's wrapper engine to track agent status. The id is the catalog id of the agent—for example `claude-code`, `codex`, `gemini`, `opencode`, `cursor`, or `copilot`. An unknown id fails with a usage error that lists every valid id.

If a computer needs an update

If the target computer is running an older version of Vibe42 that predates agent launching, the picker only shows **Terminal** with the message:

```
Update Vibe42 on this computer to start agents from here.
```

From the CLI, `v42 term new` returns `launch_unsupported` in this case. Update Vibe42 on that computer to enable agent launching (see Updates and uninstall).

Agent status

Understand the five agent states, sidebar indicators, sorting rules, status sources, and why status might not appear.

vibe42.ai/docs/agents/agent-status

When you run multiple coding agents across Workspaces and computers, Vibe42 groups their activity into five states so you can tell at a glance what is still running and what needs your input. Read this page to learn what each state means, how the app sorts and resets them, and why status might not show up in certain panes (splits of a terminal window).

States and sidebar indicators

Any pane with active agent status displays a robot icon next to its entry in the sidebar along with a colored status dot.

Status	Sidebar indicator	Meaning
Idle	No dot	The agent is waiting at a prompt with no unread result, or you have already viewed its previous **Done** state.
Working	Breathing dot	The agent is actively reasoning, editing files, or running tools. Shows `{{progress}}%` when progress is reported.
Done	Green dot	The agent finished its turn and has new output you have not viewed yet.
Needs input	Orange dot	The agent paused and cannot continue without you. Shows `{{progress}}%` if reported, and specifies one of three reasons below.
Error	Red dot	The agent stopped due to an error.

When an agent enters **Needs input**, Vibe42 displays a more specific label when available:

- **Needs permission**: Waiting for you to approve a tool call, command execution, or file edit.
- **Needs an answer**: Asking a clarifying question or waiting for you to choose an option.
- **Needs sign-in**: Waiting for authentication or login before it can proceed.

Sorting in the Agents tab and home view

Vibe42 shows active agents in two dedicated views across all your connected devices:

- **Agents** tab: Groups agents by Workspace or computer. Within each group, entries sort by urgency—**Error** > **Needs input** > **Done** > **Working** > **Idle**—and then by update time.
- **Home screen** (Current Agents): Segments all tracked agents by state so blocked or finished tasks stand out immediately.

If a single pane tracks multiple sub-tasks via OSC 7501 (a terminal control code for reporting status), the pane's summary status follows the same priority order: **Needs input** (`blocked`) > **Error** > **Working** > **Done** > **Idle**.

How Done resets to Idle

A green **Done** dot works like an unread badge:

- When a background pane finishes work, it stays **Done** until you open that pane.
- As soon as you open the pane, its outward status changes from **Done** to **Idle** ("seen").
- If the pane is already open in front of you when the agent finishes, a newly reported **Done** state does **not** auto-reset to **Idle**, so you can still see that the turn completed.

Because status is maintained by the Vibe42 background service on your computer and syncs across your connected devices, viewing a **Done** pane on your phone clears the green dot on your desktop app and web app as well.

Where status comes from

Depending on the tool, status arrives through one of three tiers (see Supported coding agents):

1. **Native full OSC 7501**: Claude Code (≥ 2.1.295) and Pi (≥ 1.1.0) report states, summaries, and **Needs input** kinds directly.
2. **Partial self-reporting + wrapper engine**: Codex, OMP, Crush, Amp, Antigravity, and Grok Build report partial signals while the background service's wrapper engine fills in the rest.
3. **Wrapper engine**: For other catalog agents, the background service infers status from screen rules and local session records.

Why status might not appear

If an agent is running in a terminal but shows no robot icon or status dot, check these four cases:

- **Started manually in a shell**: The wrapper engine only wraps agents started via the new terminal flow or `v42 term new --agent`. Manually typed commands do not show status unless the agent natively supports full OSC 7501.
- **Terminals opened outside Vibe42**: Only panes created by Vibe42 receive a reply from the background service when a tool probes for OSC 7501 (`ESC ] 7501 ; ? ST`). Terminals or splits you opened outside Vibe42 stay silent.
- **Pane created before a Vibe42 update**: Panes that already existed before you updated Vibe42 do not answer OSC 7501 probes. Open a new window in Vibe42 to enable tracking.
- **Slow startup (> 20 seconds) or exited process**: If no recognized agent process appears within 20 seconds of launch, the wrapper engine stops waiting on that pane. Once an agent exits back to your shell, its status records are cleared.

Notifications

Get notified on phone, desktop, web, or Telegram when an agent needs input or finishes, and see how

alert text reaches Apple and Google via our servers.

vibe42.ai/docs/agents/notifications

Vibe42 notifies you when a coding agent needs your decision or finishes a task, as well as when terminal or computer events occur, so you can step away from your desk without checking the screen. Read this page to configure delivery channels, per-computer rules, Telegram linking, and deduplication behavior.

Events that trigger notifications

Notifications are organized into three categories, and each event type can be toggled individually per computer.

Category	Event types	When it fires
Agent events	**Agent needs input**, **Agent finished**, **Agent error**	Enabled by default. Fires once each time a pane's (a terminal split's) summary agent status transitions into **Needs input**, **Done**, or **Error**.
Terminal events	Terminal bell, Terminal notification	Fires when a program emits a standard BEL character (terminal bell) or terminal notification escape sequence.
Session, device, and computer events	Window or pane closed, device disconnected, session created or closed, computer paired / unpaired / online / offline, high-frequency folding summary	Fires when workspace layout, connectivity, or pairing state changes, or when burst protection folds notifications.

For agent events, the notification title follows the pattern `<Status>: <Title>` (for example, `Done: Fix login bug`). The body includes the status line and turn summary (`msg`), followed by three context lines at the bottom: the Workspace name (or working directory if outside a Workspace), the computer name, and the timestamp.

Delivery channels across devices

- **Mobile push (iPhone, iPad, Android)**: The mobile app does not stay running in the background or keep a persistent connection open, so it does not drain your battery. Notifications arrive through system push services (APNs on iOS and FCM on Android). Tapping a notification opens Vibe42 directly to the exact window on that computer.
- **Desktop app (macOS, Linux)**: Delivers native system notifications while the desktop app is running. Events that occur while the desktop app is quit are not replayed on next launch.
- **Web app**: Displays in-app alerts while `app.vibe42.ai` is open in your browser.
- **Telegram**: Open settings and go to **Notifications → Message Channels** to link your Telegram account via a QR code or pairing code. You can unlink it from the same screen at any time.

Per-computer notification settings

Every computer has its own notification configuration, letting you stay alerted on your primary workstation while keeping a noisy build server quiet:

- **Notifications for this computer**: Enable or pause all notifications from that computer.

- **Deliver to**: Choose independently whether events from that computer go to web and desktop apps, Telegram, and mobile push.
- **Event types**: Enable or disable specific agent, terminal, session, and computer events.

Summary waiting, deduplication, and burst folding

Vibe42 filters noisy terminal output before delivering alerts:

- **Up to 15-second summary window**: When an agent transitions into a notify-worthy state, Vibe42 waits up to 15 seconds for the turn summary (`msg`) to arrive before sending, so your notification includes useful context rather than a blank body.
- **3-second bell deduplication**: Many agents ring the terminal bell right after reporting status. Any terminal bell or terminal notification from the same pane within 3 seconds of an OSC 7501 status update is suppressed automatically.
- **Rate-limit folding**: When notifications exceed roughly 20 per minute, Vibe42 folds them into a periodic summary instead of flooding your lock screen.

How notification text reaches your phone

Notification titles and bodies are the only content in Vibe42 that passes through and is stored on our servers in readable form. Here is how a notification moves from your computer to your phone, step by step:

1. **Evaluated and generated on the controlled computer**: Events that trigger notifications happen on the controlled computer: agent status changes (waiting for confirmation, finished, or error), terminal bells, terminal notifications, window or pane closures, device disconnections, session creation or closure, and watch rule matches. The background service on that computer checks the computer's notification settings first: if the event type is disabled, or if a terminal bell or notification hits throttling, the event is dropped on the computer immediately—no text is generated and nothing is sent. Only events that pass this check are rendered into a notification title and body in your account's interface language, so notifications arrive already in your language.
2. **Sent from the computer to Vibe42 servers (encrypted, but not end-to-end)**: The rendered title and body travel upstream over the encrypted control session between the background service and Vibe42's control servers. That session is forwarded through an AP (access point), but its endpoint is a Vibe42 server. Because Vibe42 servers must read the text to hand it off to Apple, Google, or Telegram, notification titles and bodies are **not** end-to-end encrypted—unlike terminal and file traffic, which only your devices and computers can decrypt. If the computer is offline or unpaired when an event occurs, the event is dropped immediately and is not queued for later delivery.
3. **Processed by Vibe42 servers**:
    - Replaces the trailing working directory line with the Workspace name (if the terminal belongs to a Workspace) and replaces the computer name with the name you gave that computer in Vibe42.
    - Saves a notification record (title, body, event type, computer, timestamp, and delivery outcome) as your notification history, which lets you view recent notifications in the app. This record is stored on Vibe42 servers and deleted along with your account if you delete the account.
    - Creates an individual delivery job for each destination: each signed-in mobile phone or tablet (each iPhone, iPad, and Android device counts separately), each linked Telegram chat, and the web/desktop channel. Delivery is attempted immediately; if it fails, the server retries at increasing intervals up to 6

times and expires the job if still undelivered after 24 hours. If the computer is deleted or that phone signs out during this window, delivery is canceled.
    - Truncates push titles beyond 200 characters and bodies beyond 3,500 characters.
    - Server logs do not write notification titles or bodies; audit logs record only identity, routing, and management metadata—**notification bodies are never recorded in audit logs** (see Security).
4. **Sent from Vibe42 servers to Apple and Google**:
    - **iPhone and iPad**: Vibe42 servers send an HTTPS request to Apple's push service APNs (`api.push.apple.com`), signed with Vibe42's APNs authentication key. The request contains your device's push token (the device token Apple issued to the Vibe42 app on your phone and registered with Vibe42 at sign-in; it is not your Apple ID and does not contain your email address or phone number), the title and body (a standard `alert` notification with default sound), and a jump target `vibexTarget` (computer ID, window ID, pane ID, and event type, used only to open the right window when tapped).
    - **Android**: Vibe42 servers send an HTTPS request to Google's Firebase Cloud Messaging (FCM, `fcm.googleapis.com`), authenticated with Vibe42's service account. The request contains your device's registration token, the title and body (shown in Vibe42's event notification channel on Android), and the same `vibexTarget`.
    - **What Apple and Google can read**: Both requests use TLS-encrypted HTTPS, but **the text inside the request body is plaintext**: Apple's and Google's servers can read the notification title and body before forwarding them to your phone. Vibe42 does not add an extra encryption layer that only the phone can decrypt (the iOS app does not use a notification service extension for decryption, and Android displays the delivered content directly). Apple and Google receive only the push token, title, body, and `vibexTarget`—no account information and no terminal or file content. How they handle the payload afterward is governed by their respective push service policies.
5. **Displayed on your phone**: Your phone's operating system displays the notification from the content delivered by Apple or Google; the Vibe42 app does not need to be running or keep a background connection. When you tap the notification, the app reads `vibexTarget` to open the corresponding computer and window.

What is in a notification—and what is not

- **What the title and body contain**:
    - Agent events: the title is "Status: Session title"; the body is the status line, the turn summary (`msg`), or the tool name and reason when waiting for confirmation.
    - Terminal notifications: the title and message text provided by the program itself. Terminal bells: the terminal title, current command, or a location such as "Window N Pane M", with the event name as the body.
    - Window or pane closed: the name of what was closed. Watch rules: the rule name and the summary or matched text produced by the rule.
    - Three trailing lines: the terminal's working directory, the computer name, and the time of the event.
- **What the title and body do not contain**: any other screen output, command history, keystrokes, or file contents. Only the few lines of text that a program explicitly puts into a status report or terminal notification are included.

Telegram, web, and desktop apps

- **Telegram**: Vibe42 servers send the title and body over HTTPS via Telegram's Bot API (`api.telegram.org`) to your linked chat; Telegram's servers can also read the text.
- **Web and desktop apps**: Delivered over your signed-in connection to Vibe42 servers only while online, without passing through Apple, Google, or Telegram; however, the text still passes through and is stored on Vibe42 servers (in your notification history).

When turned off, no party receives notification content

- Turning off **Notifications for this computer** in that computer's notification settings, or turning off a specific event category under **Event types**: the decision happens locally on the controlled computer, and disabled events are dropped on the computer immediately—**no notification text is generated, nothing is sent to Vibe42 servers, nothing reaches Apple, Google, Telegram, or any of your devices, and nothing appears in your notification history**. No party receives any notification content.
- Turning off only mobile push under **Deliver to** (while keeping web/desktop or Telegram enabled): the text is still sent to Vibe42 servers and saved in your notification history, but is not sent to Apple or Google; turning off Telegram works the same way and does not send to Telegram.
- These settings are stored in your account and synced to that computer, which evaluates events according to the synced settings.

Troubleshooting

If alerts are not arriving on your phone or desktop, see Notifications troubleshooting to check OS permissions, per-computer **Deliver to** toggles, and pane launch conditions.

OSC 7501 status protocol

Full specification for the OSC 7501 terminal escape sequence used by coding agents and custom scripts to report status in Vibe42.

`vibe42.ai/docs/agents/osc-7501`

OSC 7501 is the terminal escape sequence protocol (a short control code printed to the terminal) that connects CLI programs to Vibe42's **Agents** tab, sidebar indicators, and push notifications. Read this specification if you build a coding agent, maintain an internal CLI, or want your own long-running shell scripts to report progress and request user input in Vibe42.

Syntax and wire format

An OSC 7501 sequence consists of colon-separated `key=value` pairs terminated by `BEL` (`0x07`) or `ST` (`ESC \`):

```
ESC ] 7501 ; key=value:key=value ... (BEL | ESC \)
state=<idle|working|done|blocked|error|clear>   required; entire sequence dropped if invalid
id=<seg/seg/...>   record path, each segment [A-Za-z0-9_.+-], <=8 levels, <=128 bytes; defaults
to root record
```

```
kind=<permission|question|auth>   valid only when state=blocked
progress=<0-100>   valid only when state=working or state=blocked
app=<printable ASCII, <=32 bytes>   agent identifier, e.g. claude
title=<base64(UTF-8)>   decoded <=192 bytes, no control characters
msg=<base64(UTF-8)>   decoded <=2048 bytes, turn summary
Probe: ESC ] 7501 ; ? ST   program starts reporting only after terminal replies
```

Parser rules and limits:

- **Maximum sequence length**: 4096 bytes total.
- **Keys**: Lowercase ASCII only. Unknown keys are ignored; an individual malformed `key=value` pair is skipped, while a missing or invalid `state` drops the entire sequence.
- **Stripping**: The Vibe42 background service strips OSC 7501 sequences from the stream before forwarding, so clients never see raw escape bytes.
- **No OSC 9;4 mapping**: Standard OSC `9;4` terminal progress sequences are not mapped to OSC 7501.

Field reference

Field	Required	Constraints and behavior
`state`	Yes	One of `idle`, `working`, `done`, `blocked`, `error`, or `clear`. `blocked` renders as **Needs input** in the UI.
`id`	No	Slash-delimited path (`seg/seg/...`) using `[A-Za-z0-9_.+-]`, up to 8 levels and 128 bytes. Omitting `id` targets the root record.
`kind`	No	Only honored when `state=blocked`. Maps `permission` → **Needs permission**, `question` → **Needs an answer**, `auth` → **Needs sign-in**.
`progress`	No	Integer `0` – `100`. Only honored when `state=working` or `state=blocked`; displayed as `{{progress}}%`.
`app`	No	Printable ASCII identifier up to 32 bytes (for example, `claude`).
`title`	No	Base64-encoded UTF-8 string. Decoded length ≤ 192 bytes, containing no control characters.
`msg`	No	Base64-encoded UTF-8 summary for the current turn. Decoded length ≤ 2048 bytes.

Probe handshake

Before emitting status updates, tools probe for support by sending:

```
ESC ] 7501 ; ? ST
```

Wait for the terminal's reply before enabling OSC 7501 reporting. Raw tmux does not answer this probe on its own. Inside panes (terminal splits) created by Vibe42, the background service intercepts and replies to the probe (and the wrapper engine replies directly for wrapped processes).

> **NOTE**
>
> Panes split manually inside tmux and panes that existed before updating Vibe42 do not receive a probe reply, so probing tools stay quiet automatically.

Per-pane record table and lifecycle

Each pane maintains its own record table holding up to **256 records**, allowing a parent agent and sub-tasks to report concurrently under hierarchical `id` paths:

- **Full replacement**: Updating an existing `id` replaces that record completely rather than merging omitted fields.
- **Clearing records**: Sending `state=clear` with an `id` deletes that record and all descendant `id/*` records. Sending `state=clear` on the root record clears the entire table.
- **Terminal reset and prompt hooks**: A full terminal reset (RIS) clears the table. Receiving an `OSC 133 ; A` shell prompt marker drops all `working` and `blocked` records so crashed tools do not stay stuck as running.

Aggregation, Done reset, and push rules

- **Summary priority**: Across all records in a pane, Vibe42 picks the highest-priority state in the order `blocked > error > working > done > idle`. Ties prefer the root record first, then the most recently updated record.
- **Done-to-Idle reset**: Once a user opens a pane whose summary state is `done`, its outward state becomes `idle` ("seen"). If the pane is already open while a new `done` arrives, it stays `done`.
- **Notification trigger**: Push notifications fire when the pane's **summary state transitions** into `done`, `blocked`, or `error`. Vibe42 waits up to 15 seconds for `msg` and deduplicates terminal bells on the same pane for 3 seconds after an update.

Shell examples

Reporting `working` with progress, then transitioning to `done`:

```
printf '\e]7501;state=working:app=demo:progress=30:msg=%s\e\\' "$(printf 'Running tests' |
base64)"

# ... run build or test steps ...

printf '\e]7501;state=done:app=demo:title=%s:msg=%s\e\\' \
  "$(printf 'CI check' | base64)" \
  "$(printf 'All suites passed' | base64)"
```

Reporting `blocked` while waiting for user permission:

```
printf '\e]7501;state=blocked:kind=permission:app=demo:title=%s:msg=%s\e\\' \
  "$(printf 'Database migration' | base64)" \
  "$(printf 'Approve schema changes?' | base64)"
```

Let your agent use Vibe42

Give your coding agent install.md to set up Vibe42, automate tasks with the JSON-first v42 CLI, or emit OSC 7501 status.

`vibe42.ai/docs/agents/for-agents`

Vibe42 is built not only for humans controlling agents from a phone or laptop, but also for coding agents operating Vibe42 directly. Read this page when you want an agent to install and pair Vibe42 for you, script workspace workflows with the `v42` CLI (command-line tool), work safely inside isolated copies, or read Vibe42 documentation in plain Markdown.

1. Hand off installation with install.md

If you already have a coding agent running on a Mac or Linux machine, point it to `https://vibe42.ai/install.md`:

```
Read https://vibe42.ai/install.md and install Vibe42 on this machine.
```

`install.md` gives the agent step-by-step instructions to inspect the computer environment (macOS, Linux desktop, headless Linux server, or container), pick the right installation method, run setup, and show the 8-character pairing code or approval URL back to you.

It also enforces strict safety rules on the agent:

- **Never touch passwords**: The agent never asks for your Vibe42 password, two-factor authentication code, or session cookies. Pairing always completes through your own approval on an already signed-in device or in Console.
- **Pair one machine at a time**: The agent installs and waits for your approval on the current computer rather than batch-scripting credentials across machines.

For a step-by-step walkthrough, see Let your agent install Vibe42.

2. Automate workflows with the v42 CLI

The `v42` CLI is designed for non-interactive automation by default (see CLI overview and CLI automation):

- **JSON output by default**: Commands emit structured JSON so agents can parse IDs, window names, and network status reliably without scraping human-formatted tables.
- **Stable exit codes**: Predictable exit codes distinguish temporary connection issues from permission, slot, or argument errors.
- **Audit attribution with `V42_AGENT_NAME`**: When an agent runs `v42`, set `V42_AGENT_NAME` in its environment so management audit records clearly show which agent performed the action:

```
V42_AGENT_NAME=claude v42 term new my-mac --workspace api --agent claude-code
```

Teach agents about isolated copies

When you run multiple agents in parallel using Terminal isolation, each terminal works in its own copy of the Workspace. Run `v42 copy skill` to print a ready-to-use `SKILL.md` document that teaches your coding agent how isolated copies work and how to inspect and merge changes cleanly:

```
v42 copy skill
```

3. Report status from custom tools via OSC 7501

When you ask an agent to build a long-running test runner, deployment script, or internal tool, ask it to emit OSC 7501 terminal escape sequences (short control codes printed to the terminal to report status). Any script that probes and writes OSC 7501 automatically appears in Vibe42's **Agents** tab with progress percentages and triggers mobile notifications when it finishes, fails, or pauses for user confirmation.

Machine-readable documentation

Vibe42 publishes an `llms.txt` index on the documentation site, and every documentation page is available in clean `.md` format. You can feed documentation pages directly to your coding agent without wasting context tokens on HTML page markup.

Claude Code in Vibe42

Launch Anthropic's Claude Code in Vibe42 with native OSC 7501 status reporting, push notifications, and mobile terminal controls.

`vibe42.ai/docs/agents/claude-code`

Claude Code is Anthropic's coding agent that runs in your terminal. Read this page to learn how to launch Claude Code in Vibe42, see its live status, and continue your sessions from your phone.

Launching in Vibe42

Claude Code is the first option in Vibe42's default agent list. Select **Claude Code** in step three of the new terminal flow, or launch it from the command line with `v42 term new <computer> --agent claude-code`.

Vibe42 runs your locally installed binary inside an interactive login shell (`<shell> -l -i -c`) with zero injection—we never touch `~/.claude`, install hooks, or proxy your Anthropic credentials. To pass custom startup flags or run a wrapper script, add an entry in **Agents** settings using `<program>` and your arguments on a single line. When Claude Code exits, the window stays open at your shell prompt.

Status reporting

Claude Code 2.1.295 and newer natively supports OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42) without using the Vibe42 background service's wrapper engine. Older versions fall back to the wrapper engine automatically.

In the sidebar and **Agents** tab, you see all five agent states—**Idle**, **Working**, **Done**, **Needs input**, and **Error**—along with turn summaries. Because reporting is native, Claude Code (≥ 2.1.295) even reports status when you type its command manually inside any pane (one split of a terminal window) created by Vibe42. When it pauses for **Needs input**, Vibe42 shows the exact reason:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before it can continue.

Notifications

Whenever a pane transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification with the status and turn summary to your paired phone (via APNs/FCM), running desktop app, web app, or linked Telegram. Tap the alert to jump straight to that Claude Code window.

Continuing from your phone

Because the session keeps running on your computer even when you close the app, you can start a task at your desk and follow up from iPhone, iPad, or Android (see Start Claude Code before you leave and Mobile input):

- Answer permission prompts or multiple-choice questions with the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Draft multi-line instructions in the command bar (**Send to terminal**) before sending.
- Take a photo or pick a screenshot from your phone to upload it and insert its local file path into the prompt.

Known limitations

- Claude Code must already be installed and signed in on your computer; Vibe42 connects you to the real terminal UI rather than a separate chat wrapper.
- Terminals you opened outside Vibe42 and panes created before updating Vibe42 do not answer OSC 7501 probes, so status does not appear in those panes.
- On Claude Code versions older than 2.1.295, status relies on the wrapper engine and only appears when launched from the new terminal flow or `v42 term new --agent`.

OpenAI Codex CLI in Vibe42

Run OpenAI Codex CLI in Vibe42 with hybrid status tracking, mobile push notifications, and persistent terminal sessions.

`vibe42.ai/docs/agents/codex`

OpenAI Codex CLI is OpenAI's coding agent that runs directly in your terminal. Read this page to learn how to launch OpenAI Codex CLI in Vibe42, how hybrid status reporting works, and how to supervise tasks from your phone.

Launching in Vibe42

OpenAI Codex CLI is included in Vibe42's default agent list. Pick **OpenAI Codex CLI** in step three of the new terminal flow, or launch it from scripts with `v42 term new <computer> --agent codex`.

Vibe42 runs your local installation inside an interactive login shell (`<shell> -l -i -c`) with zero injection—your OpenAI credentials, config files, and API traffic stay on your computer. To launch via a wrapper script or pass custom startup arguments, create or edit an entry in **Agents** settings using a single-line `<program>` command. If terminal isolation is enabled on the Workspace, the session starts in a fresh isolated copy. When the agent exits, the window returns to your shell.

Status reporting

OpenAI Codex CLI uses **partial self-reporting plus the wrapper engine**: the CLI emits partial status signals on its own, and the Vibe42 background service's wrapper engine fills in the remaining transitions from screen patterns and session records via OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42).

In the sidebar and **Agents** tab, Vibe42 tracks agent status across **Idle**, **Working**, **Done**, **Needs input**, and **Error**. When the agent pauses for **Needs input**, Vibe42 distinguishes between:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Each time a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone (APNs/FCM with zero background battery drain), running desktop app, web app, or linked Telegram. Tap the alert to open that window on your computer.

Continuing from your phone

Because sessions keep running on your computer even when you close the app, you can hand off work to Codex at your desk and check back from iPhone, iPad, or Android (see Mobile input):

- Approve prompts and navigate terminal menus using the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Compose multi-line follow-up prompts in the command bar (**Send to terminal**) before sending them to the pane.
- Upload a photo or screenshot from your phone to insert its local file path into the terminal.

Known limitations

- OpenAI Codex CLI must already be installed and signed in on your computer; Vibe42 renders the real terminal UI rather than a separate chat view.
- Launch through the new terminal flow or `v42 term new --agent` so the wrapper engine is attached. Typing the command manually in an existing shell skips wrapper supplementation, and terminals you opened outside Vibe42 or panes left over from before updating Vibe42 do not answer OSC 7501 probes.

- If the agent process does not appear within 20 seconds of launch, the wrapper engine stops trying to wrap that pane.

OMP in Vibe42

Run OMP in Vibe42 with hybrid status tracking, mobile push notifications, and persistent terminal sessions.

`vibe42.ai/docs/agents/omp`

oh-my-pi (**OMP**) is an MIT-licensed open-source terminal coding agent forked from Mario Zechner's Pi—described as a "coding agent with the IDE wired in" (bringing IDE capabilities such as LSP and debuggers into the terminal agent). Read this page to learn how to launch OMP in Vibe42, how hybrid status reporting works, and how to supervise tasks from your phone.

Launching in Vibe42

OMP is included in Vibe42's default agent list. Pick **OMP** in step three of the new terminal flow (which runs `omp`), or launch it from scripts with `v42 term new <computer> --agent omp`.

Vibe42 runs your local installation inside an interactive login shell (`<shell> -l -i -c`) with zero injection—your credentials, config files, and API traffic stay on your computer. To launch via a wrapper script or pass custom startup arguments, create or edit an entry in **Agents** settings using a single-line `<program>` command. If terminal isolation is enabled on the Workspace, the session starts in a fresh isolated copy. When the agent exits, the window returns to your shell.

Status reporting

OMP uses **partial self-reporting plus the wrapper engine**: the CLI emits partial status signals on its own, and the Vibe42 background service's wrapper engine fills in the remaining transitions from screen patterns and session records via OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42).

In the sidebar and **Agents** tab, Vibe42 tracks agent status across **Idle**, **Working**, **Done**, **Needs input**, and **Error**. When the agent pauses for **Needs input**, Vibe42 distinguishes between:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Each time a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone (APNs/FCM with zero background battery drain), running desktop app, web app, or linked Telegram. Tap the alert to open that window on your computer.

Continuing from your phone

Because sessions keep running on your computer even when you close the app, you can hand off work to OMP at your desk and check back from iPhone, iPad, or Android (see Mobile input):

- Approve prompts and navigate terminal menus using the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Compose multi-line follow-up prompts in the command bar (**Send to terminal**) before sending them to the pane.
- Upload a photo or screenshot from your phone to insert its local file path into the terminal.

Known limitations

- OMP must already be installed and configured on your computer; Vibe42 renders the real terminal UI rather than a separate chat view.
- Launch through the new terminal flow or `v42 term new --agent` so the wrapper engine is attached. Typing `omp` manually in an existing shell skips wrapper supplementation, and terminals you opened outside Vibe42 or panes left over from before updating Vibe42 do not answer OSC 7501 probes.
- If the agent process does not appear within 20 seconds of launch, the wrapper engine stops trying to wrap that pane.

Pi Coding Agent in Vibe42

Launch Pi Coding Agent in Vibe42 with native OSC 7501 status reporting, push notifications, and mobile terminal controls.

`vibe42.ai/docs/agents/pi`

Pi (**Pi Coding Agent**) is an open-source terminal coding agent developed by Mario Zechner—described as "a minimal agent harness" designed to stay minimal and extensible for your workflow. Read this page to learn how to launch Pi in Vibe42, see its live status, and continue your sessions from your phone.

Launching in Vibe42

Pi Coding Agent is included in Vibe42's default agent list. Select **Pi Coding Agent** in step three of the new terminal flow (which runs `pi`), or launch it from the command line with `v42 term new <computer> --agent pi`.

Vibe42 runs your locally installed binary inside an interactive login shell (`<shell> -l -i -c`) with zero injection—we never touch its configuration files, install hooks, or proxy your API requests or credentials. To pass custom startup flags or run a wrapper script, add an entry in **Agents** settings using `<program>` and your arguments on a single line. When Pi exits, the window stays open at your shell prompt.

Status reporting

Pi 1.1.0 and newer natively supports OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42), outputting full status, progress, turn summaries, and pause reasons on its own without using the Vibe42 background service's wrapper engine. Older versions fall back to the wrapper engine automatically.

In the sidebar and **Agents** tab, you see all five agent states—**Idle**, **Working**, **Done**, **Needs input**, and **Error**—along with turn summaries. Because reporting is native, Pi (≥ 1.1.0) even reports status when you type `pi` manually inside any pane (one split of a terminal window) created by Vibe42. When it pauses for **Needs input**, Vibe42 shows the exact reason:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before it can continue.

Notifications

Whenever a pane transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification with the status and turn summary to your paired phone (via APNs/FCM), running desktop app, web app, or linked Telegram. Tap the alert to jump straight to that Pi window.

Continuing from your phone

Because the session keeps running on your computer even when you close the app, you can start a task at your desk and follow up from iPhone, iPad, or Android (see Mobile input):

- Answer permission prompts or multiple-choice questions with the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Draft multi-line instructions in the command bar (**Send to terminal**) before sending.
- Take a photo or pick a screenshot from your phone to upload it and insert its local file path into the prompt.

Known limitations

- Pi Coding Agent must already be installed and configured on your computer; Vibe42 connects you to the real terminal UI rather than a separate chat wrapper.
- Terminals you opened outside Vibe42 and panes created before updating Vibe42 do not answer OSC 7501 probes, so status does not appear in those panes.
- On Pi versions older than 1.1.0, status relies on the wrapper engine and only appears when launched from the new terminal flow or `v42 term new --agent`.

OpenCode in Vibe42

Run OpenCode in Vibe42 with wrapper-engine status tracking, push notifications, and mobile terminal

navigation.

vibe42.ai/docs/agents/opencode

OpenCode is an open-source coding agent built for the terminal. Read this page to learn how to start OpenCode in Vibe42, how status tracking and notifications work through the wrapper engine, and how to control its terminal interface from your phone.

Launching in Vibe42

OpenCode is preconfigured in Vibe42's default agent list. Choose **OpenCode** in step three of the new terminal flow, or start a window from the command line with `v42 term new <computer> --agent opencode`.

Vibe42 runs your installed binary inside an interactive login shell (`<shell> -l -i -c`) with zero injection: we never alter OpenCode's configuration files, inject plugins, or route your model provider keys through our servers. To pass custom arguments or use a wrapper script, edit or create an entry in **Agents** settings with a single-line `<program>` command. Inside a Workspace with terminal isolation enabled, OpenCode launches inside a dedicated isolated copy. When you quit OpenCode, the window stays open at your shell prompt.

Status reporting

OpenCode status is reported by the **Vibe42 background service's wrapper engine**. When started through Vibe42, the wrapper engine inspects screen patterns and local session records to report agent status across your devices.

In the sidebar and **Agents** tab, you see **Idle**, **Working**, **Done**, **Needs input**, and **Error** indicators (specific sub-state display depends on current screen rules; refer to what is shown in the app). Under **Needs input**, Vibe42 maps prompts to three standard categories:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Whenever a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone via APNs or FCM, running desktop app, web app, or linked Telegram. Tap the notification to jump directly to that OpenCode window.

Continuing from your phone

Because OpenCode keeps running on your computer even when you close the app, you can pick up the full TUI (an interactive interface drawn inside the terminal) from iPhone, iPad, or Android without interrupting an in-progress run (see Mobile input):

- Navigate TUI dialogs and confirm actions with the draggable D-pad, modifier keys, Escape, and custom shortcut buttons.
- Prepare multi-line instructions in the command bar (**Send to terminal**) and send them in one tap.
- Upload images or files from your phone to insert their local path on your computer into the prompt.

Known limitations

- OpenCode must already be installed and configured with your chosen model provider on your computer; Vibe42 displays the real terminal UI rather than a separate chat interface.
- Status tracking and agent push notifications require launching OpenCode from the new terminal flow or `v42 term new --agent`. Starting OpenCode manually inside a regular shell skips the wrapper engine.
- If the OpenCode process does not appear within 20 seconds of launch, the wrapper engine stops wrapping that pane.

Grok Build in Vibe42

Run Grok Build in Vibe42 with hybrid status tracking, mobile push notifications, and persistent terminal sessions.

`vibe42.ai/docs/agents/grok-build`

Grok Build is xAI's terminal coding agent—described as "a powerful and extensible coding agent" that opens an interactive terminal interface when you run `grok` in a project directory. Read this page to learn how to launch Grok Build in Vibe42, how hybrid status reporting works, and how to supervise tasks from your phone.

Launching in Vibe42

Grok Build is included in Vibe42's default agent list. Pick **Grok Build** in step three of the new terminal flow (which runs `grok`), or launch it from scripts with `v42 term new <computer> --agent grok-build`.

Vibe42 runs your local installation inside an interactive login shell (`<shell> -l -i -c`) with zero injection—your credentials, config files, and API traffic stay on your computer. To launch via a wrapper script or pass custom startup arguments, create or edit an entry in **Agents** settings using a single-line `<program>` command. If terminal isolation is enabled on the Workspace, the session starts in a fresh isolated copy. When the agent exits, the window returns to your shell.

Status reporting

Grok Build uses **partial self-reporting plus the wrapper engine**: the CLI emits partial status signals on its own, and the Vibe42 background service's wrapper engine fills in the remaining transitions from screen patterns and session records via OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42).

In the sidebar and **Agents** tab, Vibe42 tracks agent status across **Idle**, **Working**, **Done**, **Needs input**, and **Error**. When the agent pauses for **Needs input**, Vibe42 distinguishes between:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Each time a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone (APNs/FCM with zero background battery drain), running desktop app, web app, or linked Telegram. Tap the alert to open that window on your computer.

Continuing from your phone

Because sessions keep running on your computer even when you close the app, you can hand off work to Grok Build at your desk and check back from iPhone, iPad, or Android (see Mobile input):

- Approve prompts and navigate terminal menus using the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Compose multi-line follow-up prompts in the command bar (**Send to terminal**) before sending them to the pane.
- Upload a photo or screenshot from your phone to insert its local file path into the terminal.

Known limitations

- Grok Build must already be installed and signed in on your computer; Vibe42 renders the real terminal UI rather than a separate chat view.
- Launch through the new terminal flow or `v42 term new --agent` so the wrapper engine is attached. Typing `grok` manually in an existing shell skips wrapper supplementation, and terminals you opened outside Vibe42 or panes left over from before updating Vibe42 do not answer OSC 7501 probes.
- If the agent process does not appear within 20 seconds of launch, the wrapper engine stops trying to wrap that pane.

Devin CLI in Vibe42

Run Devin CLI in Vibe42 with wrapper-engine status tracking, push notifications, and mobile terminal navigation.

`vibe42.ai/docs/agents/devin`

Devin CLI is Cognition's local command-line coding agent—described as "a local command-line coding agent with deep Devin Cloud integration" that starts an interactive session when you run `devin` in a project directory. Read this page to learn how to start Devin CLI in Vibe42, how status tracking and notifications work through the wrapper engine, and how to control its terminal interface from your phone.

Launching in Vibe42

Devin CLI is preconfigured in Vibe42's default agent list. Choose **Devin CLI** in step three of the new terminal flow (which runs `devin`), or start a window from the command line with `v42 term new <computer> --agent devin`.

Vibe42 runs your installed binary inside an interactive login shell (`<shell> -l -i -c`) with zero injection: we never alter Devin CLI's configuration files, inject plugins, or proxy your API requests or credentials through our servers. To pass custom arguments or use a wrapper script, edit or create an entry in **Agents** settings with a single-line `<program>` command. Inside a Workspace with terminal isolation enabled, Devin CLI launches inside a dedicated isolated copy. When you quit Devin CLI, the window stays open at your shell prompt.

Status reporting

Devin CLI status is reported by the **Vibe42 background service's wrapper engine**. When started through Vibe42, the wrapper engine inspects screen patterns and local session records to report agent status across your devices.

In the sidebar and **Agents** tab, you see **Idle**, **Working**, **Done**, **Needs input**, and **Error** indicators (specific sub-state display depends on current screen rules; refer to what is shown in the app). Under **Needs input**, Vibe42 maps prompts to three standard categories:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Whenever a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone via APNs or FCM, running desktop app, web app, or linked Telegram. Tap the notification to jump directly to that Devin CLI window.

Continuing from your phone

Because Devin CLI keeps running on your computer even when you close the app, you can pick up the full TUI (an interactive interface drawn inside the terminal) from iPhone, iPad, or Android without interrupting an in-progress run (see Mobile input):

- Navigate TUI dialogs and confirm actions with the draggable D-pad, modifier keys, Escape, and custom shortcut buttons.
- Prepare multi-line instructions in the command bar (**Send to terminal**) and send them in one tap.
- Upload images or files from your phone to insert their local path on your computer into the prompt.

Known limitations

- Devin CLI must already be installed on your computer and signed in to your Devin account; Vibe42 displays the real terminal UI rather than a separate chat interface.
- Status tracking and agent push notifications require launching Devin CLI from the new terminal flow or `v42 term new --agent`. Starting `devin` manually inside a regular shell skips the wrapper engine.
- If the Devin CLI process does not appear within 20 seconds of launch, the wrapper engine stops wrapping that pane.

Antigravity CLI in Vibe42

Run Antigravity CLI in Vibe42 with hybrid status tracking, mobile push notifications, and persistent terminal sessions.

vibe42.ai/docs/agents/antigravity

Antigravity CLI (`agy`) is Google's terminal coding agent and the command-line form of Antigravity. Read this page to learn how to launch Antigravity CLI in Vibe42, how hybrid status reporting works, and how to supervise tasks from your phone.

Launching in Vibe42

Antigravity CLI is included in Vibe42's default agent list. Pick **Antigravity CLI** in step three of the new terminal flow (which runs `agy`), or launch it from scripts with `v42 term new <computer> --agent antigravity` .

Vibe42 runs your local installation inside an interactive login shell (`<shell> -l -i -c`) with zero injection—your credentials, config files, and API traffic stay on your computer. To launch via a wrapper script or pass custom startup arguments, create or edit an entry in **Agents** settings using a single-line `<program>` command. If terminal isolation is enabled on the Workspace, the session starts in a fresh isolated copy. When the agent exits, the window returns to your shell.

Status reporting

Antigravity CLI uses **partial self-reporting plus the wrapper engine**: the CLI emits partial status signals on its own, and the Vibe42 background service's wrapper engine fills in the remaining transitions from screen patterns and session records via OSC 7501 (a terminal control sequence that programs use to report their status to Vibe42).

In the sidebar and **Agents** tab, Vibe42 tracks agent status across **Idle**, **Working**, **Done**, **Needs input**, and **Error**. When the agent pauses for **Needs input**, Vibe42 distinguishes between:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before continuing.

Notifications

Each time a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone (APNs/FCM with zero background battery drain), running desktop app, web app, or linked Telegram. Tap the alert to open that window on your computer.

Continuing from your phone

Because sessions keep running on your computer even when you close the app, you can hand off work to Antigravity CLI at your desk and check back from iPhone, iPad, or Android (see Mobile input):

- Approve prompts and navigate terminal menus using the draggable D-pad, Enter, Escape, and custom shortcut buttons.
- Compose multi-line follow-up prompts in the command bar (**Send to terminal**) before sending them to the pane.
- Upload a photo or screenshot from your phone to insert its local file path into the terminal.

Known limitations

- Antigravity CLI must already be installed and signed in on your computer; Vibe42 renders the real terminal UI rather than a separate chat view.
- Launch through the new terminal flow or `v42 term new --agent` so the wrapper engine is attached. Typing `agy` manually in an existing shell skips wrapper supplementation, and terminals you opened outside Vibe42 or panes left over from before updating Vibe42 do not answer OSC 7501 probes.
- If the agent process does not appear within 20 seconds of launch, the wrapper engine stops trying to wrap that pane.

Cursor Agent in Vibe42

Launch Cursor Agent in Vibe42 with wrapper-engine status tracking, push notifications, and mobile terminal access.

`vibe42.ai/docs/agents/cursor-agent`

Cursor Agent brings Cursor's coding agent workflow into your terminal. Read this page to learn how to launch Cursor Agent in Vibe42, monitor its status through the Vibe42 background service's wrapper engine, and answer prompts from your phone.

Launching in Vibe42

Cursor Agent is included in Vibe42's default agent list. Pick **Cursor Agent** in step three of the new terminal flow, or start it from the command line with `v42 term new <computer> --agent cursor`.

Vibe42 runs your locally installed command inside an interactive login shell (`<shell> -l -i -c`) with zero injection: we never modify your Cursor configuration, install extensions, or proxy your Cursor account traffic. To customize startup arguments or wrap the launch in a script, add or edit an entry in **Agents** settings using a single-line `<program>` command. When Cursor Agent exits, the window remains open at your shell prompt so you can inspect changes.

Status reporting

Cursor Agent status is tracked by the **background service's wrapper engine**. When you start Cursor Agent through Vibe42, the wrapper engine watches terminal screen patterns and session records to report agent status across your connected devices.

The sidebar and **Agents** tab display **Idle**, **Working**, **Done**, **Needs input**, and **Error** indicators (specific sub-state labels depend on screen rules; refer to what is shown in the app). When **Needs input** is reported, its three sub-states correspond to:

- **Needs permission**: Waiting for you to approve a tool call.
- **Needs an answer**: Waiting for you to answer a question.
- **Needs sign-in**: Needing sign-in before proceeding.

Notifications

Each time a pane (one split of a terminal window) transitions into **Done**, **Needs input**, or **Error**, Vibe42 sends a notification to your phone (APNs/FCM), running desktop app, web app, or linked Telegram. Tap the alert to open Vibe42 directly to that Cursor Agent window.

Continuing from your phone

Because Cursor Agent keeps running on your computer even when you close the app, you can leave your desk and stay in control from iPhone, iPad, or Android (see Mobile input):

- Confirm tool approvals or choose options using the draggable D-pad, Enter, Escape, and synced custom shortcut buttons.
- Write multi-line instructions in the command bar (**Send to terminal**) before sending them into the terminal.
- Upload screenshots or files from your phone to insert their local path on your computer into the session.

Known limitations

- Cursor Agent must already be installed and signed in on your computer; Vibe42 streams the real terminal UI and does not provide a separate chat view.
- Because status comes from the wrapper engine, you must launch Cursor Agent via the new terminal flow or `v42 term new --agent`. Running the command manually inside an existing shell does not enable status dots or agent notifications.
- If the Cursor Agent process does not start within 20 seconds of launch, the wrapper engine stops wrapping that pane.

05 Workspaces & files

Workspaces

Link a folder on any computer to a named Workspace so you and your Team can open terminals in that directory, browse its files, and organize sessions.

`vibe42.ai/docs/workspaces/workspaces`

A Workspace links a folder path on one of your computers to a named project in the Vibe42 sidebar. Use Workspaces when you want new terminals to start in the right project folder automatically, browse the live file tree in the right panel next to your terminal, or organize coding agent tasks across your Team.

What a Workspace is—and is not

A Workspace is a simple pointer: **one computer + one absolute folder path (the full path from the root directory) + a name**.

- **Live from your disk**: Clicking **Files** under a Workspace in the sidebar reads the folder tree directly from the Vibe42 background service on that computer.
- **No cloud file syncing**: Vibe42 servers store only the Workspace record (name, sidebar order, and archive status). Your source code, untracked files, and isolated copies stay on your computer and are never uploaded to Vibe42 storage.
- **Multiple Workspaces per folder**: You can point more than one Workspace to the exact same path on the same computer—for example, to separate feature work from release tasks.

Navigating the sidebar

The top of the sidebar has three tabs: **Agents**, **Workspaces**, and **Recents**. Each device remembers which tab you last selected.

At the end of the tab row, click **+** to open the **New Terminal / Open App** launcher. It groups targets under **Workspaces**, **Computers**, and **Remote Desktop** with instant search so you can start a terminal, coding agent, or remote desktop session from one place.

When you expand a Workspace under the **Workspaces** tab, the first row is **Files**, followed by its terminals. Clicking **Files** opens a folder tab rooted at that Workspace folder in the right panel (the sidebar no longer embeds a folder tree); when the computer is offline, this row is grayed out. See Files.

Creating a Workspace

Create a Workspace in three steps:

1. Select the computer that holds the folder under **Computer**.
2. Browse or type the absolute folder path on that computer under **Location**.
3. Enter a name under **Details**, and keep **Open a terminal in this workspace after creation** checked if you want to open a terminal or agent right away.

Workspace actions and archiving

Open the menu on any Workspace to manage it:

Menu item	What it does
New terminal in this workspace	Opens a new terminal window starting in the Workspace folder
Open in Files	Opens the Workspace folder in the full Files view
Copies…	Shows isolated terminal copies and change history (see Terminal isolation)
Edit…	Changes the Workspace name, path, or folder options such as isolation and compression & deduplication
Move up / **Move down**	Moves the Workspace up or down in the Team sidebar
Archive	Removes the Workspace from the sidebar without touching any files on disk

> NOTE
>
> When you select **Archive**, Vibe42 asks for confirmation: *"The workspace leaves the sidebar and its terminals return to Recents. Files are not touched."* Running terminals keep working normally—see Sessions.

Terminal ownership, cwd, and Recents

Terminals and Workspaces follow three rules:

- **Starting directory**: A terminal opened inside a Workspace starts with that folder as its current working directory (`cwd`, the folder where commands run)—or the matching path inside a new copy when Terminal isolation is on. If you right-click a subfolder in the Workspace's folder tab and choose **New terminal here**, the terminal starts with `cwd` set to that subfolder while staying in the Workspace.
- **Moving terminals**: You can move a running terminal between Workspaces on the **same computer**, or move it out of a Workspace. Moving only changes how the terminal is grouped in the sidebar; it does not change the running shell's `cwd`.
- **Recents**: Terminals opened outside any Workspace—and terminals left running when a Workspace is archived—appear under the **Recents** tab.

Team sharing and permissions

Workspaces belong to your Team, and every Team member can see and use them by default:

- **Owner** and **Admin** roles can create, edit, reorder, and archive Workspaces.
- **Member** roles can open existing Workspaces, browse their file trees, and run terminals inside them.

If teammates' projects clutter your sidebar, toggle **Show team shared workspaces / Hide team shared workspaces**. "Shared" means Workspaces created by someone else on your Team. Hiding them only changes the view on your current device and does not affect other members.

Files

Open folder tabs in the right panel, follow a terminal's current directory, preview code and media, transfer large files, and insert paths into shells.

`vibe42.ai/docs/workspaces/files`

Vibe42 lets you browse, preview, upload, and download files on any connected computer without setting up SFTP or SCP (standard SSH file transfer tools). Read this page to see what file actions are available, how preview size limits and media streaming work, and how file transfers use Credits and system permissions.

What Files can and cannot do

The Files view shows every folder that the operating system user account running the Vibe42 background service can read, starting in that user's home directory by default.

Supported actions:

- Move between folders with **Up one level**, **Home**, or **Edit path** (type any full path from the root directory and press Enter)
- Filter long folder lists locally with **Filter this folder**, and show or hide dotfiles (hidden files whose names start with `.`) with **Show hidden files / Hide hidden files**
- Create folders with **New folder**
- Preview source code, Markdown, images, PDFs, audio, and video
- Upload files with the upload button or drag-and-drop (**"Drop files here to upload"**), download files of any size, and export files with **Share** on iOS and Android
- Copy file paths, or insert shell-escaped paths (wrapped in quotes so spaces and special characters are safe in commands) straight into a running terminal, or insert a path from the computer through the terminal attachment menu (see below)

Intentionally omitted:

> **IMPORTANT**
>
> Vibe42 does **not** edit, delete, rename, or move files in the app interface. Make edits, deletions, renames, and moves inside your terminal using your shell, editor, or coding agent.

Entry points and context menu

You can browse files in two places: the full **Files** page (opened by choosing **Open in Files** from a Workspace menu) and the right panel (the side panel to the right of your terminal). Both use the same file browser.

Folder tabs in the right panel

When you expand a Workspace in the sidebar, the first row under it is **Files**, followed by the terminals in that Workspace. Click **Files** to open a folder tab in the right panel with that Workspace folder as its root; the sidebar itself does not embed a folder tree. When the computer is offline, this row is grayed out and cannot be opened. (**Open in Files** in the Workspace menu still opens the full-page **Files** browser.)

A folder tab shows the directory tree rooted at that folder, and the path breadcrumb at the top stops at that root without going above it. Use the toolbar buttons to show or hide hidden files (**Show hidden files / Hide hidden files**), **Refresh**, **New folder**, or **Upload** (which uploads into the currently selected folder). When you click **New folder**, type the new folder name directly in the tree: press Enter to create it or Esc to cancel. If creation fails, the error appears right next to the input box so you can edit the name and try again; once created, the new folder becomes the currently selected folder.

The right panel has two permanent tabs that cannot be closed: **Memo** and **Current Folder**. All other tabs—file preview tabs and folder tabs—can be closed and share a limit of **9 tabs in total**. Opening the same file again, or the same folder on the same computer, switches to the existing tab instead of taking another slot. If all 9 slots are full when you open a new one, Vibe42 automatically closes the **oldest opened** tab. On phones, the right panel is closed by default and opens full-screen (with a **Back** button on the left of the tab bar) only when you explicitly open it: by tapping the right-panel button, opening a file, tapping **Files** under a Workspace, or tapping a file path in the terminal.

The Current Folder tab

The **Current Folder** tab follows the folder of your **active terminal** (the terminal's current working directory, `cwd`):

- **Root folder**: If the active terminal belongs to a Workspace and its folder is inside that Workspace folder, the tree root is the Workspace folder; otherwise, the root is the top level of that computer's file system.
- **Automatic follow**: Switching to another terminal or running `cd` to change folders in the terminal automatically jumps to and expands the new location in the tree.
- **Manual browsing**: If you click another folder in the tree yourself, the tree stays where you clicked until the terminal changes folders again.
- **No active terminal or offline**: When no terminal is open or that computer is offline, the tab displays **"Open a terminal and this tab will follow its current folder."**.

Right-click (or long-press on a touchscreen) any file or folder in any file tree for terminal shortcuts:

Context menu action	Target	Behavior
New terminal here	Folders	Opens a new terminal with its working directory (`cwd`) set to that folder and attaches it to the Workspace
Open in terminal	Files & folders	Opens a terminal at the item's location
Send path to current window	Files & folders	Inserts the single-quote-escaped path into the active terminal without pressing Enter
Copy absolute path / **Copy relative path**	Files & folders	Copies the full path or Workspace-relative path to your clipboard
Upload files here	Folders	Uploads local files into the selected folder

File previews and size budgets

Vibe42 picks a preview viewer based on the file extension (plus common filenames without extensions):

Category	Supported formats
Text & source code	Common code extensions plus `Makefile`, `Dockerfile`, `.gitignore`, and config files
Markdown	`.md` and Markdown documents
Images	`png`, `jpg`, `jpeg`, `gif`, `webp`, `bmp`, `ico`, `svg`, `avif`
Documents	`pdf`
Audio	`mp3`, `wav`, `ogg`, `flac`, `m4a`, `aac`
Video	`mp4`, `webm`, `mkv`, `mov`, `avi`, `m4v`
Archives	Not previewed in the app—download only

To keep the app fast on large repositories, previews follow strict size limits:

- **2 MiB text cap**: Text files over 2 MiB show **"File is too large to preview"** with a download button. Unrecognized binary files show **"Binary file cannot be previewed"** with a download button.
- **Rich-text rendering budget**: Syntax highlighting and rich formatting run on files up to roughly 260,000 characters, 3,000 lines, or 4,000 characters on a single line. Files under 2 MiB that pass those limits open as paged plain text instead.
- **Audio and video seeking**: Desktop and mobile apps stream media in small byte ranges so you can jump ahead on the timeline right away; the web app loads the whole media file first.

Uploads, downloads, and Credits

Check active and finished transfers in the **Transfers** panel:

- **Safe uploads**: Uploads write to a temporary `.part` file first and rename it to the final filename only when complete, so a dropped connection never leaves a half-written file.
- **Unlimited streaming downloads**: Downloads stream straight to your device with no file-size limit.
- **Separate transfer channel**: Every connection uses separate `control`, `terminal`, and `file` channels, so multi-gigabyte transfers never slow down typing in your terminal (see Connections).
- **Credits**: File traffic uses Credits whenever the connection goes through a Vibe42 relay server (an AP, or access point, which includes all browser sessions). Direct peer-to-peer transfers between native apps and the background service use zero Credits; see Credits and usage.

Insert a path from the computer into a terminal

The terminal shortcut bar (on both desktop and mobile) includes a paperclip button (**Add photo or file**) with **Insert Remote Path** in its menu. Use it to insert the path of a file or folder that is **already on that computer** into your terminal without uploading anything (unlike taking a photo, picking from the photo library, or choosing a file, which upload files from the device in your hand to the computer).

If the terminal's computer is the same device you are currently using and your operating system provides a file picker, Vibe42 opens the system picker; otherwise, it opens Vibe42's own file browser window where you can pick either a file or a folder (the web app and iPhone / iPad always use Vibe42's window). Once you pick an item, Vibe42 inserts its single-quote-escaped path (so spaces and special characters are safe in

commands) at your cursor **without pressing Enter**. The file browser window starts in the folder last used on that computer—or the active terminal's current folder the first time—and shares this last-used folder with the folder picker when you create a new Workspace.

Platform permissions and dependencies

- **When a folder won't open**: If that computer denies access to a folder, or reading it takes longer than 8 seconds, Vibe42 shows a notice in its place. The notice reads **"Listing is taking a while. That computer may be waiting for you to grant access."** or **"That computer denied access to this location."**, followed by **"Grant access from the remote desktop or a terminal, then refresh."**, and offers up to two buttons:
  - **Open remote desktop**: Shown only when that computer supports Remote Desktop and is not the device you are currently using. Use it to connect and click Allow on that computer's screen.
  - **New terminal here**: Opens a new terminal in that folder. Running `ls` once in the terminal triggers the system permission prompt.

  On macOS, when accessing protected folders such as Desktop or Documents, the system permission prompt appears on **that computer's own screen**, and someone must click Allow on that computer before Vibe42 can read the folder. Once access is granted, click **Refresh**.
- **macOS Full Disk Access (optional)**: Turning on **Full Disk Access** for Vibe42 on macOS lets you drag attachments directly from sandboxed apps (apps restricted by macOS security rules, such as WeChat or Apple Mail) into Vibe42.
- **Linux desktop preview sandbox**: Code, Markdown, HTML, and PDF previews in the Linux desktop app run inside an isolated WebKitGTK browser view and require `bubblewrap` (`bwrap`) and `xdg-dbus-proxy` on your system.

Previewing results and forwarding ports

Check what your agent built by previewing files, forwarding dev server ports with v42 forward, or opening Remote Desktop.

`vibe42.ai/docs/workspaces/preview-and-forwarding`

When a coding agent generates a report, saves a screenshot, or starts a local development server on your remote computer, you have three ways to inspect the result. Read this page to pick the best option and set up TCP port forwarding with `v42 forward`.

Three ways to inspect what an agent built

Method	Best for	Works on
Files preview	Static outputs: Markdown reports, images, PDFs, videos, logs, and source code	Phone, tablet, web app, desktop app
`v42 forward` (CLI)	Interactive web apps, APIs, and local dev servers listening on `127.0.0.1`	Any laptop or desktop computer running the `v42` CLI

Method	Best for	Works on
Remote Desktop	Desktop GUI apps, Xcode or mobile simulators, or viewing a browser from a phone or tablet	Phone, tablet, web app, desktop app

Previewing static artifacts in Files

When an agent writes files to disk—such as test screenshots, architecture diagrams, PDFs, recorded `.mp4` videos, or Markdown summaries—open the Files panel or Workspace folder tree and click the file to view it right away. You do not need to run a web server or forward a port, and it works the same way on a phone or in a browser.

Forwarding ports with `v42 forward`

To use a live development server (such as Vite, Next.js, or a local API) from a laptop or desktop computer, forward its network port with the Vibe42 CLI:

```
v42 forward <computer>:<remote-port> [<local-addr>:]<local-port>
```

`v42 forward` listens on a local TCP port on the computer in front of you and forwards traffic through an encrypted tunnel straight to `127.0.0.1:<remote-port>` on the remote computer.

Common examples

Forward remote port `3000` on `my-mac` to local port `3000`:

```
v42 forward my-mac:3000 3000
```

Then open `http://127.0.0.1:3000` in your local browser.

Pass `0` as the local port to let your operating system pick a free port automatically:

```
v42 forward my-mac:3000 0
```

Security boundaries and connection limits

- **Local-only by default**: If you leave out `<local-addr>`, `v42 forward` listens only on `127.0.0.1` (the loopback address that only your own computer can reach), so other devices on the same Wi-Fi cannot connect to the tunnel. Binding to a non-loopback address prints a clear security warning.
- **Remote loopback isolation**: By default, forwarding can only reach the remote computer's own loopback address (`forward:loopback` is granted along with terminal access). Forwarding through that computer to other machines on its network is blocked unless you explicitly allow it on that computer.
- **Concurrency cap**: Each `v42 forward` session supports up to **64 simultaneous TCP connections**.
- **End-to-end encrypted, zero payload logging**: Forwarded traffic uses the same Noise end-to-end encryption as your terminals. Audit logs never record the contents of forwarded traffic (see Security).
- **Credits**: When the tunnel between your CLI and the Vibe42 background service goes through a relayed AP connection, forwarded traffic counts toward Credits and usage. Direct peer-to-peer tunnels use zero Credits.

> **NOTE**
>
> Port forwarding is available only through the `v42 forward` CLI command. Vibe42 does not include a port-forwarding button in the app interface, and it never exposes your ports on public internet URLs.

Using Remote Desktop as a fallback

If your agent builds a desktop macOS or Linux app, opens an iOS Simulator, or you want to test `localhost:3000` from an iPhone, iPad, or browser where the `v42` CLI is not running locally, open Remote Desktop to view and control the remote computer's screen directly.

Terminal isolation

Give every terminal window its own copy-on-write clone of your Workspace folder so parallel agents can edit, build, and commit without overwriting each other.

`vibe42.ai/docs/workspaces/terminal-isolation`

When two coding agents edit the same project folder at the same time, they overwrite each other's files, break incremental builds, and conflict in git. Terminal isolation solves this by giving each new terminal window its own instant copy-on-write clone (a fast filesystem copy that only uses extra disk space for files that actually change) of the entire Workspace folder. Read this page for filesystem requirements, automatic version rules, the **Copies…** workflow, known limitations, and the local `v42 copy` CLI.

How it differs from `git worktree`

`git worktree` only checks out files tracked by git, so untracked `.env` files, downloaded assets, and dependency folders (`node_modules`, `target`) are missing unless you copy or link them yourself.

Terminal isolation clones the **entire folder**—including `.git`, untracked config files, and installed dependencies—using filesystem copy-on-write blocks. Clones finish in seconds regardless of project size, use extra disk space only for modified blocks, and work on folders that do not use git at all.

Terminal isolation is a per-Workspace setting. It is **off by default**, can be turned on when creating a Workspace or later under **Edit…**, and **cannot be turned off once enabled**.

Requirements and unsupported cases

If Vibe42 cannot create an isolated copy safely, it refuses to open the terminal and shows the exact reason instead of quietly dropping you into the original folder:

Condition	Result / message
Filesystem does not support copy-on-write (for example, `ext4`)	Rejected: **"This file system doesn't support copy-on-write clones (for example ext4). Use APFS, btrfs, XFS or ZFS."**
Folder is `/`, your home directory, or a parent folder above home	Rejected: **"Your home folder, its parent folders and / can't use terminal isolation."**

Condition	Result / message
Folder is a `git` linked worktree	Rejected
Target path does not exist or is not a folder	Rejected
Outdated Vibe42 background service	Rejected until the background service is updated
Corrupted copy index	Rejected; run `v42 copy rebuild-index --yes` on the computer
Initial compression & deduplication conversion running	Rejected until the conversion finishes

Where copies live and how panes share them

Opening a new terminal window creates a numbered copy next to your Workspace folder:

```
<parent-dir>/.<folder-name>.v42-copies/<number>/work
```

- **Numbering**: `<number>` always uses the lowest available positive whole number (`1` , `2` , `3` , …).
- **Split panes share a copy**: Splitting a terminal window into multiple panes (side-by-side or stacked views inside one window) keeps all panes in that window inside the same copy.
- **Subfolder launches**: Choosing **New terminal here** on a subfolder inside a copy creates a new numbered copy and starts the terminal in that same relative subfolder.
- **Shared across same-path Workspaces**: Multiple Workspaces pointing to the same path on the same computer share the same copies and version history.

Automatic versions

Vibe42 saves a version snapshot of a copy automatically whenever files have changed:

- When an agent finishes a turn (Agent status changes via OSC 7501—a terminal status sequence—to **Done**, **Needs input**, or **Error**)
- Every 10 minutes while there are unsaved changes
- When the terminal window closes
- Manually when you click **Save version** (or run `v42 copy save`)

If no files changed, no new version is created. `.git` folders at any depth and files ignored by git are left out of version history.

Reviewing and applying changes in Copies…

Open **Copies…** from the Workspace menu to see every copy, its version history, and its modified files.

- **Copy states**: **In use**, **Closed**, **Preparing**, **Folder missing**, **Changed outside Vibe42**, **No history**, or **Starting point missing**.
- **Per-file status**: Each changed file is marked **Not applied**, **Applied**, or **Conflict** (**"This file also changed in the workspace folder. Resolve it by hand."**).
- **Apply selected** / **Apply all**: Writes unapplied file changes back to the original Workspace folder. Vibe42 checks each file again right before writing and skips any file that changed after you opened the list.

- **Push as branch**: In git repositories, pushes the copy's current `HEAD` commit into the original Workspace repository as a new branch. Git itself blocks pushes to the branch currently checked out in the main folder and blocks non-fast-forward updates.
- **Delete copy**: Deletes the copy from disk, asking for confirmation first if unapplied or unsaved changes remain.

Lifecycle, cleanup, and external edits

Copies stay on your computer's local disk. When you close a terminal window, Vibe42 saves one final version first. If no other pane is using that copy and the copy is **clean**—meaning zero unapplied changes and, for git repos, its `HEAD` commit is already in the main Workspace repo—Vibe42 deletes the copy automatically. Copies with unapplied work stay on disk until you apply, push, or delete them.

If files in the copy or the main Workspace folder are edited outside Vibe42 in a way that breaks change tracking, Vibe42 shows a clear error or conflict rather than quietly overwriting files or guessing a merge.

Known limitations

- Hardcoded full paths inside generated files (such as Python `venv` scripts) still point to the original Workspace path.
- The first `cargo build` in a new Rust copy recompiles crates whose build outputs depend on the full folder path.
- Files ignored by git are not saved in version history and are deleted when a clean copy is removed automatically.
- Changing a Workspace's folder path leaves existing copies at the old location.

Local `v42 copy` CLI and agent skill

Inside an isolated terminal, the `V42_COPY` environment variable points to the active copy. You can manage copies locally on the computer without signing in:

```
v42 copy status
v42 copy save
v42 copy versions
v42 copy changes
v42 copy diff
v42 copy apply
v42 copy push
v42 copy rebuild-index --yes
v42 copy skill
```

Running `v42 copy skill` prints a `SKILL.md` guide titled **"Working in a Vibe42 isolated copy"** that you can add to your coding agent if you want (see Zero injection).

When both terminal isolation and Compression and deduplication are enabled, `.v42-copies` is deduplicated and compressed too; creating copies, applying changes, pushing, deleting, and running `rebuild-index` pause only during the initial full-folder conversion.

Compression and deduplication

Save disk space on APFS, btrfs, XFS, and ZFS by sharing identical file blocks and applying transparent filesystem compression across your Workspace and copies.

`vibe42.ai/docs/workspaces/storage-dedup`

Dependency folders, build caches, and parallel agent copies often leave thousands of identical or easily compressible files on disk. Compression and deduplication is a per-Workspace setting that reclaims local disk space automatically at the filesystem level without changing what your programs read. Read this page for supported filesystems, conversion rules, safety checks, and how it works with terminal isolation.

What it does—and never touches

When turned on for a Workspace, the Vibe42 background service merges duplicate file blocks that have the exact same content (deduplication) and turns on built-in filesystem compression (transparent compression, where the operating system compresses data on disk and decompresses it automatically when read) where supported.

- **Identical file reads**: Applications, compilers, and shells read the exact same file contents, permissions, and modification times as before.
- **Untouched paths**: Vibe42 never touches `.git` folders at any depth, symbolic links, hard links, files owned by other operating system users, or files with special filesystem flags.
- **One-way switch**: Compression and deduplication is **off by default**. Once turned on for a Workspace, **it cannot be turned off**.

Filesystem support matrix

Only filesystems that support safe block sharing are supported. Note that `ext4` is not supported here:

Operating system & filesystem	Deduplication	Transparent compression	Status
macOS **APFS**	Content deduplication	`decmpfs` (LZFSE)	Supported
Linux **btrfs**	Content deduplication	`zstd`	Supported
Linux **XFS**	Content deduplication	None (deduplication only)	Supported
Linux **ZFS**	Content deduplication (block cloning)	Set by the dataset `compression` property	Supported (OpenZFS with block cloning turned on)
ext4 and others	—	—	Not supported

Trying to turn the setting on for an unsupported filesystem shows:

> **"This file system doesn't support it. Use APFS, btrfs, XFS or ZFS."**

You also cannot turn it on when the target folder is `/`, your home directory, or a parent folder above home, when the background service is outdated, or while the computer is offline.

Initial conversion and background watching

Turning the setting on immediately starts a full conversion pass across the Workspace folder:

- **Runs independently on the background service**: The conversion runs in the background on your computer whether or not any Vibe42 app stays connected, and picks up where it left off automatically if the background service or computer restarts.
- **Ongoing maintenance**: After the first pass finishes, the background service watches the folder—processing changed files once they have stopped changing for about **1 minute**, plus a full check every **30 minutes**.

Temporary limits during the initial conversion

While the first full-folder conversion is running:

- Opening new isolated terminals—and running copy apply, push, delete, or `rebuild-index`—is paused with a message that conversion is in progress.
- **Edit…** and **Archive** on that Workspace are disabled until conversion finishes.
- Regular non-isolated terminals and the Files browser keep working normally.

File processing rules

Every file must pass strict safety checks before and during replacement:

Rule	Behavior
Minimum file size	Only regular files **≥ 16 KiB** are processed
Quiet period	Files modified within the last **1 minute** are skipped
Open write handles (macOS APFS, Linux ZFS)	Files currently open for writing by any program are skipped
Excluded entries	`.git` (at any depth), symlinks, hard links, other users' files, and specially flagged files are skipped
Atomic verification	Vibe42 checks again right before replacement that the file did not change, and restores its original permissions and timestamps

Progress states and space savings

You can check conversion progress and saved disk space in the Workspace settings:

- **Active states**: **Waiting to start…** → **Scanning files…** → **Comparing files (…%)** → **Converting files (… of …)** → **Saving…**.
- **Completed summary**: Shows **"Saved …"** along with how much space was saved by deduplication versus transparent compression.
- **Diagnostics**: If a step ran into a problem, Vibe42 shows **"Last error: …"**.

Relationship with terminal isolation

Compression and deduplication is separate from Terminal isolation—you can turn on either one alone or both together. When both are enabled on a Workspace, all `.v42-copies` clones are deduplicated and compressed automatically too, keeping parallel build outputs small on disk.

Memos, rest reminders and housekeeping

Save reusable prompts and notes in Workspace memos, set daily rest reminders, toggle the local macOS background service, and clean up offline computers.

`vibe42.ai/docs/workspaces/extras`

Alongside terminals and file trees, Vibe42 includes simple tools for keeping reusable prompts next to a project, setting daily break times, choosing whether your Mac can be controlled remotely, and removing old offline machines. Read this page to set up Workspace memos, rest reminders, the local background service toggle, and cleanup actions.

Workspace memos

Memos live inside a Workspace so you can keep recurring agent prompts, test commands, or handoff notes right next to the project they belong to. Memos are in the **Memo** tab of the right panel, which shows the memos for the Workspace that owns your currently open terminal or file.

Visibility levels

Every memo uses one of three visibility settings inside your Team:

Visibility	Who can view	Who can edit
Private	Only you	Only you
Team read-only	All Team members	Only you
Team editable	All Team members	All Team members

Sending memos to a terminal and lifecycle

- **Send to current terminal**: Pastes the memo text straight into your active terminal without pressing Enter, so you can check flags or add arguments before running it.
- **Mark as done**, **Restore**, **Delete memo**: Mark finished items done, restore completed memos when you need them again, or delete them permanently.
- **Active memo cap**: Each Workspace has a limit on the number of incomplete memos; mark old items done or delete them when you hit the limit.

Rest reminders

When you can check on agents from your phone at any hour, it is easy to work late into the night without noticing. Set a daily break window under **Settings → Wellbeing**:

- **Synced schedule, local clock**: You set one daily time window. The schedule is saved on the server and synced across all your signed-in devices, using each device's own local time zone.
- **Locked workspace during rest**: While the rest window is active, only **Settings** stays open in the app (so you can change or turn off the schedule in an emergency).
- **Notifications continue**: Push notifications still arrive normally during your rest window so you do not miss urgent alerts.

Local background service toggle on macOS

When you first open the macOS desktop app, it asks **"Use this computer as a remotely controllable computer?"** with **Enable this computer** and **Not now** options, and you can turn the local Vibe42 background service on or off later:

- **Uses a computer slot**: Turning on the local background service registers your Mac as a computer you can control remotely and uses one computer slot on your Team's plan.
- **Local use without enabling**: If you choose **Not now** (or have no free computer slots), you can still use local terminals inside the Mac app and connect to your other remote computers.
- **Scoped to the active Team**: Turning the local background service on or off applies only to the Team currently selected.

Housekeeping: offline computers and Recents

Cleaning up computers offline for more than 3 days

Temporary cloud VMs and rebuilt containers can leave dead entries in your **Computers** list. On that page, click **Remove offline computers** to remove computers that have been offline for **more than 3 days**.

> CAUTION
>
> Cleaning up offline computers **permanently deletes** all cloud Workspace records and Workspace memos tied to those computers. Do not run this cleanup if a laptop or desktop is just turned off while you are away.

Hide from Recents

When the **Recents** tab in the sidebar collects items you no longer want to see, choose **Hide from Recents** to clear an entry from the list; exact availability and behavior match what is shown in the app.

06 Remote desktop

Remote desktop: getting started

Stream and control your Mac or Linux desktop from any paired device, grant screen and input permissions, and check platform support.

`vibe42.ai/docs/remote-desktop/getting-started`

Remote Desktop lets you view and control the graphical desktop or individual app windows on your Mac or Linux computer from any paired device. Read this page when you want to use desktop apps like Xcode, browsers, or simulators alongside your terminals, or when you need to grant screen and input permissions for the first time.

Supported platforms

Remote Desktop requires a logged-in graphical desktop on the computer you want to control, and a supported Vibe42 app or browser on the device you are connecting from. You cannot connect from a device back to that same device.

Platform	As a controlled Computer	As a controlling Device
macOS 14+ (macOS app)	Yes (built-in background service)	Yes
Linux desktop (X11)	Yes	Yes (Linux app)
Linux desktop (Wayland)	Yes on KDE Plasma 6, GNOME, and Hyprland	Yes (Linux app)
iPhone and iPad (iOS 17+)	No	Yes
Android (11+)	No	Yes
Web app (`app.vibe42.ai`)	No	Yes (browser with WebCodecs support)
Windows	No	Yes (via the web app in a browser)

The controlled computer must run a version of the Vibe42 background service that includes Remote Desktop support. If an older background service is running, Vibe42 shows **This computer does not support Remote Desktop yet** and asks you to update. See Updating and uninstalling for upgrade instructions.

Granting permissions on macOS

Before another device can view or control your Mac, macOS requires two system permissions:

1. **Screen Recording** — Needed to capture display frames and app windows. Without it, remote devices cannot see your screen. Starting in macOS 15, macOS periodically asks you on the Mac to re-confirm screen recording access.
2. **Accessibility** — Needed to send remote mouse clicks, pointer movement, scrolling, and keystrokes to your Mac. If you turn on **Screen Recording** without **Accessibility**, you can view the screen but cannot click or type.

The macOS app walks you through both permissions during setup. You can also check or grant them later in **Settings → Remote Desktop** (this settings page only appears in the macOS app).

That same page has the **Allow other devices to control this Mac** switch, which is on by default. Turning it off ends any active Remote Desktop session to this Mac right away and hides the Mac from the Remote Desktop list on your other devices.

First-time setup on Linux

Permission prompts and video encoding on a controlled Linux computer depend on your display server (X11 or Wayland) and graphics card:

- **X11**: Connects immediately without a system permission prompt.
- **KDE Plasma 6 and GNOME (Wayland)**: On the first incoming connection, the Linux desktop shows a system dialog on the computer's screen. Approve the prompt and enable remote interaction; the desktop remembers your choice for future connections.
- **Hyprland (Wayland)**: Connects directly without a system dialog.

For video encoding, the background service on Linux uses your system `ffmpeg` for hardware encoding (VA-API, NVENC, or V4L2) whenever available. If hardware encoding is unavailable, it falls back to the Cisco OpenH264 software encoder. Because OpenH264 is not bundled in the installer, turn it on in the Linux app settings or run this command on the computer to download and verify the codec from Cisco:

```
~/.v42/bin/v42-companion codec enable
```

You can turn off the downloaded codec later if you switch to hardware encoding. For video decoding requirements on the device you connect from, see Controls, quality and clipboard.

Opening a Remote Desktop session

You can open Remote Desktop from several places in Vibe42:

- Click **Remote Desktop** in the sidebar and pick an online computer.
- Open the new terminal dialog and select a computer that supports Remote Desktop (you can also open a single application window directly; see Unlocking the screen and opening apps).
- Search for the computer's desktop in global search.
- Go directly to `/computers/<id>/desktop` in the web app.

Recent desktop sessions show up under **Recents** in the sidebar, and you can use **Pin** to keep frequent desktops pinned next to your terminals and Workspaces.

> **IMPORTANT**
>
> Remote Desktop streams live video and uses much more data than terminal sessions. Native apps connect directly peer-to-peer whenever possible at zero Credit cost, while the web app always routes through a Vibe42 relay server. All relayed Remote Desktop data counts toward your Team's monthly Credits. See Connections and Credits and usage.

What Remote Desktop does not support

Keep these limits in mind when using Remote Desktop:

- **Pre-login system screens**: You cannot control a Mac or Linux computer that is sitting at the system login screen after a reboot before a user has signed in (unlocking a locked screen for an already signed-in user is supported; see Unlocking the screen and opening apps).
- **Virtual and privacy displays**: Creating virtual monitors or blacking out the physical monitor is not supported.
- **Audio**: Remote audio playback and microphone input are not supported.
- **Rich clipboard and file drag-and-drop**: Only plain text (up to 60 KiB) syncs through the clipboard, and dragging files onto the desktop view does not upload them. Use the Files panel or `v42 cp` / `v42 sync` to transfer files.
- **Multi-user input locking**: If two devices connect to the same desktop at the same time, both can move the mouse and type without input locking.

Controls, quality and clipboard

Adjust stream quality and scaling, configure hardware or OpenH264 codecs, switch displays, and use keyboard, touch, and text clipboard sync.

`vibe42.ai/docs/remote-desktop/controls`

Once you connect to a Remote Desktop session, you can adjust video quality for your network, switch between monitors, choose physical or touch controls, and copy and paste text between devices. Read this page when you want to reduce lag or Credit usage, control a desktop from a phone or tablet, or check how video decoding and clipboard sync work.

Stream quality presets

Use the control bar at the top of the session to pick from five video quality presets:

Preset	Behavior	When to use
Auto	Adjusts resolution and frame rate automatically based on your network speed and round-trip latency (default)	Everyday connections, cellular networks, and unstable Wi-Fi
Smooth	Prioritizes low lag and steady frames over fine visual detail	Slow connections or quick window switching
Balanced	Balances readable text with moderate data usage	Routine desktop work over relayed connections
Sharp	Keeps text edges sharp and UI details clear	Reading code, logs, or dense inspector panels
Ultra sharp	Streams full native resolution at 60 frames per second (fps)	Local network or direct peer-to-peer (P2P) connections where clarity matters most

> **TIP**
>
> Check whether your connection is direct or relayed before leaving **Ultra sharp** open for a long time. When traffic goes through a Vibe42 relay server (such as in the web app), native-resolution 60 fps video uses Credits quickly. See Credits and usage.

Scaling and codecs

Two scaling modes control how the remote screen fits inside your window:

- **Fit to window** — Scales the entire remote screen or window proportionally so everything fits inside your window without scrolling.
- **Original size** — Shows the remote screen at 1:1 pixel size so you can pan around a high-resolution monitor on a smaller screen.

The video codec defaults to **Auto**. When your device can decode multiple formats sent by the computer, you can switch codecs from the control bar. Video decoding support depends on your platform:

- **macOS, iOS, and Android apps** use the operating system's built-in video decoders.
- **Web app (`app.vibe42.ai`)** requires a browser that supports the **WebCodecs** API.
- **Linux desktop app** uses hardware decoding (VA-API or NVIDIA NVDEC) when available, and downloads Cisco OpenH264 on demand if hardware decoding is not available.

For setting up video encoding on a controlled Linux computer, see Remote desktop: getting started.

Multiple displays

When the controlled computer has more than one physical monitor plugged in:

- Open the **Display** menu or use the display switcher in the top control bar to switch monitors.
- Expand the computer in the sidebar tree to pick `Screen 1`, `Screen 2` (`Screen N`) directly, or select an individual app window (see Unlocking the screen and opening apps).

Keyboard and pointer controls

On desktop apps and web

Physical keystrokes go straight to the remote computer. If you use an Input Method Editor (IME) for Chinese, Japanese, Korean, or accented characters, you compose the text on your local device and Vibe42 sends the finished characters to the remote window.

When the macOS app is active and your cursor is inside the Remote Desktop view, Vibe42 captures system shortcuts (such as app switching) and sends them to the remote computer instead of your local Mac. Click outside the Remote Desktop view to use your local Mac shortcuts again.

On iPhone, iPad, and Android

On phones and tablets, you can switch between two pointer modes in the toolbar:

- **Mouse mode** — Turns your screen into a virtual trackpad. Drag anywhere to move the mouse cursor, tap to left-click, and use the right-click button on the toolbar to open context menus. Best for clicking small

buttons or dragging window edges.

- **Touch mode** — Tapping anywhere on the image clicks that exact spot on the remote screen. Best for large buttons and quick taps.

The mobile toolbar also includes a row of modifier and function keys (for combinations using Command, Control, Option/Alt, Shift, Esc, or F-keys), a right-click button, and **Paste to remote** to paste text from your phone or tablet clipboard.

Clipboard synchronization

Remote Desktop syncs **plain text only, up to 60 KiB** per copy. Images, formatted text, and copied files are not sent through the clipboard.

To avoid overwriting your clipboard by accident, sync follows three rules:

1. **No automatic push on connect**: Text already in either machine's clipboard before you connect is never sent automatically when the session starts.
2. **Remote to local (automatic)**: Whenever you copy new plain text (up to 60 KiB) on the controlled computer during an active session, Vibe42 copies it to your local device's clipboard automatically.
3. **Local to remote (on paste)**: Vibe42 does not read your local clipboard in the background. It only sends your local clipboard text to the remote computer when you press `⌘V` (or trigger paste / **Paste to remote**).

Unlocking the screen and opening apps

Unlock a locked Mac or Linux session remotely, launch individual apps into single-window views, and pin frequent desktop sessions.

`vibe42.ai/docs/remote-desktop/unlock-and-apps`

When your computer locks while you are away from your desk, Vibe42 can unlock your logged-in desktop session remotely and let you open a single app window instead of viewing the whole monitor. Read this page when you need to unlock a remote screen, check what an unlock status message means, or work with single-window app views.

Unlocking a locked screen

When you connect to a locked computer—and the Vibe42 background service reports that the screen can be unlocked and remote control is allowed—Vibe42 shows a password prompt on your device. Enter the system login password for the user currently signed in on that computer, and the background service submits it locally to unlock the screen.

After you submit your password, the background service returns one of six results:

Result	Meaning
`Unlocked`	The lock screen accepted your password and the desktop session is now unlocked.
`NotLocked`	The session was already unlocked when the request arrived; no action was needed.

Result	Meaning
`WrongPassword`	The operating system rejected the password. Check your password and try again.
`NotOnConsole`	Your user account is not the active physical console session (for example, fast user switching moved the screen to another user account).
`Untypable`	The password contains characters that cannot be typed by the current input method, or the operating system blocked simulated keystrokes.
`Failed`	The screen locker or session manager ran into an unexpected error while trying to unlock.

Supported screen lockers by platform

- **macOS**: Built-in macOS lock screen for an already signed-in user account (requires **Accessibility** and **Allow other devices to control this Mac**; see Remote desktop: getting started).
- **Linux — KDE Plasma 6 and GNOME (Wayland)**: Unlocked through `logind` session integration.
- **Linux — Hyprland (Wayland)**: Supports `hyprlock`.
- **Linux — X11**: Supports screen lockers that implement the standard freedesktop screensaver D-Bus interface (such as `xfce4-screensaver`). Standalone X11 lockers that grab keyboard input directly without a freedesktop interface cannot be detected or unlocked remotely.

> **WARNING**
>
> Remote unlock only works when your system user account is **already signed in** and the screen is locked. Vibe42 cannot sign into the system login screen shown right after a boot or restart, because the user-level background service does not start until you sign in.

Password security

Your system password is sent only once when you submit the unlock dialog, going straight to the background service on your computer over the end-to-end Noise-encrypted connection. Neither your controlling device, the background service, nor Vibe42 servers ever store or log the password. For full encryption details, see Security.

Opening apps and single-window views

Streaming a full desktop monitor onto a phone, an iPad split view, or a small laptop window often makes text hard to read. Single-window view lets you open or focus just one remote application window at a time:

- **From the Remote Desktop sidebar tree**: Expand a computer to see its physical monitors (`Screen N`), an **Open App** action, and a live list of open windows grouped by running app. Click any window to view it on its own, or click **Open App** to launch a new application.
- **From the new terminal dialog**: Select a Remote Desktop computer in the new terminal dialog to open the **Open an app on …** picker. Pick an app to launch it on the remote computer and open its single-window view right away.

Inside single-window view, the window menu gives you three controls:

- Open the remote app's **App Menu**
- Choose **Close Window** to close that remote window cleanly

- Choose **Fit Window to Canvas** to resize the application window on your remote computer so it matches the size and aspect ratio of your local Vibe42 window

Fit Window to Canvas is handy when you use an IDE or simulator from an iPad or smaller laptop screen. For a step-by-step example, see Unlock your Mac and run Xcode.

Recents and pinning

Every Remote Desktop session—at `/computers/<id>/desktop`—is added to **Recents** in the sidebar as soon as you open it. If you switch back and forth between a coding agent terminal and a GUI window (such as a simulator or browser), use **Pin** to keep the desktop session pinned at the top of your sidebar.

07 Account, teams & plans

Account and sign-in

Create your account, verify your email, link GitHub or Google, set up Passkeys and TOTP 2FA, sign into devices, and understand account deletion.

`vibe42.ai/docs/account/account-and-login`

Use this page when creating a Vibe42 account, setting up Passkeys or two-factor authentication (2FA), signing into desktop apps or the CLI, or checking what happens if you delete your account.

Creating an account and verifying your email

Register at `https://vibe42.ai/register` in one of two ways:

- **Email and password**: Your password must be at least **10 characters** long.
- **Third-party sign-in (OAuth)**: Click **Continue with GitHub** or **Continue with Google**.

If you sign up with an email and password, Vibe42 sends you a verification email right away. The verification link is valid for **24 hours**, and resending has a short cooldown. **Unverified accounts cannot sign in on any device.** If the email does not arrive, check the email troubleshooting steps in FAQ.

GitHub and Google sign-in and account linking

Vibe42 only accepts primary email addresses that GitHub or Google has marked as verified:

- If your verified GitHub or Google primary email matches an existing **verified** Vibe42 account, signing in links the provider to that account.
- If it matches an **unverified** password-registered account, Vibe42 links the provider, marks the account verified, and **clears the old unverified password** so nobody can pre-set a password on your email address. If you still want to sign in with a password later, set a new one at `/password-reset`.

You can also link or unlink GitHub and Google anytime in **Console → Sign-in & security**.

Resetting a forgotten password

If you forget your password, request a reset email at `/password-reset` and set a new password of at least 10 characters.

Passkeys, TOTP, and recovery codes

Open **Console → Sign-in & security** to add stronger sign-in protection to your personal account:

- **Passkeys**: Register a hardware security key or a built-in passkey (Touch ID, Windows Hello, or a password manager). Your private key never leaves your device.
- **Authenticator app** (time-based one-time password 2FA): Pair an authenticator app and save your one-time **recovery codes** offline. Never share your password, 6-digit codes, or recovery codes with anyone, including support.

When step-up verification triggers

Even when you are already signed in, Vibe42 asks for a fresh **step-up verification** check before running sensitive account actions. It prompts you in the order **Passkey > TOTP > password** when you:

- Change credentials or 2FA settings in **Console → Sign-in & security**.
- Link or unlink a GitHub or Google account.
- Revoke installations on your account-level devices page.
- Delete your Vibe42 account.

How different clients sign in

- **Console & web app (`https://app.vibe42.ai`)**: Sign in directly in your browser.
- **Desktop apps (macOS / Linux) & `v42` CLI**: Click **Sign in** (or run the CLI sign-in command) to open your browser to a confirmation page titled **Sign in the Vibe42 app** or **Authorize a device**. Check the device details, make sure you started the request yourself, and click **Approve**.
- **iPhone, iPad & Android apps**: Mobile apps do not use passwords or third-party sign-in. Pair them using an 8-character code or QR code from an already signed-in device (see Pair a device and start your first session).

Active sessions and signing out

In Console, you can view all active web sessions and authorized installations, sign out individual sessions, or revoke a lost device. Revoked devices lose access within 60 seconds.

Deleting your account

> CAUTION
> Deleting your account requires step-up verification and permanently dissolves every Team you own.

When you confirm account deletion:

- Every Team where you are the **Owner** is dissolved immediately.
- Active subscriptions end right away with **no automatic refund** (see Trial, promo codes and refunds).
- All Team members, computers, and devices under your owned Teams lose access immediately.
- Your personal account data is deleted or anonymized within **30 days**. Invoices and tax records are kept as required by law.

Teams, roles and sharing computers

Manage multiple Teams, compare Owner, Admin, and Member permissions, invite teammates, and share computers safely across your Team.

`vibe42.ai/docs/account/teams`

Use this page when inviting teammates, checking what the **Owner**, **Admin**, and **Member** roles can do, or sharing a computer with your Team using the **Only me** and **Whole team** settings.

Personal Teams and belonging to multiple Teams

Every Vibe42 account starts with a personal Team where you are the **Owner**. You can also accept invitations to join other Teams and switch between multiple Teams under a single login.

Plans, slots, and Credits belong to the Team, not to individual users. When you join a Team that has a multi-member Team plan, you use that Team's slots and do not need your own subscription (see Plans and slots and Share a computer with your team).

Role permissions

Capability	Owner	Admin	Member
Select plan, pay & manage billing	Yes	No	No
Manage members & invitations	Yes	Yes (except Owner/billing)	No
Delete computers & enable/disable computers	Any computer in Team	Any computer in Team	No
Sign out devices	Any device in Team	Any device in Team	Own devices only
View Team Credits & slot usage	Yes	Yes	No
Create (add) new computers	Allowed by default	Allowed by default	Disabled by default (can be granted per member)

Inviting members

1. Send an email invitation from Console. Sending a new invite to the same email address cancels any earlier pending invite for that address.
2. Have the recipient sign into a **verified** Vibe42 account whose email address **exactly matches** the invited address.
3. Have the recipient open `https://vibe42.ai/console/invites` (or the Team picker when signing into the app) and accept the invite.

Vibe42 checks member seats **when the invite is accepted**, not when you send it. If all seats are full at that moment, the invitee sees a seat-limit message until a seat opens up or the Owner upgrades the plan.

Computer access scopes: Only me vs. Whole team

Every computer starts with its access scope set to **Only me**. Only the **creator of that computer** can change its access scope:

- **Only me** (default): Teammates with the **Member** role cannot see or connect to the computer.
- **Whole team**: Everyone with the **Member** role can see, connect to, and use the computer's terminals, files, and screen. **Whole team** only grants usage access; regular members still cannot delete or reconfigure someone else's computer.

> IMPORTANT
>
> The Team **Owner** and every **Admin** can always see and connect to **all** computers in the Team, whether a computer is set to **Only me** or **Whole team**. Owners and Admins also control which computers are enabled against the Team's computer slots.

Leaving or being removed from a Team

- Any **Admin** or **Member** can leave a Team at any time. An **Owner** cannot leave their own Team.
- When a member leaves or is removed, their computers and devices under that Team lose Team access within 60 seconds. Local files and running programs on their machines are not affected.

Switching Teams and unavailable Team states

- **Team selection screen**: If you belong to multiple Teams or have pending invitations, signing in shows a Team picker first so you can choose which Team to open.
- **Unavailable Team screen**: If your current Team cannot be used—for example, monthly Credits ran out, the Team is suspended, or you were removed—the app explains why and lets you switch to another Team on your account.

Plans and slots

See what Vibe42 plans include, how slot limit prompts work in the app, and how plan changes, renewals, and cancellations take effect.

`vibe42.ai/docs/account/plans-and-slots`

Use this page when choosing between an individual plan and a Team plan, handling a slot-limit prompt in the app, or checking when upgrades, downgrades, and cancellations take effect.

What a plan provides

Vibe42 does not have a free plan. Every active plan belongs to a **Team** and provides:

- **Computer slots** for enabled computers
- **Device slots** for signed-in apps, signed-in web browsers, and Server edition installations running the Vibe42 background service

- **Member seats** for people in the Team
- **Monthly Credits** for traffic relayed through Vibe42 access points (APs; see Credits and usage)

Individual plans are for one person, while **Team plans let multiple members share computers** inside the same Team. You can check exact slot numbers and prices at `https://vibe42.ai/#pricing` and in Console (`https://vibe42.ai/console/checkout?resource=access`). If you subscribed to an earlier plan, you keep that plan unless you switch.

> NOTE
>
> The iPhone and Android apps do not show pricing, checkout, or subscription settings. Manage your plan in a web browser through Console.

Slot counting summary

- Every **enabled** computer uses 1 computer slot.
- Every signed-in native app, signed-in `app.vibe42.ai` browser, and one-line Server installation (**Server edition**) uses 1 device slot.
- Member seats are counted when an invitee accepts an invitation.
- Turning off a computer or going offline does **not** free its slot. To free a slot, open Console and delete unused computers or sign out devices (you can click **Unpair offline devices** to sign out devices offline for more than 3 days at once).

For step-by-step slot counting examples—such as why a Mac running the desktop app uses 1 device slot plus 1 computer slot—see Computers, devices and teams.

In-app limit and blocking prompts

When a Team reaches a slot limit or cannot connect due to subscription status, the app shows one of four screens with a button to open Console (or switch Teams):

Prompt	Why it appears and what to do
Device limit reached	All device slots on the Team are in use. Open Console to sign out unused devices or upgrade the plan.
Computer limit reached	All computer slots are in use (newly paired computers appear as **Over plan limit**). Open Console to delete or disable an older computer, or upgrade the plan.
No member seat available	You tried to accept an invite to a Team whose member seats are full. Ask the Team Owner to free a seat or upgrade in Console.
Can't connect to this team	The Team has no active plan, ran out of monthly Credits, is suspended, or no longer includes you. Open Console to check billing, or switch to another Team on your account.

Changing plans, billing cycles, and cancellations

Only the **Team Owner** can buy, change, or cancel a plan. In Console's billing page, you manage subscriptions using **Change plan** and **Cancel subscription**.

- **Recurring vs. one-time billing**: Plans renew monthly by default, or you can choose **Pay once · no auto-renewal** at checkout.

- **7-day renewal billing & cancellation cutoff**: Recurring plans create the next renewal invoice **7 days before the renewal date**. To stop the next renewal, cancel **at least 7 days before the renewal date**.
- **Upgrades and downgrades**:
  - Qualifying upgrades take effect **immediately**.
  - All other plan changes and **all downgrades** follow the 7-day cutoff rule and take effect at the **end of the current billing period**.

For details on the 3-day free trial, promo codes, and refunds, see Trial, promo codes and refunds.

Credits and usage

See how Vibe42 counts relayed traffic at 64 MiB per Credit, why direct P2P connections are free, and what happens when Credits run out.

`vibe42.ai/docs/account/credits-and-usage`

Use this page to understand what uses Credits on your Team plan, why direct peer-to-peer (P2P) connections cost zero Credits, and how to check or reduce your monthly usage.

What Credits are

Credits are Vibe42's unit for measuring relayed network traffic. Your Team plan gives you a fresh pool of Credits every **resource month**, counted from your subscription start date. Unused Credits reset on that monthly date and **do not roll over** (see `https://vibe42.ai/#pricing` and Console for each plan's monthly amount).

What counts—and what is completely free

Only application data relayed through a Vibe42 access point (AP) uses Credits.

- **Conversion rate**: `64 MiB = 1 Credit`.
- **Two-leg relay counting**: When traffic goes through an AP relay, the AP counts both the incoming leg (from your device into the AP) and the outgoing leg (from the AP to your computer).
  - *Example*: Sending **64 MiB** of files or terminal text from your device through an AP to your computer (`App → AP → Computer`) counts `64 MiB` on the `App → AP` leg (1 Credit) plus `64 MiB` on the `AP → Computer` leg (1 Credit), for a total of **2 Credits**.

The following traffic is **always free (0 Credits)**:

- **Direct P2P connections** between native apps, the Vibe42 background service, and the `v42` CLI
- Encryption and transport protocol headers
- Connection setup and NAT hole-punching (direct-connection discovery) probes
- Control sessions (`control` status channel)
- Console management requests

Which features consume Credits when relayed

When a connection runs through an AP relay, all four data features are counted using the same rule:

1. **Terminal sessions** (text input and output, which uses very little data)
2. **File transfers and previews** (Files)
3. **Remote Desktop video and input streams** (Remote Desktop controls)
4. **CLI port forwarding** (`v42 forward`)

Because live screen video and large file transfers move far more data than terminal text, **Remote Desktop** and **large file transfers** use Credits the fastest when relayed.

Checking whether you are direct or relayed

- **In the app**: Open **Settings → Network diagnostics** and check whether each channel on your computer shows **Direct QUIC** / **Direct TCP** (free direct connection) or **Relayed** (metered relay).
- **From the CLI**: Run `v42 network [computer]` to see the live connection path in JSON (see How connections work and CLI commands).

Why browsers and corporate networks always use relay

- **Browsers**: Web browsers cannot open Vibe42's direct peer-to-peer connections, so `https://app.vibe42.ai` **always uses an AP relay**.
- **Strict firewalls and routers**: Office, campus, or hotel networks that only allow outbound web traffic (TCP port `443`) or use strict NAT (network address translation) often block direct connections, keeping native apps on the relay.
- **Force relay**: Check that neither your device nor your computer has **Force relay** turned on.

What happens when Credits run out

If your Team uses all of its monthly Credits, the app shows **This month's Credits are used up**:

- **Relayed data connections pause** (this stops browser sessions and any native sessions that need an AP relay).
- **A basic control connection stays open** so your apps notice right away when the plan is renewed or upgraded.
- **Direct P2P connections keep working** as normal.
- Relayed connections resume automatically on your monthly reset date, or right away when the Team Owner renews or upgrades the plan in Console.

Where to view your usage

- **In the app**: Open **Settings → Usage** to see **Used / total**, **Remaining**, and the **Resets on** date for this month.
- **In Console**: Team Owners and Admins can also check usage in Console.

Practical ways to save Credits

- **Use native apps or the CLI instead of the browser**: Native apps on macOS, Linux, iOS, and Android switch to free direct P2P connections whenever your network allows it.
- **Lower Remote Desktop quality on relayed connections**: Pick a lower quality preset when viewing your screen over a relay, and close the Remote Desktop view when you are done.
- **Transfer large files over direct connections**: Wait until **Network diagnostics** shows your `file` channel as **Direct QUIC** or **Direct TCP** before moving large files.

Trial, promo codes and refunds

Check eligibility rules for the 3-day free trial, how to redeem promo codes in Console, refund policies, and checkout error messages.

`vibe42.ai/docs/account/trial-redeem-refunds`

Use this page when starting a 3-day free trial, redeeming a promo code for your Team, checking whether a charge qualifies for a refund, or looking up a checkout error message.

3-day free trial

Eligible Team Owners can start a **3-day free trial** at `https://vibe42.ai/console/checkout?resource=access`:

- **Card required**: You must add a valid payment card. Each card can be used for a free trial **only once**.
- **One trial per Owner and Team**: You cannot start a free trial if either the Team or its Owner has already used a trial or made a paid purchase.
- **Automatic billing at trial end**: Unless you cancel, your card is charged automatically for the first period of your selected plan when the 3-day trial ends. You can also pay manually during the trial to switch to a paid subscription right away.
- **Canceling during the trial**: You can cancel anytime from Console's billing page. After you cancel, your trial keeps working until the end of the 3-day window and then stops without charging your card.
- **Failed charge at trial end**: If the initial charge fails when the trial ends, the trial stops and the Team becomes inactive.

Promo and redeem codes

Redeem codes use the format `XXXX-XXXX-XXXX`. You can apply them at `https://vibe42.ai/console/redeem?code=XXXX-XXXX-XXXX`.

- **Team Owner only**: Only a Team's **Owner** can redeem a code. If you enter a code in the desktop or web app, it opens Console in your browser so you can confirm.
- **Four benefit types**: Depending on the promotion, a code provides one of four benefits:
  1. **Discounted subscription** (applies a discount when subscribing or renewing)
  2. **Extend your subscription** (adds time to an active plan)
  3. **Gift plan** (activates gifted plan access)

4. **Extend your subscription** or **Gift plan** (extends an active subscription if you have one, or activates a free period if you do not)

Refund policy summary

All refund requests are reviewed by our team. Email `support@vibe42.ai` from your account email address. Once approved, refunds are sent to the payment processor within **3–5 business days**.

- **First-purchase unused guarantee**: A direct first-time purchase qualifies for a **full refund** if you request it within **3 days** of the start of the first billing period **and** your total metered traffic is **under 1 MB** (treated as unused).
- **Trial conversion exclusion**: Subscriptions that convert from a free trial started on or after **2026-10-27** are **not** covered by the first-purchase 3-day refund guarantee.
- **Billing errors & plan upgrades**: Duplicate charges, billing mistakes on our side, and charge-new / refund-old adjustments during eligible immediate plan upgrades receive a **full refund**.
- **Other approved refunds**: Any other refund approved outside the full-refund cases above deducts **payment processor fees plus a 3% manual handling fee**.

> WARNING
>
> Filing a bank or card **chargeback** (payment dispute) instead of contacting `support@vibe42.ai` immediately suspends your account. For full legal terms, see `https://vibe42.ai/refunds` and `https://vibe42.ai/terms`.

Common checkout and trial messages

Message	What it means
A free trial isn't available for this team. Subscribe directly instead.	This Team or its Owner has already used a free trial or bought a plan before. Subscribe directly to activate the Team.
This payment method has already been used for a free trial	Each payment card can start only one free trial across Vibe42. Subscribe directly without a trial or use a different card.
Payment setup expired / Payment setup canceled	The card verification page timed out or was closed before finishing. Start checkout again.
Your payment is being confirmed. You can leave this page and return later; please don't start another payment.	The payment processor is still confirming your payment. Wait a moment and refresh Console—do not pay a second time.
Only the Team owner can make this change. Ask them to review the plan or payment.	You are signed in as an **Admin** or **Member**. Ask the Team **Owner** to start the trial, change plans, or redeem the code (see Plans and slots).

08 Platforms

macOS app

System requirements, LaunchAgent background service, permissions, bundled CLI, and updates for Vibe42 on macOS.

`vibe42.ai/docs/platforms/macos`

The Vibe42 macOS app works both as a device for controlling your remote machines and as a computer you can reach from elsewhere for terminals, coding agents, files, and Remote Desktop. Read this page for macOS requirements, background service behavior, permissions, and updates.

Requirements and download

- **Operating system**: macOS 14 (Sonoma) or later.
- **Architecture**: Separate `.dmg` installers are available for Apple Silicon (`arm64`) and Intel (`x86_64`) Macs. The download page at vibe42.ai/#downloads detects your chip automatically.
- **macOS 13 (Ventura)**: The desktop app requires macOS 14+. On macOS 13, use the one-line Server edition installer (the version with only the background service and no desktop app) to use terminals, coding agents, files, and the CLI (see Linux server and headless machines).

Installation and first launch

1. Download the `.dmg` and drag **Vibe42.app** into `Applications` .
2. Open the app, click **Sign in** to sign in through your browser, and click **Approve**. For the full setup guide, see Install Vibe42.
3. When asked whether to make this Mac remotely controllable, choose **Enable this computer** (or **Not now** if you only want to use this Mac to control other computers).
4. Keep Vibe42 enabled under **System Settings → General → Login Items & Extensions** so the Vibe42 background service stays online.

> **IMPORTANT**
>
> If `Vibe42.app` (or an earlier `Vibe X.app` installation) is already on your Mac, use the built-in updater instead of dragging a new `.dmg` over it.

macOS-specific behavior and settings

Built-in background service and LaunchAgent lifecycle

When you enable your Mac as a computer, Vibe42 installs the background service under `~/.v42` (older upgraded installations may use `~/.vibex`) and registers the user LaunchAgent `ai.vibecafe.v42.companion` (a macOS background startup item).

Closing the Vibe42 window or pressing `⌘Q` quits only the desktop window—**it does not stop the background service**. Your paired phone, tablet, or laptop can still reach your terminals and screen. To stop both the app window and the background service, choose **Stop Services and Quit** from the menu bar icon, or click **Quit completely** under **Settings → Network diagnostics**. Toggle **Launch at login** in settings to control whether the desktop window opens automatically when you log in to macOS.

Shared installation with the `v42` CLI

The macOS app puts the `v42` and `v42-companion` binaries inside `~/.v42/bin`. The bundled CLI shares the same directory and Ed25519 device key as the desktop app. Once you sign in to the macOS app, `v42` works in your local terminal without a second login or an extra device slot. See v42 CLI overview.

Remote Desktop permissions

Only the macOS app has the **Settings → Remote Desktop** panel. Use it to toggle **Allow other devices to control this Mac** and grant the macOS **Screen Recording** and **Accessibility** permissions. See Remote desktop: getting started.

Desktop notifications

The macOS app shows system notifications only while the desktop window is running. Events that happen while the app window is closed are not replayed when you reopen it. For alerts while the app is closed, pair an iPhone or Android device or link Telegram (see Notifications).

Updates

The macOS app checks for updates in the background and shows **Restart to update to …** when a new release is ready.

Every update downloads over HTTPS, verifies a SHA-256 checksum and an Ed25519 signature before installing, and checks that the background service starts cleanly afterward. For manual update commands and uninstall steps—including removing `~/.v42` with `v42-companion uninstall --purge`—see Updating and uninstalling.

Limitations and known behaviors

- **Periodic Screen Recording prompts (macOS 15+)**: macOS Sequoia and later periodically ask on the Mac itself to re-confirm the **Screen Recording** permission for background screen capture apps.
- **Pre-login boot screen**: Because `ai.vibecafe.v42.companion` runs as a user LaunchAgent, your Mac comes online in Vibe42 only after your macOS user account logs in once after a reboot. Unlocking an already logged-in lock screen works normally (see Unlocking the screen and opening apps).
- **Sleep and clamshell mode**: When macOS sleeps, network connections pause until the Mac wakes. Adjust your macOS Battery or Energy Saver settings if you use a closed MacBook as an unattended computer.

Logs and diagnostics

To find local app and background service logs or prepare diagnostics for support, see Logs and diagnostics.

Linux desktop app

AppImage requirements, systemd user service setup, optional packages for tray, previews, input methods, and codecs on Linux desktops.

vibe42.ai/docs/platforms/linux-desktop

The Vibe42 Linux desktop app comes as an AppImage (a single-file Linux application) for `x86_64` and `arm64` that works both as a GPU-accelerated client and as a manager for the local Vibe42 background service. Read this page for system requirements, optional packages for tray icons, file previews, X11 input methods, and video codecs, and how updates work.

Requirements and download

- **Architecture**: `x86_64` or `arm64` (aarch64).
- **Distribution baseline**: Built on Ubuntu 24.04, requiring `glibc 2.39+`, `GTK 4.14+`, and `libadwaita 1.5+`. Works out of the box on Ubuntu 24.04 LTS, Debian 13, Fedora 40, and newer releases.
- **Required runtime services**: FUSE (`fuse3`, a Linux component for mounting filesystems, unless launched with `--appimage-extract-and-run`) and an active `systemd` user session.
- **Download**: Get `Vibe42-<version>-x86_64-unknown-linux-gnu.AppImage` (on arm64, `Vibe42-<version>-aarch64-unknown-linux-gnu.AppImage`) and its `.sha256` checksum file from vibe42.ai/#downloads.

If your Linux machine runs an older glibc distribution (such as Ubuntu 22.04 or Debian 12), install the Server edition with the one-line installer instead (Linux server and headless machines) and control machines through the Web app.

Installation and first launch

1. Verify the download against the `.sha256` file, make the AppImage executable, and run it:

```
sha256sum -c Vibe42-*.AppImage.sha256
chmod +x Vibe42-*.AppImage
./Vibe42-*.AppImage
```

2. On first launch, Vibe42 registers a `.desktop` entry, application icons, and the `vibex://` URI handler under `~/.local/share` so browser sign-in callbacks open the app. If you move the AppImage file to another folder later, launching it from the new location updates the `.desktop` entry automatically. For the account sign-in steps, see Install Vibe42.

Linux-specific behavior and optional dependencies

Background service management via systemd

When started inside a `systemd` user session, the Linux app installs and manages the background service under `~/.v42/` using the `v42-companion.service` user unit—the same path and service used by the one-line installer. If you already installed Vibe42 with the one-line command, the desktop app adopts that installation and keeps it updated. If you launch the AppImage without an active `systemd` user session, it runs only as a controlling client and does not manage the background service.

Optional system integrations

Several desktop features rely on packages installed on the host system outside the AppImage:

Feature	Host dependency	Fallback when missing
System tray icon	`StatusNotifierItem` watcher (on GNOME, install the AppIndicator extension)	Closing the main window quits the desktop app immediately
Rich file previews (code, Markdown, HTML, PDF)	`bubblewrap` and `xdg-dbus-proxy` (required by the bundled WebKitGTK sandbox)	Code, Markdown, and HTML fall back to plain text; PDFs offer download only
X11 CJK / IME input	`fcitx5` or `ibus` GTK4 IM module on the host (Wayland works natively without extra GTK4 modules)	Input method composition is unavailable inside X11 windows
Remote Desktop decoding (controlling)	VA-API or NVIDIA NVDEC drivers	Downloads and verifies Cisco OpenH264 on demand
Remote Desktop encoding (controlled)	System `ffmpeg` with VA-API, NVENC, or V4L2	Run `v42-companion codec enable` or enable Cisco OpenH264 in settings

Updates

The Linux app checks for updates in the background. When a system tray is available, the tray menu also provides **Check for Updates** and **Restart to update to …**. Verified AppImage updates are staged locally and swapped in on the next launch while keeping `v42-companion.service` on the same version. For uninstall steps—including removing `~/.local/share` desktop files and running `v42-companion uninstall`—see Updating and uninstalling.

Limitations and known issues

- **Wayland Remote Desktop compositors**: Controlling a Linux machine over Wayland is supported on KDE Plasma 6, GNOME, and Hyprland (KDE and GNOME prompt once on the local screen to allow remote control; see Remote desktop: getting started). Other Wayland compositors are not supported as controlled desktops.
- **No FUSE environments**: In restricted environments without `/dev/fuse`, run the binary with `--appimage-extract-and-run`.
- **Window close without a tray**: On unmodified GNOME without a StatusNotifierItem tray extension, closing the window exits the desktop app (though `v42-companion.service` continues running in your systemd user session).

Logs and diagnostics

To view background service logs with `v42-companion logs -f` or find desktop app log files, see Logs and diagnostics.

Linux server and headless machines

Install and manage the Vibe42 Server edition on Linux servers, containers, ARM boards, and macOS 13 with systemd or supervisor mode.

vibe42.ai/docs/platforms/linux-server

The Vibe42 Server edition (the version with only the background service and no desktop app) installs the Vibe42 background service and the `v42` CLI with a single shell command—no desktop environment or specific glibc version required. Read this page when setting up cloud VMs, home servers, Docker or dev containers, ARM64 boards, WSL on Windows, or macOS 13 machines, and when configuring `systemd` lingering or supervisor startup.

Requirements and one-line installer

- **Supported platforms**:
  - Linux `x86_64` and `arm64` — statically linked binaries that run on any Linux distribution (Ubuntu, Debian, Alpine, RHEL/Rocky, Arch, NixOS, and container base images).
  - macOS (`arm64` and `x86_64`), including macOS 13 where the desktop app is unavailable.
- **Prerequisites**: `curl`, `tar`, and `tmux` 3.0 or newer.
- **Install command**:

```
curl -fsSL https://get.vibe42.app/install.sh | sh
```

> **IMPORTANT**
>
> Run the installer as the OS user who owns your projects and terminals—do not run it with `sudo` or as `root` unless your entire environment is a root-only container. Do not run the one-line installer on a machine that already has the Vibe42 desktop app installed; the installer detects the desktop app and refuses to overwrite it.

First-run pairing

The installer verifies the SHA-256 checksum of the downloaded archive, installs binaries under `~/.v42` (older installations may use `~/.vibex`), and adds `~/.v42/bin` to your shell `PATH`. Both `v42-companion` (with a `vibex-companion` compatibility alias) and `v42` are installed together.

At the end of installation, the terminal prints an 8-character pairing code (`Pairing code: ABCD-EFGH`) and an approval URL (`https://vibe42.ai/console/pair?code=ABCD-EFGH`), waiting up to 10 minutes for confirmation. Open the URL in your browser or enter the code under **Computers & Devices → Connect a new computer or server** in any signed-in Vibe42 app. For full walkthroughs, see Install Vibe42 and Connect a Linux server.

Service modes and server administration

`systemd` vs. `supervisor` mode

The installer picks a service manager automatically, or you can override it with `VIBEX_SERVICE_MODE=systemd|supervisor|auto`:

- **`systemd` mode** (default on systemd hosts): Registers and starts the `v42-companion.service` user unit along with `v42-watchdog.timer` to keep the background service healthy.
- **`supervisor` mode** (automatic when systemd user sessions are absent, such as in most containers): Runs a lightweight user-space supervisor process and adds a managed autostart block to your shell rc files so the background service starts whenever an interactive shell opens. Pass `VIBEX_SERVICE_MODE=supervisor` to force this mode.

Keeping SSH-only servers online after logout

On many Linux servers, `systemd` stops a user's `systemd --user` instance as soon as their last SSH session closes. Enable user lingering once so `v42-companion.service` stays online after you log out and starts automatically on boot:

```
sudo loginctl enable-linger "$USER"
```

Everyday management commands

```
~/.v42/bin/v42-companion status
~/.v42/bin/v42-companion status --json
~/.v42/bin/v42-companion logs -f
~/.v42/bin/v42-companion start
~/.v42/bin/v42-companion stop
~/.v42/bin/v42-companion update --check
```

WSL on Windows

On a Windows PC, you can run the exact same one-line installer inside WSL (requires WSL 2 and a Linux distribution such as Ubuntu with `curl`, `tar`, and `tmux` 3.0+; if the distribution does not have a systemd user session enabled, the installer automatically switches to supervisor mode). The computer stays online only while WSL is running: when Windows shuts down, sleeps, or stops WSL (for example via `wsl --shutdown`), it shows as offline in Vibe42, and when you reopen the WSL terminal, the background service recovers automatically via the systemd user session or supervisor autostart. See Windows for full details.

Updates and in-place repair

Self-managed Server edition installations check for updates automatically once a day around **07:00 local server time**. You can also check or run updates manually with `v42-companion update [--check] [--json]`, or re-run `curl -fsSL https://get.vibe42.app/install.sh | sh` to upgrade or repair an existing installation in place while keeping its identity, pairing, and computer slot.

Every update runs as an atomic transaction: **download → signature verification → preflight check (disk space, terminal environment, and active transfers) → idle wait → atomic binary switch → health check**. The background service never restarts while an active terminal is open in a client or a file transfer is in

progress. Updates do not roll back automatically. For uninstallation (`uninstall` vs. `uninstall --purge`), see Updating and uninstalling.

Limitations

- **Headless machines and Remote Desktop**: Headless servers without an X11 or supported Wayland desktop session do not offer Remote Desktop; terminals, coding agents, file transfers, isolated copies, and `v42 forward` all work normally.
- **Container cold starts**: In `supervisor` mode inside a container, if no login shell runs at container startup, call `~/.v42/bin/v42-companion start` from your container entrypoint script.

Logs and diagnostics

Stream live logs with `~/.v42/bin/v42-companion logs -f`. For log paths and what to redact before sharing logs, see Logs and diagnostics.

---

iPhone and iPad

Pair Vibe42 on iOS and iPadOS 17+, receive push notifications with zero background battery drain, and use iPad multi-window and keyboards.

`vibe42.ai/docs/platforms/ios`

The Vibe42 app for iPhone and iPad gives you a Metal-accelerated terminal, coding agent alerts, file browsing, and Remote Desktop without draining your battery in the background. Read this page for iOS and iPadOS requirements, QR pairing, iPad multi-window and hardware keyboard support, and platform limits.

Requirements and download

- **Operating system**: iOS 17 or later / iPadOS 17 or later.
- **Download**: Install from the App Store at apps.apple.com/us/app/vibe42/id6792229966.
- **Role**: iPhone and iPad act only as controlling **Devices** for your paired Mac and Linux computers.

Pairing and permissions

Vibe42 on iOS never asks you to type your account password on a phone keyboard:

1. On an already signed-in device (a desktop app or the web app at `https://app.vibe42.ai`), open **Computers & Devices → Downloads and setup → Pair a phone or tablet**.
2. On your iPhone or iPad, tap **Scan to pair** to scan the QR code or enter the 8-character pairing code.
3. Approve the request on your original device.

For the full setup flow—including pairing when you only have your phone with you via the mobile browser—see Pair a device and start your first session.

System permissions are requested only when a feature needs them:

- **Camera**: Requested only when you tap to scan a pairing QR code or take a photo to upload into a terminal.
- **Notifications**: Requested after pairing finishes so your device can receive push alerts when agents finish or need input.

iOS and iPadOS-specific behavior

Zero-background residency and APNs alerts

Vibe42 keeps **zero background connections** on iOS and iPadOS. As soon as you lock your screen or switch to another app, Vibe42 closes its network connections cleanly instead of running in the background or draining battery. Your shells and coding agents keep running on your computers, and state changes (**Done**, **Needs input**, **Error**) arrive as Apple Push Notification service (APNs) alerts. Tapping a notification reconnects and opens that computer and window. See Sessions and Notifications.

GPU rendering and touch terminal controls

Terminals render natively through `libghostty-vt` and Metal, matching your display's refresh rate on ProMotion and standard screens. On touchscreens, you get a **Send to terminal** command bar, a draggable D-pad, tap-or-double-tap-to-lock modifier keys, pinch-to-zoom font sizing, and synced custom shortcut buttons. See Mobile input.

iPad multi-window and hardware keyboards

On iPadOS, Vibe42 works with system multitasking:

- **Multiple windows**: Use Split View or Stage Manager to open multiple Vibe42 windows side by side—for example, keeping Claude Code in one window and a Remote Desktop view in another. (Opening a view in a new window is available on iPad, not on iPhone.)
- **External keyboard shortcuts**: Attach a Magic Keyboard or Bluetooth keyboard for full physical key support, including the Kitty keyboard protocol. Hold down the `⌘` key on your iPad keyboard to view the shortcut cheat sheet (see Shortcuts).

Updates

Updates arrive through the App Store. If an older app build falls below the server's minimum supported version, Vibe42 prompts you to update from the App Store before reconnecting. To unpair a phone and free its device slot, remove it from Console or another signed-in device; see Updating and uninstalling.

Limitations

- **No in-app pricing or billing UI**: The iOS and iPadOS app does not show plan pricing, checkout, or subscription management screens. Manage trials, plans, slots, and Credits in your browser via Console (see Plans and slots).
- **Single-window on iPhone**: Opening items in a separate OS window is an iPadOS-only feature and is not available on iPhone.

Logs and crash reports

For connection diagnostics and general troubleshooting, see Logs and diagnostics. To find an iOS or iPadOS crash report for support, open the Apple **Settings** app on your device and go to **Privacy & Security → Analytics & Improvements → Analytics Data**.

Android

Pair Vibe42 on Android 11+, get FCM push notifications with zero background residency, use foldable and tablet layouts, and collect logs with adb.

`vibe42.ai/docs/platforms/android`

The Vibe42 app for Android turns your phone, foldable, or tablet into a controlling device for your Mac and Linux computers: a GPU-rendered terminal, coding agent status with push notifications, file browsing, and Remote Desktop. Read this page for Android requirements, pairing and permissions, Android-specific behavior, updates, and how to collect logs.

Requirements and download

- **Operating system**: Android 11 or later.
- **Download**: Install from Google Play at play.google.com/store/apps/details?id=ai.vibecafe.x. The download page at vibe42.ai/#downloads links there too.
- **Role**: Android devices act only as controlling **Devices**. They cannot run the Vibe42 background service or be controlled as a computer.

Pairing and permissions

The Android app never asks for your account password. Pair it from a device that is already signed in:

1. On a desktop app or at `https://app.vibe42.ai`, open **Computers & Devices → Downloads and setup → Pair a phone or tablet** to show a QR code and an 8-character code.
2. On the phone, tap **Scan to pair** and scan the QR code, or type the code.
3. Back on the original device, check the request and approve it.

Codes expire after 10 minutes, work once, and pair one device at a time. If your phone is the only device you have with you, sign in to `https://app.vibe42.ai` in the phone's browser first and use that session to pair the app. See Pair a device and start your first session.

Permissions are requested when a feature needs them: camera access only when you scan a pairing QR code or take a photo to upload into a terminal, and notification permission after pairing completes so the app can receive push alerts.

Android-specific behavior

Zero background residency and FCM

The app keeps no background service and no background network connection. When you switch away or lock the phone, it closes its connections; your shells and agents keep running on the computer. Status changes to **Done**, **Needs input**, or **Error** reach you as push notifications through Firebase Cloud Messaging (FCM). Tapping a notification reconnects and opens that computer's window. Returning to the foreground reconnects within a few seconds. See Notifications and Sessions, windows and Recents.

Rendering and input

Terminals render with `libghostty-vt` on a `TextureView`, matching the display's refresh rate so scrolling stays smooth on high-refresh-rate screens. Touch input uses the same tools as iOS: the command bar (**Send to terminal**), a draggable D-pad, tap-once or double-tap-to-lock modifier keys, long-press selection, pinch-to-zoom font sizing, three soft-keyboard avoidance modes, and custom shortcut buttons that sync across your devices. See Typing on a phone or tablet.

Photos from the camera, gallery, or file picker upload to the computer and insert the file path into the terminal. Images over 4 MiB and HEIC photos convert to JPEG automatically.

Foldables, tablets, and desktop mode

The layout adapts to foldable inner screens and tablet widths. **Open in New Window** is available only when Android runs in desktop mode; on a standard phone layout it is not shown. With a hardware keyboard attached, the shortcuts in the **other** column of Keyboard shortcuts apply.

Connections and Credits

The Android app opens a direct peer-to-peer connection to the computer whenever the networks allow it and falls back to the nearest relay server otherwise. Direct traffic is free; relayed bytes count toward your Team's Credits. Check the current connection path under **Settings → Network diagnostics**. See How connections work and Credits and usage.

Updates

Updates arrive through Google Play. If a build is older than the server's minimum supported version, the app shows **This version is no longer supported. Go to vibe42.ai to download the latest version.** and will not connect until you update from Google Play. To remove a phone from your Team and free its device slot, sign it out from Console or another signed-in device. See Updating and uninstalling.

Limitations

- **No pricing or checkout in the app**: The Android app does not show plan prices, purchases, or subscription management. Use Console in a browser instead (see Plans and slots).
- **Controller only**: Android cannot host terminals or be reached by other devices.
- **Multi-window**: Separate app windows are available only in desktop mode.

Logs and crash reports

App logs stay in the app's private storage and cannot be exported from inside the app. For support requests, note the time, the app version shown in settings, and a screenshot. If the app crashed and you can connect the phone to a computer, enable USB debugging and filter the crash tag:

```
adb logcat -s VibexPanic:*
```

See Collecting logs for the full checklist, including what to redact before sending logs to support.

Web app and PWA

Use Vibe42 in any modern browser at app.vibe42.ai, install it as a PWA, and understand relay connections, browser shortcuts, and WebCodecs requirements.

`vibe42.ai/docs/platforms/web`

The web app at `https://app.vibe42.ai` gives you terminals, agent status, files, and Remote Desktop on any computer with a modern browser, including Windows machines and borrowed laptops, with nothing to install. Read this page when you want to work from a browser, install the web app as a PWA (a web app you can install to your desktop or home screen), or see what differs from the native apps.

Requirements and access

- **Browser**: Any modern browser. WebGL is used for terminal rendering, with an automatic fallback to Canvas 2D when WebGL is unavailable. Remote Desktop requires the **WebCodecs** API.
- **Sign-in**: Open `https://app.vibe42.ai` and sign in with your `vibe42.ai` account (email and password, or **Continue with GitHub** / **Continue with Google**). Unverified accounts cannot sign in. See Account and sign-in.
- **Device slot**: Each signed-in browser uses one device slot on your Team; signing in again on the same browser does not use a second slot. Signing in to Console does not use a slot. See Computers, devices and teams.

The web app is a controlling device only. Your Mac or Linux computer still needs Vibe42 installed; see Install on your computer.

Installing as a PWA

Use your browser's install option (such as **Install app** or **Add to Home Screen**) to run the web app as a standalone window with its own icon. The PWA works just like the browser tab: same sign-in, same device slot, and same relayed connections.

On a phone without the native app, the web app is also the fastest way to get started: sign in at `https://app.vibe42.ai` in the mobile browser, then open **Computers & Devices → Downloads and setup → Pair a phone or tablet** from that session to pair the native app. See Pair a device and start your first session.

Browser-specific behavior

Always relayed

Browsers cannot establish direct peer-to-peer connections, so every web session runs through the nearest relay server. Terminal, file, and Remote Desktop bytes sent through the relay count toward your Team's Credits, whereas native apps use no Credits once a direct connection is established. If you transfer large files or stream Remote Desktop for long periods, use a native app when possible. See How connections work and Credits and usage.

Terminal engine

The web terminal runs the same Ghostty core as the native apps, compiled to WebAssembly (`ghostty-vt`) and rendered with WebGL. Inline images use the Kitty graphics protocol, with up to 5 images shown per pane (one split of a terminal window) in the browser. Input method composition appears inside the terminal before text is committed. See Terminal engine.

Keyboard shortcuts

Browsers reserve certain key combinations, so the web app uses alternatives: for example **New Window** is `⌥⌘N` on Apple platforms and `Alt+Shift+N` elsewhere, and **New Terminal** is `⌥⌘T` or `Alt+Shift+T` . In Firefox, `Ctrl+Shift+P` is reserved by the browser, so global **Search** has no keyboard shortcut there. See the full table in Keyboard shortcuts.

Notifications

The web app shows in-app notifications while the tab or PWA window is open. It does not deliver notifications when closed. For alerts while you are away, pair a phone or link Telegram under **Notifications → Message Channels**; see Notifications.

Remote Desktop

Remote Desktop in the browser requires WebCodecs support. Quality presets, display switching, and plain-text clipboard sync work the same as in the native apps. See Remote desktop: getting started.

Updates

Reload the page to get the latest version. If an open tab is too old for the server, the app shows **This version is out of date. Refresh the page to get the latest version.**; refresh the page to continue.

Limitations

- **No direct connections**: Every session is relayed and counts toward Credits, as described above.
- **No port forwarding**: `v42 forward` is a CLI feature; the web app does not forward ports. See Previewing results and forwarding ports.
- **Browser clipboard rules apply**: OSC 52 copies (clipboard writes initiated by remote terminal programs) go to your local browser clipboard, subject to the browser's clipboard permissions.
- **Media previews read whole files**: Audio and video previews in Files load the entire file in the browser, unlike the native apps' partial range reads.

Logs and diagnostics

Open your browser's developer tools and check the **Console** tab for errors and the **Network** tab for failed requests or WebSocket connections. Combine this with **Settings → Network diagnostics** inside the app. See Collecting logs.

Windows

Make a Windows PC a controlled computer with the Server edition in WSL, or control other computers from a browser. Native Windows support is coming soon.

`vibe42.ai/docs/platforms/windows`

A Windows PC can act as a remotely controlled computer by running the Vibe42 Server edition inside WSL (Windows Subsystem for Linux), while native Windows support is coming soon. You can also control your other computers from Windows through the web app in any modern browser.

Use a Windows PC as a controlled computer via WSL

Set up the Vibe42 background service inside WSL 2 using the same one-line installer as a Linux server:

1. **Install WSL 2 and a Linux distribution** (such as Ubuntu): In Windows PowerShell or Command Prompt, run `wsl --install`.
2. **Open your Linux distribution's terminal** and make sure `curl`, `tar`, and `tmux` 3.0+ are installed.
3. **Run the one-line installer**:

   ```
   curl -fsSL https://get.vibe42.app/install.sh | sh
   ```

   If your distribution does not have a systemd user session enabled, the installer automatically switches to supervisor mode.
4. **Approve pairing**: Open the approval URL printed in the terminal in your browser, or enter the pairing code under **Computers & Devices → Connect a new computer or server** on any signed-in device.
5. **Connect from your devices**: Once approved, this computer appears in the app and is ready for your first session. See Install on your computer and Connect a Linux server for full setup details.

What works and what is not yet available

What you get through WSL:

- **Terminals, coding agents, and files**: Access persistent terminals, coding agents, and files inside your WSL environment just like any other Linux computer. Your Windows drives are mounted under paths such as `/mnt/c` in WSL and can be accessed in both the terminal and the files panel.
- **Port forwarding and CLI**: Forward services running inside WSL with `v42 forward` (see Preview and port forwarding), and use the `v42` CLI installed alongside the Server edition inside WSL (it is not a native Windows binary; see CLI overview).
- **Agent status and push notifications**: Receive agent status updates and push notifications as on any other Linux computer.

- **Online status**: The computer is online only while WSL is running. When Windows shuts down, sleeps, or stops WSL (for example via `wsl --shutdown`), the computer shows as offline in Vibe42; when you reopen the WSL terminal, the background service recovers automatically via the systemd user session or supervisor autostart.

What is not yet available on Windows:

- **No Windows screen in Remote Desktop**: You cannot view or control the Windows desktop screen with Remote Desktop (Remote Desktop only supports macOS and Linux desktops as controlled computers).
- **No native Windows app yet**: Connecting from this Windows PC to control other computers still uses the browser web app, which always connects through a relay.

What you can do from Windows

To control your computers from Windows, open `https://app.vibe42.ai` in any modern browser and sign in. You get the full web app: terminals and coding agents on your Mac and Linux computers, agent status, files, and Remote Desktop (your browser needs WebCodecs for Remote Desktop). You can also install the web app as a PWA (a web app that opens in its own standalone window). Everything in Web app and PWA applies:

- Browser sessions always run through a relay server, so their traffic counts toward your Team's Credits (see Credits and usage).
- Shortcuts follow the **other** column in Keyboard shortcuts, using the web key bindings such as `Alt+Shift+N` for **New Window**.
- Each signed-in browser uses one device slot.

Native Windows version

A native Windows version (including the desktop app) is coming soon. See What is Vibe42 for the full platform matrix.

Updating and uninstalling

How each Vibe42 platform updates, how update signatures and health checks work, what version-unsupported messages mean, and how to uninstall cleanly.

`vibe42.ai/docs/platforms/updates-and-uninstall`

Read this page when you want to know how a Vibe42 app or the Vibe42 background service gets updated, what to do when a client says its version is no longer supported, or how to remove Vibe42 from a machine cleanly.

How updates work on each platform

Platform	How it updates
macOS app	Checks for updates automatically and shows **Restart to update to …** when a new version is ready.

Platform	How it updates
Linux desktop app	Checks automatically; the tray menu also offers **Check for Updates** and **Restart to update to …**. The verified new version is staged and swapped in on the next launch. The background service it manages is updated in sync.
Server edition (one-line installer)	Checks around **07:00 local time** every day. Update manually with `v42-companion update`, or re-run the install command, which upgrades or repairs in place and keeps the pairing.
iPhone and iPad	App Store.
Android	Google Play.
Web app	Reload the page.

```
~/.v42/bin/v42-companion update --check
~/.v42/bin/v42-companion update
~/.v42/bin/v42-companion update --json
```

The `v42` CLI shares the installation and version of the background service it ships with, so updating the app or background service updates the CLI too.

What an update checks

Desktop apps download updates over HTTPS only and verify both a SHA-256 digest and an Ed25519 signature before installing. After the swap, the bundled background service must pass a health check.

Server edition updates run as a single transaction:

1. Download
2. Signature verification
3. Preflight: free disk space, the terminal environment, and whether any terminal or file transfer is active
4. Wait until the background service is idle
5. Atomic switch to the new binary
6. Health check

While a terminal is open in a client or a file is transferring, the background service waits instead of restarting. There is no automatic rollback: if an update leaves the background service unhealthy, re-run the install command to repair it, or see Connection problems.

Updating never interrupts your running terminal sessions, and a repaired or upgraded installation keeps its identity, pairing, and slot.

"This version is no longer supported"

Old clients are rejected by server policy rather than by a network error. The message depends on the client:

Where	Message	What to do
Desktop and mobile apps	**This version is no longer supported. Go to vibe42.ai to download the latest version.**	Update the app from vibe42.ai/#downloads, the App Store, or Google Play.

Where	Message	What to do
Web app	**This version is out of date. Refresh the page to get the latest version.**	Reload the tab.
CLI and background service	**Upgrade the Vibe42 CLI to continue.**	Run `v42-companion update` or re-run the one-line install command.

Uninstalling

Uninstalling has two separate parts: removing the software from the machine, and removing the computer or device from your Team in Console so its slot is freed. Do both when retiring a machine.

macOS app

1. Choose **Quit completely** so the background service stops (closing the window or pressing `⌘Q` leaves it running).
2. Delete `Vibe42.app` from `Applications`.
3. To also delete the background service's identity, pairing, and local data under `~/.v42` (or `~/.vibex` on older installations), run the command below and confirm when prompted.
4. In Console, delete the computer under **Computers** and sign the app out under **Authorized devices**.

```
~/.v42/bin/v42-companion uninstall --purge
```

Linux desktop app

1. Quit the app.
2. Delete the AppImage and the desktop integration files it registered under `~/.local/share` (the `.desktop` entry, icons, and the `vibex://` handler).
3. Remove the background service with `v42-companion uninstall`, adding `--purge` to delete identity, pairing, and local data as well.
4. Delete the computer and sign out the device in Console.

```
~/.v42/bin/v42-companion uninstall --purge
```

Server edition

Run the uninstaller as the OS user who installed it. Without `--purge`, it removes the service and binaries but keeps your account link and local data, so reinstalling later restores the same computer. With `--purge`, it deletes the identity, pairing, and all local data. `--purge` asks for confirmation; in scripts, pass `--yes`.

```
~/.v42/bin/v42-companion uninstall
~/.v42/bin/v42-companion uninstall --purge
~/.v42/bin/v42-companion uninstall --purge --yes
```

In supervisor mode, the uninstaller also removes the autostart block it added to your shell rc files. In systemd mode, it removes `v42-companion.service` and `v42-watchdog.timer`.

iPhone, iPad, and Android

Delete the app from your device, then sign the device out in Console under **Authorized devices** (or from another signed-in device) to free its device slot.

Web app

Sign out in the web app, or sign the browser out from Console, to free its device slot. Uninstall the PWA (installed web app) using your browser's standard uninstall option.

What stays and what goes

- **Local terminal sessions**: Uninstalling the background service does not close running terminal sessions or the programs inside them. Close them yourself if you want them stopped.
- `uninstall` **without** `--purge`: Keeps the account link and local data; reinstalling on the machine restores it as the same computer.
- `uninstall --purge`: Deletes identity, pairing, and all local Vibe42 data. Reinstalling later requires pairing the machine as a new computer.
- **Deleting a computer in Console**: Removes the computer, its cloud Workspace records, and its memos from your Team. Files and directories on the machine are not touched. On a Mac with the desktop app, deleting the computer frees only the computer slot; sign out the app under **Authorized devices** to free its device slot too. See Computers, devices and teams.
- **Signing out a device**: The device must sign in or pair again before it can be used.

09 v42 CLI

v42 CLI overview

What the v42 CLI is for, where it comes from, how sign-in and exit codes work, and the JSON output format that scripts and agents use.

vibe42.ai/docs/cli/overview

`v42` is the Vibe42 command-line tool for scripts, CI jobs, and coding agents. It connects to the same computers, Workspaces, and terminals as the apps, outputs JSON by default, and returns predictable exit codes. Read this page before automating Vibe42, or when you want to check where `v42` is installed and how it signs in.

What it is for

The CLI covers the features that work without a graphical interface: opening and attaching to terminals, starting agents, copying and syncing files, forwarding ports, checking plan limits and network paths, and working inside isolated copies. It does not replace the apps—Remote Desktop, file previews, and notifications stay in the apps.

Like the apps, `v42` connects to the nearest access point (AP) first and switches to a direct peer-to-peer (P2P) connection to your computer whenever your network allows it. Direct connections cost 0 Credits; relayed traffic uses Credits (see Credits and usage). Pass `--force-relay` if you want to keep a command on the relay.

Where to get it

In most cases, `v42` is already on your machine:

Where you are	How `v42` gets there	Identity and device slot
Mac or Linux desktop app	Included with the app	Shares the app's installation and sign-in; uses no extra device slot
One-line Server installation	Installed next to `v42-companion` in `~/.v42/bin`	Shares the Vibe42 background service's installation
Any other machine	Run `v42 login --standalone`	Registers a separate CLI installation that uses 1 device slot

The `v42` version always matches the version of the background service it comes with. Update it the same way you update the desktop app or the background service (see Updating and uninstalling).

Signing in

You sign in through a web browser, never by pasting a token or API key:

```
v42 login
```

The CLI prints a `verify_url`, a `user_code`, and an `expires_at` timestamp, then waits until you approve. It does **not** open a browser automatically. Open `verify_url` yourself, check that the page titled **Authorize a device** shows your code, and approve it. If the desktop app is already signed in on the same machine, `v42` reuses that sign-in by default, so you do not need to run `v42 login`.

Options:

- `--standalone`: Register this CLI as its own installation instead of reusing the desktop app's sign-in. This uses 1 device slot on the Team.
- `--no-wait`: Print the verification details and exit right away (useful when a script sends the link and code to someone else to approve).
- `--poll`: Continue waiting for a sign-in you started earlier with `--no-wait`.

Run `v42 logout` to sign out, or `v42 whoami` to see which account and Team are active.

> **NOTE**
> Vibe42 does not support token or API-key sign-in. If an automated job needs its own sign-in, run `v42 login --standalone` once interactively on the machine where the job runs.

Output contract

By default, `v42` prints JSON so scripts do not need to parse human-readable text:

- On success, it writes `{"ok":true,"data":…}` to **stdout**.
- On failure, it writes `{"ok":false,"error":{"code":…,"message":…,"details":…}}` to **stderr**.
- Pass `--format text` for human-readable output (`--locale` or `V42_LOCALE` sets the language).
- Streaming commands—`attach`, `ws enter`, `term attach`, and `term new --attach`—write only raw PTY bytes (the direct input/output stream of the remote terminal) to stdout.

```
{"ok":true,"data":{"user_id":"…","install_id":"…","team_id":"…"}}
```

The values above are examples, while the top-level JSON structure stays the same across commands.

Exit codes

Code	Meaning
`0`	Success
`2`	Usage error: unknown command, invalid argument, or unknown `--agent` ID
`3`	Not signed in, or your sign-in session expired
`4`	Forbidden: HTTP 403, the Team is unavailable, or the network rejected the request
`5`	Local storage or file I/O error
`6`	Other client-side error (such as running `login` when already signed in)
`7`	Server error (5xx) or network failure
`130` / `143`	Stopped by `SIGINT` / `SIGTERM`

Streaming commands return the exit code of the remote program instead.

Global options and environment variables

Every option has a matching environment variable, which makes configuring CI jobs and agent environments straightforward:

Option	Environment variable	What it does
`--format json|text`	—	Output format (default `json`)
`--locale <tag>`	`V42_LOCALE`	Language for `--format text`
`--profile <name>`	`V42_PROFILE`	Use a named local profile with its own sign-in state
`--team <uuid>`	`V42_TEAM`	Run the command on a specific Team instead of the current Team
`--agent-name <s>`	`V42_AGENT_NAME`	Record the automation name in the User-Agent and audit logs (up to 128 bytes)
`--force-relay`	—	Keep this command on the AP relay
`--data-dir <path>`	`V42_DATA_DIR`	Directory where the CLI stores its local state
`--companion-root <path>`	`V42_COMPANION_ROOT`	Directory where the background service is installed

Set `V42_AGENT_NAME` whenever an agent runs `v42` so your Team's audit logs show which automation ran each action (see Let your agent use Vibe42).

A minimal example

```
v42 whoami
v42 resources
```

`whoami` confirms your sign-in and active Team. `resources` returns a single JSON object listing the Team's computers, your installations, Workspaces, and open terminals on each online computer. From there, pick a computer name and use the commands in Command reference or the scripts in Automation recipes.

Command reference

Every v42 command by group—session, Teams, Workspaces, terminals, files, quota, network, forwarding, isolated copies—with syntax, arguments, and result fields.

`vibe42.ai/docs/cli/commands`

This page lists every `v42` command, its arguments, and the fields in its JSON output. Read v42 CLI overview first for sign-in, the JSON output structure, exit codes, and global options. All commands below assume you are already signed in.

Conventions

- `<computer>` is a computer name returned by `v42 resources`.
- Remote file paths use `<Computer>:/absolute/path`. A path without a computer prefix refers to the local machine.
- Every command accepts global options (`--format`, `--team`, `--agent-name`, `--force-relay`, and others) before the subcommand.
- The result fields described below are inside the `data` object of the `{"ok":true,"data":…}` JSON response. Example values are illustrative.

Session

```
v42 login [--standalone] [--poll] [--no-wait]
v42 logout
v42 whoami
```

- `login` prints `verify_url`, `user_code`, and `expires_at`, then waits for you to approve the sign-in in a browser (the CLI does not open the browser automatically). `--standalone` registers a separate CLI installation that uses 1 device slot; `--no-wait` exits right after printing the details; `--poll` resumes waiting for that pending sign-in. If the desktop app on the same machine is already signed in, you do not need `login` — `v42` reuses the app's sign-in.
- `logout` signs out the CLI session.
- `whoami` shows the signed-in account and active Team.

Teams

```
v42 team list
v42 team use &lt;team_id&gt;
```

`team list` returns all Teams your account belongs to. `team use` sets the active Team for future commands. Pass `--team <uuid>` (or `V42_TEAM`) to run a single command against another Team without changing your saved default.

Workspaces

```
v42 ws list
v42 ws enter &lt;workspace&gt; [--stdio]
```

`ws list` returns the Team's Workspaces along with their computer and folder path. `ws enter` opens a new terminal in that Workspace's folder on its computer and connects your terminal to it. Because `ws enter` is a streaming command, stdout carries only raw PTY bytes (the terminal's input/output stream) and the exit code comes from the remote program. If the Workspace has terminal isolation turned on, the terminal starts inside a fresh isolated copy.

`--stdio` skips interactive terminal setup (it does not switch your local TTY—your local terminal device—into raw mode) and sends plain bytes over stdin and stdout. Use this when another program or script controls the session.

Terminals

```
v42 term new <computer> [--workspace <ws>] [--cwd <dir>] [--name <n>] [--agent <id>] [--command "
<cmd>"] [--attach [--stdio]]
v42 term attach <computer> <target> [--stdio]
v42 attach <computer> <target> [--stdio]
```

`term new` creates a terminal window on the computer—just like opening a new terminal in the apps—and returns:

```
{
  "computer_id": "…",
  "window_id": "@12",
  "session": "…",
  "target": "…:@12",
  "workspace_id": null,
  "terminal_isolation": false,
  "agent": "claude-code",
  "command": "claude"
}
```

- `--workspace <ws>` opens the terminal in a Workspace (starting in the Workspace folder, or in a new isolated copy when isolation is enabled, which sets `terminal_isolation` to `true`). Without a Workspace, `--cwd <dir>` sets the starting folder. `--name <n>` sets the window title.
- `--agent <id>` starts an agent from the built-in catalog through the Vibe42 background service so you get live status and notifications. Use the catalog ID rather than the display name: `claude-code`, `codex`, `gemini`, `opencode`, `cursor`, `copilot`, `pi`, `omp`, `devin`, `antigravity`, `cline`, `kimi`, `kiro`, `droid`, `amp`, `grok-build`, `aider`, `goose`, `crush`, or any other catalog ID. Passing an unknown ID fails with exit code `2` and lists all valid IDs. If Vibe42 on the computer is too old to launch agents, the command fails with error code `launch_unsupported`.
- `--command "<cmd>"` runs a single-line `<program> <args>` command instead of a catalog agent, following the same rules as custom entries in **Agents** (see Launching an agent).
- `--attach` connects to the window right after creating it. Add `--stdio` for a plain byte stream.

`term attach` (or its shorthand `attach`) connects your terminal to an existing window. `<target>` is the `target` string returned by `term new`—a session name optionally followed by `:@<window>`. Passing only the session name attaches to the session itself. Both are streaming commands: stdout carries the remote PTY stream, and the exit code comes from the remote program.

Files

```
v42 cp <src> <dst> [-r] [--restart]
v42 sync <src> <dst> [--delete] [--restart]
```

- `cp` copies files between local paths and `<Computer>:/absolute/path` locations in either direction. Pass `-r` to copy folders recursively.
- `sync` updates the destination folder so its contents match the source folder in one direction. Extra files in the destination are **kept** unless you pass `--delete`.

- `--restart` starts an interrupted transfer from the beginning instead of resuming where it left off.

The JSON result reports what changed, including `files_copied`, `files_skipped`, `bytes`, and `entries_deleted`. File transfers run over a dedicated `file` channel so they never slow down an open terminal (see How connections work). Traffic relayed through an AP uses Credits.

Quota, resources, and devices

```
v42 quota
v42 resources
v42 device revoke <install_id>
```

- `quota` returns the current Team's plan limits—its slots and Credits—so a script can check available capacity before pairing or starting tasks (see Plans and slots).
- `resources` returns a single JSON object containing the Team's computers, your installations, Workspaces, and the open terminals on each online computer (offline computers appear with `status: "offline"` and an empty terminal list).
- `device revoke <install_id>` revokes an installation's access to the current Team and frees its device slot. Get the `install_id` from `resources`. The command returns `{"team_id":…,"install_id":…,"revoked":true}`.

Network

```
v42 network [<computer>]
```

Without a `<computer>` argument, `network` reports the CLI's connection to the nearest AP: `ap_path`, `ap_round_trip_ms`, `ap_lanes`, and `force_relay`. With a `<computer>` argument, it also connects to that computer, waits briefly to see if a direct P2P path opens, and includes a `target` object:

```
{
  "ap_path": "relay",
  "ap_round_trip_ms": 23.4,
  "ap_lanes": ["…", "…", "…"],
  "force_relay": false,
  "target": {
    "computer_id": "…",
    "p2p_allowed": true,
    "p2p_connected": true,
    "lanes": ["…", "…", "…"]
  }
}
```

`p2p_connected` is `true` when at least one of the three channels (`control`, `terminal`, `file`) uses a direct connection. `lanes` lists the path for each channel in that order. This matches the information shown under **Settings → Network diagnostics** in the apps. Passing `--force-relay` sets `force_relay` to `true` and skips direct connections for that command.

Port forwarding

```
v42 forward <computer>:<remote-port> [<local-addr>:]<local-port>
```

Listens on a local TCP port and forwards incoming connections to `127.0.0.1:<remote-port>` on the remote computer over the same encrypted connection used for terminals:

- If you omit `<local-addr>`, the listener binds to `127.0.0.1` only. Binding to any other address prints a warning that other machines on your network can reach the port.
- Setting `<local-port>` to `0` lets the operating system pick a free port and prints the chosen address once ready.
- Up to **64** connections can be forwarded at the same time.
- By default, forwarding only reaches the computer's own loopback interface (`127.0.0.1`, granted by the `forward:loopback` permission included with terminal access). Reaching other hosts through that computer requires explicit permission on the computer.
- Relayed bytes count toward Credits when the connection runs through an AP. Audit logs never record the forwarded data itself.

The command runs until you stop it with `Ctrl+C`. See Previewing results and forwarding ports for common dev-server workflows.

Isolated copies

```
v42 copy status [--root <dir>]
v42 copy save [-m <message>]
v42 copy versions
v42 copy changes [--from <id>] [--to <id>]
v42 copy diff <path> [--from <id>] [--to <id>]
v42 copy apply <paths…> | --all
v42 copy push --branch <branch>
v42 copy rebuild-index [--root <dir>] --yes
v42 copy skill
```

The `copy` commands run locally on the computer itself, inside an isolated copy created by terminal isolation. They do not require signing in. Run them from a folder inside the copy, or set `V42_COPY` to the copy's path.

- `status` shows details for the current copy, or with `--root <dir>` lists all copies of a Workspace folder.
- `save` snapshots the current files as a new version (`-m` adds a note); `versions` lists saved versions.
- `changes` lists files modified since the starting state (along with whether they have been applied) or between two version IDs; `diff <path>` prints a unified diff for one file.
- `apply <paths…>` or `apply --all` writes modified files back to the main Workspace folder; `push --branch <branch>` pushes the copy's Git `HEAD` to the Workspace repository as a new branch.
- `rebuild-index --yes` clears and rebuilds the version index for a Workspace folder. Because this deletes the version history of every copy of that folder, `--yes` is required.
- `skill` prints a `SKILL.md` guide that teaches a coding agent how to work inside an isolated copy.

Automation recipes

Copy-ready v42 snippets: launch an agent terminal, sync build output back, check for a direct path with jq, forward a dev server, push from an isolated copy.

vibe42.ai/docs/cli/automation

These recipes show how to use `v42` in scripts, CI jobs, and coding agents. Each recipe notes which machine runs the commands and what setup it expects. For full command syntax and JSON fields, see Command reference; for the JSON response structure and exit codes, see v42 CLI overview.

Before you start

Every recipe assumes:

- `v42` is installed and signed in on the machine running the script (included with the desktop app, installed by the one-line Server installer, or signed in with `v42 login --standalone`).
- `my-mac` is the name of an online computer from `v42 resources` (replace it with your own computer name).
- `jq` is installed for reading JSON output. All `jq` paths start at `.data` because responses use the `{"ok":true,"data":…}` wrapper.
- When an agent runs these commands, `V42_AGENT_NAME` is set so audit logs show which automation ran each command.

Check the exit code rather than parsing output text to see whether a command succeeded:

```
out=$(v42 whoami)
status=$?
if [ "$status" -ne 0 ]; then
  echo "v42 failed with exit code $status" >&2
  exit 1
fi
```

1. Start an agent terminal and attach to it

Run this on your laptop or any machine with `v42`. It starts Claude Code in the `api` Workspace on `my-mac` and connects your terminal to it right away:

```
V42_AGENT_NAME=nightly-runner v42 term new my-mac --workspace api --agent claude-code --attach
```

Because `--attach` is a streaming command, stdout carries the remote PTY stream (the terminal's raw input/output channel) and the exit code comes from the remote program. To create the window first and attach later (or from another shell), read `target` from the JSON output:

```
target=$(v42 term new my-mac --workspace api --agent claude-code | jq -r '.data.target')
v42 term attach my-mac "$target"
```

The window keeps running on the computer even when nothing is attached, and it appears in the apps with live agent status like any other terminal. To run your own command or script instead of a catalog agent, pass `--command "<program> <args>"` instead of `--agent`.

2. Sync build output back to this machine

Run this on the machine where you want to receive the files. It copies the `dist` folder from `my-mac` into `./dist` locally while keeping any extra local files:

```
v42 sync my-mac:/Users/me/app/dist ./dist
```

Pass `--delete` to make `./dist` match the remote folder exactly, or `--restart` to start over from the beginning after an interrupted transfer:

```
v42 sync my-mac:/Users/me/app/dist ./dist --delete
```

The JSON output reports what changed, such as `.data.files_copied` and `.data.bytes`. To copy a single file or copy files in the other direction, use `cp`:

```
v42 cp ./config.toml my-mac:/Users/me/app/config.toml
```

3. Check for a direct path before moving a lot of data

Run this on any machine with `v42`. Because large transfers over an access point (AP) relay use Credits, you can have your script check for a direct peer-to-peer connection first:

```
if [ "$(v42 network my-mac | jq -r '.data.target.p2p_connected')" = "true" ]; then
  v42 sync my-mac:/Users/me/app/dist ./dist
else
  echo "relayed path only; skipping the large sync" >&2
fi
```

`p2p_connected` is `true` when at least one channel is connected directly. Check `.data.target.lanes` if you want to inspect the path for each individual channel (`control`, `terminal`, `file`). Web browsers always use a relay, while the CLI upgrades to a direct connection whenever the two networks allow it (see How connections work and Credits and usage).

4. Forward a dev server to your laptop

Run this on the machine where you want to open your browser. It forwards port `3000` on `my-mac` to `127.0.0.1:3000` on your local machine:

```
v42 forward my-mac:3000 3000
```

Open `http://127.0.0.1:3000` in your browser. The command keeps running until you press `Ctrl+C`. In a script, you can start it in the background and stop it when your checks finish:

```
v42 forward my-mac:3000 0 &
forward_pid=$!
# … run your checks against the port printed by the listener …
kill "$forward_pid"
```

Setting the local port to `0` lets the operating system pick an open port and prints the chosen address once ready. By default, the listener binds only to `127.0.0.1`, forwards only to the remote computer's loopback address (`127.0.0.1`), and handles up to 64 connections at a time.

5. Save and push work from an isolated copy

Run these commands **on the computer itself**, inside a terminal started in an isolated copy by terminal isolation. No sign-in is required. A typical sequence when an agent finishes a task looks like this:

```
v42 copy save -m "tests green"
v42 copy changes
v42 copy push --branch agent/fix-login
```

`save` records a new version snapshot, `changes` lists files changed since the starting state and whether they have been applied, and `push --branch` pushes the copy's Git `HEAD` as a new branch in the Workspace repository for review. If you want to write the changed files directly back into the main Workspace folder instead, run `v42 copy apply --all`. When running these commands outside the copy's folder, set `V42_COPY` to the copy's path.

> **TIP**
>
> `v42 copy skill` prints a `SKILL.md` guide explaining this workflow to a coding agent. Pass that output to your agent yourself—Vibe42 never modifies your agent configuration files (see Nothing injected, nothing locked in).

10 Troubleshooting

Connection problems

Fix offline computers, expired pairing codes, reconnect loops, relay fallbacks, and background service or authorization errors.

vibe42.ai/docs/troubleshooting/connection

Use this page when a computer shows as offline, pairing fails, the app keeps reconnecting, or Vibe42 shows an authorization, service, or version error. For how access point (AP) routing, direct connections, and reconnection work, see Connections.

Check network diagnostics first

Open settings and select **Network diagnostics** to see your nearest AP (access point, a regional routing server), current RTT (round-trip latency), and whether the `control`, `terminal`, and `file` channels for each computer use a `direct` or `relayed` path. From the CLI, check the active path with:

```
v42 network
```

> **TIP**
>
> On a Server edition computer (the headless version with only the Vibe42 background service), run `~/.v42/bin/v42-companion status` (or `~/.vibex/bin/v42-companion status` on older installs) to check whether the background service is running, paired, and reachable.

Pairing and sign-in

Pairing code expired or lost

An 8-character pairing code (`XXXX-XXXX`) expires after 10 minutes and pairs only one device or computer at a time.

Generate a new code and approve it right away. On a Server edition computer, print the active code (or JSON output with `pairing.code` and `pairing.approve_url`) and approve it at `https://vibe42.ai/console/pair` or in **Computers & Devices → Connect a new computer or server**:

```
~/.v42/bin/v42-companion status --json
```

For a phone or tablet, open **Computers & Devices → Downloads and setup → Pair a phone or tablet** on a signed-in device, then tap **Scan to pair** on your mobile device.

"Login or pairing did not finish in time (...)"

The one-line installer waits up to 10 minutes for you to approve the printed pairing code before exiting.

Run `~/.v42/bin/v42-companion status` to get a new pairing code and URL, then open the link and approve the computer within 10 minutes.

"Sign-in is not complete. Finish signing in in your system browser."

The desktop app opened an approval page in your default browser, but you have not clicked **Approve** yet or closed the browser tab early.

Switch to your system browser and click **Approve**. If you closed the tab, return to Vibe42 and click **Sign in** to start a new request.

"This device's authorization was revoked."

Someone signed out or removed this device from Console or another paired device. Revoked devices disconnect within 60 seconds and stop retrying automatically.

Sign in again on desktop or web, or open **Computers & Devices → Downloads and setup → Pair a phone or tablet** on an active device to pair your phone again. If a Server edition computer was removed, run `~/.v42/bin/v42-companion status` to get a new pairing code.

Offline computers and network paths

Computer shows offline or "The cloud service is temporarily unreachable. Check your network and try again."

The computer is asleep, offline, has its background service (`v42-companion`) stopped, or is behind a firewall blocking outbound HTTPS or QUIC (a UDP-based transport protocol) traffic to Vibe42 APs.

Wake the computer, check its internet connection, and inspect the live background service log on that machine (if the problem continues, see Collecting logs):

```
~/.v42/bin/v42-companion status
~/.v42/bin/v42-companion logs -f
```

App keeps reconnecting

Unstable Wi-Fi, network switches, or a manually locked distant AP can interrupt the session. Vibe42 tries to resume on the same AP within 30 seconds, then waits between 1 and 60 seconds between reconnect attempts.

Open **Network diagnostics** and clear any manual AP lock so Vibe42 can measure RTT and pick the closest region automatically. If direct UDP connections keep dropping on your router, turn on **Force relay** on the computer card or in settings.

Always relayed, never direct

You are connecting from a browser (the web app at `https://app.vibe42.ai` always uses AP relay), **Force relay** is turned on, or router NAT and UDP firewall rules on either network block peer-to-peer (P2P) direct connections.

Use the desktop app, mobile app, or CLI instead of a browser, and confirm that **Force relay** is turned off on both the device and the computer. Run `v42 network` to check whether channels switch to `direct`.

Cannot connect on a corporate network

Corporate firewalls and proxies often drop the UDP/QUIC traffic used for direct connections or block specific regional endpoints.

Turn on **Force relay** in settings or on the computer card to keep traffic on AP relay, confirm outbound HTTPS is allowed, and lock a reachable AP region in **Network diagnostics**.

"Device limit reached", "No computer slots left", or "This month's Credits are used up"

Quota and billing limits stop automatic retries immediately. Slot errors ("Device limit reached", "Computer limit reached", "No computer slots left", "No member seat available", "Can't connect to this team") mean your Team has no free slots. "This month's Credits are used up" pauses AP-relayed connections while leaving P2P direct connections working.

Remove an unused computer or device in Console, or ask the Team Owner to upgrade the plan (see Plans and slots and Computers, devices, and Teams). When Credits run out, connect over P2P direct or renew or upgrade in Console (see Credits and usage).

Background service, installer, and version errors

"Vibe42 is installed but the companion service is not running."

The background service stopped, or macOS background permission is turned off (`"Allow Vibe42 to run in the background in System Settings → General → Login Items & Extensions, then retry."`). If you see `"This computer's Vibe42 companion is managed by the Vibe42 App…"` , the desktop app manages the service and you do not need the command-line installer.

On macOS, allow Vibe42 under **System Settings → General → Login Items & Extensions** and retry. On Server edition installs, start the background service manually (only run `uninstall --purge` if you deleted the desktop app and want to switch to the Server edition):

```
~/.v42/bin/v42-companion start
```

"systemd user session bus is unavailable; log in normally or run loginctl enable-linger $USER"

On Linux servers accessed only over SSH, the user-level `systemd --user` manager is often inactive when linger is disabled.

Enable linger for your user, open a **new** SSH session, and run the installer again—or force supervisor mode without systemd:

```
sudo loginctl enable-linger "$USER"
# Reconnect in a new SSH session and re-run install.sh, or force supervisor mode:
curl -fsSL https://get.vibe42.app/install.sh | VIBEX_SERVICE_MODE=supervisor sh
```

"No safe tmux installation was found, or its version is incompatible." or "Cannot mount AppImage, please check your FUSE setup."

The Server edition requires `tmux` 3.0 or newer (`"tmux 3.0+ is required"`), and the Linux desktop AppImage requires FUSE (a Linux component for mounting filesystems) to run.

Install or upgrade `tmux` (3.0+) with your package manager and run the installer again. For the Linux AppImage, install `fuse3` (see Linux desktop) or run the binary with `--appimage-extract-and-run`.

"This version is no longer supported. Go to vibe42.ai to download the latest version."

Your app or background service is too old for the current protocol. On the web app you see "This version is out of date. Refresh the page to get the latest version."; on the CLI you see "Upgrade the Vibe42 CLI to continue.".

Download the latest desktop app from `https://vibe42.ai/#downloads`, update mobile apps from the App Store or Google Play, refresh the browser tab for the web app, or update the Server edition in place (see Updates and uninstall):

```
~/.v42/bin/v42-companion update
```

Not getting notifications

Troubleshoot missing mobile push notifications, skipped agent status alerts, desktop or Telegram delivery, and notification links.

vibe42.ai/docs/troubleshooting/notifications

Use this page when phone push alerts do not arrive, a coding agent finishes without an alert, desktop or Telegram messages are missing, or tapping an alert opens the home screen instead of the terminal. For how to configure event rules and delivery channels, see Notifications.

How delivery channels behave

Each channel has different runtime and permission requirements:

Channel	When alerts arrive	Offline behavior
Phone push (iOS / Android)	Anytime via Apple APNs or Google FCM (system push services) without running in the background	Delivered normally while the app is closed
Desktop app (macOS / Linux)	Only while the desktop app is running	Alerts missed while quit are not replayed
Web app (`app.vibe42.ai`)	In-app pop-ups while the tab is open	None when the tab is closed
Telegram	Anytime once your Telegram account is linked	Queued and delivered by Telegram

Common symptoms

No notifications on your phone at all

The Vibe42 mobile app asks for OS notification permission only after you pair the device. If you dismissed the prompt, turned off notifications in OS settings, turned on Focus or Do Not Disturb, or had the device authorization revoked, system pushes cannot appear.

Allow notifications for Vibe42 in your iPhone, iPad, or Android settings and confirm that Focus mode allows Vibe42. Then open settings → **Notifications** in Vibe42, check that the master toggle is on for your computers, and confirm **Phone push** is checked under **Deliver to**.

Missing notifications from only one computer

Notification settings are configured per computer. That computer may have its master switch, delivery target, or specific event types turned off—or the Vibe42 background service on that computer may be offline.

Open settings → **Notifications**, select that computer, and confirm its master toggle is on, your target channels are checked under **Deliver to**, and the relevant events are enabled under **Event types**. Also confirm the computer is online (see Connection problems).

Agent finished or stopped, but no push arrived

Typing an agent command directly in a plain shell skips status tracking from the background service unless the agent natively sends OSC 7501 (a terminal control sequence that reports status to Vibe42, supported in Claude Code 2.1.295+ and Pi 1.1.0+). Pushes also fire only when status **changes** to Done, Needs input, or Error, with a 3-second per-pane (one split of a terminal window) deduplication window and a 15-second summary wait—or the agent event toggles may be off.

Start the agent from step three of the new-terminal flow or from the CLI so status tracking is active (see Launching agents and Agent status):

```
v42 term new <computer> --agent claude-code
```

If you run Claude Code or Pi directly in any shell, upgrade to Claude Code 2.1.295+ or Pi 1.1.0+. Finally, open settings → **Notifications → Event types** and confirm **Agent finished**, **Agent needs input**, and **Agent error** are turned on.

Receiving too many notifications

High-frequency events—such as terminal bells, window or pane closes, and computer online or offline changes—are turned on for a busy computer. Vibe42 groups bursts above roughly 20 alerts per minute into a single summary, but steady activity below that rate still arrives one by one.

Open settings → **Notifications** for that computer and turn off noisy events under **Event types** (for example, keep **Agent needs input**, **Agent finished**, and **Agent error** while turning off terminal bell and pane-close events), or uncheck channels under **Deliver to** where you do not want alerts.

No notifications on the desktop app or web app

The desktop app shows system notifications only while it is running and does not replay missed alerts after you reopen it. The web app at `https://app.vibe42.ai` shows in-page pop-ups only while the tab is open. OS notification permissions or the web and desktop target under **Deliver to** may also be off.

Open settings → **Notifications → Deliver to** to confirm web and desktop delivery is enabled, and allow Vibe42 notifications in your macOS or Linux system settings. If you need alerts while your computer or browser is closed, enable **Phone push** or link Telegram.

No messages in Telegram

Your Vibe42 account is not linked to Telegram, the bot chat was stopped or muted in Telegram, or Telegram delivery is unchecked for that computer.

Open settings → **Notifications** to check or re-link your Telegram account and confirm the bot chat is active in Telegram. Then select each computer you want to monitor and enable Telegram under **Deliver to**.

Tapping a notification opens the home screen instead of the window

When you tap a notification, Vibe42 checks that the target event, computer, window, and pane still exist. If the window or pane already closed after the command exited, or the computer went offline or was unpaired, Vibe42 opens the home screen instead.

Check the home screen to confirm the computer is online. If a finished script closes its window right away, keep the shell open after the command finishes so the target pane is still there when you tap the notification (see Sessions).

Terminal and input problems

Fix keyboard shortcut conflicts, CJK input method issues, image paste limits, OSC 52 clipboard behavior, Linux file previews, and mobile keyboard overlap.

`vibe42.ai/docs/troubleshooting/terminal-and-input`

Use this page when keyboard shortcuts do not work, Chinese, Japanese, or Korean (CJK) input methods fail to type characters, pasted or inline images do not appear, clipboard copy does nothing, or Linux file previews and Remote Desktop fail to display.

Keyboard, input methods, and clipboard

Shortcuts are swallowed by the terminal or ignored in the browser

When focus is inside the terminal, Vibe42 sends keystrokes straight to your shell and running programs (such as `Ctrl+B`) and only intercepts app-specific shortcuts. In the web app, browser-reserved shortcuts like `Ctrl+Shift+N` cannot be captured by web pages, and `Ctrl+Shift+P` is unavailable in Firefox.

Use the web-safe alternative shortcuts (such as `Alt+Shift+N`) in a browser, or use the desktop app for full shortcut support. See the full shortcut table in Shortcuts. On iPad, hold `⌘` on an external keyboard to see available shortcuts.

Chinese, Japanese, or Korean input method does not produce characters

On Linux X11 desktop sessions, the Vibe42 AppImage needs GTK4 input method modules for `fcitx5` or `ibus` to show candidate windows and insert composed text. Wayland sessions and macOS use native input methods without extra packages.

Install the GTK4 module for your input method framework (for example, `fcitx5-frontend-gtk4` or `ibus-gtk4` on Ubuntu/Debian—package names vary by distribution) and restart the app, or switch to a Wayland session. See Linux desktop.

Copying to the clipboard does nothing

Programs running on the remote computer (such as Neovim) can copy text via OSC 52 (a terminal sequence that writes to your local clipboard), which copies to **the device you are currently viewing**. On the desktop app, the Vibe42 window must be focused when the OSC 52 sequence arrives.

Keep the Vibe42 window focused when you run the copy command, and confirm OSC 52 passthrough is enabled in your terminal or editor configuration (see Terminal engine). On phones and tablets, you can also long-press terminal text to select and copy words directly.

Soft keyboard covers the active input line on mobile

Full-screen TUIs (programs that draw interfaces inside the terminal, such as Claude Code) and plain shells react differently when the on-screen keyboard shrinks the screen on a phone or tablet.

Switch between the three soft-keyboard avoidance modes—**Shift page**, **Resize terminal**, and **Follow cursor**—in mobile terminal settings until the input line stays visible. See Mobile input.

Rendering, images, and fonts

Image paste fails or says the file is too large

Pasting from the clipboard accepts up to 1 MiB of plain text and 4 MiB per image. Larger images and HEIC photos are converted to JPEG automatically; other content above these limits cannot be pasted directly from the clipboard.

Drag and drop the file onto the terminal (or use the photo and file picker on mobile) to upload it to the computer and insert its file path at the cursor, or upload large files through the Files panel. If the file is already on that computer, choose **Insert Remote Path** from the attachment menu to insert its path directly without uploading.

Inline images do not render inside the terminal

Vibe42 displays inline terminal images using the Kitty graphics protocol (a standard for drawing images inside terminals), up to 16 MiB per image. In the web app, each pane (one split of a terminal window) shows at most 5 inline images at a time.

Configure your CLI image viewer to use the Kitty graphics protocol and keep individual images at or below 16 MiB (see Terminal engine). If older images are replaced in the browser, open the file from the Files panel to preview or download it, or use the desktop or mobile app.

Wide characters or box-drawing glyphs are misaligned

CJK double-width characters or terminal borders can look misaligned when font ligatures change character widths or a font lacks consistent cell widths.

Open theme and font settings, switch to one of the built-in monospace fonts (**Geist Mono**, **JetBrains Mono**, **Fira Code**, **Blex Mono**, or **Noto Sans Mono**), turn off ligatures, and reset line height if needed. See Themes and fonts.

Terminal is briefly blank after unlocking your phone or switching apps

The mobile app does not stay connected in the background and closes live streams when suspended. Returning to the app reconnects and redraws the terminal screen, while your terminal session on the computer keeps running uninterrupted.

Wait a moment for the app to reconnect and redraw the screen (see Sessions). If the screen stays blank, check your network connection in Connection problems.

File preview and Remote Desktop

File previews fall back to plain text on Linux

The Linux desktop app runs rich previews for code, Markdown, HTML, and PDF files inside a sandbox using `bubblewrap` (`bwrap`) and `xdg-dbus-proxy`. Without these packages, text files display as plain text and PDFs can only be downloaded.

Install both packages with your distribution's package manager and restart the Vibe42 AppImage (see Linux desktop):

```
sudo apt install bubblewrap xdg-dbus-proxy
```

Remote Desktop shows a black screen or missing decoder prompt

In browsers, Remote Desktop requires WebCodecs (a browser API for hardware video decoding). On the Linux desktop app, computers without VA-API or NVIDIA hardware decoding need the Cisco OpenH264 decoder. On a controlled Mac, missing screen recording permission also causes a black screen.

Use a modern browser with WebCodecs enabled for the web app. On Linux, accept the prompt to download Cisco OpenH264 on demand or enable VA-API or NVIDIA drivers. On a controlled Mac, grant **Screen Recording** and **Accessibility** under **Settings → Remote Desktop**. See Remote Desktop getting started.

Collecting logs

Find background service, desktop, mobile, and browser logs across platforms, check for sensitive metadata, and send a complete report to support.

`vibe42.ai/docs/troubleshooting/logs`

Use this page when the steps in Connection problems, Not getting notifications, or Terminal and input problems do not fix your issue and you want to check local logs or send a report to `support@vibe42.ai`.

Before you collect logs

Write down the **exact time and time zone** when the problem happened (for example, `2026-11-02 14:20 UTC+8`). Desktop logs rotate at 4 MiB, so collecting logs right after reproducing the issue keeps the relevant entries in the file.

On any computer running the Vibe42 background service, also save its JSON status output to attach with your logs (older installs may use `~/.vibex` instead of `~/.v42`):

```
~/.v42/bin/v42-companion status --json > v42-status.json
```

> **NOTE**
>
> If the computer is not paired yet, `status --json` includes `pairing.code` and `pairing.approve_url`. Remove those two fields before sharing the file, or wait 10 minutes for the code to expire.

Log locations at a glance

Platform	Primary log location	Notes
macOS app	`~/Library/Application Support/ai.vibecafe.x/ai.vibecafe.x/logs/native.log`	Rotates at 4 MiB to `native.log.1`
Background service (macOS / Linux / Server)	`~/.v42/logs/companion.log`, `companion-install.log`	Supervisor mode adds `supervisor.log`; legacy path `~/.vibex/logs/`
iOS / Android	Private app storage (not user-exportable)	Share screenshots, timestamps, app version, or OS crash data
Web app	Browser DevTools **Console** and **Network** tabs	Export or screenshot failed requests and errors

Platform commands

macOS app

Copy `native.log` (and `native.log.1` if the file just rotated), the background service logs, recent macOS system logs, and any matching crash report names:

```
ls -lh ~/Library/Application\ Support/ai.vibecafe.x/ai.vibecafe.x/logs/
ls -lh ~/.v42/logs/

# Unified system log entries from the last hour
log show --predicate 'subsystem == "ai.vibecafe.x.native"' --last 1h --info > v42-macos-
system.log

# Recent Vibe42 crash reports, if any
ls -t ~/Library/Logs/DiagnosticReports/ | grep -i -E 'v42|vibe' | head
```

Linux and one-line Server installs

Use these commands on headless Linux servers, containers, command-line installs on macOS, and the background service managed by the Linux desktop AppImage:

```
~/.v42/bin/v42-companion status --json
~/.v42/bin/v42-companion logs -f
```

Log files are stored in `~/.v42/logs/companion.log` and `~/.v42/logs/companion-install.log` (plus `~/.v42/logs/supervisor.log` in supervisor mode). On systemd hosts, the user service journal only records service start and stop events:

```
journalctl --user -u v42-companion.service -n 100
```

iPhone, iPad, and Android

Runtime logs on iOS, iPadOS, and Android are stored in private app sandboxes and cannot be exported directly. Share screenshots, the exact timestamp, and the app version shown in settings.

If the app crashed:

- **iPhone / iPad**: Open **Settings → Privacy & Security → Analytics & Improvements → Analytics Data** on your device to find and share the crash report.
- **Android**: Turn on Developer Options and USB debugging, connect the phone to a computer, and filter `adb logcat` for the crash tag `VibexPanic`:

```
adb logcat -s VibexPanic:*
```

Web app

In `https://app.vibe42.ai`, open your browser's Developer Tools, check the **Console** tab for errors, and check the **Network** tab for failed HTTP or WebSocket requests.

Check for sensitive details before sending

Vibe42 logs never contain access tokens, pairing codes, terminal input or output, or file contents (see Security). Still, review log excerpts before emailing them and remove details you do not want to share—such as **hostnames, local file paths, email addresses, and IP addresses**.

> CAUTION
>
> Never send passwords, email verification links, payment card numbers, TOTP (authenticator app) secrets or codes, or two-factor recovery codes to support or anyone else.

Checklist for emailing support

Email your report to `support@vibe42.ai` with:

1. What you were doing when the problem happened.
2. The exact error message on screen or a screenshot.
3. The timestamp and time zone of the failure.
4. Operating systems and Vibe42 versions on both the computer and the device, plus the `~/.v42/bin/v42-companion status --json` output.
5. Relevant log excerpts or crash reports around that timestamp.

FAQ

Quick answers on supported platforms, Teams, coding agents, Credits, end-to-end security, email verification, free trials, and billing errors.

vibe42.ai/docs/troubleshooting/faq

Find short answers to common questions about installing Vibe42, managing Teams and slots, running coding agents, understanding Credits, and fixing account or checkout errors.

Installation and platforms

Can I use Vibe42 on Windows?

Yes. You can install the Server edition inside WSL to use a Windows PC as a controlled computer (terminals, coding agents, and files; no Remote Desktop), and control other computers from Windows in any modern browser at `https://app.vibe42.ai` (which you can install as a PWA, a web app that runs in its own window from your desktop or taskbar). A native Windows version is coming soon. See Windows.

What if my Mac runs macOS 13?

The macOS desktop app requires macOS 14+. On macOS 13, run the one-line Server edition installer (the headless version with only the background service and no desktop app) on that Mac, then connect to it from another device or browser. See macOS and Install.

Do I need a public IP address or port forwarding?

No. Both the background service and your devices make outbound connections to the nearest AP (access point, a regional routing server), establish direct peer-to-peer (P2P) connections automatically when possible, and fall back to encrypted relay otherwise. See Connections.

Do I need to rent a VPS?

No. Install Vibe42 directly on the Mac or Linux computer you already use. If you also have cloud servers or containers, you can add them with the one-line installer. See Linux server.

Does Vibe42 modify my shell configuration?

The macOS and Linux desktop apps never touch your shell startup files. The one-line Server installer only adds one `PATH` entry for `~/.v42/bin` to your shell configuration file and adds no hooks or plugins. See Zero injection.

Accounts and Teams

Do invited Team members need their own plan?

No. Plans, computer slots, and device slots belong to the Team and are paid for by the Team Owner. Invited members share the Team's slots without buying a separate subscription. See Teams.

How do I move Vibe42 to a new computer or phone?

Install Vibe42 and sign in or pair on the new computer or phone, then remove the old computer or device in Console or **Computers & Devices** to free its slot. See Plans and slots and Computers, devices, and Teams.

Does removing a computer delete my local files?

No. Removing a computer only deletes its cloud-saved Workspace records and notes in Vibe42. All source code, folders, and files on your disk stay untouched. See Workspaces.

Agents

Which coding agents are supported?

Any terminal-based coding agent runs in Vibe42. The built-in list includes roughly 30 agents (with Claude Code, OpenAI Codex CLI, OMP, Pi Coding Agent, OpenCode, Grok Build, Devin CLI, Antigravity CLI, and Cursor Agent shown by default), and you can add custom commands under **Agents**. See Supported agents.

Do my model API keys pass through Vibe42?

No. Agents run locally on your computer and connect directly to your model provider. Vibe42 does not change agent configurations or proxy API keys. See Zero injection and Security.

Why doesn't an agent I started manually show status?

The background service only tracks status for agents started from the new-terminal flow or `v42 term new --agent`. Commands typed into a plain shell show status only if the agent natively sends OSC 7501 (a terminal control sequence for reporting status, built into Claude Code 2.1.295+ and Pi 1.1.0+). See Launching agents and Agent status.

What happens to my terminals and agents if my computer restarts?

When the background service starts again after a computer restart, it recreates your windows, panes, and working directories and replays the saved output at the top of each pane. Agents started from the new-terminal flow or `v42 term new --agent` reopen their last conversation, though any turn that was interrupted by the restart needs you to send another message to continue. Agents you typed into a plain shell and other programs are not re-run automatically. See Sessions.

Credits and usage

What consumes Credits?

Only AP-relayed data—terminal traffic, Remote Desktop, file transfers, and `v42 forward`—uses Credits at 64 MiB = 1 Credit per side (sending 64 MiB through relay end to end uses 2 Credits). Signaling, connection setup, control channels, and Console APIs cost nothing, and unused monthly Credits do not roll over. See Credits and usage.

Are direct P2P connections free?

Yes. Desktop apps, mobile apps, the CLI, and the background service use zero Credits whenever traffic flows over a direct P2P connection (browsers always use relay). Run `v42 network` to check whether your path is `direct` or `relayed`. See Connections.

What happens when Credits run out?

When you see `"This month's Credits are used up"`, AP-relayed data channels pause while basic control sessions and direct P2P connections keep working. Renewing or upgrading your plan restores relay access right away. See Credits and usage.

Security

Can Vibe42 see my terminal sessions?

No. Every installation creates a local Ed25519 root key whose private key never leaves your machine, and every connection is end-to-end encrypted with a Noise handshake. APs only forward encrypted data, and we never train models on your work. See Security.

Where is my data stored, and what is logged?

Your code and terminal sessions stay on your computers. Our servers store account credentials (Argon2id password hashes), Team metadata, and routing audit records (sign-ins, pairings, revocations, AP region used, and byte counts)—never terminal text or file contents. Notification titles and bodies are the exception: they must pass through and be stored on our servers to be pushed to your phone, and can be turned off per computer (see Notifications). For details, see Security and Privacy.

How do I revoke a lost or untrusted device?

Remove the device from Console or **Computers & Devices** on any signed-in device. Active connections close within 60 seconds and show `"This device's authorization was revoked."`. See Account and login.

Email verification

Why do I see "Please verify your email before signing in."?

Unverified accounts (also shown as "Not verified yet. Open the link in the email, then try again.") cannot sign in on any device. Open the verification link sent to your email before signing in. See Account and login.

What if I see "This verification link is invalid or has expired."?

Verification links expire after 24 hours. Click resend on the registration page—or submit the registration form again with the same email—to get a new link without changing your password. See Account and login.

Can I verify my email with GitHub or Google instead?

Yes. Clicking **Continue with GitHub** or **Continue with Google** with the same email verifies the account right away. This clears the unverified password; use `/password-reset` on `vibe42.ai` afterward if you still want to sign in with a password. See Account and login.

Trials and payments

Why do I see "A free trial isn't available for this team. Subscribe directly instead."?

The 3-day free trial is available only once to eligible new Team Owners. Subscribe directly at `https://vibe42.ai/console/checkout?resource=access`. See Trial, redeem, and refunds.

Why does checkout say "This card has already been used for a free trial"?

Each payment card can start a free trial only once. Use a different card or subscribe to a plan directly without a trial. See Trial, redeem, and refunds.

What do "Card setup expired" and "Card setup canceled" mean?

Card verification timed out or was closed before finishing—your trial has not started and you were not charged. Click **Start trial again** to try again. See Trial, redeem, and refunds.

What should I do on "Your payment is being confirmed. You can leave this page and return later; please don’t start another payment."?

Your payment provider is still confirming the charge. Wait a moment and refresh Console; do not submit a second payment. See Trial, redeem, and refunds.

Why do I see "Only the Team owner can make this change. Ask them to review the plan or payment."?

Only the Team Owner can choose plans, manage billing, or redeem codes (and mobile apps do not show pricing or subscription controls). Ask the Owner to manage billing in Console on the web. See Teams and Trial, redeem, and refunds.