LENNE.TECH · DEVELOPER WORKFLOW GUIDE

Parallel entwickeln — ohne Reibung, ohne Kollisionen.

Mit **lt dev** und **lt ticket** betreibst du beliebig viele Projekte *und* beliebig viele Tickets gleichzeitig — jedes mit eigener URL, eigener Datenbank, eigenen Ports. Ein Befehl, und du legst los.

- Pro Projekt isoliert
- Pro Ticket isoliert
- Parallele E2E-Tests
- Claude-aware
- Schutz vor Datenverlust

Voraussetzung **lt CLI ≥ 1.28.0 · neuestes lt-dev Plugin** Stack **Nest Server · Nuxt Base**

Routing **Caddy · *.localhost (HTTPS)** Zielgruppe **lenne.Tech Entwickler:innen**

WARUM DAS ALLES?

Schluss mit „erst runterfahren, dann das andere starten"

Bisher band jedes Projekt die Framework-Default-Ports `3000` / `3001`. Zwei Projekte gleichzeitig? Port-Kollision. Auth über Cookies? Cross-wiring. Ein zweites Ticket testen, während das erste läuft? Ging nicht.

VORHER

Sequenziell & fragil

Ein Projekt zur Zeit. Feste Ports kollidieren. Branch wechseln heißt: Server neu starten, DB-Zustand verlieren, Kontext verlieren. Tickets parallel? Nur über getrennte Klone — langsam und unübersichtlich.

NACHHER

Parallel & isoliert

Jedes Projekt und jedes Ticket läuft hinter Caddy unter `https://<name>.localhost` — eigene Ports, eigene DB, eigener Caddy-Block. Mehrere Browser-Tabs, mehrere Claude-Sessions, mehrere Test-Läufe — alles gleichzeitig, nichts beeinflusst sich.

0 s Worktree anlegen (geteiltes .git)	**1** Befehl bis „läuft im Browser"
N Tickets & Projekte gleichzeitig	**~2×** schnellere E2E durch Sharding

VORAUSSETZUNGEN

Bevor du loslegst — einmal aktuell ziehen

Dieser Workflow braucht die **lt CLI ≥ 1.28.0** (enthält `lt dev` + `lt ticket`) und das **neueste lt-dev Claude Plugin** (macht Claude ticket-aware). Beides ist in Sekunden installiert bzw. aktualisiert.

lt CLI ≥ 1.28.0

Version prüfen mit `lt --version` — muss **1.28.0 oder neuer** sein.

```
# Neu installieren (global)
npm install -g @lenne.tech/cli

# Auf die neueste Version aktualisieren
lt update

# Version prüfen (≥ 1.28.0)
lt --version
```

Neuestes lt-dev Plugin

Aus dem **lenne-tech Marketplace** ziehen — installiert/aktualisiert lt-dev (+ weitere lt-Plugins) und richtet die Berechtigungen ein.

```
# lt-dev Claude Plugin installieren / aktualisieren
lt claude plugins
```

Aktualisiert sich zusätzlich automatisch beim Start einer Claude-Code-Session.

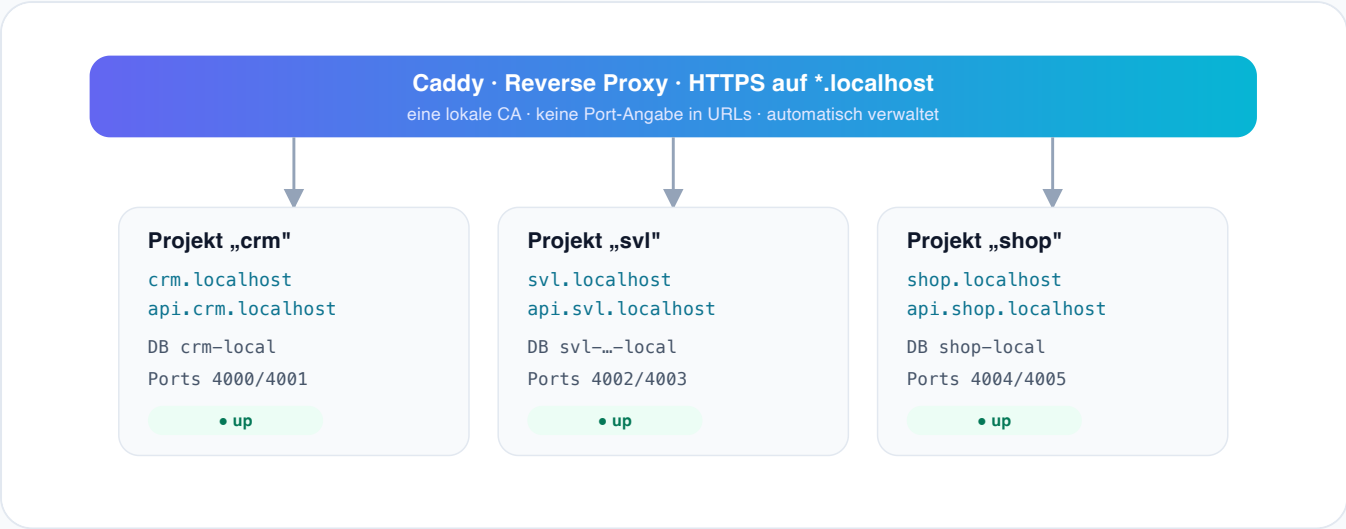
🚀 **In 3 Schritten startklar:** `lt update` (CLI auf ≥ 1.28.0) → `lt claude plugins` (Plugin ziehen) → im Projekt `lt dev init`, dann `lt ticket start <ticket>`.

BAUSTEIN 1

`lt dev` — der isolierte Stack pro Projekt

Ein Projekt, eine eigene Welt. **lt dev up** startet API + App hinter Caddy unter projekteigenen URLs — beliebig viele Projekte nebeneinander, garantiert

kollisionsfrei.



Drei Projekte, gleichzeitig „up" — eigene URLs, eigene DBs, eigene Ports. Caddy routet alles über HTTPS, ganz ohne Port-Angabe.

Der Lebenszyklus

```
# Einmal pro Rechner
lt dev install            # Caddy + lokale CA + Service einrichten

# Einmal pro Projekt (idempotent)
lt dev init               # env-aware patchen + registrieren (idempotent)

# Täglich
lt dev up                 → https://svl.localhost · https://api.svl.localhost
lt dev status --all      # was läuft gerade? (alle Projekte)
lt dev down               # sauber stoppen + Caddy-Block entfernen
```

Befehlsübersicht

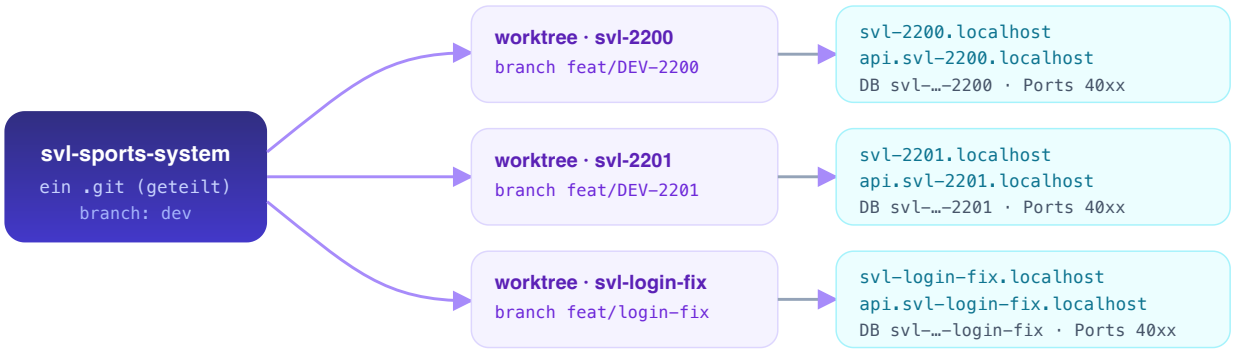
BEFEHL	WAS ES TUT
`lt dev install`	Einmal pro Rechner: Caddy, lokale CA und Hintergrund-Service einrichten.
`lt dev init`	Einmal pro Projekt: Ports env-aware machen, registrieren, `ignoreHTTPSErrors` + shard-aware Test-Timeouts injizieren. Idempotent.
`lt dev up`	API + App hinter Caddy starten: `https://<slug>.localhost` / `api.<slug>.localhost`, DB `<slug>-local`.
`lt dev down`	Prozesse stoppen, Caddy-Block entfernen, etwaige Test-Stacks abräumen.
`lt dev status` · `--all`	URLs, Ports, PIDs, Live-Zustand — fürs aktuelle Projekt oder alle registrierten.
`lt dev doctor`	Caddy/CA/DNS/Ports prüfen — NEU warnt auch, wenn ein DB-löschendes `global-setup` die Ticket-/Shard-Test-DBs nicht zurücksetzen würde.
`lt dev test`	E2E in einem **isolierten, parallelen** Stack auf dedizierter DB `<slug>-test` — restfreier Auto-Teardown.
`lt dev test --shard N`	Suite auf N isolierte Stacks aufteilen (Default 2) — lokale CI-Parität, ~2× schneller.
`lt dev tunnel`	Öffentliche `*.trycloudflare.com`-URL für ein laufendes Projekt (Demo/Review).

🔌 **Nie wieder `localhost:3000`.** Alle URLs kommen von `lt dev` über die Env-Bridge (`.lt-dev/.env`) — Playwright, IDE und Skripte ziehen sie automatisch.

BAUSTEIN 2

`lt ticket` — mehrere Tickets gleichzeitig

Ein Repo, beliebig viele Tickets — jedes in einem eigenen **git worktree** mit eigenem `lt dev`-Stack. Frisch aus `origin/dev`, in Sekunden startklar, voneinander komplett unabhängig.



Ticket-ID auch ohne Ticket möglich: ein freier Feature-Name funktioniert genauso (z. B. „login-fix").

Ein Repo → mehrere Worktrees (geteiltes `.git`, sofort) → je ein vollständig isolierter Stack. Ordnername = URL = DB = Ticket — du weißt immer, wo du bist.

So einfach geht's

```
lt ticket start DEV-2200          # fetch + worktree aus origin/dev + install +
lt dev up
                                → https://svl-2200.localhost · DB svl-…-2200
(leer)
lt ticket start login-fix         # kein Ticket? freier Name → svl-login-
fix.localhost
lt ticket start DEV-2200 --as cof  # Kurzname überschreiben; --branch / --base
ebenso

lt ticket list                   # Dashboard: alle Tickets + URLs + Branch +
Status + DB
lt ticket switch 2200           # Pfad zeigen + im Editor öffnen
lt ticket test 2200 --shard 2    # E2E im isolierten Ticket-Stack/-DB
lt ticket stop 2200 --drop-db    # down + worktree entfernen (Branch bleibt); --
drop-db löscht DBs
```

Eigene Identität, überall

Jedes Ticket bekommt `<slug>-<id>` in URL, DB, Ports, Caddy-Block, Ordnername. Verwechslung ausgeschlossen.

Immer frisch aus dev

Jeder Start macht `git fetch` und zweigt von `origin/dev` ab — alle Tickets unabhängig. `--base` für Ausnahmen.

0 s & leicht

Worktree teilt das `.git` — kein Re-Clone. Ein `git fetch` aktualisiert alle. pnpm hardlinkt aus dem Store.

ABLAUF

Ein Ticket — von „los" bis „weg"

Vor dem Aufräumen warnt `lt ticket stop`, falls noch **uncommittete** oder **ungepushte** Arbeit existiert — nichts geht versehentlich verloren.

🚢 **Ein Befehl bis in `dev` — der Claude-Command `/lt-dev:git:ship`.** Bringt deinen fertig entwickelten Ticket-Branch **autonom** nach `dev`: `check`-Skript grün → committen → auf `dev` **rebasen** → testen → **Merge Request** (Squash + Auto-Merge) → wartet auf die CI-Pipeline (**Auto-Retry** bei Fehlschlag) → **squash-merged** → löscht den Branch. Die automatisierte **Schlussklammer** zu `/lt-dev:take-ticket` — genau der Ablauf, den du sonst von Hand machst. Flags: `--base=<branch>`, `--no-squash`, `--keep-branch`, `--max-pipeline-retries=<n>`.

Ein Tag mit drei Tickets — gleichzeitig

1. **Morgens:** `lt ticket start DEV-2200`, `lt ticket start DEV-2201`, `lt ticket start login-fix` — drei Tabs, drei VS-Code-Fenster, drei Claude-Sessions. Jede Umgebung ist sofort im Browser unter ihrer eigenen URL.
2. **Mittags:** Review-Feedback zu 2201 kommt rein — du wechselst per `lt ticket switch 2201`, fixst, `lt ticket test 2201` läuft *parallel* während 2200 weiter im Browser offen ist.
3. **Nachmittags:** 2200 ist fertig → committen, pushen, MR. `lt ticket stop 2200 --drop-db` räumt restlos auf. 2201 und login-fix laufen ungestört weiter.
4. **Überblick jederzeit:** `lt ticket list` zeigt alle Umgebungen mit URLs, Branch, Status und DB — du verlierst nie den Faden.

AUCH OHNE NEUES VERZEICHNIS

Schnell mal ein Ticket im aktuellen Checkout

Nicht jedes Ticket braucht einen eigenen Worktree. Willst du *im aktuellen* Projektverzeichnis ein Ticket angehen — ohne neuen Ordner — startest du es direkt mit dem Claude-Command `/lt-dev:take-ticket`.

PARALLEL

`lt ticket start`

Eigener Worktree + eigener Stack. **Wenn du mehrere Tickets gleichzeitig** bearbeiten, im Browser vergleichen oder parallel testen willst. Volle Isolation, ein Befehl.

IN-PLACE

`/lt-dev:take-ticket`

Im **aktuellen** Verzeichnis & Branch — kein neuer Ordner. Claude holt sich Ticket-Kontext (Linear, Figma, Flows), plant und setzt um. Ideal für **ein** Ticket nacheinander, ohne Worktree-Overhead.

🧭 **Faustregel:** Mehrere Tickets gleichzeitig oder im Browser nebeneinander testen → `lt ticket`. Ein Ticket fokussiert im aktuellen Checkout → `/lt-dev:take-ticket`. Beide nutzen denselben isolierten `lt dev`-Unterbau.

TESTS

Parallele E2E — pro Ticket *und* per Sharding

Jeder Ticket-Test fährt seinen **eigenen** Test-Stack hoch — eigene Test-DB `<base>-<id>-test`, eigene Ports. Ein Registry-Lock garantiert atomare Port-Vergabe, sodass selbst zwei *gleichzeitig* gestartete Test-Läufe nie kollidieren.

Zwei Ticket-Test-Suites liefen gleichzeitig auf distinkten Ports und eigenen Test-DBs — beide grün, null Interferenz.

Sharding in einem Stack

`lt dev test --shard N` teilt die Suite auf N isolierte Stacks (Default 2) — die lokale Entsprechung der CI-Matrix, ~2× schneller. Die Timeouts werden nur unter Shard-Last gelockert (CI bleibt schnell).

Pro Ticket über Worktrees

`lt ticket test <id>` testet im eigenen Ticket-Stack mit eigener Test-DB. Mehrere Tickets können **gleichzeitig** testen — atomare Port-Vergabe per Lock verhindert jede Kollision.

⚠ **Voraussetzung für DB-Reset:** Projekte mit DB-löschendem `global-setup` müssen die Allow-List ticket-/shard-sicher halten (`/^<base>-(?:[a-z0-9-]+-)?test(?:-\d+)?$/`). `lt dev doctor` warnt, falls nicht — neue Projekte aus dem Starter brauchen hier nichts.

EINGEBAUT

Sicherheit & Komfort, ohne dass du dran denkst

Kein versehentlicher Datenverlust

`lt ticket stop` verweigert das Entfernen, solange **uncommittete** oder **ungepushte** Arbeit existiert — mit klarer Auflistung. `--force` überschreibt bewusst. Generierte Dateien (`.nuxtrc` & Co.) blockieren nicht.

Selbstdiagnose

`lt dev doctor` prüft Caddy, CA, DNS, Ports — und warnt, wenn ein `global-setup` Ticket-Test-DBs nicht zurücksetzen würde, samt exaktem Fix.

Claude weiß Bescheid

Jede Claude-Session in einem Ticket-Worktree erkennt am `.lt-dev/ticket`-Marker automatisch ihr Ticket — und sieht jede Runde Ticket-ID, URLs und DB. Keine getrackte Datei wird verändert.

Restfreier Teardown

Prozesse, Caddy-Block, Session, Registry-Eintrag, Ports — alles wird sauber zurückgegeben. `lt ticket stop` ohne ID räumt sogar das *aktuelle* Worktree auf.

SPICKZETTEL

Alles auf einen Blick

Setup & Projekt

```
lt dev install          # 1× pro Rechner
lt dev init             # 1× pro Projekt
lt dev up / down        # Stack an/aus
lt dev status --all     # Überblick
lt dev doctor           # Diagnose
lt dev test --shard 2   # schnelle E2E
lt dev tunnel           # öffentliche URL
```

Tickets parallel

```
lt ticket start DEV-2200   # neues Ticket-Env
lt ticket start login-fix  # freier Name
lt ticket list            # Dashboard
lt ticket switch 2200     # öffnen
lt ticket test 2200       # isolierte E2E
lt ticket stop 2200       # aufräumen (Branch bleibt)
/lt-dev:take-ticket        # Ticket im aktuellen Checkout
/lt-dev:git:ship           # fertig → autonom nach dev
```

🚦 **In 3 Schritten startklar:** `lt dev install` (einmalig) → im Projekt `lt dev init` → `lt ticket start <ticket>`. Fertig — die URL steht im Terminal und in `lt ticket list`.

DER GEWINN

Warum dich das spürbar schneller macht

🧠 Kein Kontext-Verlust

Jedes Ticket behält seinen eigenen, laufenden Zustand — Server, DB, Browser-Tab, Claude-Session. Du springst zwischen Tickets, ohne irgendetwas neu aufzusetzen.

⏱️ Kein Warten

Worktrees entstehen in 0 s, Stacks starten in Sekunden, Tests laufen parallel statt nacheinander. Review-Feedback fixst du sofort, ohne dein Hauptticket zu unterbrechen.

🛟 Kein Risiko

Volle Isolation + Schutz vor Datenverlust + Selbstdiagnose. Du experimentierst frei — die Werkzeuge fangen die Fehler ab, bevor sie wehtun.

✨ **Probier es heute:** Such dir zwei offene Tickets und starte sie parallel mit `lt ticket start`. Du wirst nicht mehr zurückwollen.

---

lenne.Tech · `lt dev` × `lt ticket` — Parallel entwickeln ohne Reibung.
Lebendes Dokument in der lt CLI unter `docs/lt-dev-ticket-workflow.html` — Beiträge & Erweiterungen willkommen.