Instinto

Lince Documentation

Typst Version: 0.14.2

Starting Date: 2025-12-15

Date of Print: 2025-12-16

Days of Construction: 1

Source: Lince (@lince-social/lince)[1]

[1]"https://github.com/lince-social/lince"

Contents

Cough

1. Birth

`Morbus est medicus naturae` \Disease is the doctor of nature, illness forces correction

— ChatGPT (circa 2025)

Lince is not great, it always needs to get better, until it dies, then it doesn't. The way to do it is giving it remedy (teaching people how to use it) while we cure it (making it simpler).

1.1. Summary

Lince is a tool for registry, interconnection and automation of Needs and Contributions with open scope.

The phrase above is the densest way to explain Lince. The current page serves as an overview, and the subsequent chapters cover it in much more detail. Let's start with the registry part:

Registry

Users will start their Lince application for the first time, and that will create a personal database

(for UNIX systems [Linux and macOS] it is located at ' /.config/lince/lince.db').

That database is a Sqlite file, that is reffered by the application as a DNA. That file can be altered to make the app behave differently.

The behavior can be of: setting personal/professional tasks, modeling personal items, scheduling computer actions, act out economic trades between parties, and many more, the sky, your imagination and computing power is the limit.

Any app behavior mentioned above can be boiled down to a Need, or a Contribution to a Need. The differentiator of the two is a Quantity. If such quantity is negative, that will be a Need.

Having -2 Apples means you need to go and get two apples to be neutral, to satiate your needs. The same applies to a habit, having a -1 Quantity in Exercise means you need to exercise.

The previous description is the basis of the whole app, everything revolves around it.

Automation

If there is predictability in your Needs, and a fixed frequency is enough to describe them, then a simple automation can be constructed.

One can set the automatic every day change of the 'Studying' habit's Quantity to -1. And with another feature, if that Quantity is -1 a Command is sent to the computer that closes all programs that are gaming related.

Another point of automation is the forementioned DNA. One aspect of constructing systems is that many great ideas often are lost with time, memory and the passing away of those contributors. Common patterns and knowledge about production and maintenance of commodities, economic trades, access to resources and ways of living is scattered.

These patterns could be saved into some form of database, that when fed into a program like Lince, will create Needs and Contributions, so users can have a headstart in any endeavor. From production according to a bill-of-materials to morning routines and parties organization, these activities create Needs for gathering resources and acting out tasks.

Contributions to Lince DNAs are like creating blueprints for others to get up and running. They might not be a perfect fit, but allow for great customization. Currently the way of centralizing DNAs is the [Github Repository](https://github.com/lince-social/dna).

Interconnection

With great customizability and automation of people's DNAs, the next barrier is the connection of those individual apps, turning each machine into a p2p node, also called a Lince Cell. The same way you create triggers and rules for the quantites of your DNA, you can do the same for others.

If one Market has 200 apples and you need 5 you can create a Transfer Proposal. The Quantity of money diminishes on your side, and the Apple one increases. On the Market side, a manual price can be inputed and accepted when proposals come, or it can be adaptive. Donations of clothes and asking people close to you for a tool would be similar, just without the money part.

1.2. Installation

The easiest way is

You can run Lince as an executable, here are the [releases](https://github.com/lince-social/lince/tags). Pick the latest one for your machine and operating system. After the download, unzip it and run the binary:

```
./lince
```

If you don't have Cargo installed yet, you can install it with [Rustup](https://www.rust-lang.org/tools/install). It comes with Rust and its toolchain.

If `cargo` isn't in your `PATH`, add it to your shell configuration (`~/.bashrc` or `~/.zshrc`) so you don't have to type the full path every time (i.e., `~/.cargo/bin/{program}`). Run this in your terminal:

```
echo 'export PATH="$HOME/.cargo/bin:$PATH"' >> ~/.bashrc
echo 'export PATH="$HOME/.cargo/bin:$PATH"' >> ~/.zshrc
```

If you preffer to compile the source code, the easiest way is to download the repo and run:

```
cargo run
```

If you don't have cargo installed, you can install it with with [Rustup](https://www.rust-lang.org/tools/install).

This uses [Typst](https://typst.app/) to generate the documentation. I reccomend installing [mask](https://github.com/jacobdeichert/mask) with `cargo install mask`, then running `mask docs`. If you are in Arch or MacOS systems it will automatically install typst and [tinymist](https://github.com/Myriad-Dreamin/tinymist) (web server), else it will probably fail, learn how to install typst and tinymist manually.

1.3. Disclaimers

All the repositories in the Lince ecosystem have the [GNU GPLv3](https://www.gnu.org/licenses/gpl-3.0.html) license. Lince is a non-profit project and crowdfunding is the source of development compensation:

[GitHub Sponsors](https://github.com/sponsors/lince-social) | [Patreon](https://www.patreon.com/lince_social) | [Apoia.se](https://www.apoia.se/lince)

Lince tries to facilitate and automate the connection between people and resources, by transforming needs and contributions into data. The gains and losses related to the interaction, such as transportation, production and services themselves, remain the responsibility and risk of the parties involved.

1.4. Organization

The workflow of contributing to the Lince ecosystem can be of simply making a Pull Request. But if one wants to have certainty of developing a wanted feature, the best way is to see the DNA of Lince programming tasks.

The online resource to communicate is the [Discord](https://discord.gg/3Gr9rYWHpu).

The git repositories hosted currently in the [lince-social](https://github.com/lince-social) GitHub Organization mean this:

- [lince](https://github.com/lince-social/lince): the app's code and this documentation.
- [dna](https://github.com/lince-social/dna): the databases for different Lince behaviors.
- [.github](https://github.com/lince-social/.github): general stuff, like photos, icons and information about GitHub sponsors.

1.5. Tech Stack

Backend:

- Programming Language: [Rust](https://www.rust-lang.org)
- Server: [Axum](https://github.com/tokio-rs/axum)
- Database: [Sqlite](https://www.sqlite.org/) | Easiest to replicate the concept of DNA.
- Database Driver: [SQLx](https://github.com/launchbadge/sqlx) | Fun async, previously was Rusqlite but didn't prototype that well.

Frontend:
- Web:
  - Base: HTML, CSS
  - Templating (Backend): [Maud](https://github.com/lambda-fairy/maud)
  - Framework: [HTMX](https://github.com/bigskysoftware/htmx) + [Datastar](https://github.com/starfederation/datastar) (transitioning into)
- Bevy:
  - Don't know yet, still learning, probably will use WASM.

Dirty Architecture:

- Application
  - Providers: simple CRUD operations.
  - Use_cases: anything complex, business ruly, that calls providers or manipulates data.
  - Schema: structure of data that comes in or goes out through the endpoints, or an alias for a data structure as a Type so it doesnt clutter the screen.
- Domain:
  - Entities: the most accurate representation of the main drivers of the application. The source of truth of Lince's concepts.
  - Repositories: traits for defining the methods of the real Infrastructure repositories (database operations).
- Infrastructure:
  - Http:
    - Routers: the paths and arguments of endpoints.
    - Handlers: the functions that are called by endpoints, that receive the query params, path variables, header, body if wanted. Responsible for calling providers, use_cases and/or returning presentation functions (frontend).
  - Database:
    - Management: database connection, migrations and schema.
    - Repositories: implementation of Domain traits for database operations like get, update...
  - Utils: functions for quality of life, like logging.
  - Cross_cutting: dependency injection for a collection of many methods in different layers. Mostly for single database connection initiation.
- Presentation
  - Html: the templates made with maud, or sometimes just big strings. that are returned from functions. These HTML may contain HTMX and/or Datastar.

  The directories are divided rudimentary by concepts/entities like ‘operation’, ‘table’, ‘pages’.
- ▸ Bevy: not implemented yet, but mainly for a more featureful and immersive GUI.

1.6. Rust

Rust is a Procedural, Compiled, Borrow Checked (Memory Safe(r)), Statically Typed, Lower Level Than Go And Higher Level Than C programming language. It offers great concurrency, async (hard) and robust unwanted behavior elimination. Has a powerful standard library but shines greatly with crates (dependencies), using cargo, it’s package management system.

There are many ways to write procedures like any language. Some more idiomatic than others, with one liners, syntax sugar and macros. This manual serves as a way of helping to write good Rust procedures, outside of executing Clippy’s suggestions (more info below).

You install Rust preferably through [Rustup](https://www.rust-lang.org/tools/install), which comes with the whole toolchain (you are going to need it), including:
- rustc: the Rust compiler.
- [cargo](https://crates.io/): the package manager and build tool.
- rustfmt: formats Rust code according to style guidelines.
- [clippy](https://doc.rust-lang.org/stable/clippy/index.html): a linter that provides suggestions to improve your code (micro best practices).
- rust-analyzer: a language server for IDE support and code navigation.

There are [awesome](https://github.com/rust-unofficial/awesome-rust?tab=readme-ov-file#games) things built with Rust, including:

- [Bevy](https://bevy.org/): a game engine that you use by writing ‘pretty normal’ Rust.
- [Helix](https://helix-editor.com/) | [Zed](https://zed.dev/): text editors/IDEs.
- [uutils](https://uutils.github.io/): rewrite of GNU’s core utils.
- [Deno](https://deno.com/): a JavaScript/TypeScript runtime.
- [Rust for Linux](https://github.com/Rust-for-Linux/linux): an effort to support Linux kernel modules in Rust.
- [Dioxus](https://dioxuslabs.com/) | [Tauri](https://tauri.app/): cross-platform application builders.
- [Axum](https://docs.rs/axum) | [Actix Web](https://actix.rs/): backend frameworks.
- [Redox](https://www.redox-os.org/): an operating system.
- [Limbo](https://github.com/tursodatabase/limbo): an SQLite ‘evolution’.
- [Fish](https://fishshell.com/) | [Nushell](https://www.nushell.sh/): shells.
- [Servo](https://servo.org/): a browser engine.

- [Alacritty](https://github.com/alacritty/alacritty): a GPU-accelerated terminal emulator.
- [Cosmic Desktop Environment](https://system76.com/cosmic/): a desktop environment native to Pop_OS.
- [Niri](https://github.com/YaLTeR/niri): a scrollable window manager for Wayland.
- [Maud (awesome)](https://maud.lambda.xyz/) | [Yew](https://yew.rs/) | [Leptos](https://leptos.dev/): frontend.
- [Iced](https://iced.rs/) | [egui](https://www.egui.rs/): GUI toolkits.
- [Polars](https://github.com/pola-rs/polars): DataFrames.

1.7. Learning Resources:

1. [Rust Book](https://doc.rust-lang.org/stable/book): chapters 1-6 will give you a simple base if you just want to hack something together. Passing through the entire book, doing the examples and understanding them deeply, will set you up for great vibe coding.

2. [Rustlings](https://rustlings.rust-lang.org/): for when you are done with the book. You will complete exercises on a broad spectrum of the language's features.

3. [Rust for Rustaceans](https://rust-for-rustaceans.com/): for a deeper dive in the language, good practices and crate recomendations.

4. [This Week in Rust](https://this-week-in-rust.org/): a weekly newsletter about Rust projects, crates and language changes. Amazing for keeping up to date with the ecosystem.

5. Youtube Channels: [Jon Gjengset](https://www.youtube.com/@jonhoo/videos) | [No Boilerplate](https://www.youtube.com/@NoBoilerplate) | [The Rustagen](https://www.youtube.com/@TheVimeagen) | [Tsoding](https://www.youtube.com/@TsodingDaily) |[Code to the Moon](https://www.youtube.com/@codetothemoon) | [fasterthanlime](https://www.youtube.com/@fasterthanlime) | [chris biscardi](https://www.youtube.com/@chrisbiscardi/featured) | [Mike Code](https://www.youtube.com/@mikecode-ns7tq/videos)

6. Podcast: [Rustacean Station](https://rustacean-station.org/)

7. Discord, Matrix and many more online communities are very welcoming to newbies.

There are some crates to help your CI:

1.8. Cargo Audit

> Audit your dependencies for crates with security vulnerabilities reported to the [RustSec Advisory Database](https://github.com/RustSec/advisory-db/).

```
== Install
cargo install cargo-audit --locked

== Run
cargo audit
```

1.9. Cargo Udeps

See your unused cargo dependencies:

```
== Install
cargo install cargo-udeps --locked

== Run
cargo +nightly udeps
```

1.10. Cargo Vet

> [...] tool to help projects ensure that third-party Rust dependencies have been audited by a trusted entity.

```
== Install
cargo install cargo-vet --locked

== Initialize a standard Vet criteria, this can be changed
cargo vet init

== Run
cargo vet
```

Cough

2. Aging

`Morbus est medicus naturae` \Disease is the doctor of nature, illness forces correction

— ChatGPT (circa 2025)

Cough

3. Sickness

`Morbus est medicus naturae` \Disease is the doctor of nature, illness forces correction

— ChatGPT (circa 2025)

Lince is not great, it always needs to get better, until it dies, then it doesn't. The way to do it is by giving it medicine (teaching people how to use it) while we cure it (making it simpler and featureful). The tasks and community to improve are found in the Discord server. So for now, with this configuration, we can focus on improving your usage of Lince.

Making a Lince DNA, modeling items and actions to perform is a continuous effort, if it makes sense to you.

3.1. Good practices

– Create a view or more with all the data and configuration for managing your DNA, so Configurations, Collections, Views...

Karma

Until Lince has Deterministic Simulation Testing (DST), you have to be mindfull of the Command table you produce, every command you set may possibly break your system if you don't tidy things up. If you have logic of running a command every hour if one record has quantity > X and you forget about it, any simple change will trigger it, so put guardrails for running things, be it Commands, Queries or even changing Record Quantities. Changes might cascade and deliver unforeseen consequences.

With DST this is easier to do, the plan is to have a containerized environment that runs a simulation of your system, isolated from the outside world. Your DNA (your personal configuration of lince) is a seed that can be run multiple times arriving at the same result (hopefully). Being able to run it with a high speed, changing the date (affecting the Frequency table) and record quantities will bring reproducible results. When you want to add something to your DNA you can check it's effects with a simulation and get info if it breaks anything in an edge case.

Command

Lince works with Sqlite files for it's database. It is recomended to frequently backup your DNAs, weekly, daily or hourly if you are paranoid. If some error or mistake happens, your information is safe.

3.1.1. Personal Tasks

– Create Views for understanding your tasks, select the ones that have negative quantity, check this SQL:

```sql
SELECT * FROM record WHERE quantity < 0 AND LOWER(head) LIKE '%task%'
```

– One can use the 'head' column in 'record' table as a tag holder. So Record's heads with Items and Tasks that have negative quantities appear in my 'Negatives' view.

3.1.2. Finance

3.1.3. Transactions

Live

4. The Death of Lince

Death is only the beginning
— The Mummy (1999)

4. The Death of Lince

Like every good contribution, the goal is to not have to contribute anymore; because it was a one time need, or the frequent need has been solved. Lince is a contribution to the need of having Lince. The death of Lince is the death of the need of Lince. If there is ever any configuration of resources that makes Lince useless, then it's job is fullfilled, there is nothing else to be done, no commit, no push, just vibes.

Bibliography