# Dreams

This page documents the dreams of the Lix team. These are features which we have generally not roadmapped yet, and which may not have complete and thoroughly thought-through plans, and which we would like to think about more completely before implementing. We are writing them down publicly so that others can dream with us.

- language versioning https://wiki.lix.systems/books/lix-contributors/page/language-versioning
- split the evaluator into a separate process, interact with it only via rpc (horrors)
- bytecode evaluator with all the possible trappings (horrors)
  - allows arbitrary runtime-define breakpoints, watchpoints, program inspection and manipulation
  - interacts with rpc to allow perfect lsp hosts, better debuggers etc
- new gc for the evaluator to replace bdw, prototype/template for gc in eventual rust evaluator (horrors)
- flakes as a library of code that provides new nix subcommands (horrors, others)
- lix.conf `prelude-path =` for system-wide subcommands a la git (horrors)
  - also can make per-repo `lix *` commands (jade, janik)
- eval caching with a `memoize :: str -> any -> any` builtin that is overridden by `scopedImport` with a unique, deterministic subscope (horrors)
  - `import := f: memoize (toString f) (scopedImport builtins f)` (horrors)
- flake eval caching entire attrpaths: `mapAttrsRecursive (n: const (memoize n))` on all scopes/attrsets in the "flake" (horrors)
- lazyUpdate is a disaster waiting to happen, turns all values into even worse errors sources than simple thunks (and is deeply intrusive to the evaluator for little gain). why not special attrset ops `__members, __getMember` to simulate lazyUpdate in a library that doesn't infect all future versions of the language and can be transpiled when necessary? (horrors)
  - pureImport is too fine grained, store paths as boundaries actually make sense (and give memoize stable starting scopes), pure eval mode could be "ask thing to pack itself up, add to store, eval from there like nix flakes do" (horrors)
- all authoritative information about the store attached to store objects, not an sqlite database (eg in xattrs or similar) (horrors)
  - would make overlayfs stores for containers/vms trivial
- redo the lazy trees infra on the basis of "virtual" store paths and mountpoints (turning eg a zip file into a virtual mountpoint `/nix/store/lazy/thing.zip/...`) (horrors)
  - notably do not use fuse for this, just a pure vfs implementation
- fully decouple evaluator and store (horrors)
  - tvix has kind of done this with EvalIO, lix needs it too (otherwise the eval-process split will not be possible)
- store operations state, like "what derivations were realized in the last build" (Qyriad)
  - "what attrpath was this accessed by to build"
- profiler for nix code (jade)

- nix develop replace store path but actually good, with bind mounts (jade)
- nixos-rebuild gets unfucked perhaps with samueldr code (jade)
- we kinda wanna have inherits consistent by container type such that you can write inherit (thing) [ a b c ] to create a list, inherit (thing) { a b c } to create a set, or nest those in existing lists or sets to extend them in-place like current inherit (horrors)
- unbreak the io model (horrors)
  - currently nix has an async io model shoved into a sync runtime, and an async model that can't decide whether it's push or pull. this ***sucks***
- a dependency graph for builds which explains why different dependencies are being built
  - store path truncated to unique names in output...?
- native nix-output-monitor (`nom`) style (slash bazel-style) output formatting (showing a live updating list of stuff being built/fetched, with warnings stacking up above it)
  - web viewer for the build graph as it is happening with a nice live log viewer (jade)
    - relatedly: show closure graphs nicely (jade)
- make the store properly multi-tenant, with things like, e.g. authentication and maybe even certain http done via hooks on the client side (jade)
  - see e.g. https://git.lix.systems/lix-project/lix/issues/254
  - overall improve the clarity of what is actually running on the daemon vs the client (jade)
- replace `nix profile` with something not broken with a clear ramp to either have a manifest mutably in the store or operate mutably against a configuration directory. ideally out of tree. (jade)
- fix fs builtin problems (jade)
  - can't read symlinks
  - filterSource gives no metadata of interest esp on symlinks
  - can't synthesize symlinks or files into the store except by serious nar abuse
    - (Zoe) We can imagine a generalized transformSource builtin which presents an fs subtree as a nested attrSet containing the full metadata and contents of all files and links in the subtree, and expects an nested attrSet in the same format as output, allowing arbitrary transformations in pure nix code. As long any other other operations that touch touch the file system are disallowed inside the transformation function (evaluating other paths, building derivations, pathExists, etc) this should be a consistent operation. There may be performance/usability reasons to not use this precise interface, but I think it's a good abstract guide stone of what to strive for.
  - is lib.filesets made of evil? how does it work?
    - answer: it's filterSource in a trench coat with some set operations
  - what if you could take a source tree of a monorepo and rewrite cross project symlinks to refer to store paths of those other projects so you don't copy the entire giant repo to store every time and can have each subproject as its own store path?
  - what if you had a fetch git subtree primitive that was free if there's no modification?
    - (Zoe) It's a little trickier than just that because if you want a filtered git subtree you need some way to ensure that the filter hasn't changed either.
- Better facilities for writing performant code (Zoe)

    - Builtins should document their algorithmics and when they cause files to be written to the store
    - More opt-in persistent data structures with different performance tradeoffs that can be coerced to from the standard values
        - RRB vectors or similar for lists
        - HAMT or similar for attrSets
            - should allow using arbitrary values as keys
            - will probably need an explicit distinction between strings and symbols
            - also a separate set type, so you don't have to bother faking it with null keys
        - StringView like type for strings
            - or maybe just convert in place the first time we'd need to get the length?
- Doing something about IFD being bad (raito, pennae?):

  https://pad.lix.systems/sW0nbPohTgqy2UdIJjPeUA

# fixing ux

- some way of having a persistent short lived evaluator for fast completions in CLI (Dawn)
- □ fancy □ repl, a la IPython and pry (Qyriad)
- Support instance of Lix running locally off the main page to try out
    - Obviously WebAssembly schenanigans involved
- replacing nixos-option (jade)
    - CLI commands should be possible to actually deprecate (jade)
- a debug macro like rust's `dbg!` https://doc.rust-lang.org/std/macro.dbg.html
- pipe operator (Qyriad)
    - and either haskell's $ or left pipe operator
- hyperlinked sources in docs (jade)
- a VFS mirror of the Nix store that puts the names first, attaches a more descriptive label if necessary, and then the hash, literally just for convenience (Qyriad)

# slaying the hydra

these are problems that make hydra sad

- make -jsem jobserver built into Lix (horrors actually wrote one years ago)
    - this would allow much better build density in Lix and eliminate most need to tune NIX_BUILD_CORES
    - see: https://github.com/NixOS/nixpkgs/pull/143820, it turns out the make jobserver protocol is actually *horrible*, and we should instead do this with a reasonable socket protocol injected into the sandbox by Lix
- externalize deciding which host to build things on (delroth, jade)
    - this is necessary because /etc/nix/machines is really stupid and doesn't have nearly enough information to decide whether a machine can admit a job.
- make the remote protocol not suck (jade)

    - latency is bad
    - a lot of stuff blocks in ways it only dubiously needs to?
- what if you could have build cost estimates on large installations, which could go into scheduling decisions? (jade)
    - galaxy brained idea: build a neural net for derivation build costs for scheduling purposes. probably take as input the `derivation show` json with the hashes removed and then a pile of historical hydra data
    - do we have the data to do this? we want cpu time, io, and (ugh these would be very fake though because measuring memory is fraught) memory stats for builds.
    - schedule on machines that have space for the expected cpu-time/memory-time/io-time of the derivation
- make the nix daemon know what is actually building (jade)

---

Revision #22
Created 2024-03-27 07:14:59 UTC by jade
Updated 2026-02-01 13:14:00 UTC by piegames