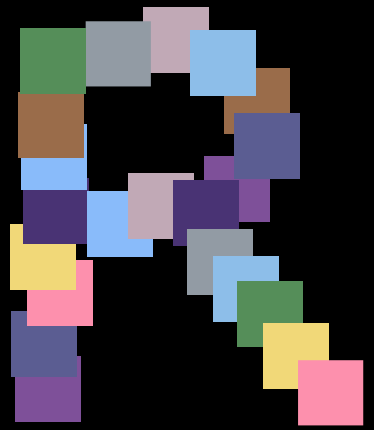
NixとGuixとld.soと

ROCKTAKEY

2025-03-09

自己紹介 --- ROCKTAKEY



- Emacs、Guix、C++あたりをよく使う
  - C++は最近下火、Pythonのが使ってる
  - メインOSはGuix System
- NixはGuixのサブとして使用
- 寝ても寝ても眠いのが悩み

- 気軽に話しかけてください

突然ですが…

このロゴを見たことがある人…？

Guix

GNU Guixとは

- “Geeks”のように読む
- GNUプロジェクトの一部
- Nixの後発プロジェクト
- Nixと同様に1bitレベルで再現性のあるビルドを目指す
- Nixに対するNixOSと同様に、Guix SystemというOSが存在

https://guix.gnu.org/

なぜここでGuixの話を？

- NixとGuixは近縁で、似た目的を持つ
- お互いの良い点を参考に、お互いを高めたい
- GuixはNixに比べて圧倒的に知名度がないため、とりあえず存在だけでも知って欲しい

- Nixについてなにか正確でないことを言ってしまっていたら教えてください🤦

NixとGuixの似たところ

使い方

- ユーザのプロファイルにインストール
  - `nix profile install` **`nixpkgs#hello`**
    - チャンネルを指定する必要あり
    - 同名パッケージがあっても混乱しない
  - `guix install` **`hello`**
    - チャンネル名を指定する必要なし
    - パッケージを一意に特定できるなら楽
    - そうでない場合はチャンネル間で混乱

発表後改変

再現性のあるビルド

- ハッシュ値を用いてソースコードの不変性を担保
- 1bit単位でのミスマッチすら許さない構え
  - たとえばGuixにはsubstituteと手元でのビルドを比較する `guix challenge` コマンドがある
  - もしかしたらnixにもあるかも
    - 追記: 以下のコマンドらしいです。
      - `nix build nixpkgs#hello --rebuild --keep-failed`
      - satlerさんのポストより知りました
        - https://x.com/satleri_sentler/status/1898629237734621486
        - https://reproducible.nixos.org

バイナリを落としてくる

- 全てを手元でビルドすると非常に遅い
- 信頼できるサーバがビルドした成果物を落としてきて、ビルドの代わりとする機能がある
  - そのような代替された成果物をsubstituteと呼ぶ
- GuixでもNixでもデフォルトで利用可能

パッケージを手軽に自分で書ける

Nixの場合：Nix言語（関数型，関数として定義）

```nix
{
  lib,
  stdenvNoCC,
  fetchFromGitHub,
  bash,
  makeWrapper,
  pciutils,
  x11Support ? !stdenvNoCC.isOpenBSD,
  ueberzug,
  fetchpatch,
}:

stdenvNoCC.mkDerivation rec {
  pname = "neofetch";
  version = "unstable-2021-12-10";

  src = fetchFromGitHub {
    owner = "dylanaraps";
    repo = "neofetch";
    rev = "ccd5d9f52609bbdcd5d8fa78c4fdb0f12954125f";
    sha256 = "sha256-9MoX6ykqvd2iB0VrZCfhSyhtztMpBTukeKejfAWYW1w=";
  };

  patches = [
    (fetchpatch {
      url = "https://github.com/dylanaraps/neofetch/commit/413c32e55dc16[illegible]0360f8e84af2b59fe45505f81b.patch";
      sha256 = "1fapdg9z79f0j3vw7fgi72b54aw4brn42bjsj48brbvg3ixsciph";
      name = "avoid_overwriting_gio_extra_modules_env_var.patch";
    })
    ...
  ];

  outputs = [
    "out"
    "man"
  ];

  strictDeps = true;
  buildInputs = [ bash ];
  nativeBuildInputs = [ makeWrapper ];
  postPatch = ''
    patchShebangs --host neofetch
  '';

  postInstall = ''
    wrapProgram $out/bin/neofetch \
      --prefix PATH : ${
        lib.makeBinPath (lib.optional (!stdenvNoCC.isOpenBSD) pciutils ++ lib.optional x11Support ueberzug)
      }
  '';

  makeFlags = [
    "PREFIX=${placeholder "out"}"
    "SYSCONFDIR=${placeholder "out"}/etc"
  ];

  meta = with lib; {
    description = "Fast, highly customizable system info script";
    homepage = "https://github.com/dylanaraps/neofetch";
    license = licenses.mit;
    platforms = platforms.all;
    maintainers = with maintainers; [ konimex ];
    mainProgram = "neofetch";
  };
}
```

依存

ビルド方法

ソース

パッチ

依存

ビルドフラグ

メタデータ

https://github.com/NixOS/nixpkgs/blob/master/pkgs/by-name/ne/neofetch/package.nix, 一部省略

パッケージを手軽に自分で書ける

Guixの場合: Scheme (Guile処理系, オブジェクトとして定義)

```
(define-public neofetch
  (package
    (name "neofetch")
    (version "7.1.0")
    (source (origin
              (method git-fetch)
              (uri (git-reference
                    (url "https://github.com/dylanaraps/neofetch")
                    (commit version)))
              (file-name (git-file-name name version))
              (sha256
               (base32
                "0i7wpisipwzk0j62pzaigbiq42y1mn4sbraz4my2jlz6ahwf00kv")))
    (build-system gnu-build-system)
    (arguments
     (list #:tests? #f                      ; there are no tests
           #:make-flags
           #~(list (string-append "PREFIX=" #$output))
           #:phases
           #~(modify-phases %standard-phases
               (delete 'configure))))         ; no configure script
    (home-page "https://github.com/dylanaraps/neofetch")
    (synopsis "System information script")
    (description "Neofetch is a command-line system information tool written in
Bash.  Neofetch displays information about your system next to an image, your OS
logo, or any ASCII file of your choice.  The main purpose of Neofetch is to be
used in screenshots to show other users what operating system or distribution
you are running, what theme or icon set you are using, etc.")
    (license license:expat)))
```

```
(inputs '(bash))
(native-inputs '(bash))
```

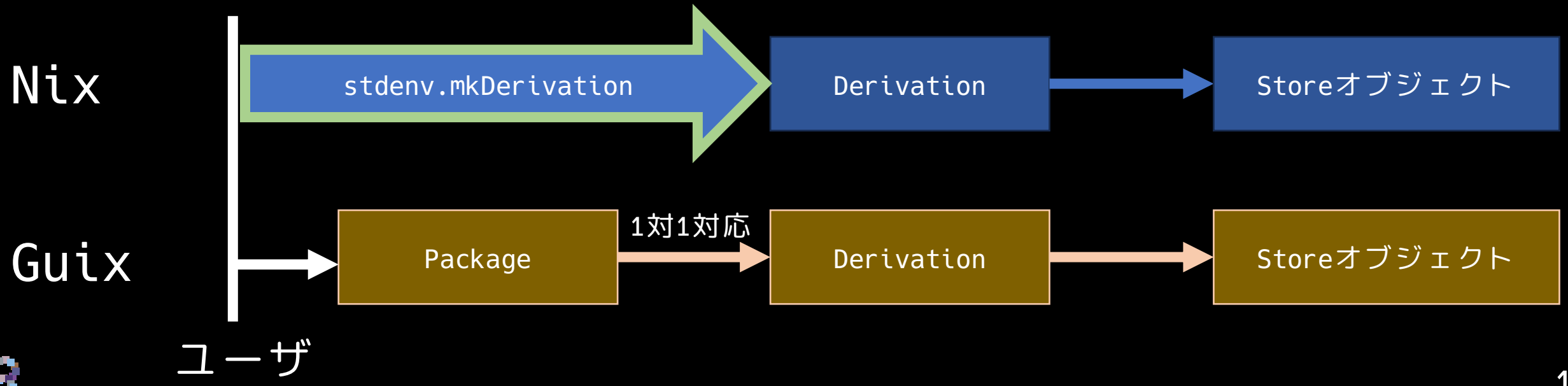
ソース
パッチ
(このパッケージにはない)
ビルド方法
ビルドフラグ
依存 (このパッケージにはない)
メタデータ

https://git.savannah.gnu.org/cgit/guix.git/tree/gnu/packages/admin.scm?id=591d2492e8517220db57098a6c9d1242533a8c8f#n4404 より抜粋

発表後改変

パッケージ定義の終着点はderivation

- Guixの場合も、最終的には**derivation**と呼ばれるパッケージのレシピへ変換される
- GuixのpackageとNixの~~derivation~~ stdenv.mkDerivationあたりが対応すると思ってもそんなに問題はなさそう



他のパッケージマネージャと干渉しない

- 隔離されたディレクトリにパッケージをインストールする
  - Nix: /nix/store/
  - Guix: /gnu/store/
    - /gnu/を使ってるの結構勇気ある

他のパッケージマネージャと干渉しない

- 実行ファイルに埋め込まれるパスは
  全てstore上の絶対パスを埋めこむ
  - 例えば
    - /usr/lib/ld.so.2 -> /nix/store/xxxxxxx-glibc-2.39/lib/ld-linux-x86-64.so.2
    - libcurl.so.4-> /nix/store/xxxxxxx-curl-8.6.0/lib/libcurl.so.4
- これにより、OS-wideに環境を変更しなくても動作が可能
  - とはいえ環境変数経由で干渉することはたまにある

ロールバック可能

- 戻したくなったらすぐに戻せる
  - パッケージいれたけどなんか動かない
- aptなどと違い、完全に元に戻せる！
  - しかも何世代か前のものも辿れる
- Nixの場合:
  - `nix profile rollback`

- Guixの場合
  - `guix package --roll-back`

お試し用shellを手軽に立ち上げ

- 例えば以下のコマンド一発で、helloコマンドが使えるシェルが立ち上がる！

- Nixの場合:
  - `nix shell nixpkgs#hello`

- Guixの場合:
  - `guix shell hello`

開発環境を用意する

- Nixの場合:
  - flake.nixを用意
    - flake-utilsを使うと短くなるらしい
  - nix develop
- Guixの場合:
  - manifest.scmを用意

(specifications->manifest '("hello"))

- --export-manifestでコマンドから自動生成可能

  - guix shell

例: helloパッケージのみを用意した環境

```
{
  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs?ref=nixpkgs-unstable";
  };

  outputs =
    inputs@{
      self,
      nixpkgs,
      ...
    }:
    let
      system = "x86_64-linux";

      # Macなら以下のように指定する
      # system = "x86_64-darwin";

      pkgs = nixpkgs.legacyPackages.${system};
    in {
      devShells.${system}.default = pkgs.mkShell {
        packages = [ pkgs.hello ];
      };
      packages.${system}.default = pkgs.hello;
    };
}
```

https://zenn.dev/comamoca/articles/2025-02-09-start-to-use-nix-flake-by-one-file より抜粋

Nixのよい点

圧倒的パッケージ数(&ユーザ数)

Repository	Total
nixpkgs unstable	102,889
nixpkgs stable 24.11	100,059
GNU Guix	30,138

https://repology.org/repositories/statistics/total より抜粋 (2025-03-09)

Non-freeのソフトウェアが手軽かつ豊富

- Nixの場合:
  - allowUnfreeを有効にするだけでよい
    https://nixos.org/manual/nixpkgs/stable/#sec-allow-unfree
- Guixの場合:
  - GNUプロジェクトなので、そもそも公式チャンネルに不自由ソフトウェアは存在しない
    - DRMの問題でFirefoxすら認められない
  - 非公式のNonguixに有名どころは揃っているが、足りない
    https://gitlab.com/nonguix/nonguix
    - Firefoxはこちらにある

macOSで利用できる

- NixはmacOSで利用可能
- Guixは現状Linuxのみ

- 人によっては致命的かも
- (どっちかWindowsで使えるようになって欲しい)

出力がカラフルでみやすい

nix --help

```
      ┃ Warning
      ┃ This program is experimental and its interface is subject to change.

Name

    nix - a tool for reproducible and declarative configuration management

Synopsis

    nix [option...] subcommand

    where subcommand is one of the following:

    Help commands:

      · nix help - show help about nix or a particular subcommand
      · nix help-stores - show help about store types and their settings

    Main commands:

      · nix build - build a derivation or fetch a store path
      · nix develop - run a bash shell that provides the build environment of a derivation
      · nix flake - manage Nix flakes
      · nix profile - manage Nix profiles
      · nix repl - start an interactive environment for evaluating Nix expressions
      · nix run - run a Nix application
      · nix search - search for packages
      · nix shell - run a shell in which the specified packages are available

    Infrequently used commands:

      · nix bundle - bundle an application so that it works outside of the Nix store
      · nix copy - copy paths between Nix stores
      · nix edit - open the Nix expression of a Nix package in $EDITOR
      · nix eval - evaluate a Nix expression
      · nix fmt - reformat your code in the standard style
      · nix log - show the build log of the specified packages or paths, if available
      · nix path-info - query information about store paths
      · nix registry - manage the flake registry
      · nix why-depends - show why a package has another package in its closure

    Utility/scripting commands:
```

guix --help

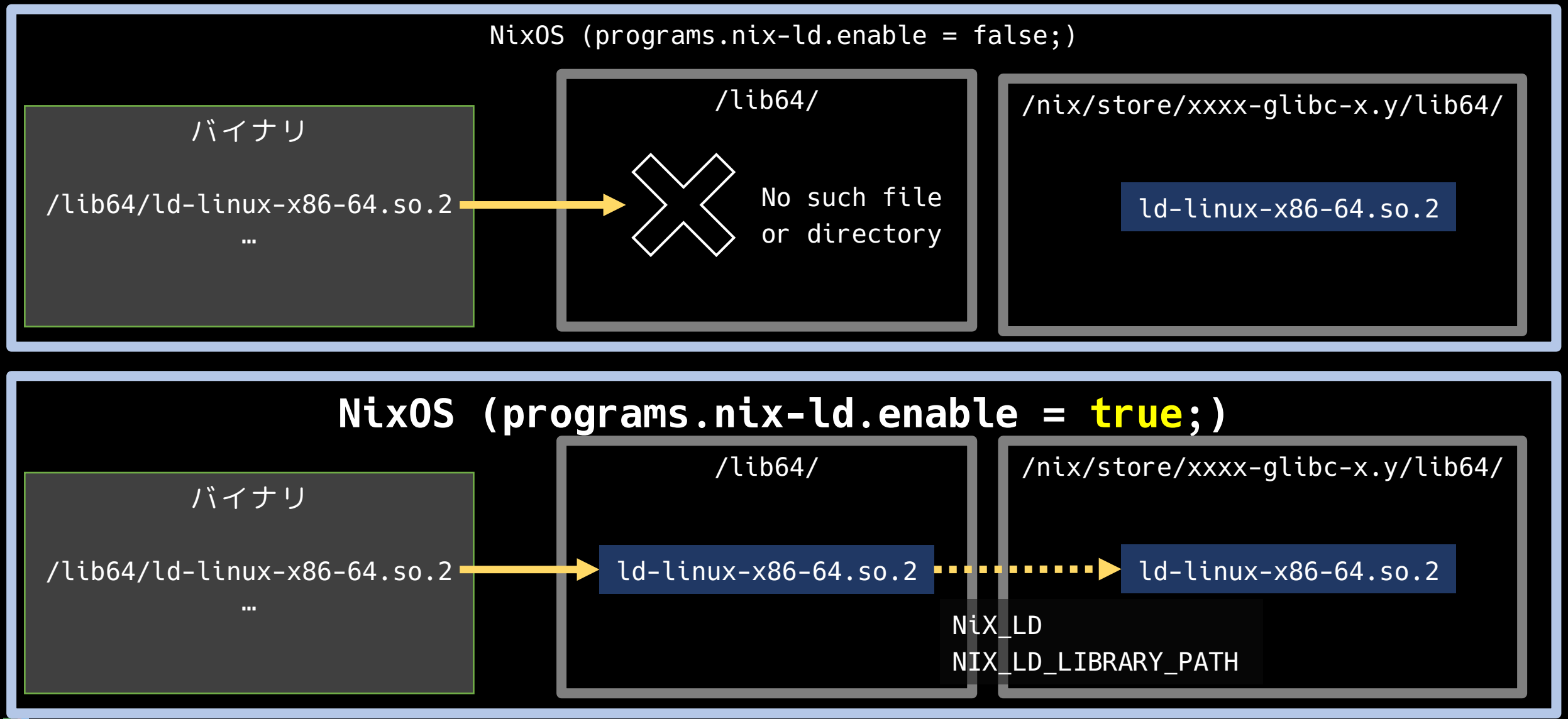
```
Usage: guix OPTION ¦ COMMAND ARGS...
COMMANDを実行します。ARGSが与えられればそれを引数とします。

  -h, --help             display this helpful text again and exit
  -V, --version          display version and copyright information and exit

COMMAND must be one of the sub-commands listed below:

  main commands
    deploy        deploy operating systems on a set of machines
    describe      describe the channel revisions currently used
    gc            invoke the garbage collector
    home          build and deploy home environments
    install       install packages
    locate        search for packages providing a given file
    package       manage packages and profiles
    pull          最新リビジョンのGuixをpullします
    remove        remove installed packages
    search        search for packages
    show          show information about packages
    system        build and deploy full operating systems
    time-machine  run commands from a different revision
    upgrade       各パッケージをそれぞれ最新のバージョンへアップグレードします
    weather       report on the availability of pre-built package binaries
    workflow      execute or visualize workflows

  software development commands
    container     run code in containers created by 'guix environment -C'
    environment   spawn one-off software environments (deprecated)
    pack          create application bundles
    shell         spawn one-off software environments

  packaging commands
    build      build packages or derivations without installing them
    challenge  challenge substitute servers, comparing their binaries
    download   download a file to the store and print its hash
    edit       view and edit package definitions
    graph      view and query package dependency graphs
    hash       compute the cryptographic hash of a file
    import     import a package definition from an external repository
```

NixOSで非nixのバイナリを動かせる

- `programs.nix-ld.enable = true`とすることで、そのへんに落ちてるバイナリを動かせるように！
  - https://github.com/nix-community/nix-ld
- Guix Systemに対応する機能はない

- もちろん自分でビルドしていないバイナリにリスクはある
- 一方、そのリスクを飲んで実行したいときもある
  - プロプライエタリなものとか

nix-ldの仕組み



- 詳しくは https://github.com/nix-community/nix-ld を参照

devenv

- Shellを上げるだけでなく、
  デーモンをあげたりもできるらしい
  - Docker compose相当？

- Guixでも理論上はできそうだけど、現状はない

Guixのよい点

Graftによるビルド回数の削減

- 再現性のため、Nix/Guixはパッケージが変更されるとそれに依存するパッケージは全てビルドし直される

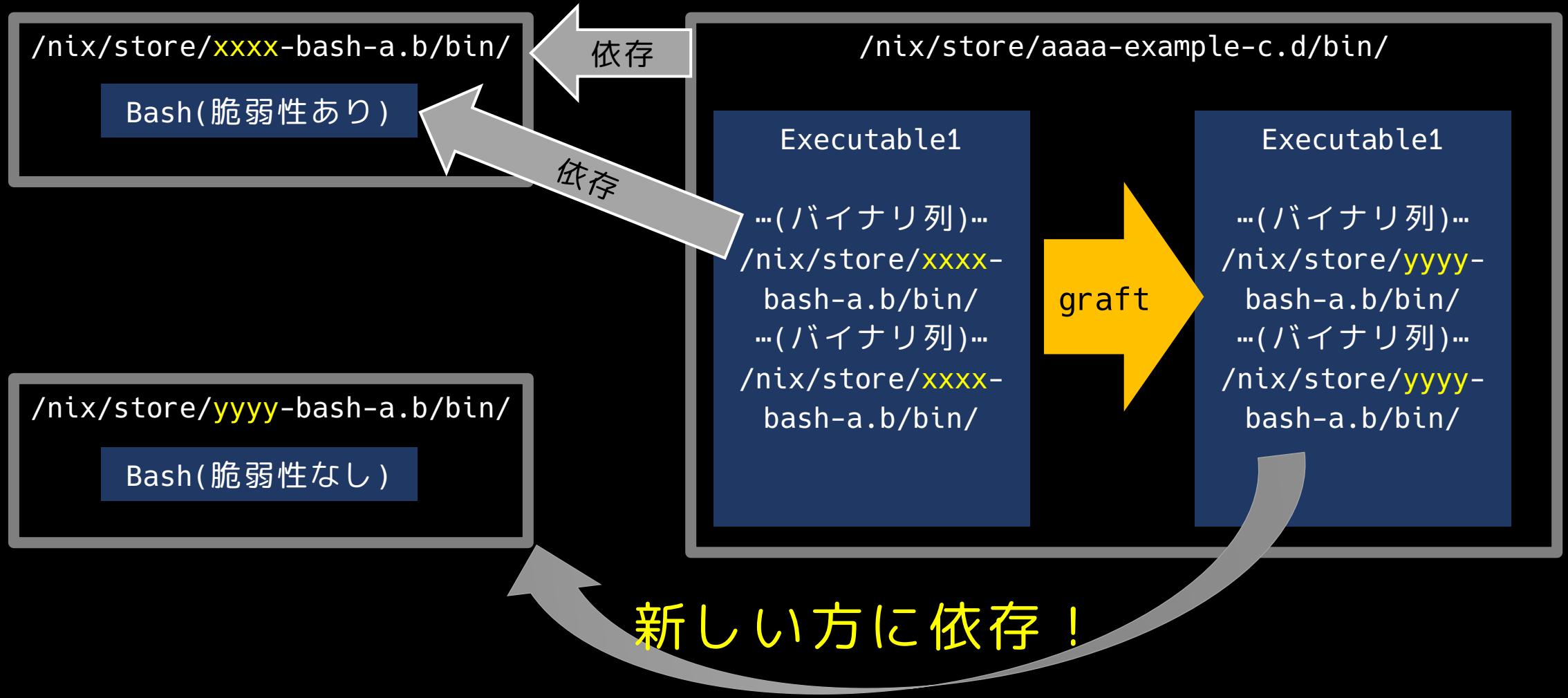
- bashのような広く使われているパッケージが脆弱性対応などで緊急アップデートされた場合、大量のパッケージがリビルドされてしまう
  - Bashやgccだとほぼ全部では？

Graftによるビルド回数の削減

- Guixにはそれを解決するべく、
  Graftという概念がある
  - 継ぎ木、移植などの意味合い

- 対象のパッケージに依存する成果物を全て走査
- 対象パッケージのパスを全てアップデート後のパスに置換する

Graftの仕組み



https://guix.gnu.org/blog/2020/grafts-continued/
https://guix.gnu.org/manual/devel/en/guix.html#Security-Updates
https://git.savannah.gnu.org/cgit/guix.git/tree/guix/build/graft.scm
あたりが参考になります

Statストームの解消

- Nix/Guixでは1依存1ディレクトリ
  →ld.soが探すべきディレクトリが依存数に比例
  - 通常は/usr/lib/だけ探せば済む

- 依存の依存なども考えると、ld.soは
  大量のディレクトリを巡回する必要がある
- しかも、ほとんどの場合絶対にないディレクトリ
  - 例えば、libcurlを探すために/gnu/store/xxx-ffmpeg-x.y/以下を探す

- Guixはld.so.cacheをうまく取り回して、これを解決

くわしくは https://guix.gnu.org/blog/2021/taming-the-stat-storm-with-a-loader-cache/ を参照

まとめ

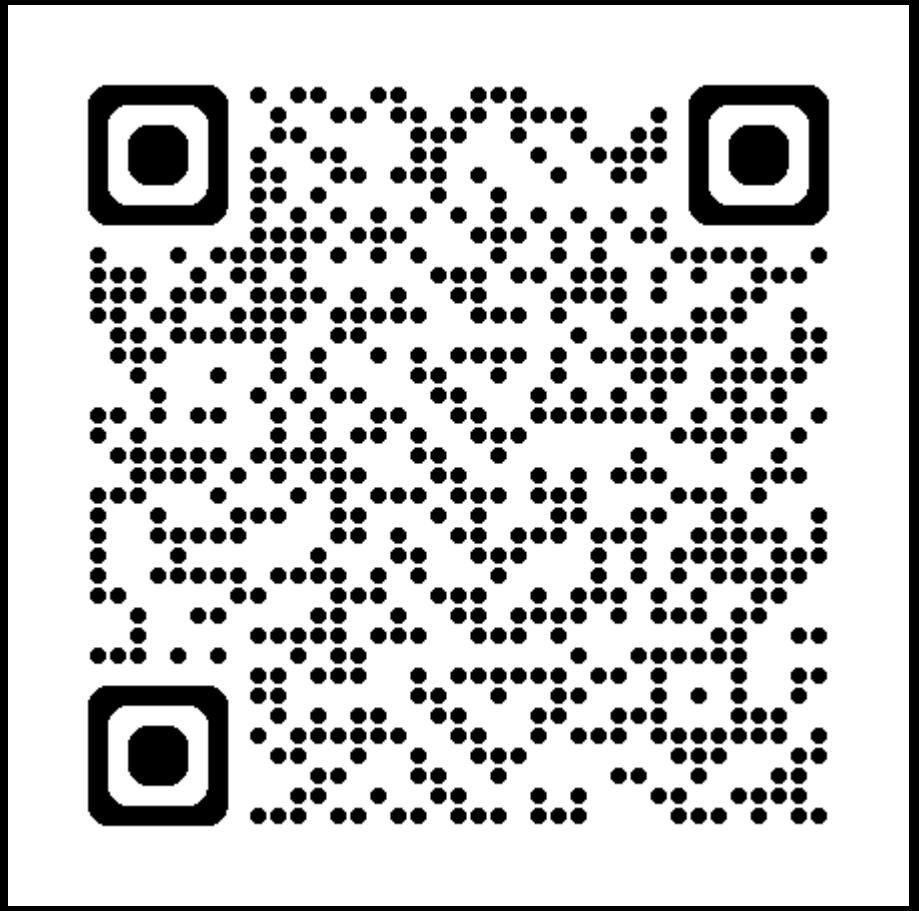
- Nixには後発のGuixにもない機能がたくさんある
  - Nix-ld、macOSサポート…
- Nixはユーザ数も多く、パッケージも多い
- 一方Guixもがんばっているところがある
  - Graft、statストームの解消…
- おたがいに高め合えると嬉しい

Reference

- “Dissecting Guix, Part 1: Derivations”, https://guix.gnu.org/blog/2023/dissecting-guix-part-1-derivations/
- “Derivations”, https://nix.dev/manual/nix/2.22/language/derivations
- “Glossary”, https://nix.dev/manual/nix/2.22/glossary

Guix-jpもあるよ

- https://guix-jp.gitlab.io/
- 数少ない日本語ユーザが集まってます
- 少しでも興味のある人は是非覗いてね
- Slack/Matrix/IRC



発表後追加

質問への回答など

Packageオブジェクトが存在する理由は何？

発表後追加

- Packageオブジェクトは単なるユーザインターフェースに過ぎず、実体はderivationです。
- そういう意味では、Guixのパッケージに対応するのはstdenv.mkDerivationのほうが近いかも
  - 該当の図を書き換えました
- 裏の構造はたぶんNixもGuixも大体同じです。

発表後追加

Lispに抵抗はなかったのか？

- 私は特に抵抗はなかった
  - EmacsでLispにどっぷり使った後だったため
- 一見構文がわかりにくいかもしれないが、実はどの言語もコンパイラやインタプリタが読み込むときに似たような形の構造に落とし込まれている(抽象構文木と呼ぶ)
- Lispはその構文木を(ほぼ)直接記述する形なので、文法は非常に簡単

発表後追加

Lispの文法

- 基本的な文法はこれくらい
  - 関数呼び出し：(関数名 引数1 引数2…)
    - 演算子もないため、足し算すら(+ 1 1)みたいになる
  - クオート：'任意の式
    - 書いたものを評価?したくないときに使う(後述)
  - 関数定義，変数定義：Lisp言語の種類による
    - Schemeでは：
      - (define 変数名 body)
      - (define (関数名 引数…) body)
    - 言語によってはdefvarやdefunのような単語を使う
      - 使い方は違うが基本的に関数呼び出しと同じ形

発表後追加

Lispにおける「評価」

- 評価とは
  - Lispでは書いた式は全てデータ(S式と呼ぶ)
  - 書いたデータはそのまま構文木として解釈、実行される(これを評価と呼ぶ)
  - データをデータのまま使いたいとき、つまり評価されて欲しくないときにクオートを付ける
    - i.e.クオートのついた式は評価すると元の式そのものを返す
- 例(矢印は評価&実行):
  - (+ 1 1) → 2
  - '(+ 1 1) → (+ 1 1) というデータ

発表後追加

Lispの文法まとめ

- こんな感じなので、文法は非常に少ない
  - 構文木=データであることを逆手にとり、関数呼び出しの見た目をした新しい構文を作りだすマクロという機能もある
    - これがあまりにも強力なため、なんでもできる
- ちゃんと知りたくなったらちゃんとした入門記事や本、処理系のチュートリアルを読んでね

発表後追加

Nixにはマクロがなくて悲しい

- 私がポロっと「マクロかな…」みたいに言ったやつ(specification->manifest)はただの関数でした、すみません
  - 文字列のリスト渡すだけなのでマクロである必要がない

- Nix言語でマクロが欲しいシチュエーションを聞いてみたいかも
  - Nix言語をほぼ全く書いたことがないため、Nix言語にマクロがないことによる苦しさがどのあたりにあるのかを知りたい

発表後追加

manifest.scmに バージョンロック機能はあるか

- manifest.scm自体にはないが、 channels.scmがその役割を果たす。

- Nixでは駆逐されつつあるチャンネルだが、 Guixでは現役で、主たるパッケージ提供方法。
  - Guixはパッケージ定義を単体で与えることも可能なので、 主たるという表現とした

発表後追加

チャンネルの定義

- チャンネルはパッケージの集合体で、gitレポジトリと対応する(Mercuryとかもいけるはず)
- channels.scmは以下のように利用したいチャンネルを列挙する

```
(list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (commit
          "e0fa68c7718fffd33d81af415279d6ddb518f727")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA")))…))
```

- url節にリポジトリの場所を書き、commit節にコミットハッシュを書く
  - commitではなくbranchにしてブランチ名を書けば、固定されていないチャンネルを表現できる
- introduction節は署名の情報で、レポジトリが本物かどうか検証できるようになっている(この節は任意)
- レポジトリにはパッケージ定義がたくさん書かれている。

発表後追加

manifest.scmにバージョンロック機能はあるか

- チャンネルの固定がパッケージの固定
  - つまり、各プロジェクトはmanifest.scmとchannels.scmを持つのが理想
- 利用方法としては以下のようになる
  - `guix time-machine -C channels.scm shell -m manifest.scm`
- guix time-machineによって過去のチャンネルを参照してmanifest.scmを実行する
  - 便宜上過去と言っているが、別に現在のチャンネルから戻って辿れる必要はない

発表後追加

Graftでパッケージは壊れないの？

- 壊れないことを保証はできない
  - 壊れないことを担保するのはパッケージ定義者の責任になる
- 基本的にはセキュリティ系のアップデートを即座に行うことが目的なので、大規模な変更についてはそもそもgraftで扱わない
  - 基本的には一から全てビルドをすることが正義
  - **graftはあくまでも緊急用**