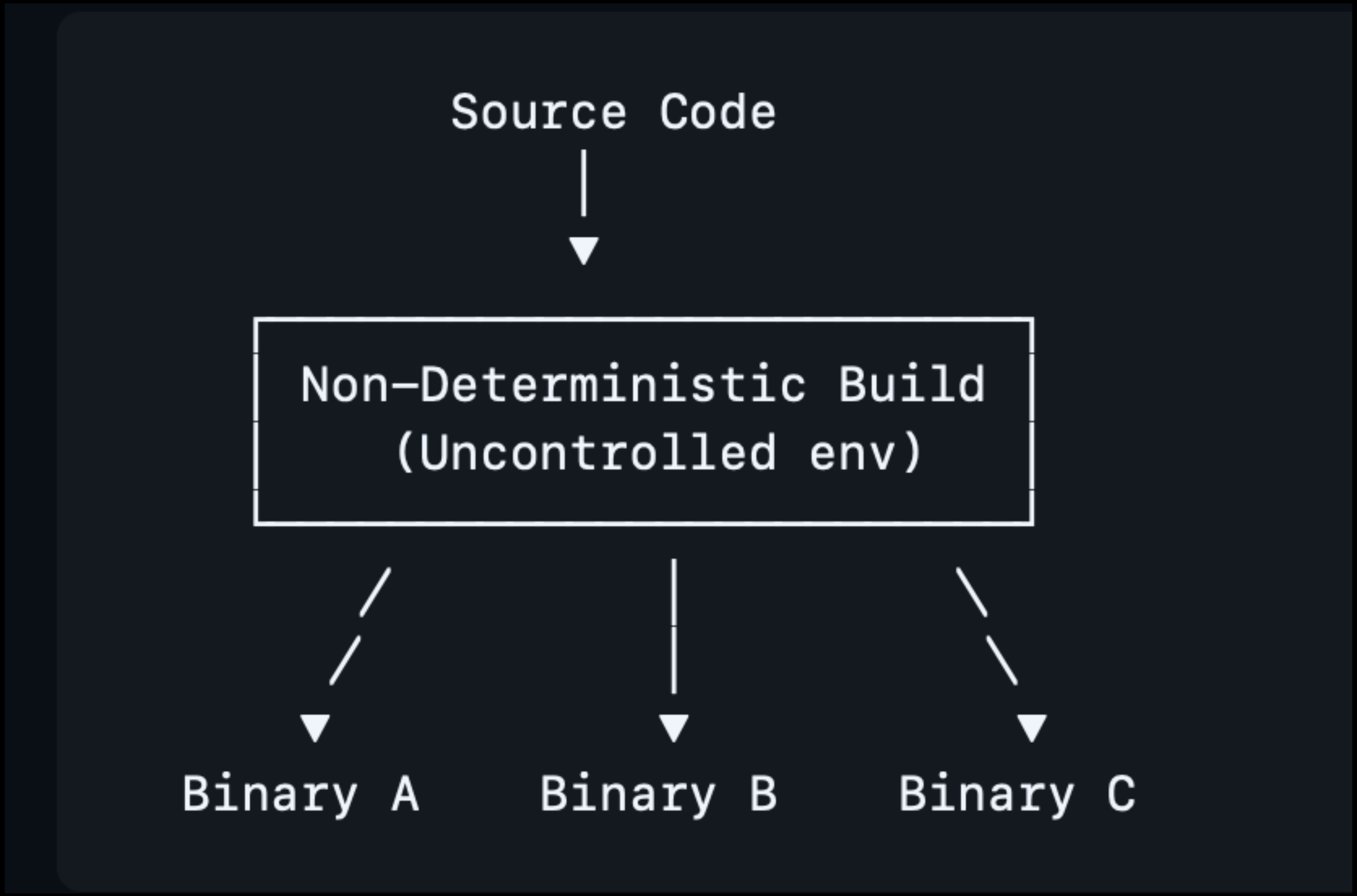
# On reproducibility

Czarek / Developer @ Cake Wallet / Feb 13th 2026

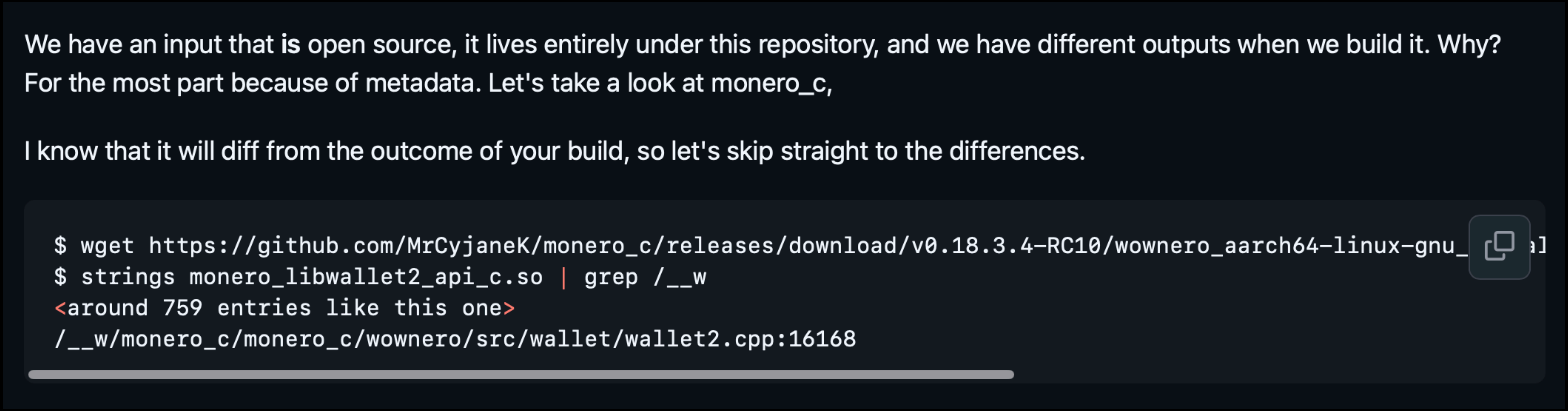
## What is a reproducible build?

## What is a non-reproducible build?

## What is a non-reproducible build?



## What is a non-reproducible build?

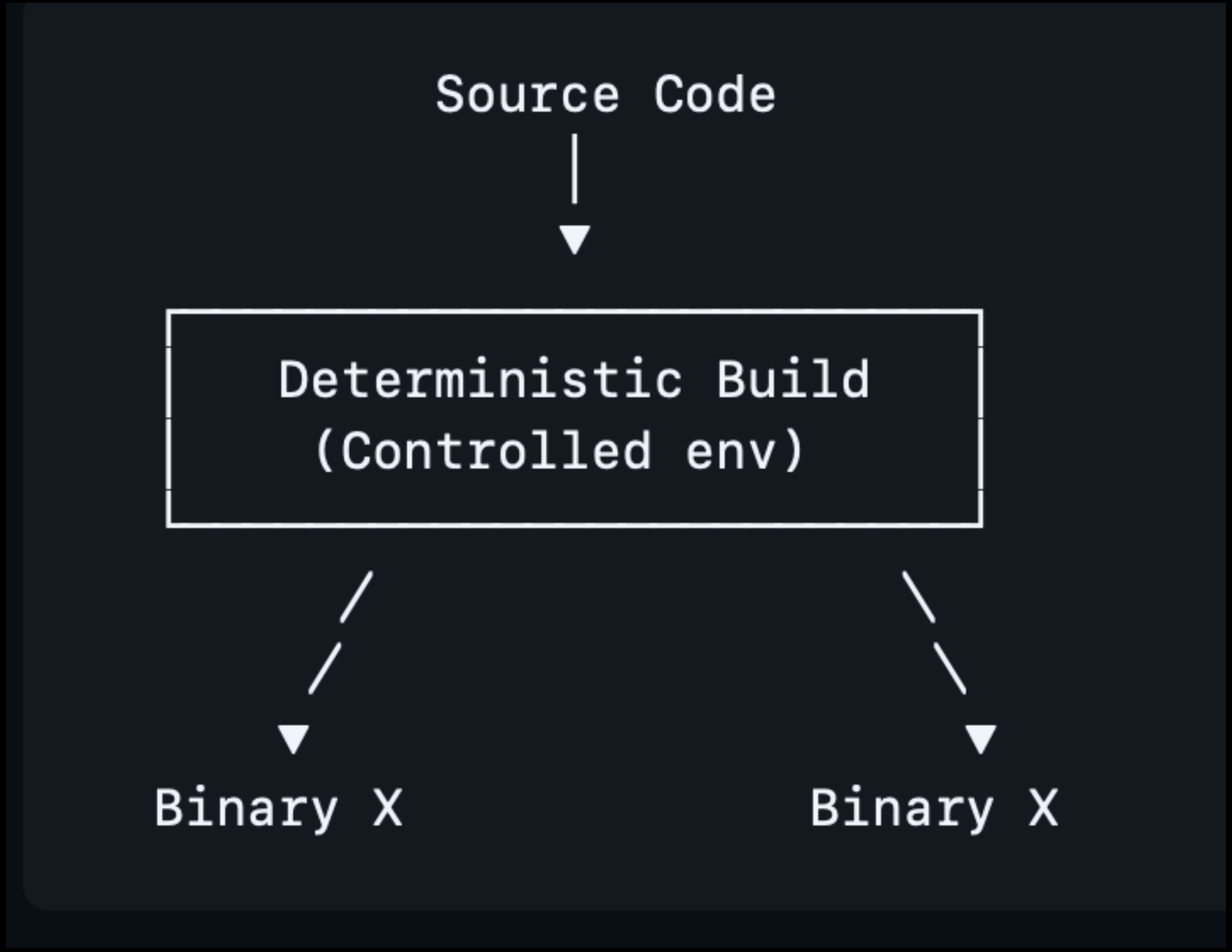


We have an input that **is** open source, it lives entirely under this repository, and we have different outputs when we build it. Why? For the most part because of metadata. Let's take a look at monero_c,

I know that it will diff from the outcome of your build, so let's skip straight to the differences.

```
$ wget https://github.com/MrCyjaneK/monero_c/releases/download/v0.18.3.4-RC10/wownero_aarch64-linux-gnu_  al
$ strings monero_libwallet2_api_c.so | grep /__w
<around 759 entries like this one>
/__w/monero_c/monero_c/wownero/src/wallet/wallet2.cpp:16168
```

## What is a reproducible build?



## Reproducibility at Cake Wallet

## How it started

## How it started

Bounty got opened to build Cake Wallet reproducibly

## How it started

Bounty got opened to build Cake Wallet reproducibly

(We didn’t know about it)

## How it started

Bounty got opened to build Cake Wallet reproducibly

(We didn’t know about it)

(The requirements were basically non-existent)

## How it started

Bounty got opened to build Cake Wallet reproducibly

(We didn’t know about it)

(The requirements were basically non-existent)

(Most efforts/PRs were bare minimum only)

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”*

* he didn’t say that exactly but you get the point.

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”

Me: :|

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”

Me: Uhhh

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”

Me: Probably a few days of work to get Android and Linux.

## Probably a few days of work to get Android and Linux.

### “Android and Linux”

We are primarily iOS wallet.

## Probably a few days of work to get Android and Linux.

### “Android and Linux”

We are primarily iOS wallet.

What about macOS and Windows?

## Probably a few days of work to get Android and Linux.

“a few days”

## Probably a few days of work to get Android and Linux.

**“a few days”**

It’s never a few days

## Probably a few days of work to get Android and Linux.

**“a few days”**

It’s never a few days

One extra build path (noticeably different from current ones)

## Probably a few days of work to get Android and Linux.

### “a few days”

It’s never a few days

One extra build path (noticeably different from current ones)

    -> one more testing vector

## Probably a few days of work to get Android and Linux.

**“a few days”**

It’s never a few days

One extra build path (noticeably different from current ones)

-> one more testing vector

-> one more thing to maintain

## Probably a few days of work to get Android and Linux.

**“a few days”**

It’s never a few days

One extra build path (noticeably different from current ones)

- -> one more testing vector
- -> one more thing to maintain
- -> significantly worse/different developer experience

## Why not “just do it”

- Spend a week to get it done

## Why not “just do it”

- Spend a week to get it done
- Get slowed down development for free

## Why not “just do it”

- Spend a week to get it done
- Get slowed down development for free
- Require testing on a different environment than development

## Why not “just do it”

- Spend a week to get it done
- Get slowed down development for free
- Require testing on a different environment than development
- Most of our dev machines are arm64 Macs

## Why not “just do it”

Essentially it would

- Make everyone unhappy

## Why not “just do it”

Essentially it would

- Make everyone unhappy
- Make developers life worse

## Why not “just do it”

Essentially it would

- Make everyone unhappy
- Make developers life worse
- Wouldn’t actually do what’s is supposed to do

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”*

Me: Probably a few days of work to get Android and Linux.

*thought process occurred*

Me: Let me research that in background and let you know soon.

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”*

Me: Probably a few days of work to get Android and Linux.

*thought process occurred*

Me: Let me research that in background and let you know soon.

(It was Q1 2025)

## How it started

Seth: “Hey Czarek, what would it take for us to get reproducible builds?”*

Me: Probably a few days of work to get Android and Linux.

*thought process occurred*

Me: Let me research that in background and let you know soon.

(It was Q1 2025)

*Subtle foreshadowing*

## About me

My name is Czarek

## About me

My name is Czarek

I love building software

## About me

My name is Czarek

I love *compiling* software

## About me

My name is Czarek

I love *compiling* software

I like reproducible builds

## About me

My name is Czarek

I love *compiling* software

I like reproducible builds

I have to rewrite/cleanup/update contrib/depends

## About me

My name is Czarek

I love *compiling* software

I like reproducible builds

I have to rewrite/cleanup/update contrib/depends

I have heavily underestimated the work required to achieve reproducible builds

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go

+ Rust

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go (Gomobile + C ABI)

+ Rust (flutter_rust_bridge + C ABI)

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go (Gomobile + C ABI)

+ Rust (flutter_rust_bridge + C ABI)

Swift/ObjC = some dependencies built by CocoaPods

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go (Gomobile + C ABI)

+ Rust (flutter_rust_bridge + C ABI)

Swift/ObjC = some dependencies built by CocoaPods

C++ = some dependencies built by CMake, Gradle

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go (Gomobile + C ABI)

+ Rust (flutter_rust_bridge + C ABI)

Swift/ObjC = some dependencies built by CocoaPods

C++ = some dependencies built by CMake, Gradle

Some dependencies ship prebuilt libraries

## About Cake Wallet

Old and big project

Flutter/Dart = Java, Kotlin, Swift, ObjC, C++, C (depending on platform)

+ Go (Gomobile + C ABI)

+ Rust (flutter_rust_bridge + C ABI)

Swift/ObjC = some dependencies built by CocoaPods

C++ = some dependencies built by CMake, Gradle

Some dependencies ship prebuilt libraries (which we remove)

# Let’s do it!

**Enough talking, I want reproducible builds**

## Computers are deterministic

some input -> some output

## Computers are deterministic

some input -> some output (always the same)

## Computers are deterministic

**Are they? Let’s check!**

## Computers are deterministic

**Are they? Let’s check!**

1. Power on the laptop

## Computers are deterministic

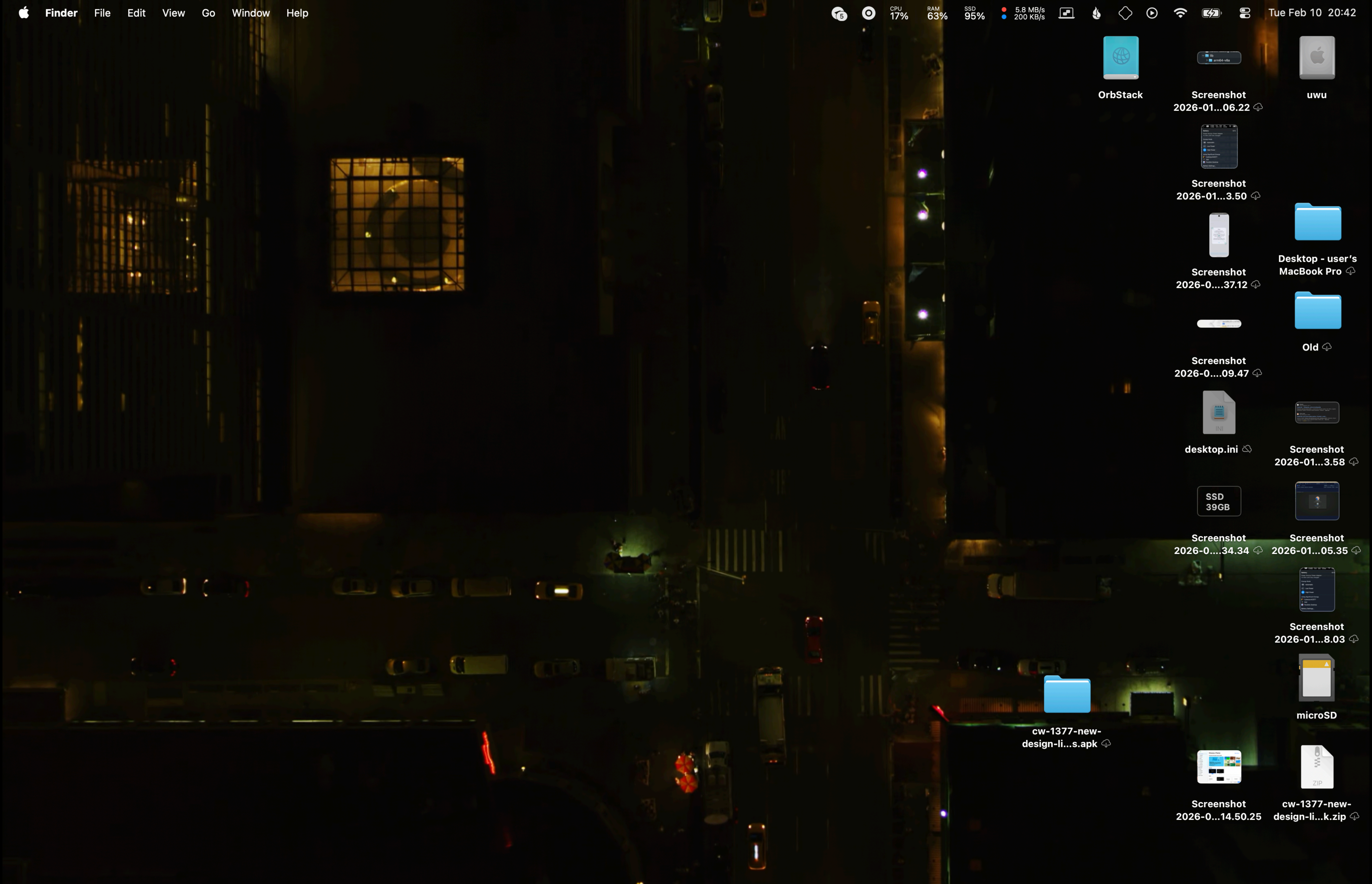
**Are they? Let's check!**

1. Power on the laptop
2. Type in your password

## Computers are deterministic

**Are they? Let’s check!**

1. Power on the laptop
2. Type in your password
3. Take a screenshot of the desktop


Finder
File
Edit
View
Go
Window
Help
CPU 17%
RAM 63%
SSD 95%
5.8 MB/s
200 KB/s
Tue Feb 10 20:42
OrbStack
Screenshot 2026-01...06.22
uwu
Screenshot 2026-01...3.50
Screenshot 2026-0....37.12
Desktop - user's MacBook Pro
Old
Screenshot 2026-0....09.47
desktop.ini
Screenshot 2026-01...3.58
SSD 39GB
Screenshot 2026-0....34.34
Screenshot 2026-01...05.35
Screenshot 2026-01...8.03
cw-1377-new-design-li...s.apk
microSD
Screenshot 2026-0...14.50.25
cw-1377-new-design-li...k.zip

Recycle Bin
Screenshot 2026-01-...
cw-1377-ne...
Mac Files
Screenshot 2026-01-...
Screenshot 2026-01-...
Microsoft Edge
Screenshot 2026-02-...
cw-1377-ne...
Screenshot 2026-02-...
Desktop - user's Mac...
Screenshot 2026-02-...
Old
Screenshot 2026-02-...
Screenshot 2026-02-...
Screenshot 2026-01-...
Screenshot 2026-02-...
Screenshot 2026-01-...
Screenshot 2026-02-...
31°F
Clear
Search
9:03 PM
2/10/2026

## The environment

Reduce amount of variables

## The environment

Reduce amount of variables

Define those that can’t be removed

## The environment

Reduce amount of variables

Define those that can’t be removed

That’s it.

## The environment

Reduce amount of variables

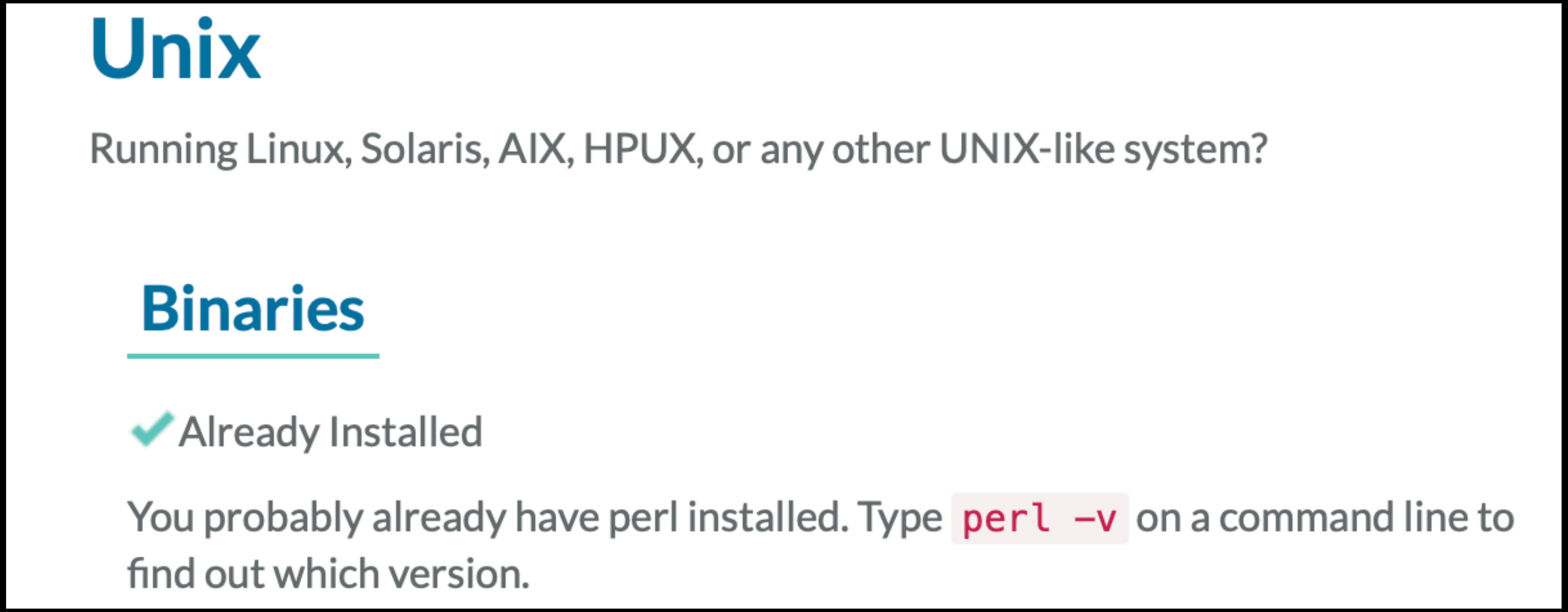
Define those that can’t be removed

That’s it. It’s just a lot more than you probably expect.

## Real world examples

**Based on a real world software that everyones uses**

## Quiz time: perl



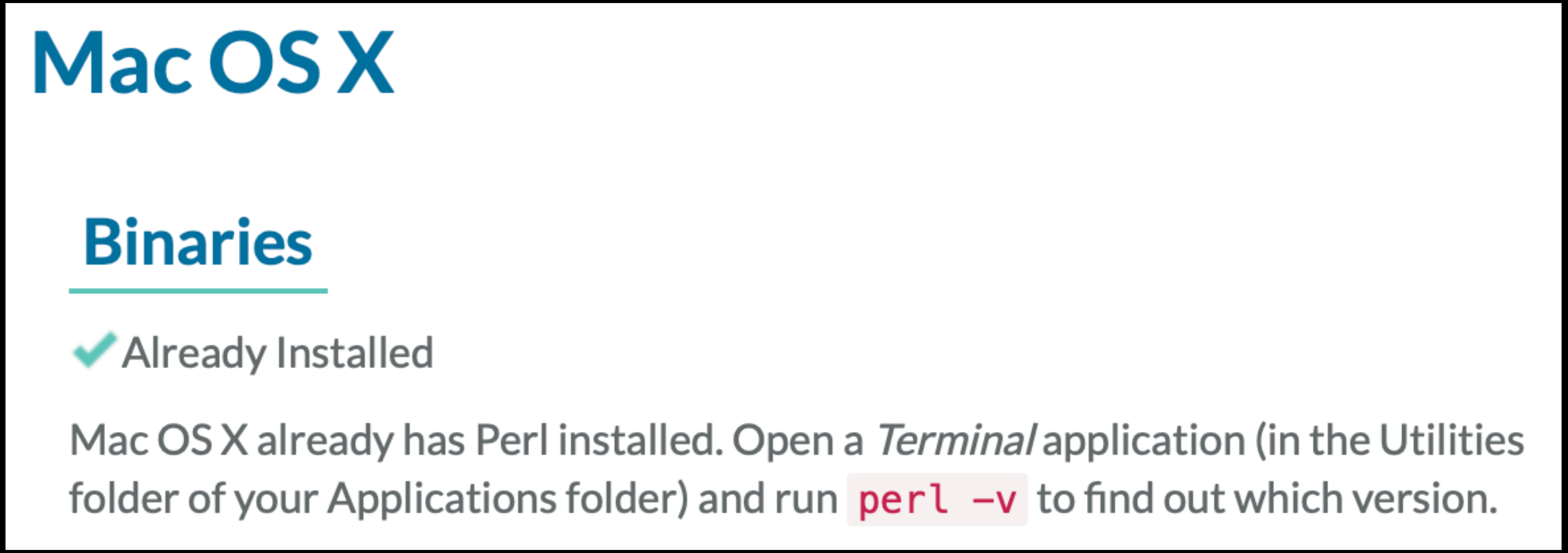
Unix

Running Linux, Solaris, AIX, HPUX, or any other UNIX-like system?

Binaries

✓ Already Installed

You probably already have perl installed. Type `perl -v` on a command line to find out which version.



Mac OS X

Binaries

✓ Already Installed

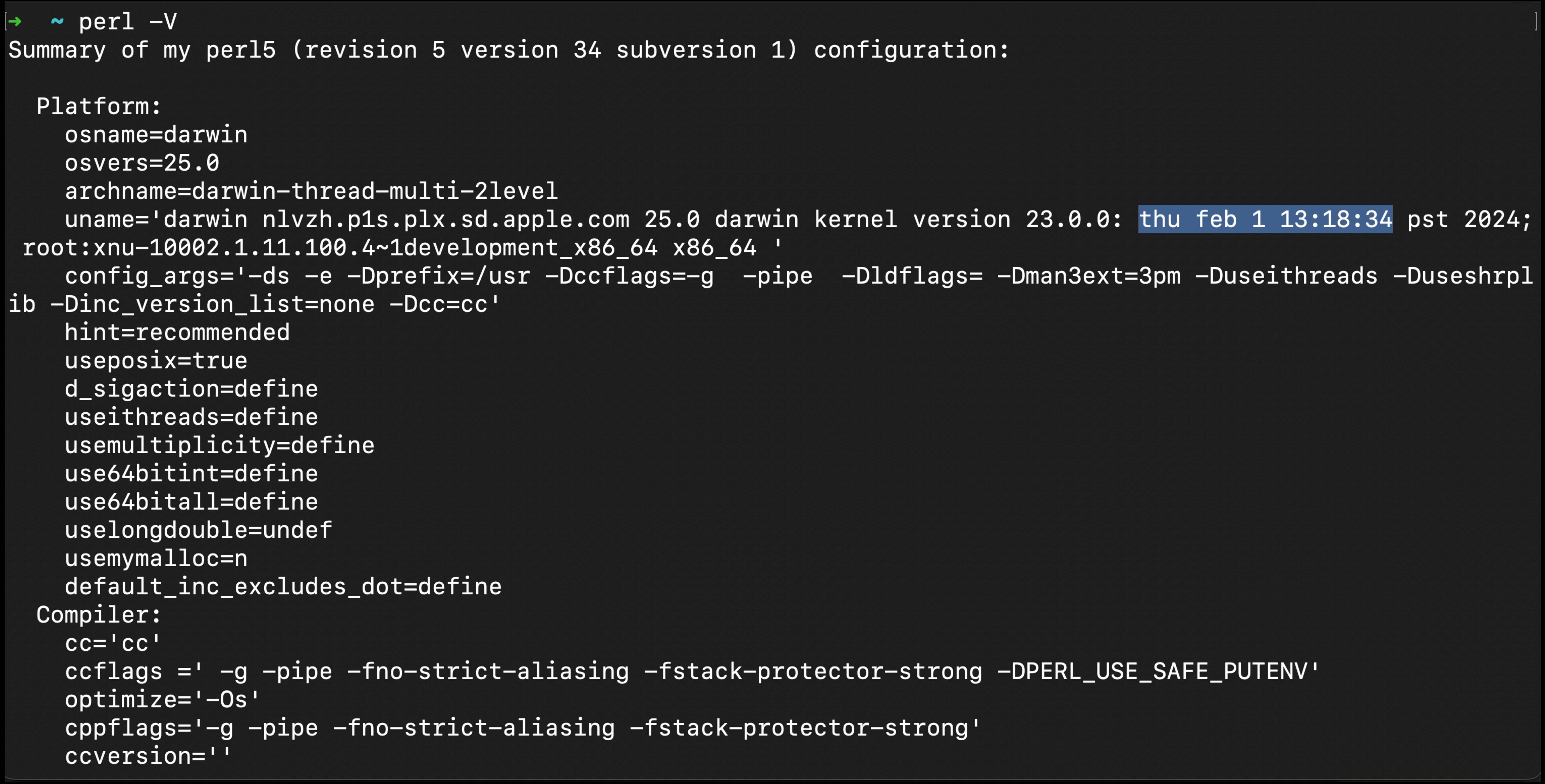
Mac OS X already has Perl installed. Open a *Terminal* application (in the Utilities folder of your Applications folder) and run `perl -v` to find out which version.

## Quiz time: perl

Assuming two identical laptops, with the disks being bit-for-bit perfect (the same OS and software) built perl in the same directory using the same tools will the build be reproducible?

# Is perl reproducible: no

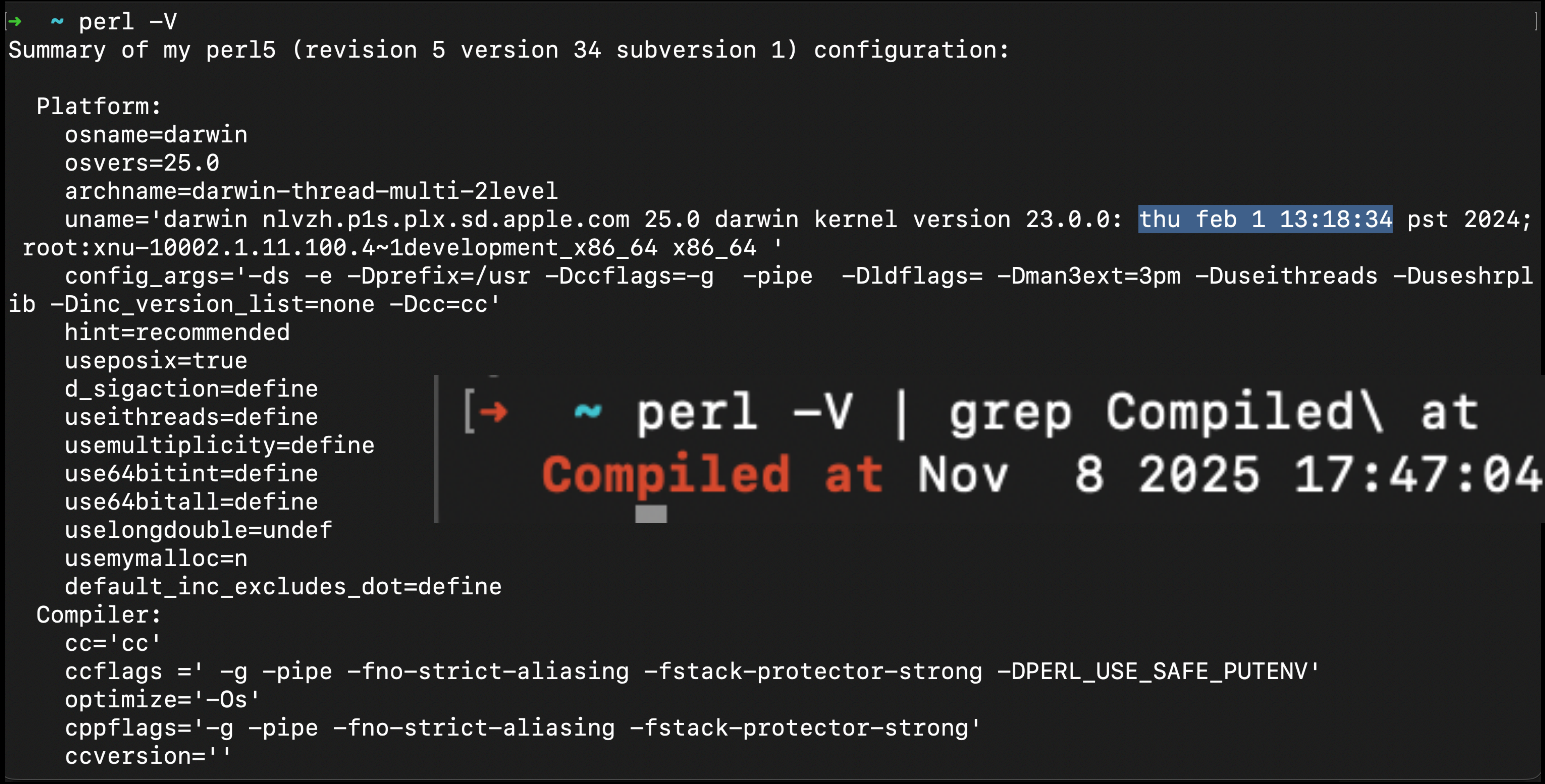
No. It will be different.



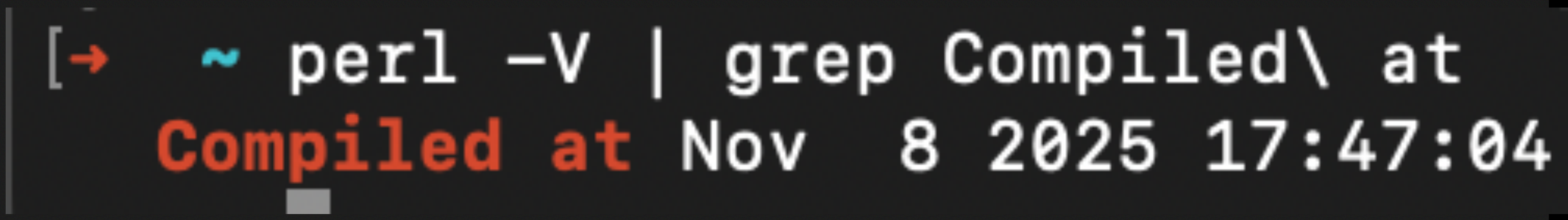
```
→  ~ perl -V
Summary of my perl5 (revision 5 version 34 subversion 1) configuration:
   
  Platform:
    osname=darwin
    osvers=25.0
    archname=darwin-thread-multi-2level
    uname='darwin nlvzh.p1s.plx.sd.apple.com 25.0 darwin kernel version 23.0.0: thu feb 1 13:18:34 pst 2024;
 root:xnu-10002.1.11.100.4~1development_x86_64 x86_64 '
    config_args='-ds -e -Dprefix=/usr -Dccflags=-g  -pipe  -Dldflags= -Dman3ext=3pm -Duseithreads -Duseshrpl
ib -Dinc_version_list=none -Dcc=cc'
    hint=recommended
    useposix=true
    d_sigaction=define
    useithreads=define
    usemultiplicity=define
    use64bitint=define
    use64bitall=define
    uselongdouble=undef
    usemymalloc=n
    default_inc_excludes_dot=define
  Compiler:
    cc='cc'
    ccflags =' -g -pipe -fno-strict-aliasing -fstack-protector-strong -DPERL_USE_SAFE_PUTENV'
    optimize='-Os'
    cppflags='-g -pipe -fno-strict-aliasing -fstack-protector-strong'
    ccversion=''
```

# Is perl reproducible: no

No. It will be different.



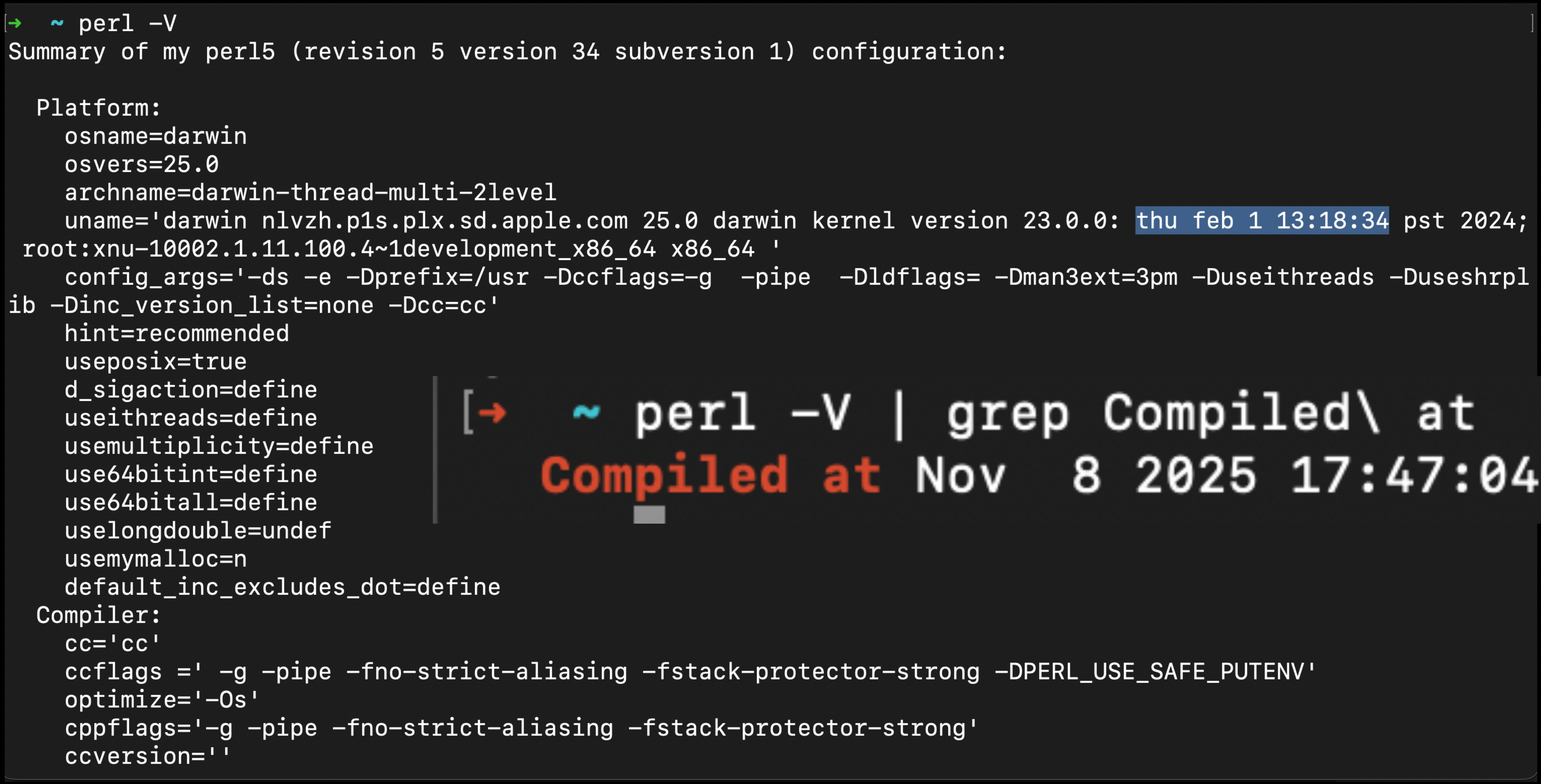
```
→  ~ perl -V
Summary of my perl5 (revision 5 version 34 subversion 1) configuration:

  Platform:
    osname=darwin
    osvers=25.0
    archname=darwin-thread-multi-2level
    uname='darwin nlvzh.p1s.plx.sd.apple.com 25.0 darwin kernel version 23.0.0: thu feb 1 13:18:34 pst 2024;
 root:xnu-10002.1.11.100.4~1development_x86_64 x86_64 '
    config_args='-ds -e -Dprefix=/usr -Dccflags=-g  -pipe  -Dldflags= -Dman3ext=3pm -Duseithreads -Duseshrpl
ib -Dinc_version_list=none -Dcc=cc'
    hint=recommended
    useposix=true
    d_sigaction=define
    useithreads=define
    usemultiplicity=define
    use64bitint=define
    use64bitall=define
    uselongdouble=undef
    usemymalloc=n
    default_inc_excludes_dot=define
  Compiler:
    cc='cc'
    ccflags =' -g -pipe -fno-strict-aliasing -fstack-protector-strong -DPERL_USE_SAFE_PUTENV'
    optimize='-Os'
    cppflags='-g -pipe -fno-strict-aliasing -fstack-protector-strong'
    ccversion=''
```



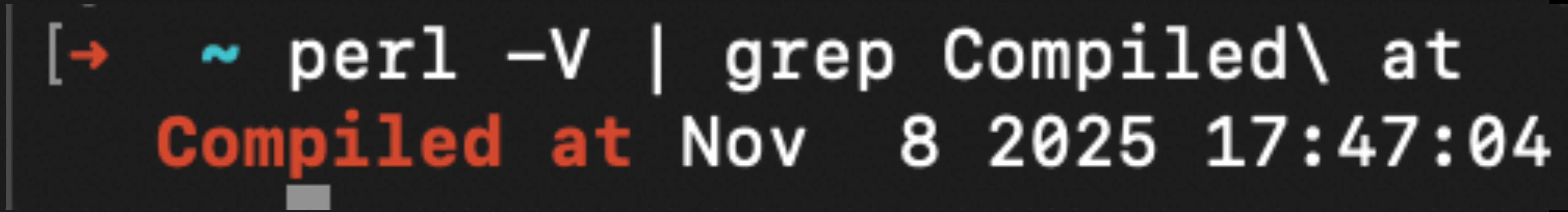
```
[→  ~ perl -V | grep Compiled\ at
  Compiled at Nov  8 2025 17:47:04
```

# Is perl reproducible: no

No. It will be different.



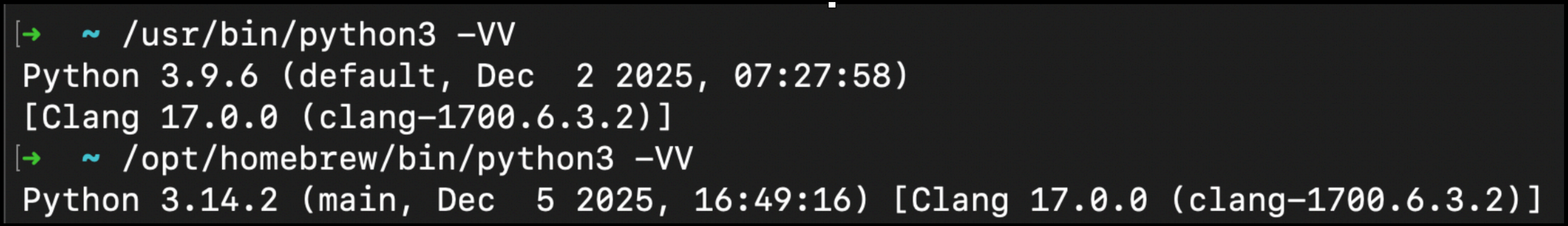
```
→  ~ perl -V
Summary of my perl5 (revision 5 version 34 subversion 1) configuration:

  Platform:
    osname=darwin
    osvers=25.0
    archname=darwin-thread-multi-2level
    uname='darwin nlvzh.p1s.plx.sd.apple.com 25.0 darwin kernel version 23.0.0: thu feb 1 13:18:34 pst 2024;
 root:xnu-10002.1.11.100.4~1development_x86_64 x86_64 '
    config_args='-ds -e -Dprefix=/usr -Dccflags=-g  -pipe  -Dldflags= -Dman3ext=3pm -Duseithreads -Duseshrpl
ib -Dinc_version_list=none -Dcc=cc'
    hint=recommended
    useposix=true
    d_sigaction=define
    useithreads=define
    usemultiplicity=define
    use64bitint=define
    use64bitall=define
    uselongdouble=undef
    usemymalloc=n
    default_inc_excludes_dot=define
  Compiler:
    cc='cc'
    ccflags =' -g -pipe -fno-strict-aliasing -fstack-protector-strong -DPERL_USE_SAFE_PUTENV'
    optimize='-Os'
    cppflags='-g -pipe -fno-strict-aliasing -fstack-protector-strong'
    ccversion=''
```



```
[→  ~ perl -V | grep Compiled\ at
  Compiled at Nov  8 2025 17:47:04
```

## Quiz time: python3

Assuming two identical laptops, with the disks being bit-for-bit perfect (the same OS and software) built python3 in the same directory using the same tools will the build be reproducible?



```
[→  ~ /usr/bin/python3 -VV
Python 3.9.6 (default, Dec  2 2025, 07:27:58)
[Clang 17.0.0 (clang-1700.6.3.2)]
[→  ~ /opt/homebrew/bin/python3 -VV
Python 3.14.2 (main, Dec  5 2025, 16:49:16) [Clang 17.0.0 (clang-1700.6.3.2)]
```

## Quiz time: yt-dlp

**Yt-dlp is a video streaming platform downloader written in python3**

Assuming two identical laptops, with the disks being bit-for-bit perfect (the same OS and software) built yt-dlp in the same directory using the same tools will the build be reproducible?

# Maybe???

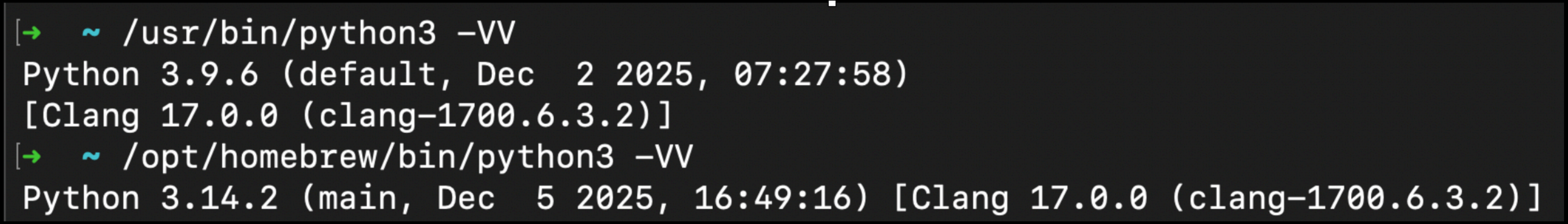
```
➜  yt-dlp_macos strings $(find . -type f) 2>&1 | grep '[0-1][0-9]:[0-6][0-9]:[0-6][0-9] ' | head -4
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
```

## Nevermind...

```
➜  yt-dlp_macos strings $(find . -type f) 2>&1 | grep '[0-1][0-9]:[0-6][0-9]:[0-6][0-9] ' | head -6
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
built on: Fri Dec  5 17:07:45 2025 UTC
built on: Fri Dec  5 17:04:22 2025 UTC
```

## Nevermind...

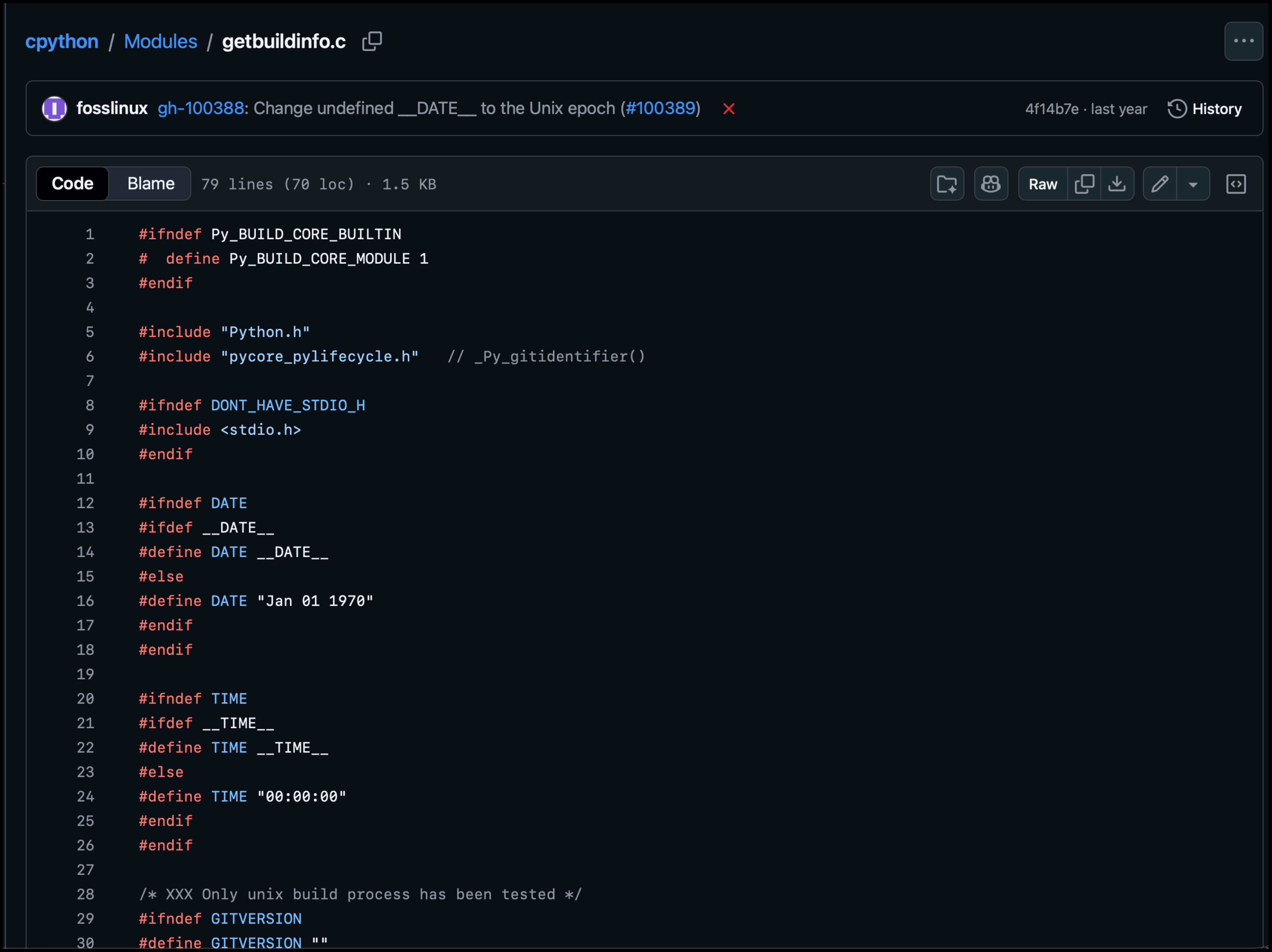
```
[→  yt-dlp_macos strings $(find . -type f) 2>&1 | grep '[0-1][0-9]:[0-6][0-9]:[0-6][0-9] ' | head -6
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
It is January 1, 1970, 00:00:00 (UTC) on all platforms.
Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
built on: Fri Dec  5 17:07:45 2025 UTC
built on: Fri Dec  5 17:04:22 2025 UTC
```



```
[→  ~ /usr/bin/python3 -VV
Python 3.9.6 (default, Dec  2 2025, 07:27:58)
[Clang 17.0.0 (clang-1700.6.3.2)]
[→  ~ /opt/homebrew/bin/python3 -VV
Python 3.14.2 (main, Dec  5 2025, 16:49:16) [Clang 17.0.0 (clang-1700.6.3.2)]
```

## Given enough will…

# Given enough will…



fosslinux gh-100388: Change undefined __DATE__ to the Unix epoch (#100389)

4f14b7e · last year

79 lines (70 loc) · 1.5 KB

```c
#ifndef Py_BUILD_CORE_BUILTIN
#  define Py_BUILD_CORE_MODULE 1
#endif

#include "Python.h"
#include "pycore_pylifecycle.h"   // _Py_gitidentifier()

#ifndef DONT_HAVE_STDIO_H
#include <stdio.h>
#endif

#ifndef DATE
#ifdef __DATE__
#define DATE __DATE__
#else
#define DATE "Jan 01 1970"
#endif
#endif

#ifndef TIME
#ifdef __TIME__
#define TIME __TIME__
#else
#define TIME "00:00:00"
#endif
#endif

/* XXX Only unix build process has been tested */
#ifndef GITVERSION
#define GITVERSION ""
```

## Given enough will…

- Patch python to fake the build date

## Given enough will…

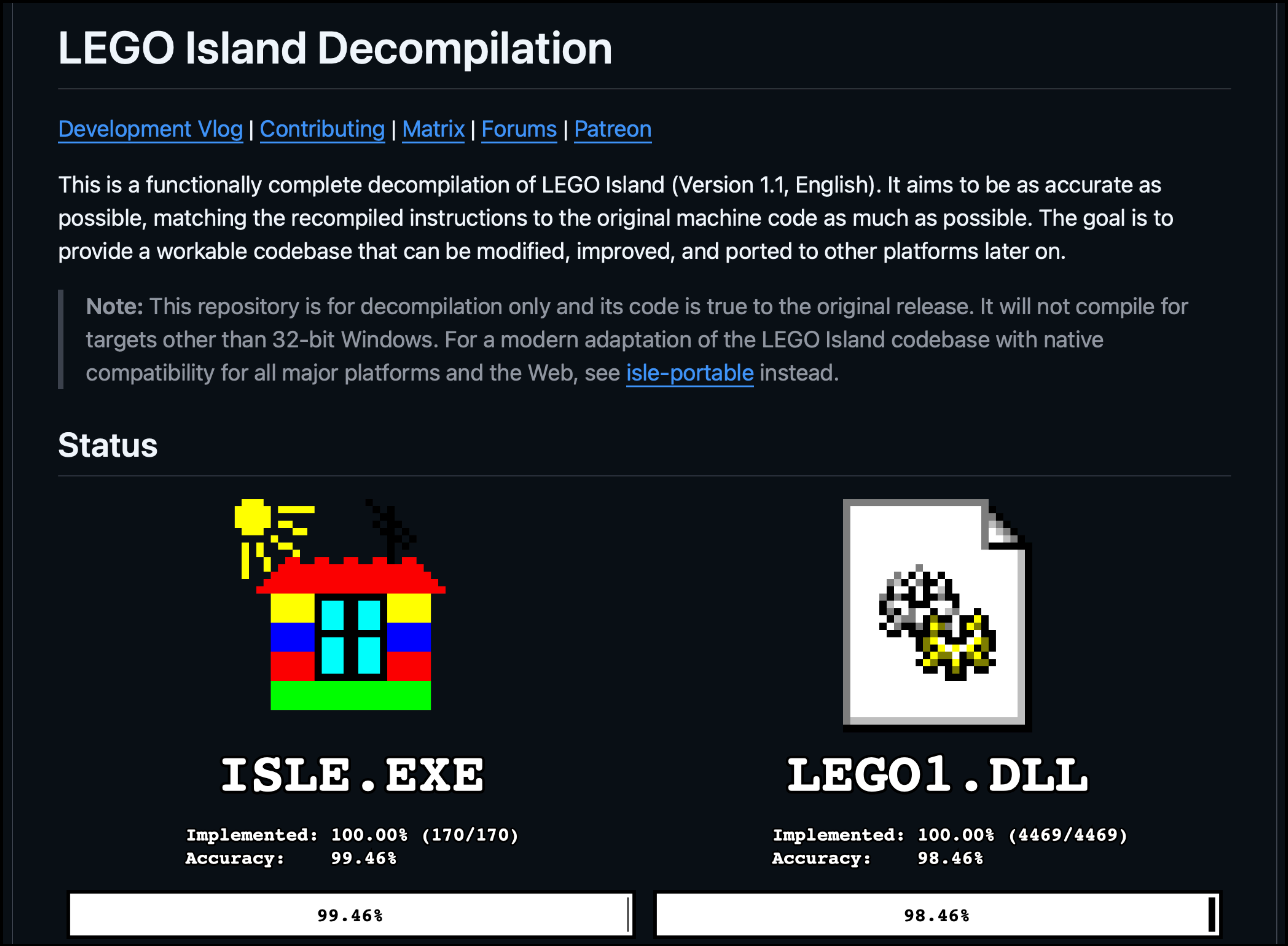
- Patch python to fake the build date
- Build the app in the same directory as the developer

## Given enough will…

- Patch python to fake the build date
- Build the app in the same directory as the developer
- Look for other differences -> patch them the same way…

## Given enough will…

- Patch python to fake the build date
- Build the app in the same directory as the developer
- Look for other differences -> patch them the same way…
- We don’t even need source code…



# LEGO Island Decompilation

Development Vlog | Contributing | Matrix | Forums | Patreon

This is a functionally complete decompilation of LEGO Island (Version 1.1, English). It aims to be as accurate as possible, matching the recompiled instructions to the original machine code as much as possible. The goal is to provide a workable codebase that can be modified, improved, and ported to other platforms later on.

> **Note:** This repository is for decompilation only and its code is true to the original release. It will not compile for targets other than 32-bit Windows. For a modern adaptation of the LEGO Island codebase with native compatibility for all major platforms and the Web, see isle-portable instead.

## Status

**ISLE.EXE**

```
Implemented: 100.00% (170/170)
Accuracy:    99.46%
```

99.46%

**LEGO1.DLL**

```
Implemented: 100.00% (4469/4469)
Accuracy:    98.46%
```

98.46%

## It’s easier the other way around

Instead of patching python3 for everything that uses python3, let’s use the same python3 (with the same metadata) everywhere?

```
[➜  ~ /usr/bin/python3 -VV
Python 3.9.6 (default, Dec  2 2025, 07:27:58)
[Clang 17.0.0 (clang-1700.6.3.2)]
[➜  ~ /opt/homebrew/bin/python3 -VV
Python 3.14.2 (main, Dec  5 2025, 16:49:16) [Clang 17.0.0 (clang-1700.6.3.2)]
[➜  ~ /opt/_/native/bin/python3 -VV
Python 3.14.0 (main, Jan  1 1970, 00:00:01) [Clang 17.0.0 (clang-1700.6.3.2)]
➜  ~ 
```

## It’s easier the other way around

Instead of patching python3 for everything that uses python3, let’s use the same python3 (with the same metadata) everywhere?

- Build directory

## It’s easier the other way around

Instead of patching python3 for everything that uses python3, let’s use the same python3 (with the same metadata) everywhere?

- Build directory
- Build environment (OS, library versions, compiler, etc.)

## It’s easier the other way around

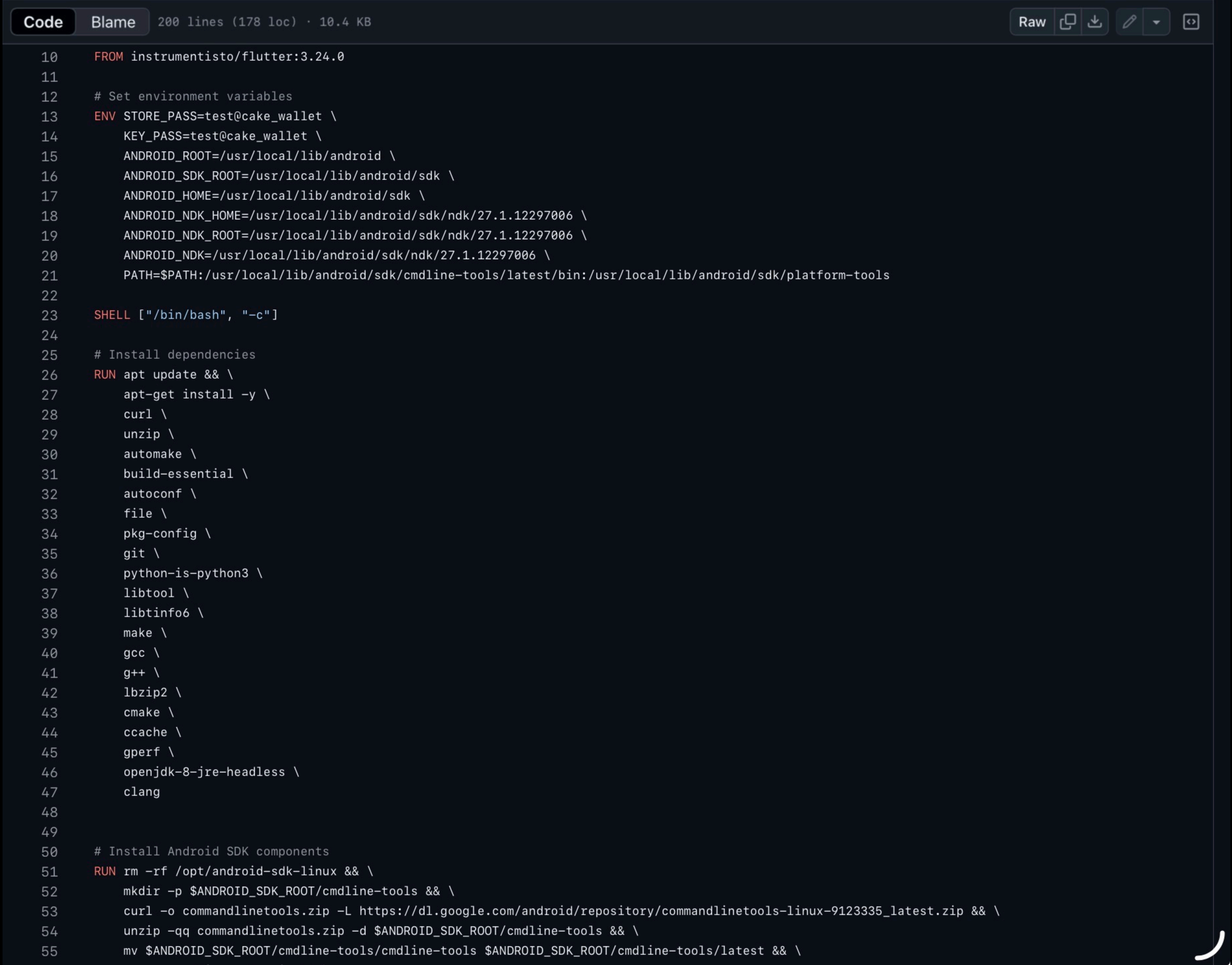
Instead of patching python3 for everything that uses python3, let’s use the same python3 (with the same metadata) everywhere?

- Build directory
- Build environment (OS, library versions, compiler, etc.)
- Patches to get rid of non-deterministic code

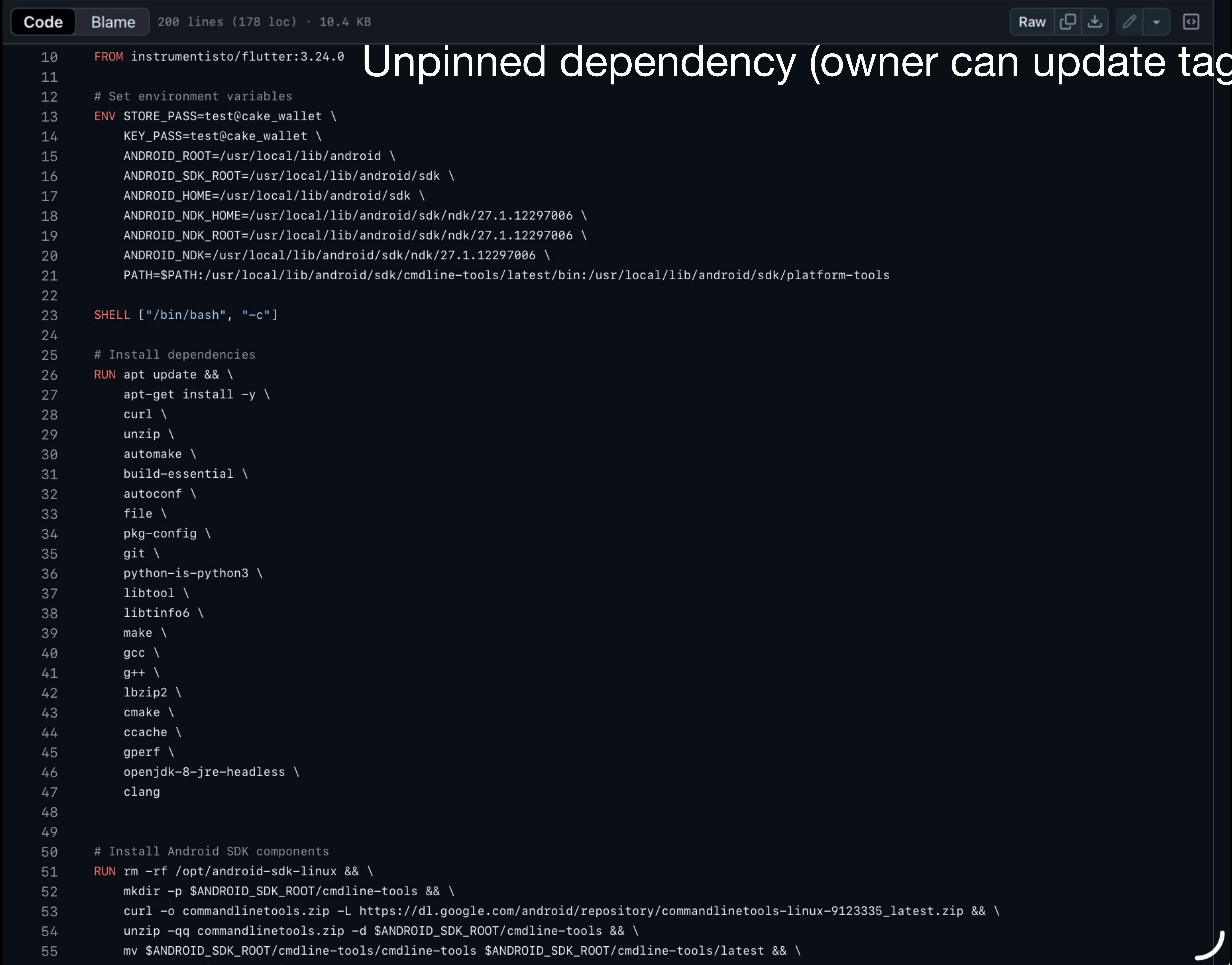
## Let’s simplify it

### It’s just too much work

What software makes it easy to “ship your machine”?

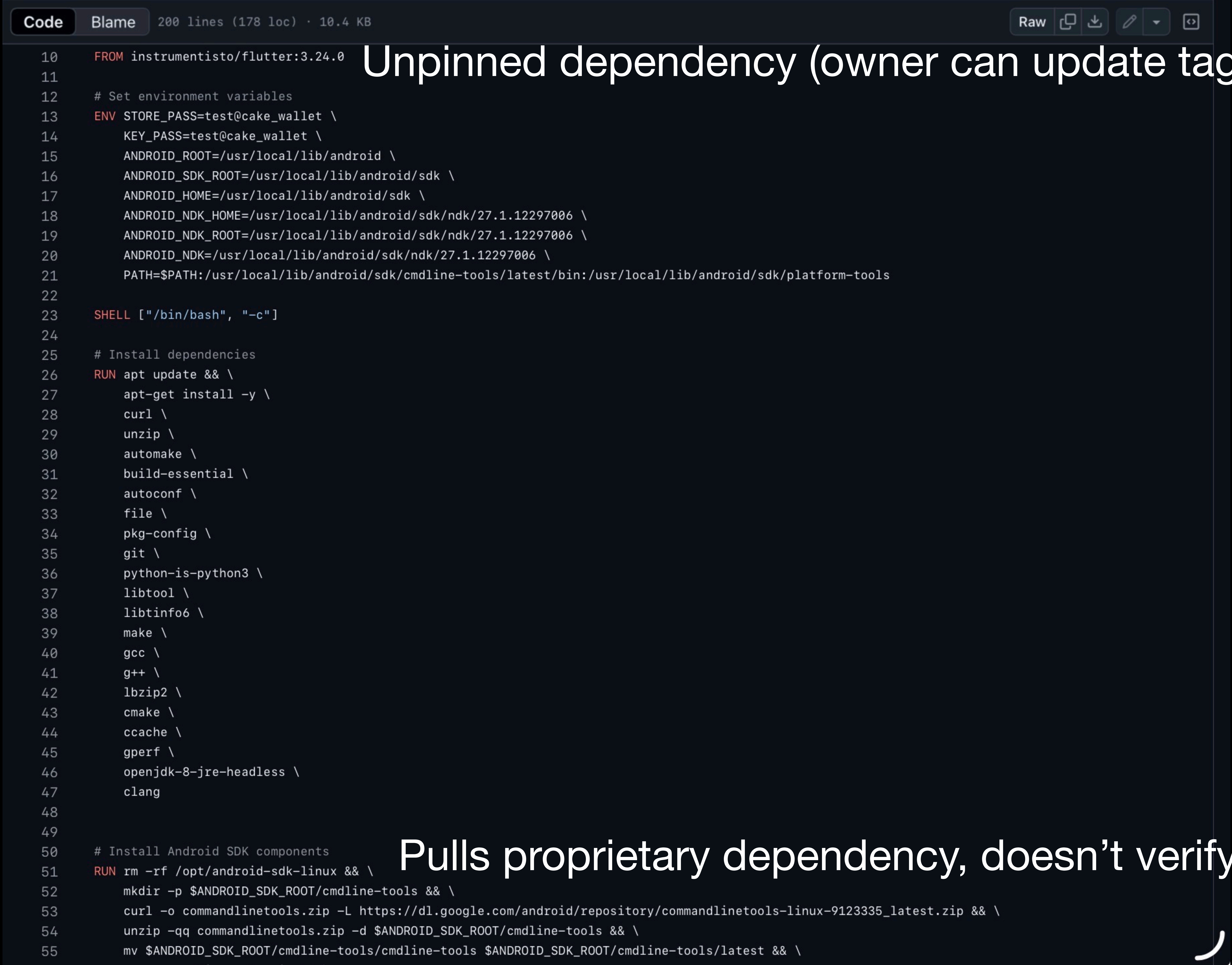


```
FROM instrumentisto/flutter:3.24.0

# Set environment variables
ENV STORE_PASS=test@cake_wallet \
    KEY_PASS=test@cake_wallet \
    ANDROID_ROOT=/usr/local/lib/android \
    ANDROID_SDK_ROOT=/usr/local/lib/android/sdk \
    ANDROID_HOME=/usr/local/lib/android/sdk \
    ANDROID_NDK_HOME=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK_ROOT=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    PATH=$PATH:/usr/local/lib/android/sdk/cmdline-tools/latest/bin:/usr/local/lib/android/sdk/platform-tools

SHELL ["/bin/bash", "-c"]

# Install dependencies
RUN apt update && \
    apt-get install -y \
    curl \
    unzip \
    automake \
    build-essential \
    autoconf \
    file \
    pkg-config \
    git \
    python-is-python3 \
    libtool \
    libtinfo6 \
    make \
    gcc \
    g++ \
    lbzip2 \
    cmake \
    ccache \
    gperf \
    openjdk-8-jre-headless \
    clang


# Install Android SDK components
RUN rm -rf /opt/android-sdk-linux && \
    mkdir -p $ANDROID_SDK_ROOT/cmdline-tools && \
    curl -o commandlinetools.zip -L https://dl.google.com/android/repository/commandlinetools-linux-9123335_latest.zip && \
    unzip -qq commandlinetools.zip -d $ANDROID_SDK_ROOT/cmdline-tools && \
    mv $ANDROID_SDK_ROOT/cmdline-tools/cmdline-tools $ANDROID_SDK_ROOT/cmdline-tools/latest && \
```



```
FROM instrumentisto/flutter:3.24.0

# Set environment variables
ENV STORE_PASS=test@cake_wallet \
    KEY_PASS=test@cake_wallet \
    ANDROID_ROOT=/usr/local/lib/android \
    ANDROID_SDK_ROOT=/usr/local/lib/android/sdk \
    ANDROID_HOME=/usr/local/lib/android/sdk \
    ANDROID_NDK_HOME=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK_ROOT=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    PATH=$PATH:/usr/local/lib/android/sdk/cmdline-tools/latest/bin:/usr/local/lib/android/sdk/platform-tools

SHELL ["/bin/bash", "-c"]

# Install dependencies
RUN apt update && \
    apt-get install -y \
    curl \
    unzip \
    automake \
    build-essential \
    autoconf \
    file \
    pkg-config \
    git \
    python-is-python3 \
    libtool \
    libtinfo6 \
    make \
    gcc \
    g++ \
    lbzip2 \
    cmake \
    ccache \
    gperf \
    openjdk-8-jre-headless \
    clang


# Install Android SDK components
RUN rm -rf /opt/android-sdk-linux && \
    mkdir -p $ANDROID_SDK_ROOT/cmdline-tools && \
    curl -o commandlinetools.zip -L https://dl.google.com/android/repository/commandlinetools-linux-9123335_latest.zip && \
    unzip -qq commandlinetools.zip -d $ANDROID_SDK_ROOT/cmdline-tools && \
    mv $ANDROID_SDK_ROOT/cmdline-tools/cmdline-tools $ANDROID_SDK_ROOT/cmdline-tools/latest && \
```

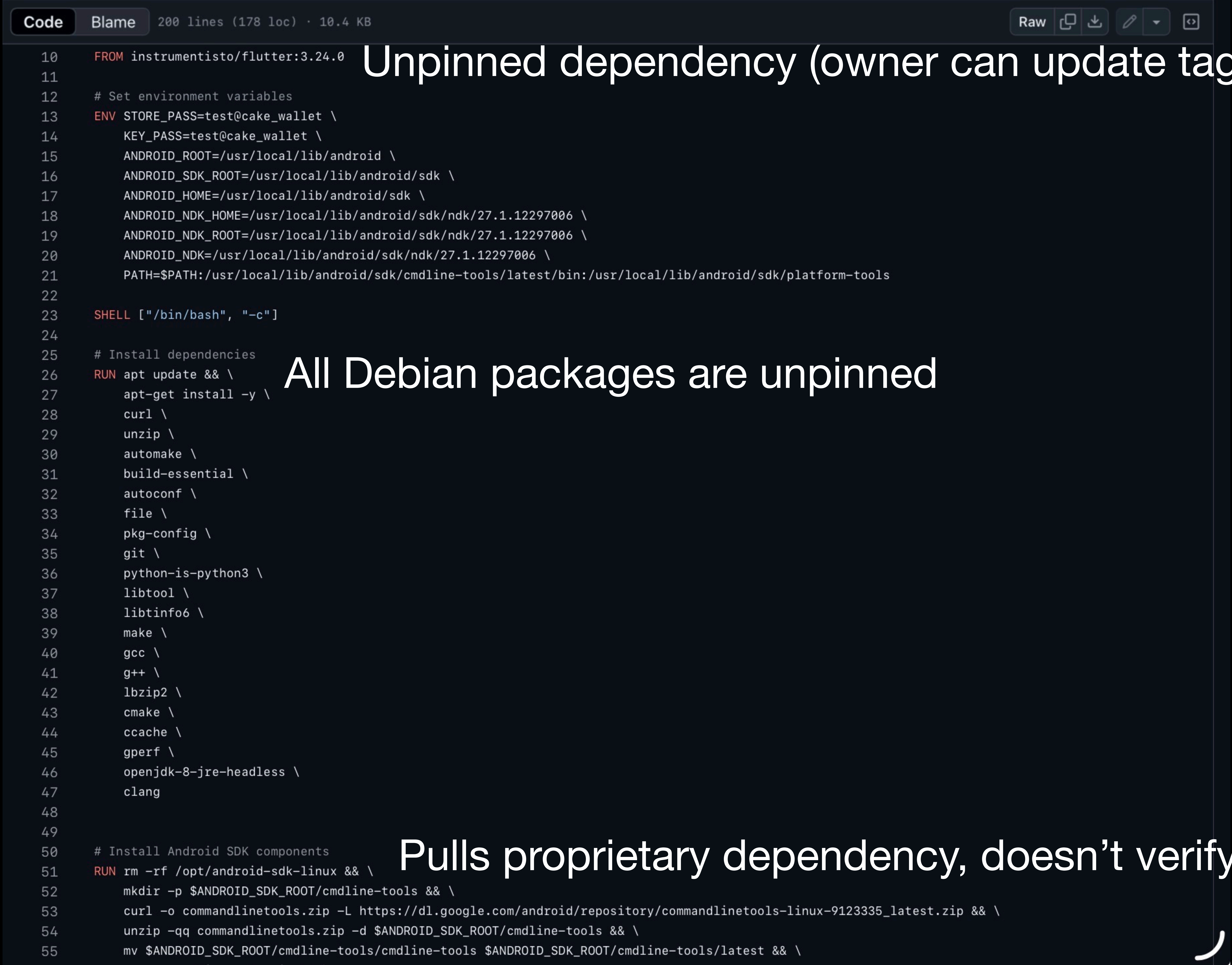
Unpinned dependency (owner can update tag at any time)



```
FROM instrumentisto/flutter:3.24.0

# Set environment variables
ENV STORE_PASS=test@cake_wallet \
    KEY_PASS=test@cake_wallet \
    ANDROID_ROOT=/usr/local/lib/android \
    ANDROID_SDK_ROOT=/usr/local/lib/android/sdk \
    ANDROID_HOME=/usr/local/lib/android/sdk \
    ANDROID_NDK_HOME=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK_ROOT=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    PATH=$PATH:/usr/local/lib/android/sdk/cmdline-tools/latest/bin:/usr/local/lib/android/sdk/platform-tools

SHELL ["/bin/bash", "-c"]

# Install dependencies
RUN apt update && \
    apt-get install -y \
    curl \
    unzip \
    automake \
    build-essential \
    autoconf \
    file \
    pkg-config \
    git \
    python-is-python3 \
    libtool \
    libtinfo6 \
    make \
    gcc \
    g++ \
    lbzip2 \
    cmake \
    ccache \
    gperf \
    openjdk-8-jre-headless \
    clang


# Install Android SDK components
RUN rm -rf /opt/android-sdk-linux && \
    mkdir -p $ANDROID_SDK_ROOT/cmdline-tools && \
    curl -o commandlinetools.zip -L https://dl.google.com/android/repository/commandlinetools-linux-9123335_latest.zip && \
    unzip -qq commandlinetools.zip -d $ANDROID_SDK_ROOT/cmdline-tools && \
    mv $ANDROID_SDK_ROOT/cmdline-tools/cmdline-tools $ANDROID_SDK_ROOT/cmdline-tools/latest && \
```

Unpinned dependency (owner can update tag at any time)

Pulls proprietary dependency, doesn’t verify checksum



```
FROM instrumentisto/flutter:3.24.0
```

Unpinned dependency (owner can update tag at any time)

```
# Set environment variables
ENV STORE_PASS=test@cake_wallet \
    KEY_PASS=test@cake_wallet \
    ANDROID_ROOT=/usr/local/lib/android \
    ANDROID_SDK_ROOT=/usr/local/lib/android/sdk \
    ANDROID_HOME=/usr/local/lib/android/sdk \
    ANDROID_NDK_HOME=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK_ROOT=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    ANDROID_NDK=/usr/local/lib/android/sdk/ndk/27.1.12297006 \
    PATH=$PATH:/usr/local/lib/android/sdk/cmdline-tools/latest/bin:/usr/local/lib/android/sdk/platform-tools

SHELL ["/bin/bash", "-c"]

# Install dependencies
RUN apt update && \
```

All Debian packages are unpinned

```
    apt-get install -y \
    curl \
    unzip \
    automake \
    build-essential \
    autoconf \
    file \
    pkg-config \
    git \
    python-is-python3 \
    libtool \
    libtinfo6 \
    make \
    gcc \
    g++ \
    lbzip2 \
    cmake \
    ccache \
    gperf \
    openjdk-8-jre-headless \
    clang


# Install Android SDK components
```

Pulls proprietary dependency, doesn’t verify checksum

```
RUN rm -rf /opt/android-sdk-linux && \
    mkdir -p $ANDROID_SDK_ROOT/cmdline-tools && \
    curl -o commandlinetools.zip -L https://dl.google.com/android/repository/commandlinetools-linux-9123335_latest.zip && \
    unzip -qq commandlinetools.zip -d $ANDROID_SDK_ROOT/cmdline-tools && \
    mv $ANDROID_SDK_ROOT/cmdline-tools/cmdline-tools $ANDROID_SDK_ROOT/cmdline-tools/latest && \
```

## Usual build process

**Used to prove reproducibility**

- Uses SDK

## Usual build process

**Used to prove reproducibility**

- Uses SDK
- May pull non-source dependencies (common in gradle/flutter packages)

## Usual build process

**Used to prove reproducibility**

- Uses SDK
- May pull non-source dependencies (common in gradle/flutter packages)
- Rust most likely sourced from rustup

## Usual build process

**Used to prove reproducibility**

- Uses SDK
- May pull non-source dependencies (common in gradle/flutter packages)
- Rust most likely sourced from rustup
- Java distribution

## Usual build process

**Used to prove reproducibility**

- Uses SDK
- May pull non-source dependencies (common in gradle/flutter packages)
- Rust most likely sourced from rustup
- Java distribution
- All of that ran in a Docker container

## Usual build process

**Used to prove reproducibility**

- Uses SDK
- May pull non-source dependencies (common in gradle/flutter packages)
- Rust most likely sourced from rustup
- Java distribution
- All of that ran in a Docker container
- Or even worse in a GitHub Action…

## Usual build process

**Used to prove reproducibility**

- Non reproducible untrusted prebuilt
- May pull non-source dependencies (common in gradle/flutter packages)
- Rust most likely sourced from rustup
- Java distribution
- All of that ran in a Docker container
- Or even worse in a GitHub Action…

## Usual build process

**Used to prove reproducibility**

- Non reproducible untrusted prebuilt
- Non reproducible untrusted prebuilt
- Rust most likely sourced from rustup
- Java distribution
- All of that ran in a Docker container
- Or even worse in a GitHub Action…

## Usual build process

**Used to prove reproducibility**

- Non reproducible untrusted prebuilt
- Non reproducible untrusted prebuilt
- Reproducible prebuilt (built using reproducible prebuilt…)
- Java distribution
- All of that ran in a Docker container
- Or even worse in a GitHub Action…

## Usual build process

**Used to prove reproducibility**

- Non reproducible untrusted prebuilt
- Non reproducible untrusted prebuilt
- Reproducible prebuilt (built using reproducible prebuilt…)
- Adoptium distribution is reproducible since Java 21!
- All of that ran in a Docker container
- Or even worse in a GitHub Action…

## Usual build process

**Used to prove reproducibility**

- Non reproducible untrusted prebuilt
- Non reproducible untrusted prebuilt
- Reproducible prebuilt (built using reproducible prebuilt…)
- Adoptium distribution is reproducible since Java 21!
- Non reproducible untrusted prebuilt (that can be replaced by attacker)
- Or even worse in a GitHub Action…

## Usual build process

### Used to prove reproducibility

- Non reproducible untrusted prebuilt
- Non reproducible untrusted prebuilt
- Reproducible prebuilt (built using reproducible prebuilt…)
- Adoptium distribution is reproducible since Java 21!
- Non reproducible untrusted prebuilt (that can be replaced by attacker)
- Non reproducible untrusted prebuilt (>50GiB) that changes quite often

## What’s the fix?

## What’s the fix?

Don’t use these prebuilt binaries.

## Reproducible Builds

## Bootstrappable Builds

## Bootstrappable

It means that the number of prebuilt components should be kept near zero.

Everything should be derived from a source code using a known trusted binary.

## Reproducible

**What does it actually mean?**

Upon running a build script output will be the same given the same source code.

## Let’s compare them

**Using rust**

Reproducible: To build rust 1.N you need working rust 1.N-1 (1.85 can build 1.86), build script by default fetches that release in a binary format.

Bootstrappable: 2010 version of rust was written in OCaml

Rust 1.54 has an alternative compiler (mrustc written in C++)

Let’s build GCC (written in C++) (and GCC 4.7 (written in C) as intermediate step)

## That’s it

## That’s it

### YMMV

- “Just build 9 different versions of 4 different abandoned projects to get it going” for Java

## That’s it

### YMMV

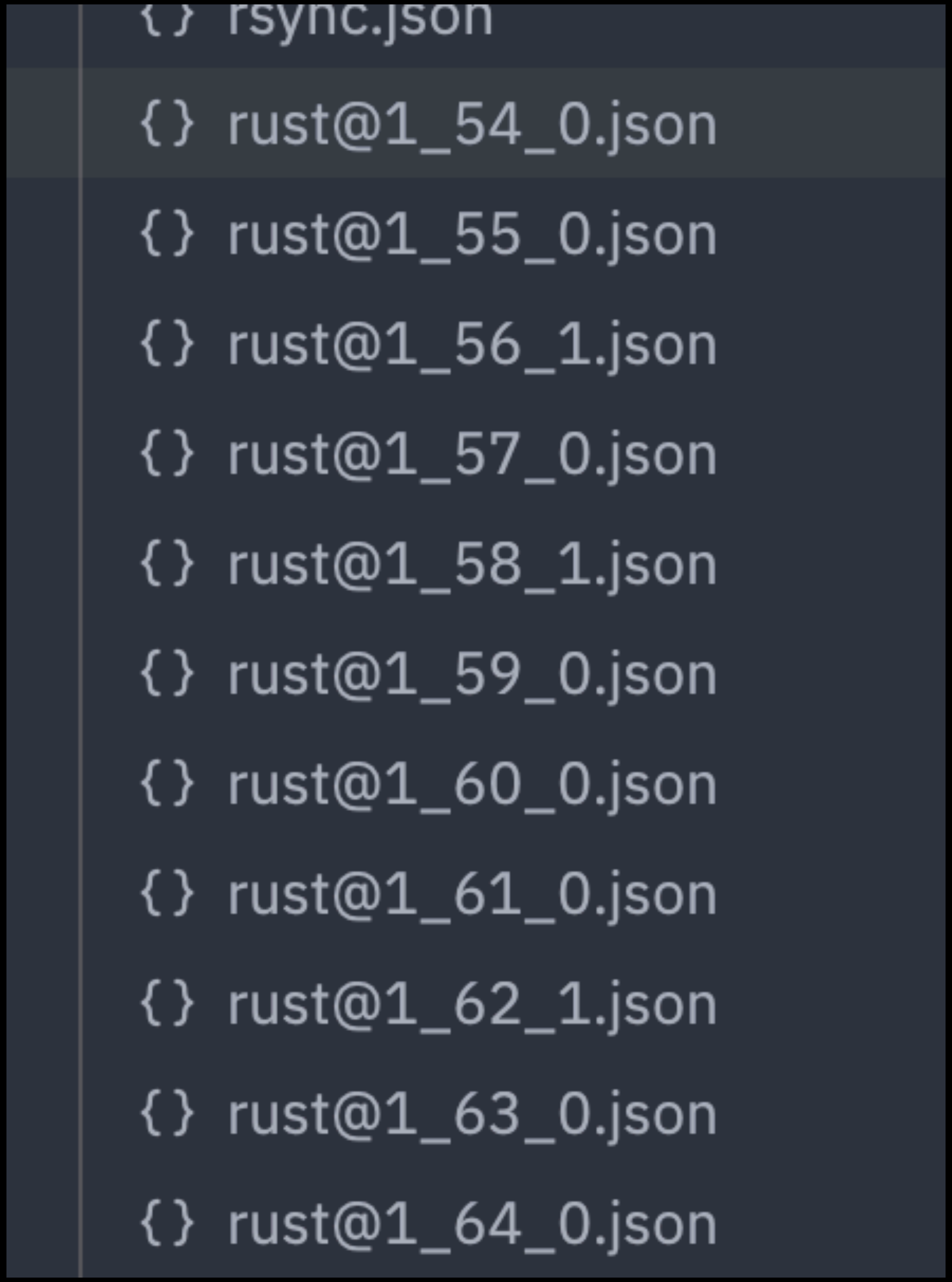
- “Just build 9 different versions of 4 different abandoned projects to get it going” for Java
- “It depends on proprietary software” for Scala

## That’s it
**YMMV**

- “Just build 9 different versions of 4 different abandoned projects to get it going” for Java
- “It depends on proprietary software” for Scala
- “We don’t even know where to start” for Kotlin.

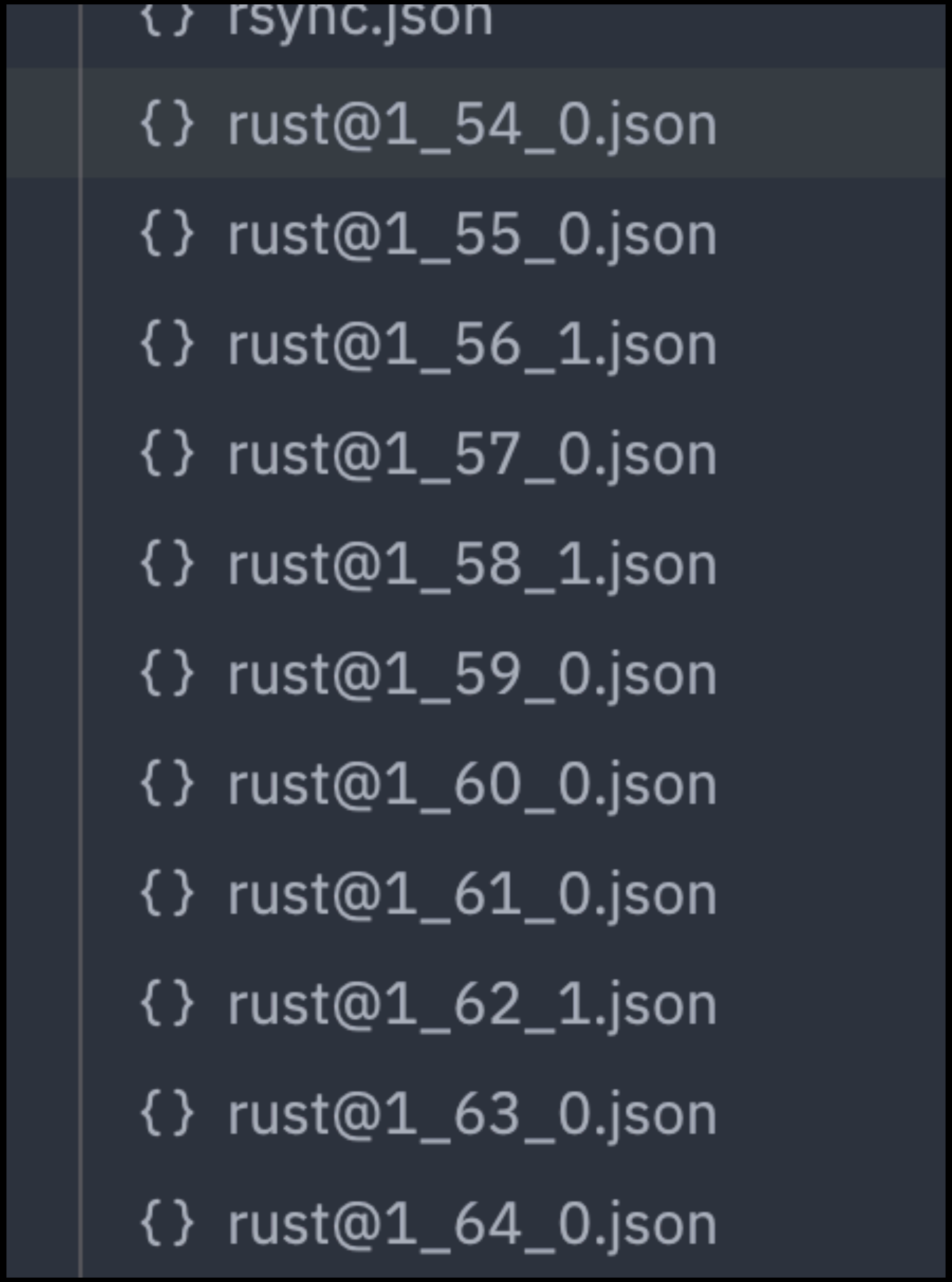
~ bootstrappable.org

## Rust



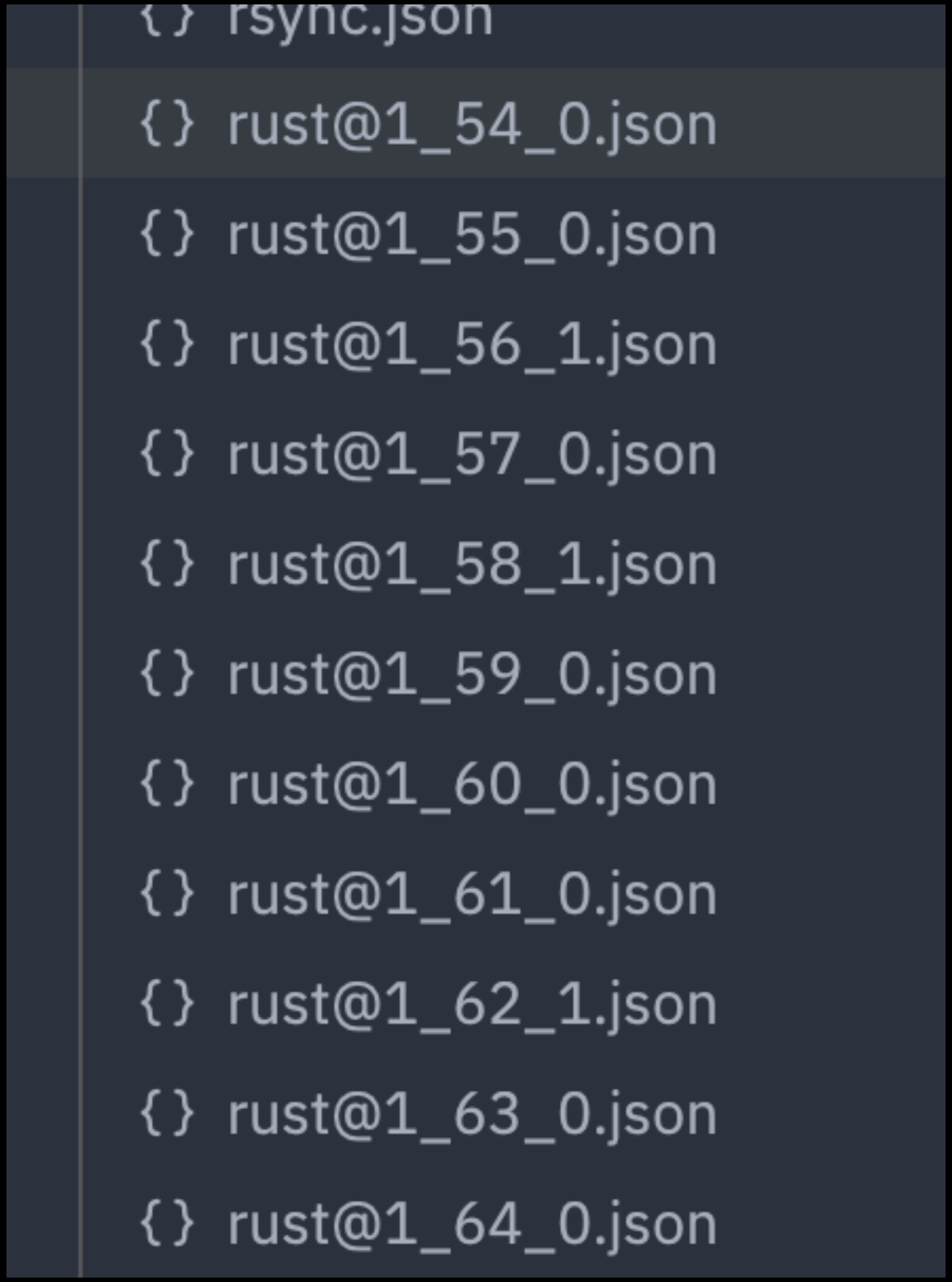
```
{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json
```

# Rust


{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json

- Clear path

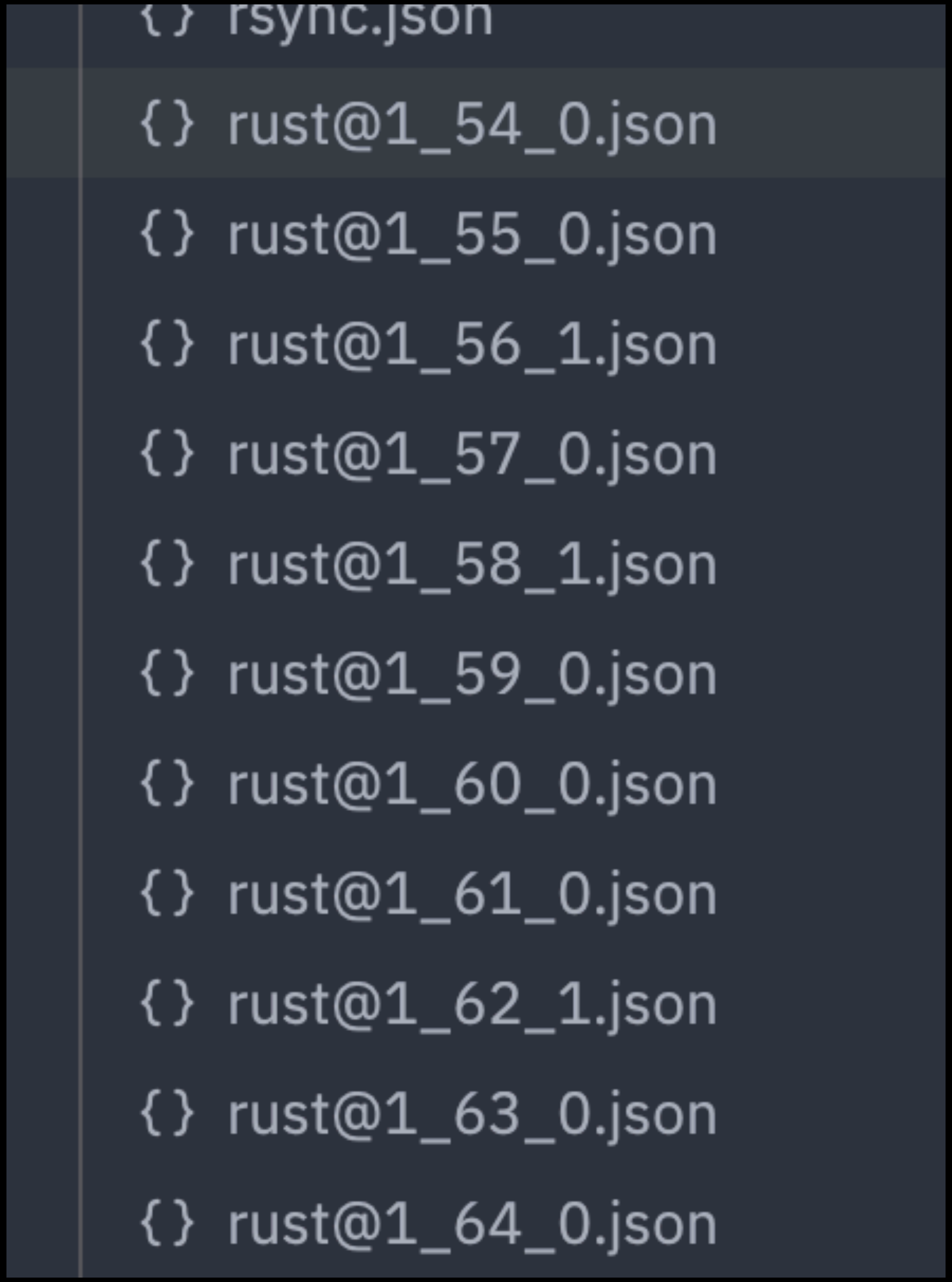
# Rust



{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json

- Clear path
- Well documented breaking changes

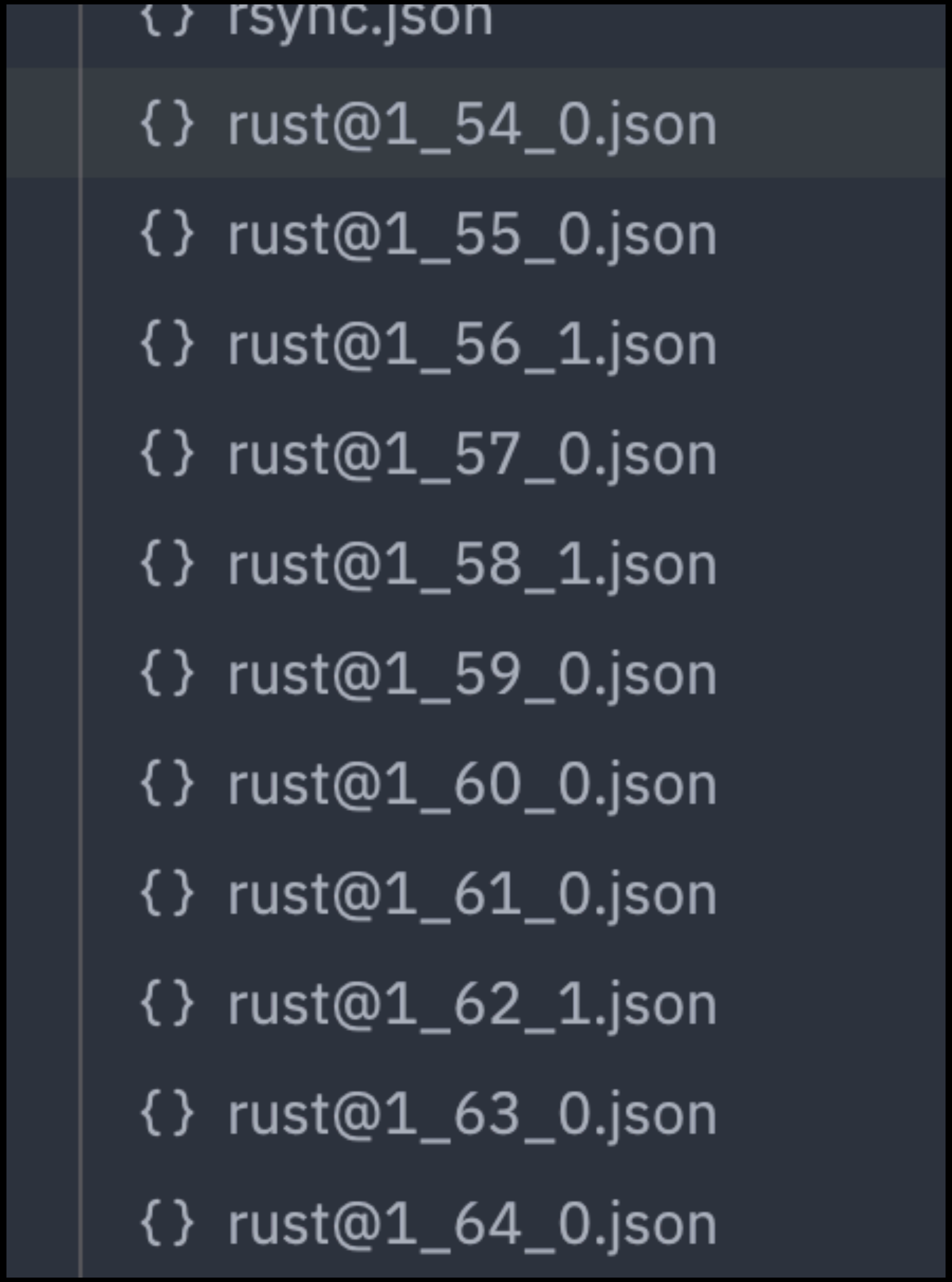
## Rust



```
{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json
```

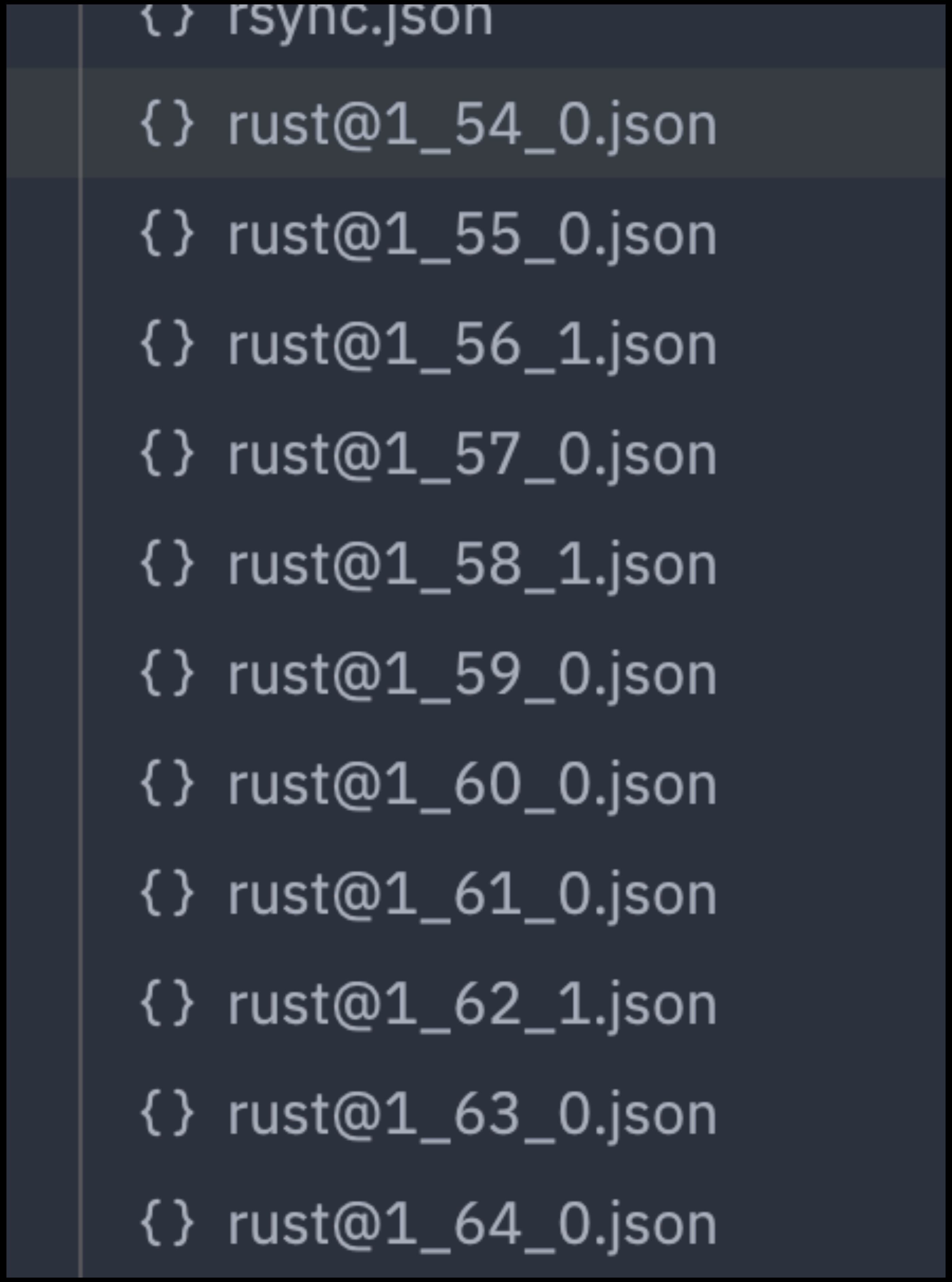
- Clear path
- Well documented breaking changes
- Good software

## Rust


{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json

- Clear path
- Well documented breaking changes
- Good software
- Used for system programming

## Rust

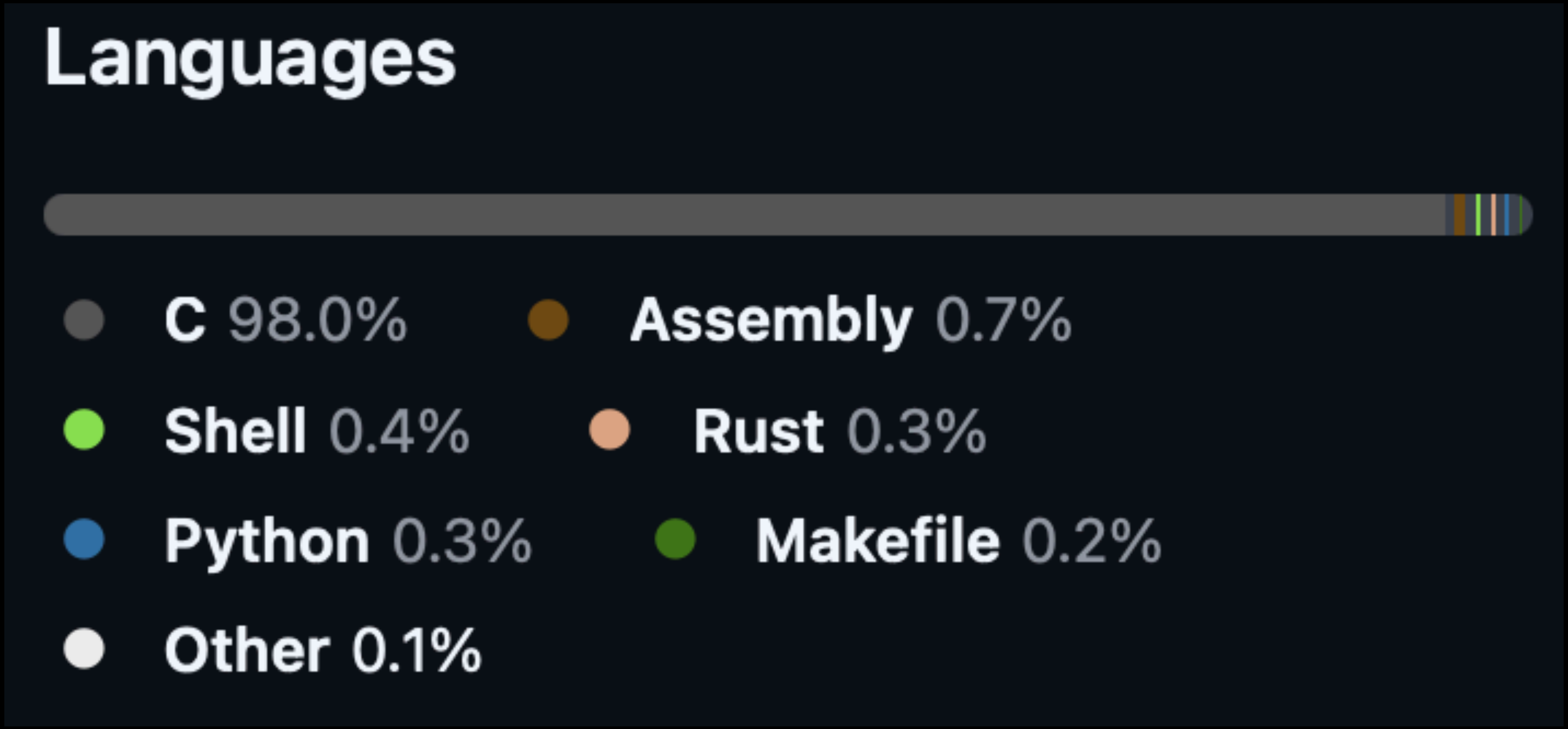

{} rsync.json
{} rust@1_54_0.json
{} rust@1_55_0.json
{} rust@1_56_1.json
{} rust@1_57_0.json
{} rust@1_58_1.json
{} rust@1_59_0.json
{} rust@1_60_0.json
{} rust@1_61_0.json
{} rust@1_62_1.json
{} rust@1_63_0.json
{} rust@1_64_0.json

- Clear path
- Well documented breaking changes
- Good software
- Used for system programming
- Can bootstrap from C++

...

## Where to build rust?

## Where to build rust?


Languages

● C 98.0% ● Assembly 0.7%
● Shell 0.4% ● Rust 0.3%
● Python 0.3% ● Makefile 0.2%
● Other 0.1%

## Where to build rust?

- Previous version of linux can be used

## Where to build rust?

- Previous version of linux can be used
- System configuration that doesn’t use rust

## Where to build rust?

- Previous version of linux can be used
- System configuration that doesn’t use rust
- Old kernel gets unmaintained

## Where to build rust?

- Previous version of linux can be used
- System configuration that doesn’t use rust
- Old kernel gets unmaintained
- Dependencies fail to run on old kernel

## Where to build rust?

- Previous version of linux can be used
- System configuration that doesn’t use rust
- Old kernel gets unmaintained
- Dependencies fail to run on old kernel
- Bootstrap path requires ancient hardware

## Where to build rust?

- Previous version of linux can be used
- System configuration that doesn’t use rust
- Old kernel gets unmaintained
- Dependencies fail to run on old kernel
- Bootstrap path requires ancient hardware
- Ancient hardware may not be able to run latest software

## Cat and mouse

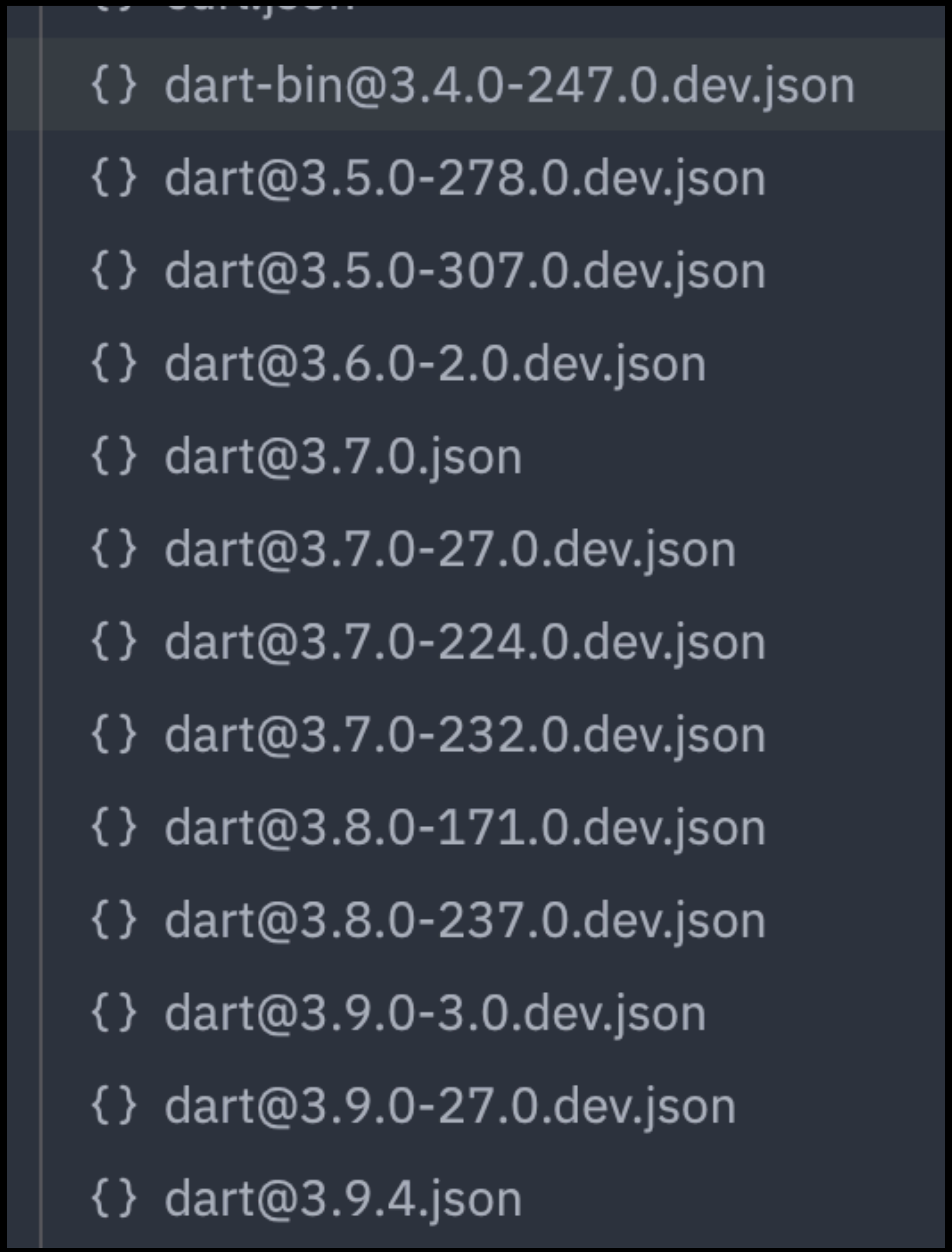
## Cat and mouse

- Older software gets harder to run on newer hardware

## Cat and mouse

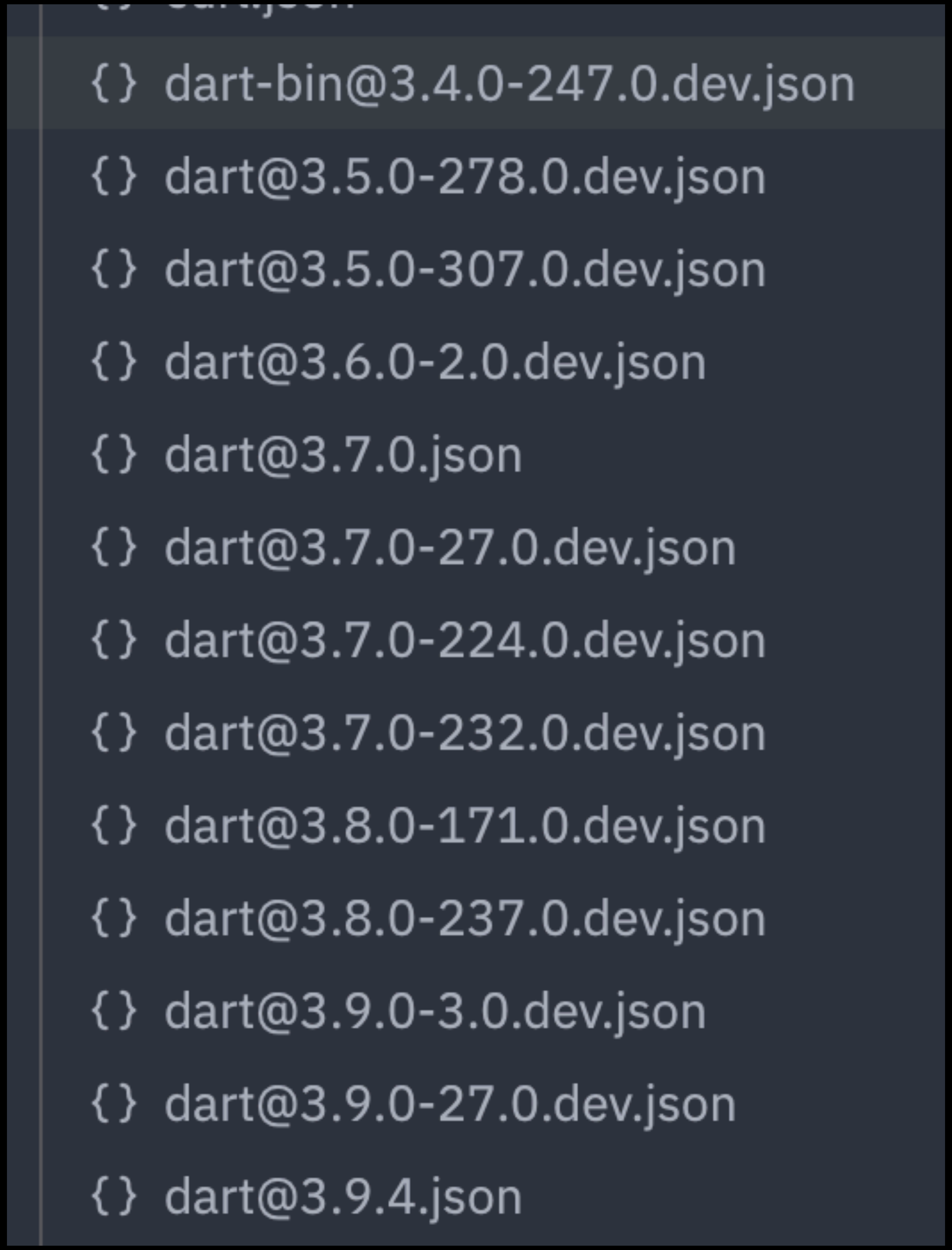
- Older software gets harder to run on newer hardware
- New hardware requires more and more old software to run

## Dart



```
{} dart-bin@3.4.0-247.0.dev.json
{} dart@3.5.0-278.0.dev.json
{} dart@3.5.0-307.0.dev.json
{} dart@3.6.0-2.0.dev.json
{} dart@3.7.0.json
{} dart@3.7.0-27.0.dev.json
{} dart@3.7.0-224.0.dev.json
{} dart@3.7.0-232.0.dev.json
{} dart@3.8.0-171.0.dev.json
{} dart@3.8.0-237.0.dev.json
{} dart@3.9.0-3.0.dev.json
{} dart@3.9.0-27.0.dev.json
{} dart@3.9.4.json
```

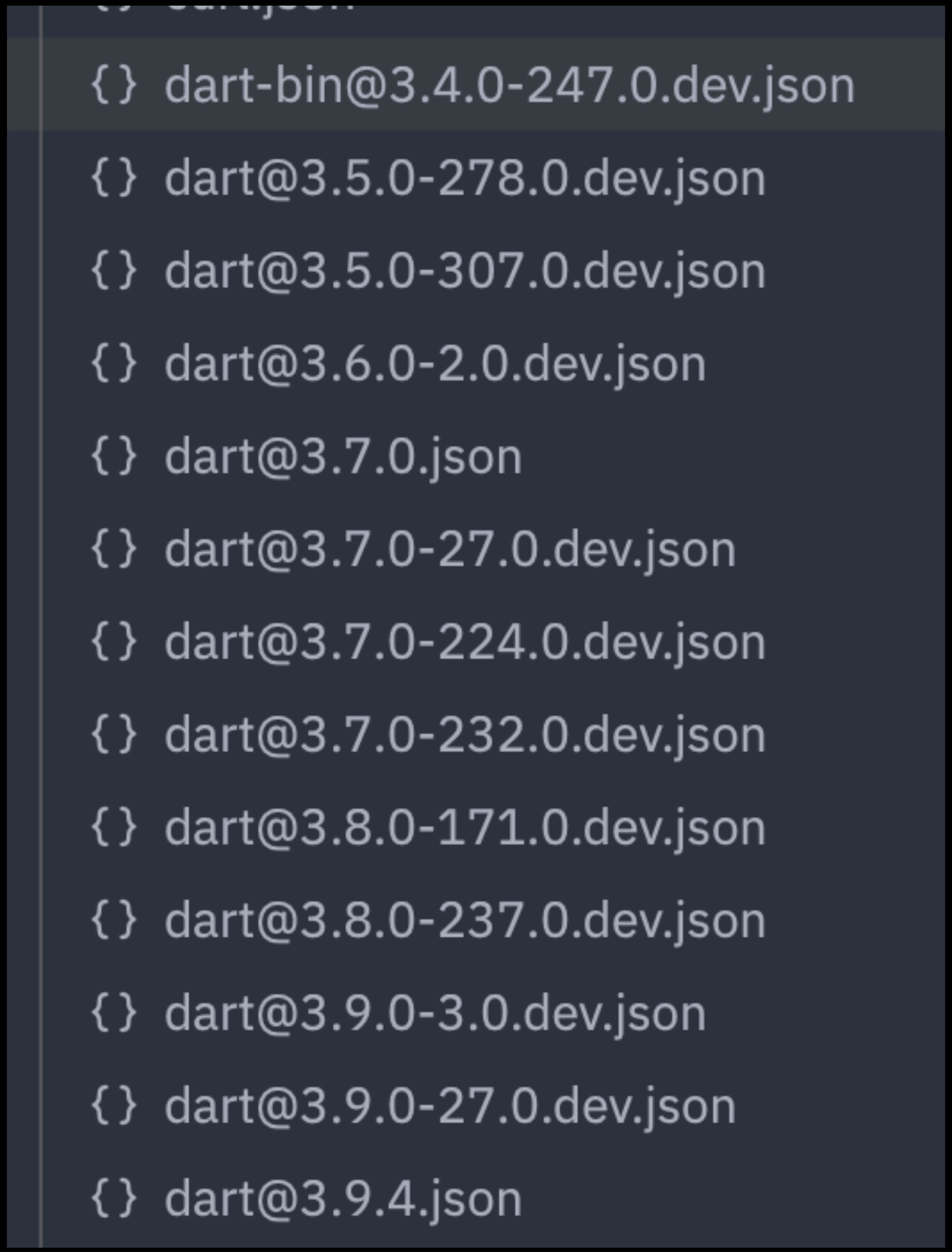
## Dart



```
{} dart-bin@3.4.0-247.0.dev.json
{} dart@3.5.0-278.0.dev.json
{} dart@3.5.0-307.0.dev.json
{} dart@3.6.0-2.0.dev.json
{} dart@3.7.0.json
{} dart@3.7.0-27.0.dev.json
{} dart@3.7.0-224.0.dev.json
{} dart@3.7.0-232.0.dev.json
{} dart@3.8.0-171.0.dev.json
{} dart@3.8.0-237.0.dev.json
{} dart@3.9.0-3.0.dev.json
{} dart@3.9.0-27.0.dev.json
{} dart@3.9.4.json
```

- Moved documentation several times (destroying historical build instructions)

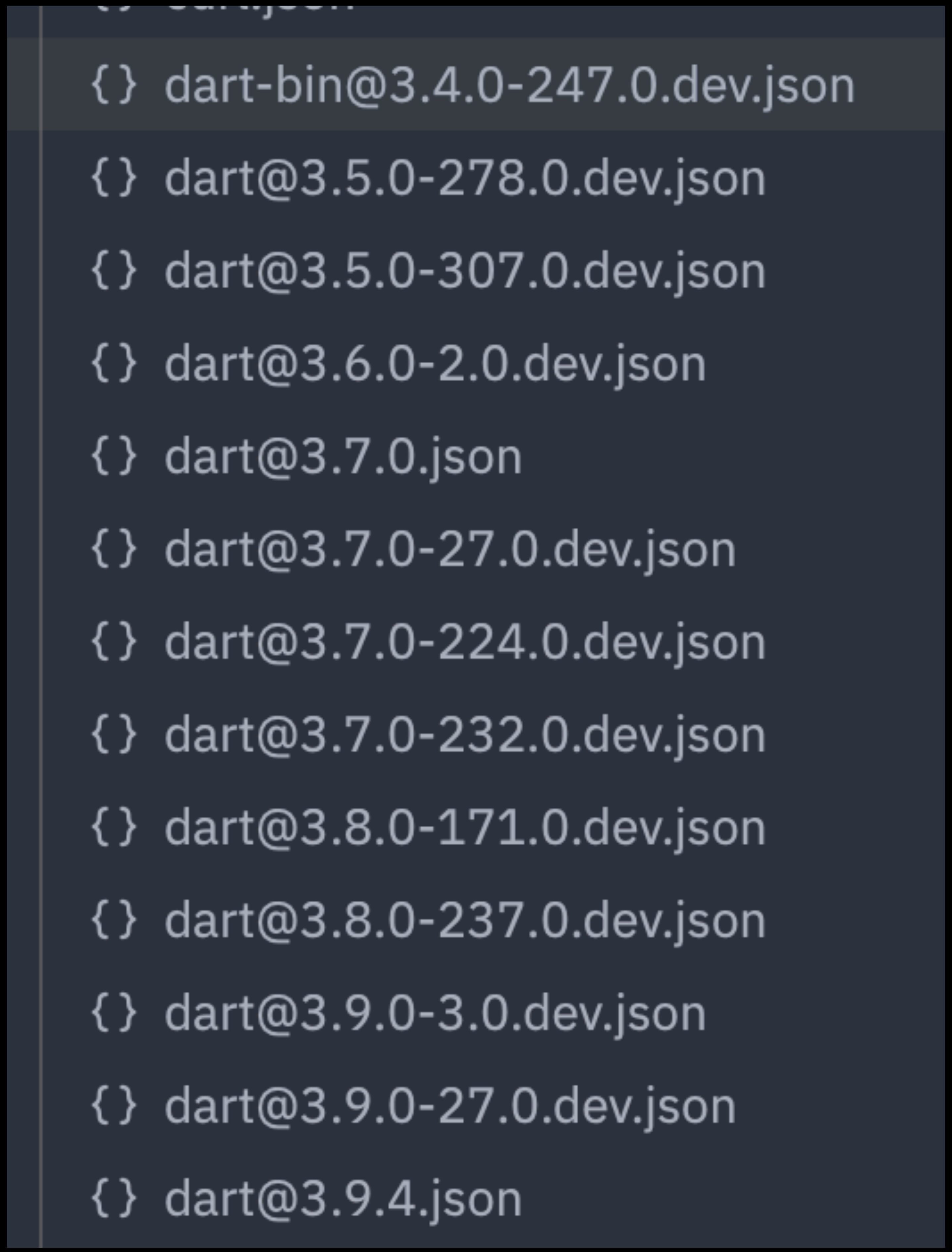
## Dart



```
{} dart-bin@3.4.0-247.0.dev.json
{} dart@3.5.0-278.0.dev.json
{} dart@3.5.0-307.0.dev.json
{} dart@3.6.0-2.0.dev.json
{} dart@3.7.0.json
{} dart@3.7.0-27.0.dev.json
{} dart@3.7.0-224.0.dev.json
{} dart@3.7.0-232.0.dev.json
{} dart@3.8.0-171.0.dev.json
{} dart@3.8.0-237.0.dev.json
{} dart@3.9.0-3.0.dev.json
{} dart@3.9.0-27.0.dev.json
{} dart@3.9.4.json
```

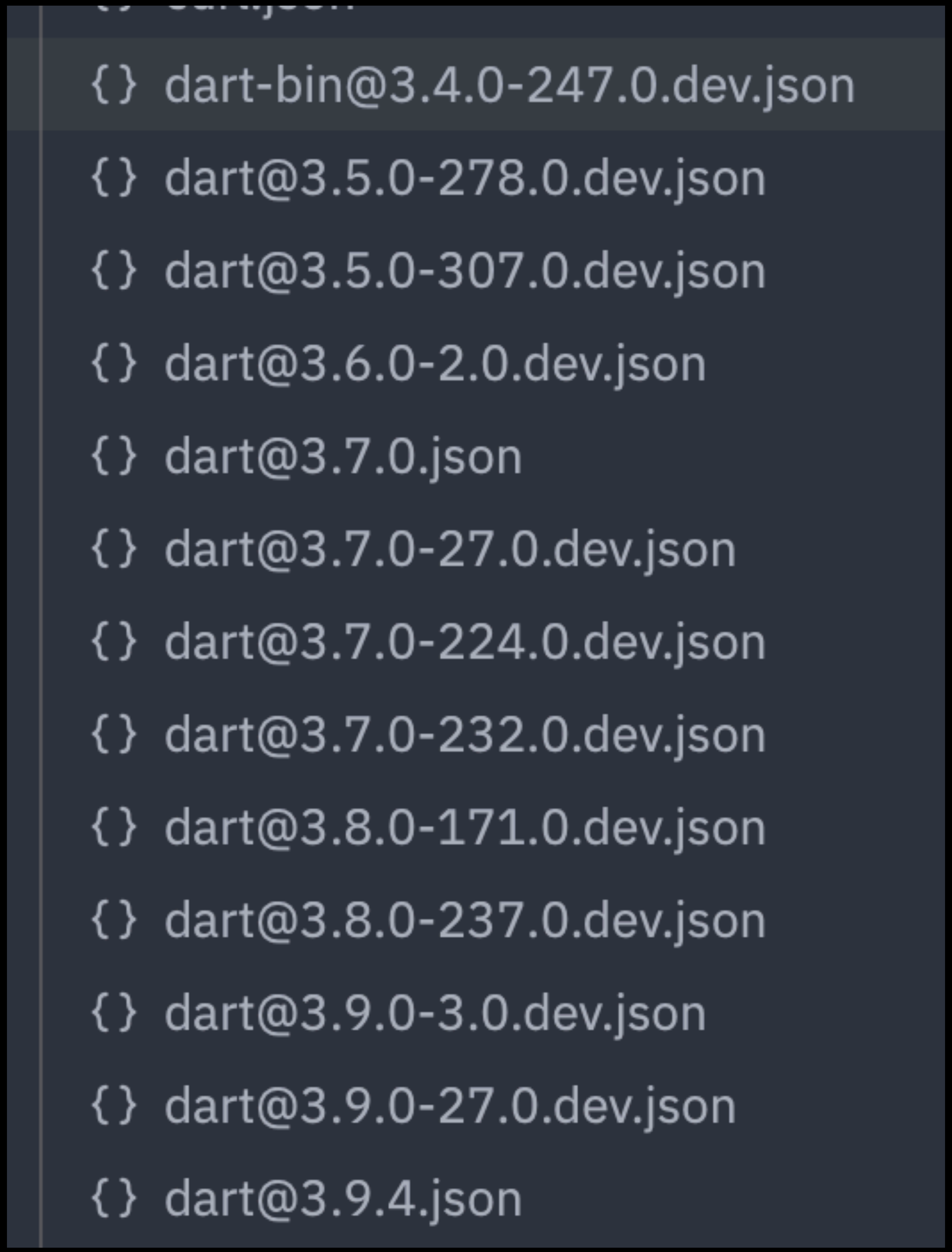
- Moved documentation several times (destroying historical build instructions)
- Moved source code location of dependencies

## Dart


{} dart-bin@3.4.0-247.0.dev.json
{} dart@3.5.0-278.0.dev.json
{} dart@3.5.0-307.0.dev.json
{} dart@3.6.0-2.0.dev.json
{} dart@3.7.0.json
{} dart@3.7.0-27.0.dev.json
{} dart@3.7.0-224.0.dev.json
{} dart@3.7.0-232.0.dev.json
{} dart@3.8.0-171.0.dev.json
{} dart@3.8.0-237.0.dev.json
{} dart@3.9.0-3.0.dev.json
{} dart@3.9.0-27.0.dev.json
{} dart@3.9.4.json

- Moved documentation several times (destroying historical build instructions)
- Moved source code location of dependencies
- Depends on Fuchsia, Chromium and several other CI actions to get prebuilt binaries

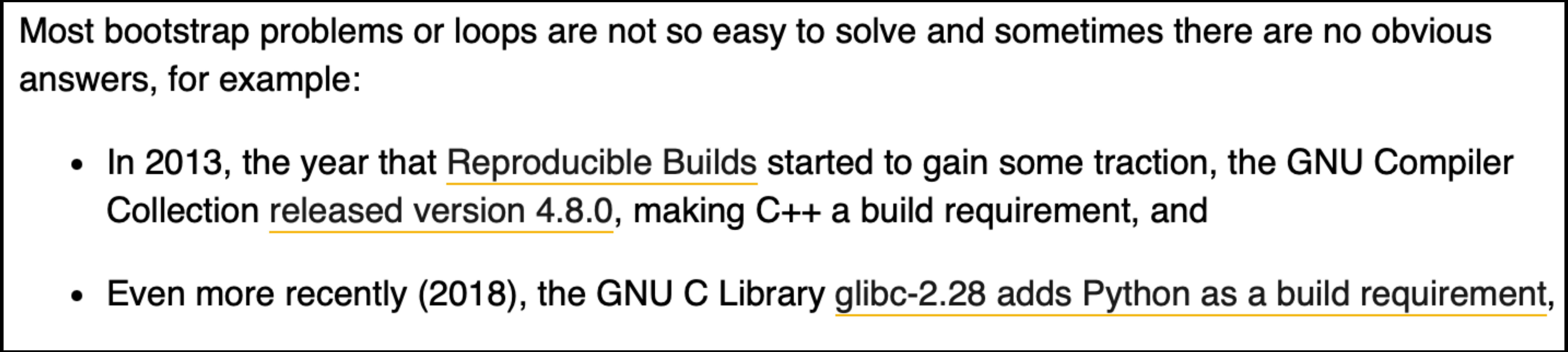
## Dart



```
{} dart-bin@3.4.0-247.0.dev.json
{} dart@3.5.0-278.0.dev.json
{} dart@3.5.0-307.0.dev.json
{} dart@3.6.0-2.0.dev.json
{} dart@3.7.0.json
{} dart@3.7.0-27.0.dev.json
{} dart@3.7.0-224.0.dev.json
{} dart@3.7.0-232.0.dev.json
{} dart@3.8.0-171.0.dev.json
{} dart@3.8.0-237.0.dev.json
{} dart@3.9.0-3.0.dev.json
{} dart@3.9.0-27.0.dev.json
{} dart@3.9.4.json
```

- Moved documentation several times (destroying historical build instructions)
- Moved source code location of dependencies
- Depends on Fuchsia, Chromium and several other CI actions to get prebuilt binaries
- Depends on dev snapshots to build the next compiler

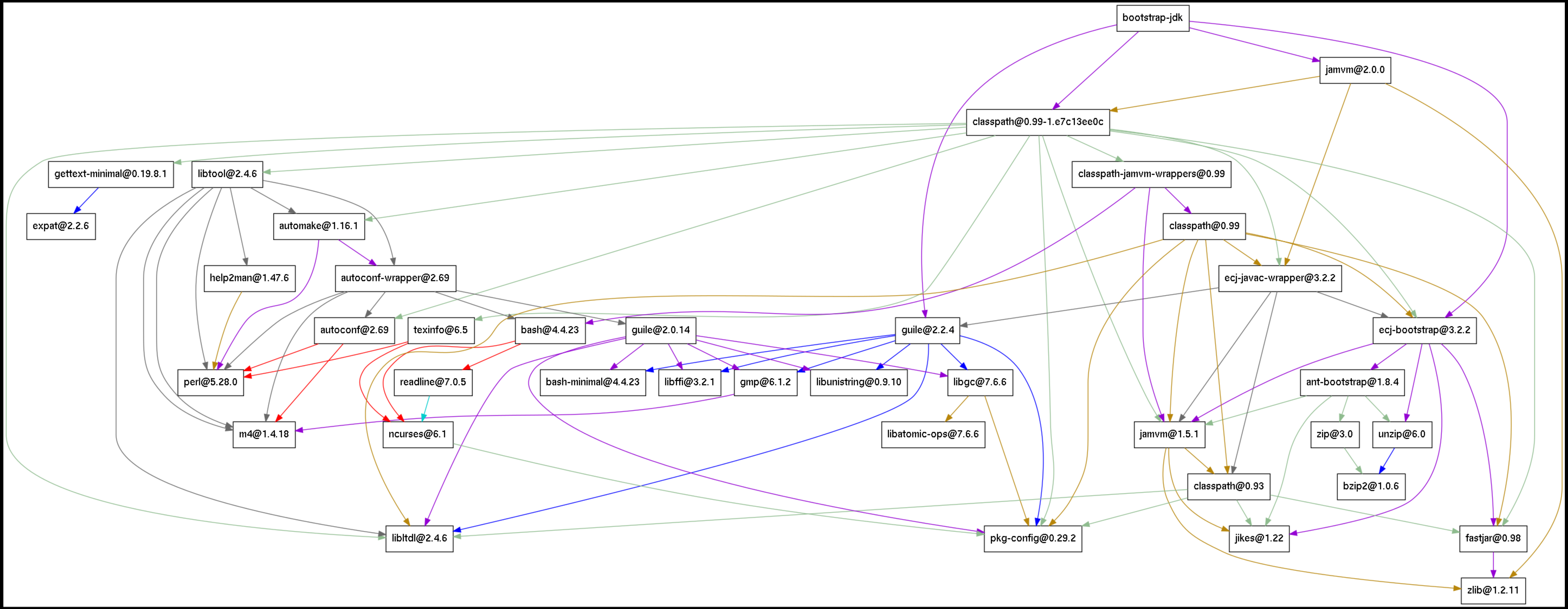
# GCC and GNU C Library



Most bootstrap problems or loops are not so easy to solve and sometimes there are no obvious answers, for example:

- In 2013, the year that Reproducible Builds started to gain some traction, the GNU Compiler Collection released version 4.8.0, making C++ a build requirement, and
- Even more recently (2018), the GNU C Library glibc-2.28 adds Python as a build requirement,

# Java


bootstrap-jdk
jamvm@2.0.0
classpath@0.99-1.e7c13ee0c
gettext-minimal@0.19.8.1
libtool@2.4.6
classpath-jamvm-wrappers@0.99
expat@2.2.6
automake@1.16.1
classpath@0.99
help2man@1.47.6
autoconf-wrapper@2.69
ecj-javac-wrapper@3.2.2
autoconf@2.69
texinfo@6.5
bash@4.4.23
guile@2.0.14
guile@2.2.4
ecj-bootstrap@3.2.2
perl@5.28.0
readline@7.0.5
bash-minimal@4.4.23
libffi@3.2.1
gmp@6.1.2
libunistring@0.9.10
libgc@7.6.6
ant-bootstrap@1.8.4
m4@1.4.18
ncurses@6.1
libatomic-ops@7.6.6
jamvm@1.5.1
zip@3.0
unzip@6.0
classpath@0.93
bzip2@1.0.6
libltdl@2.4.6
pkg-config@0.29.2
jikes@1.22
fastjar@0.98
zlib@1.2.11

## Guix

# Guix

# Guix

## Auditable bootstrap binary

## Guix

**Auditable bootstrap binary**

- Previous example of reproducible build trusted around 100 GiB of binaries

## Guix

### Auditable bootstrap binary

- Previous example of reproducible build trusted around 100 GiB of binaries
- Guix trusts 357 Bytes of binary code

## Guix

**Auditable bootstrap binary**

- Previous example of reproducible build trusted around 100 GiB of binaries
- Guix trusts 357 Bytes of binary code
- Maintains many build paths for popular software

## Guix

### Auditable bootstrap binary

- Previous example of reproducible build trusted around 100 GiB of binaries
- Guix trusts 357 Bytes of binary code
- Maintains many build paths for popular software
- Used by Monero

## Guix

**Auditable bootstrap binary**

- Previous example of reproducible build trusted around 100 GiB of binaries
- Guix trusts 357 Bytes of binary code
- Maintains many build paths for popular software
- Used by Monero
- Just use Guix to get the environment and build the app there?

## Guix

- Must use Linux

## Guix

- Must use Linux (Flutter cannot cross compile)

## F-Droid

## F-Droid

- Doesn’t solve the problem for all other platforms

## F-Droid

- Doesn’t solve the problem for all other platforms
- Replaces some proprietary components of android SDK with open source reimplementations

## F-Droid

- Doesn’t solve the problem for all other platforms
- Replaces some proprietary components of android SDK with open source reimplementations
- Still pulls NDK and many other tools/dependencies

## Why bother?

## Why bother?

**Bootstrappable builds = recursive reproducible builds**

## Why bother?

**Bootstrappable builds = recursive reproducible builds**

It’s not just a “party trick” to build everything from source.

It’s a defense

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

Proof of concept compiler virus

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

Proof of concept compiler virus that could

- detect when it is building /usr/bin/login and inject backdoor

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

Proof of concept compiler virus that could

- detect when it is building /usr/bin/login and inject backdoor
- detect when it is compiling itself and inject the virus.

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

*In demonstrating the possibility of this kind of attack, I picked on the C compiler. I could have picked on any program-handling program such as an assembler, a loader, or even hardware microcode. As the level of program gets lower, these bugs will be harder and harder to detect. A well installed microcode bug*[^1] *will be almost impossible to detect.*

[^1]: As in a listening device, not an error

## *Reflections On Trusting Trust*

**1983, Ken Thompson (Known for C, UNIX, Go)**

All compilers, assemblers, interpreters, linkers and even microcode are vulnerable to this kind of attack.

## *Reflections On Trusting Trust*
**1983, Ken Thompson (Known for C, UNIX, Go)**

All compilers, assemblers, interpreters, linkers and even microcode are vulnerable to this kind of attack.

And you can access PoC viruses like this online, or read a really good story of Mick Stute on Quora as an answer to question “What is a Coder’s worst Nightmare”

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

- Concept still exist, together with PoC implementations

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

- Concept still exist, together with PoC implementations
- Less people compile from source

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

- Concept still exist, together with PoC implementations
- Less people compile from source
- Easier to target programs

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

- Concept still exist, together with PoC implementations
- Less people compile from source
- Easier to target programs
- And libraries

## *Reflections On Trusting Trust*

**Does it still apply over 40 years later?**

- Concept still exist, together with PoC implementations
- Less people compile from source
- Easier to target programs
- And libraries
- CVE-2024-3094 xz-utils backdoor

## CVE-2024-3094

### Xz-utils backdoor incident

- Maintainer spent 2 years doing honest work

## CVE-2024-3094

**Xz-utils backdoor incident**

- Maintainer spent 2 years doing honest work
- Added some broken test cases (binary blobs)

## CVE-2024-3094

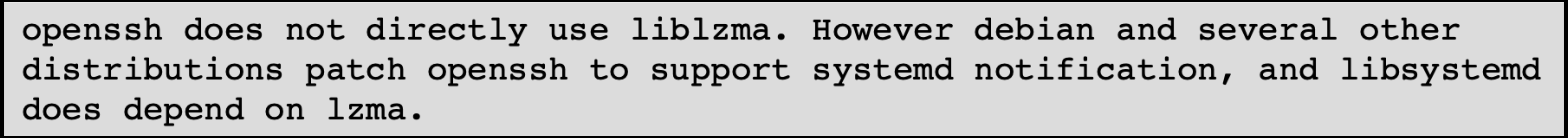
### Xz-utils backdoor incident

- Maintainer spent 2 years doing honest work
- Added some broken test cases (binary blobs)
- Tarball release contained a line to extract and run the payload

## CVE-2024-3094

### Xz-utils backdoor incident

- Maintainer spent 2 years doing honest work
- Added some broken test cases (binary blobs)
- Tarball release contained a line to extract and run the payload
- Virus hooked into patched openssh versions (via systemd)



```
openssh does not directly use liblzma. However debian and several other
distributions patch openssh to support systemd notification, and libsystemd
does depend on lzma.
```

## CVE-2024-3094

**Xz-utils backdoor incident**

- Maintainer spent 2 years doing honest work
- Added some broken test cases (binary blobs)
- Tarball release contained a line to extract and run the payload
- Virus hooked into patched openssh versions (via systemd)
- It got detected by accident

## CVE-2024-3094

### Xz-utils backdoor incident

One compromised library (even build from source) can compromise entire system

## Reproducibility at Cake

### Guix

Not applicable, only support Linux, doesn’t have all packages

## Reproducibility at Cake

### Nix/NixOS

Supports both Linux and Mac.

## Reproducibility at Cake

### Nix/NixOS

Supports both Linux and Mac.

Is kind of bootstrappable.

## Reproducibility at Cake

**Nix/NixOS**

Supports both Linux and Mac.

Is kind of bootstrappable.

It doesn’t have a strong policy on binaries in source, it’s fairly common to find packages shipping prebuilts (even non-free).

## Reproducibility at Cake

### Nix/NixOS

Supports both Linux and Mac.

Is kind of bootstrappable.

It doesn’t have a strong policy on binaries in source, it’s fairly common to find packages shipping prebuilts (even non-free).

Requires a system-wide install.

## Reproducibility at Cake

### contrib/depends

# Reproducibility at Cake

**contrib/depends**

- Already used in monero_c

## Reproducibility at Cake

**contrib/depends**

- Already used in monero_c
- Modular

## Reproducibility at Cake

**contrib/depends**

- Already used in monero_c
- Modular
- Easy to understand/extend

## Reproducibility at Cake

**contrib/depends**

- Already used in monero_c
- Modular
- Easy to understand/extend
- Doesn’t scale well

## Reproducibility at Cake

### contrib/depends reimplementation

## Reproducibility at Cake
### contrib/depends reimplementation

- All the benefits of contrib/depends

## Reproducibility at Cake

**contrib/depends reimplementation**

- All the benefits of contrib/depends
- One-level dependencies and better use of cache

## Reproducibility at Cake

**contrib/depends reimplementation**

- All the benefits of contrib/depends
- One-level dependencies and better use of cache
- Cloud cache (optionally self-hosted)

## Reproducibility at Cake

### contrib/depends reimplementation

- All the benefits of contrib/depends
- One-level dependencies and better use of cache
- Cloud cache (optionally self-hosted)
- Flat learning curve

## Reproducibility at Cake

**contrib/depends reimplementation**

- All the benefits of contrib/depends
- One-level dependencies and better use of cache
- Cloud cache (optionally self-hosted)
- Flat learning curve
- Compatible with everything that can be built from command line

## Reproducibility at Cake

### contrib/depends reimplementation

- Allows to get rid of binary dependencies slowly.

## Reproducibility at Cake

**contrib/depends reimplementation**

- Allows to get rid of binary dependencies slowly.
- Everything is a cross-compilation

## Reproducibility at Cake

**contrib/depends reimplementation**

- Allows to get rid of binary dependencies slowly.
- Everything is a cross-compilation
- Use QEMU to build alien software

# Reproducibility at Cake

**contrib/depends reimplementation**

- Allows to get rid of binary dependencies slowly.
- Everything is a cross-compilation
- Use QEMU to build alien software
- Works on Linux and macOS

## Reproducibility at Cake

### contrib/depends reimplementation

- Allows to get rid of binary dependencies slowly.
- Everything is a cross-compilation
- Use QEMU to build alien software
- Works on Linux and macOS
- Assumes only working C and C++ compiler

## Reproducibility at Cake

**contrib/depends reimplementation**

- Allows to get rid of binary dependencies slowly.
- Everything is a cross-compilation
- Use QEMU to build alien software
- Works on Linux and macOS
- Assumes only working C and C++ compiler
- Already used in Cake Wallet (libtorch.{so,dylib,dll})

## Reproducibility at Cake

**Small steps to achieve the goal.**

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path
- Improve or maintain developer experience

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path
- Improve or maintain developer experience
- Replace all native dependencies with reproducible versions

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path
- Improve or maintain developer experience
- Replace all native dependencies with reproducible versions
- Make it in a way in which it will benefit largest amount of projects

## Reproducibility at Cake

**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path
- Improve or maintain developer experience
- Replace all native dependencies with reproducible versions
- Make it in a way in which it will benefit largest amount of projects
- We are getting there. It’s just much more work.

## Reproducibility at Cake

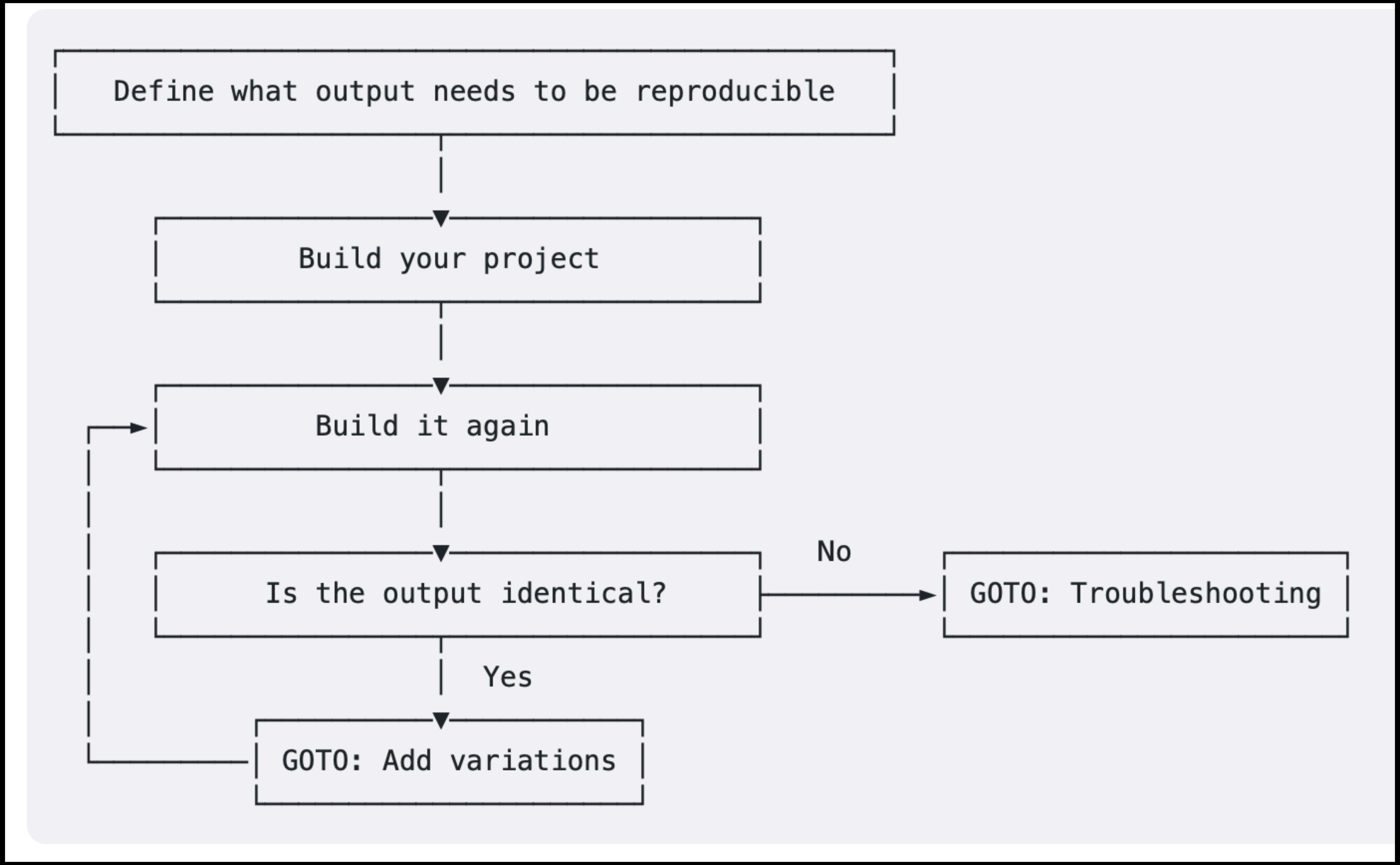
**Small steps to achieve the goal.**

- It’s not .secrets.g.dart
- It’s not us being evil
- Reproducibility with bootstrappable path
- Improve or maintain developer experience
- Replace all native dependencies with reproducible versions
- Make it in a way in which it will benefit largest amount of projects
- We are getting there. It’s just much more work.
- “We are not doing it because it’s easy, we are doing it because we thought it will be easy”

## How can you help?

## How can you help?

**reproducible-builds.org**



## How can you help?

- Approach maintainers

## How can you help?

- Approach maintainers
- Understand the project structure, requirements and goals

## How can you help?

- Approach maintainers
- Understand the project structure, requirements and goals
- Open an issue with questions / ideas

## How can you help?

- Approach maintainers
- Understand the project structure, requirements and goals
- Open an issue with questions / ideas
- Check out Nix’s issue tracker for “reproducibility” tag

## How can you help?

- Approach maintainers
- Understand the project structure, requirements and goals
- Open an issue with questions / ideas
- Check out Nix’s issue tracker for “reproducibility” tag
- Package your favorite software for Guix/NixOS/Debian

## How can you help Cake Wallet?

- Continue dart bootstrap journey

## How can you help Cake Wallet?

- Continue dart bootstrap journey
- Figure out bootstrap path for Kotlin

## How can you help Cake Wallet?

- Continue dart bootstrap journey
- Figure out bootstrap path for Kotlin
- Compile Flutter apps through wine/darling

## How can you help Cake Wallet?

- Continue dart bootstrap journey
- Figure out bootstrap path for Kotlin
- Compile Flutter apps through wine/darling
- Check out Guix blog for pain points

## How can you help Cake Wallet?

- Continue dart bootstrap journey
- Figure out bootstrap path for Kotlin
- Compile Flutter apps through wine/darling
- Check out Guix blog for pain points
- Follow updates that we release

# On reproducibility

Czarek / Developer @ Cake Wallet / Feb 13th 2026