Engineer

Codebase

all the source code of one app, kept in one repository

What it is

A codebase, sometimes written as two words, code base, is the whole body of source that a team edits, reviews and ships as one thing. When engineers say “the codebase”, they mean the place where every change to an application lands: the code itself, its tests, the scripts that build and check it, the database changes, and the documents that explain how the pieces fit.

The Twelve-Factor App, a methodology for building web services, gives the most useful working definition, because it makes the boundary something you can test. In a centralised version control system such as Subversion, a codebase is one repository. In a distributed system such as Git, it is any set of repositories that share a root commit, the first commit in the history, the one with no parent. Two repositories that grew from the same first commit are one codebase, however far apart they have drifted. Two that started separately are two codebases, even if they hold nearly identical files.

The same document pairs that definition with a second rule: one codebase, many deploys. Each deploy is one running instance of the application, and there are usually several at once. Production is one, staging another, and every developer’s local copy counts as well. They all run the same codebase, often at different commits: a developer has work staging has not seen, and staging has work production has not seen.

What belongs inside follows from that. The source and everything needed to turn it into a working program belong there, including a manifest that names each dependency and its version. The dependencies themselves usually stay outside, fetched by a package manager. Values that change from one deploy to the next stay outside too, and above all credentials: a password committed once travels with every copy of the history afterwards.

A codebase can hold several programs. A monorepo keeps a frontend, an API and the database code side by side in one repository, so a single commit can change all three. By the root-commit rule that is still one codebase. The opposite choice spreads an application across repositories with separate histories, and then each repository is its own codebase. Twelve-factor is blunt about the result: if there are several codebases, it is a distributed system, and each component is an app in its own right.

Use the concept in a decision

Put “What is a codebase?” to work in a real decision. We can help relate the concept to the system you are building or buying. The shape of a system worked out on paper first, so the parts fit and the cost is known before anyone writes code.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. One infrastructure plan for 14 departments — 14 research departments on one infrastructure plan

One infrastructure plan for 14 departments Get in touch →

Why it matters

The boundary decides what can change together. Inside one codebase, a change to an API and the screen that calls it can be written, reviewed and merged as one unit. Across two codebases the same change becomes two pull requests, two releases and a period in which one side is ahead of the other. Both are workable; the price is set by where the line was drawn.

The codebase is also the unit that standards run over. A test suite, a linter, a formatter, a commit message rule and a task runner all take “the codebase” as their scope, and whatever sits outside it is checked by none of them.

Early decisions harden in a codebase faster than anywhere else. Every line written after a choice assumes that choice, so a layering rule or a quality bar set in the first weeks becomes the shape of everything that follows, and reversing it later means rewriting the code that grew around it.

For people, a codebase is the thing a newcomer receives. If it holds everything needed to build, test and run the application, one clone and one command get them started. If the knowledge lives partly in the code and partly in someone’s memory, the first week goes on finding out which is which.

What it looks like in practice

Go, TypeScript, SQL, Python and shell share one platform's repository, where the codebase is a polyglot monorepo and each language arrives with its own way to build, test and lint. The answer was a single entry point: a go-task layer that grew to 578 named targets, 50 in the root file and 528 in namespaced files beneath it, covering native, Docker and HTTPS development modes, code quality, database management and deployment. Anyone can run `task --list` and see the whole toolchain. The number is also a warning: 578 targets is past what anyone can hold in their head, so the naming and the namespacing are what keep it usable.

Another project began with an empty repository on 25 November 2025, and three decisions made in its first weeks still held nine months and several thousand commits later: a layered stack, data access behind stored functions, and a zero-warnings bar put in before there was much code to hold to it. The reasoning behind that last one is the heart of this article: a standard introduced at commit 5,000 is a cleanup project, whereas the same standard at commit 50 is just how the repository works.

The walls inside a codebase can be the design itself. In the founder's architecture contribution under NextMariner, a maritime platform built in three layers, the Next.js frontend handles presentation and nothing else, a deliberately thin Go API built with Huma over Fiber routes, validates and orchestrates, and the business logic lives in PostgreSQL functions. When a rule changes, it changes in one layer, in SQL. The boundaries were written down and enforced in review, because a convention nobody polices stops being one.

This site's own codebase draws a different kind of line. Its project pages are written in a separate repository, exported into this one by a script and committed, so the site builds on its own. Inside this codebase those pages are generated files, and the instructions for working here say they are never edited by hand.

Where it goes wrong

Copied code drifts apart

Two applications need the same helper, and the fastest move is to copy it across. A year later there are two versions with different fixes, and a bug patched in one is still live in the other. Twelve-factor calls several apps sharing the same code a

violation and names the fix: move the shared part into a library, pull it in through the dependency manager, and let each codebase state the version it uses.

A secret lands in the history

A connection string or an API key gets committed in a configuration file, and from then on it sits in the history that every clone carries, even after a later commit deletes the line. The twelve-factor test is worth running before each commit: could this codebase be published as open source today without exposing a single credential? Anything that would fail that test belongs in the environment of each deploy.

A generated file gets a hand edit

A file written by a script looks like any other source file, so someone fixes a typo in it directly. The next run of the script writes the old text back and the fix vanishes without a word. Mark generated paths clearly, keep the generator as their only writer, and send every correction to the generator’s input.

Internal walls wear thin

Under a deadline, a query goes straight into a request handler because it is the shortest path, and the layering the project started with stops describing the code. Each shortcut is small. Together they turn a two-line change into one that touches every layer. A written rule helps, review helps more, and a check that fails the build when one layer reaches into another helps most.

A standard covers half the code

A linter set up for the frontend leaves the shell scripts and the SQL alone, and those files collect exactly the problems the linter was meant to stop. A codebase-wide bar has to name every language in the codebase, which is the hidden price of a polyglot repository.

A fork starts a second product

A team copies a repository to start a related product. Both copies keep the same root commit, so by the twelve-factor rule they are still one codebase, now serving two apps, and the drift described above begins at the scale of a whole repository. Comparing root commits shows which situation you are in:

```
git -C app-a rev-list --max-parents=0 HEAD
```

```
git -C app-b rev-list --max-parents=0 HEAD
```

Identical output means a shared root and a shared codebase; different output means two separate histories.

Further reading

- I. Codebase — the one-codebase, many-deploys rule and the shared-root-commit test behind it.
- II. Dependencies — declaring every dependency in a manifest and isolating it, so a codebase states exactly what it pulls in.
- III. Config — the open-source litmus test for what has to stay out of a codebase.
- gitsubmodules — how Git embeds one repository in another while each keeps its own history.
- git-rev-list — the reference for listing commits, including the root commits of a history.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a codebase?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/codebase/