Gleam Typed BEAM and JS language

CONCEPTS

Gleam compiles to Erlang (BEAM) or JavaScript. The target comes from gleam.toml, else Erlang, and --target overrides it.

src/*.gleam your modules
↓ gleam build
build/ BEAM files or JS
↓ gleam run
main() entry point
From source to running code

Tests live in test/ and gleam test runs main in <package>_test; dev code goes in dev/ with gleam dev.

Dependencies are in gleam.toml; manifest.toml is the resolved lock file, so commit it for applications.

PROJECTS

`gleam new <dir>`
Create a new package

`gleam run`
Compile and run main

`gleam run --target javascript`
Run on the JS target

`gleam run --runtime <rt>`
JS runtime: nodejs, deno or bun

`gleam shell`
Erlang REPL with your code loaded

`gleam clean`
Delete the build directory

BUILD & CHECK

`gleam build`
Compile package and dependencies

`gleam build --target <t>`
Pick erlang or javascript

`gleam build --warnings-as-errors`
Fail on any warning

`gleam check`
Type-check without code generation

`gleam export erlang-shipment`
Directory with bytecode and start script

TEST & FORMAT

`gleam test`
Run main of <package>_test

`gleam format`
Format the project

`gleam format --check`
Fail if a file is unformatted

DEPENDENCIES

`gleam add <pkg>`
Add latest compatible version

`gleam add --dev <pkg>`
Add as a dev dependency

`gleam remove <pkg>`
Remove a dependency

`gleam deps update [<pkg>...]`
Upgrade within gleam.toml limits

SYNTAX BASICS

```
let x = 1
```
Immutable binding

```
x |> f |> g
```
Pipe into the first argument

```
case v { 0 -> a  _ -> b }
```
Exhaustive pattern match

```
[first, ..rest]
```
List pattern or prepend

```
pub type T { A(n: Int)  B }
```
Custom type with variants

```
Person(..p, age: 30)
```
Record update

```
use x <- result.try(f())
```
Flatten callbacks

```
Ok(v) / Error(e)
```
Result variants

```
import gleam/io.{println}
```
Import module and items

HELP

`gleam help`
List all commands

`gleam help <command>`
Help for one command

`gleam --version`
Print the compiler version