Gleam Typed BEAM and JS language

START HERE

`gleam new <dir>`
Create a project

`gleam build --target javascript`
Compile to JS

`gleam run`
Compile and run main

`gleam help <cmd>`
Show command help

`gleam test`
Run the test suite

`gleam add <pkg>`
Add a dependency

`gleam check`
Type-check only

`gleam format`
Format the code

src/*.gleam your modules → `gleam build` → **build/** BEAM files or JS → `gleam run` → **main()** entry point

From source to running code

CONCEPTS

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

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
--template javascript sets the JS target.

`gleam new <dir> --skip-git`
Skip .git and .github
--skip-github omits only .github.

`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

PROJECTS (cont.)

`gleam shell`
Erlang REPL with your code loaded
It speaks Erlang syntax, not Gleam.

`gleam clean`
Delete the build directory

`gleam dev`
Run main of <package>_dev

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
Handy in CI.

`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 test --target <t>`
Test on one target

TEST & FORMAT (cont.)

`gleam format`
Format the project

`gleam format <path>...`
Format given files or dirs

`gleam format --check`
Fail if a file is unformatted
Use it in CI.

DEPENDENCIES

`gleam add <pkg>`
Add latest compatible version

`gleam add <pkg>@<version>`
Pin a major or full version
Example: gleam add gleam_json@3

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

`gleam remove <pkg>`
Remove a dependency

`gleam deps download`
Fetch deps, write manifest.toml
Normally runs automatically.

`gleam deps list`
List all dependency versions

`gleam deps outdated`
Show dependencies behind latest

`gleam deps update [<pkg>...]`
Upgrade within gleam.toml limits
Does not edit gleam.toml.

DEPENDENCIES (cont.)

`gleam update`
Alias for deps update

`gleam deps tree`
Print the dependency tree

`gleam deps tree --invert <pkg>`
Who depends on this package

SYNTAX BASICS

`let x = 1`
Immutable binding

`const limit = 10`
Top-level constant
Value must be a literal.

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

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

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

`"pre" <> rest`
String prefix pattern, concat

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

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

`fn f(x: Int, add n: Int) -> Int`
Labelled argument

SYNTAX BASICS (cont.)

`f(1, add:)`
Label shorthand when names match

`use x <- result.try(f())`
Flatten callbacks
Rest of the block becomes the callback.

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

`Some(v) / None`
Option, from gleam/option

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

`<<1, 2>>`
Bit array literal

`let assert Ok(v) = r`
Crash if the pattern fails

`todo as "msg"`
Placeholder that crashes if run

`assert a == b`
Boolean check, used in tests

`echo value`
Print a debug value

PUBLISHING & DOCS

`gleam docs build --open`
Build HTML docs and open them

`gleam publish`
Publish the package to Hex

PUBLISHING & DOCS (cont.)

`gleam hex authenticate`
Log in to Hex

RECIPES

Start a project and add libraries

```
gleam new app && cd app
gleam add gleam_stdlib
gleam_json
gleam add --dev gleeunit
gleam test
```

CI check

```
gleam format --check
gleam build --warnings-as-
errors
gleam test
```

Same tests on both targets

```
gleam test --target erlang
gleam test --target javascript
```

Pipeline with use and Result

```
pub fn load(id) {
  use user ←
result.try(find(id))
  use age ←
result.try(parse(user.age))
  Ok(age + 1)
}
```

HELP

`gleam help`
List all commands

`gleam help <command>`
Help for one command

`gleam --version`
Print the compiler version