Go — Go toolchain commands

START HERE

`go mod init <module>`
Start a new module

`go run .`
Compile and run the main package

`go build ./...`
Compile every package

`go test ./...`
Run all tests

`go fmt ./...`
Format all sources

`go vet ./...`
Report suspicious code

`go get <pkg>@<version>`
Add or change a dependency

`go mod tidy`
Sync go.mod with imports

CONCEPTS

A module (go.mod) is a tree of packages; a package is one directory of .go files. Commands take package patterns: . is the current package, ./... is it plus everything below.

Module go.mod, versioned unit
Package one directory, one import path
.go file declarations; _test.go for tests

Module, package and file

go build only checks and discards non-main packages; for a main package it writes a binary. go install puts the binary in $GOBIN (default $GOPATH/bin).

go.sum holds checksums of dependencies: commit it together with go.mod.

BUILD AND RUN

`go run .`
Compile and run the package here

`go run <file.go> <args>`
Run files, passing program args
Arguments after the package or files go to your program, not to go. A literal -- is passed on too.

`go build ./...`
Compile all packages, discard results

`go build -o <path> .`
Write the binary to a chosen path

`go install <pkg>@latest`
Install a tool binary into $GOBIN

`go build -race`
Build with the race detector

`go build -ldflags="-s -w"`
Strip symbols and debug info

`go build -ldflags="-X main.version=<v>"`
Set a string variable at link time

`go build -tags <tag,...>`
Enable build tags

`go build -trimpath`
Remove local paths from the binary

`go build -v`
List packages as they compile

`go clean -cache`
Delete the build cache

`go clean -modcache`
Delete downloaded modules

BUILD AND RUN (cont.)

`go clean -testcache`
Expire cached test results

CROSS-COMPILING

`GOOS=<os> GOARCH=<arch> go build`
Build for another platform

`go tool dist list`
List supported GOOS/GOARCH pairs

`CGO_ENABLED=0 go build`
Static build without C toolchain

TESTING

`go test`
Test the package in the current dir

`go test ./...`
Test every package below here

`go test -v`
Print each test and its output

`go test -run <regexp>`
Run only matching tests
Anchor it: -run '^TestFoo$'. Use / to match subtests: -run 'TestFoo/case'.

`go test -count=1`
Bypass the test result cache
Package-list runs cache passing results; any -count disables that, 1 is the idiom.

`go test -race`
Run with the race detector

`go test -short`
Skip tests that check testing.Short()

`go test -failfast`
Stop after the first failure

`go test -timeout <d>`
Abort a test binary after d
Default is 10m.

`go test -skip <regexp>`
Skip matching tests

`go test -shuffle=on`
Randomise test order

`go test -cover`
Show statement coverage

`go test -coverprofile=<file>`
Write a coverage profile

`go tool cover -html=<file>`
Browse coverage in a browser

`go test -bench <regexp>`
Run benchmarks (. for all)
Benchmarks run after the tests; add -run '^$' to skip those.

`go test -benchmem`
Add allocation stats to benchmarks

TESTING (cont.)

`go test -cpuprofile <file>`
Write a CPU profile

`go tool pprof <file>`
Inspect a profile interactively

`go test -fuzz <regexp>`
Fuzz one target, one package at a time

FORMAT AND CHECK

`go fmt ./...`
Format files with gofmt -l -w

`go vet ./...`
Find likely mistakes

`go doc <pkg>[.<Symbol>]`
Show docs for a package or symbol

`go doc -all <pkg>`
Show every exported item

`go generate ./...`
Run //go:generate directives
Never runs automatically during build or test.

DEPENDENCIES

`go get <pkg>`
Add a dependency at the latest version

`go get <pkg>@<version>`
Pin a version, branch or commit

`go get <pkg>@none`
Remove a dependency

`go get -u ./...`
Upgrade deps of this module

`go get -u=patch ./...`
Upgrade to newer patch releases only

`go get -tool <pkg>`
Track a tool dependency in go.mod
Go 1.24+; run it later with go tool <name>.

`go list -m all`
List all modules in the build

`go list -m -u all`
Show modules with available updates

`go list ./...`
List package paths below here

`go list -f '{{.Dir}}' <pkg>`
Format output with a template

`go mod why <pkg>`
Why a package is needed

MODULES

`go mod init <module>`
Create go.mod here

MODULES (cont.)

`go mod tidy`
Add missing, remove unused modules

`go mod tidy -diff`
Show tidy changes, exit 1 if any
Handy in CI.

`go mod download`
Fetch modules into the cache

`go mod verify`
Check cached modules against go.sum

`go mod vendor`
Copy dependencies to vendor/

`go mod edit -replace <old>=<new>`
Add a replace directive

WORKSPACES

`go work init <dir>...`
Create go.work for local modules

`go work use <dir>`
Add a module to the workspace

`go env GOWORK`
Show which go.work is active

`GOWORK=off go <cmd>`
Ignore the workspace once

ENVIRONMENT

`go version`
Print the Go toolchain version

`go env`
Print all Go environment settings

`go env <VAR>...`
Print specific settings

`go env -w <VAR>=<value>`
Persist a setting

`GOPRIVATE=<glob>`
Skip proxy and checksum db for paths

`GOPROXY=<url>,direct`
Choose the module proxy

RECIPES

Reproducible small release binary

```
CGO_ENABLED=0 GOOS=linux
GOARCH=amd64 \
  go build -trimpath -ldflags="-
s -w -X main.version=1.2.3" \
  -o dist/app .
```

Coverage report of the whole module

```
go test -
coverprofile=cover.out ./...
go tool cover -html=cover.out
```

Debug a flaky test

```
go test -race -count=50 -run
'^TestFoo$' ./pkg/...
```