

Go 1.25 Release Notes



- [Introduction to Go 1.25 {#introduction}](#)
- [Changes to the language {#language}](#)
- [Tools {#tools}](#)
 - [Go command {#go-command}](#)
 - [Vet {#vet}](#)
- [Runtime {#runtime}](#)
 - [Container-aware GOMAXPROCS](#)
 - [New experimental garbage collector](#)
 - [Trace flight recorder](#)
 - [Change to unhandled panic output](#)
 - [VMA names on Linux](#)
- [Compiler {#compiler}](#)
 - [nil pointer bug](#)
 - [DWARF5 support](#)
 - [Faster slices](#)
- [Linker {#linker}](#)
- [Standard library {#library}](#)
 - [New testing/synctest package](#)
 - [New experimental encoding/json/v2 package {#json_v2}](#)
 - [Minor changes to the library {#minor_library_changes}](#)
 - [archive/tar](#)
 - [encoding/asn1](#)
 - [crypto](#)
 - [crypto/ecdsa](#)
 - [crypto/ed25519](#)
 - [crypto/elliptic](#)
 - [crypto/rsa](#)
 - [crypto/sha1](#)

- [crypto/sha3](#)
 - [crypto/tls](#)
 - [crypto/x509](#)
 - [debug/elf](#)
 - [go/ast](#)
 - [go/parser](#)
 - [go/token](#)
 - [go/types](#)
 - [hash](#)
 - [hash/maphash](#)
 - [io/fs](#)
 - [log/slog](#)
 - [mime/multipart](#)
 - [net](#)
 - [net/http](#)
 - [os](#)
 - [reflect](#)
 - [regexp/syntax](#)
 - [runtime](#)
 - [runtime/pprof](#)
 - [sync](#)
 - [testing](#)
 - [testing/fstest](#)
 - [unicode](#)
 - [unique](#)
 - [Ports {#ports}](#)
 - [Darwin](#)
 - [Windows](#)
 - [AMD64](#)
 - [Loong64](#)
 - [RISC-V](#)
-

Introduction to Go 1.25

{ #introduction }

The latest Go release, version 1.25, arrives in [August 2025](#), six months after [Go 1.24](#). Most of its changes are in the implementation of the toolchain, runtime, and libraries. As always, the release maintains the Go 1 promise of compatibility. We expect almost all Go programs to continue to compile and run as before.

Changes to the language

{ #language }

There are no language changes that affect Go programs in Go 1.25. However, in the [language specification](#) the notion of core types has been removed in favor of dedicated prose. See the respective [blog post](#) for more information.

Tools { #tools }

Go command { #go-command }

The `go build -asan` option now defaults to doing leak detection at program exit. This will report an error if memory allocated by C is not freed and is not referenced by any other memory allocated by either C or Go. These new error reports may be disabled by setting `ASAN_OPTIONS=detect_leaks=0` in the environment when running the program.

The Go distribution will include fewer prebuilt tool binaries. Core toolchain binaries such as the compiler and linker will still be included, but tools not invoked by build or test operations will be built and run by `go tool` as needed.

The new `go.mod ignore` [directive](#) can be used to specify directories the `go` command should ignore. Files in these directories and their subdirectories will be ignored by the `go` command when matching package patterns, such as `all` or `./...`, but will still be included in module zip files.

The new `go doc -http` option will start a documentation server showing documentation for the requested object, and open the documentation in a browser window.

The new `go version -m -json` option will print the JSON encodings of the `runtime/debug.BuildInfo` structures embedded in the given Go binary files.

The `go` command now supports using a subdirectory of a repository as the path for a module root, when [resolving a module path](#) using the syntax `<meta name="go-import" content="root-path vcs repo-url subdir">` to indicate that the `root-path` corresponds to the `subdir` of the `repo-url` with version control system `vcs`.

The new `work` package pattern matches all packages in the work (formerly called main) modules: either the single work module in module mode or the set of workspace modules in workspace mode.

When the `go` command updates the `go` line in a `go.mod` or `go.work` file, it [no longer](#) adds a toolchain line specifying the command's current version.

Vet {#vet}

The `go vet` command includes new analyzers:

- [waitgroup](#), which reports misplaced calls to `sync.WaitGroup.Add`; and
- [hostport](#), which reports uses of `fmt.Sprintf("%s:%d", host, port)` to construct addresses for `net.Dial`, as these will not work with IPv6; instead it suggests using `net.JoinHostPort`.

Runtime {#runtime}

Container-aware GOMAXPROCS

The default behavior of the `GOMAXPROCS` has changed. In prior versions of Go, `GOMAXPROCS` defaults to the number of logical CPUs available at startup (`runtime.NumCPU`). Go 1.25 introduces two changes:

1. On Linux, the runtime considers the CPU bandwidth limit of the cgroup containing the process, if any. If the CPU bandwidth limit is lower than the number of logical CPUs available, `GOMAXPROCS` will default to the lower limit. In container runtime systems like Kubernetes, cgroup CPU bandwidth limits generally correspond to the "CPU limit" option. The Go runtime does not consider the "CPU requests" option.

2. On all OSes, the runtime periodically updates `GOMAXPROCS` if the number of logical CPUs available or the cgroup CPU bandwidth limit change.

Both of these behaviors are automatically disabled if `GOMAXPROCS` is set manually via the `GOMAXPROCS` environment variable or a call to `runtime.GOMAXPROCS`. They can also be disabled explicitly with the [GODEBUG settings](#) `containermaxprocs=0` and `updatemaxprocs=0`, respectively.

In order to support reading updated cgroup limits, the runtime will keep cached file descriptors for the cgroup files for the duration of the process lifetime.

New experimental garbage collector

A new garbage collector is now available as an experiment. This garbage collector's design improves the performance of marking and scanning small objects through better locality and CPU scalability. Benchmark result vary, but we expect somewhere between a 10—40% reduction in garbage collection overhead in real-world programs that heavily use the garbage collector.

The new garbage collector may be enabled by setting `GOEXPERIMENT=greenteagc` at build time. We expect the design to continue to evolve and improve. To that end, we encourage Go developers to try it out and report back their experiences. See the [GitHub issue](#) for more details on the design and instructions for sharing feedback.

Trace flight recorder

[Runtime execution traces](#) have long provided a powerful, but expensive way to understand and debug the low-level behavior of an application. Unfortunately, because of their size and the cost of continuously writing an execution trace, they were generally impractical for debugging rare events.

The new `runtime/trace.FlightRecorder` API provides a lightweight way to capture a runtime execution trace by continuously recording the trace into an in-memory ring buffer. When a significant event occurs, a program can call `FlightRecorder.WriteTo` to snapshot the last few seconds of the trace to a file. This approach produces a much smaller trace by enabling applications to capture only the traces that matter.

The length of time and amount of data captured by a `FlightRecorder` may be configured within the `FlightRecorderConfig`.

Change to unhandled panic output

The message printed when a program exits due to an unhandled panic that was recovered and repanicked no longer repeats the text of the panic value.

Previously, a program which panicked with `panic("PANIC")`, recovered the panic, and then repanicked with the original value would print:

```
panic: PANIC [recovered]
panic: PANIC
```

This program will now print:

```
panic: PANIC [recovered, repanicked]
```

VMA names on Linux

On Linux systems with kernel support for anonymous virtual memory area (VMA) names (`CONFIG_ANON_VMA_NAME`), the Go runtime will annotate anonymous memory mappings with context about their purpose. e.g., `[anon: Go: heap]` for heap memory. This can be disabled with the [GODEBUG setting](#) `decoratemappings=0`.

Compiler {#compiler}

`nil` pointer bug

This release fixes a [compiler bug](#), introduced in Go 1.21, that could incorrectly delay nil pointer checks. Programs like the following, which used to execute successfully (incorrectly), will now (correctly) panic with a nil-pointer exception:

```
package main

import "os"

func main() {
    f, err := os.Open("nonExistentFile")
    name := f.Name()
    if err != nil {
        return
    }
    println(name)
}
```

This program is incorrect because it uses the result of `os.Open` before checking the error. If `err` is non-nil, then the `f` result may be nil, in which case `f.Name()` should panic. However, in Go versions 1.21 through 1.24, the compiler incorrectly delayed the nil check until *after* the error check, causing the program to execute successfully, in violation of the Go spec. In Go 1.25, it will no longer run successfully. If this change is affecting your code, the solution is to put the non-nil error check earlier in your code, preferably immediately after the error-generating statement.

DWARF5 support

The compiler and linker in Go 1.25 now generate debug information using [DWARF version 5](#). The newer DWARF version reduces the space required for debugging information in Go binaries, and reduces the time for linking, especially for large Go binaries. DWARF 5 generation can be disabled by setting the environment variable `GOEXPERIMENT=nodwarf5` at build time (this fallback may be removed in a future Go release).

Faster slices

The compiler can now allocate the backing store for slices on the stack in more situations, which improves performance. This change has the potential to amplify the effects of incorrect [unsafe.Pointer](#) usage, see for example [issue 73199](#). In order to track down these problems, the [bisect tool](#) can be used to find the allocation causing trouble using the `-compile=variablemake` flag. All such new stack allocations can also be turned off using `-gcflags=all=-d=variablemakehash=n`.

Linker {#linker}

The linker now accepts a `-funcalign=N` command line option, which specifies the alignment of function entries. The default value is platform-dependent, and is unchanged in this release.

Standard library {#library}

New testing/synctest package

The new [testing/synctest](#) package provides support for testing concurrent code.

The `Test` function runs a test function in an isolated "bubble". Within the bubble, time is virtualized: [time](#) package functions operate on a fake clock and the clock moves forward instantaneously if all goroutines in the bubble are blocked.

The `Wait` function waits for all goroutines in the current bubble to block.

This package was first available in Go 1.24 under `GOEXPERIMENT=synctest`, with a slightly different API. The experiment has now graduated to general availability. The old API is still present if `GOEXPERIMENT=synctest` is set, but will be removed in Go 1.26.

New experimental encoding/json/v2 package {#json_v2}

Go 1.25 includes a new, experimental JSON implementation, which can be enabled by setting the environment variable `GOEXPERIMENT=jsonv2` at build time.

When enabled, two new packages are available:

- The `encoding/json/v2` package is a major revision of the `encoding/json` package.
- The `encoding/json/jsontext` package provides lower-level processing of JSON syntax.

In addition, when the "jsonv2" GOEXPERIMENT is enabled:

- The `encoding/json` package uses the new JSON implementation. Marshaling and unmarshaling behavior is unaffected, but the text of errors returned by package function may change.
- The `encoding/json` package contains a number of new options which may be used to configure the marshaler and unmarshaler.

The new implementation performs substantially better than the existing one under many scenarios. In general, encoding performance is at parity between the implementations and decoding is substantially faster in the new one. See the github.com/go-json-experiment/jsonbench repository for more detailed analysis.

See the [proposal issue](#) for more details.

We encourage users of `encoding/json` to test their programs with `GOEXPERIMENT=jsonv2` enabled to help detect any compatibility issues with the new implementation.

We expect the design of `encoding/json/v2` to continue to evolve. We encourage developers to try out the new API and provide feedback on the [proposal issue](#).

Minor changes to the library {#minor_library_changes}

[archive/tar](#)

The `Writer.AddFS` implementation now supports symbolic links for filesystems that implement `io/fs.ReadLinkFS`.

[encoding/asn1](#)

`Unmarshal` and `UnmarshalWithParams` now parse the ASN.1 types `T61String` and `BMPString` more consistently. This may result in some previously accepted malformed encodings now being rejected.

[crypto](#)

`MessageSigner` is a new signing interface that can be implemented by signers that wish to hash the message to be signed themselves. A new function is also introduced, `SignMessage`, which attempts to upgrade a `Signer` interface to `MessageSigner`, using the `MessageSigner.SignMessage` method if successful, and `Signer.Sign` if not. This can be used when code wishes to support both `Signer` and `MessageSigner`.

Changing the `fips140` [GODEBUG setting](#) after the program has started is now a no-op. Previously, it was documented as not allowed, and could cause a panic if changed.

SHA-1, SHA-256, and SHA-512 are now slower on amd64 when AVX2 instructions are not available. All server processors (and most others) produced since 2015 support AVX2.

[crypto/ecdsa](#)

The new `ParseRawPrivateKey`, `ParseUncompressedPublicKey`, `PrivateKey.Bytes`, and `PublicKey.Bytes` functions and methods implement low-level encodings, replacing the need to use `crypto/elliptic` or `math/big` functions and methods.

When FIPS 140-3 mode is enabled, signing is now four times faster, matching the performance of non-FIPS mode.

[crypto/ed25519](#)

When FIPS 140-3 mode is enabled, signing is now four times faster, matching the performance of non-FIPS mode.

[crypto/elliptic](#)

The hidden and undocumented `Inverse` and `CombinedMult` methods on some `Curve` implementations have been removed.

[crypto/rsa](#)

`PublicKey` no longer claims that the modulus value is treated as secret. `VerifyPKCS1v15` and `VerifyPSS` already warned that all inputs are public and could be leaked, and there are mathematical attacks that can recover the modulus from other public values.

Key generation is now three times faster.

`crypto/sha1`

Hashing is now two times faster on amd64 when SHA-NI instructions are available.

`crypto/sha3`

The new `SHA3.Clone` method implements `hash.Cloner`.

Hashing is now two times faster on Apple M processors.

`crypto/tls`

The new `ConnectionState.CurveID` field exposes the key exchange mechanism used to establish the connection.

The new `Config.GetEncryptedClientHelloKeys` callback can be used to set the `EncryptedClientHelloKey`s for a server to use when a client sends an Encrypted Client Hello extension.

SHA-1 signature algorithms are now disallowed in TLS 1.2 handshakes, per [RFC 9155](#). They can be re-enabled with the [GODEBUG setting](#) `tlssha1=1`.

When [FIPS 140-3 mode](#) is enabled, Extended Master Secret is now required in TLS 1.2, and Ed25519 and X25519MLKEM768 are now allowed.

TLS servers now prefer the highest supported protocol version, even if it isn't the client's most preferred protocol version.

Both TLS clients and servers are now stricter in following the specifications and in rejecting off-spec behavior. Connections with compliant peers should be unaffected.

`crypto/x509`

`CreateCertificate`, `CreateCertificateRequest`, and `CreateRevocationList` can now accept a `crypto.MessageSigner` signing interface as well as `crypto.Signer`. This allows these functions to use signers which implement "one-shot" signing interfaces, where hashing is done as part of the signing operation, instead of by the caller.

`CreateCertificate` now uses truncated SHA-256 to populate the `SubjectKeyId` if it is missing. The [GODEBUG setting](#) `x509sha256skid=0` reverts to SHA-1.

`ParseCertificate` now rejects certificates which contain a BasicConstraints extension that contains a negative pathLenConstraint.

`ParseCertificate` now handles strings encoded with the ASN.1 T61String and BMPString types more consistently. This may result in some previously accepted malformed encodings now being rejected.

`debug/elf`

The `debug/elf` package adds two new constants:

- `PT_RISCV_ATTRIBUTES`
- `SHT_RISCV_ATTRIBUTES` for RISC-V ELF parsing.

`go/ast`

The `FilterPackage`, `PackageExports`, and `MergePackageFiles` functions, and the `MergeMode` type and its constants, are all deprecated, as they are for use only with the long-deprecated `Object` and `Package` machinery.

The new `PreorderStack` function, like `Inspect`, traverses a syntax tree and provides control over descent into subtrees, but as a convenience it also provides the stack of enclosing nodes at each point.

`go/parser`

The `ParseDir` function is deprecated.

`go/token`

The new `FileSet.AddExistingFiles` method enables existing `File`s to be added to a `FileSet`, or a `FileSet` to be constructed for an arbitrary set of `File`s, alleviating the problems associated with a single global `FileSet` in long-lived applications.

`go/types`

`Var` now has a `Var.Kind` method that classifies the variable as one of: package-level, receiver, parameter, result, local variable, or a struct field.

The new `LookupSelection` function looks up the field or method of a given name and receiver type, like the existing `LookupFieldOrMethod` function, but returns the result in the form of a `Selection`.

`hash`

The new `XOF` interface can be implemented by "extendable output functions", which are hash functions with arbitrary or unlimited output length such as `SHAKE`.

Hashes implementing the new `Cloner` interface can return a copy of their state. All standard library `Hash` implementations now implement `Cloner`.

hash/maphash

The new `Hash.Clone` method implements `hash.Cloner`.

io/fs

A new `ReadLinkFS` interface provides the ability to read symbolic links in a filesystem.

log/slog

`GroupAttrs` creates a group `Attr` from a slice of `Attr` values.

`Record` now has a `Source` method, returning its source location or nil if unavailable.

mime/multipart

The new helper function `FileContentDisposition` builds multipart Content-Disposition header fields.

net

`LookupMX` and `Resolver.LookupMX` now return DNS names that look like valid IP address, as well as valid domain names. Previously if a name server returned an IP address as a DNS name, `LookupMX` would discard it, as required by the RFCs. However, name servers in practice do sometimes return IP addresses.

On Windows, `ListenMulticastUDP` now supports IPv6 addresses.

On Windows, it is now possible to convert between an `os.File` and a network connection. Specifically, the `FileConn`, `FilePacketConn`, and `FileListener` functions are now implemented, and return a network connection or listener corresponding to an open file. Similarly, the `File` methods of `TCPConn`, `UDPConn`, `UnixConn`, `IPConn`, `TCPLListener`, and `UnixListener` are now implemented, and return the underlying `os.File` of a network connection.

net/http

The new `CrossOriginProtection` implements protections against [Cross-Site Request Forgery \(CSRF\)](#) by rejecting non-safe cross-origin browser requests. It uses [modern browser Fetch metadata](#), doesn't require tokens or cookies, and supports origin-based and pattern-based bypasses.

os

On Windows, `NewFile` now supports handles opened for asynchronous I/O (that is, `syscall.FILE_FLAG_OVERLAPPED` is specified in the `syscall.CreateFile` call). These handles are associated with the Go runtime's I/O completion port, which provides the following benefits for the

resulting `File`:

- I/O methods (`File.Read`, `File.Write`, `File.ReadAt`, and `File.WriteAt`) do not block an OS thread.
- Deadline methods (`File.SetDeadline`, `File.SetReadDeadline`, and `File.SetWriteDeadline`) are supported.

This enhancement is especially beneficial for applications that communicate via named pipes on Windows.

Note that a handle can only be associated with one completion port at a time. If the handle provided to `NewFile` is already associated with a completion port, the returned `File` is downgraded to synchronous I/O mode. In this case, I/O methods will block an OS thread, and the deadline methods have no effect.

The filesystems returned by `DirFS` and `Root.FS` implement the new `io/fs.ReadLinkFS` interface. `CopyFS` supports symlinks when copying filesystems that implement `io/fs.ReadLinkFS`.

The `Root` type supports the following additional methods:

- `Root.Chmod`
- `Root.Chown`
- `Root.Chtimes`
- `Root.Lchown`
- `Root.Link`
- `Root.MkdirAll`
- `Root.ReadFile`
- `Root.Readlink`
- `Root.RemoveAll`
- `Root.Rename`
- `Root.Symlink`
- `Root.WriteFile`

`reflect`

The new `TypeAssert` function permits converting a `Value` directly to a Go value of the given type. This is like using a type assertion on the result of `Value.Interface`, but avoids unnecessary memory allocations.

`regexp/syntax`

The `\p{name}` and `\P{name}` character class syntaxes now accept the names Any, ASCII, Assigned, Cn, and LC, as well as Unicode category aliases like `\p{Letter}` for `\pL`. Following [Unicode TR18](#), they also now use case-insensitive name lookups, ignoring spaces, underscores, and hyphens.

`runtime`

Cleanup functions scheduled by `AddCleanup` are now executed concurrently and in parallel, making cleanups more viable for heavy use like the `unique` package. Note that individual cleanups should still shunt their work to a new goroutine if they must execute or block for a long time to avoid blocking the cleanup queue.

A new `GODEBUG=checkfinalizers=1` setting helps find common issues with finalizers and cleanups, such as those described [in the GC guide](#). In this mode, the runtime runs diagnostics on each garbage collection cycle, and will also regularly report the finalizer and cleanup queue lengths to `stderr` to help identify issues with long-running finalizers and/or cleanups. See the [GODEBUG documentation](#) for more details.

The new `SetDefaultGOMAXPROCS` function sets `GOMAXPROCS` to the runtime default value, as if the `GOMAXPROCS` environment variable is not set. This is useful for enabling the [new GOMAXPROCS default](#) if it has been disabled by the `GOMAXPROCS` environment variable or a prior call to `GOMAXPROCS`.

[runtime/pprof](#)

The mutex profile for contention on runtime-internal locks now correctly points to the end of the critical section that caused the delay. This matches the profile's behavior for contention on `sync.Mutex` values. The `runtimecontentionstacks` setting for `GODEBUG`, which allowed opting in to the unusual behavior of Go 1.22 through 1.24 for this part of the profile, is now gone.

[sync](#)

The new `WaitGroup.Go` method makes the common pattern of creating and counting goroutines more convenient.

[testing](#)

The new methods `T.Attr`, `B.Attr`, and `F.Attr` emit an attribute to the test log. An attribute is an arbitrary key and value associated with a test.

For example, in a test named `TestF`, `t.Attr("key", "value")` emits:

```
=== ATTR TestF key value
```

With the `-json` flag, attributes appear as a new "attr" action.

The new `Output` method of `T`, `B` and `F` provides an `io.Writer` that writes to the same test output stream as `TB.Log`. Like `TB.Log`, the output is indented, but it does not include the file and line number.

The `AllocsPerRun` function now panics if parallel tests are running. The result of `AllocsPerRun` is inherently flaky if other tests are running. The new panicking behavior helps catch such bugs.

testing/fstest

`MapFS` implements the new `io/fs.ReadLinkFS` interface. `TestFS` will verify the functionality of the `io/fs.ReadLinkFS` interface if implemented. `TestFS` will no longer follow symlinks to avoid unbounded recursion.

unicode

The new `CategoryAliases` map provides access to category alias names, such as “Letter” for “L”.

The new categories `Cn` and `LC` define unassigned codepoints and cased letters, respectively. These have always been defined by Unicode but were inadvertently omitted in earlier versions of Go. The `C` category now includes `Cn`, meaning it has added all unassigned code points.

unique

The `unique` package now reclaims interned values more eagerly, more efficiently, and in parallel. As a consequence, applications using `Make` are now less likely to experience memory blow-up when lots of truly unique values are interned.

Values passed to `Make` containing `Handle`s previously required multiple garbage collection cycles to collect, proportional to the depth of the chain of `Handle` values. Now, once unused, they are collected promptly in a single cycle.

Ports {#ports}

Darwin

As [announced](#) in the Go 1.24 release notes, Go 1.25 requires macOS 12 Monterey or later. Support for previous versions has been discontinued.

Windows

Go 1.25 is the last release that contains the [broken](#) 32-bit windows/arm port (`GOOS=windows` `GOARCH=arm`). It will be removed in Go 1.26.

AMD64

In `GOAMD64=v3` mode or higher, the compiler will now use fused multiply-add instructions to make floating-point arithmetic faster and more accurate. This may change the exact floating-point values

that a program generates.

To avoid fusing use an explicit `float64` cast, like `float64(a*b)+c`.

Loong64

The linux/loong64 port now supports the race detector, gathering traceback information from C code using [runtime.SetCgoTraceback](#), and linking cgo programs with the internal link mode.

RISC-V

The linux/riscv64 port now supports the `plugin` build mode.

The `GORISCV64` environment variable now accepts a new value `rva23u64`, which selects the RVA23U64 user-mode application profile.

```
#2
[] [] [] 2026-10-26 20:26:29 UTC
[] [] [] 2026-10-26 20:26:29 UTC
```