Go 1.22 Release Notes

Table of Contents

Introduction to Go 1.22

The latest Go release, version 1.22, arrives six months after Go 1.21. 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

Go 1.22 makes two changes to "for" loops.

- Previously, the variables declared by a "for" loop were created once and updated by each iteration. In Go 1.22, each iteration of the loop creates new variables, to avoid accidental sharing bugs. The transition support tooling described in the proposal continues to work in the same way it did in Go 1.21.

- "For" loops may now range over integers. For example:

```
package main

import "fmt"

func main() {
  for i := range 10 {
    fmt.Println(10 - i)
  }
  fmt.Println("go1.22 has lift-off!")
}
```

  See the spec for details.

Go 1.22 includes a preview of a language change we are considering for a future version of Go: range-over-function iterators. Building with `GOEXPERIMENT=rangefunc` enables this feature.

Tools

Go command

Commands in workspaces can now use a `vendor` directory containing the dependencies of the workspace. The directory is created by `go work vendor`, and used by build commands when the `–mod` flag is set to `vendor`, which is the default when a workspace `vendor` directory is present.

Note that the `vendor` directory's contents for a workspace are different from those of a single module: if the directory at the root of a workspace also contains one of the modules in the workspace, its `vendor` directory can contain the dependencies of either the workspace or of the module, but not both.

`go get` is no longer supported outside of a module in the legacy `GOPATH` mode (that is, with `GO111MODULE=off`). Other build commands, such as `go build` and `go test`, will continue to work indefinitely for legacy `GOPATH` programs.

`go mod init` no longer attempts to import module requirements from configuration files for other vendoring tools (such as `Gopkg.lock`).

`go test –cover` now prints coverage summaries for covered packages that do not have their own test files. Prior to Go 1.22 a `go test –cover` run for such a package would report

```
? mymod/mypack [no test files]
```

and now with Go 1.22, functions in the package are treated as uncovered:

```
mymod/mypack coverage: 0.0% of statements
```

Note that if a package contains no executable code at all, we can't report a meaningful coverage percentage; for such packages the `go` tool will continue to report that there are no test files.

`go` build commands that invoke the linker now error out if an external (C) linker will be used but cgo is not enabled. (The Go runtime requires cgo support to ensure that it is compatible with any additional libraries added by the C linker.)

Trace

The `trace` tool's web UI has been gently refreshed as part of the work to support the new tracer, resolving several issues and improving the readability of various sub-pages. The web UI now supports exploring traces in a thread-oriented view. The trace viewer also now displays the

full duration of all system calls.
These improvements only apply for viewing traces produced by programs built with Go 1.22 or newer. A future release will bring some of these improvements to traces produced by older version of Go.

Vet

References to loop variables

The behavior of the `vet` tool has changed to match the new semantics (see above) of loop variables in Go 1.22. When analyzing a file that requires Go 1.22 or newer (due to its go.mod file or a per-file build constraint), `vet` no longer reports references to loop variables from within a function literal that might outlive the iteration of the loop. In Go 1.22, loop variables are created anew for each iteration, so such references are no longer at risk of using a variable after it has been updated by the loop.

New warnings for missing values after append

The `vet` tool now reports calls to `append` that pass no values to be appended to the slice, such as `slice = append(slice)`. Such a statement has no effect, and experience has shown that is nearly always a mistake.

New warnings for deferring `time.Since`

The vet tool now reports a non-deferred call to `time.Since(t)` within a `defer` statement. This is equivalent to calling `time.Now().Sub(t)` before the `defer` statement, not when the deferred function is called. In nearly all cases, the correct code requires deferring the `time.Since` call. For example:

```
t := time.Now()
defer log.Println(time.Since(t)) // non-deferred call to time.Since
tmp := time.Since(t); defer log.Println(tmp) // equivalent to the previous defer

defer func() {
  log.Println(time.Since(t)) // a correctly deferred call to time.Since
}()
```

New warnings for mismatched key-value pairs in `log/slog` calls

The vet tool now reports invalid arguments in calls to functions and methods in the structured logging package, `log/slog`, that accept alternating key/value pairs. It reports calls where an argument in a key position is neither a `string` nor a `slog.Attr`, and where a final key is missing its value.

Runtime

The runtime now keeps type-based garbage collection metadata nearer to each heap object, improving the CPU performance (latency or throughput) of Go programs by 1–3%. This change also reduces the memory overhead of the majority Go programs by approximately 1% by deduplicating redundant metadata. Some programs may see a smaller improvement because this change adjusts the size class boundaries of the memory allocator, so some objects may be moved up a size class.

A consequence of this change is that some objects' addresses that were previously always aligned to a 16 byte (or higher) boundary will now only be aligned to an 8 byte boundary. Some programs that use assembly instructions that require memory addresses to be more than 8-byte aligned and rely on the memory allocator's previous alignment behavior may break, but we expect such programs to be rare. Such programs may be built with `GOEXPERIMENT=noallocheaders` to revert to the old metadata layout and restore the previous alignment behavior, but package owners should update their assembly code to avoid the alignment assumption, as this workaround will be removed in a future release.

On the `windows/amd64 port`, programs linking or loading Go libraries built with `–buildmode=c–archive` or `–buildmode=c–shared` can now use the `SetUnhandledExceptionFilter` Win32 function to catch exceptions not handled by the Go runtime. Note that this was already supported on the `windows/386` port.

Compiler

Profile-guided Optimization (PGO) builds can now devirtualize a higher proportion of calls than previously possible. Most programs from a representative set of Go programs now see between 2 and 14% improvement at runtime from enabling PGO.

The compiler now interleaves devirtualization and inlining, so interface method calls are better optimized.

Go 1.22 also includes a preview of an enhanced implementation of the compiler's inlining phase that uses heuristics to boost inlinability at call sites deemed "important" (for example, in loops) and discourage inlining at call sites deemed "unimportant" (for example, on panic paths). Building with `GOEXPERIMENT=newinliner` enables the new call-site heuristics; see issue #61502 for more info and to provide feedback.

Linker

The linker's `–s` and `–w` flags are now behave more consistently across all platforms. The `–w` flag suppresses DWARF debug information generation. The `–s` flag suppresses symbol table generation. The `–s` flag also implies the `–w` flag, which can be negated with `–w=0`. That is, `–s –w=0` will generate a binary with DWARF debug information generation but without the symbol table.

On ELF platforms, the –B linker flag now accepts a special form: with –B `gobuildid`, the linker will generate a GNU build ID (the ELF `NT_GNU_BUILD_ID` note) derived from the Go build ID.

On Windows, when building with `-linkmode=internal`, the linker now preserves SEH information from C object files by copying the `.pdata` and `.xdata` sections into the final binary. This helps with debugging and profiling binaries using native tools, such as WinDbg. Note that until now, C functions' SEH exception handlers were not being honored, so this change may cause some programs to behave differently. `-linkmode=external` is not affected by this change, as external linkers already preserve SEH information.

Bootstrap

As mentioned in the Go 1.20 release notes, Go 1.22 now requires the final point release of Go 1.20 or later for bootstrap. We expect that Go 1.24 will require the final point release of Go 1.22 or later for bootstrap.

Standard library

New math/rand/v2 package

Go 1.22 includes the first "v2" package in the standard library, `math/rand/v2`. The changes compared to `math/rand` are detailed in proposal #61716. The most important changes are:

- The `Read` method, deprecated in `math/rand`, was not carried forward for `math/rand/v2`. (It remains available in `math/rand`.) The vast majority of calls to `Read` should use `crypto/rand`'s `Read` instead. Otherwise a custom `Read` can be constructed using the `Uint64` method.
- The global generator accessed by top-level functions is unconditionally randomly seeded. Because the API guarantees no fixed sequence of results, optimizations like per-thread random generator states are now possible.
- The `Source` interface now has a single `Uint64` method; there is no `Source64` interface.
- Many methods now use faster algorithms that were not possible to adopt in `math/rand` because they changed the output streams.
- The `Intn`, `Int31`, `Int31n`, `Int63`, and `Int64n` top-level functions and methods from `math/rand` are spelled more idiomatically in `math/rand/v2`: `IntN`, `Int32`, `Int32N`, `Int64`, and `Int64N`. There are also new top-level functions and methods `Uint32`, `Uint32N`, `Uint64`, `Uint64N`, and `UintN`.
- The new generic function `N` is like `Int64N` or `Uint64N` but works for any integer type. For example a random duration from 0 up to 5 minutes is `rand.N(5*time.Minute)`.
- The Mitchell & Reeds LFSR generator provided by `math/rand`'s `Source` has been replaced by two more modern pseudo-random generator sources: `ChaCha8` and `PCG`.

> ChaCha8 is a new, cryptographically strong random number generator roughly similar to PCG in efficiency. ChaCha8 is the algorithm used for the top-level functions in `math/rand/v2`. As of Go 1.22, `math/rand`'s top-level functions (when not explicitly seeded) and the Go runtime also use ChaCha8 for randomness.

We plan to include an API migration tool in a future release, likely Go 1.23.

New go/version package

The new `go/version` package implements functions for validating and comparing Go version strings.

Enhanced routing patterns

HTTP routing in the standard library is now more expressive. The patterns used by `net/http.ServeMux` have been enhanced to accept methods and wildcards.

Registering a handler with a method, like `"POST /items/create"`, restricts invocations of the handler to requests with the given method. A pattern with a method takes precedence over a matching pattern without one. As a special case, registering a handler with `"GET"` also registers it with `"HEAD"`.

Wildcards in patterns, like `/items/{id}`, match segments of the URL path. The actual segment value may be accessed by calling the `Request.PathValue` method. A wildcard ending in "...", like `/files/{path...}`, must occur at the end of a pattern and matches all the remaining segments.

A pattern that ends in "/" matches all paths that have it as a prefix, as always. To match the exact pattern including the trailing slash, end it with `{$}`, as in `/exact/match/{$}`.

If two patterns overlap in the requests that they match, then the more specific pattern takes precedence. If neither is more specific, the patterns conflict. This rule generalizes the original precedence rules and maintains the property that the order in which patterns are registered does not matter.

This change breaks backwards compatibility in small ways, some obvious—patterns with "{" and "}" behave differently— and some less so—treatment of escaped paths has been improved. The change is controlled by a `GODEBUG` field named `httpmuxgo121`. Set `httpmuxgo121=1` to restore the old behavior.

Minor changes to the library

As always, there are various minor changes and updates to the library, made with the Go 1 promise of compatibility in mind. There are also various performance improvements, not enumerated here.

archive/tar

The new method `Writer.AddFS` adds all of the files from an `fs.FS` to the archive.

archive/zip

The new method `Writer.AddFS` adds all of the files from an `fs.FS` to the archive.

bufio

When a `SplitFunc` returns `ErrFinalToken` with a `nil` token, `Scanner` will now stop immediately. Previously, it would report a final empty token before stopping, which was usually not desired. Callers that do want to report a final empty token can do so by returning `[]byte{}` rather than `nil`.

cmp

The new function `Or` returns the first in a sequence of values that is not the zero value.

crypto/tls

`ConnectionState.ExportKeyingMaterial` will now return an error unless TLS 1.3 is in use, or the `extended_master_secret` extension is supported by both the server and client. `crypto/tls` has supported this extension since Go 1.20. This can be disabled with the `tlsunsafeekm=1` GODEBUG setting.

By default, the minimum version offered by `crypto/tls` servers is now TLS 1.2 if not specified with `config.MinimumVersion`, matching the behavior of `crypto/tls` clients. This change can be reverted with the `tls10server=1` GODEBUG setting.

By default, cipher suites without ECDHE support are no longer offered by either clients or servers during pre-TLS 1.3 handshakes. This change can be reverted with the `tlsrsakex=1` GODEBUG setting.

crypto/x509

The new `CertPool.AddCertWithConstraint` method can be used to add customized constraints to root certificates to be applied during chain building.

On Android, root certificates will now be loaded from `/data/misc/keychain/certs-added` as well as `/system/etc/security/cacerts`.

A new type, `OID`, supports ASN.1 Object Identifiers with individual components larger than 31 bits. A new field which uses this type, `Policies`, is added to the `Certificate` struct, and is now populated during parsing. Any OIDs which cannot be represented using a `asn1.ObjectIdentifier` will appear in `Policies`, but not in the old `PolicyIdentifiers` field. When calling `CreateCertificate`, the `Policies` field is

ignored, and policies are taken from the `PolicyIdentifiers` field. Using the `x509usepolicies=1` GODEBUG setting inverts this, populating certificate policies from the `Policies` field, and ignoring the `PolicyIdentifiers` field. We may change the default value of `x509usepolicies` in Go 1.23, making `Policies` the default field for marshaling.

database/sql

The new `Null[T]` type provide a way to scan nullable columns for any column types.

debug/elf

Constant R_MIPS_PC32 is defined for use with MIPS64 systems.

Additional R_LARCH_* constants are defined for use with LoongArch systems.

encoding

The new methods `AppendEncode` and `AppendDecode` added to each of the `Encoding` types in the packages `encoding/base32`, `encoding/base64`, and `encoding/hex` simplify encoding and decoding from and to byte slices by taking care of byte slice buffer management.

The methods `base32.Encoding.WithPadding` and `base64.Encoding.WithPadding` now panic if the `padding` argument is a negative value other than `NoPadding`.

encoding/json

Marshaling and encoding functionality now escapes `'\b'` and `'\f'` characters as `\b` and `\f` instead of `\u0008` and `\u000c`.

go/ast

The following declarations related to syntactic identifier resolution are now deprecated: `Ident.Obj`, `Object`, `Scope`, `File.Scope`, `File.Unresolved`, `Importer`, `Package`, `NewPackage`. In general, identifiers cannot be accurately resolved without type information. Consider, for example, the identifier K in `T{K: ""}`: it could be the name of a local variable if T is a map type, or the name of a field if T is a struct type. New programs should use the go/types package to resolve identifiers; see `Object`, `Info.Uses`, and `Info.Defs` for details.

The new `ast.Unparen` function removes any enclosing parentheses from an expression.

go/types

The new `Alias` type represents type aliases. Previously, type aliases were not represented explicitly, so a reference to a type alias was equivalent to spelling out the aliased type, and the name of the alias was lost. The new representation retains the intermediate `Alias`. This enables improved error reporting (the name of a type alias can be reported), and allows for better handling of cyclic type declarations involving type aliases. In a future release, `Alias` types will also carry type parameter information. The new function `Unalias` returns the actual type denoted by an `Alias` type (or any other `Type` for that matter).

Because `Alias` types may break existing type switches that do not know to check for them, this functionality is controlled by a `GODEBUG` field named `gotypesalias`. With `gotypesalias=0`, everything behaves as before, and `Alias` types are never created. With `gotypesalias=1`, `Alias` types are created and clients must expect them. The default is `gotypesalias=0`. In a future release, the default will be changed to `gotypesalias=1`. *Clients of `go/types` are urged to adjust their code as soon as possible to work with `gotypesalias=1` to eliminate problems early.*

The `Info` struct now exports the `FileVersions` map which provides per-file Go version information.

The new helper method `PkgNameOf` returns the local package name for the given import declaration.

The implementation of `SizesFor` has been adjusted to compute the same type sizes as the compiler when the compiler argument for `SizesFor` is `"gc"`. The default `Sizes` implementation used by the type checker is now `types.SizesFor("gc", "amd64")`.

The start position (`Pos`) of the lexical environment block (`Scope`) that represents a function body has changed: it used to start at the opening curly brace of the function body, but now starts at the function's `func` token.

html/template

JavaScript template literals may now contain Go template actions, and parsing a template containing one will no longer return `ErrJSTemplate`. Similarly the GODEBUG setting `jstmpllitinterp` no longer has any effect.

io

The new `SectionReader.Outer` method returns the `ReaderAt`, offset, and size passed to `NewSectionReader`.

log/slog

The new `SetLogLoggerLevel` function controls the level for the bridge between the `slog` and `log` packages. It sets the minimum level for calls to the top-level `slog` logging functions, and it sets the level for calls to `log.Logger` that go through `slog`.

math/big

The new method `Rat.FloatPrec` computes the number of fractional decimal digits required to represent a rational number accurately as a floating-point number, and whether accurate decimal representation is possible in the first place.

net

When `io.Copy` copies from a `TCPConn` to a `UnixConn`, it will now use Linux's `splice(2)` system call if possible, using the new method `TCPConn.WriteTo`.

The Go DNS Resolver, used when building with "-tags=netgo", now searches for a matching name in the Windows hosts file, located at `%SystemRoot%\System32\drivers\etc\hosts`, before making a DNS query.

net/http

The new functions `ServeFileFS`, `FileServerFS`, and `NewFileTransportFS` are versions of the existing `ServeFile`, `FileServer`, and `NewFileTransport`, operating on an `fs.FS`.

The HTTP server and client now reject requests and responses containing an invalid empty `Content-Length` header. The previous behavior may be restored by setting `GODEBUG` field `httplaxcontentlength=1`.

The new method `Request.PathValue` returns path wildcard values from a request and the new method `Request.SetPathValue` sets path wildcard values on a request.

net/http/cgi

When executing a CGI process, the `PATH_INFO` variable is now always set to the empty string or a value starting with a `/` character, as required by RFC 3875. It was previously possible for some combinations of `Handler.Root` and request URL to violate this requirement.

net/netip

The new `AddrPort.Compare` method compares two `AddrPorts`.

os

On Windows, the `Stat` function now follows all reparse points that link to another named entity in the system. It was previously only following `IO_REPARSE_TAG_SYMLINK` and

`IO_REPARSE_TAG_MOUNT_POINT` reparse points.

On Windows, passing `O_SYNC` to `OpenFile` now causes write operations to go directly to disk, equivalent to `O_SYNC` on Unix platforms.

On Windows, the `ReadDir`, `File.ReadDir`, `File.Readdir`, and `File.Readdirnames` functions now read directory entries in batches to reduce the number of system calls, improving performance up to 30%.

When `io.Copy` copies from a `File` to a `net.UnixConn`, it will now use Linux's `sendfile(2)` system call if possible, using the new method `File.WriteTo`.

os/exec

On Windows, `LookPath` now ignores empty entries in `%PATH%`, and returns `ErrNotFound` (instead of `ErrNotExist`) if no executable file extension is found to resolve an otherwise-unambiguous name.

On Windows, `Command` and `Cmd.Start` no longer call `LookPath` if the path to the executable is already absolute and has an executable file extension. In addition, `Cmd.Start` no longer writes the resolved extension back to the `Path` field, so it is now safe to call the `String` method concurrently with a call to `Start`.

reflect

The `Value.IsZero` method will now return true for a floating-point or complex negative zero, and will return true for a struct value if a blank field (a field named _) somehow has a non-zero value. These changes make `IsZero` consistent with comparing a value to zero using the language == operator.

The `PtrTo` function is deprecated, in favor of `PointerTo`.

The new function `TypeFor` returns the `Type` that represents the type argument T. Previously, to get the `reflect.Type` value for a type, one had to use `reflect.TypeOf((*T)(nil)).Elem()`. This may now be written as `reflect.TypeFor[T]()`.

runtime/metrics

Four new histogram metrics `/sched/pauses/stopping/gc:seconds`, `/sched/pauses/stopping/other:seconds`, `/sched/pauses/total/gc:seconds`, and `/sched/pauses/total/other:seconds` provide additional details about stop-the-world pauses. The "stopping" metrics report the time taken from deciding to stop the world until all goroutines are stopped. The "total" metrics report the time taken from deciding to stop the world until it is started again.

The `/gc/pauses:seconds` metric is deprecated, as it is equivalent to the new `/sched/pauses/total/gc:seconds` metric.

`/sync/mutex/wait/total:seconds` now includes contention on runtime-internal locks in addition to `sync.Mutex` and `sync.RWMutex`.

runtime/pprof

Mutex profiles now scale contention by the number of goroutines blocked on the mutex. This provides a more accurate representation of the degree to which a mutex is a bottleneck in a Go program. For instance, if 100 goroutines are blocked on a mutex for 10 milliseconds, a mutex profile will now record 1 second of delay instead of 10 milliseconds of delay.

Mutex profiles also now include contention on runtime-internal locks in addition to `sync.Mutex` and `sync.RWMutex`. Contention on runtime-internal locks is always reported at `runtime._LostContendedRuntimeLock`. A future release will add complete stack traces in these cases.

CPU profiles on Darwin platforms now contain the process's memory map, enabling the disassembly view in the pprof tool.

runtime/trace

The execution tracer has been completely overhauled in this release, resolving several long-standing issues and paving the way for new use-cases for execution traces.

Execution traces now use the operating system's clock on most platforms (Windows excluded) so it is possible to correlate them with traces produced by lower-level components. Execution traces no longer depend on the reliability of the platform's clock to produce a correct trace. Execution traces are now partitioned regularly on-the-fly and as a result may be processed in a streamable way. Execution traces now contain complete durations for all system calls. Execution traces now contain information about the operating system threads that goroutines executed on. The latency impact of starting and stopping execution traces has been dramatically reduced. Execution traces may now begin or end during the garbage collection mark phase.

To allow Go developers to take advantage of these improvements, an experimental trace reading package is available at golang.org/x/exp/trace. Note that this package only works on traces produced by programs built with Go 1.22 at the moment. Please try out the package and provide feedback on the corresponding proposal issue.

If you experience any issues with the new execution tracer implementation, you may switch back to the old implementation by building your Go program with `GOEXPERIMENT=noexectracer2`. If you do, please file an issue, otherwise this option will be removed in a future release.

slices

The new function `Concat` concatenates multiple slices.

Functions that shrink the size of a slice (`Delete`, `DeleteFunc`, `Compact`, `CompactFunc`, and `Replace`) now zero the elements between the new length and the old length.

`Insert` now always panics if the argument `i` is out of range. Previously it did not panic in this situation if there were no elements to be inserted.

syscall

The `syscall` package has been frozen since Go 1.4 and was marked as deprecated in Go 1.11, causing many editors to warn about any use of the package. However, some non-deprecated functionality requires use of the `syscall` package, such as the `os/exec.Cmd.SysProcAttr` field. To avoid unnecessary complaints on such code, the `syscall` package is no longer marked as deprecated. The package remains frozen to most new functionality, and new code remains encouraged to use `golang.org/x/sys/unix` or `golang.org/x/sys/windows` where possible.

On Linux, the new `SysProcAttr.PidFD` field allows obtaining a PID FD when starting a child process via `StartProcess` or `os/exec`.

On Windows, passing `O_SYNC` to `Open` now causes write operations to go directly to disk, equivalent to `O_SYNC` on Unix platforms.

testing/slogtest

The new `Run` function uses sub-tests to run test cases, providing finer-grained control.

Ports

Darwin

On macOS on 64-bit x86 architecture (the `darwin/amd64` port), the Go toolchain now generates position-independent executables (PIE) by default. Non-PIE binaries can be generated by specifying the `-buildmode=exe` build flag. On 64-bit ARM-based macOS (the `darwin/arm64` port), the Go toolchain already generates PIE by default.

Go 1.22 is the last release that will run on macOS 10.15 Catalina. Go 1.23 will require macOS 11 Big Sur or later.

ARM

The `GOARM` environment variable now allows you to select whether to use software or hardware floating point. Previously, valid `GOARM` values were `5`, `6`, or `7`. Now those same values can be

optionally followed by `,softfloat` or `,hardfloat` to select the floating-point implementation.

This new option defaults to `softfloat` for version 5 and `hardfloat` for versions 6 and 7.

Loong64

The `loong64` port now supports passing function arguments and results using registers.

The `linux/loong64` port now supports the address sanitizer, memory sanitizer, new-style linker relocations, and the `plugin` build mode.

OpenBSD

Go 1.22 adds an experimental port to OpenBSD on big-endian 64-bit PowerPC (`openbsd/ppc64`).