Go 1.16 Release Notes

Table of Contents

Introduction to Go 1.16

The latest Go release, version 1.16, arrives six months after Go 1.15. 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

There are no changes to the language.

Ports

Darwin and iOS

Go 1.16 adds support of 64-bit ARM architecture on macOS (also known as Apple Silicon) with `GOOS=darwin`, `GOARCH=arm64`. Like the `darwin/amd64` port, the `darwin/arm64` port supports cgo, internal and external linking, `c-archive`, `c-shared`, and `pie` build modes, and the race detector.

The iOS port, which was previously `darwin/arm64`, has been renamed to `ios/arm64`. `GOOS=ios` implies the `darwin` build tag, just as `GOOS=android` implies the `linux` build tag. This change should be transparent to anyone using gomobile to build iOS apps.

The introduction of `GOOS=ios` means that file names like `x_ios.go` will now only be built for `GOOS=ios`; see `go help buildconstraint` for details. Existing packages that use file names of this form will have to rename the files.

Go 1.16 adds an `ios/amd64` port, which targets the iOS simulator running on AMD64-based macOS. Previously this was unofficially supported through `darwin/amd64` with the `ios` build tag set. See also `misc/ios/README` for details about how to build programs for iOS and iOS simulator.

Go 1.16 is the last release that will run on macOS 10.12 Sierra. Go 1.17 will require macOS 10.13 High Sierra or later.

NetBSD

Go now supports the 64-bit ARM architecture on NetBSD (the `netbsd/arm64` port).

OpenBSD

Go now supports the MIPS64 architecture on OpenBSD (the `openbsd/mips64` port). This port does not yet support cgo.

On the 64-bit x86 and 64-bit ARM architectures on OpenBSD (the `openbsd/amd64` and `openbsd/arm64` ports), system calls are now made through `libc`, instead of directly using the `SYSCALL`/`SVC` instruction. This ensures forward-compatibility with future versions of OpenBSD. In particular, OpenBSD 6.9 onwards will require system calls to be made through `libc` for non-static Go binaries.

386

As announced in the Go 1.15 release notes, Go 1.16 drops support for x87 mode compilation (`GO386=387`). Support for non-SSE2 processors is now available using soft float mode (`GO386=softfloat`). Users running on non-SSE2 processors should replace `GO386=387` with `GO386=softfloat`.

RISC-V

The `linux/riscv64` port now supports cgo and `-buildmode=pie`. This release also includes performance optimizations and code generation improvements for RISC-V.

Tools

Go command

Modules

Module-aware mode is enabled by default, regardless of whether a `go.mod` file is present in the current working directory or a parent directory. More precisely, the `GO111MODULE` environment variable now defaults to `on`. To switch to the previous behavior, set `GO111MODULE` to `auto`.

Build commands like `go build` and `go test` no longer modify `go.mod` and `go.sum` by default. Instead, they report an error if a module requirement or checksum needs to be added or updated (as if the `-mod=readonly` flag were used). Module requirements and sums may be adjusted with `go mod tidy` or `go get`.

`go install` now accepts arguments with version suffixes (for example, `go install example.com/cmd@v1.0.0`). This causes `go install` to build and install packages in module-aware mode, ignoring the `go.mod` file in the current directory or any parent directory, if there is one. This is useful for installing executables without affecting the dependencies of the main module.

`go install`, with or without a version suffix (as described above), is now the recommended way to build and install packages in module mode. `go get` should be used with the `-d` flag to adjust the current module's dependencies without building packages, and use of `go get` to build and install packages is deprecated. In a future release, the `-d` flag will always be enabled.

`retract` directives may now be used in a `go.mod` file to indicate that certain published versions of the module should not be used by other modules. A module author may retract a version after a severe problem is discovered or if the version was published unintentionally.

The `go mod vendor` and `go mod tidy` subcommands now accept the `-e` flag, which instructs them to proceed despite errors in resolving missing packages.

The `go` command now ignores requirements on module versions excluded by `exclude` directives in the main module. Previously, the `go` command used the next version higher than an excluded version, but that version could change over time, resulting in non-reproducible builds.

In module mode, the `go` command now disallows import paths that include non-ASCII characters or path elements with a leading dot character (`.`). Module paths with these characters were already disallowed (see Module paths and versions), so this change affects only paths within module subdirectories.

Embedding Files

The `go` command now supports including static files and file trees as part of the final executable, using the new `//go:embed` directive. See the documentation for the new `embed` package for details.

`go test`

When using `go test`, a test that calls `os.Exit(0)` during execution of a test function will now be considered to fail. This will help catch cases in which a test calls code that calls `os.Exit(0)` and thereby stops running all future tests. If a `TestMain` function calls `os.Exit(0)` that is still considered to be a passing test.

`go test` reports an error when the `-c` or `-i` flags are used together with unknown flags. Normally, unknown flags are passed to tests, but when `-c` or `-i` are used, tests are not run.

`go get`

The `go get -insecure` flag is deprecated and will be removed in a future version. This flag permits fetching from repositories and resolving custom domains using insecure schemes such as HTTP, and also bypasses module sum validation using the checksum database. To permit the use of insecure schemes, use the `GOINSECURE` environment variable instead. To bypass module sum validation, use `GOPRIVATE` or `GONOSUMDB`. See `go help environment` for details.

`go get example.com/mod@patch` now requires that some version of `example.com/mod` already be required by the main module. (However, `go get -u=patch` continues to patch even newly-added dependencies.)

`GOVCS` environment variable

`GOVCS` is a new environment variable that limits which version control tools the `go` command may use to download source code. This mitigates security issues with tools that are typically used in trusted, authenticated environments. By default, `git` and `hg` may be used to download code from any repository. `svn`, `bzr`, and `fossil` may only be used to download code from repositories with module paths or package paths matching patterns in the `GOPRIVATE` environment variable. See `go help vcs` for details.

The `all` pattern

When the main module's `go.mod` file declares `go 1.16` or higher, the `all` package pattern now matches only those packages that are transitively imported by a package or test found in the main module. (Packages imported by *tests of* packages imported by the main module are no longer included.) This is the same set of packages retained by `go mod vendor` since Go 1.11.

The `-toolexec` build flag

When the `-toolexec` build flag is specified to use a program when invoking toolchain programs like compile or asm, the environment variable `TOOLEXEC_IMPORTPATH` is now set to the import path of the package being built.

The `-i` build flag

The `-i` flag accepted by `go build`, `go install`, and `go test` is now deprecated. The `-i` flag instructs the `go` command to install packages imported by packages named on the command line. Since the build cache was introduced in Go 1.10, the `-i` flag no longer has a significant effect on build times, and it causes errors when the install directory is not writable.

The `list` command

When the `-export` flag is specified, the `BuildID` field is now set to the build ID of the compiled package. This is equivalent to running `go tool buildid` on `go list -exported -f {{.Export}}`, but without the extra step.

The `-overlay` flag

The `-overlay` flag specifies a JSON configuration file containing a set of file path replacements. The `-overlay` flag may be used with all build commands and `go mod` subcommands. It is primarily intended to be used by editor tooling such as gopls to understand the effects of unsaved changes to source files. The config file maps actual file paths to replacement file paths and the `go` command and its builds will run as if the actual file paths exist with the contents given by the replacement file paths, or don't exist if the replacement file paths are empty.

Cgo

The cgo tool will no longer try to translate C struct bitfields into Go struct fields, even if their size can be represented in Go. The order in which C bitfields appear in memory is implementation dependent, so in some cases the cgo tool produced results that were silently incorrect.

Vet

New warning for invalid testing.T use in goroutines

The vet tool now warns about invalid calls to the `testing.T` method `Fatal` from within a goroutine created during the test. This also warns on calls to `Fatalf`, `FailNow`, and `Skip{,f,Now}` methods on `testing.T` tests or `testing.B` benchmarks.

Calls to these methods stop the execution of the created goroutine and not the `Test*` or `Benchmark*` function. So these are required to be called by the goroutine running the test or benchmark function. For example:

```
func TestFoo(t *testing.T) {
    go func() {
        if condition() {
            t.Fatal("oops") // This exits the inner func instead of TestFoo.
        }
        ...
    }()
}
```

Code calling `t.Fatal` (or a similar method) from a created goroutine should be rewritten to signal the test failure using `t.Error` and exit the goroutine early using an alternative method, such as using a `return` statement. The previous example could be rewritten as:

```
func TestFoo(t *testing.T) {
    go func() {
        if condition() {
            t.Error("oops")
            return
        }
        ...
    }()
}
```

New warning for frame pointer

The vet tool now warns about amd64 assembly that clobbers the BP register (the frame pointer) without saving and restoring it, contrary to the calling convention. Code that doesn't preserve the BP register must be modified to either not use BP at all or preserve BP by saving and restoring it. An easy way to preserve BP is to set the frame size to a nonzero value, which causes the generated prologue and epilogue to preserve the BP register for you. See CL 248260 for example fixes.

New warning for asn1.Unmarshal

The vet tool now warns about incorrectly passing a non-pointer or nil argument to `asn1.Unmarshal`. This is like the existing checks for `encoding/json.Unmarshal` and `encoding/xml.Unmarshal`.

Runtime

The new `runtime/metrics` package introduces a stable interface for reading implementation-defined metrics from the Go runtime. It supersedes existing functions like `runtime.ReadMemStats` and `debug.GCStats` and is significantly more general and efficient. See the package documentation for more details.

Setting the `GODEBUG` environment variable to `inittrace=1` now causes the runtime to emit a single line to standard error for each package `init`, summarizing its execution time and memory allocation. This trace can be used to find bottlenecks or regressions in Go startup performance. The `GODEBUG` documentation describes the format.

On Linux, the runtime now defaults to releasing memory to the operating system promptly (using `MADV_DONTNEED`), rather than lazily when the operating system is under memory pressure (using `MADV_FREE`). This means process-level memory statistics like RSS will more accurately reflect the amount of physical memory being used by Go processes. Systems that are currently using `GODEBUG=madvdontneed=1` to improve memory monitoring behavior no longer need to set this environment variable.

Go 1.16 fixes a discrepancy between the race detector and the Go memory model. The race detector now more precisely follows the channel synchronization rules of the memory model. As

a result, the detector may now report races it previously missed.

Compiler

The compiler can now inline functions with non-labeled `for` loops, method values, and type switches. The inliner can also detect more indirect calls where inlining is possible.

Linker

This release includes additional improvements to the Go linker, reducing linker resource usage (both time and memory) and improving code robustness/maintainability. These changes form the second half of a two-release project to modernize the Go linker.

The linker changes in 1.16 extend the 1.15 improvements to all supported architecture/OS combinations (the 1.15 performance improvements were primarily focused on ELF-based OSes and `amd64` architectures). For a representative set of large Go programs, linking is 20-25% faster than 1.15 and requires 5-15% less memory on average for `linux/amd64`, with larger improvements for other architectures and OSes. Most binaries are also smaller as a result of more aggressive symbol pruning.

On Windows, `go build -buildmode=c-shared` now generates Windows ASLR DLLs by default. ASLR can be disabled with `--ldflags=-aslr=false`.

Standard library

Embedded Files

The new `embed` package provides access to files embedded in the program during compilation using the new `//go:embed` directive.

File Systems

The new `io/fs` package defines the `fs.FS` interface, an abstraction for read-only trees of files. The standard library packages have been adapted to make use of the interface as appropriate.

On the producer side of the interface, the new `embed.FS` type implements `fs.FS`, as does `zip.Reader`. The new `os.DirFS` function provides an implementation of `fs.FS` backed by a tree of operating system files.

On the consumer side, the new `http.FS` function converts an `fs.FS` to an `http.FileSystem`. Also, the `html/template` and `text/template` packages' `ParseFS` functions and methods read templates from an `fs.FS`.

For testing code that implements `fs.FS`, the new `testing/fstest` package provides a `TestFS` function that checks for and reports common mistakes. It also provides a simple in-memory file system implementation, `MapFS`, which can be useful for testing code that accepts `fs.FS` implementations.

Deprecation of io/ioutil

The `io/ioutil` package has turned out to be a poorly defined and hard to understand collection of things. All functionality provided by the package has been moved to other packages. The `io/ioutil` package remains and will continue to work as before, but we encourage new code to use the new definitions in the `io` and `os` packages. Here is a list of the new locations of the names exported by `io/ioutil`:

- `Discard` => `io.Discard`
- `NopCloser` => `io.NopCloser`
- `ReadAll` => `io.ReadAll`
- `ReadDir` => `os.ReadDir` (note: returns a slice of `os.DirEntry` rather than a slice of `fs.FileInfo`)
- `ReadFile` => `os.ReadFile`
- `TempDir` => `os.MkdirTemp`
- `TempFile` => `os.CreateTemp`
- `WriteFile` => `os.WriteFile`

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.

archive/zip

The new `Reader.Open` method implements the `fs.FS` interface.

crypto/dsa

The `crypto/dsa` package is now deprecated. See issue #40337.

crypto/hmac

New will now panic if separate calls to the hash generation function fail to return new values. Previously, the behavior was undefined and invalid outputs were sometimes generated.

crypto/tls

I/O operations on closing or closed TLS connections can now be detected using the new `net.ErrClosed` error. A typical use would be `errors.Is(err, net.ErrClosed)`.

A default write deadline is now set in `Conn.Close` before sending the "close notify" alert, in order to prevent blocking indefinitely.

Clients now return a handshake error if the server selects an ALPN protocol that was not in the list advertised by the client.

Servers will now prefer other available AEAD cipher suites (such as ChaCha20Poly1305) over AES-GCM cipher suites if either the client or server doesn't have AES hardware support, unless both `Config.PreferServerCipherSuites` and `Config.CipherSuites` are set. The client is assumed not to have AES hardware support if it does not signal a preference for AES-GCM cipher suites.

`Config.Clone` now returns nil if the receiver is nil, rather than panicking.

crypto/x509

The `GODEBUG=x509ignoreCN=0` flag will be removed in Go 1.17. It enables the legacy behavior of treating the `CommonName` field on X.509 certificates as a host name when no Subject Alternative Names are present.

`ParseCertificate` and `CreateCertificate` now enforce string encoding restrictions for the `DNSNames`, `EmailAddresses`, and `URIs` fields. These fields can only contain strings with characters within the ASCII range.

`CreateCertificate` now verifies the generated certificate's signature using the signer's public key. If the signature is invalid, an error is returned, instead of a malformed certificate.

DSA signature verification is no longer supported. Note that DSA signature generation was never supported. See issue #40337.

On Windows, `Certificate.Verify` will now return all certificate chains that are built by the platform certificate verifier, instead of just the highest ranked chain.

The new `SystemRootsError.Unwrap` method allows accessing the `Err` field through the `errors` package functions.

On Unix systems, the `crypto/x509` package is now more efficient in how it stores its copy of the system cert pool. Programs that use only a small number of roots will use around a half megabyte less memory.

debug/elf

More `DT` and `PT` constants have been added.

encoding/asn1

`Unmarshal` and `UnmarshalWithParams` now return an error instead of panicking when the argument is not a pointer or is nil. This change matches the behavior of other encoding packages such as `encoding/json`.

encoding/json

The `json` struct field tags understood by `Marshal`, `Unmarshal`, and related functionality now permit semicolon characters within a JSON object name for a Go struct field.

encoding/xml

The encoder has always taken care to avoid using namespace prefixes beginning with `xml`, which are reserved by the XML specification. Now, following the specification more closely, that check is case-insensitive, so that prefixes beginning with `XML`, `XmL`, and so on are also avoided.

flag

The new `Func` function allows registering a flag implemented by calling a function, as a lighter-weight alternative to implementing the `Value` interface.

go/build

The `Package` struct has new fields that report information about `//go:embed` directives in the package: `EmbedPatterns`, `EmbedPatternPos`, `TestEmbedPatterns`, `TestEmbedPatternPos`, `XTestEmbedPatterns`, `XTestEmbedPatternPos`.

The `Package` field `IgnoredGoFiles` will no longer include files that start with "_" or ".", as those files are always ignored. `IgnoredGoFiles` is for files ignored because of build constraints.

The new `Package` field `IgnoredOtherFiles` has a list of non-Go files ignored because of build constraints.

go/build/constraint

The new `go/build/constraint` package parses build constraint lines, both the original `//+build` syntax and the `//go:build` syntax that will be introduced in Go 1.17. This package exists so that tools built with Go 1.16 will be able to process Go 1.17 source code. See https://golang.org/design/draft-gobuild for details about the build constraint syntaxes and the planned transition to the `//go:build` syntax. Note that `//go:build` lines are **not** supported in Go 1.16 and should not be introduced into Go programs yet.

html/template

The new `template.ParseFS` function and `template.Template.ParseFS` method are like `template.ParseGlob` and `template.Template.ParseGlob`, but read the templates from an `fs.FS`.

io

The package now defines a `ReadSeekCloser` interface.

The package now defines `Discard`, `NopCloser`, and `ReadAll`, to be used instead of the same names in the `io/ioutil` package.

log

The new `Default` function provides access to the default `Logger`.

log/syslog

The `Writer` now uses the local message format (omitting the host name and using a shorter time stamp) when logging to custom Unix domain sockets, matching the format already used for the default log socket.

mime/multipart

The `Reader`'s `ReadForm` method no longer rejects form data when passed the maximum int64 value as a limit.

net

The case of I/O on a closed network connection, or I/O on a network connection that is closed before any of the I/O completes, can now be detected using the new `ErrClosed` error. A typical use would be `errors.Is(err, net.ErrClosed)`. In earlier releases the only way to reliably detect this case was to match the string returned by the `Error` method with `"use of closed network connection"`.

In previous Go releases the default TCP listener backlog size on Linux systems, set by `/proc/sys/net/core/somaxconn`, was limited to a maximum of 65535. On Linux kernel version 4.1 and above, the maximum is now 4294967295.

On Linux, host name lookups no longer use DNS before checking `/etc/hosts` when `/etc/nsswitch.conf` is missing; this is common on musl-based systems and makes Go programs match the behavior of C programs on those systems.

net/http

In the `net/http` package, the behavior of `StripPrefix` has been changed to strip the prefix from the request URL's `RawPath` field in addition to its `Path` field. In past releases, only the `Path` field was trimmed, and so if the request URL contained any escaped characters the URL

would be modified to have mismatched `Path` and `RawPath` fields. In Go 1.16, `StripPrefix` trims both fields. If there are escaped characters in the prefix part of the request URL the handler serves a 404 instead of its previous behavior of invoking the underlying handler with a mismatched `Path`/`RawPath` pair.

The `net/http` package now rejects HTTP range requests of the form `"Range": "bytes=--N"` where `"-N"` is a negative suffix length, for example `"Range": "bytes=--2"`. It now replies with a `416 "Range Not Satisfiable"` response.

Cookies set with `SameSiteDefaultMode` now behave according to the current spec (no attribute is set) instead of generating a SameSite key without a value.

The `Client` now sends an explicit `Content-Length: 0` header in `PATCH` requests with empty bodies, matching the existing behavior of `POST` and `PUT`.

The `ProxyFromEnvironment` function no longer returns the setting of the `HTTP_PROXY` environment variable for `https://` URLs when `HTTPS_PROXY` is unset.

The `Transport` type has a new field `GetProxyConnectHeader` which may be set to a function that returns headers to send to a proxy during a `CONNECT` request. In effect `GetProxyConnectHeader` is a dynamic version of the existing field `ProxyConnectHeader`; if `GetProxyConnectHeader` is not `nil`, then `ProxyConnectHeader` is ignored.

The new `http.FS` function converts an `fs.FS` to an `http.FileSystem`.

net/http/httputil

`ReverseProxy` now flushes buffered data more aggressively when proxying streamed responses with unknown body lengths.

net/smtp

The `Client`'s `Mail` method now sends the `SMTPUTF8` directive to servers that support it, signaling that addresses are encoded in UTF-8.

os

`Process.Signal` now returns `ErrProcessDone` instead of the unexported `errFinished` when the process has already finished.

The package defines a new type `DirEntry` as an alias for `fs.DirEntry`. The new `ReadDir` function and the new `File.ReadDir` method can be used to read the contents of a directory into a slice of `DirEntry`. The `File.Readdir` method (note the lower case `d` in `dir`) still exists, returning a slice of `FileInfo`, but for most programs it will be more efficient to switch to `File.ReadDir`.

The package now defines `CreateTemp`, `MkdirTemp`, `ReadFile`, and `WriteFile`, to be used instead of functions defined in the `io/ioutil` package.

The types `FileInfo`, `FileMode`, and `PathError` are now aliases for types of the same name in the `io/fs` package. Function signatures in the `os` package have been updated to refer to the names in the `io/fs` package. This should not affect any existing code.

The new `DirFS` function provides an implementation of `fs.FS` backed by a tree of operating system files.

os/signal

The new `NotifyContext` function allows creating contexts that are canceled upon arrival of specific signals.

path

The `Match` function now returns an error if the unmatched part of the pattern has a syntax error. Previously, the function returned early on a failed match, and thus did not report any later syntax error in the pattern.

path/filepath

The new function `WalkDir` is similar to `Walk`, but is typically more efficient. The function passed to `WalkDir` receives a `fs.DirEntry` instead of a `fs.FileInfo`. (To clarify for those who recall the `Walk` function as taking an `os.FileInfo`, `os.FileInfo` is now an alias for `fs.FileInfo`.)

The `Match` and `Glob` functions now return an error if the unmatched part of the pattern has a syntax error. Previously, the functions returned early on a failed match, and thus did not report any later syntax error in the pattern.

reflect

The Zero function has been optimized to avoid allocations. Code which incorrectly compares the returned Value to another Value using == or DeepEqual may get different results than those obtained in previous Go versions. The documentation for `reflect.Value` describes how to compare two `Values` correctly.

runtime/debug

The `runtime.Error` values used when `SetPanicOnFault` is enabled may now have an `Addr` method. If that method exists, it returns the memory address that triggered the fault.

strconv

`ParseFloat` now uses the Eisel-Lemire algorithm, improving performance by up to a factor of 2. This can also speed up decoding textual formats like `encoding/json`.

syscall

`NewCallback` and `NewCallbackCDecl` now correctly support callback functions with multiple sub-`uintptr`-sized arguments in a row. This may require changing uses of these functions to eliminate manual padding between small arguments.

`SysProcAttr` on Windows has a new `NoInheritHandles` field that disables inheriting handles when creating a new process.

`DLLError` on Windows now has an `Unwrap` method for unwrapping its underlying error.

On Linux, `Setgid`, `Setuid`, and related calls are now implemented. Previously, they returned an `syscall.EOPNOTSUPP` error.

On Linux, the new functions `AllThreadsSyscall` and `AllThreadsSyscall6` may be used to make a system call on all Go threads in the process. These functions may only be used by programs that do not use cgo; if a program uses cgo, they will always return `syscall.ENOTSUP`.

testing/iotest

The new `ErrReader` function returns an `io.Reader` that always returns an error.

The new `TestReader` function tests that an `io.Reader` behaves correctly.

text/template

Newlines characters are now allowed inside action delimiters, permitting actions to span multiple lines.

The new `template.ParseFS` function and `template.Template.ParseFS` method are like `template.ParseGlob` and `template.Template.ParseGlob`, but read the templates from an `fs.FS`.

text/template/parse

A new `CommentNode` was added to the parse tree. The `Mode` field in the `parse.Tree` enables access to it.

time/tzdata

The slim timezone data format is now used for the timezone database in `$GOROOT/lib/time/zoneinfo.zip` and the embedded copy in this package. This reduces the size of the timezone database by about 350 KB.

unicode

The `unicode` package and associated support throughout the system has been upgraded from Unicode 12.0.0 to Unicode 13.0.0, which adds 5,930 new characters, including four new scripts, and 55 new emoji. Unicode 13.0.0 also designates plane 3 (U+30000-U+3FFFF) as the tertiary ideographic plane.