Go 1.19 Release Notes

- Introduction to Go 1.19 {#introduction}
- Changes to the language {#language}
- Memory Model {#mem}
- Ports {#ports}
  - LoongArch 64-bit {#loong64}
  - RISC-V {#riscv64}
- Tools {#tools}
  - Doc Comments {#go-doc}
  - New unix build constraint {#go-unix}
  - Go command {#go-command}
  - Vet {#vet}
- Runtime {#runtime}
- Compiler {#compiler}
- Assembler {#assembler}
- Linker {#linker}
- Standard library {#library}
  - New atomic types {#atomic_types}
  - PATH lookups {#os-exec-path}
  - Minor changes to the library {#minor_library_changes}
    - archive/zip
    - crypto/elliptic
    - crypto/rand
    - crypto/tls
    - crypto/x509
    - crypto/x509/pkix
    - debug/elf
    - debug/pe
    - encoding/binary

- encoding/csv
- encoding/xml
- flag
- fmt
- go/parser
- go/types
- hash/maphash
- html/template
- image/draw
- io
- mime
- mime/multipart
- net
- net/http
- net/url
- os/exec
- reflect
- regexp/syntax
- runtime
- runtime/metrics
- runtime/pprof
- runtime/race
- runtime/trace
- sort
- strconv
- syscall
- time

Introduction to Go 1.19 {#introduction}

The latest Go release, version 1.19, arrives five months after Go 1.18. 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 is only one small change to the language, a very small correction to the scope of type parameters in method declarations. Existing programs are unaffected.

Memory Model {#mem}

The Go memory model has been revised to align Go with the memory model used by C, C++, Java, JavaScript, Rust, and Swift. Go only provides sequentially consistent atomics, not any of the more relaxed forms found in other languages. Along with the memory model update, Go 1.19 introduces new types in the `sync/atomic` package that make it easier to use atomic values, such as atomic.Int64 and atomic.Pointer[T].

Ports {#ports}

LoongArch 64-bit {#loong64}

Go 1.19 adds support for the Loongson 64-bit architecture LoongArch on Linux (`GOOS=linux`, `GOARCH=loong64`). The implemented ABI is LP64D. Minimum kernel version supported is 5.19.

Note that most existing commercial Linux distributions for LoongArch come with older kernels, with a historical incompatible system call ABI. Compiled binaries will not work on these systems, even if statically linked. Users on such unsupported systems are limited to the distribution-provided Go package.

RISC-V {#riscv64}

The `riscv64` port now supports passing function arguments and result using registers. Benchmarking shows typical performance improvements of 10% or more on `riscv64`.

Tools {#tools}

Doc Comments {#go-doc}

Go 1.19 adds support for links, lists, and clearer headings in doc comments. As part of this change, `gofmt` now reformats doc comments to make their rendered meaning clearer. See "Go Doc Comments" for syntax details and descriptions of common mistakes now highlighted by `gofmt`. As another part of this change, the new package go/doc/comment provides parsing and reformatting of doc comments as well as support for rendering them to HTML, Markdown, and text.

New `unix` build constraint {#go-unix}

The build constraint `unix` is now recognized in `//go:build` lines. The constraint is satisfied if the target operating system, also known as `GOOS`, is a Unix or Unix-like system. For the 1.19 release it is satisfied if `GOOS` is one of `aix`, `android`, `darwin`, `dragonfly`, `freebsd`, `hurd`, `illumos`, `ios`, `linux`, `netbsd`, `openbsd`, or `solaris`. In future releases the `unix` constraint may match additional newly supported operating systems.

Go command {#go-command}

The `-trimpath` flag, if set, is now included in the build settings stamped into Go binaries by `go build`, and can be examined using `go version -m` or `debug.ReadBuildInfo`.

`go generate` now sets the `GOROOT` environment variable explicitly in the generator's environment, so that generators can locate the correct `GOROOT` even if built with `-trimpath`.

`go test` and `go generate` now place `GOROOT/bin` at the beginning of the `PATH` used for the subprocess, so tests and generators that execute the `go` command will resolve it to same `GOROOT`.

`go env` now quotes entries that contain spaces in the `CGO_CFLAGS`, `CGO_CPPFLAGS`, `CGO_CXXFLAGS`, `CGO_FFLAGS`, `CGO_LDFLAGS`, and `GOGCCFLAGS` variables it reports.

`go list -json` now accepts a comma-separated list of JSON fields to populate. If a list is specified, the JSON output will include only those fields, and `go list` may avoid work to compute fields that are not included. In some cases, this may suppress errors that would otherwise be reported.

The `go` command now caches information necessary to load some modules, which should result in a speed-up of some `go list` invocations.

Vet {#vet}

The `vet` checker “errorsas” now reports when `errors.As` is called with a second argument of type `*error`, a common mistake.

Runtime {#runtime}

The runtime now includes support for a soft memory limit. This memory limit includes the Go heap and all other memory managed by the runtime, and excludes external memory sources such as mappings of the binary itself, memory managed in other languages, and memory held by the operating system on behalf of the Go program. This limit may be managed via `runtime/debug.SetMemoryLimit` or the equivalent `GOMEMLIMIT` environment variable. The limit works in conjunction with `runtime/debug.SetGCPercent` / `GOGC`, and will be respected even if `GOGC=off`, allowing Go programs to always make maximal use of their memory limit, improving resource efficiency in some cases. See the GC guide for a detailed guide explaining the soft memory limit in more detail, as well as a variety of common use-cases and scenarios. Please note that small memory limits, on the order of tens of megabytes or less, are less likely to be respected due to external latency factors, such as OS scheduling. See issue 52433 for more details. Larger memory limits, on the order of hundreds of megabytes or more, are stable and production-ready.

In order to limit the effects of GC thrashing when the program's live heap size approaches the soft memory limit, the Go runtime also attempts to limit total GC CPU utilization to 50%, excluding idle time, choosing to use more memory over preventing application progress. In practice, we expect this limit to only play a role in exceptional cases, and the new runtime metric `/gc/limiter/last-enabled:gc-cycle` reports when this last occurred.

The runtime now schedules many fewer GC worker goroutines on idle operating system threads when the application is idle enough to force a periodic GC cycle.

The runtime will now allocate initial goroutine stacks based on the historic average stack usage of goroutines. This avoids some of the early stack growth and copying needed in the average case in exchange for at most 2x wasted space on below-average goroutines.

On Unix operating systems, Go programs that import package os now automatically increase the open file limit (`RLIMIT_NOFILE`) to the maximum allowed value; that is, they change the soft limit to match the hard limit. This corrects artificially low limits set on some systems for compatibility with very old C programs using the *select* system call. Go programs are not helped by that limit, and instead even simple programs like `gofmt` often ran out of file descriptors on such systems when

processing many files in parallel. One impact of this change is that Go programs that in turn execute very old C programs in child processes may run those programs with too high a limit. This can be corrected by setting the hard limit before invoking the Go program.

Unrecoverable fatal errors (such as concurrent map writes, or unlock of unlocked mutexes) now print a simpler traceback excluding runtime metadata (equivalent to a fatal panic) unless `GOTRACEBACK=system` or `crash`. Runtime-internal fatal error tracebacks always include full metadata regardless of the value of `GOTRACEBACK`

Support for debugger-injected function calls has been added on ARM64, enabling users to call functions from their binary in an interactive debugging session when using a debugger that is updated to make use of this functionality.

The address sanitizer support added in Go 1.18 now handles function arguments and global variables more precisely.

Compiler {#compiler}

The compiler now uses a jump table to implement large integer and string switch statements. Performance improvements for the switch statement vary but can be on the order of 20% faster. (`GOARCH=amd64` and `GOARCH=arm64` only)

The Go compiler now requires the `-p=importpath` flag to build a linkable object file. This is already supplied by the `go` command and by Bazel. Any other build systems that invoke the Go compiler directly will need to make sure they pass this flag as well.

The Go compiler no longer accepts the `-importmap` flag. Build systems that invoke the Go compiler directly must use the `-importcfg` flag instead.

Assembler {#assembler}

Like the compiler, the assembler now requires the `-p=importpath` flag to build a linkable object file. This is already supplied by the `go` command. Any other build systems that invoke the Go assembler directly will need to make sure they pass this flag as well.

Linker {#linker}

On ELF platforms, the linker now emits compressed DWARF sections in the standard gABI format (`SHF_COMPRESSED`), instead of the legacy `.zdebug` format.

Standard library {#library}

New atomic types {#atomic_types}

The `sync/atomic` package defines new atomic types `Bool`, `Int32`, `Int64`, `Uint32`, `Uint64`, `Uintptr`, and `Pointer`. These types hide the underlying values so that all accesses are forced to use the atomic APIs. `Pointer` also avoids the need to convert to `unsafe.Pointer` at call sites. `Int64` and `Uint64` are automatically aligned to 64-bit boundaries in structs and allocated data, even on 32-bit systems.

PATH lookups {#os-exec-path}

`Command` and `LookPath` no longer allow results from a PATH search to be found relative to the current directory. This removes a common source of security problems but may also break existing programs that depend on using, say, `exec.Command("prog")` to run a binary named `prog` (or, on Windows, `prog.exe`) in the current directory. See the `os/exec` package documentation for information about how best to update such programs.

On Windows, `Command` and `LookPath` now respect the `NoDefaultCurrentDirectoryInExePath` environment variable, making it possible to disable the default implicit search of "`.`" in PATH lookups on Windows systems.

Minor changes to the library {#minor_library_changes}

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/zip

`Reader` now ignores non-ZIP data at the start of a ZIP file, matching most other implementations. This is necessary to read some Java JAR files, among other uses.

crypto/elliptic

Operating on invalid curve points (those for which the `IsOnCurve` method returns false, and which are never returned by `Unmarshal` or by a `Curve` method operating on a valid point) has always been undefined behavior and can lead to key recovery attacks. If an invalid point is supplied to `Marshal`, `MarshalCompressed`, `Add`, `Double`, or `ScalarMult`, they will now panic.

`ScalarBaseMult` operations on the `P224`, `P384`, and `P521` curves are now up to three times faster, leading to similar speedups in some ECDSA operations. The generic (not platform optimized) `P256` implementation was replaced with one derived from a formally verified model; this might lead to significant slowdowns on 32-bit platforms.

crypto/rand

`Read` no longer buffers random data obtained from the operating system between calls. Applications that perform many small reads at high frequency might choose to wrap `Reader` in a `bufio.Reader` for performance reasons, taking care to use `io.ReadFull` to ensure no partial reads occur.

On Plan 9, `Read` has been reimplemented, replacing the ANSI X9.31 algorithm with a fast key erasure generator.

The `Prime` implementation was changed to use only rejection sampling, which removes a bias when generating small primes in non-cryptographic contexts, removes one possible minor timing leak, and better aligns the behavior with BoringSSL, all while simplifying the implementation. The change does produce different outputs for a given random source stream compared to the previous implementation, which can break tests written expecting specific results from specific deterministic random sources. To help prevent such problems in the future, the implementation is now intentionally non-deterministic with respect to the input stream.

crypto/tls

The `GODEBUG` option `tls10default=1` has been removed. It is still possible to enable TLS 1.0 client-side by setting `Config.MinVersion`.

The TLS server and client now reject duplicate extensions in TLS handshakes, as required by RFC 5246, Section 7.4.1.4 and RFC 8446, Section 4.2.

crypto/x509

`CreateCertificate` no longer supports creating certificates with `SignatureAlgorithm` set to `MD5WithRSA`.

`CreateCertificate` no longer accepts negative serial numbers.

`CreateCertificate` will not emit an empty SEQUENCE anymore when the produced certificate has no extensions.

Removal of the `GODEBUG` option `x509sha1=1`, originally planned for Go 1.19, has been rescheduled to a future release. Applications using it should work on migrating. Practical attacks against SHA-1 have been demonstrated since 2017 and publicly trusted Certificate Authorities have not issued SHA-1 certificates since 2015.

`ParseCertificate` and `ParseCertificateRequest` now reject certificates and CSRs which contain duplicate extensions.

The new `CertPool.Clone` and `CertPool.Equal` methods allow cloning a `CertPool` and checking the equivalence of two `CertPool`s respectively.

The new function `ParseRevocationList` provides a faster, safer to use CRL parser which returns a `RevocationList`. Parsing a CRL also populates the new `RevocationList` fields `RawIssuer`, `Signature`, `AuthorityKeyId`, and `Extensions`, which are ignored by `CreateRevocationList`.

The new method `RevocationList.CheckSignatureFrom` checks that the signature on a CRL is a valid signature from a `Certificate`.

The `ParseCRL` and `ParseDERCRL` functions are now deprecated in favor of `ParseRevocationList`. The `Certificate.CheckCRLSignature` method is deprecated in favor of `RevocationList.CheckSignatureFrom`.

The path builder of `Certificate.Verify` was overhauled and should now produce better chains and/or be more efficient in complicated scenarios. Name constraints are now also enforced on non-leaf certificates.

crypto/x509/pkix

The types `CertificateList` and `TBSCertificateList` have been deprecated. The new `crypto/x509` CRL functionality should be used instead.

debug/elf

The new `EM_LOONGARCH` and `R_LARCH_*` constants support the loong64 port.

debug/pe

The new `File.COFFSymbolReadSectionDefAux` method, which returns a `COFFSymbolAuxFormat5`, provides access to COMDAT information in PE file sections. These are supported by new `IMAGE_COMDAT_*` and `IMAGE_SCN_*` constants.

encoding/binary

The new interface `AppendByteOrder` provides efficient methods for appending a `uint16`, `uint32`, or `uint64` to a byte slice. `BigEndian` and `LittleEndian` now implement this interface.

Similarly, the new functions `AppendUvarint` and `AppendVarint` are efficient appending versions of `PutUvarint` and `PutVarint`.

encoding/csv

The new method `Reader.InputOffset` reports the reader's current input position as a byte offset, analogous to `encoding/json`'s `Decoder.InputOffset`.

encoding/xml

The new method `Decoder.InputPos` reports the reader's current input position as a line and column, analogous to `encoding/csv`'s `Decoder.FieldPos`.

flag

The new function `TextVar` defines a flag with a value implementing `encoding.TextUnmarshaler`, allowing command-line flag variables to have types such as `big.Int`, `netip.Addr`, and `time.Time`.

fmt

The new functions `Append`, `Appendf`, and `Appendln` append formatted data to byte slices.

go/parser

The parser now recognizes `~x` as a unary expression with operator token.TILDE, allowing better error recovery when a type constraint such as `~int` is used in an incorrect context.

go/types

The new methods `Func.Origin` and `Var.Origin` return the corresponding `Object` of the generic type for synthetic `Func` and `Var` objects created during type instantiation.

It is no longer possible to produce an infinite number of distinct-but-identical `Named` type instantiations via recursive calls to `Named.Underlying` or `Named.Method`.

hash/maphash

The new functions `Bytes` and `String` provide an efficient way hash a single byte slice or string. They are equivalent to using the more general `Hash` with a single write, but they avoid setup overhead for small inputs.

html/template

The type `FuncMap` is now an alias for `text/template`'s `FuncMap` instead of its own named type. This allows writing code that operates on a `FuncMap` from either setting.

Go 1.19.8 and later disallow actions in ECMAScript 6 template literals. This behavior can be reverted by the `GODEBUG=jstmpllitinterp=1` setting.

image/draw

`Draw` with the `Src` operator preserves non-premultiplied-alpha colors when destination and source images are both `image.NRGBA` or both `image.NRGBA64`. This reverts a behavior change accidentally introduced by a Go 1.18 library optimization; the code now matches the behavior in Go 1.17 and earlier.

io

`NopCloser`'s result now implements `WriterTo` whenever its input does.

`MultiReader`'s result now implements `WriterTo` unconditionally. If any underlying reader does not implement `WriterTo`, it is simulated appropriately.

mime

On Windows only, the mime package now ignores a registry entry recording that the extension `.js` should have MIME type `text/plain`. This is a common unintentional misconfiguration on Windows systems. The effect is that `.js` will have the default MIME type `text/javascript; charset=utf-8`. Applications that expect `text/plain` on Windows must now explicitly call `AddExtensionType`.

mime/multipart

In Go 1.19.8 and later, this package sets limits the size of the MIME data it processes to protect against malicious inputs. `Reader.NextPart` and `Reader.NextRawPart` limit the number of headers in a part to 10000 and `Reader.ReadForm` limits the total number of headers in all `FileHeaders` to 10000. These limits may be adjusted with the `GODEBUG=multipartmaxheaders` setting. `Reader.ReadForm` further limits the number of parts in a form to 1000. This limit may be adjusted with the `GODEBUG=multipartmaxparts` setting.

net

The pure Go resolver will now use EDNS(0) to include a suggested maximum reply packet length, permitting reply packets to contain up to 1232 bytes (the previous maximum was 512). In the unlikely event that this causes problems with a local DNS resolver, setting the environment variable `GODEBUG=netdns=cgo` to use the cgo-based resolver should work. Please report any such problems on the issue tracker.

When a net package function or method returns an "I/O timeout" error, the error will now satisfy `errors.Is(err, context.DeadlineExceeded)`. When a net package function returns an "operation was canceled" error, the error will now satisfy `errors.Is(err, context.Canceled)`. These changes are intended to make it easier for code to test for cases in which a context cancellation or timeout causes a net package function or method to return an error, while preserving backward compatibility for error messages.

`Resolver.PreferGo` is now implemented on Windows and Plan 9. It previously only worked on Unix platforms. Combined with `Dialer.Resolver` and `Resolver.Dial`, it's now possible to write portable programs and be in control of all DNS name lookups when dialing.

The `net` package now has initial support for the `netgo` build tag on Windows. When used, the package uses the Go DNS client (as used by `Resolver.PreferGo`) instead of asking Windows for DNS results. The upstream DNS server it discovers from Windows may not yet be correct with complex system network configurations, however.

net/http

`ResponseWriter.WriteHeader` now supports sending user-defined 1xx informational headers.

The `io.ReadCloser` returned by `MaxBytesReader` will now return the defined error type `MaxBytesError` when its read limit is exceeded.

The HTTP client will handle a 3xx response without a `Location` header by returning it to the caller, rather than treating it as an error.

net/url

The new `JoinPath` function and `URL.JoinPath` method create a new `URL` by joining a list of path elements.

The `URL` type now distinguishes between URLs with no authority and URLs with an empty authority. For example, `http:///path` has an empty authority (host), while `http:/path` has none.

The new `URL` field `OmitHost` is set to `true` when a `URL` has an empty authority.

os/exec

A `Cmd` with a non-empty `Dir` field and nil `Env` now implicitly sets the `PWD` environment variable for the subprocess to match `Dir`.

The new method `Cmd.Environ` reports the environment that would be used to run the command, including the implicitly set `PWD` variable.

reflect

The method `Value.Bytes` now accepts addressable arrays in addition to slices.

The methods `Value.Len` and `Value.Cap` now successfully operate on a pointer to an array and return the length of that array, to match what the builtin `len` and `cap` functions do.

regexp/syntax

Go 1.18 release candidate 1, Go 1.17.8, and Go 1.16.15 included a security fix to the regular expression parser, making it reject very deeply nested expressions. Because Go patch releases do not introduce new API, the parser returned `syntax.ErrInternalError` in this case. Go 1.19 adds a more specific error, `syntax.ErrNestingDepth`, which the parser now returns instead.

runtime

The `GOROOT` function now returns the empty string (instead of `"go"`) when the binary was built with the `-trimpath` flag set and the `GOROOT` variable is not set in the process environment.

runtime/metrics

The new `/sched/gomaxprocs:threads` metric reports the current `runtime.GOMAXPROCS` value.

The new `/cgo/go-to-c-calls:calls` metric reports the total number of calls made from Go to C. This metric is identical to the `runtime.NumCgoCall` function.

The new `/gc/limiter/last-enabled:gc-cycle` metric reports the last GC cycle when the GC CPU limiter was enabled. See the runtime notes for details about the GC CPU limiter.

runtime/pprof

Stop-the-world pause times have been significantly reduced when collecting goroutine profiles, reducing the overall latency impact to the application.

`MaxRSS` is now reported in heap profiles for all Unix operating systems (it was previously only reported for `GOOS=android`, `darwin`, `ios`, and `linux`).

runtime/race

The race detector has been upgraded to use thread sanitizer version v3 on all supported platforms except `windows/amd64` and `openbsd/amd64`, which remain on v2. Compared to v2, it is now typically 1.5x to 2x faster, uses half as much memory, and it supports an unlimited number of goroutines. On Linux, the race detector now requires at least glibc version 2.17 and GNU binutils 2.26.

The race detector is now supported on `GOARCH=s390x`.

Race detector support for `openbsd/amd64` has been removed from thread sanitizer upstream, so it is unlikely to ever be updated from v2.

runtime/trace

When tracing and the CPU profiler are enabled simultaneously, the execution trace includes CPU profile samples as instantaneous events.

sort

The sorting algorithm has been rewritten to use pattern-defeating quicksort, which is faster for several common scenarios.

The new function `Find` is like `Search` but often easier to use: it returns an additional boolean reporting whether an equal value was found.

strconv

`Quote` and related functions now quote the rune U+007F as `\x7f`, not `\u007f`, for consistency with other ASCII values.

syscall

On PowerPC (`GOARCH=ppc64`, `ppc64le`), `Syscall`, `Syscall6`, `RawSyscall`, and `RawSyscall6` now always return 0 for return value `r2` instead of an undefined value.

On AIX and Solaris, `Getrusage` is now defined.

time

The new method `Duration.Abs` provides a convenient and safe way to take the absolute value of a duration, converting -2^{63} to $2^{63}-1$. (This boundary case can happen as the result of subtracting a recent time from the zero time.)

The new method `Time.ZoneBounds` returns the start and end times of the time zone in effect at a given time. It can be used in a loop to enumerate all the known time zone transitions at a given location.

□□□ #2
□ □□□ □□□ 2026-10-26 20:26:29 UTC
□ □□□ □□□ 2026-10-26 20:26:29 UTC