Go 1.3 Release Notes

- Changes to the supported operating systems and architectures {#os}
  - Removal of support for Windows 2000 {#win2000}
  - Support for DragonFly BSD {#dragonfly}
  - Support for FreeBSD {#freebsd}
  - Support for Native Client {#nacl}
  - Support for NetBSD {#netbsd}
  - Support for OpenBSD {#openbsd}
  - Support for Plan 9 {#plan9}
  - Support for Solaris {#solaris}
- Changes to the memory model {#memory}
- Changes to the implementations and tools {#impl}
  - Stack {#stacks}
  - Changes to the garbage collector {#garbage_collector}
  - Map iteration {#map}
  - The linker {#liblink}
  - Status of gccgo {#gccgo}
  - Changes to the go command {#gocmd}
  - Changes to cgo {#cgo}
  - SWIG 3.0 required for programs that use SWIG {#swig}
  - Command-line flag parsing {#gc_flag}
  - Changes to godoc {#godoc}
  - Miscellany {#misc}
- Performance {#performance}
- Changes to the standard library {#library}
  - New packages {#new_packages}
  - Major changes to the library {#major_library_changes}
  - Minor changes to the library {#minor_library_changes}

The latest Go release, version 1.3, arrives six months after 1.2, and contains no language changes. It focuses primarily on implementation work, providing precise garbage collection, a major refactoring of the compiler toolchain that results in faster builds, especially for large projects, significant performance improvements across the board, and support for DragonFly BSD, Solaris, Plan 9 and Google's Native Client architecture (NaCl). It also has an important refinement to the memory model regarding synchronization. As always, Go 1.3 keeps the promise of compatibility, and almost everything will continue to compile and run without change when moved to 1.3.

Changes to the supported operating systems and architectures {#os}

Removal of support for Windows 2000 {#win2000}

Microsoft stopped supporting Windows 2000 in 2010. Since it has implementation difficulties regarding exception handling (signals in Unix terminology), as of Go 1.3 it is not supported by Go either.

Support for DragonFly BSD {#dragonfly}

Go 1.3 now includes experimental support for DragonFly BSD on the `amd64` (64-bit x86) and `386` (32-bit x86) architectures. It uses DragonFly BSD 3.6 or above.

Support for FreeBSD {#freebsd}

It was not announced at the time, but since the release of Go 1.2, support for Go on FreeBSD requires FreeBSD 8 or above.

As of Go 1.3, support for Go on FreeBSD requires that the kernel be compiled with the `COMPAT_FREEBSD32` flag configured.

In concert with the switch to EABI syscalls for ARM platforms, Go 1.3 will run only on FreeBSD 10. The x86 platforms, 386 and amd64, are unaffected.

Support for Native Client {#nacl}

Support for the Native Client virtual machine architecture has returned to Go with the 1.3 release. It runs on the 32-bit Intel architectures (`GOARCH=386`) and also on 64-bit Intel, but using 32-bit pointers (`GOARCH=amd64p32`). There is not yet support for Native Client on ARM. Note that this is Native Client (NaCl), not Portable Native Client (PNaCl). Details about Native Client are here; how to set up the Go version is described here.

Support for NetBSD {#netbsd}

As of Go 1.3, support for Go on NetBSD requires NetBSD 6.0 or above.

Support for OpenBSD {#openbsd}

As of Go 1.3, support for Go on OpenBSD requires OpenBSD 5.5 or above.

Support for Plan 9 {#plan9}

Go 1.3 now includes experimental support for Plan 9 on the `386` (32-bit x86) architecture. It requires the `Tsemacquire` syscall, which has been in Plan 9 since June, 2012.

Support for Solaris {#solaris}

Go 1.3 now includes experimental support for Solaris on the `amd64` (64-bit x86) architecture. It requires illumos, Solaris 11 or above.

Changes to the memory model {#memory}

The Go 1.3 memory model adds a new rule concerning sending and receiving on buffered channels, to make explicit that a buffered channel can be used as a simple semaphore, using a send into the channel to acquire and a receive from the channel to release. This is not a language change, just a clarification about an expected property of communication.

Changes to the implementations and tools {#impl}

Stack {#stacks}

Go 1.3 has changed the implementation of goroutine stacks away from the old, "segmented" model to a contiguous model. When a goroutine needs more stack than is available, its stack is transferred to a larger single block of memory. The overhead of this transfer operation amortizes well and eliminates the old "hot spot" problem when a calculation repeatedly steps across a segment boundary. Details including performance numbers are in this design document.

Changes to the garbage collector {#garbage_collector}

For a while now, the garbage collector has been *precise* when examining values in the heap; the Go 1.3 release adds equivalent precision to values on the stack. This means that a non-pointer Go value such as an integer will never be mistaken for a pointer and prevent unused memory from being reclaimed.

Starting with Go 1.3, the runtime assumes that values with pointer type contain pointers and other values do not. This assumption is fundamental to the precise behavior of both stack expansion and garbage collection. Programs that use package unsafe to store integers in pointer-typed values are illegal and will crash if the runtime detects the behavior. Programs that use package unsafe to store pointers in integer-typed values are also illegal but more difficult to diagnose during execution. Because the pointers are hidden from the runtime, a stack expansion or garbage collection may reclaim the memory they point at, creating dangling pointers.

Updating: Code that uses `unsafe.Pointer` to convert an integer-typed value held in memory into a pointer is illegal and must be rewritten. Such code can be identified by `go vet`.

Map iteration {#map}

Iterations over small maps no longer happen in a consistent order. Go 1 defines that "The iteration order over maps is not specified and is not guaranteed to be the same from one iteration to the next." To keep code from depending on map iteration order, Go 1.0 started each map iteration at a

random index in the map. A new map implementation introduced in Go 1.1 neglected to randomize iteration for maps with eight or fewer entries, although the iteration order can still vary from system to system. This has allowed people to write Go 1.1 and Go 1.2 programs that depend on small map iteration order and therefore only work reliably on certain systems. Go 1.3 reintroduces random iteration for small maps in order to flush out these bugs.

Updating: If code assumes a fixed iteration order for small maps, it will break and must be rewritten not to make that assumption. Because only small maps are affected, the problem arises most often in tests.

The linker {#liblink}

As part of the general overhaul to the Go linker, the compilers and linkers have been refactored. The linker is still a C program, but now the instruction selection phase that was part of the linker has been moved to the compiler through the creation of a new library called `liblink`. By doing instruction selection only once, when the package is first compiled, this can speed up compilation of large projects significantly.

Updating: Although this is a major internal change, it should have no effect on programs.

Status of gccgo {#gccgo}

GCC release 4.9 will contain the Go 1.2 (not 1.3) version of gccgo. The release schedules for the GCC and Go projects do not coincide, which means that 1.3 will be available in the development branch but that the next GCC release, 4.10, will likely have the Go 1.4 version of gccgo.

Changes to the go command {#gocmd}

The `cmd/go` command has several new features. The `go run` and `go test` subcommands support a new `-exec` option to specify an alternate way to run the resulting binary. Its immediate purpose is to support NaCl.

The test coverage support of the `go test` subcommand now automatically sets the coverage mode to `-atomic` when the race detector is enabled, to eliminate false reports about unsafe access to coverage counters.

The `go test` subcommand now always builds the package, even if it has no test files. Previously, it would do nothing if no test files were present.

The `go build` subcommand supports a new `-i` option to install dependencies of the specified target, but not the target itself.

Cross compiling with `cgo` enabled is now supported. The CC_FOR_TARGET and CXX_FOR_TARGET environment variables are used when running all.bash to specify the cross compilers for C and C++ code, respectively.

Finally, the go command now supports packages that import Objective-C files (suffixed `.m`) through cgo.

Changes to cgo {#cgo}

The `cmd/cgo` command, which processes `import "C"` declarations in Go packages, has corrected a serious bug that may cause some packages to stop compiling. Previously, all pointers to incomplete struct types translated to the Go type `*[0]byte`, with the effect that the Go compiler could not diagnose passing one kind of struct pointer to a function expecting another. Go 1.3 corrects this mistake by translating each different incomplete struct to a different named type.

Given the C declaration `typedef struct S T` for an incomplete `struct S`, some Go code used this bug to refer to the types `C.struct_S` and `C.T` interchangeably. Cgo now explicitly allows this use, even for completed struct types. However, some Go code also used this bug to pass (for example) a `*C.FILE` from one package to another. This is not legal and no longer works: in general Go packages should avoid exposing C types and names in their APIs.

Updating: Code confusing pointers to incomplete types or passing them across package boundaries will no longer compile and must be rewritten. If the conversion is correct and must be preserved, use an explicit conversion via `unsafe.Pointer`.

SWIG 3.0 required for programs that use SWIG {#swig}

For Go programs that use SWIG, SWIG version 3.0 is now required. The `cmd/go` command will now link the SWIG generated object files directly into the binary, rather than building and linking with a shared library.

Command-line flag parsing {#gc_flag}

In the gc toolchain, the assemblers now use the same command-line flag parsing rules as the Go flag package, a departure from the traditional Unix flag parsing. This may affect scripts that invoke the tool directly. For example, `go tool 6a -SDfoo` must now be written `go tool 6a -S -D foo`. (The same change was made to the compilers and linkers in Go 1.1.)

Changes to godoc {#godoc}

When invoked with the `-analysis` flag, godoc now performs sophisticated static analysis of the code it indexes. The results of analysis are presented in both the source view and the package documentation view, and include the call graph of each package and the relationships between definitions and references, types and their methods, interfaces and their implementations, send and receive operations on channels, functions and their callers, and call sites and their callees.

Miscellany {#misc}

The program `misc/benchcmp` that compares performance across benchmarking runs has been rewritten. Once a shell and awk script in the main repository, it is now a Go program in the `go.tools` repo. Documentation is here.

For the few of us that build Go distributions, the tool `misc/dist` has been moved and renamed; it now lives in `misc/makerelease`, still in the main repository.

Performance {#performance}

The performance of Go binaries for this release has improved in many cases due to changes in the runtime and garbage collection, plus some changes to libraries. Significant instances include:

- The runtime handles defers more efficiently, reducing the memory footprint by about two kilobytes per goroutine that calls defer.
- The garbage collector has been sped up, using a concurrent sweep algorithm, better parallelization, and larger pages. The cumulative effect can be a 50-70% reduction in collector pause time.
- The race detector (see this guide) is now about 40% faster.
- The regular expression package `regexp` is now significantly faster for certain simple expressions due to the implementation of a second, one-pass execution engine. The choice of which engine to use is automatic; the details are hidden from the user.

Also, the runtime now includes in stack dumps how long a goroutine has been blocked, which can be useful information when debugging deadlocks or performance issues.

Changes to the standard library {#library}

New packages {#new_packages}

A new package `debug/plan9obj` was added to the standard library. It implements access to Plan 9 a.out object files.

Major changes to the library {#major_library_changes}

A previous bug in `crypto/tls` made it possible to skip verification in TLS inadvertently. In Go 1.3, the bug is fixed: one must specify either ServerName or InsecureSkipVerify, and if ServerName is specified it is enforced. This may break existing code that incorrectly depended on insecure behavior.

There is an important new type added to the standard library: `sync.Pool`. It provides an efficient mechanism for implementing certain types of caches whose memory can be reclaimed automatically by the system.

The `testing` package's benchmarking helper, `B`, now has a `RunParallel` method to make it easier to run benchmarks that exercise multiple CPUs.

Updating: The crypto/tls fix may break existing code, but such code was erroneous and should be updated.

Minor changes to the library {#minor_library_changes}

The following list summarizes a number of minor changes to the library, mostly additions. See the relevant package documentation for more information about each change.

- In the `crypto/tls` package, a new `DialWithDialer` function lets one establish a TLS connection using an existing dialer, making it easier to control dial options such as timeouts. The package also now reports the TLS version used by the connection in the `ConnectionState` struct.

- The `CreateCertificate` function of the `crypto/tls` package now supports parsing (and elsewhere, serialization) of PKCS #10 certificate signature requests.
- The formatted print functions of the `fmt` package now define `%F` as a synonym for `%f` when printing floating-point values.
- The `math/big` package's `Int` and `Rat` types now implement `encoding.TextMarshaler` and `encoding.TextUnmarshaler`.
- The complex power function, `Pow`, now specifies the behavior when the first argument is zero. It was undefined before. The details are in the documentation for the function.
- The `net/http` package now exposes the properties of a TLS connection used to make a client request in the new `Response.TLS` field.
- The `net/http` package now allows setting an optional server error logger with `Server.ErrorLog`. The default is still that all errors go to stderr.
- The `net/http` package now supports disabling HTTP keep-alive connections on the server with `Server.SetKeepAlivesEnabled`. The default continues to be that the server does keep-alive (reuses connections for multiple requests) by default. Only resource-constrained servers or those in the process of graceful shutdown will want to disable them.
- The `net/http` package adds an optional `Transport.TLSHandshakeTimeout` setting to cap the amount of time HTTP client requests will wait for TLS handshakes to complete. It's now also set by default on `DefaultTransport`.
- The `net/http` package's `DefaultTransport`, used by the HTTP client code, now enables TCP keep-alives by default. Other `Transport` values with a nil `Dial` field continue to function the same as before: no TCP keep-alives are used.
- The `net/http` package now enables TCP keep-alives for incoming server requests when `ListenAndServe` or `ListenAndServeTLS` are used. When a server is started otherwise, TCP keep-alives are not enabled.
- The `net/http` package now provides an optional `Server.ConnState` callback to hook various phases of a server connection's lifecycle (see `ConnState`). This can be used to implement rate limiting or graceful shutdown.
- The `net/http` package's HTTP client now has an optional `Client.Timeout` field to specify an end-to-end timeout on requests made using the client.
- The `net/http` package's `Request.ParseMultipartForm` method will now return an error if the body's `Content-Type` is not `multipart/form-data`. Prior to Go 1.3 it would silently fail and return `nil`. Code that relies on the previous behavior should be updated.
- In the `net` package, the `Dialer` struct now has a `KeepAlive` option to specify a keep-alive period for the connection.
- The `net/http` package's `Transport` now closes `Request.Body` consistently, even on error.
- The `os/exec` package now implements what the documentation has always said with regard to relative paths for the binary. In particular, it only calls `LookPath` when the binary's file name contains no path separators.
- The `SetMapIndex` function in the `reflect` package no longer panics when deleting from a `nil` map.
- If the main goroutine calls `runtime.Goexit` and all other goroutines finish execution, the program now always crashes, reporting a detected deadlock. Earlier versions of Go handled this situation inconsistently: most instances were reported as deadlocks, but some trivial cases exited cleanly instead.

- The runtime/debug package now has a new function `debug.WriteHeapDump` that writes out a description of the heap.
- The `CanBackquote` function in the `strconv` package now considers the `DEL` character, `U+007F`, to be non-printing.
- The `syscall` package now provides `SendmsgN` as an alternate version of `Sendmsg` that returns the number of bytes written.
- On Windows, the `syscall` package now supports the cdecl calling convention through the addition of a new function `NewCallbackCDecl` alongside the existing function `NewCallback`.
- The `testing` package now diagnoses tests that call `panic(nil)`, which are almost always erroneous. Also, tests now write profiles (if invoked with profiling flags) even on failure.
- The `unicode` package and associated support throughout the system has been upgraded from Unicode 6.2.0 to Unicode 6.3.0.

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