Go 1.17 Release Notes

Table of Contents

Introduction to Go 1.17

The latest Go release, version 1.17, arrives six months after Go 1.16. 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 1.17 includes three small enhancements to the language.

- Conversions from slice to array pointer: An expression `s` of type `[]T` may now be converted to array pointer type `*[N]T`. If `a` is the result of such a conversion, then corresponding indices that are in range refer to the same underlying elements: `&a[i] == &s[i]` for `0 <= i < N`. The conversion panics if `len(s)` is less than `N`.
- `unsafe.Add`: `unsafe.Add(ptr, len)` adds `len` to `ptr` and returns the updated pointer `unsafe.Pointer(uintptr(ptr) + uintptr(len))`.
- `unsafe.Slice`: For expression `ptr` of type `*T`, `unsafe.Slice(ptr, len)` returns a slice of type `[]T` whose underlying array starts at `ptr` and whose length and capacity are `len`.

The package unsafe enhancements were added to simplify writing code that conforms to `unsafe.Pointer`'s safety rules, but the rules remain unchanged. In particular, existing programs that correctly use `unsafe.Pointer` remain valid, and new programs must still follow the rules when using `unsafe.Add` or `unsafe.Slice`.

Note that the new conversion from slice to array pointer is the first case in which a type conversion can panic at run time. Analysis tools that assume type conversions can never panic should be updated to consider this possibility.

Ports

Darwin

As announced in the Go 1.16 release notes, Go 1.17 requires macOS 10.13 High Sierra or later; support for previous versions has been discontinued.

Windows

Go 1.17 adds support of 64-bit ARM architecture on Windows (the `windows/arm64` port). This port supports cgo.

OpenBSD

The 64-bit MIPS architecture on OpenBSD (the `openbsd/mips64` port) now supports cgo.

In Go 1.16, on the 64-bit x86 and 64-bit ARM architectures on OpenBSD (the `openbsd/amd64` and `openbsd/arm64` ports) system calls are made through `libc`, instead of directly using machine instructions. In Go 1.17, this is also done on the 32-bit x86 and 32-bit ARM architectures on OpenBSD (the `openbsd/386` and `openbsd/arm` ports). This ensures compatibility with OpenBSD 6.9 onwards, which require system calls to be made through `libc` for non-static Go binaries.

ARM64

Go programs now maintain stack frame pointers on the 64-bit ARM architecture on all operating systems. Previously, stack frame pointers were only enabled on Linux, macOS, and iOS.

loong64 GOARCH value reserved

The main Go compiler does not yet support the LoongArch architecture, but we've reserved the `GOARCH` value "`loong64`". This means that Go files named `*_loong64.go` will now be ignored by Go tools except when that GOARCH value is being used.

Tools

Go command

Pruned module graphs in go `1.17` modules

If a module specifies `go 1.17` or higher, the module graph includes only the *immediate* dependencies of other `go 1.17` modules, not their full transitive dependencies. (See Module graph pruning for more detail.)

For the `go` command to correctly resolve transitive imports using the pruned module graph, the `go.mod` file for each module needs to include more detail about the transitive dependencies relevant to that module. If a module specifies `go 1.17` or higher in its `go.mod` file, its `go.mod` file now contains an explicit `require` directive for every module that provides

a transitively-imported package. (In previous versions, the `go.mod` file typically only included explicit requirements for *directly*-imported packages.)

Since the expanded `go.mod` file needed for module graph pruning includes all of the dependencies needed to load the imports of any package in the main module, if the main module specifies `go 1.17` or higher the `go` tool no longer reads (or even downloads) `go.mod` files for dependencies if they are not needed in order to complete the requested command. (See Lazy loading.)

Because the number of explicit requirements may be substantially larger in an expanded Go 1.17 `go.mod` file, the newly-added requirements on *indirect* dependencies in a `go 1.17` module are maintained in a separate `require` block from the block containing direct dependencies.

To facilitate the upgrade to Go 1.17 pruned module graphs, the `go mod tidy` subcommand now supports a `-go` flag to set or change the `go` version in the `go.mod` file. To convert the `go.mod` file for an existing module to Go 1.17 without changing the selected versions of its dependencies, run:

```
go mod tidy -go=1.17
```

By default, `go mod tidy` verifies that the selected versions of dependencies relevant to the main module are the same versions that would be used by the prior Go release (Go 1.16 for a module that specifies `go 1.17`), and preserves the `go.sum` entries needed by that release even for dependencies that are not normally needed by other commands.

The `-compat` flag allows that version to be overridden to support older (or only newer) versions, up to the version specified by the `go` directive in the `go.mod` file. To tidy a `go 1.17` module for Go 1.17 only, without saving checksums for (or checking for consistency with) Go 1.16:

```
go mod tidy -compat=1.17
```

Note that even if the main module is tidied with `-compat=1.17`, users who `require` the module from a `go 1.16` or earlier module will still be able to use it, provided that the packages use only compatible language and library features.

The `go mod graph` subcommand also supports the `-go` flag, which causes it to report the graph as seen by the indicated Go version, showing dependencies that may otherwise be pruned out.

Module deprecation comments

Module authors may deprecate a module by adding a `// Deprecated:` comment to `go.mod`, then tagging a new version. `go get` now prints a warning if a module needed to build packages named on the command line is deprecated. `go list -m -u` prints deprecations for all dependencies (use `-f` or `-json` to show the full message). The `go` command considers

different major versions to be distinct modules, so this mechanism may be used, for example, to provide users with migration instructions for a new major version.

`go get`

The `go get -insecure` flag is deprecated and has been removed. To permit the use of insecure schemes when fetching dependencies, please use the `GOINSECURE` environment variable. The `-insecure` flag also bypassed module sum validation, use `GOPRIVATE` or `GONOSUMDB` if you need that functionality. See `go help environment` for details.

`go get` prints a deprecation warning when installing commands outside the main module (without the `-d` flag). `go install cmd@version` should be used instead to install a command at a specific version, using a suffix like `@latest` or `@v1.2.3`. In Go 1.18, the `-d` flag will always be enabled, and `go get` will only be used to change dependencies in `go.mod`.

`go.mod` files missing `go` directives

If the main module's `go.mod` file does not contain a `go` directive and the `go` command cannot update the `go.mod` file, the `go` command now assumes `go 1.11` instead of the current release. (`go mod init` has added `go` directives automatically since Go 1.12.)

If a module dependency lacks an explicit `go.mod` file, or its `go.mod` file does not contain a `go` directive, the `go` command now assumes `go 1.16` for that dependency instead of the current release. (Dependencies developed in GOPATH mode may lack a `go.mod` file, and the `vendor/modules.txt` has to date never recorded the `go` versions indicated by dependencies' `go.mod` files.)

`vendor` contents

If the main module specifies `go 1.17` or higher, `go mod vendor` now annotates `vendor/modules.txt` with the `go` version indicated by each vendored module in its own `go.mod` file. The annotated version is used when building the module's packages from vendored source code.

If the main module specifies `go 1.17` or higher, `go mod vendor` now omits `go.mod` and `go.sum` files for vendored dependencies, which can otherwise interfere with the ability of the `go` command to identify the correct module root when invoked within the `vendor` tree.

Password prompts

The `go` command by default now suppresses SSH password prompts and Git Credential Manager prompts when fetching Git repositories using SSH, as it already did previously for other Git password prompts. Users authenticating to private Git repos with password-protected SSH may configure an `ssh-agent` to enable the `go` command to use password-protected SSH keys.

`go mod download`

When `go mod download` is invoked without arguments, it will no longer save sums for downloaded module content to `go.sum`. It may still make changes to `go.mod` and `go.sum` needed to load the build list. This is the same as the behavior in Go 1.15. To save sums for all modules, use `go mod download all`.

`//go:build` lines

The `go` command now understands `//go:build` lines and prefers them over `// +build` lines. The new syntax uses boolean expressions, just like Go, and should be less error-prone. As of this release, the new syntax is fully supported, and all Go files should be updated to have both forms with the same meaning. To aid in migration, `gofmt` now automatically synchronizes the two forms. For more details on the syntax and migration plan, see https://golang.org/design/draft-gobuild.

`go run`

`go run` now accepts arguments with version suffixes (for example, `go run example.com/cmd@v1.0.0`). This causes `go run` to build and run 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 running executables without installing them or without changing dependencies of the current module.

Gofmt

`gofmt` (and `go fmt`) now synchronizes `//go:build` lines with `// +build` lines. If a file only has `// +build` lines, they will be moved to the appropriate location in the file, and matching `//go:build` lines will be added. Otherwise, `// +build` lines will be overwritten based on any existing `//go:build` lines. For more information, see https://golang.org/design/draft-gobuild.

Vet

New warning for mismatched `//go:build` and `// +build` lines

The `vet` tool now verifies that `//go:build` and `// +build` lines are in the correct part of the file and synchronized with each other. If they aren't, `gofmt` can be used to fix them. For more information, see https://golang.org/design/draft-gobuild.

New warning for calling `signal.Notify` on unbuffered channels

The vet tool now warns about calls to signal.Notify with incoming signals being sent to an unbuffered channel. Using an unbuffered channel risks missing signals sent on them as `signal.Notify` does not block when sending to a channel. For example:

```
c := make(chan os.Signal)
// signals are sent on c before the channel is read from.
// This signal may be dropped as c is unbuffered.
signal.Notify(c, os.Interrupt)
```

Users of `signal.Notify` should use channels with sufficient buffer space to keep up with the expected signal rate.

New warnings for Is, As and Unwrap methods

The vet tool now warns about methods named `As`, `Is` or `Unwrap` on types implementing the `error` interface that have a different signature than the one expected by the `errors` package. The `errors.{As,Is,Unwrap}` functions expect such methods to implement either `Is(error) bool`, `As(interface{}) bool`, or `Unwrap() error` respectively. The functions `errors.{As,Is,Unwrap}` will ignore methods with the same names but a different signature. For example:

```
type MyError struct { hint string }
func (m MyError) Error() string { ... } // MyError implements error.
func (MyError) Is(target interface{}) bool { ... } // target is interface{} inste
func Foo() bool {
        x, y := MyError{"A"}, MyError{"B"}
        return errors.Is(x, y) // returns false as x != y and MyError does not ha
}
```

Cover

The `cover` tool now uses an optimized parser from `golang.org/x/tools/cover`, which may be noticeably faster when parsing large coverage profiles.

Compiler

Go 1.17 implements a new way of passing function arguments and results using registers instead of the stack. Benchmarks for a representative set of Go packages and programs show performance improvements of about 5%, and a typical reduction in binary size of about 2%. This is currently enabled for Linux, macOS, and Windows on the 64-bit x86 architecture (the `linux/amd64`, `darwin/amd64`, and `windows/amd64` ports).

This change does not affect the functionality of any safe Go code and is designed to have no impact on most assembly code. It may affect code that violates the `unsafe.Pointer` rules when accessing function arguments, or that depends on undocumented behavior involving comparing function code pointers. To maintain compatibility with existing assembly functions, the compiler generates adapter functions that convert between the new register-based calling convention and the previous stack-based calling convention. These adapters are typically invisible to users, except that taking the address of a Go function in assembly code or taking the address of an assembly function in Go code using `reflect.ValueOf(fn).Pointer()` or `unsafe.Pointer` will now return the address of the adapter. Code that depends on the value of these code pointers may no longer behave as expected. Adapters also may cause a very small performance overhead in two cases: calling an assembly function indirectly from Go via a `func` value, and calling Go functions from assembly.

The format of stack traces from the runtime (printed when an uncaught panic occurs, or when `runtime.Stack` is called) is improved. Previously, the function arguments were printed as hexadecimal words based on the memory layout. Now each argument in the source code is printed separately, separated by commas. Aggregate-typed (struct, array, string, slice, interface, and complex) arguments are delimited by curly braces. A caveat is that the value of an argument that only lives in a register and is not stored to memory may be inaccurate. Function return values (which were usually inaccurate) are no longer printed.

Functions containing closures can now be inlined. One effect of this change is that a function with a closure may produce a distinct closure code pointer for each place that the function is inlined. Go function values are not directly comparable, but this change could reveal bugs in code that uses `reflect` or `unsafe.Pointer` to bypass this language restriction and compare functions by code pointer.

Linker

When the linker uses external linking mode, which is the default when linking a program that uses cgo, and the linker is invoked with a `-I` option, the option will now be passed to the external linker as a `-Wl,--dynamic-linker` option.

Core library

Cgo

The runtime/cgo package now provides a new facility that allows to turn any Go values to a safe representation that can be used to pass values between C and Go safely. See runtime/cgo.Handle for more information.

URL query parsing

The `net/url` and `net/http` packages used to accept `";"` (semicolon) as a setting separator in URL queries, in addition to `"&"` (ampersand). Now, settings with non-percent-encoded semicolons are rejected and `net/http` servers will log a warning to `Server.ErrorLog` when encountering one in a request URL.

For example, before Go 1.17 the `Query` method of the URL `example?a=1;b=2&c=3` would have returned `map[a:[1] b:[2] c:[3]]`, while now it returns `map[c:[3]]`.

When encountering such a query string, `URL.Query` and `Request.FormValue` ignore any settings that contain a semicolon, `ParseQuery` returns the remaining settings and an error, and `Request.ParseForm` and `Request.ParseMultipartForm` return an error but still set `Request` fields based on the remaining settings.

`net/http` users can restore the original behavior by using the new `AllowQuerySemicolons` handler wrapper. This will also suppress the `ErrorLog` warning. Note that accepting semicolons as query separators can lead to security issues if different systems interpret cache keys differently. See issue 25192 for more information.

TLS strict ALPN

When `Config.NextProtos` is set, servers now enforce that there is an overlap between the configured protocols and the ALPN protocols advertised by the client, if any. If there is no mutually supported protocol, the connection is closed with the `no_application_protocol` alert, as required by RFC 7301. This helps mitigate the ALPACA cross-protocol attack.

As an exception, when the value `"h2"` is included in the server's `Config.NextProtos`, HTTP/1.1 clients will be allowed to connect as if they didn't support ALPN. See issue 46310 for more information.

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 methods `File.OpenRaw`, `Writer.CreateRaw`, `Writer.Copy` provide support for cases where performance is a primary concern.

bufio

The `Writer.WriteRune` method now writes the replacement character U+FFFD for negative rune values, as it does for other invalid runes.

bytes

The `Buffer.WriteRune` method now writes the replacement character U+FFFD for negative rune values, as it does for other invalid runes.

compress/lzw

The `NewReader` function is guaranteed to return a value of the new type `Reader`, and similarly `NewWriter` is guaranteed to return a value of the new type `Writer`. These new types both implement a `Reset` method (`Reader.Reset`, `Writer.Reset`) that allows reuse of the `Reader` or `Writer`.

crypto/ed25519

The `crypto/ed25519` package has been rewritten, and all operations are now approximately twice as fast on amd64 and arm64. The observable behavior has not otherwise changed.

crypto/elliptic

`CurveParams` methods now automatically invoke faster and safer dedicated implementations for known curves (P-224, P-256, and P-521) when available. Note that this is a best-effort approach and applications should avoid using the generic, not

constant-time `CurveParams` methods and instead use dedicated `Curve` implementations such as `P256`.

The `P521` curve implementation has been rewritten using code generated by the fiat-crypto project, which is based on a formally-verified model of the arithmetic operations. It is now constant-time and three times faster on amd64 and arm64. The observable behavior has not otherwise changed.

crypto/rand

The `crypto/rand` package now uses the `getentropy` syscall on macOS and the `getrandom` syscall on Solaris, Illumos, and DragonFlyBSD.

crypto/tls

The new `Conn.HandshakeContext` method allows the user to control cancellation of an in-progress TLS handshake. The provided context is accessible from various callbacks through the new `ClientHelloInfo.Context` and `CertificateRequestInfo.Context` methods. Canceling the context after the handshake has finished has no effect.

Cipher suite ordering is now handled entirely by the `crypto/tls` package. Currently, cipher suites are sorted based on their security, performance, and hardware support taking into account both the local and peer's hardware. The order of the `Config.CipherSuites` field is now ignored, as well as the `Config.PreferServerCipherSuites` field. Note that `Config.CipherSuites` still allows applications to choose what TLS 1.0–1.2 cipher suites to enable.

The 3DES cipher suites have been moved to `InsecureCipherSuites` due to fundamental block size-related weakness. They are still enabled by default but only as a last resort, thanks to the cipher suite ordering change above.

Beginning in the next release, Go 1.18, the `Config.MinVersion` for `crypto/tls` clients will default to TLS 1.2, disabling TLS 1.0 and TLS 1.1 by default. Applications will be able to override the change by explicitly setting `Config.MinVersion`. This will not affect `crypto/tls` servers.

crypto/x509

`CreateCertificate` now returns an error if the provided private key doesn't match the parent's public key, if any. The resulting certificate would have failed to verify.

The temporary `GODEBUG=x509ignoreCN=0` flag has been removed.

`ParseCertificate` has been rewritten, and now consumes ~70% fewer resources. The observable behavior when processing WebPKI certificates has not otherwise changed, except for error messages.

On BSD systems, `/etc/ssl/certs` is now searched for trusted roots. This adds support for the new system trusted certificate store in FreeBSD 12.2+.

Beginning in the next release, Go 1.18, `crypto/x509` will reject certificates signed with the SHA-1 hash function. This doesn't apply to self-signed root certificates. Practical attacks against SHA-1 have been demonstrated in 2017 and publicly trusted Certificate Authorities have not issued SHA-1 certificates since 2015.

database/sql

The `DB.Close` method now closes the `connector` field if the type in this field implements the `io.Closer` interface.

The new `NullInt16` and `NullByte` structs represent the int16 and byte values that may be null. These can be used as destinations of the `Scan` method, similar to NullString.

debug/elf

The `SHT_MIPS_ABIFLAGS` constant has been added.

encoding/binary

`binary.Uvarint` will stop reading after `10 bytes` to avoid wasted computations. If more than `10 bytes` are needed, the byte count returned is `-11`.
Previous Go versions could return larger negative counts when reading incorrectly encoded varints.

encoding/csv

The new `Reader.FieldPos` method returns the line and column corresponding to the start of a given field in the record most recently returned by `Read`.

encoding/xml

When a comment appears within a `Directive`, it is now replaced with a single space instead of being completely elided.

Invalid element or attribute names with leading, trailing, or multiple colons are now stored unmodified into the `Name.Local` field.

flag

Flag declarations now panic if an invalid name is specified.

go/build

The new `Context.ToolTags` field holds the build tags appropriate to the current Go toolchain configuration.

go/format

The `Source` and `Node` functions now synchronize `//go:build` lines with `// +build` lines. If a file only has `// +build` lines, they will be moved to the appropriate location in the file, and matching `//go:build` lines will be added. Otherwise, `// +build` lines will be overwritten based on any existing `//go:build` lines. For more information, see https://golang.org/design/draft-gobuild.

go/parser

The new `SkipObjectResolution` `Mode` value instructs the parser not to resolve identifiers to their declaration. This may improve parsing speed.

image

The concrete image types (`RGBA`, `Gray16` and so on) now implement a new `RGBA64Image` interface. The concrete types that previously implemented `draw.Image` now also implement `draw.RGBA64Image`, a new interface in the `image/draw` package.

io/fs

The new `FileInfoToDirEntry` function converts a `FileInfo` to a `DirEntry`.

math

The math package now defines three more constants: `MaxUint`, `MaxInt` and `MinInt`. For 32-bit systems their values are `2^32 - 1`, `2^31 - 1` and `-2^31`, respectively. For 64-bit systems their values are `2^64 - 1`, `2^63 - 1` and `-2^63`, respectively.

mime

On Unix systems, the table of MIME types is now read from the local system's Shared MIME-info Database when available.

mime/multipart

`Part.FileName` now applies `filepath.Base` to the return value. This mitigates potential path traversal vulnerabilities in applications that accept multipart messages, such as `net/http` servers that call `Request.FormFile`.

net

The new method `IP.IsPrivate` reports whether an address is a private IPv4 address according to RFC 1918 or a local IPv6 address according RFC 4193.

The Go DNS resolver now only sends one DNS query when resolving an address for an IPv4-only or IPv6-only network, rather than querying for both address families.

The `ErrClosed` sentinel error and `ParseError` error type now implement the `net.Error` interface.

The `ParseIP` and `ParseCIDR` functions now reject IPv4 addresses which contain decimal components with leading zeros. These components were always interpreted as decimal, but some operating systems treat them as octal. This mismatch could hypothetically lead to security issues if a Go application was used to validate IP addresses which were then used in their original form with non-Go applications which interpreted components as octal. Generally, it is advisable to always re-encode values after validation, which avoids this class of parser misalignment issues.

net/http

The `net/http` package now uses the new `(*tls.Conn).HandshakeContext` with the `Request` context when performing TLS handshakes in the client or server.

Setting the `Server` `ReadTimeout` or `WriteTimeout` fields to a negative value now indicates no timeout rather than an immediate timeout.

The `ReadRequest` function now returns an error when the request has multiple Host headers.

When producing a redirect to the cleaned version of a URL, `ServeMux` now always uses relative URLs in the `Location` header. Previously it would echo the full URL of the request, which could lead to unintended redirects if the client could be made to send an absolute request URL.

When interpreting certain HTTP headers handled by `net/http`, non-ASCII characters are now ignored or rejected.

If `Request.ParseForm` returns an error when called by `Request.ParseMultipartForm`, the latter now continues populating `Request.MultipartForm` before returning it.

net/http/httptest

`ResponseRecorder.WriteHeader` now panics when the provided code is not a valid three-digit HTTP status code. This matches the behavior of `ResponseWriter` implementations in the `net/http` package.

net/url

The new method `Values.Has` reports whether a query parameter is set.

os

The `File.WriteString` method has been optimized to not make a copy of the input string.

reflect

The new `Value.CanConvert` method reports whether a value can be converted to a type. This may be used to avoid a panic when converting a slice to an array pointer type

if the slice is too short. Previously it was sufficient to use `Type.ConvertibleTo` for this, but the newly permitted conversion from slice to array pointer type can panic even if the types are convertible.

The new `StructField.IsExported` and `Method.IsExported` methods report whether a struct field or type method is exported. They provide a more readable alternative to checking whether `PkgPath` is empty.

The new `VisibleFields` function returns all the visible fields in a struct type, including fields inside anonymous struct members.

The `ArrayOf` function now panics when called with a negative length.

Checking the `Type.ConvertibleTo` method is no longer sufficient to guarantee that a call to `Value.Convert` will not panic. It may panic when converting `[]T` to `*[N]T` if the slice's length is less than N. See the language changes section above.

The `Value.Convert` and `Type.ConvertibleTo` methods have been fixed to not treat types in different packages with the same name as identical, to match what the language allows.

runtime/metrics

New metrics were added that track total bytes and objects allocated and freed. A new metric tracking the distribution of goroutine scheduling latencies was also added.

runtime/pprof

Block profiles are no longer biased to favor infrequent long events over frequent short events.

strconv

The `strconv` package now uses Ulf Adams's Ryū algorithm for formatting floating-point numbers. This algorithm improves performance on most inputs and is more than 99% faster on worst-case inputs.

The new `QuotedPrefix` function returns the quoted string (as understood by `Unquote`) at the start of input.

strings

The `Builder.WriteRune` method now writes the replacement character U+FFFD for negative rune values, as it does for other invalid runes.

sync/atomic

`atomic.Value` now has `Swap` and `CompareAndSwap` methods that provide additional atomic operations.

syscall

The `GetQueuedCompletionStatus` and `PostQueuedCompletionStatus` functions are now deprecated. These functions have incorrect signatures and are superseded by equivalents in the `golang.org/x/sys/windows` package.

On Unix-like systems, the process group of a child process is now set with signals blocked. This avoids sending a `SIGTTOU` to the child when the parent is in a background process group.

The Windows version of `SysProcAttr` has two new fields. `AdditionalInheritedHandles` is a list of additional handles to be inherited by the new child process. `ParentProcess` permits specifying the parent process of the new process.

The constant `MSG_CMSG_CLOEXEC` is now defined on DragonFly and all OpenBSD systems (it was already defined on some OpenBSD systems and all FreeBSD, NetBSD, and Linux systems).

The constants `SYS_WAIT6` and `WEXITED` are now defined on NetBSD systems (`SYS_WAIT6` was already defined on DragonFly and FreeBSD systems; `WEXITED` was already defined on Darwin, DragonFly, FreeBSD, Linux, and Solaris systems).

testing

Added a new testing flag `-shuffle` which controls the execution order of tests and benchmarks.

The new `T.Setenv` and `B.Setenv` methods support setting an environment variable for the duration of the test or benchmark.

text/template/parse

The new `SkipFuncCheck` `Mode` value changes the template parser to not verify that functions are defined.

time

The `Time` type now has a `GoString` method that will return a more useful value for times when printed with the `%#v` format specifier in the `fmt` package.

The new `Time.IsDST` method can be used to check whether the time is in Daylight Savings Time in its configured location.

The new `Time.UnixMilli` and `Time.UnixMicro` methods return the number of milliseconds and microseconds elapsed since January 1, 1970 UTC respectively.
The new `UnixMilli` and `UnixMicro` functions return the local `Time` corresponding to the given Unix time.

The package now accepts comma "," as a separator for fractional seconds when parsing and formatting time. For example, the following time layouts are now accepted:

- 2006-01-02 15:04:05,999999999 -0700 MST
- Mon Jan _2 15:04:05,000000 2006
- Monday, January 2 15:04:05,000 2006

The new constant `Layout` defines the reference time.

unicode

The `Is`, `IsGraphic`, `IsLetter`, `IsLower`, `IsMark`, `IsNumber`, `IsPrint`, `IsPunct`, `IsSpace`, `IsSymbol`, and `IsUpper` functions now return `false` on negative rune values, as they do for other invalid runes.