The Go Programming Language

Go 1.10 Release Notes

DRAFT RELEASE NOTES - Introduction to Go 1.10

Go 1.10 is not yet released. These are work-in-progress release notes. Go 1.10 is expected to be released in February 2018.

The latest Go release, version 1.10, arrives six months after Go 1.9. 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.

This release improves caching of built packages, adds caching of successful test results, runs vet automatically during tests, and permits passing string values directly between Go and C using cgo.

Changes to the language

There are no significant changes to the language specification.

A corner case involving shifts by untyped constants has been clarified, and as a result the compilers have been updated to allow the index expression `x[1.0 << s]` where `s` is an untyped constant; the go/types package already did.

The grammar for method expressions has been updated to relax the syntax to allow any type expression as a receiver; this matches what the compilers were already implementing. For example, `struct{io.Reader}.Read` is a valid, if unusual, method expression that the compilers already accepted and is now permitted by the language grammar.

Ports

There are no new supported operating systems or processor architectures in this release. Most of the work has focused on strengthening the support for existing ports, in particular new instructions in the assembler and improvements to the code generated by the compilers.

As announced in the Go 1.9 release notes, Go 1.10 now requires FreeBSD 10.3 or later; support for FreeBSD 9.3 has been removed.

Go now runs on NetBSD again but requires the unreleased NetBSD 8. Only `GOARCH` `amd64` and `386` have been fixed. The `arm` port is still broken.

On 32-bit MIPS systems, the new environment variable settings `GOMIPS=hardfloat` (the default) and `GOMIPS=softfloat` select whether to use hardware instructions or software emulation for floating-point computations.

Go 1.10 is the last release that will run on OpenBSD 6.0. Go 1.11 will require OpenBSD 6.2.

Go 1.10 is the last release that will run on OS X 10.8 Mountain Lion or OS X 10.9 Mavericks. Go 1.11 will require OS X 10.10 Yosemite or later.

Go 1.10 is the last release that will run on Windows XP or Windows Vista. Go 1.11 will require Windows 7 or later.

Tools

Default GOROOT & GOTMPDIR

If the environment variable `$GOROOT` is unset, the go tool previously used the default `GOROOT` set during toolchain compilation. Now, before falling back to that default, the go tool attempts to deduce `GOROOT` from its own executable path. This allows binary distributions to be unpacked anywhere in the file system and then be used without setting `GOROOT` explicitly.

By default, the go tool creates its temporary files and directories in the system temporary directory (for example, `$TMPDIR` on Unix). If the new environment variable `$GOTMPDIR` is set, the go tool will creates its temporary files and directories in that directory instead.

Build & Install

The `go build` command now detects out-of-date packages purely based on the content of source files, specified build flags, and metadata stored in the compiled packages. Modification times are no longer consulted or relevant. The old advice to add `-a` to force a rebuild in cases where the modification times were misleading for one reason or another (for example, changes in build flags) is no longer necessary: builds now always detect when packages must be rebuilt. (If you observe otherwise, please file a bug.)

The `go build` `-asmflags`, `-gcflags`, `-gccgoflags`, and `-ldflags` options now apply by default only to the packages listed directly on the command line. For example, `go build -gcflags=-m mypkg` passes the compiler the `-m` flag when building `mypkg` but not its dependencies. The new, more general form `-asmflags=pattern=flags` (and similarly for the others) applies the `flags` only to the packages matching the pattern. For example: `go install -ldflags=cmd/gofmt=-X=main.version=1.2.3 cmd/...` installs all the commands matching `cmd/...` but only applies the `-X` option to the linker flags for `cmd/gofmt`. For more details, see `go help build`.

The `go build` command now maintains a cache of recently built packages, separate from the installed packages in `$GOROOT/pkg` or `$GOPATH/pkg`. The effect of the cache should be to speed builds that do not explicitly install packages or when switching between different copies of source code (for example, when changing back and forth between different branches in a version control system). The old advice to add the `-i` flag for speed, as in `go build -i` or `go test -i`, is no longer necessary: builds run just as fast without `-i`. For more details, see `go help cache`.

The `go install` command now installs only the packages and commands listed directly on the command line. For example, `go install cmd/gofmt` installs the gofmt program but not any of the packages on which it depends. The new build cache makes future commands still run as quickly as if the dependencies had been installed. To force the installation of dependencies, use the new `go install -i` flag. Installing dependency packages should not be necessary in general, and the very concept of installed packages may disappear in a future release.

Many details of the `go build` implementation have changed to support these improvements. One new requirement implied by these changes is that binary-only packages must now declare accurate import blocks in their stub source code, so that those imports can be made available when linking a program using the binary-only package. For more details, see `go help filetype`.

Test

The `go test` command now caches test results: if the test executable and command line match a previous run and the files and environment variables consulted by that run have not changed either, `go test` will print the previous test output, replacing the elapsed time with the string “(cached).” Test caching applies only to successful test results; only to `go test` commands with an explicit list of packages; and only to command lines using a subset of the `-cpu`, `-list`, `-parallel`, `-run`, `-short`, and `-v` test flags. The idiomatic way to bypass test caching is to use `-count=1`.

The `go test` command now automatically runs `go vet` on the package being tested, to identify significant problems before running the test. Any such problems are treated like build errors and prevent execution of the test. Only a high-confidence subset of the available `go vet` checks are enabled for this automatic check. To disable the running of `go vet`, use `go test -vet=off`.

The `go test -coverpkg` flag now interprets its argument as a comma-separated list of patterns to match against the dependencies of each test, not as a list of packages to load anew. For example, `go test -coverpkg=all` is now a meaningful way to run a test with coverage enabled for the test package and all its dependencies. Also, the `go test -coverprofile` option is now supported when running multiple tests.

In case of failure due to timeout, tests are now more likely to write their profiles before exiting.

The `go test` command now always merges the standard output and standard error from a given test binary execution and writes both to `go test`'s standard output. In past releases, `go test` only applied this merging most of the time.

The `go test -v` output now includes `PAUSE` and `CONT` status update lines to mark when parallel tests pause and continue.

The new `go test -failfast` flag disables running additional tests after any test fails. Note that tests running in parallel with the failing test are allowed to complete.

Finally, the new `go test -json` flag filters test output through the new command `go tool test2json` to produce a machine-readable JSON-formatted description of test execution. This allows the creation of rich presentations of test execution in IDEs and other tools.

For more details about all these changes, see `go help test` and the test2json documentation.

Cgo

Cgo now implements a C typedef like “typedef X Y” using a Go type alias, so that Go code may use the types C.X and C.Y interchangeably. It also now supports the use of niladic function-like macros. Also, the documentation has been updated to clarify that Go structs and Go arrays are not supported in the type signatures of cgo-exported functions.

Cgo now supports direct access to Go string values from C. Functions in the C preamble may use the type _GoString_ to accept a Go string as an argument. C code may call _GoStringLen and _GoStringPtr for direct access to the contents of the string. A value of type _GoString_ may be passed in a call to an exported Go function that takes an argument of Go type string.

During toolchain bootstrap, the environment variables CC and CC_FOR_TARGET specify the default C compiler that the resulting toolchain will use for host and target builds, respectively. However, if the toolchain will be used with multiple targets, it may be necessary to specify a different C compiler for each (for example, a different compiler for darwin/arm64 versus linux/ppc64le). The new set of environment variables CC_FOR_*goos*_*goarch* allows specifying a different default C compiler for each target. Note that these variables only apply during toolchain bootstrap, to set the defaults used by the resulting toolchain. Later go build commands use the CC environment variable or else the built-in default.

Cgo now translates some C types that would normally map to a pointer type in Go, to a uintptr instead. These types include the CFTypeRef hierarchy in Darwin's CoreFoundation framework and the jobject hierarchy in Java's JNI interface.

These types must be uintptr on the Go side because they would otherwise confuse the Go garbage collector; they are sometimes not really pointers but data structures encoded in a pointer-sized integer. Pointers to Go memory must not be stored in these uintptr values.

Because of this change, values of the affected types need to be zero-initialized with the constant 0 instead of the constant nil. Go 1.10 provides gofix modules to help with that rewrite:

```
go tool fix -r cftype <pkg>
go tool fix -r jni <pkg>
```

For more details, see the cgo documentation.

Doc

The go doc tool now adds functions returning slices of T or *T to the display of type T, similar to the existing behavior for functions returning single T or *T results. For example:

```
$ go doc mail.Address
package mail // import "net/mail"

type Address struct {
        Name    string
        Address string
}
    Address represents a single mail address.

func ParseAddress(address string) (*Address, error)
func ParseAddressList(list string) ([]*Address, error)
func (a *Address) String() string
$
```

Previously, `ParseAddressList` was only shown in the package overview (`go doc mail`).

Fix

The `go fix` tool now replaces imports of `"golang.org/x/net/context"` with `"context"`. (Forwarding aliases in the former make it completely equivalent to the latter when using Go 1.9 or later.)

Get

The `go get` command now supports Fossil source code repositories.

Pprof

The blocking and mutex profiles produced by the `runtime/pprof` package now include symbol information, so they can be viewed in `go tool pprof` without the binary that produced the profile. (All other profile types were changed to include symbol information in Go 1.9.)

The `go tool pprof` profile visualizer has been updated to git version 9e20b5b (2017-11-08) from github.com/google/pprof, which includes an updated web interface.

Vet

The `go vet` command now always has access to complete, up-to-date type information when checking packages, even for packages using cgo or vendored imports. The reports should be more accurate as a result. Note that only `go vet` has access to this information; the more low-level `go tool vet` does not and should be avoided except when working on `vet` itself. (As of Go 1.9, `go vet` provides access to all the same flags as `go tool vet`.)

Diagnostics

This release includes a new overview of available Go program diagnostic tools.

Gofmt

Two minor details of the default formatting of Go source code have changed. First, certain complex three-index slice expressions previously formatted like `x[i+1 : j:k]` and now format with more consistent spacing: `x[i+1 : j : k]`. Second, single-method interface literals written on a single line, which are sometimes used in type assertions, are no longer split onto multiple lines.

Note that these kinds of minor updates to gofmt are expected from time to time. In general, we recommend against building systems that check that source code matches the output of a specific version of gofmt. For example, a continuous integration test that fails if any code already checked into a repository is not “properly formatted” is inherently fragile and not recommended.

If multiple programs must agree about which version of gofmt is used to format a source file, we recommend that they do this by arranging to invoke the same gofmt binary. For example, in the Go open source repository, our Git pre-commit hook is written in Go and could import `go/format` directly, but instead it invokes the `gofmt` binary found in the current path, so that the pre-commit hook need not be recompiled each time `gofmt` changes.

Compiler Toolchain

The compiler includes many improvements to the performance of generated code, spread fairly evenly across the supported architectures.

The DWARF debug information recorded in binaries has been improved in a few ways: constant values are now recorded; line number information is more accurate, making source-level stepping through a program work better; and each package is now presented as its own DWARF compilation unit.

The various build modes have been ported to more systems. Specifically, `c-shared` now works on `linux/ppc64le`, `windows/386`, and `windows/amd64`; `pie` now works on `darwin/amd64` and also forces the use of external linking on all systems; and `plugin` now works on `linux/ppc64le` and `darwin/amd64`.

The `linux/ppc64le` port now requires the use of external linking with any programs that use cgo, even uses by the standard library.

Assembler

For the ARM 32-bit port, the assembler now supports the instructions `BFC`, `BFI`, `BFX`, `BFXU`, `FMULAD`, `FMULAF`, `FMULSD`, `FMULSF`, `FNMULAD`, `FNMULAF`, `FNMULSD`, `FNMULSF`, `MULAD`, `MULAF`, `MULSD`, `MULSF`, `NMULAD`, `NMULAF`, `NMULD`, `NMULF`, `NMULSD`, `NMULSF`, `XTAB`, `XTABU`, `XTAH`, and `XTAHU`.

For the ARM 64-bit port, the assembler now supports the `VADD`, `VADDP`, `VADDV`, `VAND`, `VCMEQ`, `VDUP`, `VEOR`, `VLD1`, `VMOV`, `VMOVI`, `VMOVS`, `VORR`, `VREV32`, and `VST1` instructions.

For the PowerPC 64-bit port, the assembler now supports the POWER9 instructions `ADDEX`, `CMPEQB`, `COPY`, `DARN`, `LDMX`, `MADDHD`, `MADDHDU`, `MADDLD`, `MFVSRLD`, `MTVSRDD`, `MTVSRWS`, `PASTECC`, `VCMPNEZB`, `VCMPNEZBCC`, and `VMSUMUDM`.

For the S390X port, the assembler now supports the `TMHH`, `TMHL`, `TMLH`, and `TMLL` instructions.

For the X86 64-bit port, the assembler now supports 359 new instructions, including the full AVX, AVX2, BMI, BMI2, F16C, FMA3, SSE2, SSE3, SSSE3, SSE4.1, and SSE4.2 extension sets. The assembler also no longer implements `MOVL $0, AX` as an `XORL` instruction, to avoid clearing the condition flags unexpectedly.

Gccgo

Due to the alignment of Go's semiannual release schedule with GCC's annual release schedule, GCC release 7 contains the Go 1.8.3 version of gccgo. We expect that the next release, GCC 8, will contain the Go 1.10 version of gccgo.

Runtime

The behavior of nested calls to `LockOSThread` and `UnlockOSThread` has changed. These functions control whether a goroutine is locked to a specific operating system thread, so that the goroutine only runs on that thread, and the thread only runs that goroutine. Previously, calling `LockOSThread` more than once in a row was equivalent to calling it once, and a single `UnlockOSThread` always unlocked the thread. Now, the calls nest: if `LockOSThread` is called multiple times, `UnlockOSThread` must be called the same number of times in order to unlock the thread. Existing code that was careful not to nest these calls will remain correct. Existing code that incorrectly assumed the calls nested will become correct. Most uses of these functions in public Go source code falls into the second category.

Because one common use of `LockOSThread` and `UnlockOSThread` is to allow Go code to reliably modify thread-local state (for example, Linux or Plan 9 name spaces), the runtime now treats locked threads as unsuitable for reuse or for creating new threads.

Stack traces no longer include implicit wrapper functions (previously marked `<autogenerated>`), unless a fault or panic happens in the wrapper itself. As a result, skip counts passed to functions like `Caller` should now always match the structure of the code as written, rather than depending on optimization decisions and implementation details.

The garbage collector has been modified to reduce its impact on allocation latency. It now uses a smaller fraction of the overall CPU when running, but it may run more of the time. The total CPU consumed by the garbage collector has not changed significantly.

The `GOROOT` function now defaults (when the `$GOROOT` environment variable is not set) to the `GOROOT` or `GOROOT_FINAL` in effect at the time the calling program was compiled. Previously it used the `GOROOT` or `GOROOT_FINAL` in effect at the time the toolchain that compiled the calling program was compiled.

There is no longer a limit on the `GOMAXPROCS` setting. (In Go 1.9 the limit was 1024.)

Performance

As always, the changes are so general and varied that precise statements about performance are difficult to make. Most programs should run a bit faster, due to speedups in the garbage collector, better generated code, and optimizations in the core library.

Garbage Collector

Many applications should experience significantly lower allocation latency and overall performance overhead when the garbage collector is active.

Core library

All of the changes to the standard library are minor. The changes in bytes and net/url are the most likely to require updating of existing programs.

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/tar

In general, the handling of special header formats is significantly improved and expanded.

`FileInfoHeader` has always recorded the Unix UID and GID numbers from its `os.FileInfo` argument (specifically, from the system-dependent information returned by the `FileInfo`'s `Sys` method) in the returned `Header`. Now it also records the user and group names corresponding to those IDs, as well as the major and minor device numbers for device files.

The new `Header.Format` field of type `Format` controls which tar header format the `Writer` uses. The default, as before, is to select the most widely-supported header type that can encode the fields needed by the header (USTAR if possible, or else PAX if possible, or else GNU). The `Reader` sets `Header.Format` for each header it reads.

`Reader` and the `Writer` now support arbitrary PAX records, using the new `Header.PAXRecords` field, a generalization of the existing `Xattrs` field.

The `Reader` no longer insists that the file name or link name in GNU headers be valid UTF-8.

When writing PAX- or GNU-format headers, the `Writer` now includes the `Header.AccessTime` and `Header.ChangeTime` fields (if set). When writing PAX-format headers, the times include sub-second precision.

archive/zip

Go 1.10 adds more complete support for times and character set encodings in ZIP archives.

The original ZIP format used the standard MS-DOS encoding of year, month, day, hour, minute, and second into fields in two 16-bit values. That encoding cannot represent time zones or odd seconds, so multiple extensions have been introduced to allow richer encodings. In Go 1.10, the `Reader` and `Writer` now support the widely-understood Info-Zip extension that encodes the time separately in the 32-bit Unix “seconds since epoch” form. The `FileHeader`'s new `Modified` field of type `time.Time` obsoletes the `ModifiedTime` and `ModifiedDate` fields, which continue to hold the MS-DOS encoding. The `Reader` and `Writer` now adopt the common convention that a ZIP archive storing a time zone-independent Unix time also stores the local time in the MS-DOS field, so that the time zone offset can be inferred. For compatibility, the `ModTime` and `SetModTime` methods behave the same as in earlier releases; new code should use `Modified` directly.

The header for each file in a ZIP archive has a flag bit indicating whether the name and comment fields are encoded as UTF-8, as opposed to a system-specific default encoding. In Go 1.8 and earlier, the `Writer` never set the UTF-8 bit. In Go 1.9, the `Writer` changed to set the UTF-8 bit almost always. This broke the creation of ZIP archives containing Shift-JIS file names. In Go 1.10, the `Writer` now sets the UTF-8 bit only when both the name and the comment field are valid UTF-8 and at least one is non-ASCII. Because non-ASCII encodings very rarely look like valid UTF-8, the new heuristic should be correct nearly all the time. Setting a `FileHeader`'s new `NonUTF8` field to true disables the heuristic entirely for that file.

The `Writer` also now supports setting the end-of-central-directory record's comment field, by calling the `Writer`'s new `SetComment` method.

bufio

The new `Reader.Size` and `Writer.Size` methods report the `Reader` or `Writer`'s underlying buffer size.

bytes

The `Fields`, `FieldsFunc`, `Split`, and `SplitAfter` functions have always returned subslices of their inputs. Go 1.10 changes each returned subslice to have capacity equal to its length, so that appending to one cannot overwrite adjacent data in the original input.

crypto/cipher

`NewOFB` now panics if given an initialization vector of incorrect length, like the other constructors in the package always have. (Previously it returned a nil `Stream` implementation.)

crypto/tls

The TLS server now advertises support for SHA-512 signatures when using TLS 1.2. The server already supported the signatures, but some clients would not select them unless explicitly advertised.

crypto/x509

`Certificate.Verify` now enforces the name constraints for all names contained in the certificate, not just the one name that a client has asked about. Extended key usage restrictions are similarly now checked all at once. As a result, after a certificate has been validated, now it can be trusted in its entirety. It is no longer necessary to revalidate the certificate for each additional name or key usage.

Parsed certificates also now report URI names and IP, email, and URI constraints, using the new `Certificate` fields `URIs`, `PermittedIPRanges`, `ExcludedIPRanges`, `PermittedEmailAddresses`, `ExcludedEmailAddresses`, `PermittedURIDomains`, and `ExcludedURIDomains`.

The new `MarshalPKCS1PublicKey` and `ParsePKCS1PublicKey` functions convert an RSA public key to and from PKCS#1-encoded form.

The new `MarshalPKCS8PrivateKey` function converts a private key to PKCS#8-encoded form. (`ParsePKCS8PrivateKey` has existed since Go 1.)

crypto/x509/pkix

`Name` now implements a `String` method that formats the X.509 distinguished name in the standard RFC 2253 format.

database/sql/driver

Drivers that currently hold on to the destination buffer provided by `driver.Rows.Next` should ensure they no longer write to a buffer assigned to the destination array outside of that call. Drivers must be careful that underlying buffers are not modified when closing `driver.Rows`.

Drivers that want to construct a `sql.DB` for their clients can now implement the `Connector` interface and call the new `sql.OpenDB` function, instead of needing to encode all configuration into a string passed to `sql.Open`.

Drivers that want to parse the configuration string only once per `sql.DB` instead of once per `sql.Conn`, or that want access to each `sql.Conn`'s underlying context, can make their `Driver` implementations also implement `DriverContext`'s new `OpenConnector` method.

Drivers that implement `ExecerContext` no longer need to implement `Execer`; similarly, drivers that implement `QueryerContext` no longer need to implement `Queryer`. Previously, even if the context-based interfaces were implemented they were ignored unless the non-context-based interfaces were also implemented.

To allow drivers to better isolate different clients using a cached driver connection in succession, if a `Conn` implements the new `SessionResetter` interface, `database/sql` will now call `ResetSession` before reusing the `Conn` for a new client.

debug/elf

This release adds 348 new relocation constants divided between the relocation types `R_386`, `R_AARCH64`, `R_ARM`, `R_PPC64`, and `R_X86_64`.

debug/macho

Go 1.10 adds support for reading relocations from Mach-O sections, using the `Section` struct's new `Relocs` field and the new `Reloc`, `RelocTypeARM`, `RelocTypeARM64`, `RelocTypeGeneric`, and `RelocTypeX86_64` types and associated constants.

Go 1.10 also adds support for the `LC_RPATH` load command, represented by the types `RpathCmd` and `Rpath`, and new named constants for the various flag bits found in headers.

encoding/asn1

`Marshal` now correctly encodes strings containing asterisks as type UTF8String instead of PrintableString, unless the string is in a struct field with a tag forcing the use of PrintableString. `Marshal` also now respects struct tags containing `application` directives.

The new `MarshalWithParams` function marshals its argument as if the additional params were its associated struct field tag.

`Unmarshal` now respects struct field tags using the `explicit` and `tag` directives.

Both `Marshal` and `Unmarshal` now support a new struct field tag `numeric`, indicating an ASN.1 NumericString.

encoding/csv

`Reader` now disallows the use of nonsensical `Comma` and `Comment` settings, such as NUL, carriage return, newline, invalid runes, and the Unicode replacement character, or setting `Comma` and `Comment` equal to each other.

In the case of a syntax error in a CSV record that spans multiple input lines, `Reader` now reports the line on which the record started in the `ParseError`'s new `StartLine` field.

encoding/hex

The new functions `NewEncoder` and `NewDecoder` provide streaming conversions to and from hexadecimal, analogous to equivalent functions already in encoding/base32 and encoding/base64.

When the functions `Decode` and `DecodeString` encounter malformed input, they now return the number of bytes already converted along with the error. Previously they always returned a count of 0 with any error.

encoding/json

The `Decoder` adds a new method `DisallowUnknownFields` that causes it to report inputs with unknown JSON fields as a decoding error. (The default behavior has always been to discard unknown fields.)

As a result of fixing a reflect bug, `Unmarshal` can no longer decode into fields inside embedded pointers to unexported struct types, because it cannot initialize the unexported embedded pointer to point at fresh storage. `Unmarshal` now returns an error in this case.

encoding/pem

`Encode` and `EncodeToMemory` no longer generate partial output when presented with a block that is impossible to encode as PEM data.

encoding/xml

The new function `NewTokenDecoder` is like `NewDecoder` but creates a decoder reading from a `TokenReader` instead of an XML-formatted byte stream. This is meant to enable the construction of XML stream transformers in client libraries.

flag

The default `Usage` function now prints its first line of output to `CommandLine.Output()` instead of assuming `os.Stderr`, so that the usage message is properly redirected for clients using `CommandLine.SetOutput`.

`PrintDefaults` now adds appropriate indentation after newlines in flag usage strings, so that multi-line usage strings display nicely.

`FlagSet` adds new methods `ErrorHandling`, `Name`, and `Output`, to retrieve the settings passed to `NewFlagSet` and `FlagSet.SetOutput`.

go/doc

To support the doc change described above, functions returning slices of `T`, `*T`, `**T`, and so on are now reported in `T`'s `Type`'s `Funcs` list, instead of in the `Package`'s `Funcs` list.

go/importer

The `For` function now accepts a non-nil lookup argument.

go/printer

The changes to the default formatting of Go source code discussed in the gofmt section above are implemented in the go/printer package and also affect the output of the higher-level go/format package.

hash

Implementations of the `Hash` interface are now encouraged to implement `encoding.BinaryMarshaler` and `encoding.BinaryUnmarshaler` to allow saving and recreating their internal state, and all implementations in the standard library (hash/crc32, crypto/sha256, and so on) now implement those interfaces.

html/template

The new actions `{{break}}` and `{{continue}}` break out of the innermost `{{range ...}}` loop, like the corresponding Go statements.

The new `Srcset` content type allows for proper handling of values within the `srcset` attribute of `img` tags.

math/big

`Int` now supports conversions to and from bases 2 through 62 in its `SetString` and `Text` methods. (Previously it only allowed bases 2 through 36.) The value of the constant `MaxBase` has been updated.

`Int` adds a new `CmpAbs` method that is like `Cmp` but compares only the absolute values (not the signs) of its arguments.

`Float` adds a new `Sqrt` method to compute square roots.

math/cmplx

Branch cuts and other boundary cases in `Asin`, `Asinh`, `Atan`, and `Sqrt` have been corrected to match the definitions used in the C99 standard.

math/rand

The new `Shuffle` function and corresponding `Rand.Shuffle` method shuffle an input sequence.

math

The new functions `Round` and `RoundToEven` round their arguments to the nearest floating-point integer; `Round` rounds a half-integer to its larger integer neighbor (away from zero) while `RoundToEven` rounds a half-integer to its even integer neighbor.

The new functions `Erfinv` and `Erfcinv` compute the inverse error function and the inverse complementary error function.

mime/multipart

`Reader` now accepts parts with empty filename attributes.

mime

`ParseMediaType` now discards invalid attribute values; previously it returned those values as empty strings.

net

The `Conn` and `Listener` implementations in this package now guarantee that when `Close` returns, the underlying file descriptor has been closed. (In earlier releases, if the `Close` stopped pending I/O in other goroutines, the closing of the file descriptor could happen in one of those goroutines shortly after `Close` returned.)

`TCPListener` and `UnixListener` now implement `syscall.Conn`, to allow setting options on the underlying file descriptor using `syscall.RawConn.Control`.

The `Conn` implementations returned by `Pipe` now support setting read and write deadlines.

The `IPConn.ReadMsgIP`, `IPConn.WriteMsgIP`, `UDPConn.ReadMsgUDP`, and `UDPConn.WriteMsgUDP`, methods are now implemented on Windows.

net/http

On the client side, an HTTP proxy (most commonly configured by `ProxyFromEnvironment`) can now be specified as an `https://` URL, meaning that the client connects to the proxy over HTTPS before issuing a standard, proxied HTTP request. (Previously, HTTP proxy URLs were required to begin with `http://` or `socks5://`.)

On the server side, `FileServer` and its single-file equivalent `ServeFile` now apply `If-Range` checks to `HEAD` requests. `FileServer` also now reports directory read failures to the `Server`'s `ErrorLog`. The content-serving handlers also now omit the `Content-Type` header when serving zero-length content.

`ResponseWriter`'s `WriteHeader` method now panics if passed an invalid (non-3-digit) status code.

`Redirect` now sets the `Content-Type` header before writing its HTTP response.

net/mail

ParseAddress and ParseAddressList now support a variety of obsolete address formats.

net/smtp

The Client adds a new Noop method, to test whether the server is still responding. It also now defends against possible SMTP injection in the inputs to the Hello and Verify methods.

net/textproto

ReadMIMEHeader now rejects any header that begins with a continuation (indented) header line. Previously a header with an indented first line was treated as if the first line were not indented.

net/url

ResolveReference now preserves multiple leading slashes in the target URL. Previously it rewrote multiple leading slashes to a single slash, which resulted in the http.Client following certain redirects incorrectly.

For example, this code's output has changed:

```
base, _ := url.Parse("http://host//path//to/page1")
target, _ := url.Parse("page2")
fmt.Println(base.ResolveReference(target))
```

Note the doubled slashes around `path`. In Go 1.9 and earlier, the resolved URL was `http://host/path//to/page2`: the doubled slash before `path` was incorrectly rewritten to a single slash, while the doubled slash after `path` was correctly preserved. Go 1.10 preserves both doubled slashes, resolving to `http://host//path//to/page2` as required by RFC 3986.

This change may break existing buggy programs that unintentionally construct a base URL with a leading doubled slash in the path and inadvertently depend on `ResolveReference` to correct that mistake. For example, this can happen if code adds a host prefix like `http://host/` to a path like `/my/api`, resulting in a URL with a doubled slash: `http://host//my/api`.

UserInfo's methods now treat a nil receiver as equivalent to a pointer to a zero `UserInfo`. Previously, they panicked.

os

File adds new methods SetDeadline, SetReadDeadline, and SetWriteDeadline that allow setting I/O deadlines when the underlying file descriptor supports non-blocking I/O operations. The definition of these methods matches those in net.Conn. If an I/O method fails due to missing a deadline, it will return a timeout error; the new IsTimeout function reports whether an error represents a timeout.

Also matching `net.Conn`, `File`'s Close method now guarantee that when `Close` returns, the underlying file descriptor has been closed. (In earlier releases, if the `Close` stopped pending I/O in other goroutines, the closing of the file descriptor could happen in one of those goroutines shortly after `Close` returned.)

On BSD, macOS, and Solaris systems, Chtimes now supports setting file times with nanosecond precision (assuming the underlying file system can represent them).

reflect

The Copy function now allows copying from a string into a byte array or byte slice, to match the built-in copy function.

In structs, embedded pointers to unexported struct types were previously incorrectly reported with an empty `PkgPath` in the corresponding StructField, with the result that for those fields, and `Value.CanSet` incorrectly returned true and `Value.Set` incorrectly succeeded. The underlying metadata has been corrected; for those fields, `CanSet` now correctly returns false and `Set` now correctly panics. This may affect reflection-based unmarshalers that could previously unmarshal into such fields but no longer can. For example, see the `encoding/json` notes.

runtime/pprof

As noted above, the blocking and mutex profiles now include symbol information so that they can be viewed without needing the binary that generated them.

strconv

`ParseUint` now returns the maximum magnitude integer of the appropriate size with any `ErrRange` error, as it was already documented to do. Previously it returned 0 with `ErrRange` errors.

strings

A new type `Builder` is a replacement for `bytes.Buffer` for the use case of accumulating text into a `string` result. The `Builder`'s API is a restricted subset of `bytes.Buffer`'s that allows it to safely avoid making a duplicate copy of the data during the `String` method.

syscall

On Windows, the new `SysProcAttr` field `Token`, of type `Token` allows the creation of a process that runs as another user during `StartProcess` (and therefore also during `os.StartProcess` and `exec.Cmd.Start`). The new function `CreateProcessAsUser` gives access to the underlying system call.

On BSD, macOS, and Solaris systems, `UtimesNano` is now implemented.

text/template

The new actions `{{break}}` and `{{continue}}` break out of the innermost `{{range ...}}` loop, like the corresponding Go statements.

time

`LoadLocation` now uses the directory or uncompressed zip file named by the `$ZONEINFO` environment variable before looking in the default system-specific list of known installation locations or in `$GOROOT/lib/time/zoneinfo.zip`.

The new function `LoadLocationFromTZData` allows conversion of IANA time zone file data to a `Location`.

unicode

The `unicode` package and associated support throughout the system has been upgraded from Unicode 9.0 to Unicode 10.0, which adds 8,518 new characters, including four new scripts, one new property, a Bitcoin currency symbol, and 56 new emoji.